CPE 626
The SystemC Language

Aleksandar Milenkovic
E-mail: milenka@ece.uah.edu
Web: http://www.ece.uah.edu/~milenka

Outline

- Introduction
- Data Types
- Modeling Combinational Logic
- Modeling Synchronous Logic
- Misc

Introduction

- What is SystemC?
- Why SystemC?
- Design Methodology
- Capabilities
- SystemC RTL

What is SystemC

An extension of C++
enabling modeling of hardware descriptions

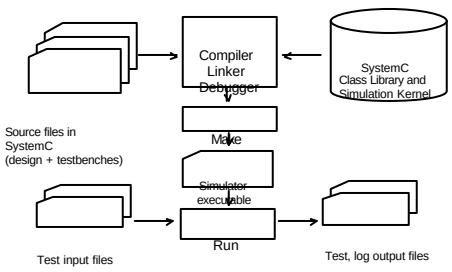
- SystemC adds a class library to C++
- Mechanisms to model system architecture
  - Concurrency – multiple processes executed concurently
  - Timed events
  - Reactive behavior
- Constructs to describe hardware
  - Signals
  - Modules
  - Ports
  - Processes
- Simulation kernel

What is SystemC

- Provides methodology for describing
  - System level design
  - Software algorithms
  - Hardware architecture
- Design Flow
  - Create a system level model
  - Explore various algorithms
    - Simulate to validate model and optimize design
  - Create executable specifications
    - Hardware team and software team use the same specification

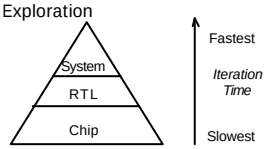
Test input files

C++/SystemC Development Environment



Why SystemC?

- Designs become
  - Bigger in size
  - Faster in speed
  - Larger in complexity
- Design description on higher levels of abstraction enables
  - Faster simulation
  - Hardware/software co-simulation
  - Architectural Exploration

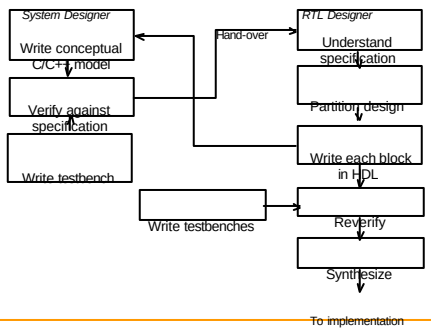


Why SystemC?

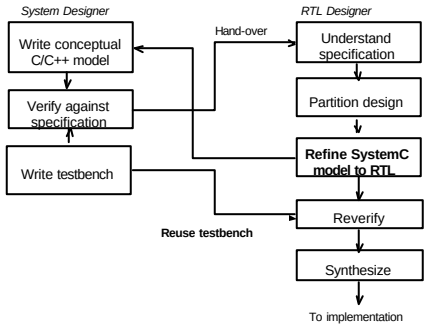
SystemC describes overall system

- System level design
- Describing hardware architectures
- Describing software algorithms
- Verification
- IP exchange

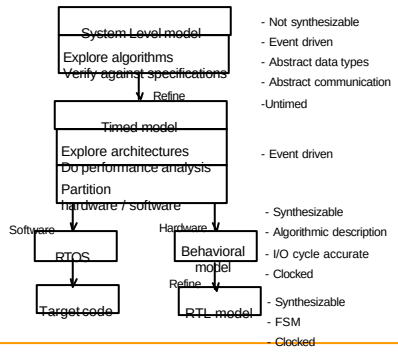
Non-SystemC Design Methodology



SystemC Design Methodology



System Level Design Process



SystemC Capabilities

- Modules
  - SC_MODULE class
- Processes
  - SC_METHOD, SC_THREAD
- Ports: input, output, inout
- Signals
  - resolved, unresolved
  - updates after a delta delay
- Rich set of data types
  - 2-value, 4-value logic
  - fixed, arbitrary size
- Clocks
  - built-in notion of clocks
- Event-base simulation
- Multiple abstraction levels
- Communication protocols
- Debugging support
- Waveform tracing
  - VCD: Value Change Dump (IEEE Std. 1364)
  - WIF: Waveform Interch. F.
  - ISDB: Integrated Signal Data Base
- RTL synthesis flow
- System and RTL modeling
- Software and Hardware

SystemC Half Adder

```
// File half_adder.h
#include “systemc.h”

SC_MODULE(half_adder) {
    sc_in<bool> a, b;
    sc_out<bool> sum, carry;

    void prc_half_adder();

    SC_CTOR(half_adder) {
       SC_METHOD(prc_half_adder);
       sensitive << a << b;
    }
};
```

```
// File half_adder.cpp
#include “half_adder.h”

void
half_adder::prc_half_adder() {
    sum = a ^ b;
    carry = a & b;
}
```


SystemC Decoder 2/4

```
// File decoder2by4.h
#include “systemc.h”

SC_MODULE(decoder2by4) {
    sc_in<bool> enable;
    sc_in<sc_uint<2> > select;
    sc_out<sc_uint<4> > z;

    void prc_decoder2by4();

    SC_CTOR(decoder2by4) {
       SC_METHOD(prc_half_adder);
       sensitive(enable, select);
    }
};
```

```
// File decoder2by4.cpp
#include “decoder2by4.h”

void decoder2by4::prc_ decoder2by4()
{
   if (enable) {
     switch(select.read()) {
        case 0: z=0xE; break;
        case 1: z=0xD; break;
        case 2: z=0xB; break;
        case 3: z=0x7; break;
     }
   }
   else
     z=0xF;
}
```

Note: Stream vs. Function notation

```
sensitive << a << b;   // Stream notation style

sensitive(a, b);       // Function notation style
```


Hierarchy: Building a full adder

```
// File full_adder.h
#include “systemc.h”

SC_MODULE(full_adder) {
   sc_in<bool> a,b,carry_in;
   sc_out<bool> sum,carry_out;
   sc_signal<bool> c1, s1, c2;
   void prc_or();
   half_adder *ha1_ptr, *ha2_ptr;

   SC_CTOR(full_adder) {
      ha1_ptr = new half_adder(“ha1”);
      ha1_ptr->a(a); ha1_ptr->b(b);
      ha1_ptr->sum(s1);
      ha1_ptr->carry(c1);
      ha2_ptr = new half_adder(“ha2”);
      (*ha2_ptr)(s1, carry_in,sum,c2);
      SC_METHOD(prc_or);
      sensitive << c1 << c2;
   }
```

```
   // a destructor
   ~full_adder() {
      delete ha1_ptr;
      delete ha2_ptr;
   }
};
```

```
// File: full_adder.cpp
#include “full_adder.h”

void full_adder::prc_or(){
   carry_out = c1 | c2;
}
```

Verifying the Functionality: Driver

```
// File driver.h
#include “systemc.h”

SC_MODULE(driver) {
   sc_out<bool> d_a,d_b,d_cin;

   void prc_driver();

   SC_CTOR(driver) {
      SC_THREAD(prc_driver);
   }
};
```

```
// File: driver.cpp
#include “driver.h”

void driver::prc_driver(){
   sc_uint<3> pattern;
   pattern=0;

   while(1) {
      d_a=pattern[0];
      d_b=pattern[1];
      d_cin=pattern[2];
      wait(5, SC_NS);
      pattern++;
   }
}
```


Verifying the Functionality: Monitor

```
// File monitor.h
#include “systemc.h”

SC_MODULE(monitor) {
   sc_in<bool> m_a,m_b,m_cin,
               m_sum, m_cout;

   void prc_monitor();

   SC_CTOR(monitor) {
      SC_THREAD(prc_monitor);
      sensitive << m_a,m_b,m_cin,
               m_sum, m_cout;

   }
};
```

```
// File: monitor.cpp
#include “monitor.h”

void monitor::prc_monitor(){
   cout << “At time “ <<
      sc_time_stamp() << “::”;
   cout <<”(a,b,carry_in): ”;
   cout << m_a << m_b << m_cin;
   cout << “(sum,carry_out): ”;
   cout << m_sum << m_cout << endl;
}
```


Verifying the Functionality: Main

```
// File full_adder_main.cpp
#include “driver.h”
#include “monitor.h”
#include “full_adder.h”

int sc_main(int argc, char * argv[]) {
   sc_signal<bool> t_a, t_b, t_cin, t_sum, t_cout;

   full_adder f1(“FullAdderWithHalfAdders”);
   f1 << t_a << t_b << t_cin << t_sum << t_cout;

   driver d1(“GenWaveforms”);
   d1.d_a(t_a);
   d1.d_b(t_b);
   d1.d_cin(t_cin);

   monitor m1(“MonitorWaveforms”);
   m1 << t_a << t_b << t_cin << t_sum << t_cout;

   sc_start(100, SC_NS);
   return(0);
}
```

Modeling Combinational Logic

```
// File: bist_cell.h
#include "systemc.h"
SC_MODULE(bist_cell) {
   sc_in<bool> b0,b1,d0,d1;
   sc_out<bool> z;
   void prc_bist_cell();
   SC_CTOR(bist_cell) {
      SC_METHOD(prc_bist_cell);
      sensitive <<b0<<b1<<d0<<d1;
   }
}
```

```
// File: bist_cell.cpp
#include "bist_cell.h"

void bist_cell::prc_bist_cell(){
 z = !(!(d0&b1)&!(b0&d1))|!
   (!(d0&b1)|!(b0&d1)));
}
```

```
// File: bist_cell.cpp
#include "bist_cell.h"

void
bist_cell::prc_bist_cell(){
   bool s1, s2, s3;

   s1 = !(b0&d1);
   s2 = !(d0&b1);
   s3 = !(s2|s1);
   s2 = s2&s1;
   z = !(s2|s3);
}
```

- Use SC_METHOD process with an event sensitivity list

Modeling Combinational Logic: Local Variables

```
// File: bist_cell.cpp
#include "bist_cell.h"

void bist_cell::prc_bist_cell(){
 z = !(!(d0&b1)&!(b0&d1))|!
   !(d0&b1)&!(b0&d1)));
}
```

```
// File: bist_cell.cpp
#include "bist_cell.h"

void
bist_cell::prc_bist_cell(){
   bool s1, s2, s3;

   s1 = !(b0&d1);
   s2 = !(d0&b1);
   s3 = !(s2|s1);
   s2 = s2&s1;
   z = !(s2|s3);
}
```

- Local variables in the process (s1, s2, s3) do not synthesize to wires
- Hold temporary values (improve readability)
- Are assigned values instantaneously (no delta delay) – one variable can represent many wires (s2)
  - signals and ports are updated after a delta delay
- Simulation is likely faster with local variables

Modeling Combinational Logic: Reading and Writing Ports and Signals

```
// File: xor_gates.h
#include "systemc.h"
SC_MODULE(xor_gates) {
   sc_in<sc_uint<4> > bre, sty;
   sc_out <sc_uint<4> > tap;
   void prc_xor_gates();
   SC_CTOR(xor_gates) {
      SC_METHOD(prc_xor_gates);
      sensitive << bre << sty;
   }
}
```

- Use read() and write() methods for reading and writing values from and to a port or signal

```
// File: xor_gates.cpp
#include "xor_gates.h"

void bist_cell::prc_xor_gates(){
 tap = bre ^ sty;
}
```

```
// File: xor_gates.cpp
#include "xor_gates.h"

void bist_cell::prc_xor_gates(){
 tap = bre.read() ^ sty.read();
}
```

Modeling Combinational Logic: Logical Operators

```
// File: xor_gates.h
#include "systemc.h"
const int SIZE=4;

SC_MODULE(xor_gates) {
   sc_in<sc_uint<SIZE> > bre, sty;
   sc_out <sc_uint<SIZE> > tap;
   void prc_xor_gates();
   SC_CTOR(xor_gates) {
      SC_METHOD(prc_xor_gates);
      sensitive << bre << sty;
   }
}
```

- ^ - xor
- & - and
- | - or

```
// File: xor_gates.cpp
#include "xor_gates.h"

void bist_cell::prc_xor_gates(){
 tap = bre.read() ^ sty.read();
}
```


Modeling Combinational Logic: Arithmetic Operations

```
sc_uint<4> write_addr;
sc_int<5> read_addr;

read_addr = write_addr + read_addr;
```

- Note: all fixed precision integer type calculations occur on a 64-bit representation and appropriate truncation occurs depending on the target result size
- E.g.:
  - write_addr is zero-extended to 64-bit
  - read_addr is sign-extended to 64-bit
  - + is performed on 64-bit data
  - result is truncated to 5-bit result and assigned back to read_addr

Modeling Combinational Logic: Unsigned Arithmetic

```
// File: u_adder.h
#include "systemc.h"

SC_MODULE(u_adder) {
   sc_in<sc_uint<4> > a, b;
   sc_out <sc_uint<5> > sum;
   void prc_u_adder();
   SC_CTOR(u_adder) {
      SC_METHOD(prc_u_adder);
      sensitive << a << b;
   }
};
```

```
// File: u_adder.cpp
#include "u_adder.h"

void u_adder ::prc_s_adder(){
 sum = a.read() + b.read();
}
```

Modeling Combinational Logic: Signed Arithmetic

```
// File: s_adder.h
#include "systemc.h"

SC_MODULE(s_adder) {
    sc_in<sc_int<4> > a, b;
    sc_out <sc_int<5> > sum;

    void prc_s_adder();

    SC_CTOR(s_adder) {
       SC_METHOD(prc_s_adder);
       sensitive << a << b;
    }
};
```

```
// File: s_adder.cpp
#include "s_adder.h"

void s_adder ::prc_s_adder(){
 sum = a.read() + b.read();
}
```


Modeling Combinational Logic: Signed Arithmetic (2)

```
// File: s_adder_c.h
#include "systemc.h"

SC_MODULE(s_adder_c) {
    sc_in<sc_int<4> > a, b;
    sc_out <sc_int<4> > sum;
    sc_out <bool> carry_out;

    void prc_s_adder_c();

    SC_CTOR(s_adder_c) {
       SC_METHOD(prc_s_adder_c);
       sensitive << a << b;
    }
};
```

```
// File: s_adder.cpp
#include "s_adder_c.h"

void s_adder_c::prc_ s_adder_c(){
    sc_int<5> temp;

    temp = a.read() + b.read();
    sum = temp.range(3,0);
    carry_out = temp[4];
}
```


Modeling Combinational Logic: Relational Operators

```
// File: gt.h
#include "systemc.h"
const WIDTH=8;

SC_MODULE(gt) {
    sc_in<sc_int<4> > a, b;
    sc_out <bool> z;
    void prc_gt();
    SC_CTOR(gt) {
       SC_METHOD(prc_gt);
       sensitive << a << b;
    }
};
```

```
// File: gt.cpp
#include "gt.h"
void gt::prc_gt(){
 sc_int<WIDTH> atemp, btemp;

 atemp = a.read();
 btemp = b.read();
 z = sc_uint<WIDTH>(atemp.range(WIDTH/2-1,0)) >
     sc_uint<WIDTH>(btemp.range(WIDTH-1,WIDTH/2))
}
```

Modeling Combinational Logic: Vector and Ranges

```
// Bit or range select of a port
// or a signal is not allowed

sc_in<sc_uint<4> > data;
sc_signal<sc_bv<6> > counter;
sc_uint<4> temp;
sc_uint<6> cnt_temp;
bool mode, preset;

mode = data[2]; // not allowed
//instead
temp = data.read();
mode = temp[2];

counter[4] = preset; // not allowed
cnt_temp = counter;
cnt_temp[4] = preset;
counter = cnt_temp;
```


...

Multiple Processes and Delta Delay

```
// File: mult_proc.h
#include "systemc.h"

SC_MODULE(mult_proc) {
    sc_in<bool> in;
    sc_out<bool> out;

    sc_signal<bool> c1, c2;
    void mult_proc1();
    void mult_proc2();
    void mult_proc3();

    SC_CTOR(mult_proc) {
       SC_METHOD(mult_proc1);
       sensitive << in;
       SC_METHOD(mult_proc2);
       sensitive << c1;
       SC_METHOD(mult_proc3);
       sensitive << c2;
    }
};
```

```
// File: mult_proc.cpp
#include "mult_proc.h"

void mult_proc::mult_proc1(){
    c1 = !in;
}

void mult_proc::mult_proc2(){
    c2 = !c1;
}

void mult_proc::mult_proc3(){
    out = !c2;
}
```

If Statement

```
// File: simple_alu.h
#include “systemc.h”
const int WS=4;

SC_MODULE(simple_alu) {
    sc_in<sc_uint<WS> > a, b;
    sc_in<bool> ctrl;
    sc_out< sc_uint<WS> > z;

    void prc_simple_alu();

    SC_CTOR(simple_alu) {
       SC_METHOD(prc_simple_alu);
       sensitive << a << b << ctrl;
    }
};
```

```
// File: simple_alu.cpp
#include “simple_alu.h”

void
simple_alu::prc_simple_alu(){
   if(ctrl)
      z = a.read() & b.read();
   else
      z = a.read() | b.read();
}
```


If Statement: Priority encoder

```
// File: priority.h
#include “systemc.h”
const int IS=4;
const int OS=3;

SC_MODULE(priority) {
    sc_in<sc_uint<IS> > sel;
    sc_out< sc_uint<OS> > z;

    void prc_priority();

    SC_CTOR(priority) {
       SC_METHOD(prc_priority);
       sensitive << sel;
    }
};
```

```
// File: priority.cpp
#include “priority.h”

void priority::prc_priority(){
    sc_uint<IS> tsel;

    tsel = sel.read();

    if(tsel[0]) z = 0;
    else if (tsel[1]) z = 1;
    else if (tsel[2]) z = 2;
    else if (tsel[3]) z = 3;
    else z = 7;
}
```


Switch Statement: ALU

```
// File: alu.h
#include “systemc.h”
const int WORD=4;
enum op_type {add, sub, mul, div};

SC_MODULE(alu) {
    sc_in<sc_uint<WORD> > a, b;
    sc_in<op_type> op;
    sc_out< sc_uint<WORD> > z;

    void prc_alu();

    SC_CTOR(alu) {
       SC_METHOD(prc_alu);
       sensitive << a << b << op;
    }
};
```

```
// File: alu.cpp
#include “alu.h”

void priority::prc_alu(){
    sc_uint<WORD> ta, tb;

    ta = a.read();
    tb = b.read();
    switch (op) {
     case add: z = ta+tb; break;
     case sub: z = ta–tb; break;
     case mul: z = ta*tb; break;
     case div: z = ta/tb; break;
    }
}
```

Loops

- C++ loops: for, do-while, while
- SystemC RTL supports only for loops
- For loop iteration must be a compile time constant

Loops: An Example

```
// File: demux.h
#include "systemc.h"
const int IW=2;
const int OW=4;

SC_MODULE(demux) {
    sc_in<sc_uint<IW> > a;
    sc_out<sc_uint<OW> > z;

    void prc_demux();

    SC_CTOR(demux) {
       SC_METHOD(prc_demux);
       sensitive << a;
    }
};
```

```
// File: demux.cpp
#include "demux.h"

void priority::prc_demux(){
    sc_uint<3> j;
    sc_uint<OW> temp;

    for(j=0; j<OW; j++)
       if(a==j) temp[j] = 1;
       else temp[j] = 0;
}
```


Methods

- Methods other than SC_METHOD processes can be used in a SystemC RTL

Methods

```
// File: odd1s.h
#include “systemc.h”
const int SIZE = 6;

SC_MODULE(odd1s) {
    sc_in<sc_uint<SIZE> > data_in;
    sc_out<bool> is_odd;

    bool isOdd(sc_uint<SIZE> abus);
    void prc_odd1s();

    SC_CTOR(odd1s) {
       SC_METHOD(prc_odd1s);
       sensitive << data_in;
    }
};
```

```
// File: odd1s.cpp
#include “odd1s.h”

void odd1s::prc_odd1s(){
    is_odd = isOdd(data_in);
}

bool odd1s:isOdd
(sc_uint<SIZE> abus) {
    bool result;
    int i;

    for(i=0; i<SIZE; i++)
       result = result ^ abus[i];

    return(result);
}
```


Modeling Synchronous Logic: Flip-flops

```
// File: dff.h
#include “systemc.h”

SC_MODULE(dff) {
    sc_in<bool> d, clk;
    sc_out<bool> q;

    void prc_dff();

    SC_CTOR(dff) {
       SC_METHOD(prc_dff);
       sensitive_pos << clk;
    }
};
```

```
// File: dff.cpp
#include “dff.h”

void dff::prc_dff(){
    q = d;
}
```


Registers

```
// File: reg.h
#include “systemc.h”
const int WIDTH = 4;

SC_MODULE(reg) {
    sc_in<sc_uint<WIDTH> > cstate;
    sc_in<bool> clock;
    sc_out< sc_uint<WIDTH> > nstate;

    void prc_reg();

    SC_CTOR(dff) {
       SC_METHOD(prc_dff);
       sensitive_neg << clock;
    }
};
```

```
// File: reg.cpp
#include “reg.h”

void reg::prc_reg(){
    cstate = nstate;
}
```

Sequence Detector: “101”

```
// File: sdet.h
#include “systemc.h”

SC_MODULE(sdet) {
    sc_in<bool> clk, data;
    sc_out<bool> sfound;

    sc_signal<bool> first, second,
                    third;
    // synchronous logic process
    void prc_sdet();
    // comb logic process
    void prc_out();

    SC_CTOR(sdet) {
       SC_METHOD(prc_sdet);
       sensitive_pos << clk;
       SC_METHOD(prc_out);
       sensitive << first << second
         << third;
    }
};
```

```
// File: sdet.cpp
#include “sdet.h”

void sdet::prc_sdet(){
    first = data;
    second = first;
    third = second
}

void sdet::prc_out(){
    sfound = first &
    (!second) & third;
}
```


Counter: Up-down, Async Negative Clear

```
// File: cnt4.h
#include “systemc.h”
const int CSIZE = 4;
SC_MODULE(sdet) {
    sc_in<bool> mclk, cl, updown;
    sc_out<sc_uint<CSIZE> > dout;

    void prc_cnt4();

    SC_CTOR(cnt4) {
       SC_METHOD(prc_cnt4);
       sensitive_pos << mclk;
       sensitive_neg << cl;
    }
};
```

```
// File: cnt4.cpp
#include “cnt4.h”

void sdet::prc_cnt4(){
    if(!clear)
       data_out = 0;
    else
       if(updown)
         dout=dout.read()+1;
       else
         dout=dout.read()-1;
}
```