

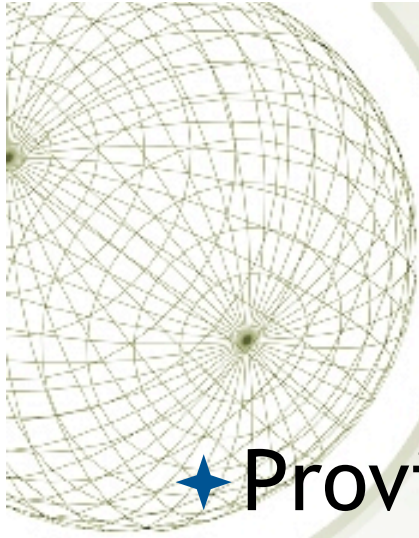
Transaction Level Modeling

Brad Matthews
ECE 652



What is Transaction Level Modeling?

- ★ Involves the abstraction of content to a level above RTL
- ★ Move from pin-level communication to a set of function calls
 - ★ `burst_write(int addr, int data, int *wdata)`
 - ★ `burst_read(int addr, int data, int *rdata)`
 - ★ `write(int addr, int data)`
 - ★ `read(int addr)`



Why is Transaction Level Modeling important?

- ★ Provides an executable specification for both hardware and software engineers
- ★ Simulation speedup (5x-1000x)
- ★ System level design exploration and verification



The TLM Dream

- ★ Transaction Level Model will be completed before a single line of RTL or software is written
- ★ Allow for hardware and software to be developed at the same time
- ★ Improve time to market and eliminate “specification errors”



Languages for TLM

★ C

- ★ Pros: Well-known, fast, can use Verilog PLI or Synopsys' DirectC to integrate model with RTL
- ★ Cons: OOP is not easy, No industry supported application program interface (API) for TLM

★ C++

- ★ Pros: Fast, OOP language, templates, inheritance, polymorphism
- ★ Cons: No industry supported API for TLM

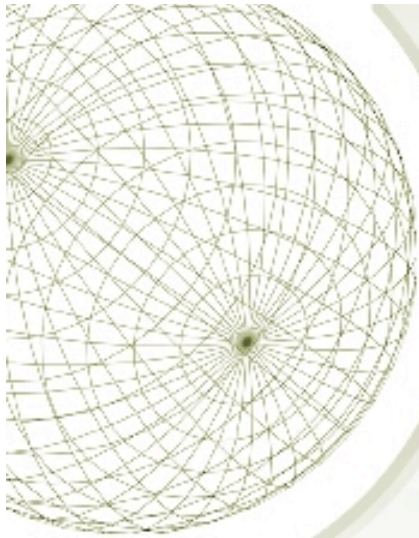
★ SystemC

- ★ Pros: C++ with classes that support System Level Modeling, industry support, can co-simulate with RTL
- ★ Cons: Knowledge-base not as deep as C++ or C.



TLM Layers

<p><u>User Layer</u></p> <p>Protocol-specific “convenience” API Targeted for embedded SW engineer Typically defined and supplied by IP vendors</p>	<pre>amba_bus->burst_read(buf, adr, n);</pre>
<p><u>Protocol Layer</u></p> <p>Protocol-specific code Adapts between user layer and transport layer Typically defined and supplied by IP vendors</p>	<pre>req.addr = adr; req.num = n; rsp = transport(req); return rsp.buf;</pre>
<p><u>Transport Layer</u></p> <p>Uses generic data transport APIs and models Facilitates interoperability of models Key focus of TLM standard May use generic fifos, arbiters, routers, xbars, pipelines, etc.</p>	<pre>sc_port<tlm_transport_if<REQ, RSP> > p;</pre>



TLM Layers (cont...)

Message Layer (L-3)

Transaction Layer (L-2)

Transfer Layer (L-1)

RTL Layer (L-0)

Abstraction removes:

Resource sharing, time

Clock, protocols

Wires, registers

Gates, gate/wire delays



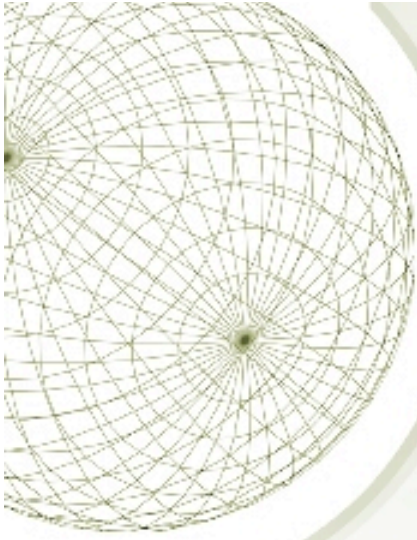
C++ Inheritance

- ★ Provides a mechanism for creating a new class using a base class. The new class is generally termed a derived class.
- ★ A derived class can inherit use the functions/data of the base class, extend the functionality, or override the functions provided by the base class
- ★ Inheritance defines an “is a” relationship
 - ★ For example, a dog “is a” animal




C++ Templates

- ★ Allows for the definition of functions once for multiple data types
- ★ The “C” way
 - ★ `float calculate_fir(int NumberOfTaps);`
 - ★ `int calculate_fir(int NumberOfTaps);`
- ★ The “C++” way
 - ★ `fir<float, 256> fir1;`
 - ★ `fir<int, 512> fir2;`
 - ★ `fir1.calculate_fir(); fir2.calculate_fir();`



C++ Template Example

```
template<type T, int N=256>
class FIR : public sc_module
{
    public:
        sc_in<T>    data_in;
        sc_out<T>  data_out;
        .....
        T    calculate_fir()
    private:
        T tap[N];
}
```



SystemC Terminology

★ Module

- ★ Container Class that can consists of processes, ports, channels, and other modules.

★ Processes

- ★ Code block that describes and implements the module functionality

★ Port

- ★ Provides mechanism to connect a module to a channel via an interface



SystemC Terminology (cont...)

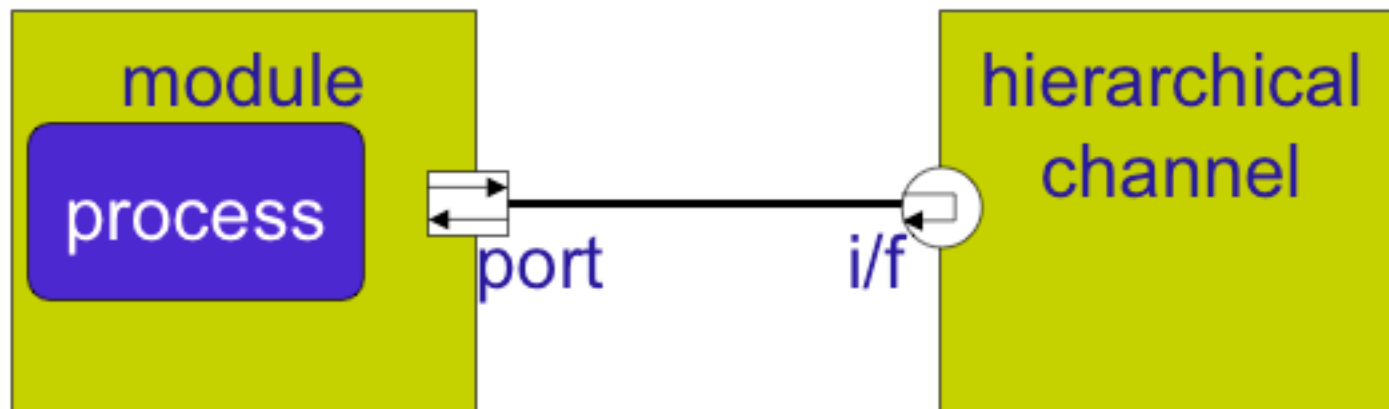
★ Interface

- ★ Describes what methods are available, but provides no method implementations or data fields
- ★ Describes what is supported such as read() or write()

★ Channels

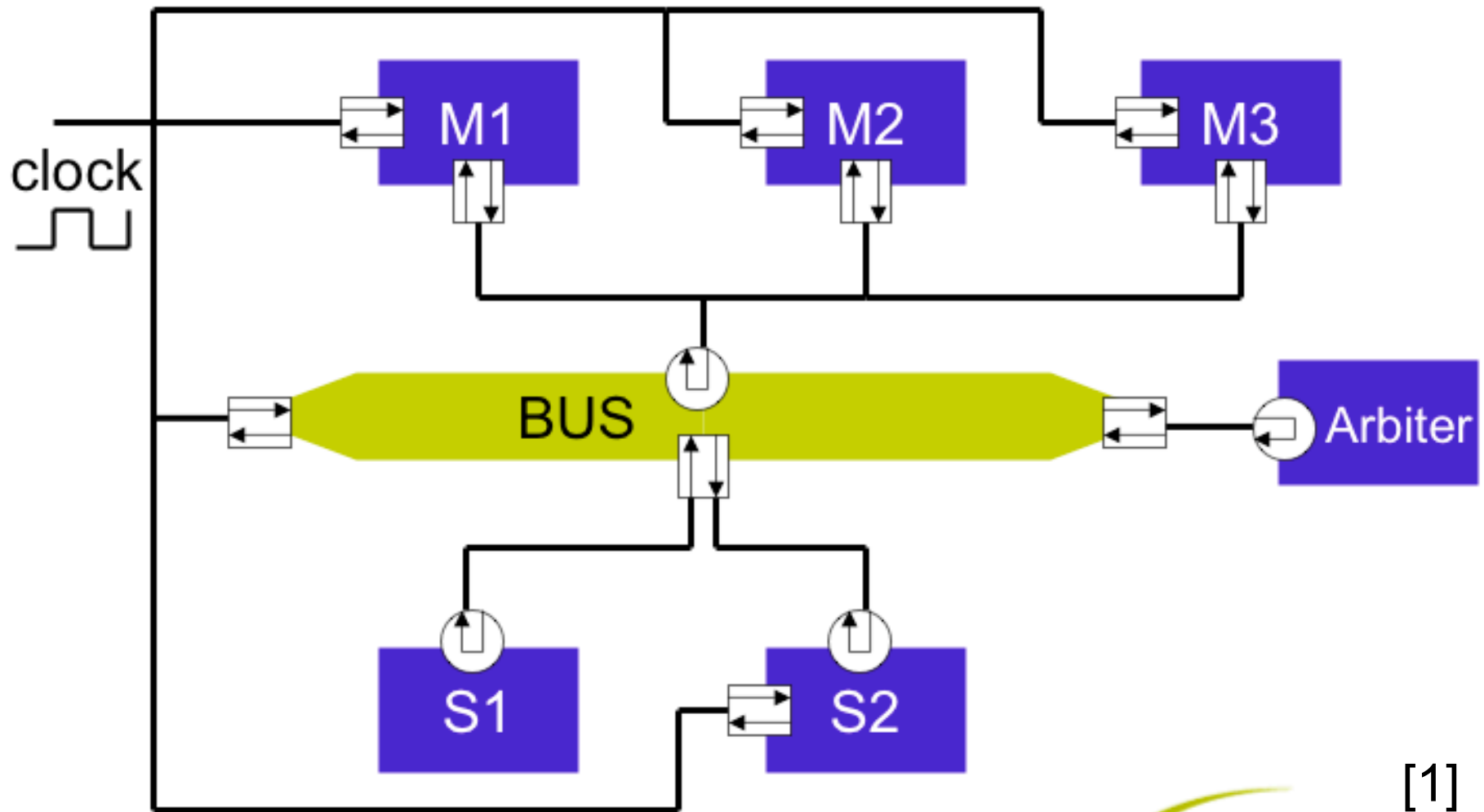
- ★ Implements the methods made available by the the interface
- ★ Performs the actual transfer

SystemC Terminology (cont...)



[1]

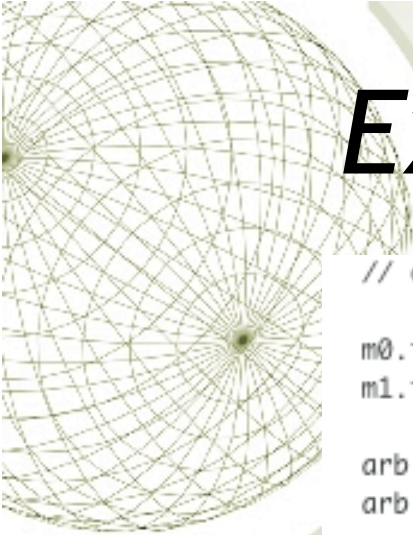
SystemC Terminology (cont...)





Example TLM Code

```
typedef tlm_req_rsp_channel< basic_request< ADDRESS_TYPE , DATA_TYPE > ,  
                             basic_response< DATA_TYPE > > arb_channel_type;  
  
int sc_main( int argc , char **argv )  
{  
  
    // masters  
  
    switch_master m0("master_0" , 0 , 57 );  
    switch_master m1("master_1" , 1 , 1000 );  
  
    // channels  
    arb_channel_type c0 , c1;  
  
    // arbiter  
  
    simple_arb< basic_request< ADDRESS_TYPE , DATA_TYPE > ,  
               basic_response< DATA_TYPE > > arb("arb");  
  
    // router  
  
    router< ADDRESS_TYPE ,  
           basic_request< ADDRESS_TYPE , DATA_TYPE > ,  
           basic_response< DATA_TYPE >  
    > router_module("router" , "master.iport.map" );  
  
    // slaves  
  
    mem_slave s0("slave_0");  
    mem_slave s1("slave_1");  
}
```



Example TLM Code (cont...)

```
// connectivity pattern is master -> channel -> arbiter -> router -> slave

m0.initiator_port( c0 ); // connect m0 to its channel
m1.initiator_port( c1 ); // connect m1 to its channel

arb.master_port[0]( c0 ); // connect arbiter to channel 0
arb.master_port[1]( c1 ); // connect arbiter to channel 1

arb.slave_port( router_module.target_port ); // connect arbiter to router

router_module.r_port( s0.target_port ); // connect router to slave 0
router_module.r_port( s1.target_port ); // connect router to slave 1

// set arbitration priorities ( can be done from file if needed )

arb.add_interface( &arb.master_port[0] , 3 );
arb.add_interface( &arb.master_port[1] , 2 );

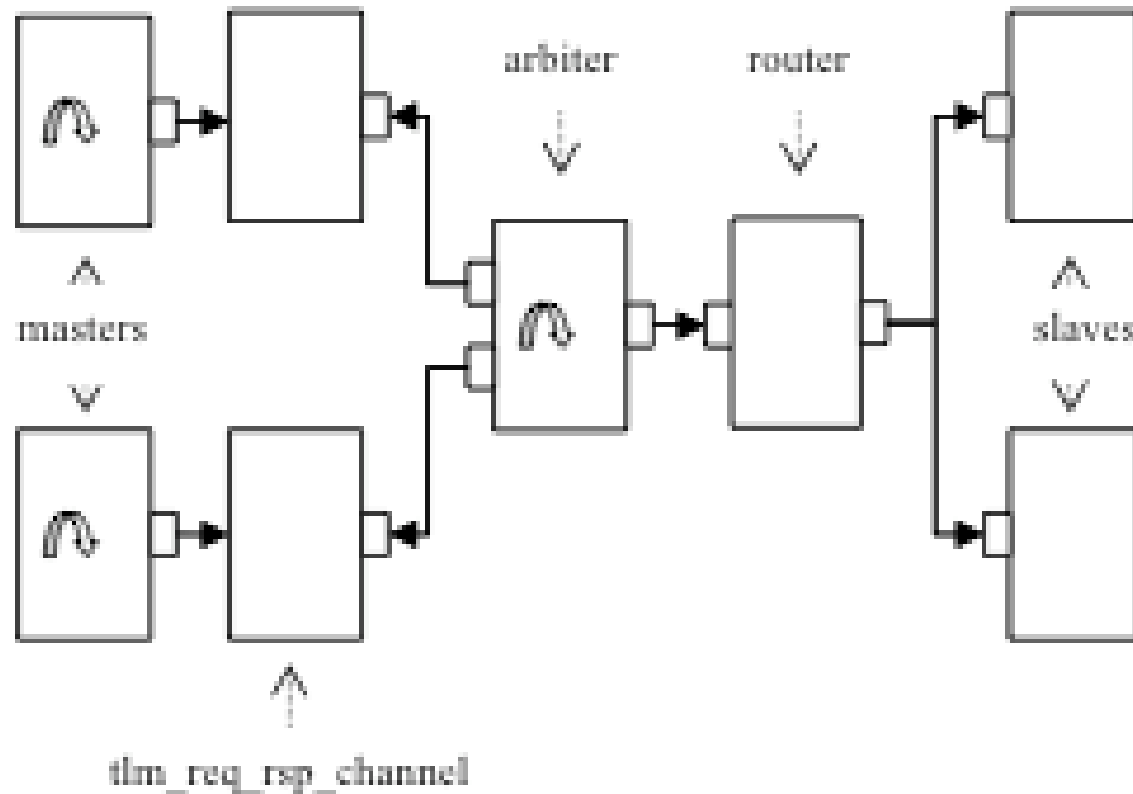
sc_start( 100 , SC_NS );

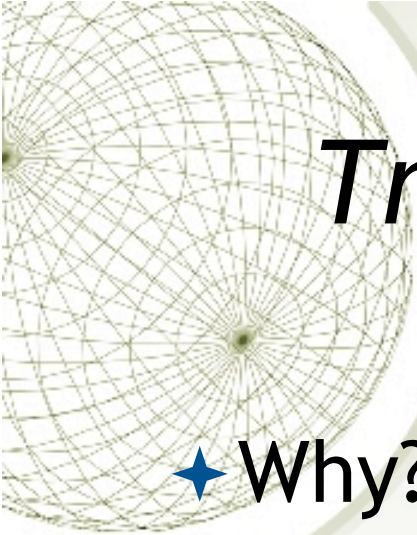
cout << "Finished" << endl;

return 0;

}
```


Resulting Structure



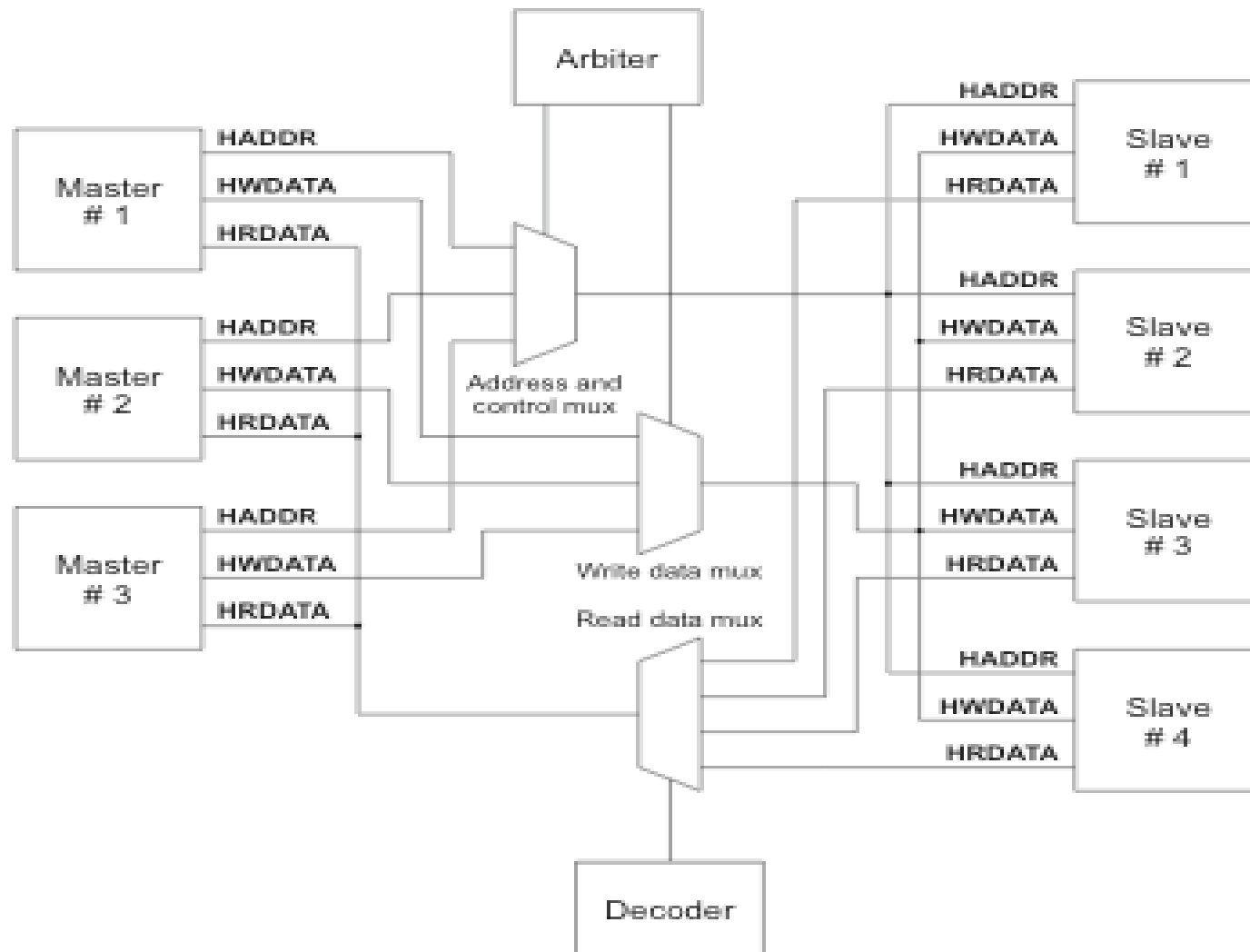


Transaction Level Modeling with AMBA

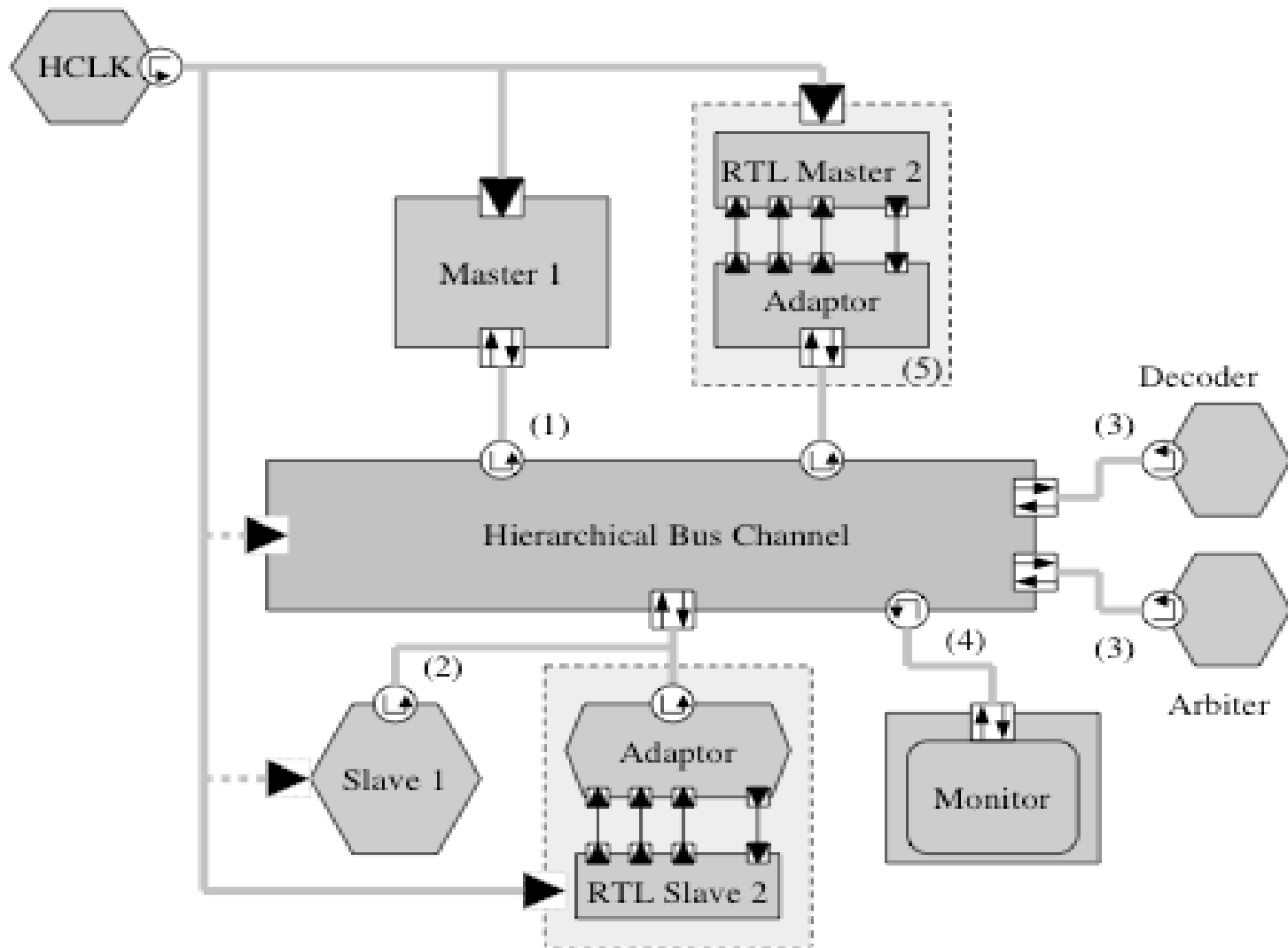
★ Why?

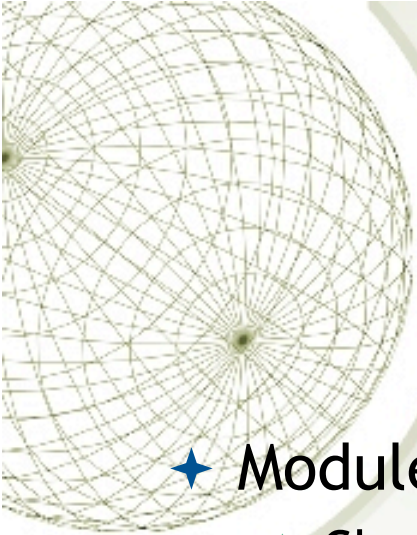
- ★ Mechanism that allows architect to:
 - ★ Explore arbitration algorithms
 - ★ Ensure enough system bandwidth is available
 - ★ Provide executable specification to designers
- ★ Extremely useful for platform-based development
 - ★ RapidChip (LSI Logic)
 - ★ SoC Mosaic (Toshiba)

AMBA Bus Structure



SystemC AMBA System





AMBA System Modeling Components

★ Module(s)

- ★ Clock Generator (HCLK)
- ★ Master(s)
- ★ Slave(s)
- ★ Bus*
- ★ Arbiter
- ★ Decoder
- ★ Monitor**

★ Interface(s)

- ★ Master Interface
- ★ Slave Interface
- ★ Arbiter Interface
- ★ Decoder Interface

★ Channel(s)

- ★ AHB Channel
- ★ Bus*



AMBA Clock Generator

- ★ Simple declaration of `sc_clock`:

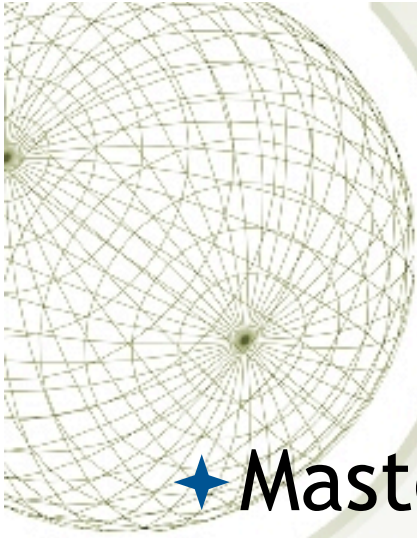
```
sc_clock clk( "clock", sc_time( 10, SC_NS ) );
```

- ★ System with multiple clock domains would be defined by multiple declarations of `sc_clock`:

```
sc_clock clk1( "clock1", sc_time( 20, SC_NS ) );
```

```
sc_clock clk2( "clock2", sc_time( 10, SC_NS ) );
```

```
sc_clock clk3( "clock3", sc_time( 5, SC_NS ) );
```



AMBA Master Module

- ★ Master has the following properties:
 - ★ Port that connects to an interface implemented by a channel (AHB Channel or Bus)
 - ★ Read/Write transaction requests are implemented in the channel but made from the master.



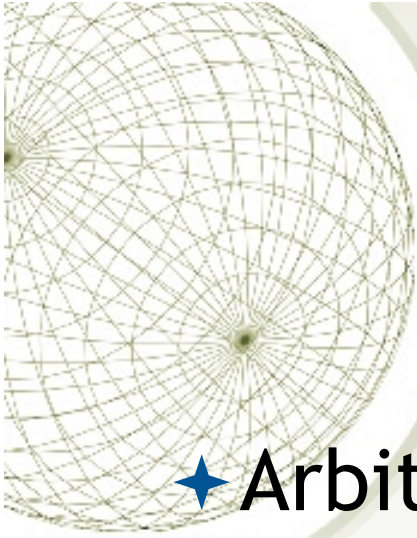
AMBA Slave Module

- ★ Slave has the following properties
 - ★ Port connects to an interface with functions again implemented by the channel
 - ★ Transactions are to be initiated using the channel
 - ★ Common to represent a slave address space using an array to read()/write() from/to a given address



AMBA Bus Module / Channel

- ★ Masters and Slaves connect to the Bus using the AHB Channel (module*)
- ★ Arbiter and Decoder connects to the Bus using specialized interfaces
- ★ Can be represented as a simple mux-bus structure or a complex pipeline structure with defined latency



AMBA Arbiter Module

- ★ Arbiter processes requests and, using a desired algorithm, provides masters with a grant signal for bus access
- ★ Key is the arbitration algorithm
 - ★ Simple Priority-Based Arbitration
 - ★ Round-Robin Arbitration
 - ★ Weighted Round-Robin Arbitration



AMBA Decoder Module

- ★ Reads the address line to determine which slave should be granted access to the Read Mux.
- ★ Contains address map for design in order to decode correctly



AMBA AHB Bus Channel

- ★ Implements the functions of the interfaces detailed in the Master, Slave, and Bus modules
- ★ Routines include:
 - ★ Read()
 - ★ Write()
 - ★ BurstRead()
 - ★ BurstWrite()



sc_main() code structure

```
int sc_main(int argc, char **argv )
{
    ahb_bus_tlm* bus1;
    ahb_master* master1, master2;
    slave* slavel, slave2;
    ahb_addr_monitor* monitor1;

    sc_set_time_resolution(1, SC_NS);
    sc_clock hclk("hclk");

    bus1      = new ahb_bus_tlm("bus1", 32 /* data bus width */);
    master1   = new ahb_master("master1", 1 /* priority */);
    master2   = new ahb_master("master2", 2 /* priority */);
    slavel    = new slave("slavel", 0x0000 /* base address */);
    slave2    = new slave("slave2", 0x2000 /* base address */);

    // Monitor instantiation, comment out to disable
    monitor1 = new ahb_addr_monitor("address_monitor1", true);

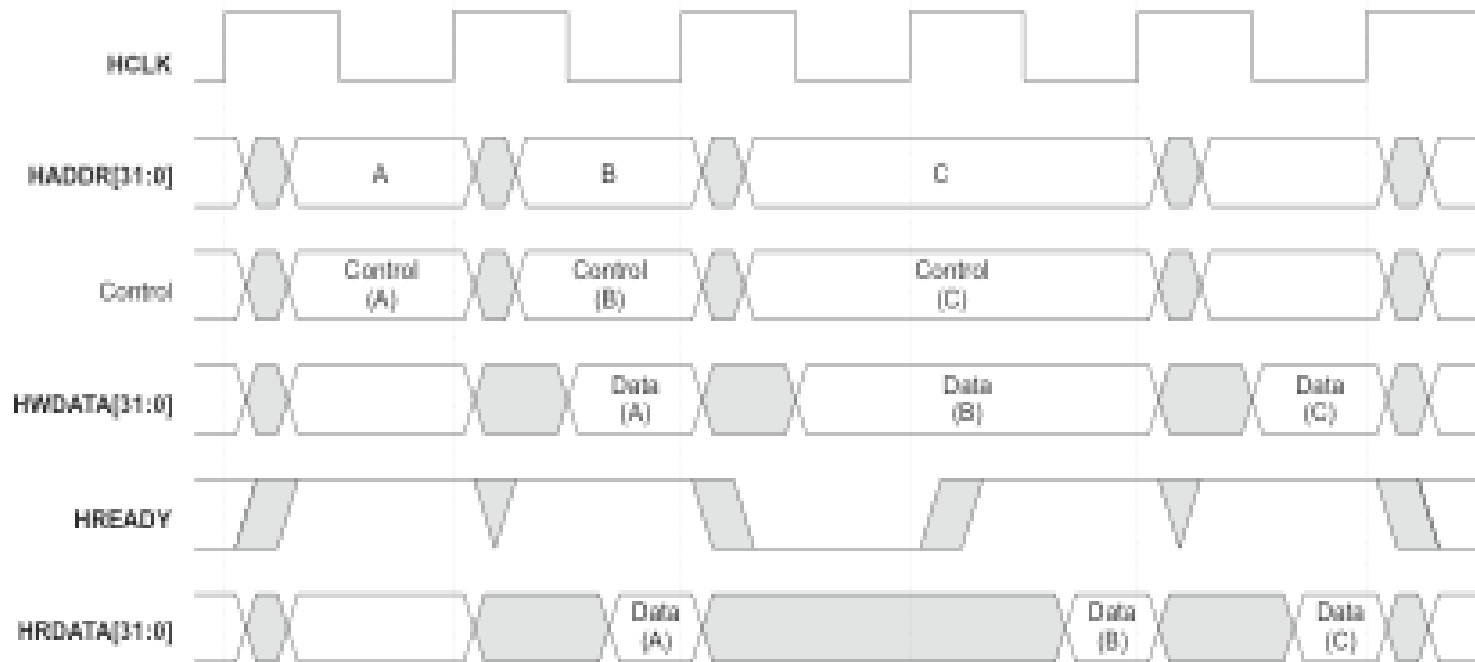
    bus1->hclk(hclk);
    master1->hclk(hclk);
    master2->hclk(hclk);

    bus1->slave_port(*slavel);
    bus1->slave_port(*slave2);

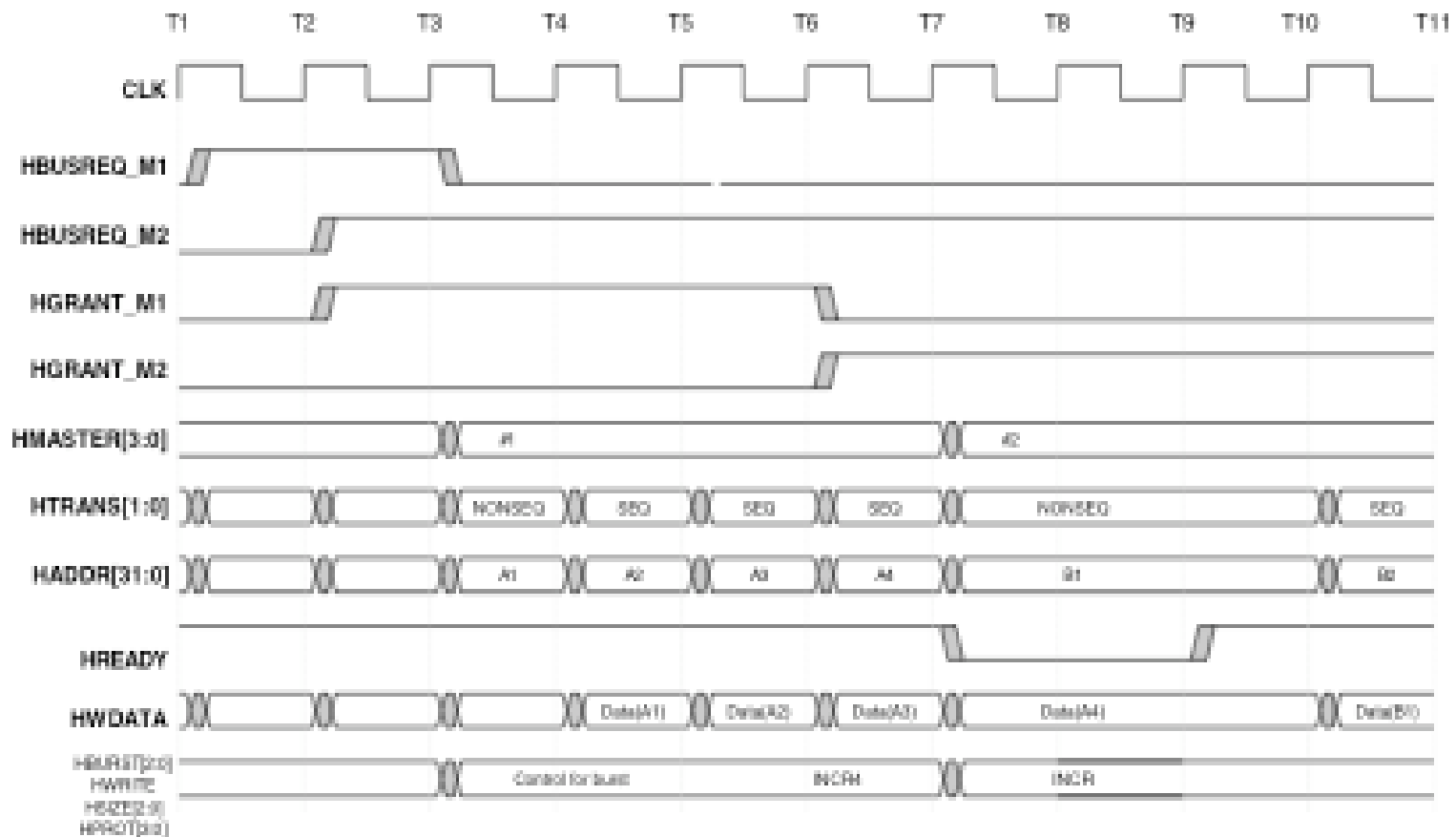
    master1->mport(*bus1);
    master2->mport(*bus1);

    // Monitor port to bus channel connection, comment out to disable
    monitor1->monitor_port(*bus1);
}
```

AMBA Signaling



AMBA Signaling (cont...)



AMBA TLM Data Flow

Time	Master 1	Master 2	bus
T1	request() has_grant() [FALSE] response()[OKAY, READY] *set_protection() *init_transaction() [A1,INCR4] +master_complete()	idle() +master_complete()	
			response() [OKAY, READY] arbitrate()
T2	has_grant() [FALSE] response()[OKAY, READY] *master_complete()	request() has_grant() [FALSE] response()[OKAY, READY] **set_protection() **init_transaction() [B1,INCR] *master_complete()	
			response() [OKAY, READY] arbitrate() [grant M1]

AMBA TLM Data Flow (cont...)

T3	has_grant() [TRUE] response()[OKAY, READY] end_request() ^master_complete()	has_grant() [FALSE] response()[OKAY, READY] ^master_complete()	
			response() [OKAY READY] arbitrate()[grant M1] get_slave() write()[address A1] control_info() [NONSEQ, INCR4]
T4	has_grant() [TRUE] response()[OKAY, READY] set_data() [A1] ^master_complete()	has_grant() [FALSE] response()[OKAY, READY] ^master_complete()	
			set_data() [A1] response() [A1-OKAY, READY] arbitrate()[grant M1] get_slave() write()[address A2] control_info()[SEQ]

AMBA TLM Data Flow (cont...)

T5	has_grant() [TRUE] response()[A1-OKAY, READY] set_data() [A2] *master_complete()	has_grant() [FALSE] response()[A1-OKAY, READY] *master_complete()	
			set_data() [A2] response() [A2-OKAY, READY] arbitrate()[grant M1] get_slave() write()[address A3] control_info()[SEQ]
T6	has_grant() [TRUE] response()[A2-OKAY, READY] set_data() [A3] *master complete()	has_grant() [FALSE] response()[A2-OKAY, READY] *master_complete()	
			set_data() [A3] response() [A3-OKAY, READY] arbitrate()[grant M2] get_slave() write()[address A4] control_info()[SEQ]

AMBA TLM Data Flow (cont...)

T7	response()[A3-OKAY, READY] set_data() [A4] idle() *master_complete()	has_grant() [TRUE] response()[A3-OKAY, READY] *master_complete()	
			set_data() [A4] response() [A4-OKAY, NOT READY] arbitrate()[grant M2]
T8	response()[A4-OKAY, NOT READY] *master_complete()	has_grant() [TRUE] response()[A4-OKAY, NOT READY] *master_complete()	
			response() [A4-OKAY, NOT READY] arbitrate()[grant M2]
T9	response()[A4-OKAY, NOT READY] *master_complete()	has_grant() [TRUE] response()[A4-OKAY, NOT READY] *master_complete()	
			response() [A4-OKAY, READY] arbitrate()[grant M2] get_slave() write()[address B1] control_info()[NONSEQ, INCR]



AMBA Master Code

```
class MasterExample
: public sc_module
{
public:

    sc_in_clk hclk;
    ahb_master_port m_port;

    void main_action();

    SC_CTOR(MasterExample)
    : m_port(name(), 1, 1, true)
    {
        SC_THREAD(main_action);
        sensitive << hclk.pos();
        dont_initialize();
    }
};
```



AMBA Master Code (cont...)

```
void MasterExample::main_action() {
    unsigned int data[4];
    bool ready_resp;
    ahb_hresp status_resp;

    // T1:
    ready_resp = m_port.response(status_resp);
    m_port.init_transaction(WR, data, A1, INCR, SIZE_32);
    wait();

    // T2:
    ready_resp = m_port.response(status_resp);
    data[0] = 0x20;
    m_port.set_data();
    m_port.busy();
    wait();

    // T3:
    ready_resp = m_port.response(status_resp);
    data[1] = 0xdeadbeef;
    m_port.set_data();
    wait();

    // T4:
    ready_resp = m_port.response(status_resp);
    data[1] = 0x24;
    wait();

    // T5:
    ready_resp = m_port.response(status_resp);
    wait();

    // T6:
    ready_resp = m_port.response(status_resp);
    data[2] = 0x28;
    m_port.set_data();
    m_port.init_transaction(RD, data+3, B1, SINGLE, SIZE_32);

    data[3] = 0x2B;
    m_port.direct_write(B1, &data[3], 4);
    data[3] = 0x0;
    wait();

    // T7:
    ready_resp = m_port.response(status_resp);
    m_port.set_data();
    m_port.idle();
    wait();

    // T8:
    ready_resp = m_port.response(status_resp);

    wait();
}
```



TLM Summary

- ★ The key word with Transaction Level Modeling is abstraction
- ★ Increased simulation speed at the cost of accuracy
- ★ Important to find the right level of abstraction to meet the desired performance/accuracy goals



References

- [1] <http://www.elet.polimi.it/upload/silvano/mioweb5/FilePDF/METODOLOGIE/Simple%20Bus%20Slides.pdf>
- [2] ARM, *Amba specification (rev 2.0)*, March 2005.
- [3] ARM Ltd. AMBA AHB Cycle Level Interface Specification, Document number AHBCLI.1.1.0, 2003.