NC State University
ECE Department ECE 720 W. Rhett Davis

Homework #4 Solution

Problem 1)

The following boxes show the modifications to each file that was provided, with the changes in bold, red type.

stub.h, stub.cpp, hw04p1.cpp: no change, other than to comment-out the “cout” statement to reduce the volume of output.

SimpleBusLT.h: added member variables to count transactions, code to initialize variables in the constructor, code to increment variables in transport method, a new method to print out the result after 10 us, and registered process in the constructor.

```
using namespace std;
. . .
public:
  target_socket_type target_socket[NR_OF_INITIATORS];
  initiator_socket_type initiator_socket[NR_OF_TARGETS];
  unsigned long bytes_read;
  unsigned long bytes_written;
public:
  SC_HAS_PROCESS(SimpleBusLT);
  SimpleBusLT(sc_core::sc_module_name name) :
    sc_core::sc_module(name),bytes_read(0),bytes_written(0)
  {
. . .
    SC_THREAD(main);
  }

void main(void)
  {
    wait(sc_core::sc_time(10000,sc_core::SC_NS));
    cout << sc_core::sc_time_stamp() << " " << sc_object::name() << endl;
    cout << "  " << bytes_read/0.00001 << " bytes read per sec" << endl;
    cout << "  " << bytes_written/0.00001 << " bytes written per sec" << endl;
  }
. . .

void initiatorBTransport(int SocketId,
        transaction_type& trans,sc_core::sc_time& t)
  {
. . .
    (*decodeSocket)->b_transport(trans, t);

    if (trans.get_command()==tlm::TLM_WRITE_COMMAND)
      bytes_written+=trans.get_data_length();
    if (trans.get_command()==tlm::TLM_READ_COMMAND)
      bytes_read+=trans.get_data_length();
```

mem.h: member variables added to record the number of bytes written & read, as well as a process to do the counting.

```
  tlm_utils::simple_target_socket<mem>  slave;

  void main(void);

  private:

  sc_dt::uint64 m_memory_size;
  unsigned long bytes_read;
  unsigned long bytes_written;

  void custom_b_transport
  ( tlm::tlm_generic_payload &gp, sc_core::sc_time &delay );
```

mem.cpp: Initialization of count variables & registration of process in constructor, implementation of process to count reads & writes, commented out “cout” lines in transport method, and added code to increment counts.

```
SC_HAS_PROCESS(mem);
mem::mem( sc_core::sc_module_name module_name, sc_dt::uint64 memory_size )
  : sc_module (module_name)
  , m_memory_size (memory_size)
  , bytes_read(0), bytes_written(0)
{
  slave.register_b_transport(this, &mem::custom_b_transport);
  SC_THREAD(main);
}

void mem::main(void)
{
  wait(sc_core::sc_time(10000,sc_core::SC_NS));
  cout << sc_core::sc_time_stamp() << " " << sc_object::name() << endl;
  cout << "  " << bytes_read/0.00001 << " bytes read per sec" << endl;
  cout << "  " << bytes_written/0.00001 << " bytes written per sec" << endl;
}

void
mem::custom_b_transport
 ( tlm::tlm_generic_payload &gp, sc_core::sc_time &delay )
{
. . .
  wait(delay+mem_delay);
  //cout << sc_core::sc_time_stamp() << " " << sc_object::name();
  if (address < m_memory_size) {
    switch (command) {
      case tlm::TLM_WRITE_COMMAND:
      {
        //cout << " WRITE 0x" << hex << address << endl;
        gp.set_response_status( tlm::TLM_OK_RESPONSE );
        bytes_written+=gp.get_data_length();
        break;
      }
      case tlm::TLM_READ_COMMAND:
      {
        //cout << " READ 0x" << hex << address << endl;
        gp.set_response_status( tlm::TLM_OK_RESPONSE );
        bytes_read+=gp.get_data_length();
        break;
      }
```

top.h: Increased the number of initiators

```
class top : public sc_core::sc_module
{
public:

  top (sc_core::sc_module_name name);

  private:

  SimpleBusLT<3, 2> bus;
  mem mem0;
  mem mem1;
  stub stub0;
  stub stub1;
  stub stub2;
};
```

top.cpp: Initialized the new initiator and bound its socket

```
top::top(sc_core::sc_module_name name)
  : sc_core::sc_module(name)
  , bus("bus")
  , mem0("mem0", 4*1024)
  , mem1("mem1", 4*1024)
  , stub0("stub0","xact0.txt")
  , stub1("stub1","xact1.txt")
  , stub2("stub2","xact2.txt")

{
  stub0.master(bus.target_socket[0]);
  stub1.master(bus.target_socket[1]);
  stub2.master(bus.target_socket[2]);
  bus.initiator_socket[0](mem0.slave);
  bus.initiator_socket[1](mem1.slave);
}
```

The output of the simulation is given below:

```
360 ns top.stub0 Completed
360 ns top.stub1 Completed
10 us top.bus
  1.212e+08 bytes read per sec
  9.73e+07 bytes written per sec
10 us top.mem0
  5.08e+07 bytes read per sec
  5.05e+07 bytes written per sec
10 us top.mem1
  7.04e+07 bytes read per sec
  4.68e+07 bytes written per sec
10027 ns top.stub2 Completed
```

Problem 2) The code to implement this behavior is given below

CPU.h

```
#pragma once
#include <tlm.h>

class CPU: public sc_core::sc_module
  , virtual public tlm::tlm_bw_transport_if<>
{

public:
  tlm::tlm_initiator_socket<> master;

  SC_HAS_PROCESS(CPU);
  CPU(sc_core::sc_module_name name);

  void main ();

private:

  // Not implemented, but required by interface
  void invalidate_direct_mem_ptr
  (sc_dt::uint64 start_range, sc_dt::uint64 end_range)
  { return; }

  // Not implemented, but required by interface
  tlm::tlm_sync_enum nb_transport_bw

  (tlm::tlm_generic_payload &payload,
   tlm::tlm_phase &phase,
   sc_core::sc_time &delta
  )
  { return tlm::TLM_ACCEPTED; }

};
```

Problem 2) (continued)

CPU.cpp

```
#include "CPU.h"

using namespace std;

CPU::CPU( sc_core::sc_module_name name) : sc_module( name )
 {
    master(*this);
    SC_THREAD(main);
 }

void CPU::main( )
{
  sc_core::sc_time delay(0,sc_core::SC_NS);

  tlm::tlm_generic_payload  *transaction_ptr;
  unsigned char             *data_buffer_ptr;

  transaction_ptr = new tlm::tlm_generic_payload();
  data_buffer_ptr = new unsigned char[1];
  transaction_ptr->set_data_ptr ( data_buffer_ptr );

  sc_dt::uint64 mem_address;
  transaction_ptr->set_command          ( tlm::TLM_READ_COMMAND );
  transaction_ptr->set_data_length      ( 1                     );
  transaction_ptr->set_streaming_width  ( 1                     );

  while (true) {
    mem_address = 0;
    transaction_ptr->set_address          ( mem_address                   );
    transaction_ptr->set_response_status  ( tlm::TLM_INCOMPLETE_RESPONSE );
    master->b_transport(*transaction_ptr, delay);
       cout << sc_core::sc_time_stamp().to_string() << " CPU read "
            << (int)(*data_buffer_ptr) << " from address 0\n";
       wait(20, sc_core::SC_NS);
    if (*data_buffer_ptr) {
      mem_address = 1;
      transaction_ptr->set_address          ( mem_address                   );
      transaction_ptr->set_response_status  ( tlm::TLM_INCOMPLETE_RESPONSE );
      master->b_transport(*transaction_ptr, delay);
         cout << sc_core::sc_time_stamp().to_string() << " CPU read "
           << (*data_buffer_ptr) << " from address 1\n";
         wait(20, sc_core::SC_NS);
       }
  }
}
```

Problem 2) (continued)

UART.h

```
#pragma once

#include <tlm.h>
#include "tlm_utils/simple_target_socket.h"
#include <list>

class UART: public sc_core::sc_module
//, virtual public tlm::tlm_fw_transport_if<>   /// inherit from TLM "forward
interface"

{
public:

  tlm_utils::simple_target_socket<UART>  slave;
  std::list<char> buffer;
  UART(sc_core::sc_module_name module_name);
  void main ();

private:

  void custom_b_transport
  ( tlm::tlm_generic_payload  &payload
  , sc_core::sc_time          &delay_time
  );

  public:
};
```

Problem 2) (continued)

UART.cpp

```
#include "UART.h"
using namespace  std;

SC_HAS_PROCESS(UART);
UART::UART( sc_core::sc_module_name module_name)
    : sc_module (module_name)
{
  slave.register_b_transport(this, &UART::custom_b_transport);
  SC_THREAD(main);
}

void UART::custom_b_transport
( tlm::tlm_generic_payload  &payload
, sc_core::sc_time          &delay_time
)
{
    sc_dt::uint64 addr = payload.get_address();
    unsigned char *data = payload.get_data_ptr();

    if (addr == 0) {
         if (buffer.size()>0) data[0]=(char)1;
         else data[0]=(char)0;
       }
       else {
      if (buffer.size()>0) {
          data[0]=buffer.front();
          buffer.pop_front();
        }
        else data[0]=(char)0;
       }
    payload.set_response_status(tlm::TLM_OK_RESPONSE);
       return;
}

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

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

Problem 2) (continued)

top.h

```
#pragma once

#include <tlm.h>
#include "CPU.h"
#include "UART.h"

class top : public sc_core::sc_module
{
  private:
  CPU   cpu_inst;
  UART  uart_inst;

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

hw4p2.cpp

```
#include "top.h"
#include <tlm.h>

int sc_main(int argc, char *argv[])
{
  top top_inst("top0");
  sc_core::sc_start(1000,sc_core::SC_NS);
  return 0;
}
```

The output of the simulation looks identical to the output of Homework #2, problem 2.