SystemC

Aleksandar Milenkovic
The University of Alabama in Huntsville
Email: milenka@ece.uah.edu

Outline

- Introduction
- Data Types
- Modeling Combinational Logic
- Modeling Synchronous Logic
- Miscellaneous Logic
- Modeling Examples

Memory Model

```
// file memory.h
#include "systemc.h"

const int WORD_SIZE = 8;
const int ADDR_SIZE = 6;
const int MEM_SIZE = 100;

SC_MODULE(memory) {
   sc_in<bool> en, rw, clk;
   sc_in<sc_uint<ADDR_SIZE> > addr;
   sc_inout<sc_lv<WORD_SIZE> > data;

   void prc_memory();
   sc_lv<WORD_SIZE> ram[MEM_SIZE];

   SC_CTOR(memory) {
      SC_METHOD(prc_memory);
      sensitive_neg << clk;
   }
};
```

Memory Model

```
// file memory.cpp
#include "memory.h"

void memory::prc_memory() {
   sc_lv<WORD_SIZE> allzs (sc_logic_Z);
   sc_lv<WORD_SIZE> allxs (sc_logic_X);

   if(en) {
     if(rw) { //read
       if ( addr.read() < MEM_SIZE)
              data = ram[addr.read()];
       else {
              data = allxs;
#ifndef SYNTHESIS
      cout << "address " << addr << "is out of range" << endl;
#endif
         } // end of else
     } // end of read
     else { // write
```

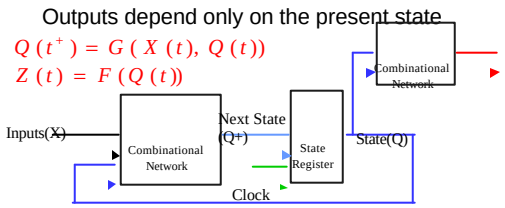

Memory Model

```
     else { // write
     if ( addr.read() < MEM_SIZE)
          ram[ addr.read()] = data;
  #ifndef SYNTHESIS
     else
          cout << "address " << addr << " is out of range" << endl;
  #endif
  else // en is not active
     data = allzs;
}
```


Modeling an FSM

- Moore Machine:
  Outputs depend only on the present state

$$Q(t^{+}) = G(X(t), Q(t))$$
$$Z(t) = F(Q(t))$$

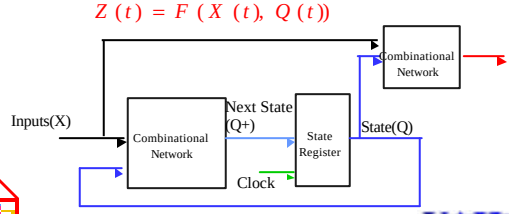


Modeling an FSM

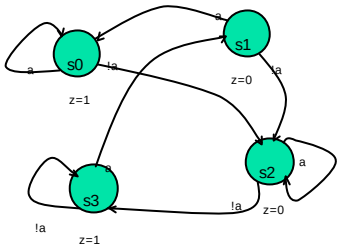
- Mealy Machine:
  Outputs depend on both the present state and inputs

$$Q(t^{+}) = G(X(t), Q(t))$$

$$Z(t) = F(X(t), Q(t))$$



Moore FSM



Moore Machine Example

```
// file moore.h
#include <systemc.h>

SC_MODULE(moore) {
   sc_in<bool> a, clk, reset;
   sc_out<bool> z;

   enum state_type {s0, s1, s2, s3};
   sc_signal<state_type> moore_st;
   void prc_moore();

   SC_CTOR(moore) {
      SC_METHOD(prc_moore);
      sensitive_pos << clk;
   }
};
```

Moore Machine Example

```
// file moore.cpp
#include "moore.h"

void moore:prc_moore() {
   if(reset)
      moore_st = s0;
   else
      switch(moore_st) {
         case s0: z=1; moore_st = a ? s0 : s2; break;
         case s1: z=0; moore_st = a ? s0 : s2; break;
         case s2: z=0; moore_st = a ? s2 : s3; break;
         case s3: z=1; moore_st = a ? s1 : s3; break;
      }
}
```

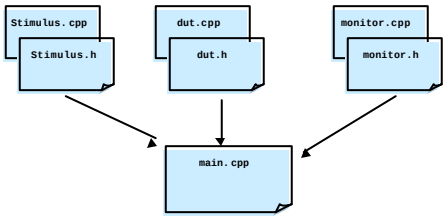

Outline

- Introduction
- Data Types
- Modeling Combinational Logic
- Modeling Synchronous Logic
- Miscellaneous Logic
- Modeling Examples
- Writing Testbenches

Why Testbenches

- Testbench: a model used to exercise and verify the correctness of a DUT (design-under-test)
- Testbenches
  - Generate stimulus for simulation
  - Apply the stimulus to the DUT
  - Collect outputs and compare with the expected results

Typical Testbench Structure



Typical structure of main.cpp

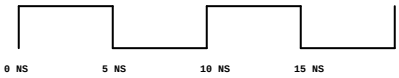
```
// file main.cpp
// # include files here (stimulus, dut, monitor)
int sc_main(int argc, char *argv[]) {
   // sc_signal declarations for signals
   // inteconnecting various modules

   // sc_clock clock declarations

   // DUT, stimulus, monitor module instantiations

   // trace file creation and monitoring
   // simulation start control
}
```


Simulation Control

- sc_clock : generate a clock signal
- sc_trace : dump trace information into a file
- sc_start : run simulation for specified time
- sc_stop : stop simulation
- sc_time_stamp : get current simulation time with time units
- sc_simulation_time : get current simulation time without time units
- sc_cycle, sc_initialize : perform clock-cycle simulation
- sc_time : specify a time value

sc_clock

```
// clock declaration
// default duty cycle is 50%, default initial value is 1
sc_clock rclk ("rclk", 10, SC_NS);
```




```
// duty cycle is 0.2, first edge occurs after 5 ns, and
// the first edge is false (0)
sc_clock mclk ("mclk", 10, SC_NS, 0.2, 5, SC_NS, false);

// default period is 1 ns; duty is 50%, initial value is 1
sc_clock clocka("clka");
```


sc_trace

- 3 formats for simulation results
  - VCD: Value Change Dump
  - WIF: Waveform Interchange Format
  - ISDB: Integrated Signal Database

```
// open a trace file
// sc_create_vcd_trace_file(), vcd extension added
// sc_create_wif_trace_file()
// sc_create_isdb_trace_file()
sc_trace_file *tfile = sc_create_vcd_trace_file("myvcddump");

// specify signals for dump
sc_trace(tfile, signal_name, "signal_name");
// close file
sc_close_vcd_trace_file(tfile);
```


sc_start, sc_stop

- Start simulation kernel
  - follows all instantiations and the trace calls
- Stop simulation

```
// run for 100 ms
sc_start(100, SC_MS);
// run forever
sc_start(-1);

// used in any process to stop simulation
sc_stop();
```

sc_time_stamp, sc_simulation_time

```
// returns current simulation time
cout << "Current time is " << sc_time_stamp() << endl

// returns an integer value of type double in terms of
// default time unit
double ctime = sc_simulation_time();
```


sc_cycle, sc_initialize

```
// initialize simulation kernel
sc_initialize();

// executes all the processes ready to run
// (could take a number of deltas) until no more are ready
// advance simulation time to 10 microseconds

sc_cycle(10, SC_US);
```


sc_time

```
// specifies time value
sc_time t1(100, SC_NS); // value of 100 ns

// units: SC_FS, SC_PS, SC_NS, SC_US, SC_MS, SC_SEC
// default time resolution is 1ps
// to set time resolution to 100 pico seconds
sc_time_resolution(100, SC_PS)
```

Arbitrary Waveform Generation

```
// file: wave.h
#include "systemc.h"
SC_MODULE(wave) {
   sc_out<bool> sig_out;

   void prc_wave();

   SC_CTOR(wave) {
      SC_THREAD(prc_wave);
   }
}
```

```
// file: wave.cpp
#include "wave.h"
void wave::prc_wave() {
   sig_out = 0;
   wait(5, SC_NS);
   sig_out = 1;
   wait(2, SC_NS);
   sig_out = 0;
   wait(5, SC_NS);
   sig_out = 1;
   wait(8, SC_NS);
   sig_out = 0;
}
```


Arbitrary Waveform Generation

Generating a Derived Clock

Reading Stimuli from Files

```
// read_vectors.h
#include <fstream.h>
#include <iostream.h>
#include "systemc.h"
#include "usr_define.h"

SC_MODULE(read_vectors) {
  sc_in<bool> rd_clk;
  sc_out<sc_uint<SEL_WIDTH> > rd_sel_op;
  sc_out<sc_bool> rd_clear, rd_leftin, rd_rightin;
  sc_out<sc_uint<WIDTH> > rd_datain, rd_usr_out;

  void prc_read_vectors();
  ifstream infile;
```


Reading Stimuli from Files

```
SC_CTOR(read_vectors) {
  SC_METHOD(prc_read_vectors);
  sensitive_neg << rd_clk;
  infile.open("usr.in");
  if(!infile) {
    cout << "**ERROR: Unable to open input file, usr.in " << endl;
    sc_stop();
  }
}
```


Reading Stimuli from Files

```
// read_vectors.cpp
#include "read_vectors.h"
#include "usr.h"

void read_vectors::prc_read_vectors() {
  bool t_clr, t_lin, t_rin;
  int t_din, t_dout, t_sel;

  if (infile >> t_sel >> t_clr >> t_lin
    >> t_rin >> t_din >> t_dout) {
    cout << "Reading line(" << sc_time_stamp() << "): sel="
        << t_sel << " clr=" << t_clr << " lin=" << t_lin
        << " rin=" << t_rin << " din=" << hex << t_din
        << " dout=" << t_dout << endl;
    rd_clear = t_clr; rd_leftin = t_lin; rd_rightin = t_rin;
    rd_datain = t_din; rd_usr_out = t_dout; rd_sel_op = t_sel;
  }
  else
    sc_stop();
}
```