NC State University
ECE Department ECE 720 W. Rhett Davis

Homework #3 Solution

Problem 1)

Below is the listing of one possible solution to this problem. This page contains the UART module.

```
#include <systemc.h>
#include "read_master_if.h"
#include "CPU.h"
#include <list>

using namespace std;

class UART : public sc_module, read_master_if<char>
{
public:
  sc_export<read_master_if<char> > slave;
  list<char> buffer;

  SC_HAS_PROCESS(UART);

  UART(sc_module_name name) : sc_module(name)  {
    SC_THREAD(main);
    slave.bind(*this);
  }

  void read(int addr, char &c) {
    if (addr == 0) {
         if (buffer.size()>0) c=1;
         else c=0;
         return;
       }
       else {
      if (buffer.size()>0) {
           c=buffer.front();
           buffer.pop_front();
         }
         else c=0;
         return;
       }
  }

  void main() {
    const char *str = "Hello, World!\n";
       const char *p = str;

    while (true) {
         cout << sc_time_stamp().to_string() << " UART receiving "
<< *p << endl;
         buffer.push_back(*p++);
         if (!*p) p=str;
         wait(100, SC_NS);
    }
  }
};

}
```

This page contains the top-level module for the system.

```
class top : public sc_module
{
public:
  CPU cpu_inst;
  UART uart_inst;

  top(sc_module_name name) :
    sc_module(name),
    cpu_inst("cpu0"),
    uart_inst("uart0")
  {
    cpu_inst.master(uart_inst.slave);
  }
};

int sc_main (int argc, char *argv[])
{

  top top1("top0");
  sc_start(1000, SC_NS);
  cout << endl << endl;
  return 0;
}
```

This page contains the simulation output for the system.

```
0 s CPU read 0 from address 0
0 s UART receiving H
20 ns CPU read 1 from address 0
40 ns CPU read H from address 1
60 ns CPU read 0 from address 0
80 ns CPU read 0 from address 0
100 ns UART receiving e
100 ns CPU read 1 from address 0
120 ns CPU read e from address 1
140 ns CPU read 0 from address 0
160 ns CPU read 0 from address 0
180 ns CPU read 0 from address 0
200 ns UART receiving l
200 ns CPU read 1 from address 0
220 ns CPU read l from address 1
240 ns CPU read 0 from address 0
260 ns CPU read 0 from address 0
280 ns CPU read 0 from address 0
300 ns UART receiving l
300 ns CPU read 1 from address 0
320 ns CPU read l from address 1
340 ns CPU read 0 from address 0
360 ns CPU read 0 from address 0
380 ns CPU read 0 from address 0
400 ns UART receiving o
400 ns CPU read 1 from address 0
420 ns CPU read o from address 1
440 ns CPU read 0 from address 0
460 ns CPU read 0 from address 0
480 ns CPU read 0 from address 0
500 ns UART receiving ,
500 ns CPU read 1 from address 0
520 ns CPU read , from address 1
540 ns CPU read 0 from address 0
560 ns CPU read 0 from address 0
580 ns CPU read 0 from address 0
600 ns UART receiving
600 ns CPU read 1 from address 0
620 ns CPU read   from address 1
640 ns CPU read 0 from address 0
660 ns CPU read 0 from address 0
680 ns CPU read 0 from address 0
700 ns UART receiving W
700 ns CPU read 1 from address 0
720 ns CPU read W from address 1
740 ns CPU read 0 from address 0
760 ns CPU read 0 from address 0
780 ns CPU read 0 from address 0
800 ns UART receiving o
800 ns CPU read 1 from address 0
820 ns CPU read o from address 1
840 ns CPU read 0 from address 0
860 ns CPU read 0 from address 0
880 ns CPU read 0 from address 0
900 ns UART receiving r
900 ns CPU read 1 from address 0
920 ns CPU read r from address 1
940 ns CPU read 0 from address 0
960 ns CPU read 0 from address 0
980 ns CPU read 0 from address 0
```

Problem 2)

The code below shows the new version of the producer:

```
class producer : public sc_module, sc_fifo_in_if<char>
{
public:
  sc_export<sc_fifo_in_if<char> > out;
  sc_event read_event, write_event;
  list<char> fifo;
  int fifo_max;

  SC_HAS_PROCESS(producer);

  producer(sc_module_name name, int size) : sc_module(name) {
    SC_THREAD(main);
    out.bind(*this);
    fifo_max=size;
  }

  bool nb_read(char &c) { return true; } // Not implemented
  const sc_event & data_written_event() const { return write_event; }
  char read() { return 0; } // Not implemented
  int num_available() const { return fifo.size(); }
  void read( char &c ) {
    while (num_available() == 0)
      wait( write_event );
    c=fifo.front();
    fifo.pop_front();
    read_event.notify(SC_ZERO_TIME);
  }

  void main () {
    const char *str = "Hello, World!\n";
    const char *p = str;

    while (true) {
      if (rand()%3==0) {  // 1-in-3 chance of executing
        cout << sc_time_stamp().to_string() << " (" << (fifo_max - fifo.size()) ;
        cout << " free) producer writing " << *p << " to fifo\n";
        if (fifo.size() == fifo_max)
          wait( read_event );
        fifo.push_back(*p++);
        write_event.notify(SC_ZERO_TIME);
        cout << sc_time_stamp().to_string() << " (" << (fifo_max - fifo.size());
        cout << " free) producer wrote to fifo\n";
        if (!*p) p=str;
      }

      wait(10, SC_NS);
    }
  }
};
```

The code below shows the new top module function, with differences highlighted in red. As with the *export_fifo* example, there is no fifo channel. The consumer and sc_main function are unchanged. The output is also unchanged, though your output may differ slightly.

```
class top : public sc_module
{
public:
  producer prod_inst;
  consumer cons_inst;

  top(sc_module_name name, int size) :
    sc_module(name),
    prod_inst("Producer1",size),
    cons_inst("Consumer1")
  {
    cons_inst.in(prod_inst.out);
  }
};
```