SystemC – Overview (00A)

SystemC

Young Won Lim
05/14/2012

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Young Won Lim
05/14/2012

Based on the following original work

[1] Aleksandar Milenkovic, 2002
CPE 626 The SystemC Language – VHDL, Verilog Designer's Guide
http://www.ece.uah.edu/~milenka/ce626-02S/lectures/cpe626-SystemC-L2.ppt

[2] Alexander de Graaf, EEMCS/ME/CAS, 2010
SystemC: an overview ET 4351
ens.ewi.tudelft.nl/Education/courses/et4351/SystemC-2010v1.pdf

[3] Joachim Gerlach, 2001
System-on-Chip Design with Systent of Computer Engineering
http://www2.cs.uni-paderborn.de/cs/ag-hardt/Forschung/Data/SystemC-Tutorial.pdf

[4] Martino Ruggiero, 2008
SystemC
polimage.polito.it/~lavagno/codes/SystemC_Lezione.pdf

[5] Deepak Kumar Tal, 1998-2012
SystemC Tutorial
http://www.asic-world.com/systemc/index.html

DFF (1) – VHDL & SystemC

```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
 port(clock : in std_logic;
        din : in std_logic;
        dout : out std_logic);
end dff;

architecture rtl of dff is
begin
  process
   begin
    wait until clock'event and clock = '1';
    dout <= din;
  end process;
end rtl;
```

```
// dff.h
#include "systemc.h"

SC_MODULE(dff)
{
  sc_in<bool> din;
  sc_in<bool> clock;
  sc_out<bool> dout;

  void doit()
  {
     dout = din;
  };

  SC_CTOR(dff)
  {
    SC_METHOD(doit);
    sensitive_pos << clock;
  }
};
```

DFF (2) – VHDL & SystemC

```
library ieee;
use ieee.std_logic_1164.all;
entity dffa is
  port( clock : in std_logic;
        reset : in std_logic;
        din : in std_logic;
        dout : out std_logic);
end dffa;

architecture rtl of dffa is
begin
  process(reset, clock)
  begin
    if reset = '1' then
        dout <= '0';
    elsif clock'event and clock = '1' then
        dout <= din;
    end if;
  end process;
end rtl;
```

```
// dffa.h
#include "systemc.h"
SC_MODULE(dffa)
{
  sc_in<bool> clock;
  sc_in<bool> reset;
  sc_in<bool> din;
  sc_out<bool> dout;

  void do_ffa()  {
    if (reset) {
       dout = false;
    } else if (clock.event()) {
        dout = din;           }
  };

  SC_CTOR(dffa) {
    SC_METHOD(do_ffa);
      sensitive(reset);
      sensitive_pos(clock);
  }
};
```

Shifter – VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity shift is
  port( din : in std_logic_vector(7 downto 0);
        clk : in std_logic;
        load : in std_logic;
        LR : in std_logic;
        dout : out std_logic_vector(7 downto 0));
end shift;

architecture rtl of shift is
  signal shiftval : std_logic_vector(7 downto 0);
begin
 nxt: process(load, LR, din, dout)
 begin
     
 end process;
end rtl;
```

```
if load = '1' then
  shiftval <= din;
elsif LR = '0' then
  shiftval(6 downto 0)
  <= dout(7 downto 1);
  shiftval(7) <= '0';
elsif LR = '1' then
  shiftval(7 downto 1)
  <= dout(6 downto 0);
  shiftval(0) <= '0';
end if;
```

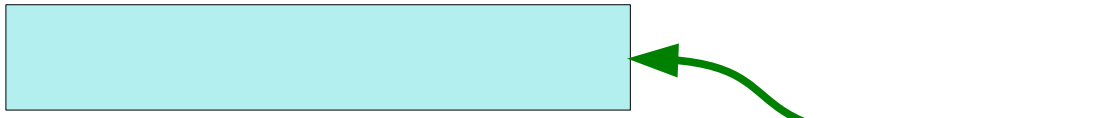
Shifter - SystemC

```
// shift.h
#include “systemc.h”
SC_MODULE(shift)
{
  sc_in<sc_bv<8> > din;
  sc_in<bool> clk;
  sc_in<bool> load;
  sc_in<bool> LR;
  sc_out<sc_bv<8> > dout;
  sc_bv<8> shiftval;

  void shifty();

  SC_CTOR(shift)
  {
     SC_METHOD(shifty);
       sensitive_pos (clk);
  }
};
```

```
// shift.cc
#include “shift.h”
void shift::shifty()
{
  if (load) {
     shiftval = din;
  } else if (!LR) {
     shiftval.range(6,0) = shiftval.range(7,1);
     shiftval[7] = ‘0’;
  } else if (LR) {
     shiftval.range(7,1) = shiftval.range(6,0);
     shiftval[0] = ‘0’;
  }
  dout = shiftval;
}
```

Counter – VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port( clock : in std_logic;
        load : in std_logic;
        clear : in std_logic;
        din : in std_logic_vector(7 downto 0);
        dout : inout std_logic_vector(7 downto 0));
end counter;

architecture rtl of counter is
  signal countval : std_logic_vector(7 downto 0);
begin
 process
 begin
   wait until clock'event and clock = '1';
   dout <= countval;
 end process;
end rtl;
```



```
process(load, clear, din, dout)
begin
  if clear = '1' then
    countval <= "00000000";
  elsif load = '1' then
    countval <= din;
  else
    countval <= dout + "00000001";
  end if;
end process;
```

Counter – SystemC

```
#include "systemc.h"
SC_MODULE(counter)
{
  sc_in<bool> clock;
  sc_in<bool> load;
  sc_in<bool> clear;
  sc_in<sc_int<8> > din;
  sc_out<sc_int<8> > dout;
  int countval;

  void onetwothree();

  SC_CTOR(counter)
  {
     SC_METHOD(onetwothree);
       sensitive_pos (clock);
  }
};
```

```
// counter.cc
#include "counter.h"
void counter::onetwothree()
{
  if (clear) {
     countval = 0;
  } else if (load) {
  countval = din.read(); // use read when a
  // type conversion is happening
  // from an input port
  } else {
    countval++;
   }
   dout = countval;
}
```

State Machine – VHDL (1)

```
package vm_pack is
  type t_vm_state is (main_st, review_st, repeat_st,   save_st, erase_st, send_st,
                      address_st, record_st, begin_rec_st, message_st);
  type t_key is (‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’,    ‘7’, ‘8’, ‘9’, ‘*’, ‘#’);
end vm_pack;

use work.vm_pack.all;
library ieee;
use ieee.std_logic_1164.all;
entity stmach is
        port(    clk : in std_logic;
                 key : in t_key;
                 play, recrd, erase, save, address : out std_logic);
end stmach;
```

State Machine – VHDL (2)

```
architecture rtl of stmach is
signal next_state, current_state : t_vm_state;
begin

process(current_state, key)
begin
  play <= '0';   save <= '0';
  erase <= '0';  recrd <= '0';
  address <= '0';
  case current_state is
    when main_st =>
    when review_st =>
    when repeat_st =>
    when save_st =>
    when erase_st =>
    when send_st =>
    when address_st =>
    when record_st =>
    when begin_rec_st =>
    when message_st =>
    end case;
end process;

process
begin
  wait until clk'event and clk = '1';
  current_state <= next_state;
end process;

end rtl;
```

State Machine – VHDL (3)

```
when main_st =>
        if key = ‘1’ then
            next_state <= review_st;
        elsif key = ‘2’ then
            next_state <= send_st;
        else
            next_state <= main_st;
        end if;
when review_st =>
        if key = ‘1’ then
            next_state <= repeat_st;
        elsif key = ‘2’ then
            next_state <= save_st;
        elsif key = ‘3’ then
            next_state <= erase_st;
        elsif key = ‘#’ then
            next_state <= main_st;
        else
            next_state <= review_st;
        end if;
when repeat_st =>
        play <= ‘1’;
        next_state <= review_st;
when save_st =>
        save <= ‘1’;
        next_state <= review_st;
when erase_st =>
        erase <= ‘1’;
        next_state <= review_st;
```

```
when send_st =>
        next_state <= address_st;
when address_st =>
        address <= ‘1’;
        if key = ‘#’ then
            next_state <= record_st;
        else
            next_state <= address_st;
        end if;
when record_st =>
        if key = ‘5’ then
            next_state <= begin_rec_st;
        else
            next_state <= record_st;
        end if;
when begin_rec_st =>
        recrd <= ‘1’;
        next_state <= message_st;
when message_st =>
        recrd <= ‘1’;
        if key = ‘#’ then
            next_state <= send_st;
        else
            next_state <= message_st;
        end if;
```

State Machine – SystemC (1)

```
// stmach.h
#include “systemc.h”
enum vm_state {
  main_st, review_st, repeat_st,
  save_st, erase_st, send_st,
  address_st, record_st,
  begin_rec_st, message_st
};

SC_MODULE(stmach)
{
  sc_in<bool> clk;
  sc_in<char> key;
  sc_out<sc_logic> play;
  sc_out<sc_logic> recrd;
  sc_out<sc_logic> erase;
  sc_out<sc_logic> save;
  sc_out<sc_logic> address;
  sc_signal<vm_state> next_state;
  sc_signal<vm_state> current_state;
```

```
  void getnextst();
  void setstate();
```

```
SC_CTOR(stmach)
{
  SC_METHOD(getnextst);
     sensitive << key << current_state;
  SC_METHOD(setstate);
    sensitive_pos (clk);
}
};
```

State Machine – SystemC (2)

```
// stmach.cc
#include “stmach.h”
void stmach::getnextst()
{
  play = sc_logic_0;
  recrd = sc_logic_0;
  erase = sc_logic_0;
  save = sc_logic_0;
  address = sc_logic_0;
  switch (current_state) {
    case main_st:
      if (key == ‘1’) {
         next_state = review_st;
      } else {
        if (key == ‘2’) {
           next_state = send_st;
        } else {
           next_state = main_st;
        }
      }
```

```
case review_st:
  if (key == ‘1’) {
      next_state = repeat_st;
  } else {
    if (key == ‘2’) {
       next_state = save_st;
    } else {
      if (key == ‘3’) {
         next_state = erase_st;
      } else {
        if (key == ‘#’) {
          next_state = main_st;
        } else {
          next_state = review_st;
        }
      }
    }
  }
```

State Machine – SystemC (3)

```
case repeat_st:
   play = sc_logic_1;
   next_state = review_st;
case save_st:
   save = sc_logic_1;
   next_state = review_st;
case erase_st:
   erase = sc_logic_1;
   next_state = review_st;
case send_st:
   next_state = address_st;
case address_st:
   address = sc_logic_1;
   if (key == '#') {
      next_state = record_st;
   } else {
      next_state = address_st;
   }
```

```
   case record_st:
      if (key == '5') {
        next_state = begin_rec_st;
      } else {
        next_state = record_st;
      }
    case begin_rec_st:
      recrd = sc_logic_1;
      next_state = message_st;
    case message_st:
      recrd = sc_logic_1;
      if (key == '#') {
         next_state = send_st;
      } else {
         next_state = message_st;
      }
  } // end switch
} // end method

void stmach::setstate()
{
  current_state = next_state;
}
```

Memory – VHDL (1)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram is
  port (enable : in std_logic;
        readwr : in std_logic;
        addr : in std_logic_vector(7 downto 0);
        data : inout std_logic_vector(15 downto 0)
  );
end ram;
```

DFF (2) – VHDL & SystemC

```
architecture rtl of ram is
begin
  process(addr, enable, readwr)
    subtype data16 is std_logic_vector(15 downto 0);
    type ramtype is array(0 to 255) of data16;
    variable ramdata : ramtype;
  begin
     if (enable = '1') then
       if readwr = '0' then
          data <= ramdata(conv_integer(addr));
       elsif readwr = '1' then
          ramdata(conv_integer(addr)) := data;
       end if;
     else
       data <= "ZZZZZZZZZZZZZZZ";
     end if;
  end process;
end rtl;
```

Memory – SystemC (1)

```
// ram.h
#include “systemc.h”
SC_MODULE(ram)
{
  sc_in<sc_int<8> > addr;
  sc_in<bool> enable;
  sc_in<bool> readwr;
  sc_inout_rv<16> data;

  void read_data();
  void write_data();
  sc_lv<16> ram_data[256];

SC_CTOR(ram)
{
  SC_METHOD(read_data);
    sensitive << addr << enable << readwr;
  SC_METHOD(write_data);
   sensitive << addr << enable << readwr << data;
}
};
```

Memory – SystemC (2)

```
// ram.cc
#include “ram.h”
void ram::read_data()
{
  if (enable && ! readwr ) {
    data = ram_data[addr.read()];
  } else {
    data = “ZZZZZZZZZZZZZZZ”;
  }
}

void ram::write_data()
{
  if (enable && readwr) {
    ram_data[addr.read()] = data;
  }
}
```

1-Bit Adder

```
// Adder.h
 #include “systemc.h”

 SC_MODULE(Adder){
    sc_out<sc_logic> sum, Cout;
    sc_in<sc_logic> A, B, Cin;

    void add(){
        sc_logic tempC, tempD, tempE;
        tempC = A.read() & B.read();
        tempD = A.read() ^ B.read();
        tempE = Cin.read() & tempD.read();
        sum.write(tempD ^ Cin.read());
        cout.write(tempC | tempE);
      }

    SC_CTOR(Adder){
            SC_METHOD(add);
            sensitive << A << B << Cin;
    }
};
```

1-Bit Adder Testbench

```
//Testbench.h
#include “systemc.h”

SC_MODULE(Testbench){
   sc_out<sc_logic> TA, TB, TCin;

   void testprocess()
   {
            TA.write(SC_LOGIC_0);
            TB.write(SC_LOGIC_0);
            TCin.write(SC_LOGIC_0);
            wait(5, SC_NS);
            TA.write(SC_LOGIC_1);
            TB.write(SC_LOGIC_1);
            TCin.write(SC_LOGIC_0);
            wait(10, SC_NS);
            sc_stop(); //Stop simulation
   }

   SC_CTOR(Testbench){
            SC_THREAD(testprocess);
   }
};
```

1-Bit Adder TOP

```
//AdderMain.cpp

#include “Adder.h”
#include “Testbench.h”
int main(int argc, char *argv[]){
   sc_signal<sc_logic> A, B, Cin;
   sc_signal<sc_logic> sum, Cout;

   Adder adder1bit(“BitAdder”);
   Testbench tb(“Testbench”);

   adder1bit.A(A);
   adder1bit.B(B);
   adder1bit.Cin(Cin);
   adder1bit.sum(sum);
   adder1bit.Cout(Cout);

   tb.TA(A);
   tb.TB(B);
   tb.TCin(Cin);

   //Start the simulation for 200ns
   sc_start(200, SC_NS);
   return 0;

}
```

2-Bit Adder

```
#include "Adder.h"
SC_MODULE(Adder2Bits) {
    sc_in<sc_logic> A0, B0, A1, B1,Cin;
    sc_out<sc_logic> sum0, sum1, Cout;
    sc_signal<sc_logic> CoutConnection;

   Adder adder1bit0, adder1bit1;

   SC_CTOR(Adder2Bits):
   adder1bit0("Adder0"),
   adder1bit1("Adder1")
   {
      adder1bit0.A(A0);
      adder1bit0.B(B0);
      adder1bit0.Cin(Cin);
      adder1bit0.sum(sum0);
      adder1bit0.Cout(CoutConnection);

      adder1bit1.A(A1);
      adder1bit1.B(B1);
      adder1bit1.Cin(CoutConnection);
      adder1bit1.sum(sum1);

      adder1bit1.Cout(Cout);
   }
};
```

FIFO Memory (1)

```
#include "systemc.h“

SC_MODULE(FIFOMemory) {
 sc_in<bool> RD; // ReadNotWrite
 sc_inout<int> data;
 sc_in_clk clock;
 sc_out<bool> status; // 0:NOERROR, 1:ERROR

 sc_fifo<int> buffer;
 const int size;

 void memory_process() {
     * * *
 }

 SC_HAS_PROCESS(FIFOMemory);

 FIFOMemory(sc_module_name name_,int size_=10):
   module(name_), size(size_), buffer(size_)
   {
      SC_THREAD(memory_process);

      sensitive << clock.pos();

    }
};
```

```
void memory_process() {
  int temp;
  while(true) {
    wait();
    if (RD.read() == true) {
       if(buffer.nb_read(temp)) {
          status.write(0);
          data.write(temp);
       } else {
          status.write(1);
       }
     } else {
       temp = data.read();
       if(buffer.nb_write(temp)) {
          status.write(0);
       } else {
          status.write(1);
       }
     }
   }
}
```

FIFO Memory (2)

```
#include "systemc.h"
SC_MODULE(Testbench){
  sc_in_clk clock; sc_inout<int> data;
  sc_out<bool> RD; sc_in<bool> status;

  void testprocess(){

     * * *
   }


   SC_CTOR(Testbench){
     SC_THREAD(testprocess);
     sensitive << clock.pos();
   }
};
```

```
void testprocess(){
  wait();
  data.write(0x010);
  RD.write(0);
  wait();
  cout << "@: " << sc_time_stamp() << ":  WR " <<
  data.read() << ": STATUS: " <<
  status.read() << endl;

  data.write(0x020);
  RD.write(0);
  wait();
  cout << "@: " << sc_time_stamp() << ":  WR " <<
  data.read() << ": STATUS: " <<
  status.read() << endl;
   // write 0x030, 0x040, 0x050, 0x060not shown


  RD.write(1);
  wait();
  cout << "@: " << sc_time_stamp() << ":  RD " <<
  data.read() << ": STATUS: " <<
  status.read() << endl;

   //also read four more times
 }
```

FIFO Memory (3)

```
#include "FIFOMemory.h"
#include "Testbench.h"
int sc_main(int argc, char *argv[]){
  sc_signal<int> data;
  sc_signal<bool> RD, status;

  sc_clock sysclock("clock1", 20, SC_NS);
  FIFOMemory memory1(“Memory1”);
  Testbench test1(“Test1”);

  memory1.clock(sysclock);

  memory1.data(data);
  memory1.RD(RD);
  memory1.status(status);
  test1.clock(sysclock);
  test1.data(data);
  test1.RD(RD);
  test1.status(status);
}
```

TMP

References

[1] Aleksandar Milenkovic, 2002
CPE 626 The SystemC Language – VHDL, Verilog Designer's Guide
http://www.ece.uah.edu/~milenka/ce626-02S/lectures/cpe626-SystemC-L2.ppt

[2] Alexander de Graaf, EEMCS/ME/CAS, 2010
SystemC: an overview ET 4351
ens.ewi.tudelft.nl/Education/courses/et4351/SystemC-2010v1.pdf

[3] Joachim Gerlach, 2001
System-on-Chip Design with Systent of Computer Engineering
http://www2.cs.uni-paderborn.de/cs/ag-hardt/Forschung/Data/SystemC-Tutorial.pdf

[4] Martino Ruggiero, 2008
SystemC
polimage.polito.it/~lavagno/codes/SystemC_Lezione.pdf

[5] Deepak Kumar Tal, 1998-2012
SystemC Tutorial
http://www.asic-world.com/systemc/index.html