ECE 720 – ESL & Physical Design

Lecture 9: Simple LT 2x2 Example System

W. Rhett Davis
NC State University

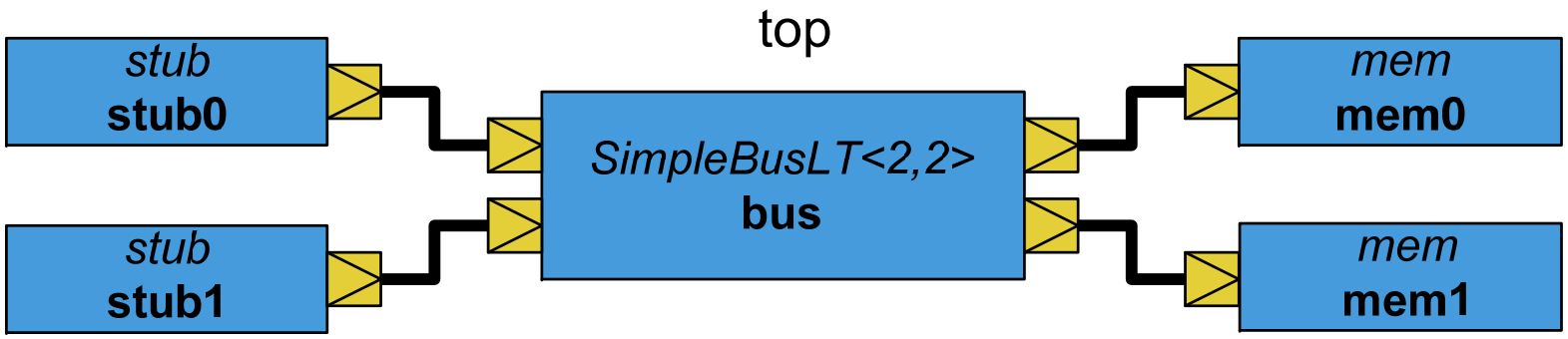
Announcements

- Project 1 Due Today
- Homework 3 Due Tuesday
- Homework 4 Due in 12 Days
- See SystemC Loosely-Timed Transaction-Level Modeling Example (sc_tlm_tut.tar.gz) for today's examples

Today's Lecture

- Simple LT 2x2 Example System
  - Structure
  - Behavior

Simple 2x2 Simulation



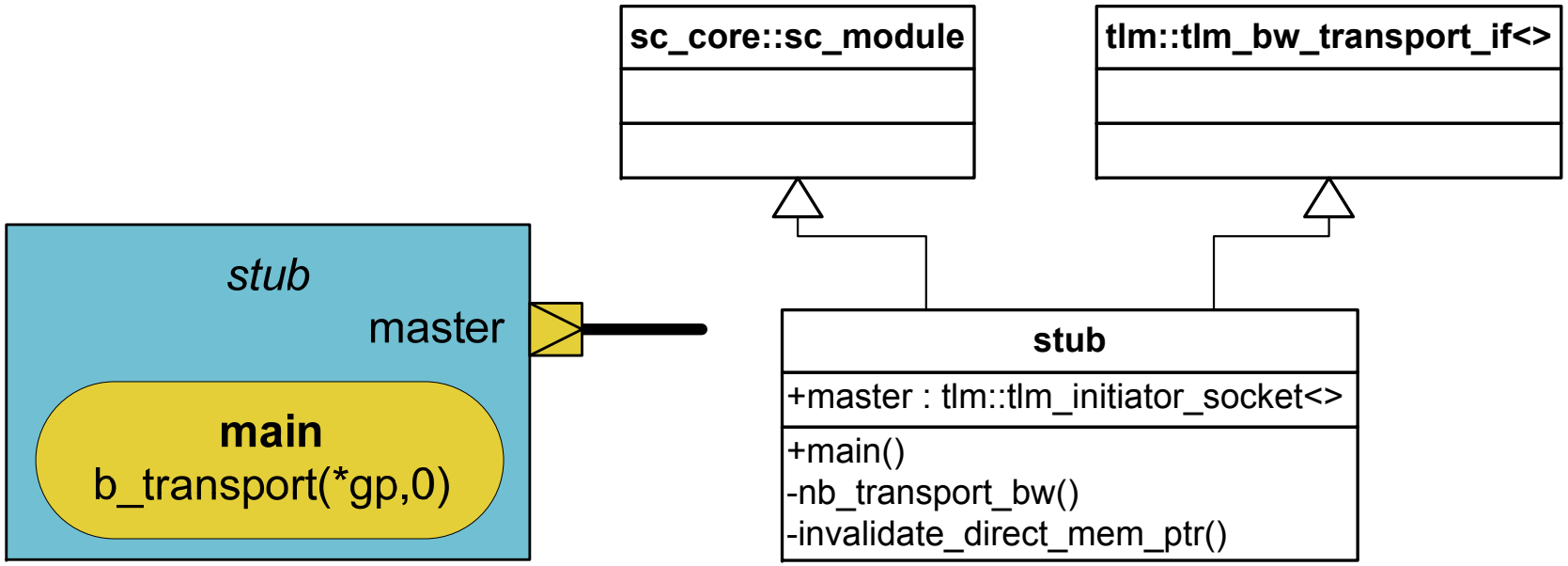
- Each initiator performs the following:
  - writes two values to one target
  - reads the same values from that target
  - repeats for the other target
- 16 transactions total
- Base addresses
  - mem0: 0x00000000
  - mem1: 0x10000000

Top Module

```
class top : public sc_core::sc_module {
  public:
  top (sc_core::sc_module_name name);
  private:
  SimpleBusLT<2, 2> bus;
  mem mem0;
  mem mem1;
  stub stub0;
  stub stub1;
};
```

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

Stub



- Bus master that generates transactions for debugging
- One thread process that initiates transactions from a file
- Two transport methods defined (as required by interface) but not implemented

Stub Module (1/4)

```
class stub : public sc_core::sc_module
           , virtual public tlm::tlm_bw_transport_if<> {
public:
  SC_HAS_PROCESS(stub);
  stub(sc_core::sc_module_name name, const char *xactfile);
  tlm::tlm_initiator_socket<> master;  const char *filename;
  void main(void);

 /// Not Implemented for this example but required by socket
    void invalidate_direct_mem_ptr( ...
    tlm::tlm_sync_enum nb_transport_bw( ...
};
```

```
stub::stub (sc_core::sc_module_name name, const char *xactfile)
          : sc_module(name), filename(xactfile)  {
    master(*this);
    SC_THREAD(main);
}
```

Stub Module (2/4)

```
void stub::main(void){
  sc_core::sc_time delay=sc_core::SC_ZERO_TIME;
  tlm::tlm_generic_payload  gp;
  gp.set_streaming_width  ( 4 );
  unsigned long length;
  sc_dt::uint64 addr;

  // Variables for reading transaction file
  ifstream f(filename,ios::in);
  double time_val;  string time_unit;
  sc_core::sc_time start_time;  string cmd;

  // Skip the first line, assume it is a comment
  if (f.good())
    getline(f,cmd);
```

Stub Module (3/4)

```
while (f.good()) {
   f >> time_val >> time_unit >> cmd >> hex >> length >> addr;
   if (!f.good()) continue;
   if (cmd=="WRITE")
     gp.set_command(tlm::TLM_WRITE_COMMAND);
   else if (cmd=="READ")
     gp.set_command(tlm::TLM_READ_COMMAND);
   else {
     cout << sc_core::sc_time_stamp()<<" "<< sc_object::name()
          << " Unrecognized command: " << cmd << endl;
     continue;
   }
   gp.set_address( addr );
   gp.set_response_status( tlm::TLM_INCOMPLETE_RESPONSE );
   gp.set_data_length( length );

   // Parse the start-time from the transaction file
   if (time_unit=="fs")
     start_time=sc_core::sc_time(time_val,sc_core::SC_FS);
   ...
```

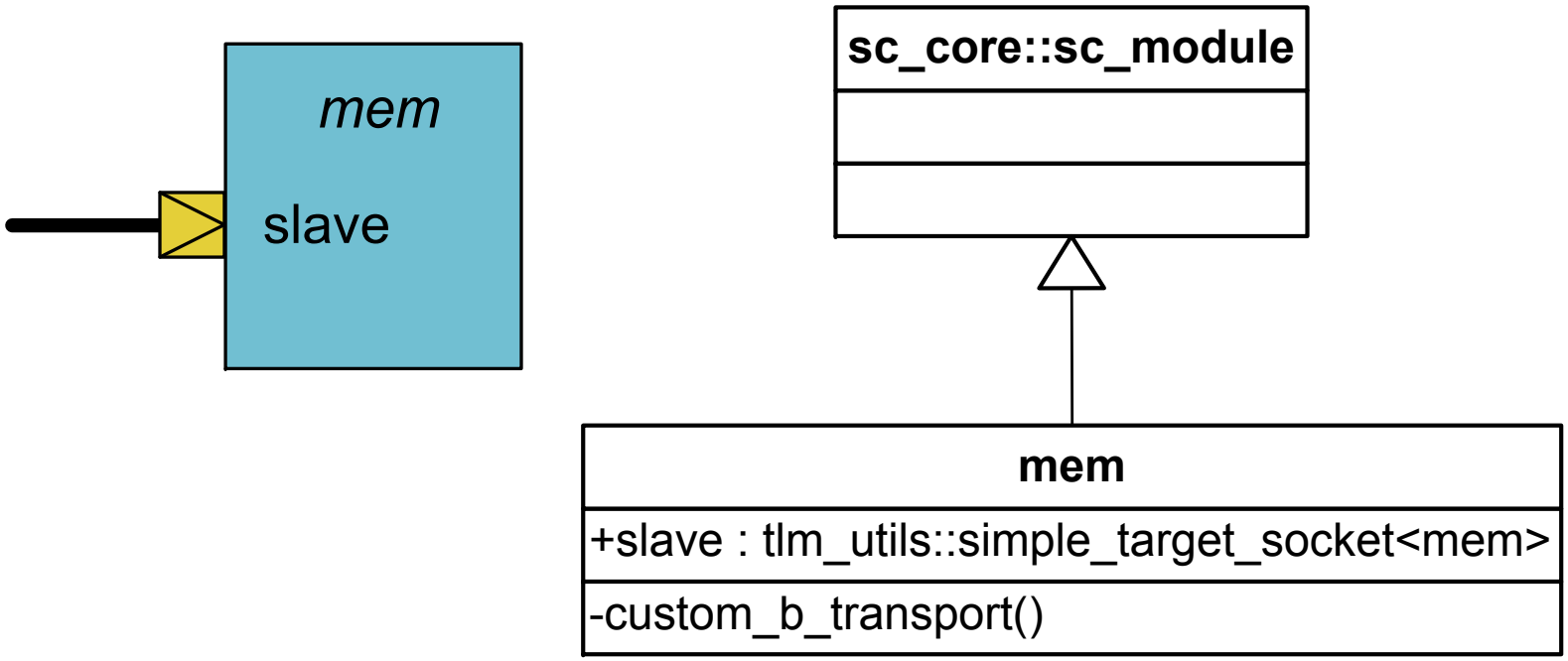
Stub Module (4/4)

```
    // Wait until the transaction start-time is reached
    if (sc_core::sc_time_stamp() < start_time)
      wait( start_time-sc_core::sc_time_stamp() );

    // Perform the transaction
    cout << sc_core::sc_time_stamp() <<" "<< sc_object::name()
         <<" "<< cmd <<" "<< hex <<length<<" "<< addr << endl;
    master->b_transport(gp, delay);
    if (gp.get_response_status() != tlm::TLM_OK_RESPONSE)
      cout << sc_core::sc_time_stamp() <<" "<<sc_object::name()
           << " ERROR Response Status “
           << gp.get_response_status() << endl;
  } //while

  cout << sc_core::sc_time_stamp() << " " << sc_object::name()
       << " Completed" << endl;
} // end main
```

Memory



- Target has no processes, only one (blocking) transport method, which calls ***wait*** to model memory delay
- Convenience socket used to avoid need to define non-blocking transport methods

Memory Module (1/3)

```
class mem: public sc_core::sc_module {
  public:
  mem( sc_core::sc_module_name module_name,
       sc_dt::uint64  memory_size );  // memory size (bytes)
  tlm_utils::simple_target_socket<mem>  slave;
  sc_dt::uint64 m_memory_size;
  void custom_b_transport
  ( tlm::tlm_generic_payload &gp, sc_core::sc_time &delay );
};
```

```
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) {
    slave.register_b_transport(this, &mem::custom_b_transport);
}
```

Memory Module (2/3)

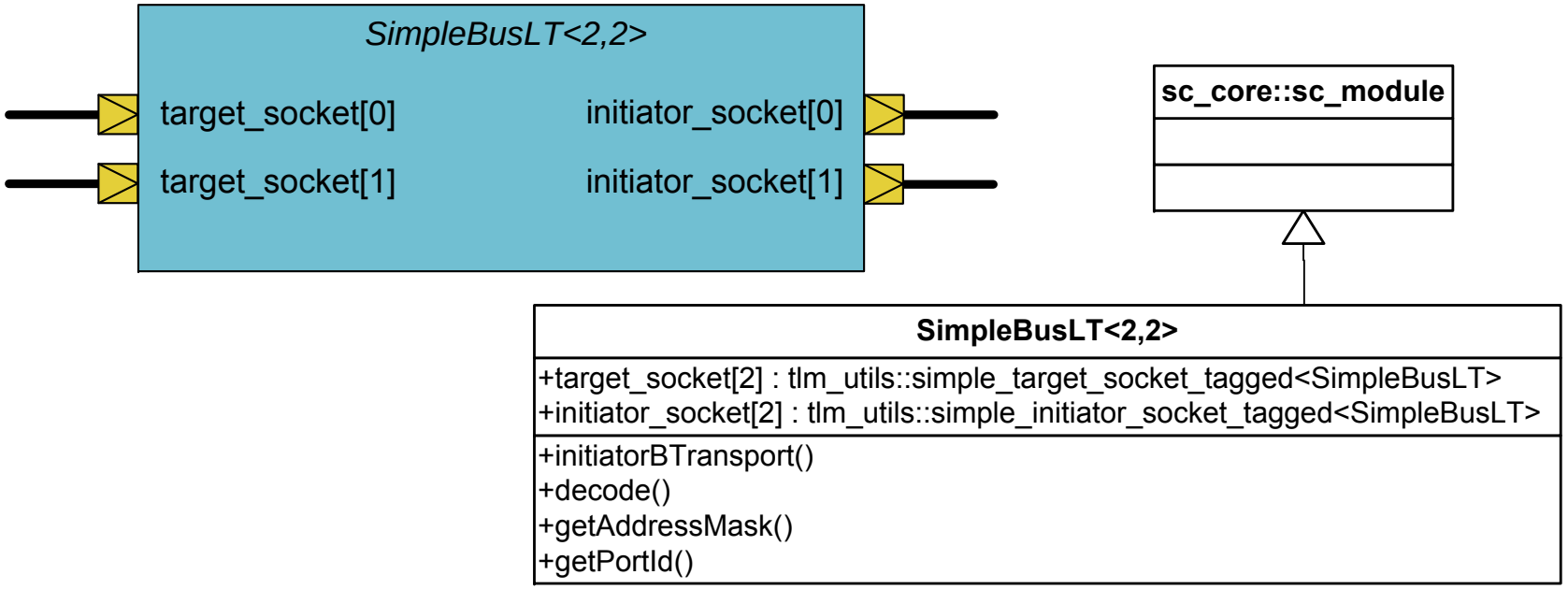
```
void mem::custom_b_transport
 ( tlm::tlm_generic_payload &gp, sc_core::sc_time &delay ) {
  sc_dt::uint64    address   = gp.get_address();
  tlm::tlm_command command   = gp.get_command();
  sc_core::sc_time mem_delay(10,sc_core::SC_NS);

  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 );
        break;
```

Memory Module (3/3)

```
case tlm::TLM_READ_COMMAND: {
     cout << " READ 0x" << hex << address << endl;
     gp.set_response_status( tlm::TLM_OK_RESPONSE );
     break;
   }
   default: {
     cout << " ERROR Command " << command
          << " not recognized" << endl;
     gp.set_response_status( tlm::TLM_COMMAND_ERROR_RESPONSE );
   }
  }
 }
 else {
    cout << " ERROR Address 0x" << hex << address
         << " out of range" << endl;
    gp.set_response_status( tlm::TLM_ADDRESS_ERROR_RESPONSE );
 }
 return;
}
```

Bus



- Convenience sockets needed because there are multiple ports of each type
- Bus also has no processes, only one (blocking) transport method
- decode, getPortId, and getAddressMask methods implement the mapping of targets to address space in multiples of 0x10000000 (28-bit address spaces)

Bus Module (1/2)

```
template <int NR_OF_INITIATORS, int NR_OF_TARGETS>
class SimpleBusLT : public sc_core::sc_module
{
...
 target_socket_type target_socket[NR_OF_INITIATORS];
 initiator_socket_type initiator_socket[NR_OF_TARGETS];
 SC_HAS_PROCESS(SimpleBusLT);
  SimpleBusAT(sc_core::sc_module_name name) :
    sc_core::sc_module(name)  {
     for (unsigned int i = 0; i < NR_OF_INITIATORS; ++i) {
       target_socket[i].register_b_transport_fw(this,
                    &SimpleBusLT::initiatorBTransport, i);
       target_socket[i].register_transport_dbg(this,
                    &SimpleBusLT::transportDebug, i);
       target_socket[i].register_get_direct_mem_ptr(this,
                    &SimpleBusLT::getDMIPointer, i);
```

Bus Module (2/2)

```
    for (unsigned int i = 0; i < NR_OF_TARGETS; ++i) {
       initiator_socket[i].
                  register_invalidate_direct_mem_ptr(this,
                  &SimpleBusLT::invalidateDMIPointers, i);
    }
}

void initiatorBTransport(int SocketId,
                         transaction_type& trans,
                         sc_core::sc_time& t)
```

Today's Lecture

- Simple LT 2x2 Example System
  - » Structure
  - » Behavior

Transaction Files

xact0.txt

```
# time command bytes address
  0 ns WRITE   0x4   0x10000200
 40 ns WRITE   0x4   0x10000204
 80 ns READ    0x4   0x10000200
140 ns READ    0x4   0x10000204
200 ns WRITE   0x4   0x00000200
240 ns WRITE   0x4   0x00000204
280 ns READ    0x4   0x00000200
340 ns READ    0x4   0x00000204
```

xact1.txt

```
# time command bytes address
  0 ns WRITE   0x4   0x00000100
 40 ns WRITE   0x4   0x00000104
 80 ns READ    0x4   0x00000100
140 ns READ    0x4   0x00000104
200 ns WRITE   0x4   0x10000100
240 ns WRITE   0x4   0x10000104
280 ns READ    0x4   0x10000100
340 ns READ    0x4   0x10000104
```

- Each stub performs two writes and two reads to each memory

Output

```
0 s top.stub0 WRITE 4 10000200
0 s top.stub1 WRITE 4 100
10 ns top.mem1 WRITE 0x200
10 ns top.mem0 WRITE 0x100
40 ns top.stub0 WRITE 4 10000204
40 ns top.stub1 WRITE 4 104
50 ns top.mem1 WRITE 0x204
50 ns top.mem0 WRITE 0x104
80 ns top.stub0 READ 4 10000200
80 ns top.stub1 READ 4 100
90 ns top.mem1 READ 0x200
90 ns top.mem0 READ 0x100
140 ns top.stub0 READ 4 10000204
140 ns top.stub1 READ 4 104
...
350 ns top.mem0 READ 0x204
350 ns top.stub0 Completed
350 ns top.mem1 READ 0x104
350 ns top.stub1 Completed
```

Questions

- What would happen if one stub wrote to location 0x2000?

Questions

- What value have we added by organizing our code this way?

Questions

- What are we not modeling in this simulation that we might want to know?

Comparison of LT & AT Styles

- A system was created that is identical to this system, but uses non-blocking transport methods

	LT	AT
lines of code	1,700	2,400
processes (defined/instantiated)	1/2	5/8
synchronization objects (defined/instantiated)	0/0	8/12
lines of output	180	570