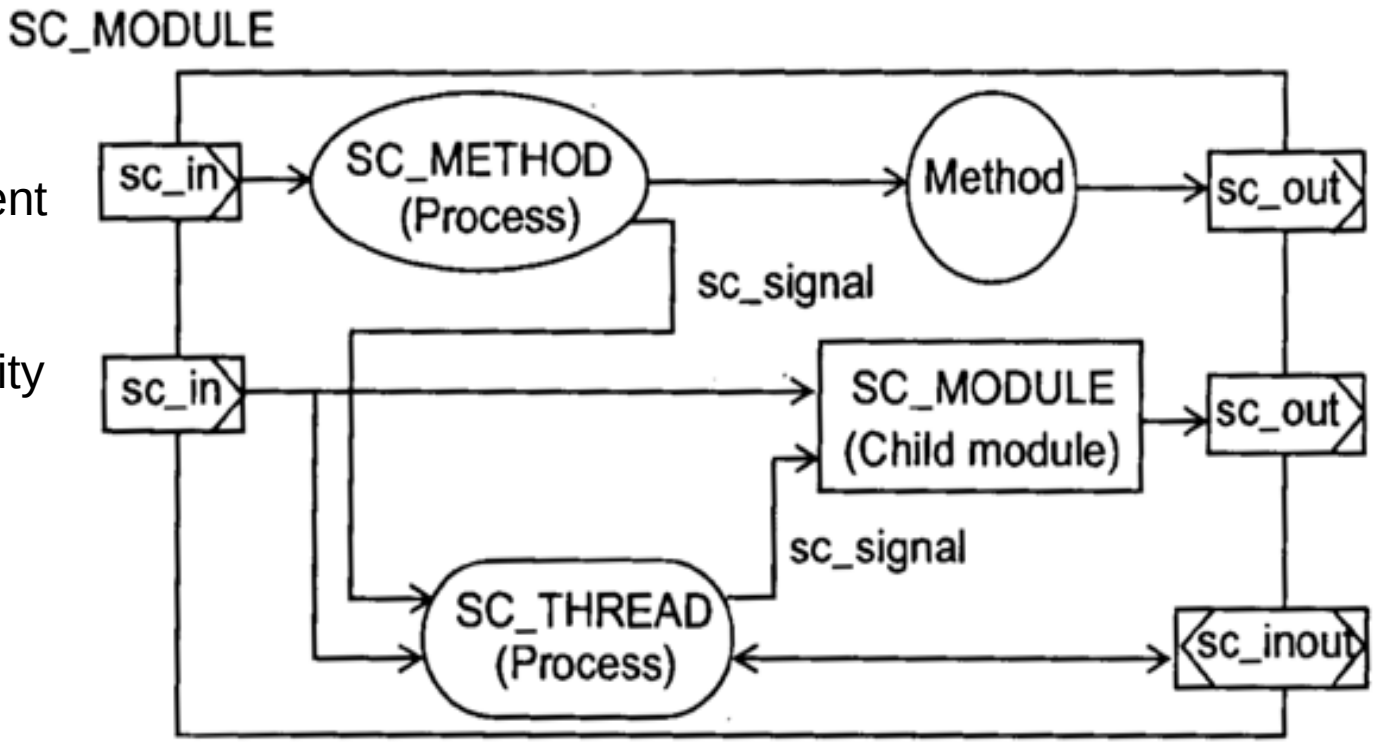
System C part 2

BITS Pilani
Hyderabad Campus

Software for Embedded Systems

SC_THREAD

- Can be suspended
- Resumed on an event
- Resumed after time delay
- Can have a sensitivity list



A counter


```
// File: upc_wait.h
#include "systemc.h"
const int SIZE = 8;
const int INCR_SIZE = 3;

SC_MODULE (upc_wait) {
  sc_in<bool> cp, res, stop, up, ld;
  sc_in<sc_uint<SIZE> > din;
  sc_in<sc_uint<INCR_SIZE> > incr;
  sc_out<sc_uint<SIZE> > dout;

  void prc_upc_wait();

  SC_CTOR (upc_wait) {
    SC_THREAD (prc_upc_wait);
    sensitive_pos << cp;
  }
};
```

```
// File: upc_wait.cpp
#include "upc_wait.h"

void upc_wait::prc_upc_wait() {
  // Never gets out of the process:
  while (1) {
    wait();

    if (res)
      dout = 0;

    wait();

    if (ld)
      dout = din;

    while (! stop) {
      wait();

      if (up)
        dout = dout.read() + incr.read();
      else
        dout = dout.read() - incr.read();
    }
  }
}
```

SC_EVENT

Fundamental synchronization primitive
Does not have a type
Does not transmit any value
Transfers control from one process to the other
An event notification causes sensitive processes to be resumed.
An event can be declared explicitly by using the SC_Event type
EX: Sc_Event write_back;
Write_back.Notify(); Triger for immediate notification

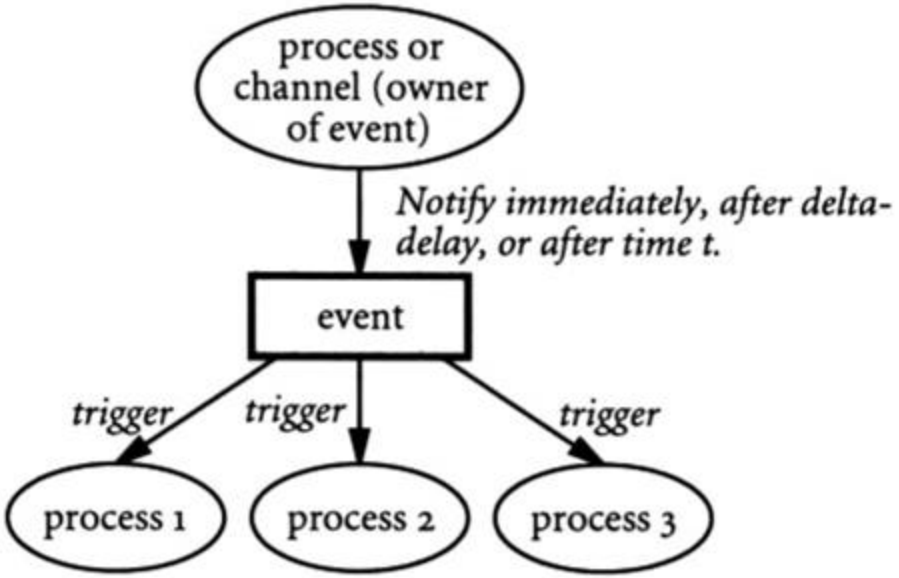
Write_back.Notify(20,SC_NS); Delayed Notification

Write_back.Cancel(); cancel a scheduled and un-occurred event

Event notification

Owner of the event is responsible for reporting the change to the event object. (notification)

Even object is responsible to keep a list of processes sensitive to it



Time unit

SC_FS ;Femto second

SC_PS ;pico second

SC_NS ;nano second

SC_US ;Micro second

SC_MS ;milli second

SC_SEC ;Seconds

Ex:

Sc_time (42,SC_PS)

Notify()

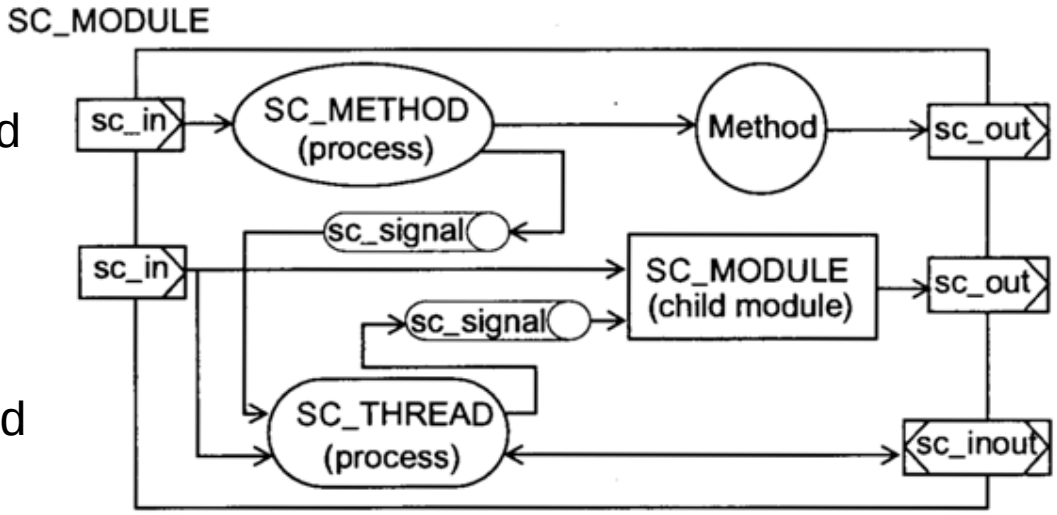

```
sc_event my_event;

my_event.notify(); // notify immediately

my_event.notify( SC_ZERO_TIME ); // notify next delta cycle

my_event.notify( 10, SC_NS ); // notify in 10 ns

sc_time t( 10, SC_NS );
my_event.notify( t ); // same
```

The *wait()* function

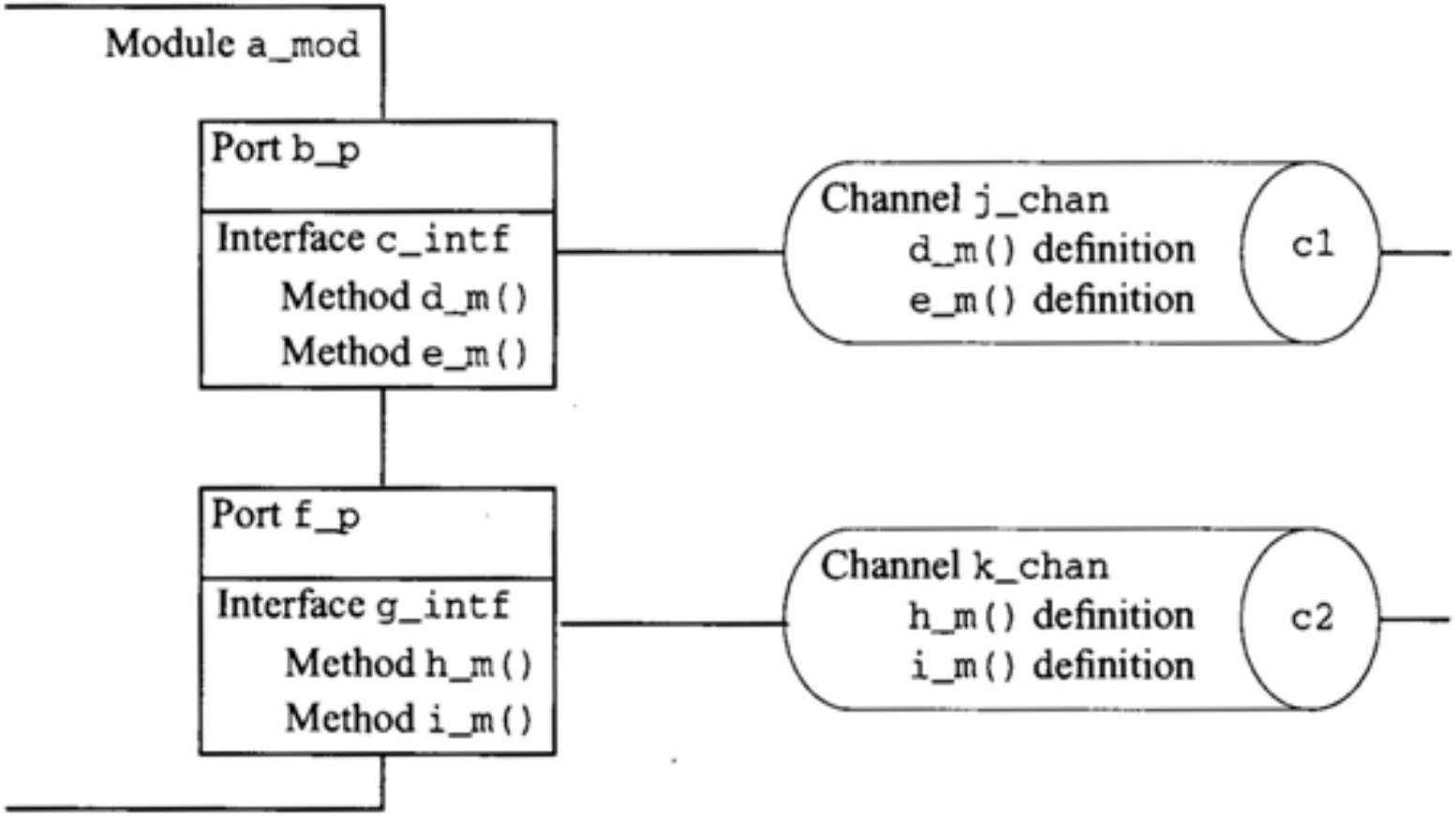

```
// wait for 200 ns.
sc_time t(200, SC_NS);
wait( t );

// wait on event e1, timeout after 200 ns.
wait( t, e1 );

// wait on events e1, e2, or e3, timeout after 200 ns.
wait( t, e1 | e2 | e3 );

// wait on events e1, e2, and e3, timeout after 200 ns.
wait( t, e1 & e2 & e3 );

// wait for 200 clock cycles, SC_CTHREAD only (SystemC 1.0).
wait( 200 );

// wait one delta cycle.
wait( 0, SC_NS );

// wait one delta cycle.
wait( SC_ZERO_TIME );
```

Channel

- An object that serves as a container for communication and synchronization
- Signal is a primitive channel
- SC_Thread process reads the value of a channel
- Output of a process is connected to a channel
- A channel implements multiple interfaces.
- A port can read a channel using Read method of channel interface
- A port can write to a channel using write method of channel interface



Ports, interfaces and channels

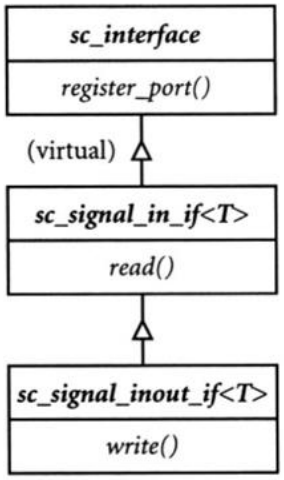


Ports

Model interacts with well defined boundaries

Ports corresponds to interfaces.

System C provides template class sc_port to create ports.




```
template<class IF, int N=1>
class sc_port: ... // class derivation details omitted
{
public:
    IF* operator->();
    // other member functions and member variables
};
```

Channels..

Interfaces and ports describe what functions are available in the communication.

Channels define how these operations are performed.

Channel implements the interface.

Different channels implement same interface in different ways.

A channel may implement more than one interface.

Primitive channels

Atomic in nature

Do not contain processes

i. `sc_signal<T>`: The basic signal that is part of SystemC RTL.

ii. `sc_signal_rv<N>`: The resolved vector logic signal.

iii. `sc_signal_resolved`: The resolved scalar logic signal.

iv. `sc_fifo<T>`: Models a fifo (first-in, first-out) register.

v. `sc_mutex`: Used to model shared variables.

vi. `sc_semaphore`: Similar to `sc_mutex`.

vii. `sc_buffer<T>`: Same as `sc_signal` except that an event is generated even if the new value being assigned is identical to the previous value.

Hierarchical channels

Can also contain channels and modules and shared data

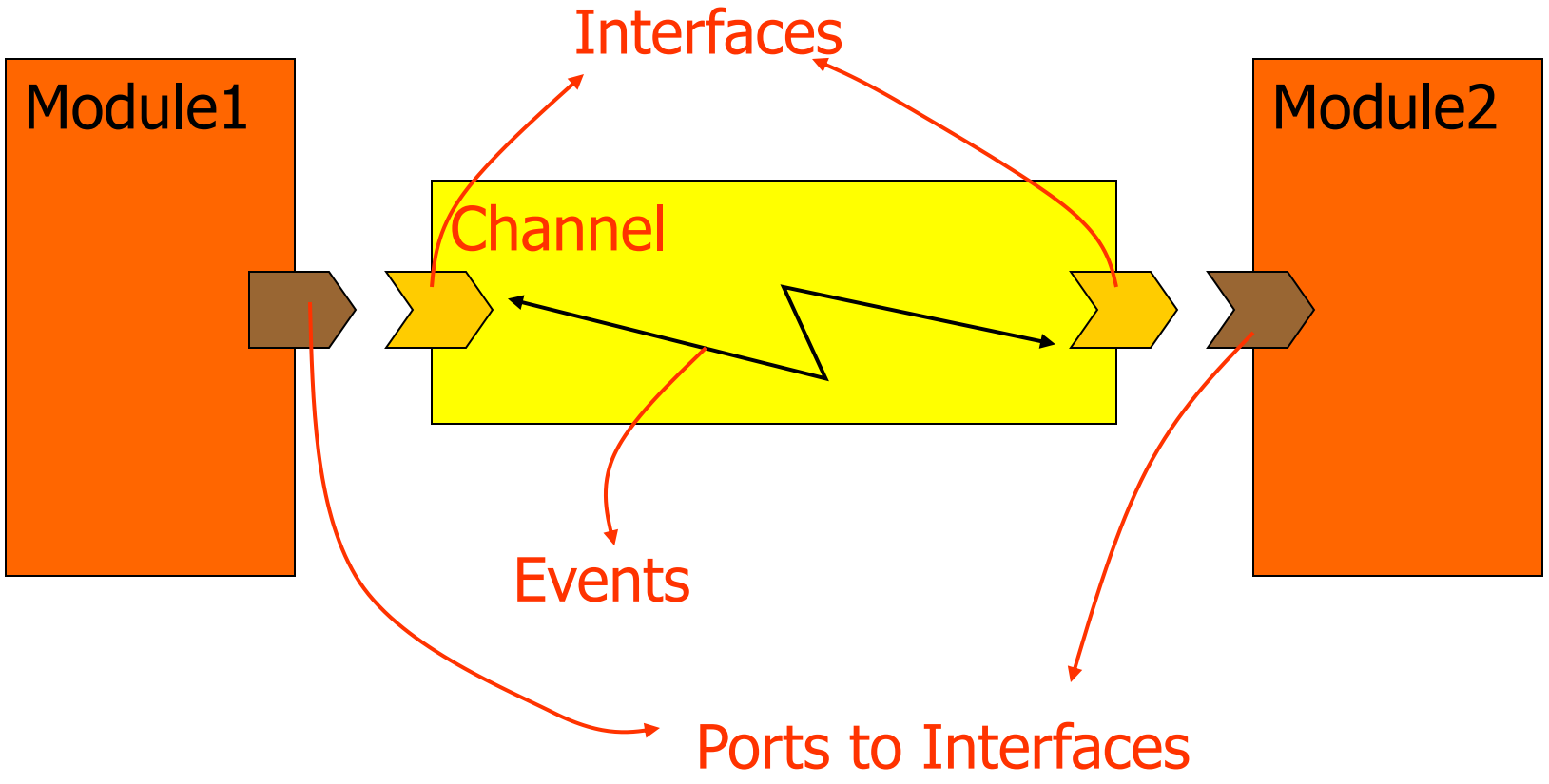
```
template <class T>
class channel_name :
  public sc_channel,
  public interface1,
  public interface2
{
public:
  // Data members
    // Can be ports, channels or variables.
  // Constructor:
    // Contains instantiation of other channels,
    // modules and processes.
  // Interface method definitions (interface1,
  // interface2)
private:
  // Data members
protected:
  // Data members
};
```

Shared data members

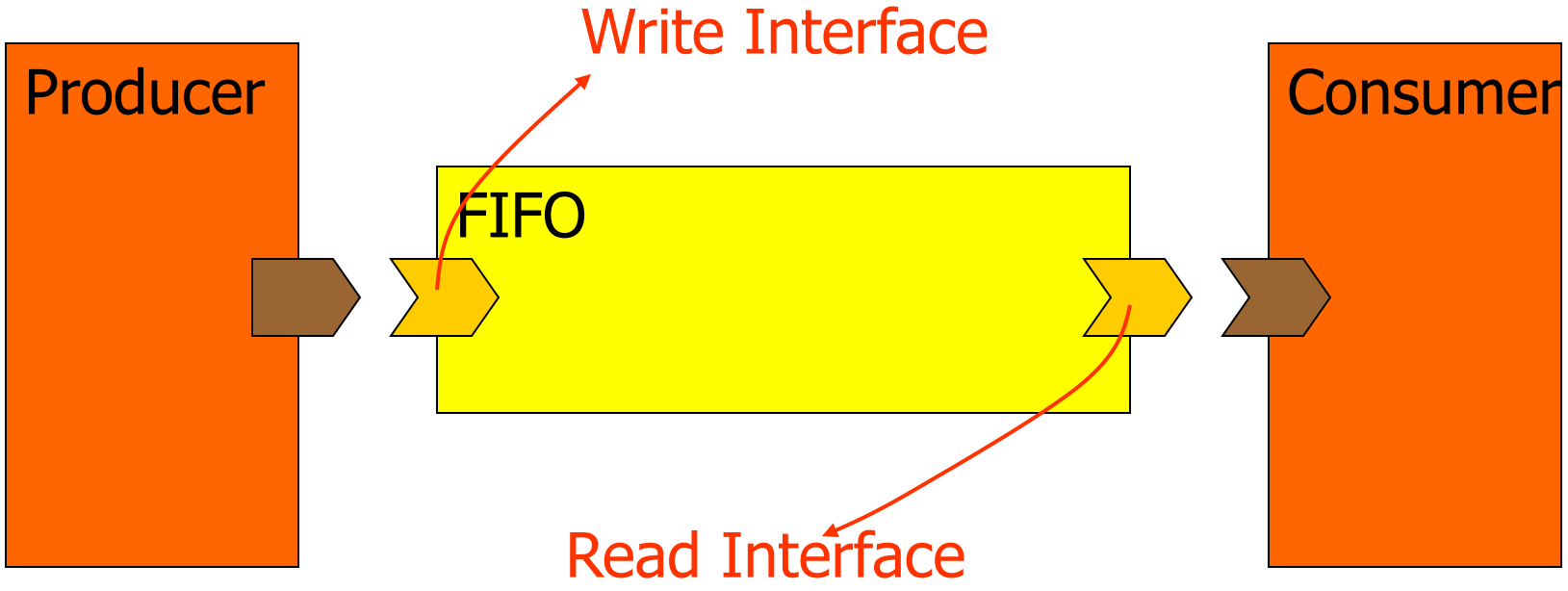
Data members declared in SC_Module class

Can be shared among other processes

Communication and Synchronization

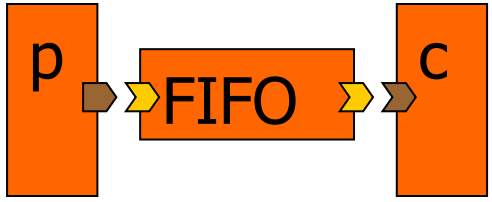


A Communication Modeling Example: FIFO

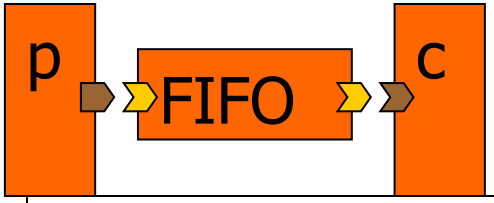


FIFO Example: Declaration of Interfaces


```
class write_if : public sc_interface
{
   public:
        virtual void write(char) = 0;
        virtual void reset() = 0;
};

class read_if : public sc_interface
{
   public:
        virtual void read(char&) = 0;
        virtual int num_available() = 0;
};
```



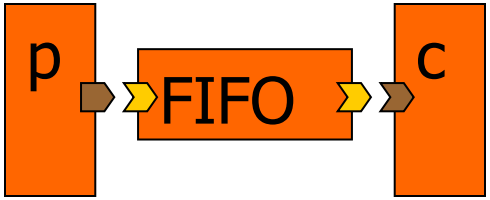
FIFO Example:
Declaration of *FIFO channel*




```
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:
        SC_CTOR(fifo) {
           num_elements = first=0;
        }
```

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

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

Declaration of *FIFO channel* (cont'd)




```
    void reset() {
         num_elements = first = 0;
    }

    int num_available() {
         return num_elements;
    }
}; // end of class declarations
```

FIFO Example (cont'd)

All channels must

- be derived from `sc_channel` class
  - SystemC internals (kernel\sc_module.h)

    ```
    typedef sc_module sc_channel;
    ```
- be derived from one (or more) classes derived from `sc_interface`
- provide implementations for all pure virtual functions defined in its parent *interfaces*

FIFO Example (cont'd)

Note the following extensions beyond SystemC 1.0

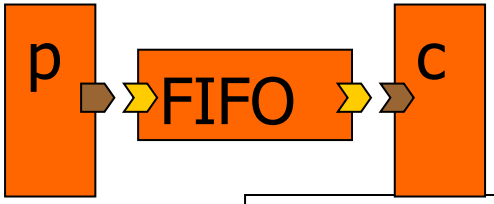
- wait() call with arguments => dynamic sensitivity
  - wait(*sc_event*)
  - wait(*time*) // e.g. wait(200, SC_NS);
  - wait(*time_out, sc_event*) //wait(2, SC_PS, e);
- Events
  - are the fundamental synch. primitive in SystemC 2.0
  - Unlike signals,
    - have no type and no value
    - always cause sensitive processes to be resumed
    - can be specified to occur:
      - immediately/ one delta-step later/ some specific time later

Completing the Comm. Modeling Example


```
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();
        }
   }
};
```

Completing the Comm. Modeling Example ..




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

   SC_CTOR(top) {
        afifo = new fifo("Fifo");

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

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

Completing the Comm. Modeling Example …

Note:

- Producer module
  - sc_port<*write_if*> out;
    - Producer can only call member functions of *write_if* interface
- Consumer module
  - sc_port<*read_if*> in;
    - Consumer can only call member functions of *read_if* interface
    - e.g., Cannot call `reset()` method of write_if
- Producer and consumer are
  - unaware of how the channel works
  - just aware of their respective *interfaces*
- Channel implementation is hidden from communicating modules

Completing the Comm. Modeling Example (cont'd)

Advantages of separating communication from functionality

- Trying different communication modules
- Refine the FIFO into a software implementation
  - Using queuing mechanisms of the underlying RTOS
- Refine the FIFO into a hardware implementation
  - Channels can contain other channels and modules
    - Instantiate the hw FIFO module within FIFO channel
    - Implement read and write interface methods to properly work with the hw FIFO
    - Refine read and write interface methods by inlining them into producer and consumer codes

Simulation semantics

Scheduler controls the timing and order of the execution processes, handles the event notifications and updates the channels.

1. Initialize: execute each process once
2. Evaluate: Selet a process ready to run and resume its execution
3. If there are still processes ready to run repeat step 2
4. Update: Execute any pending calls
5. If there are any pending delta delay notifications , go to step 2
6. If there are no timed notifications simulation is finished.
7. Else advance the time to earliest pending notification.
8. Determine which processes are ready and go to sstep 2.

Models of computation (MOC) with system c

Model of computation (MOC)

The model of time employed and the event ordering constraints within the system.

The supported method of communication across concurrent processes

Rules of process activation

Models of Computation within SystemC

- Flexible communication and synchronization capabilities in
- SystemC 2.0 enable a wide range of MOCs to be naturally modeled.
- VHDL/verilog etc have fixed model of computation
- System C is designed to customize the MOC over basic model.
- Users ability to create customized channels ,interfaces, ports and modules
- Examples: RTL, Process Networks, Static Dataflow, Transaction Level Models, Discrete Event etc.

Well known models of computation

RTL model of computation

Static multirate dataflow

Dynamic multirate dataflow

Kahn process networks

Discrete event as used for hardware modeling

Transaction level modeling (TLM)

RTL model of computation

Informal term used in digital hardware design

All communication between processes occur by signals

Processes represent sequential or combinational logic

Ports of an RTL module correspond to wires in real world implementation.

Kahn process network

Useful for building algorithmic models like DSP

Computing blocks execute concurrently.

Connected by channels which carry series of data tokens

Channels are infinite length FIFO channels

Processes do blocking read and non blocking write for interaction.

Deterministic models. (Process execution order does not effect the token values seen on FIFO channels.)

Kahn Process Network MOC in SystemC

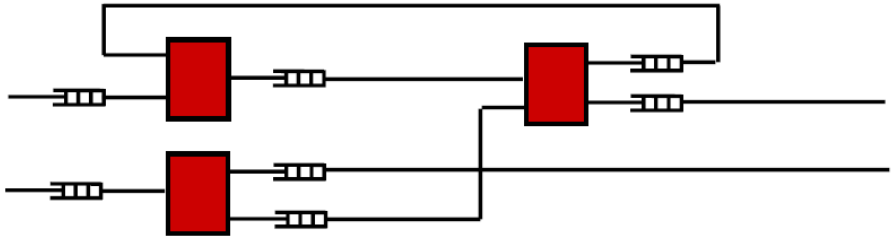
Very useful for high level system modeling

Modules communicate via FIFOs (sc_fifo<T>) that suspend readers and writers as needed to reliably deliver data items.

Easy to use and guaranteed to be deterministic

Pure KPN has no concept of time

With annotated time delays, becomes *timed functional model or performance model.*



Static dataflow models (SDF)

Special case of KPNs

Functionality of each process is to

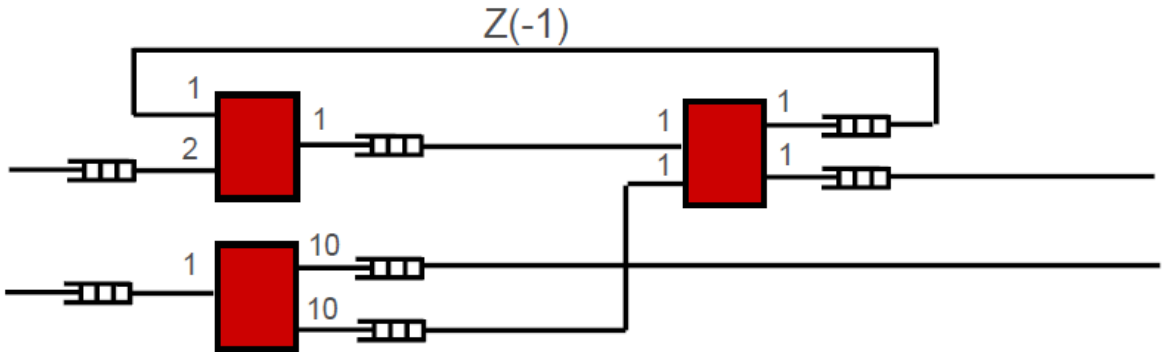
- read all input tokens
- Process
- Write all output tokens

Number of tokens read each time is fixed at compile time

Faster as it needs no dynamic scheduling

Static Dataflow MOC in System C

- A proper subset of the KPN MOC
- Each module reads and writes a fixed number of data items each time it is activated.
- Sample delays modeled by writing data items into FIFOs before simulation starts.
- Simulators and implementation tools can determine static schedule for system at compile-time, enabling high performance simulation and implementation.
- Commonly used in DSP systems, especially along with System C's fixed point types (sc_fixed<>, sc_fix).

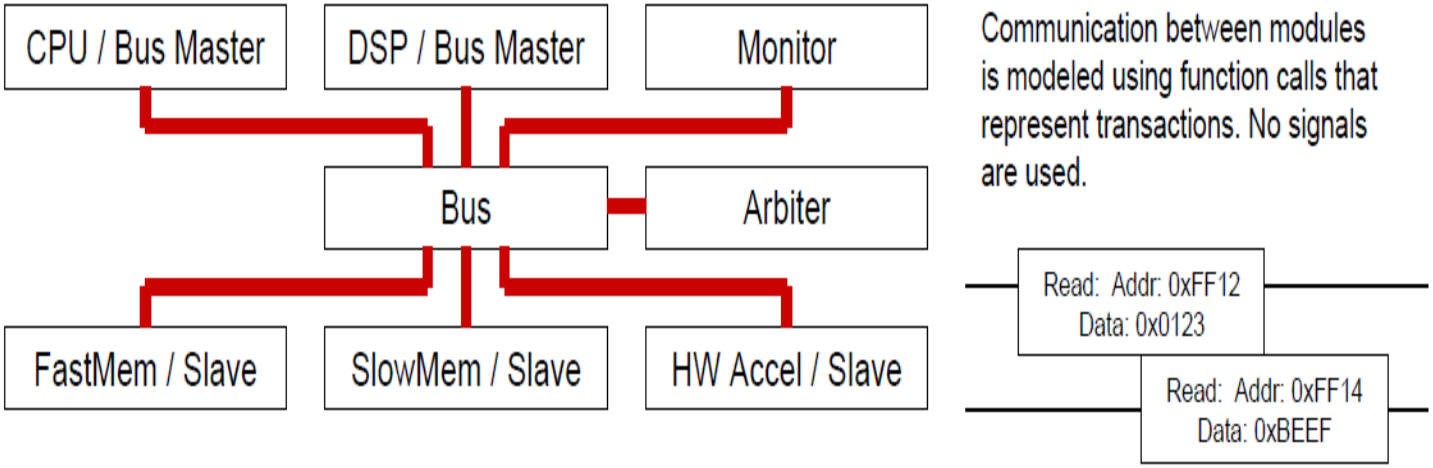


Transaction Level Modeling(TLM)

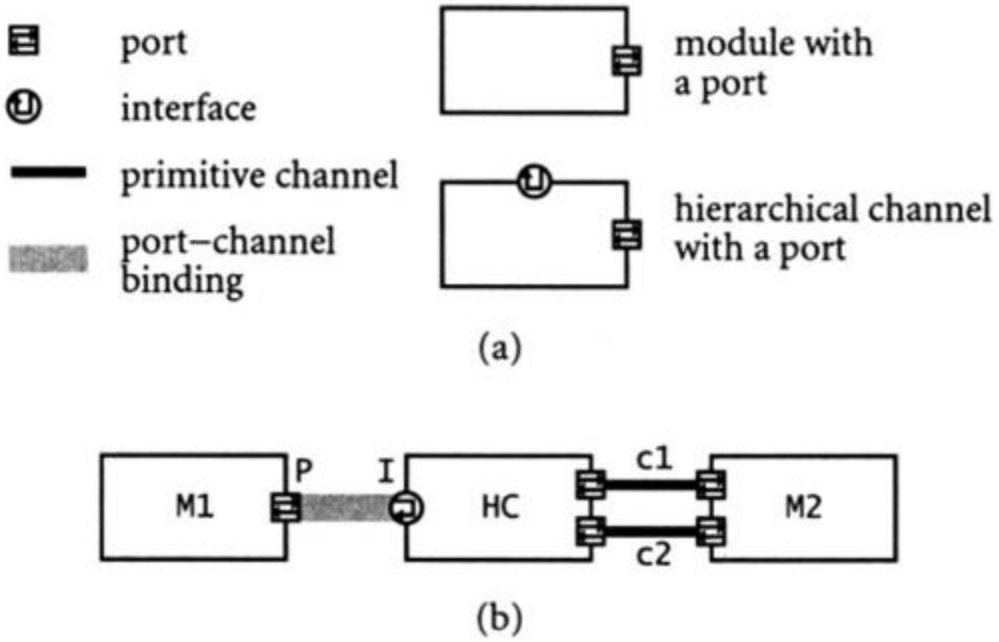
- Highlights the concept of separating communication from computation within a system.
- One specific type of discrete event MOC
- Details of communication across modules is isolated from details of functional units.
- TLM models communication between modules using function calls that represent the transactions.
- Transaction is data transfer (i.e.communication) or synchronization between two modules at an instant (i.e. SoC event)
- determined by the hardware/software system specificationCommunication handshakes are handled through blocking and non blocking IOs

Transaction-Level MOC in System C

- Communication & synchronization between modules modeled using function calls (rather than signals)
- Transactions have a start time, end time, and set of data attributes (e.g. burst_read(uint addr, char* data, uint n))
- Two-phase synchronization scheme typically used for overall system synchronization
- Much faster than RTL models (more later…)

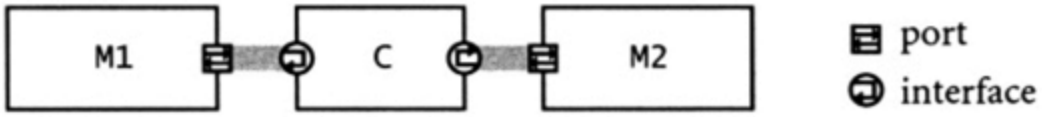


Graphical notations



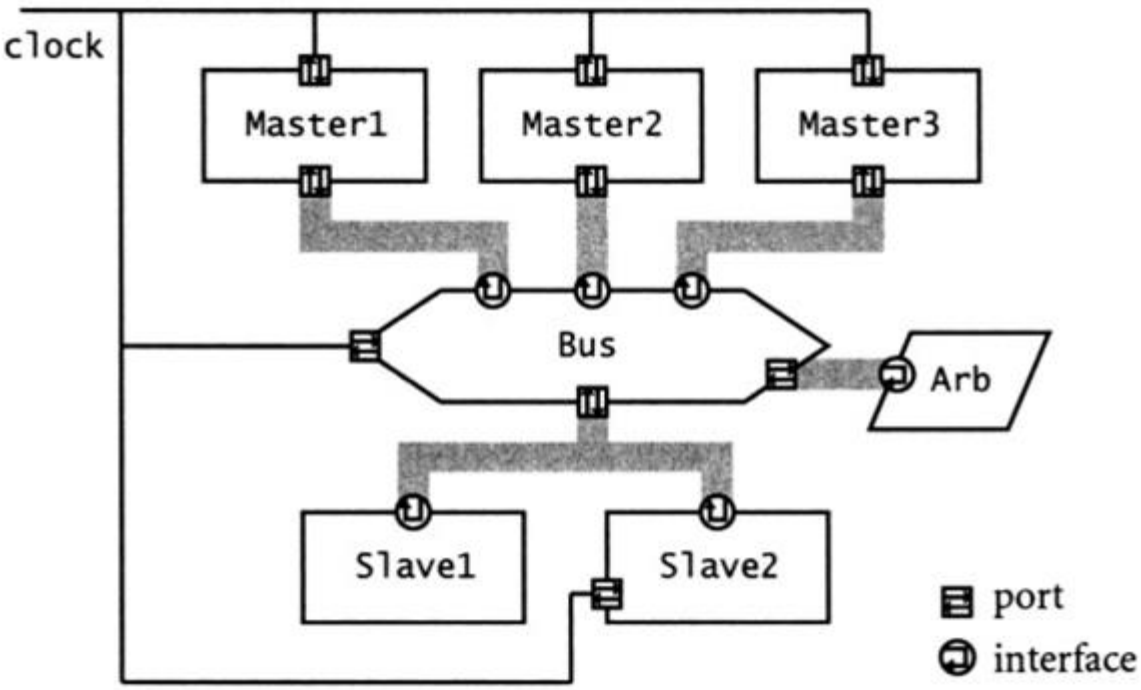
Graphical notations for modules, ports, channels and bindings

Notation

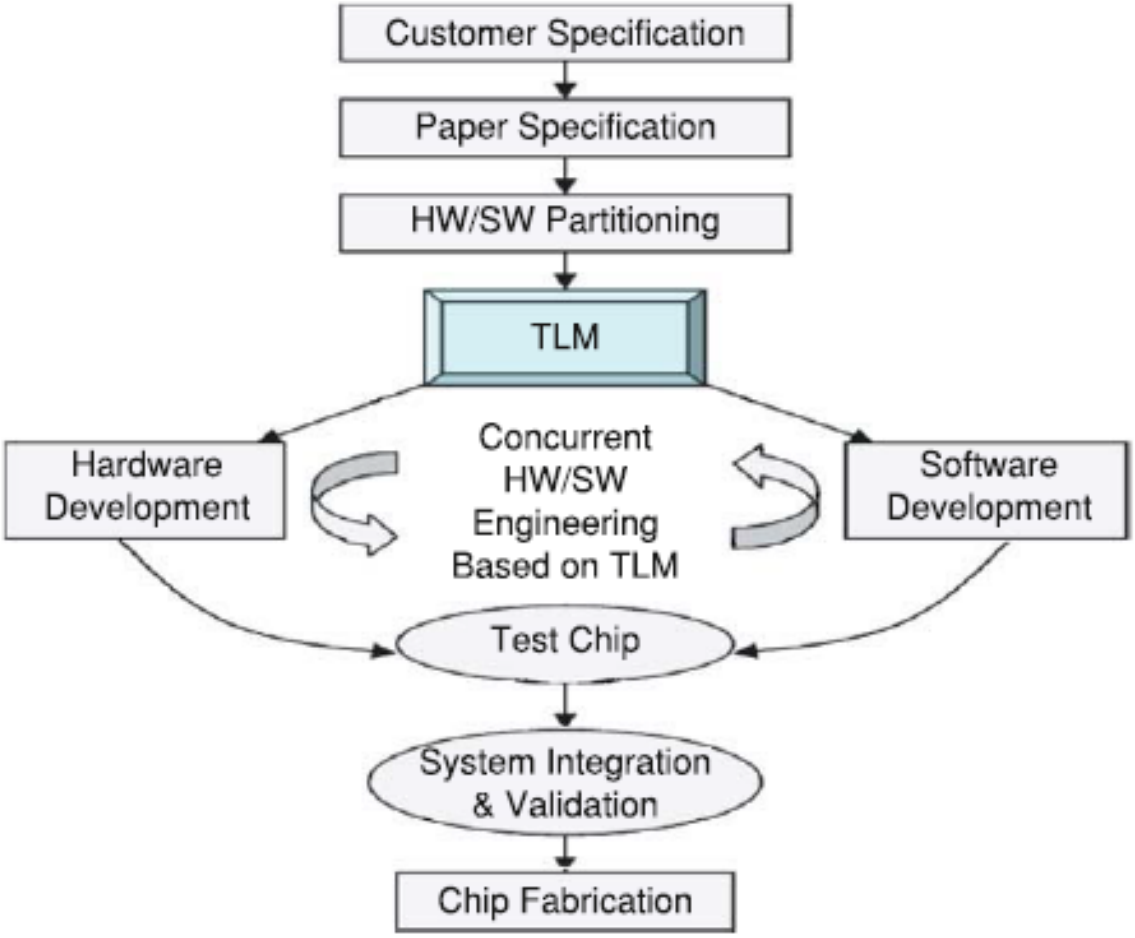


Modules M1 and M2 communicate over channel C

Simple bus design



SoC design flow using TLM



Current status

Some companies are using SystemC for RTL modeling, but this is not where the main interest is.

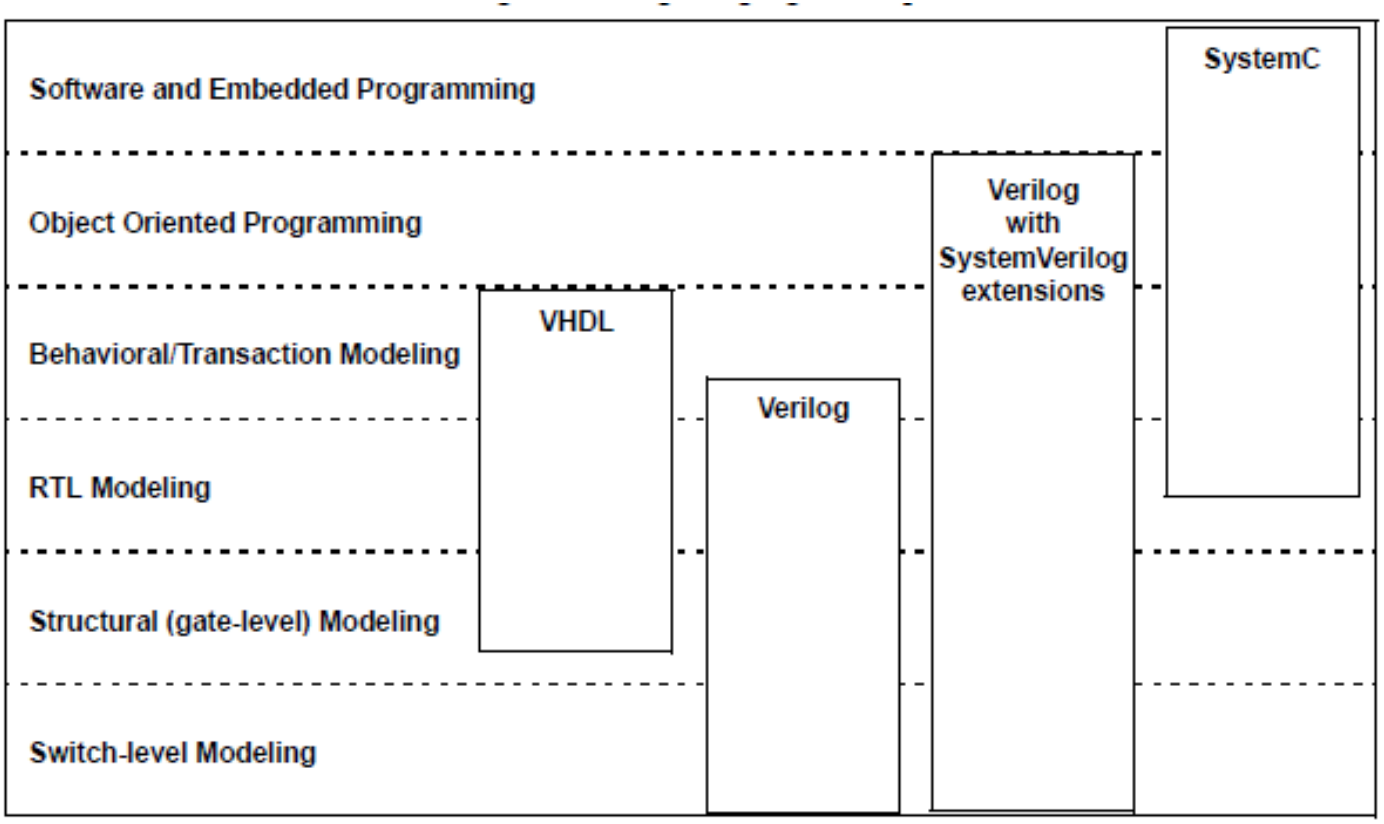
Many companies are in the process of replacing in-house C/C++ system level modeling environments with SystemC.

Many companies view SystemC as both a modeling language and a modeling “backplane” (e.g. for ISS integration).

A number of companies have completed TLM & TBV modeling efforts using SystemC 2.0 and are very enthusiastic. Some of the results are starting to be made publicly available.

Some companies are about to announce that they will provide system-level IP using SystemC.

Design language overlap

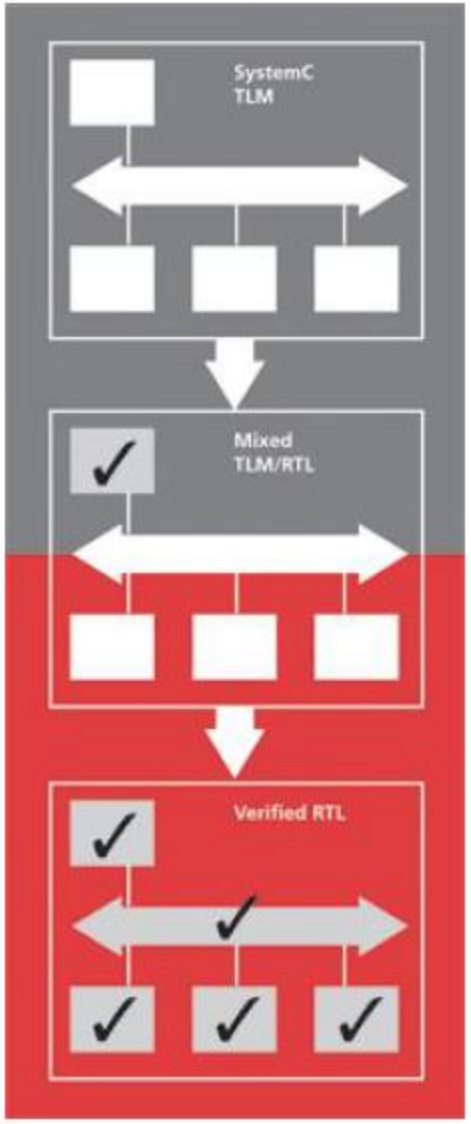


Complimentary features

	System C	System verilog
Core abstraction level	Events and messages	Logic states and transitions
Architectural design	System level hardware view and software programmer's view	Hardware implementation view;DPI link to C/C++/system C
Architectural verification and HW/SW co-verification	Cycle accurate TLM	Timing accurate RTL,TLM capability, C like extensions for algorthimetic descriptions
RTL to gate design	No gate level modeling	Logic synthesis
RTL to gates verification	TLM/RTL co-simulation	Implementation test bench including ABV and functional coverage

Co-simulation

In an emerging design and verification paradigm, design teams elaborate SystemC-based designs with SystemVerilog-based RTL as implementation proceeds. They intermingle SystemC and SystemVerilog to speed the co-simulation of hardware/software SoC designs.



Future Evolution of SystemC (cont'd)

Extensions as libraries on top of the core language

- Standardized channels for various MOC (e.g. static dataflow and Kahn process networks)
- Testbench development
  - Libraries to facilitate development of testbenches
    - + data structures that aid stimulus generation and response checking
    - + functions that help generate randomized stimulus, etc.
- System level modeling guidelines
  - library code that helps users create models following the guidelines
- Interfacing to other simulators
  - Standard APIs for interfacing SystemC with other simulators, emulators, etc.

Summary

- Mapping between model characteristics to language constructs
- PSM well suited for embedded systems
- Language constructs should be synthesizable
- Support for Verification by simulation

References

1. The Open SystemC Initiative http://www.systemc.org.
2. IEEE Standard SystemC® Language Reference manual http://standards.ieee.org/getieee/1666/download/1666-2005.pdf“
3. A system-C primer” by J Bhasker, Cadence Design systems.
4. Link to resources from systemc initiative. http://www.systemc.org/community/resources/
5. System design with system-c by Throsten Grotker ,Kluwer academic publishers.

(Some resources from above are posted on taxila as study material)