Composition concepts in SystemC

Seminar Part-1
Presented by:

Shreyas Ramakrishna
shreyas.ramakrishna@vanderbilt.edu

VANDERBILT
UNIVERSITY

CS 6377-01 (2018S)
Topics in Embedded Software and System

February 27th, 2018

Seminar Part-1 Agenda

VANDERBILT School of Engineering

- Introduction to SystemC
    - Why SystemC
    - How it performs as compared to HDL
    - Basic building blocks
    - Programming structure
    - SystemC history
    - Advantages

- Introduction to System Level Modeling in SystemC 2.0
    - Features of SystemC 1.0 and SystemC 2.0
    - Communication and synchronization in SystemC 2.0
    - Available models of computation

- Introduction to Transaction Level Modeling(TLM)
    - Motivation and industry standard requirements of TLM
    - Key concepts of TLM
    - Simple Master/ Slave examples
    - Common system level design patterns

Introduction to SystemC: The Language for System-Level Modeling, Design and Verification An OSCI Initiative

Problem

VANDERBILT School of Engineering

Electronic chip companies faced some problems as the complexity and size increased:

•How to concurrently develop software and hardware?

•How to start writing software drivers before the (register transfer level)RTL design is finished?

•How to develop a reference model to be used in with Test bench.

•How to build and re-use IP models for use in a high level model? (otherwise for every new project have to build the models from scratch).

•If developing a completely new chip, how to analyze bus bandwidths, data flows and so on - especially with multiple processors competing for bus resource.

The Birth

The SystemC was born due to the necessities of the current industry requirements:

• Most of our electronic gadgets are demanding regular incorporation of new functionalities , with short time to design and first time design success.

•The greater complexity (multi million gate design) of the current systems are making the situation worst.

• Previously, C (or C++) was used to write the software part of the design and either VHDL or Verilog were used to design the hardware part.

•It was very difficult to setup a common test bench which is common for both, since they are entirely different languages.

•This led to the birth of SystemC, which promises to solve this problem by allowing simultaneous(co-) design of hardware and software.

Introduction to SystemC

VANDERBILT School of Engineering

•Library of C++ classes which provides an event driven simulation interface.

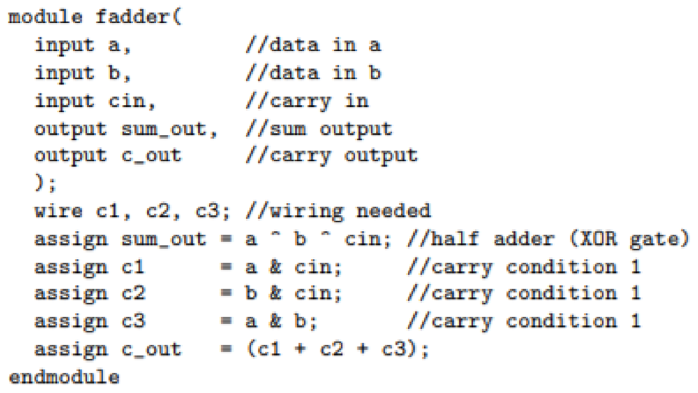
•Provides signals, events, and synchronization primitives to mimic the hardware description languages of VHDL and Verilog.

•What it offers:

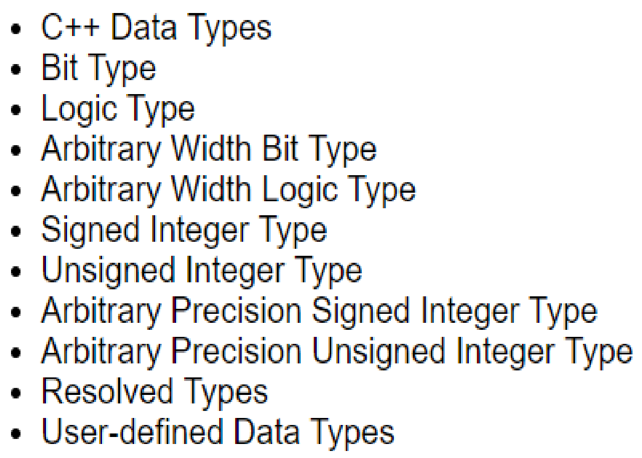
- •Modules and Hierarchy
- •Hardware data types
- •Methods and threads
- •Events, Sensitivity
- •Interface and channels

•What it is used for:

- •System-level modeling
- •Architectural exploration
- •Software development
- •Functional verification
- • High-level synthesis



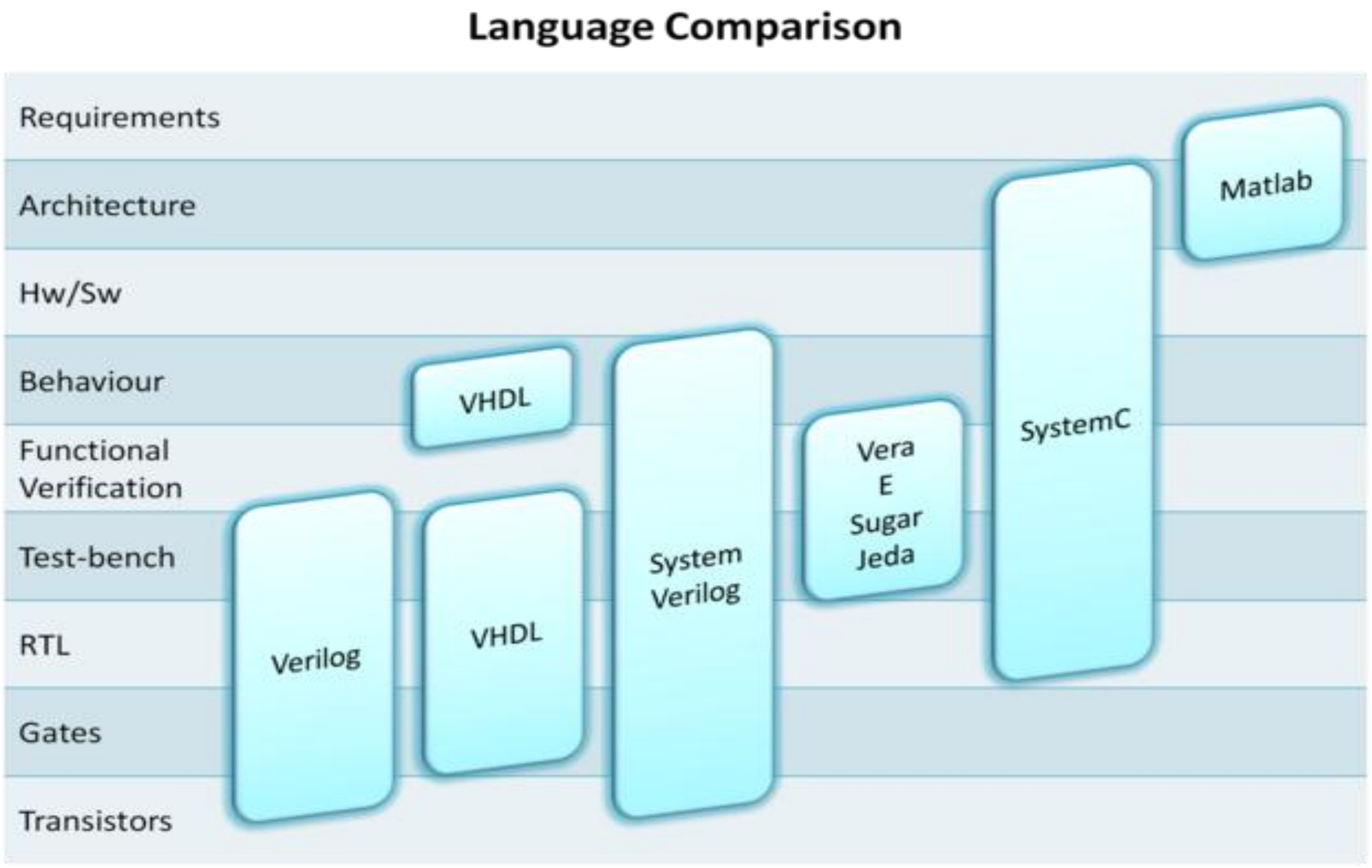
```
module fadder(
  input a,          //data in a
  input b,          //data in b
  input cin,        //carry in
  output sum_out,   //sum output
  output c_out      //carry output
  );
  wire c1, c2, c3; //wiring needed
  assign sum_out = a ^ b ^ cin; //half adder (XOR gate)
  assign c1      = a & cin;     //carry condition 1
  assign c2      = b & cin;     //carry condition 1
  assign c3      = a & b;       //carry condition 1
  assign c_out   = (c1 + c2 + c3);
endmodule
```



- C++ Data Types
- Bit Type
- Logic Type
- Arbitrary Width Bit Type
- Arbitrary Width Logic Type
- Signed Integer Type
- Unsigned Integer Type
- Arbitrary Precision Signed Integer Type
- Arbitrary Precision Unsigned Integer Type
- Resolved Types
- User-defined Data Types

Hardware Description Languages and Abstraction Levels

VANDERBILT School of Engineering



*European space Agency

Compared to other HDL’s

VANDERBILT School of Engineering

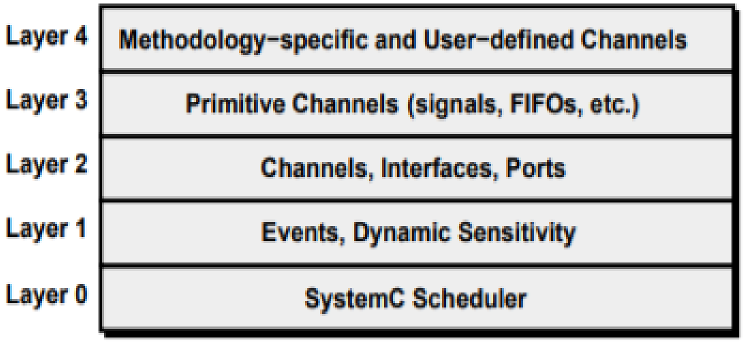
- SystemC is more on higher abstraction level then VHDL and Verilog.
- SystemC can be used on specification level, architecture level. where as VHDL and verilog are more on behavioral level and below.
- The biggest advantage of SystemC over other HDLs is that it supports **co-design** and **co-simulation** of software firmware's and hardware designs/components.
- Allows consistent changes to the design which allows evaluation of different architecture alternatives.
- Complex communication channels using hierarchy channels.
- SystemC is based on C++ class libraries. Reusable IP’s, concurrency, flexible communication.
- Another advantage is it is an open source application, whereas most of the sophisticated HDL tools are not.

Important Key-words

VANDERBILT School of Engineering

Modules: Are the basic building blocks within SystemC to partition a design.
• Break down complex systems into smaller and more manageable pieces.
•Hides internal data representation and algorithms from other modules.
•Use of public interfaces to other modules makes the entire system easier to change and easier to maintain.
•Modules are similar to module in Verilog and Entity in VHDL.

Ports: Pass data to and from the processes of a module to the external world as in Verilog and VHDL.
• Various port directions are **in**, **out**, or **inout**.
•Also can declare the data type of the port as any C++ data type, SystemC data type, or user defined type.
Types of Ports
in : Input Ports
out : Output Ports
inout : Bi-direction Ports


Layer 4
Methodology-specific and User-defined Channels
Layer 3
Primitive Channels (signals, FIFOs, etc.)
Layer 2
Channels, Interfaces, Ports
Layer 1
Events, Dynamic Sensitivity
Layer 0
SystemC Scheduler

Important Key-words

Signals : Ports are used for communicating outside the module.
• For communicating within a SystemC module we use signals.
•Signals are like wires in Verilog. It can be of any data types.
•Signal are also used for connecting two modules ports in parent module.

Process: The functionality of SystemC module is implemented in Processes.
•This is same as always block of Verilog.
•It can be either made level sensitive to model combinational logic or can be made edge sensitive logic to module sequential logic.

Channels: Provides communication between two modules.
•Primitive channel : Is derived from from sc_prim_channel, and can thus have access to the scheduler (evaluate update).
• sc_mutex, sc_fifo and sc_semaphore.

•Hierarchial channel: Is derived from sc_module, and can thus have all the features of an sc_module (processes, hierarchy etc).

Hardware data types

Hardware data types

• sc_logic and sc_lv to provide variables to be assigned with values ‘0’, ‘1’, ‘X’ and ‘Z’.
• sc_bv<100>x,y ; <sc_bit>x ------bit and bit vectors.
• sc_int<48>x; sc_uint<64>y;-----fixed and arbitrary precision.
•Function to compute resolved values. Requirement of resolved logic signals.
•New assignment values do not change immediately. There is a time slack. Propagation delay has to be considered.
•Concept of delayed signal assignment (as in HDL) and delta steps are supported in SystemC.

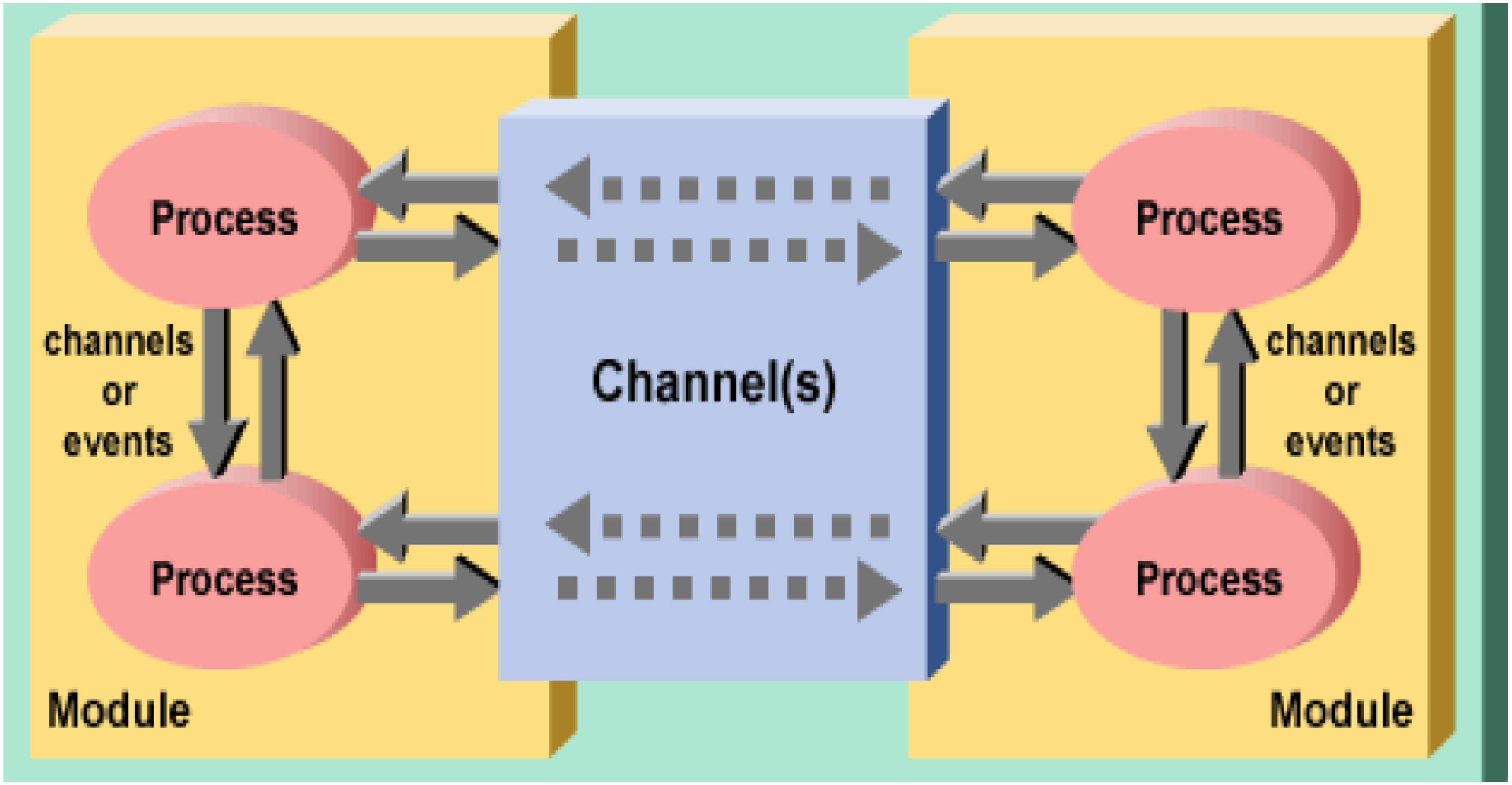
Time & clocks

•Time is an important concept in modeling hardware. SystemC provides concept of time:

```
sc_clock clock(“my_clock”, 10, 0.5, 1)
sc_clock clk (“clk”, 10, SC_NS)
```

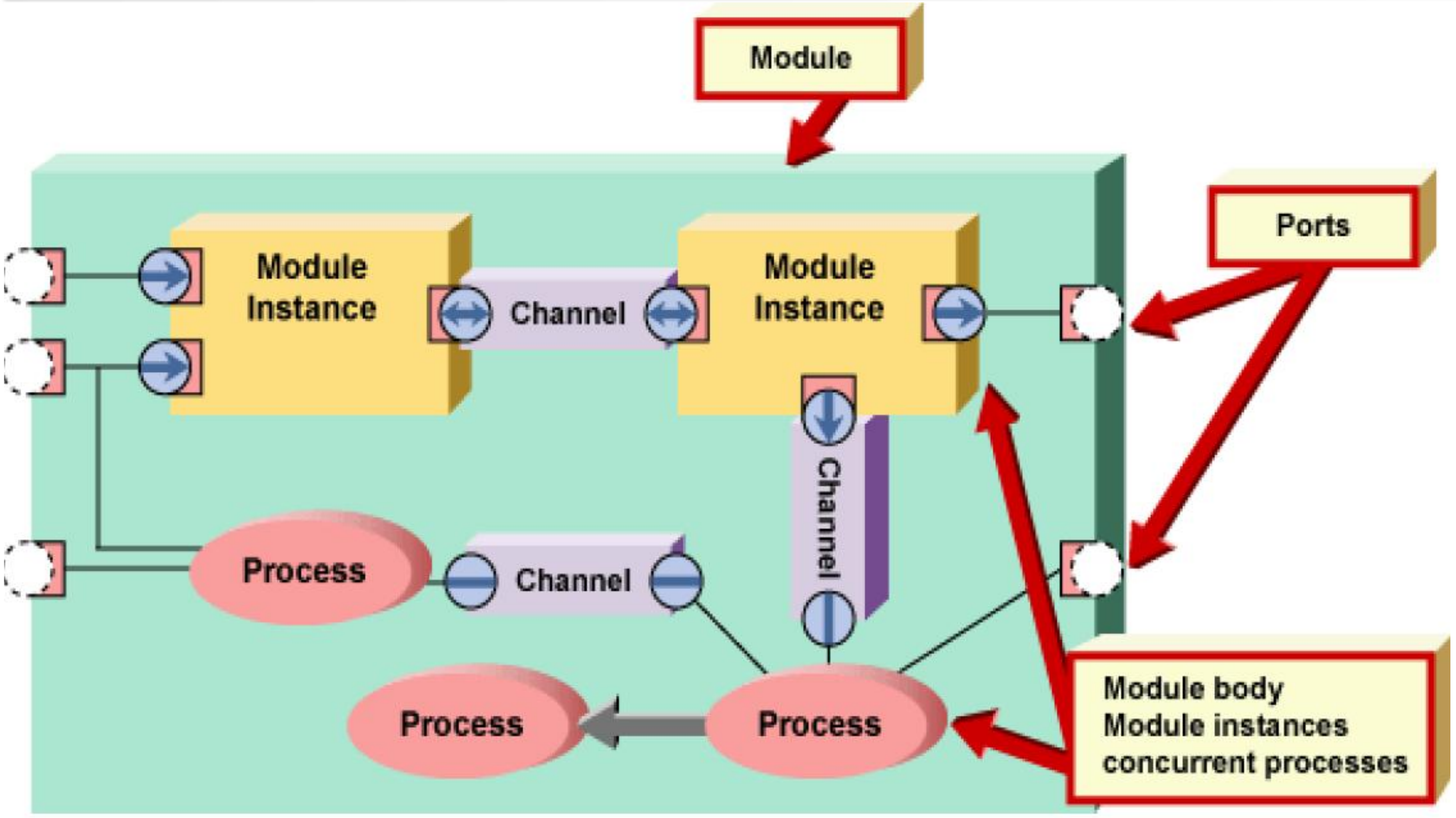
Sensitive, sensetive_pos, sensitive_neg keywords to synchronize a process to a clock.

System in SystemC



*SystemC Tutorial UC Berkeley

System in SystemC



*SystemC Tutorial UC Berkeley

Program structure in SystemC

- Every program starts with a class SC_MODULE.
- Every module must have an actor SC_CTOR with the same module name.
- Ports are used for external communication.
- Every module should have one process to give functionality to the module.

```
SC_MODULE(<module name>)
{
sc_in<data type> port1; sc_out<data type> port2; ........//multiple port declaration

<return type> <process name> (<argument type>); ......//can accommodate multiple
    processes
..................................................................//any other codes

SC_CTOR(<module name>)
{
SC_METHOD(<process name>);//immediate call to the process 'process name'
sensitive << <sensitivity list>;
...........//any other code
}
};
```

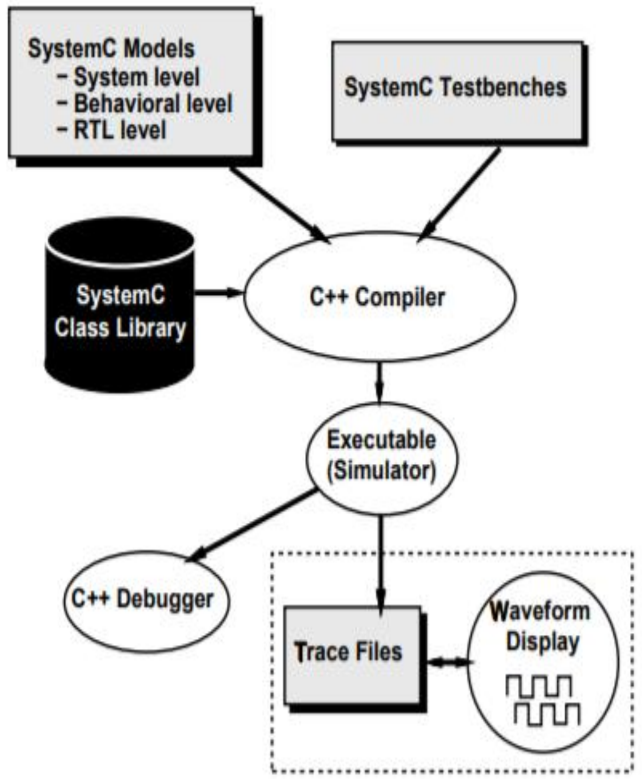
Example

```
//A simple example of and gate in SystemC
#include "systemc.h"

SC_MODULE(and_gate)
{
sc_in<sc_bit> x;
sc_in<sc_bit> y;
sc_out<sc_bit> z;

void prc_and_gate() {z=x & y;}//process which gives functionality

SC_CTOR(and_gate)
{
SC_METHOD(prc_and_gate); //process is called here
sensitive << x << y;
}
};
```

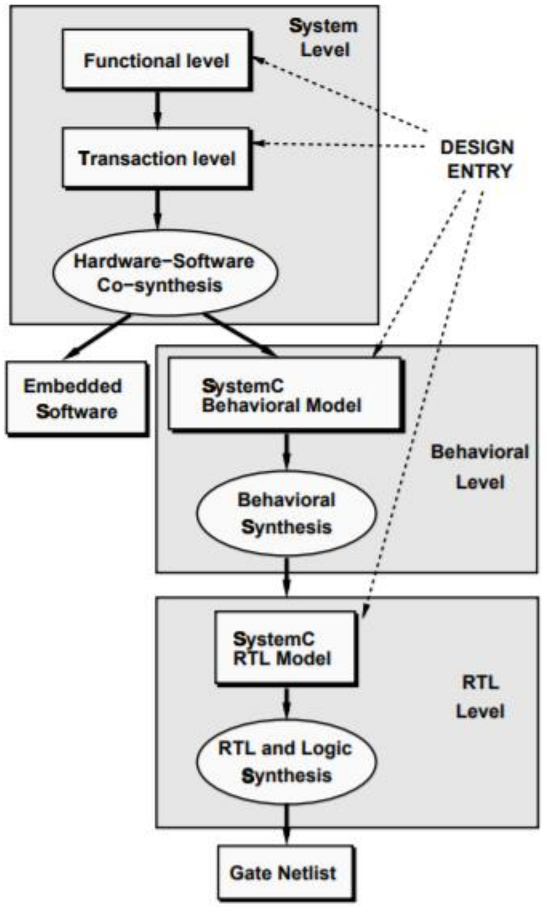
Simulation and Implementation flow

VANDERBILT School of Engineering

SystemC simulation



Implementation flow



*[3]

Advantages

•Inherits the features of C++. Which makes it convenient to work with.

•Rich in data types . Provides H/W data types.

•Strong simulation kernel making it easy to write test-benches. Cycle based simulation. (delta cycle ---initialize, evaluate, update)

•Introduces notion of time which is usually not available in C++.

•Concurrency. (which is not the case with C++)

•Offers productivity gains by letting development of both h/w and s/w simultaneously.

An insight into the material on: An Introduction to System-Level Modeling in SystemC 2.0 January 2001

History: SystemC 1.0 Highlights

SystemC 1.0: Highlights

- Provides set of modeling constructs similar to HDL.
- Simulation Kernel.
- Structural design using modules, signals and ports.
- Fixed precision arithmetic data type. (not found in any HDL's)
- Concurrent behavior is modeled using processes. (similar to Verilog)
- Communication channel.(like wires)
- Signals are prominent synchronization elements.
- The wait() cannot take parameters other than only few specified ones.(dynamic sensitivity)
- The whole structure of version1 is a bit flustered and needs a restructure to make it more generalized and reusable in structure.

Objectives of SystemC 2.0

- Primary goal is to enable System Level Modeling.
- Complete library rewrite to upgrade in system level design language (SLDL).
- General purpose modeling foundation (core language) over which specific models of computation can be built.
- Elementary component models (e.g. FIFO, timers, signals) are built on the core language.
- Channel, interfaces and events for communication and synchronization.
- Layering of specific models of computation.
- Much more powerful for transaction level modeling.
- Wait() for events and time.
- It is more structured as compared to the previous versions.

- Standardized as IEEE 1666-2011

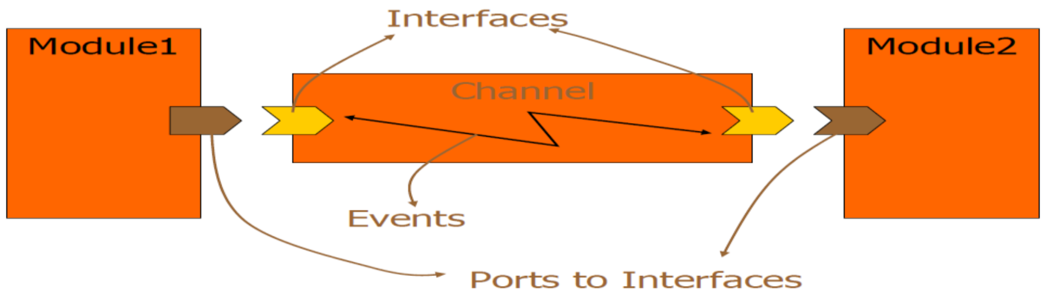
Communication and Synchronization

VANDERBILT School of Engineering

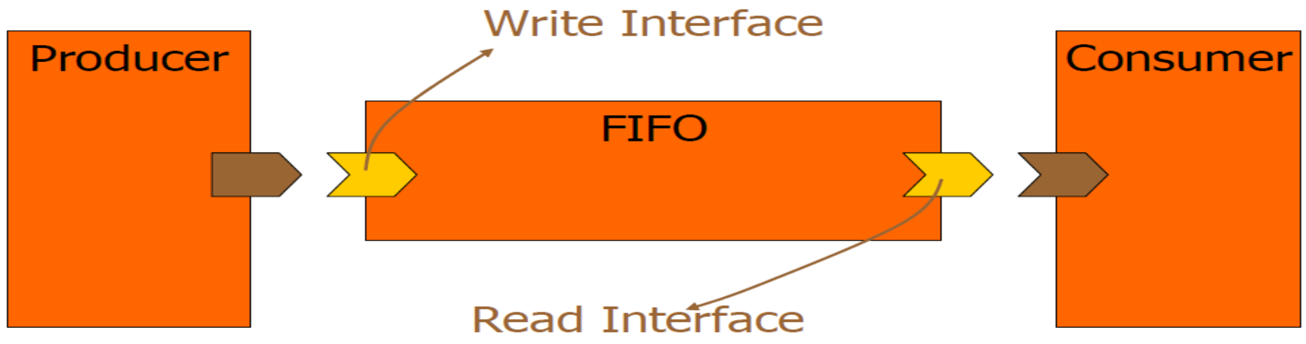
- The **SystemC 1.0** communication and synchronization is not sufficiently general for system-level modeling.

- **SystemC 2.0** introduces a generalized model for communication and synchronization using: channels, interfaces and events.

    - **Channel** is an object that serves as a container for communication and synchronization. Channels implement one or more interfaces.
    - **Interface** specifies a set of access methods to be implemented within a channel.
    - **Event** is a flexible, low-level synchronization primitive that is used to construct other forms of synchronization.



Example of FIFO

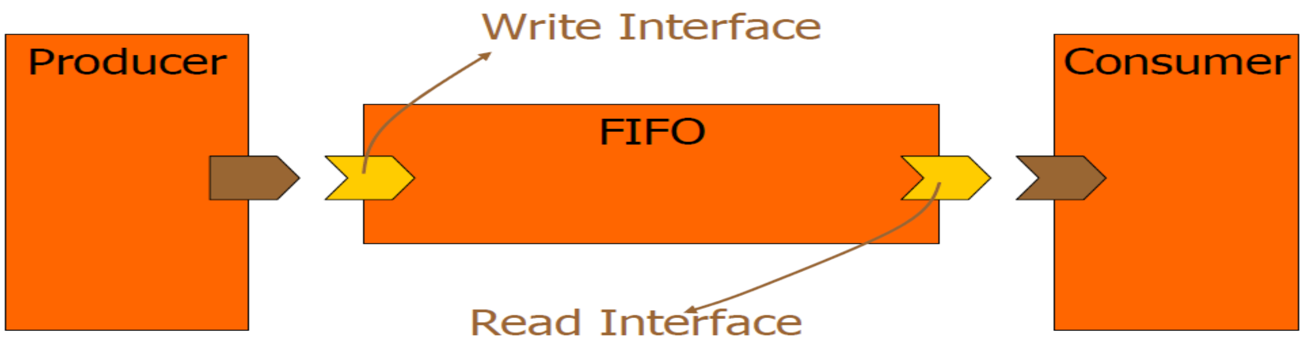


```
class fifo: public sc_channel,
   public write_if,
   public read_if
{
   private:
        enum e {max_elements=10};
        char data[max_elements];
        int num_elements, first;
        sc_event write_event,
                 read_event;
        bool fifo_empty() {…};
        bool fifo_full() {…};

   public:
        fifo() : num_elements(0),
                 first(0);
```

```
void write(char c) {
   if (fifo_full())
        wait(read_event);
   data[ <you calculate> ] = c;
   ++num_elements;
   write_event.notify();
}

void read(char &c) {
   if (fifo_empty())
        wait(write_event);
   c = data[first];
   --num_elements;
   first =  …;
   read_event.notify();
}
```

*SystemC Tutorial UC Berkeley

Example of FIFO

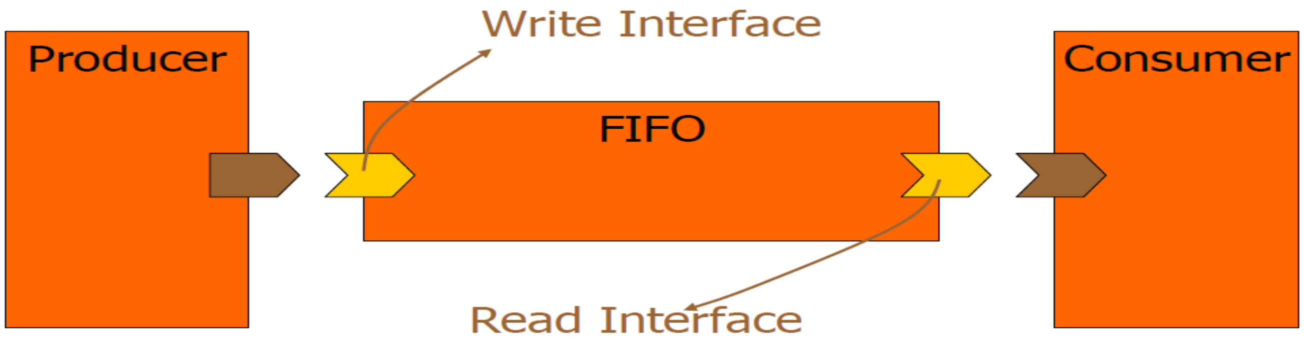


```
SC_MODULE(producer) {
   public:
        sc_port<write_if> out;

   SC_CTOR(producer) {
        SC_THREAD(main);
   }

   void main() {
        char c;
        while (true) {
           out.write(c);
           if(…)
                out.reset();
        }
   }
};
```

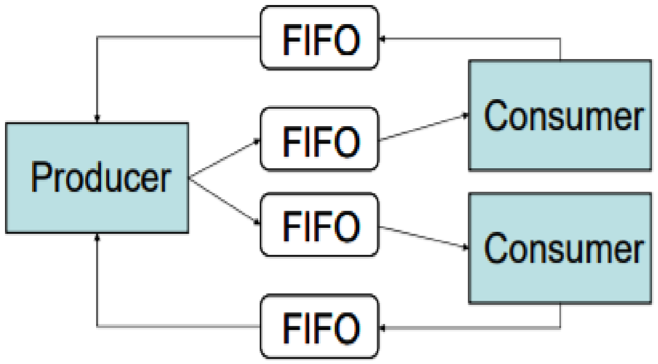
```
SC_MODULE(consumer) {
   public:
        sc_port<read_if> in;

   SC_CTOR(consumer) {
        SC_THREAD(main);
   }

   void main() {
        char c;
        while (true) {
           in.read(c);
           cout<<

   in.num_available(); }
   }
};
```

*SystemC Tutorial UC Berkeley

Example of FIFO



```
SC_MODULE(top) {
   public:
        fifo afifo;
        producer *pproducer;
        consumer *pconsumer;

   SC_CTOR(top) {
        pproducer=new producer("Producer");
        pproducer->out(afifo);

        pconsumer=new consumer("Consumer");
        pconsumer->in(afifo);
   };
```



*SystemC Tutorial UC Berkeley

Building on the core language

- Achieving a layer of specific model of computation above the general one: example: Hardware signals
- In SystemC 1.0, the hardware signal was the only mechanism available for communication and synchronization between processes.
- In SystemC 2.0, the hardware signal is now implemented completely on top of channels, interfaces and events.
- The SystemC 2.0 simulation kernel has no special support for hardware signals and is not aware if any are being used in a particular design.
- This new structure is built in-order to achieve more flexibility and versatility into the SystemC language.

Models of Computation

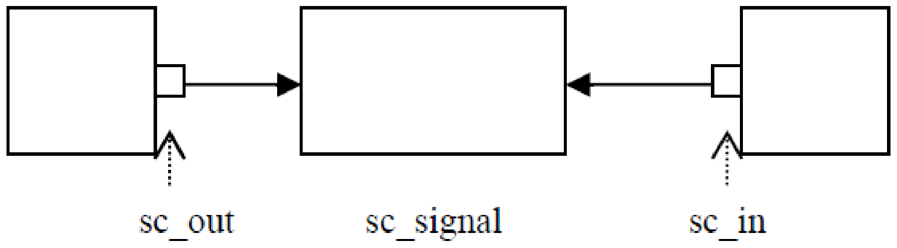
Some models of computation available in SystemC 2.0 are:

- Static Multi-rate Data-flow
- Dynamic Multi-rate Data-flow
- Kahn Process Networks
- Communicating Sequential Processes
- Discrete Event as used for
- RTL hardware modeling
- Network modeling
- Transaction-based SoC platform modeling

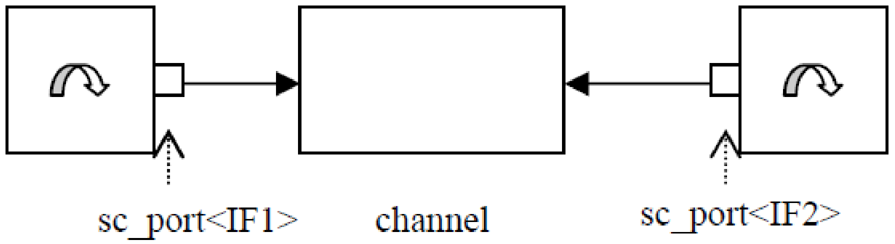
- Any model of computation other than the available ones can be built on the generalized layer beneath.

Changes through versions

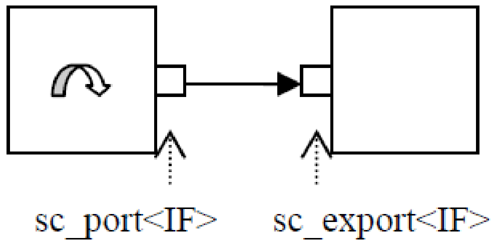
- **SystemC 1.0**



- **SystemC 2.0**



- **SystemC 2.1**



Summary of this report: My View

What this work tells us:

- Discusses briefly the features of SystemC 1.0 and its shortcoming
- Introduction to new version of 2.0 and its new features.
- Communication and synchronization features.
- Models of computation supported by the new version.
- Subtle comparison made between features of the two versions.

This work gives a good insight into the features of SystemC 2.0 which can be predominantly used in System Level Modeling.

Insight into the paper: Transaction Level Modeling in SystemC

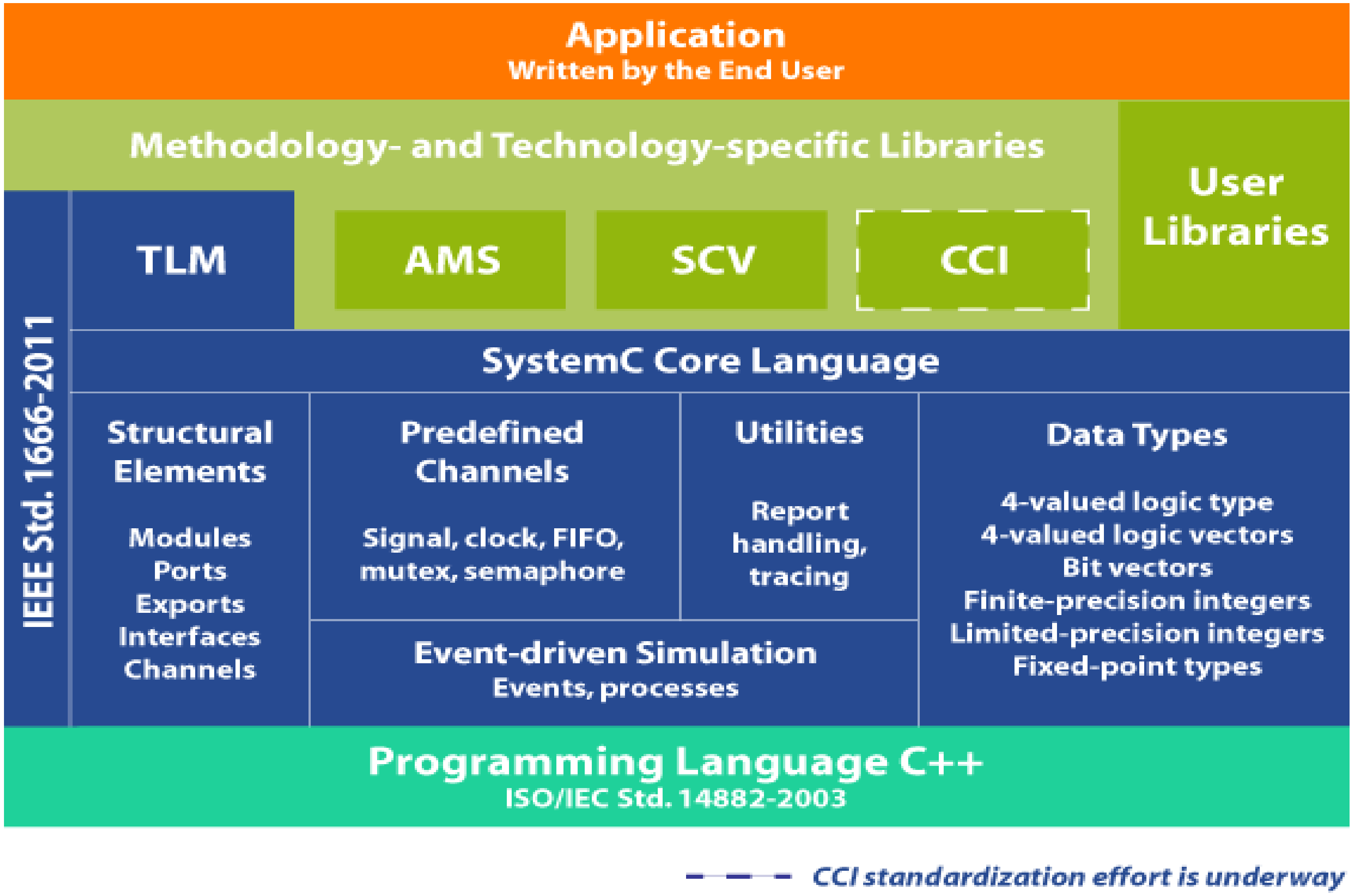
By

Adam Rose, Stuart Swan, John Pierce,
Jean-Michel Fernandez
Cadence Design Systems, Inc

January 2005

Summary of SystemC 2.0

VANDERBILT School of Engineering



*Accellera: Systems Initiative

What is TLM

- Transaction-level modeling (TLM) is a high-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units.

- TLM is an abstraction of communication among computation modules.

“**Communication is separated from computation**”

- Communication mechanisms such as **FIFOs** are modeled as **channels**, and are presented to modules using SystemC interface classes.

- Transaction requests take place by **calling interface** functions of these channel models, which encapsulate low-level details of the information exchange.

- At the transaction level, the emphasis is more on the functionality of the **data transfers** and less on their actual implementation.

Motivation and Standard

Motivation behind the use of TLM are:

- Providing an **early** platform for software development
- System Level Design Exploration and Verification
- The need to use System Level Models in Block Level Verification.

A common TLM industry standard would increase the productivity.

However, the improvement in productivity promised by such a standard can only be achieved if the standard meets a number of criteria :

- It must be easy, efficient and safe to use in a **concurrent** environment.
- It must enable **reuse** between projects and between abstraction levels within the same project.
- It must easily model hardware, software and designs which cross the hardware / software boundary.
- It must enable the design of **generic** components such as routers and arbiters.

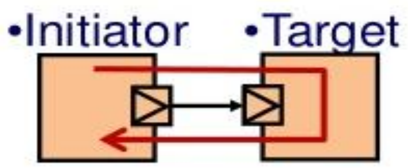
Key concepts

Key Concepts:

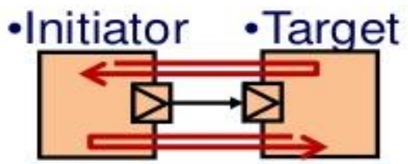
- Interfaces
  Abstract class of sc_interface

- Blocking vs. Non-blocking
  SC_THREAD: Blocking can use wait()
  SC_METHOD: non-blocking cannot use wait()

OSCI Terminology	Contains wait(.)	Can be called from
Blocking	Possibly	SC_THREAD only
Non Blocking	No	SC_METHOD or SC_THREAD



- Bi-directional vs. Uni-directional transfers.



Different TLM Interfaces

Unidirectional interface.

- Read and write is overused so we can use put and get instead.
- May have multiple implementations and hence use tag tlm_tag<T>.
- Split into two classes:

  a) Unidirectional blocking interface.

  b) Unidirectional Non-blocking interface.

Bidirectional blocking interface.

- The bidirectional blocking interface is used to model transactions where there is a tight one to one, non pipelined binding between the request going in and the response coming out.
  - e.g. address input data output.

TLM Channels

- Two widely used channels are:
  - tlm_fifo<t>
    - The implementation of the fifo is based on the implementation of sc_fifo.
    - Implements all of the unidirectional interfaces described.

  - tlm_rep_rsp_channel<REQ,RSP>
    - Two fifos, one for the request going from initiator to target and the other for the response being moved from target to initiator.
    - Four put and get interfaces.

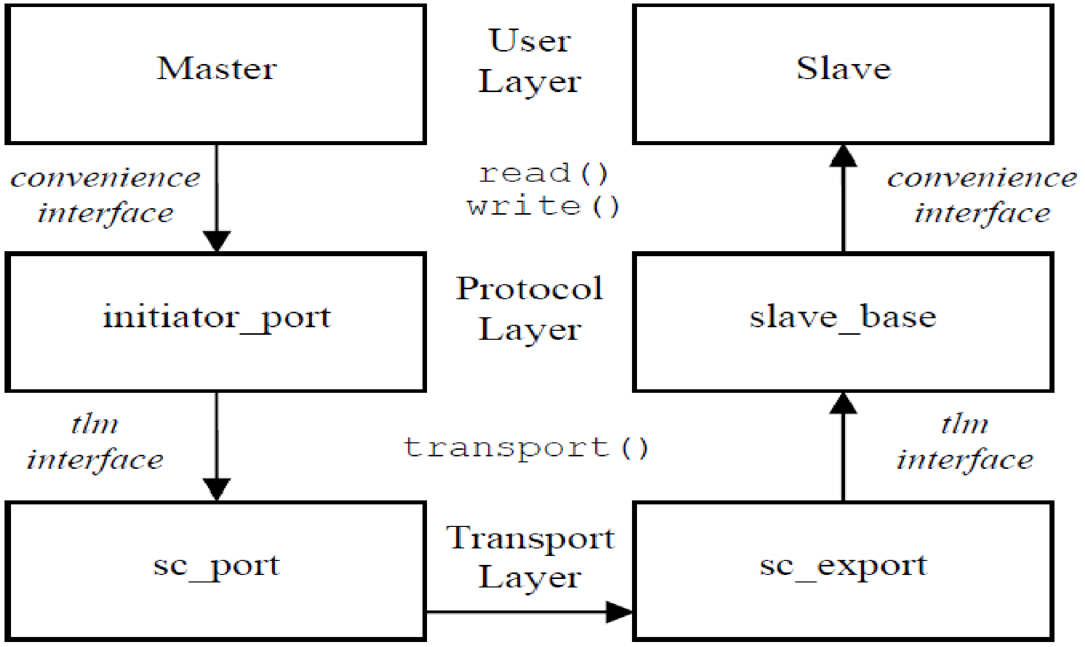
```
template < typename REQ , typename RSP >
class tlm_master_if :
  public virtual tlm_extended_put_if< REQ > ,
  public virtual tlm_extended_get_if< RSP > {};

template < typename REQ , typename RSP >
class tlm_slave_if :
  public virtual tlm_extended_put_if< RSP > ,
  public virtual tlm_extended_get_if< REQ > {};
};
```

Summary of the proposal

- The different methods discussed before form a simple transport mechanism.
- On top of this different software, hardware models and different patterns like pipelines, routers can be built.
- This will also help in modeling at different levels of abstraction. (which will be discussed further)
- The channels at different abstraction level could be easily understood as it is implemented using simple sc_fifo.
- Users can build their own channels or use the ones discussed above like tlm_fifo, tlm_rep_rsp_channel or sc_export.

Modeling of simple peripheral bus: A programmer's view

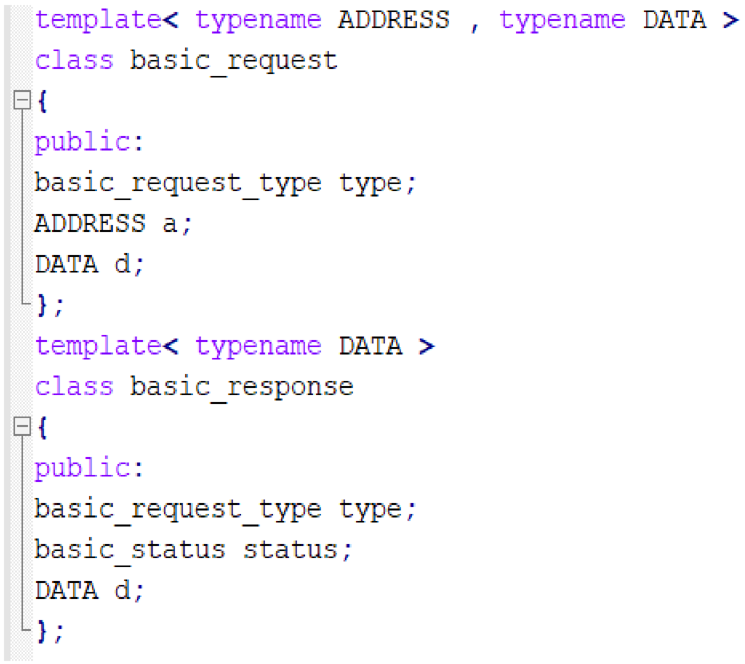
- Single master/single slave model through different levels of abstraction.
- A user will use initiator ports that supply these interfaces, and define target modules which inherit from the these interfaces.
- The infrastructure team will implement the protocol layer for the users.
- The infrastructure team then publishes the initiator port and slave base class to the users, who are then protected from the transport layer completely.



```
void master::run()
{
  DATA_TYPE d;
  for( ADDRESS_TYPE a = 0; a < 20; a++ )
  {
    initiator_port.write( a , a + 50 );
  }
  for( ADDRESS_TYPE a = 0; a < 20; a++ )
  {
    initiator_port.read( a , d );
  }
}
```

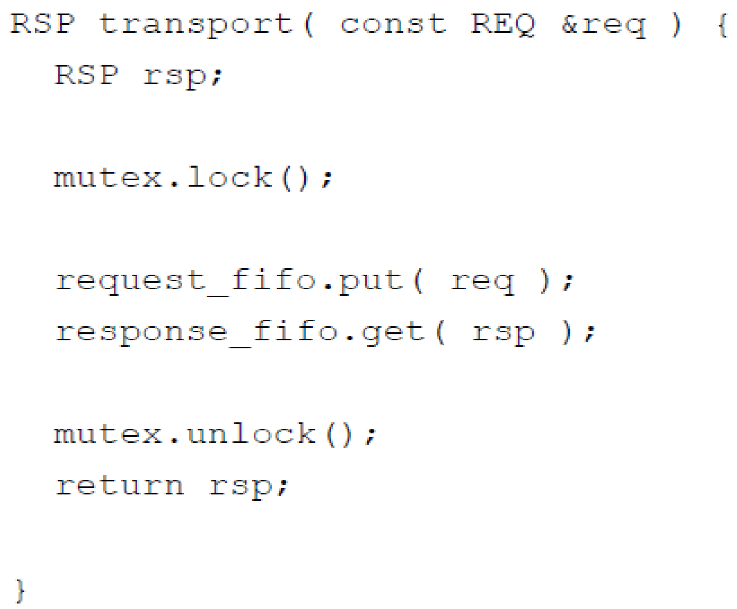
The protocol

- At this abstract modeling level, the request and response classes describe the information going in to the slave in the request and the information coming out of the slave in the response.
- Only the request, reply classes are compulsory. The implementation team may not supply the initiator port and slave base class.



```
template< typename ADDRESS , typename DATA >
class basic_request
{
public:
basic_request_type type;
ADDRESS a;
DATA d;
};
template< typename DATA >
class basic_response
{
public:
basic_request_type type;
basic_status status;
DATA d;
};
```

Protocol with response request class



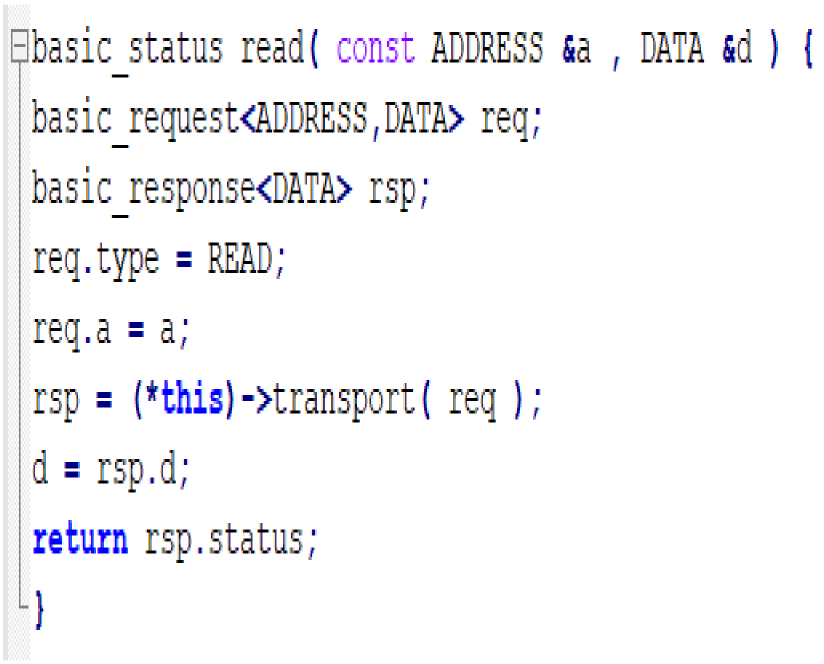
```
RSP transport( const REQ &req ) {
  RSP rsp;

  mutex.lock();

  request_fifo.put( req );
  response_fifo.get( rsp );

  mutex.unlock();
  return rsp;

}
```

Convenience layer to transport layer

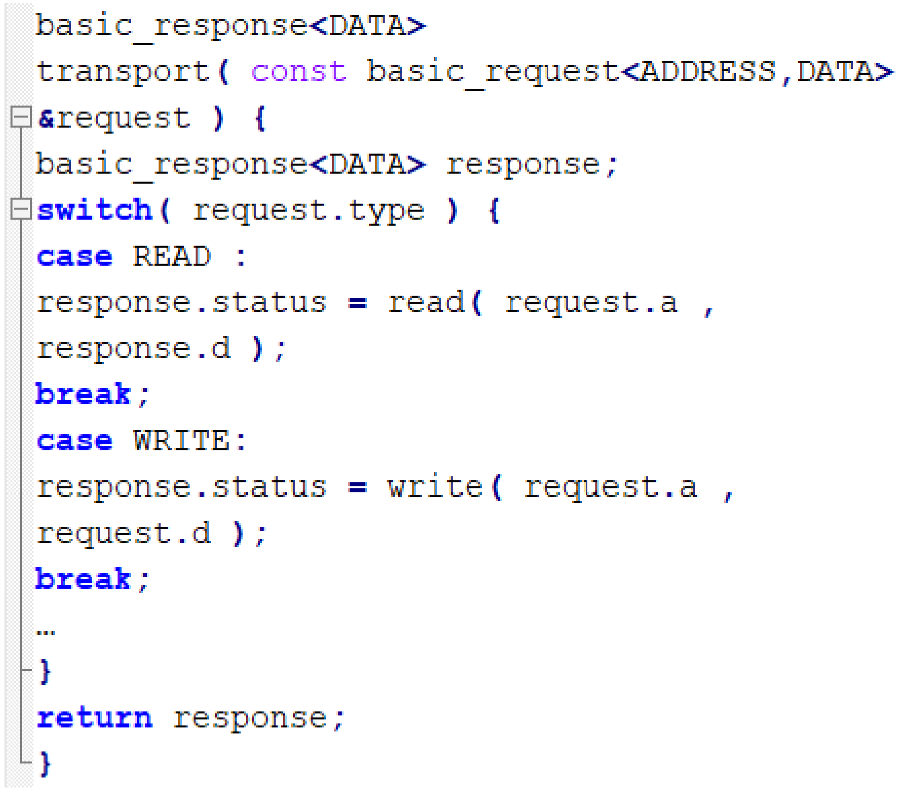
Initiator and Slave base class

VANDERBILT School of Engineering

- On the master side (Initiator port), infrastructure team supplies an initiator port which translates from the convenience layer to the transport layer.
- The slave base class translates back from the transport layer to the convenience layer.



```
basic_status read( const ADDRESS &a , DATA &d ) {
basic_request<ADDRESS,DATA> req;
basic_response<DATA> rsp;
req.type = READ;
req.a = a;
rsp = (*this)->transport( req );
d = rsp.d;
return rsp.status;
}
```

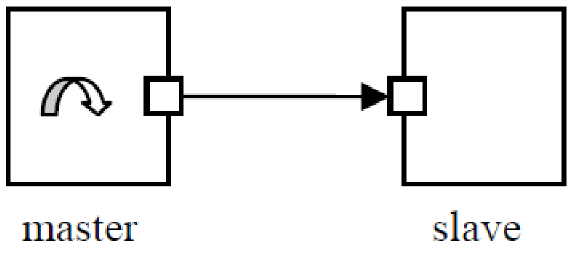


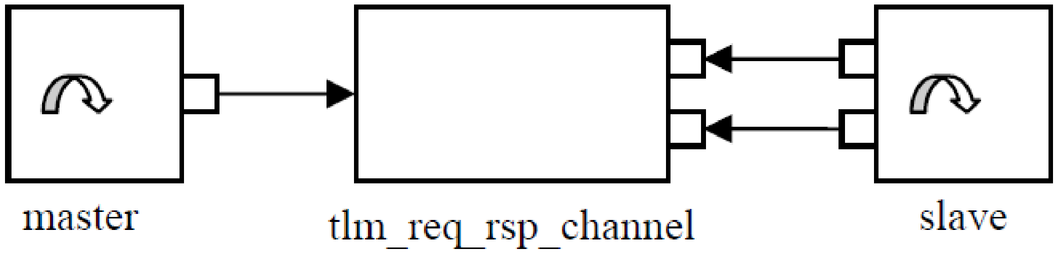
```
basic_response<DATA>
transport( const basic_request<ADDRESS,DATA>
&request ) {
basic_response<DATA> response;
switch( request.type ) {
case READ :
response.status = read( request.a ,
response.d );
break;
case WRITE:
response.status = write( request.a ,
request.d );
break;
…
}
return response;
}
```

Some Examples: PV Master/ PV Slave

VANDERBILT School of Engineering

- Single thread in master to send sequence of read and writes to slave.
- Connection between master and slave through bidirectional transport interface.





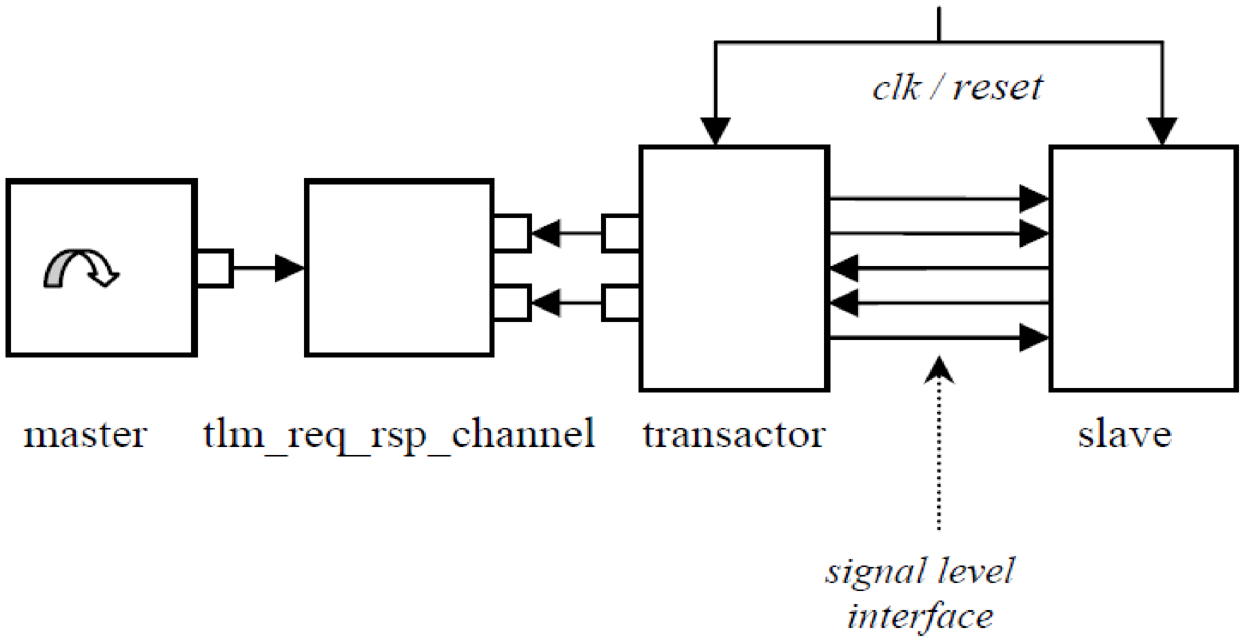
```
mem_slave::mem_slave( const sc_module_name
&module_name , int k ) :
sc_module( module_name ) ,
target_port("iport")
{
target_port( *this );
memory = new ADDRESS_TYPE[ k * 1024 ];
}
basic_status
mem_slave::
read( const ADDRESS_TYPE &a , DATA_TYPE &d )
{
d = memory[a];
return basic_protocol::SUCCESS;
}
basic_status
mem_slave::
write( const ADDRESS_TYPE &a, const DATA_TYPE &d)
{
memory[a] = d;
return basic_protocol::SUCCESS;
```

```
void mem_slave::run()
{
basic_request<ADDRESS_TYPE,DATA_TYPE> request;
basic_response<DATA_TYPE> response;
for(;;)
{
request = in_port->get();
response.type = request.type;
switch( request.type )
{
case basic_protocol::READ :
response.d = memory[request.a];
response.status = basic_protocol::SUCCESS;
break;
case basic_protocol::WRITE:
…
}
out_port->put( response );
}
}
```

Example 3

VANDERBILT School of Engineering

- Refining the slave to register transfer level.
- The key component in this system is the transactor.
- the transactor has to implement at least one state machine to control the bus. SC_METHOD.
- However in using SC_METHOD we have to use non-blocking interface while accessing the fifo.

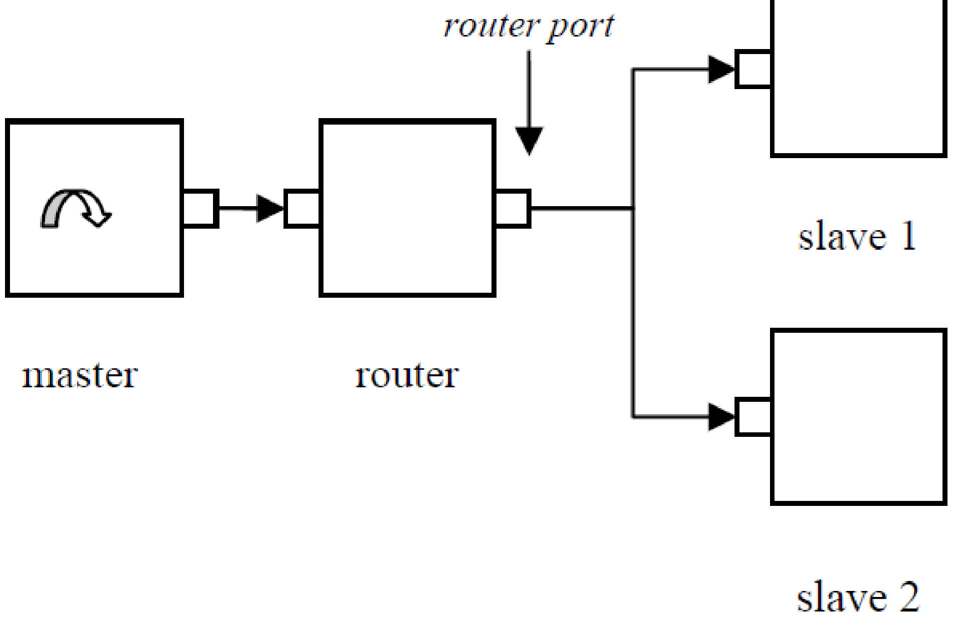


Modeling patterns using TLM Router

VANDERBILT School of Engineering

- Routers are important to route traffic between master and slaves.
- Address map, router module and router port are generic components.
- The router receives request from the master and if it finds the slave, it successfully subtracts the base address from the request, forwards the adjusted request and forwards to slave.

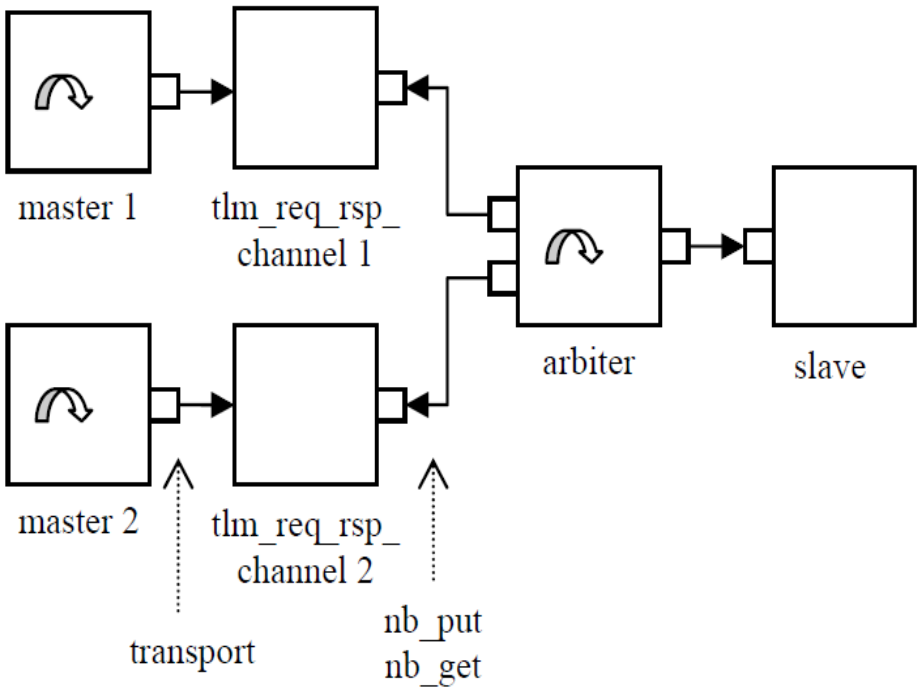
```
// an example address map
// slave one is mapped to [ 0 , 0x10 )
// slave two is mapped to [ 0x10, 0x20
slave_1.iport 0 10
slave_2.iport 10 20
```



Arbitrer

- Arbitrates between two simultaneous request, when the concept of time is involved.
- It polls the fifo_requests and forwards the more important one to the slave.

```
virtual void run() {
  port_type *port_ptr;
  multimap_type::iterator i;
  REQ req;
  RSP rsp;
  for( ;; ) {
    port_ptr = get_next_request( i , req );
    if( port_ptr != 0 ) {
      rsp = slave_port->transport( req );
      (*port_ptr)->put( rsp );
    }
    wait( arb_t );
  }
}
```



Summary of this paper: My View

VANDERBILT School of Engineering

- This paper provides the motivation behind the use of TLM. Also focuses the main aspect of creating a standard which would increase the productivity.
- Key concepts of interfaces, blocking vs non-blocking, and unidirectional vs. bidirectional message transfer.
- Various examples including master/ slave and interfaces.
- SoC components like arbiters, routers and pipe lines are discussed.
- So, most of the SystemC concepts of modules, process, signals, events have been implemented to practically show their usage in TLM.

Conclusion

- This Talk was intended to introduce SystemC and its features, and how it has been used in transaction level modeling. We have seen that SystemC works at a higher level of abstraction than its counterparts of HDL.

- We also saw that it emphasizes on communication rather than the implementation itself. The big advantage it provides is co-development and co-simulation of hardware and software, which in turn would increase the speed of production.

- This language (rather library of C++ classes) forms the basis for system level modeling and transaction level modeling.

Seminar-2

- The key concepts of this talk would be used in the next part of the seminar which would rather consider some more real examples of TLM's usage in modeling and simulation of CPS, WSN, etc.

References

VANDERBILT School of Engineering

[1] An Introduction to System level modeling in SystemC 2.0, January 2001.

[2] Adam Rose, Stuart Swan, John Pierce, Jean-Michel Fernandez, Cadence Design Systems, Inc., included in the TLM SystemC package, November 2004.

[3] P.R. Panda. SystemC: A modeling platform supporting multiple design abstractions. In Proceedings of the International Symposium on Systems Synthesis (ISSS), pages 75–80. ACM, 2001.

[4] OSCI TLM-2.0 language reference manual, July 2009

[5] John Moondanos, SystemC Tutorial.

[6] http://www.asic-world.com/verilog/intro1.html