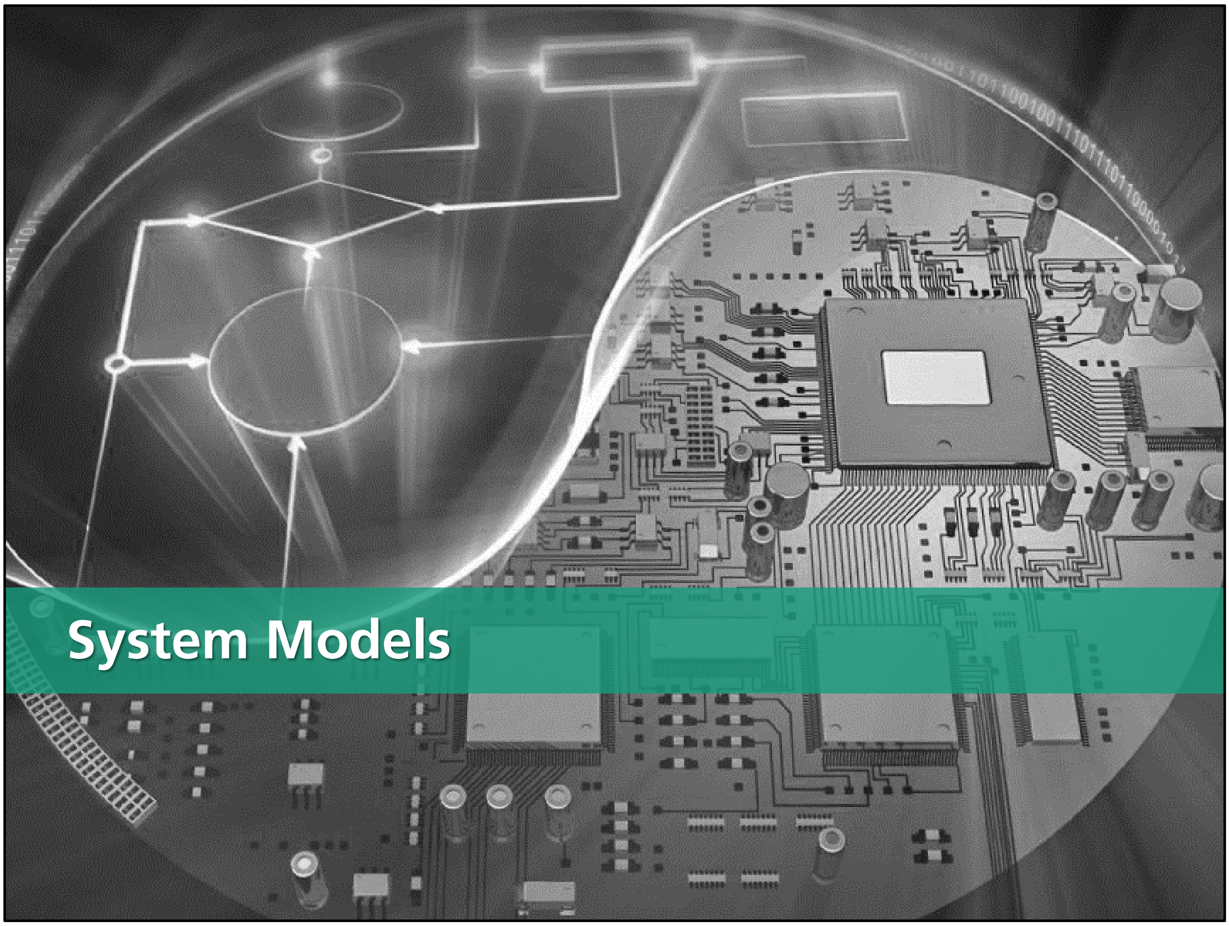
SystemC and Virtual Prototyping

Dr. Matthias Jung, Fraunhofer Institute IESE
matthias.jung@iese.fraunhofer.de

Fraunhofer
IESE

Your notes:



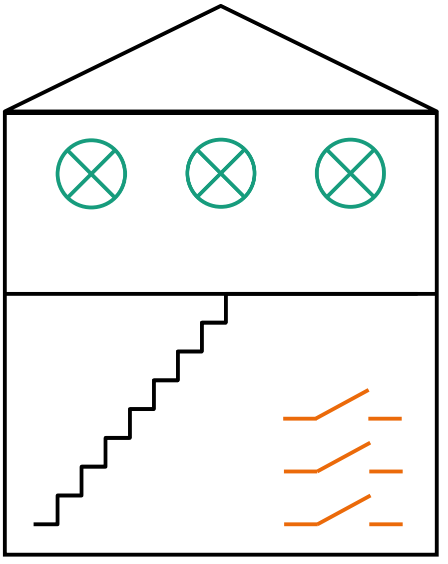
Your notes:

Test:

Models …

… enable … thinking

… hamper … perception



Fraunhofer IESE

Your notes:

Perception vs. Reality

What we perceive and how we interpret it depend on the frame through which we view the world around us.

Fraunhofer IESE

Mental models help people make sense of the world — to interpret their environment and understand themselves. Mental models include categories, concepts, identities, prototypes, stereotypes, causal narratives, and worldviews. Individuals do not respond to objective experience but to their mental representations of experience. In constructing their mental representations, people use interpretive frames provided by mental models. People may have access to multiple and conflicting mental models. Context can activate a particular mental model. Using a different mental model can change the individual's mental representation of the world around him.

Your notes:

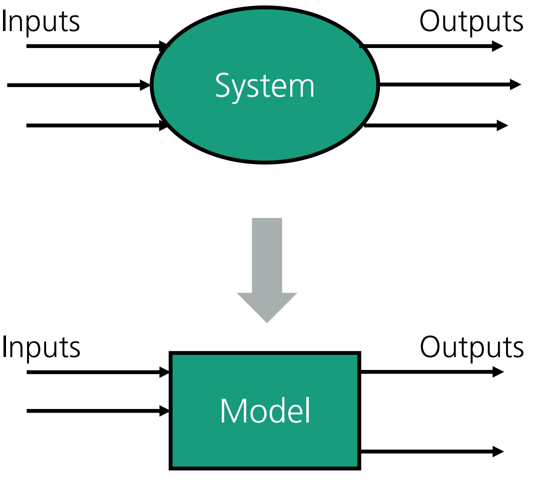
System and Model

- A **system** is a combination of components that act together to perform a function not possible with any of the individual parts

  Architecture describes how the system has to be implemented

- A **model** is a formal description of the system, which covers selected information.

  Describes how the system works



Fraunhofer IESE

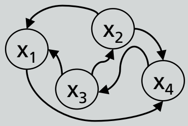
Your notes:

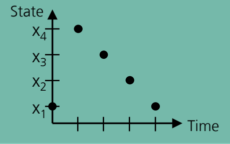
Time and States

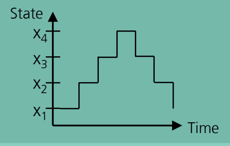
Discrete Time
Time values are countable (ℕ)

Continuous Time
Time values are real (ℝ)

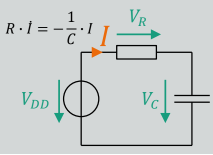
Discrete State
State is countable (ℕ)

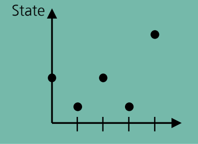


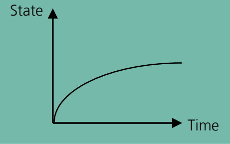




Continuous State
States are real (ℝ)



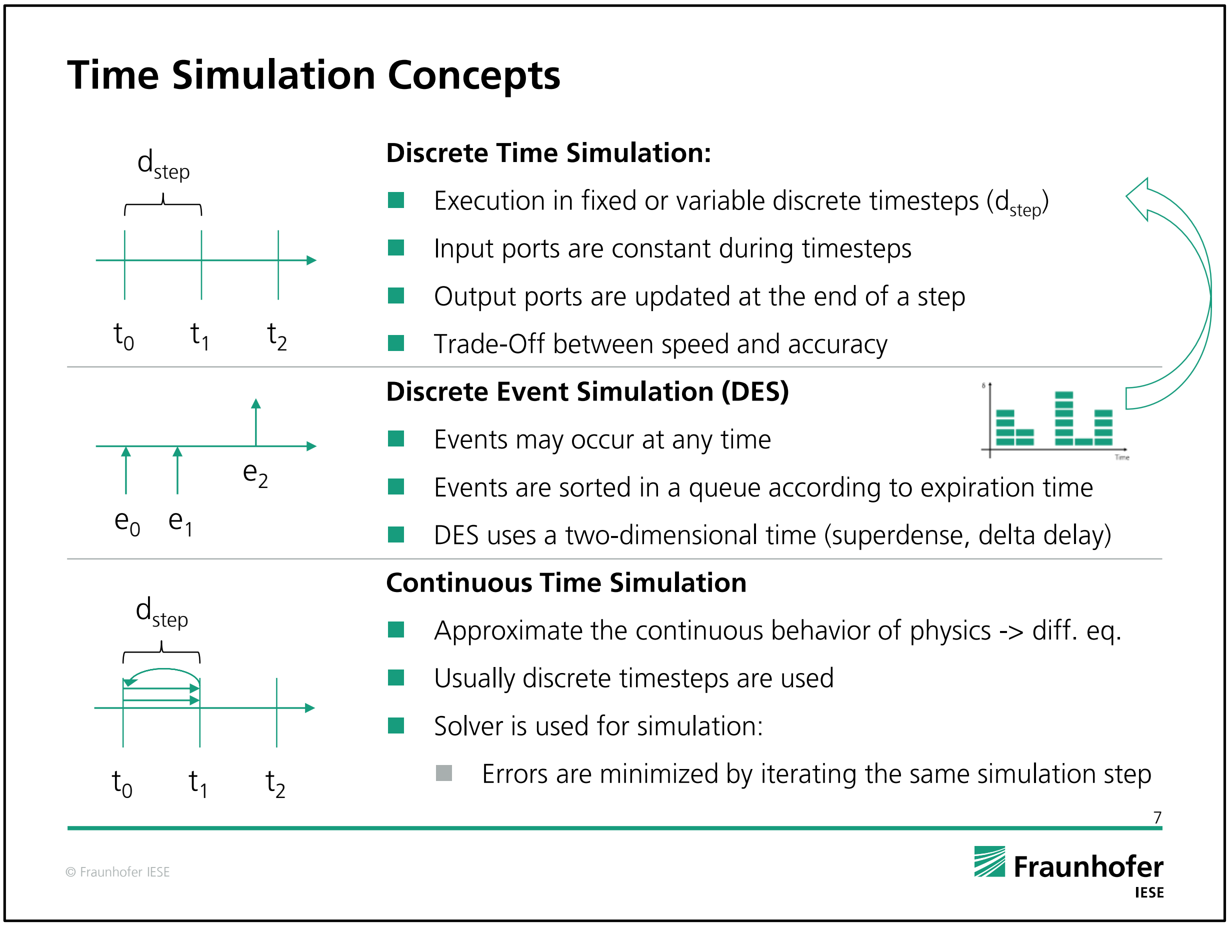




6

Fraunhofer IESE

Your notes:



- Discrete time (DT) MOCCs split the execution into discrete time steps of constant or variable size. They support for example the implementation of discre-tized physics models and control algorithms. Input ports under control of a DT MOCC store one value that may change between time steps, but that is kept constant during time steps of a simulation. Output ports communicate simula-tion results at the end of a time step. The MOCC executes controlled compo-nents in any order. The duration of a time step dstep reflects the granularity of the simulation; shorter time steps yield higher accuracy, but also require more frequent calculations and more computation time. Larger time steps require less calculations but yield a higher discretization error.
- Discrete event (DE) simulation models implement event based communication and the processing of simulation events. Events may occur at any time and are not bound to time steps. Their execution is ordered based on expiration times. This ensures that the simulation time in discrete simulation models continuously increases, but requires overhead for the necessary sorting of event. Simulation components receive events via their input ports. Discrete event simulations usually use a two-dimensional time.
- Continuous time (CT) simulation approximates the continuous behavior of re-al-world physics. Changes to individual elements have immediate impacts to dependent elements. For example, when a spring is extended, it immediately applies force to both ends. Resembling this behavior in a simulation is tricky, as necessary discretization of the simulation prior to solving yields a simulation error. Every CT simulation is controlled by a solver that evaluates one simula-tion component after another in discrete time steps. After simulating one component, it copies output values to the inputs of dependent components. Solvers control simulation errors by iterating the same simulation step until the simulation error falls below an acceptable threshold.

MOC Support in SystemC

- **Discrete Event** as used for:
  - RTL Hardware Modeling
  - State Machines
  - Network Modeling (e.g. stochastic or "waiting room" models)
  - Transaction Level Modeling
- Continous Time with AMS-Extension
- Kahn Process Networks
- Static Multi-rate Data-flow
- Dynamic Multi-rate Data-flow
- Communicating Sequential Processes
- Petri Nets
- …

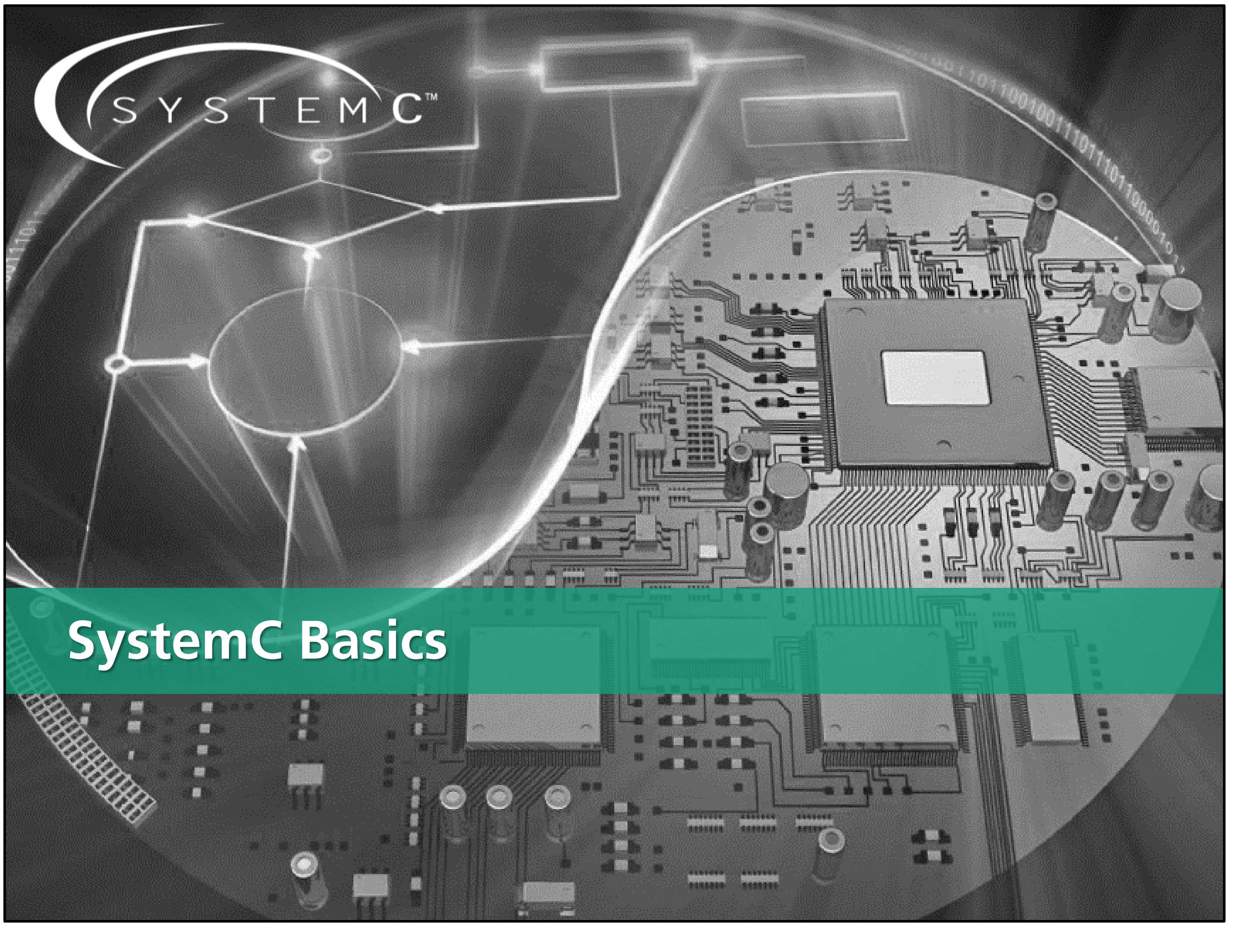
Wikipedia:

SystemC is a set of C++ classes and macros which provide an event-driven simulation interface (see also discrete event simulation). These facilities enable a designer to simulate concurrent processes, each described using plain C++ syntax.

8

Fraunhofer
IESE

Your notes:



Your notes:

Move to Virtuality?

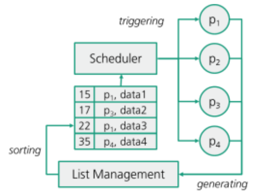
10

Fraunhofer
IESE

Your notes:

What is SystemC?

- Simulation and Modeling ~~Language~~ Library for C++
  - Discrete Event Model
  - IEEE Standard 1666 language for system-level-design
  - For complex systems consisting of hardware and software
  - Hardware / software co-design and co-simulation
  - Extension of hardware description languages to higher abstraction levels i.e. different levels of accuracy.



- Provides:
  - Set of library routines and macros implemented in C++ (class library)
  - Modeling concurrency
  - Synchronization
  - Inter-process communication
  - Simulation Kernel (scheduler) included
  - Compiler for C++ is sufficient for simulating SystemC models → binary is generated for executable simulation model

Fraunhofer IESE

Your notes:

Install SystemC on your Private Machine

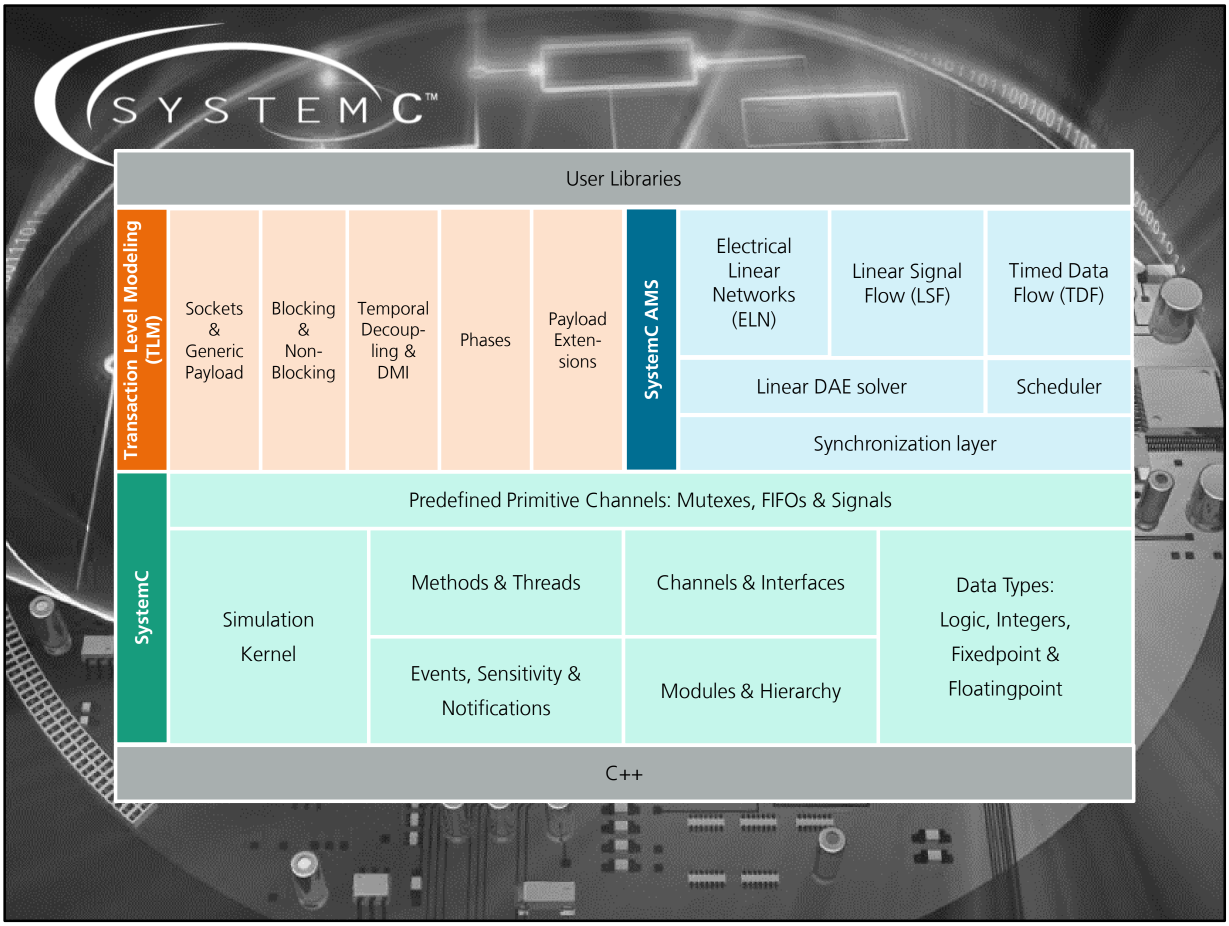
For Example on Ubuntu or Debian like Linux distributions

```
$ wget http://www.accellera.org/images/downloads/standards/systemc/systemc-2.3.1a.tar.gz
$ tar xfv systemc-2.3.1a.tar.gz
$ cd systemc-2.3.1a
$ ./configure --prefix=/opt/systemc/
$ make -j 4
$ sudo make install
```

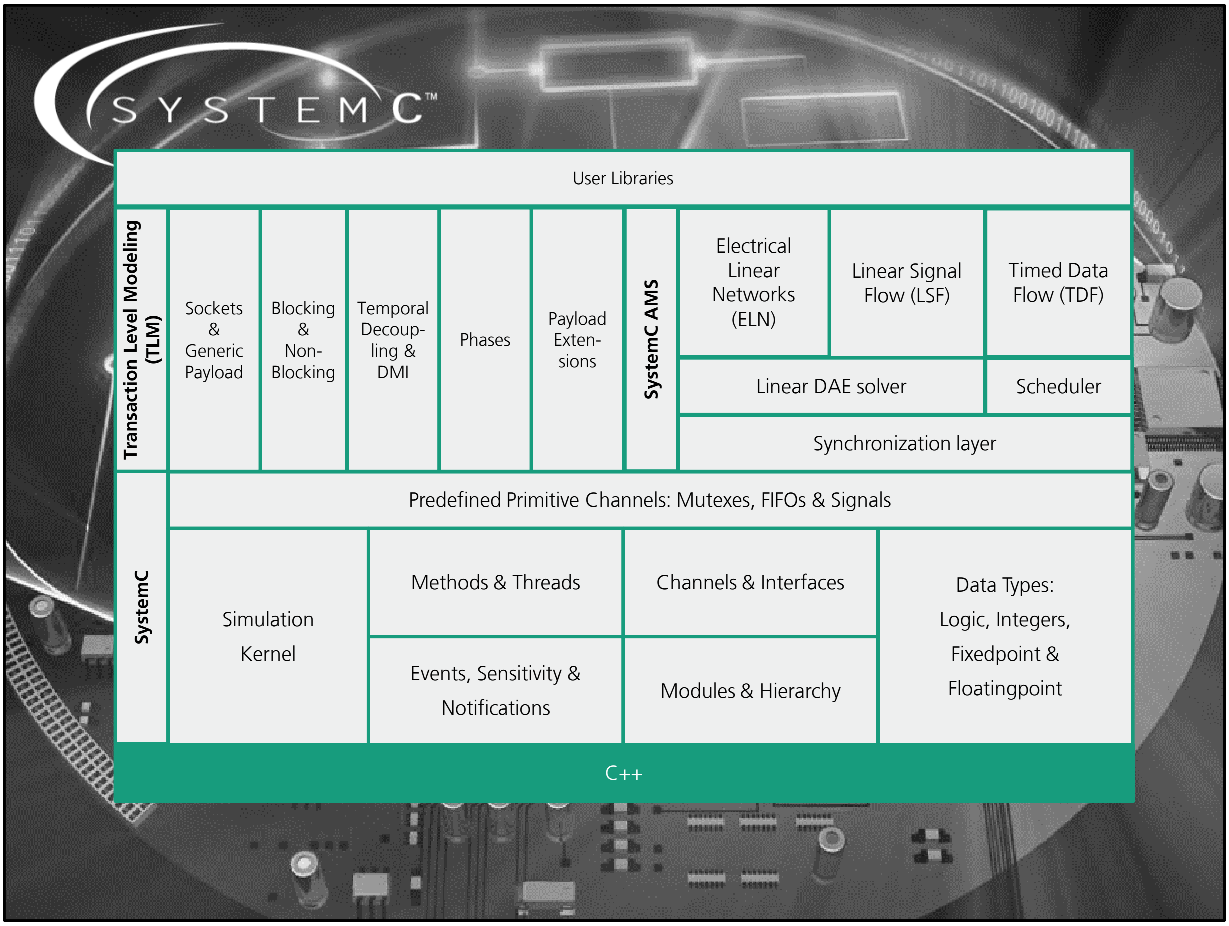
Get script on GitHub:
https://github.com/tukl-msd/SCVP.artifacts/blob/master/install_systemc.sh

Fraunhofer IESE

Your notes:


SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

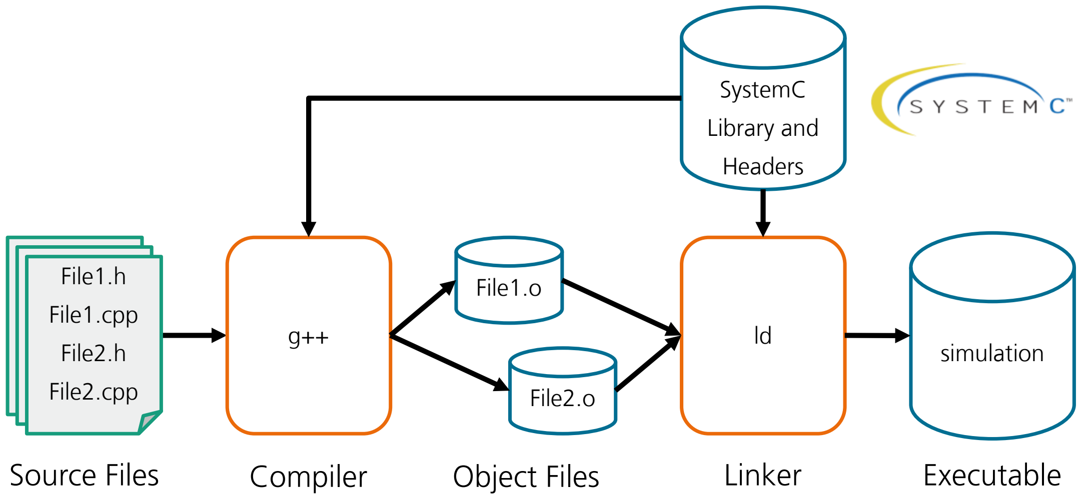
Your notes:


SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

Your notes:

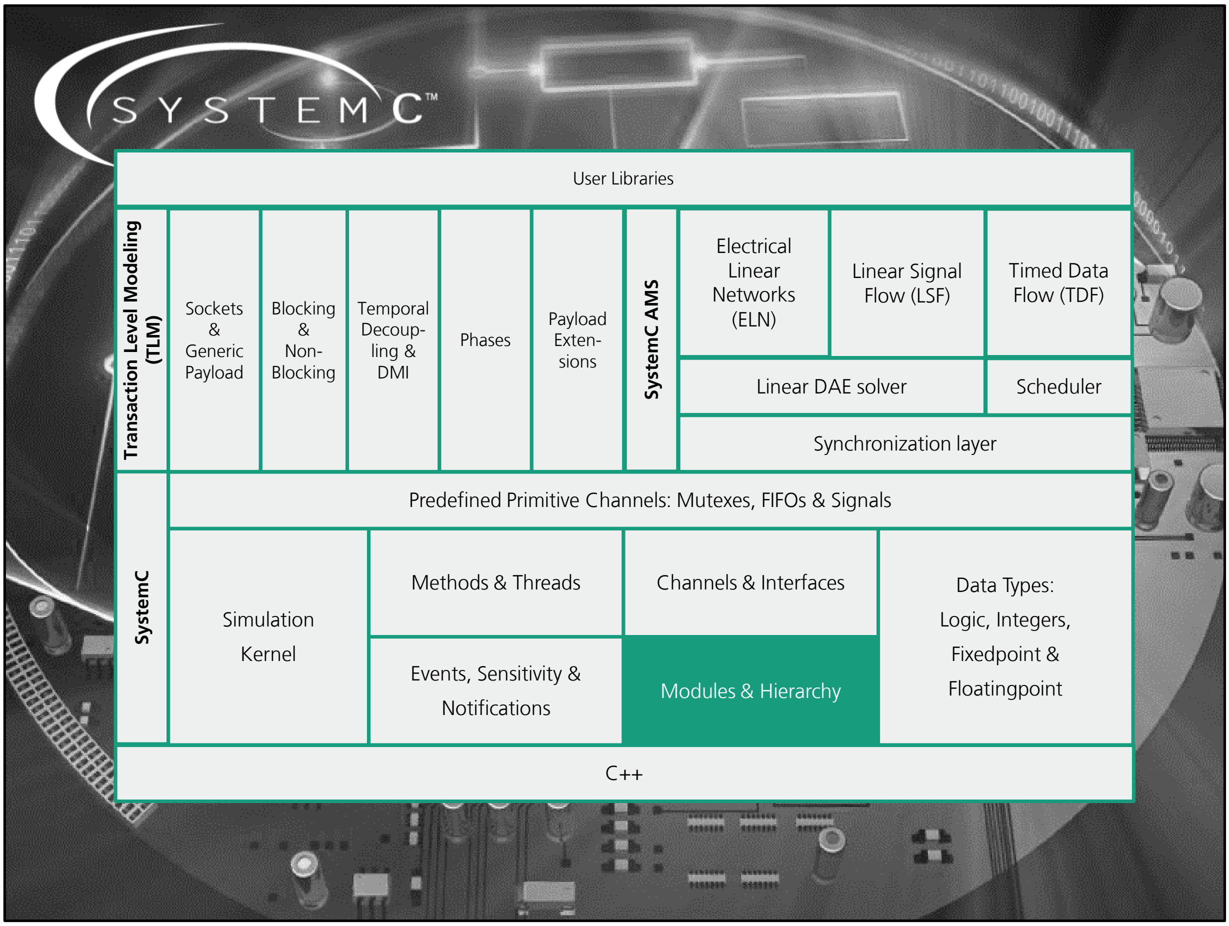
SystemC Compilation Flow

- SystemC is not a „language"!
- It's just a set of classes and macros in a a C++ library



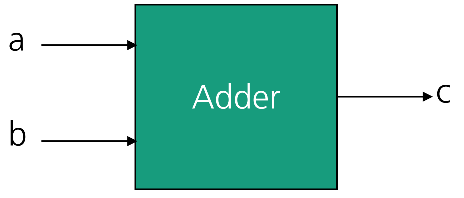
Fraunhofer IESE

Your notes:


SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

Your notes:

SystemC Basic Example



Remember an adder in VHDL:

```
entity adder is
Port (
        a: in unsigned;
        b: in unsigned;
        c: out unsigned
);
end adder;

architecture arch of adder is
    adding: process (a,b)
        c = a + b;
    end process adding;
end arch;
```


Fraunhofer IESE

Your notes:

SystemC Basic Example

```
SC_MODULE (adder)
{
    sc_in<int> a;
    sc_in<int> b;
    sc_out<int> c;

    void compute()
    {
        c.write(a.read() + b.read());
    }

    SC_CTOR (adder)
    {
        SC_METHOD (compute);
        sensitive << a << b;
    }
};
```

Module declaration

Define module input port named "**a**" with data type int

Implement functionality in member function **compute()**

Module constructor

Register function **compute()** at the SystemC scheduler as process

Tell the scheduler that **compute()** is sensitive to the input ports a and b

20

Fraunhofer IESE

Your notes:

SC_MODULE and SC_CTOR Macros

- `SC_MODULE(XYZ)` is a short macro for: `class XYZ : public sc_module`
- `SC_CTOR(XYZ)` is a short macro for:

```
SC_HASPROCESS(XYZ);
XYZ(const sc_module_name &name) : sc_module(name)
```

- `SC_HASPROCESS(XYZ)` is a short macro for:

```
typedef XYZ SC_CURRENT_USER_MODULE
```

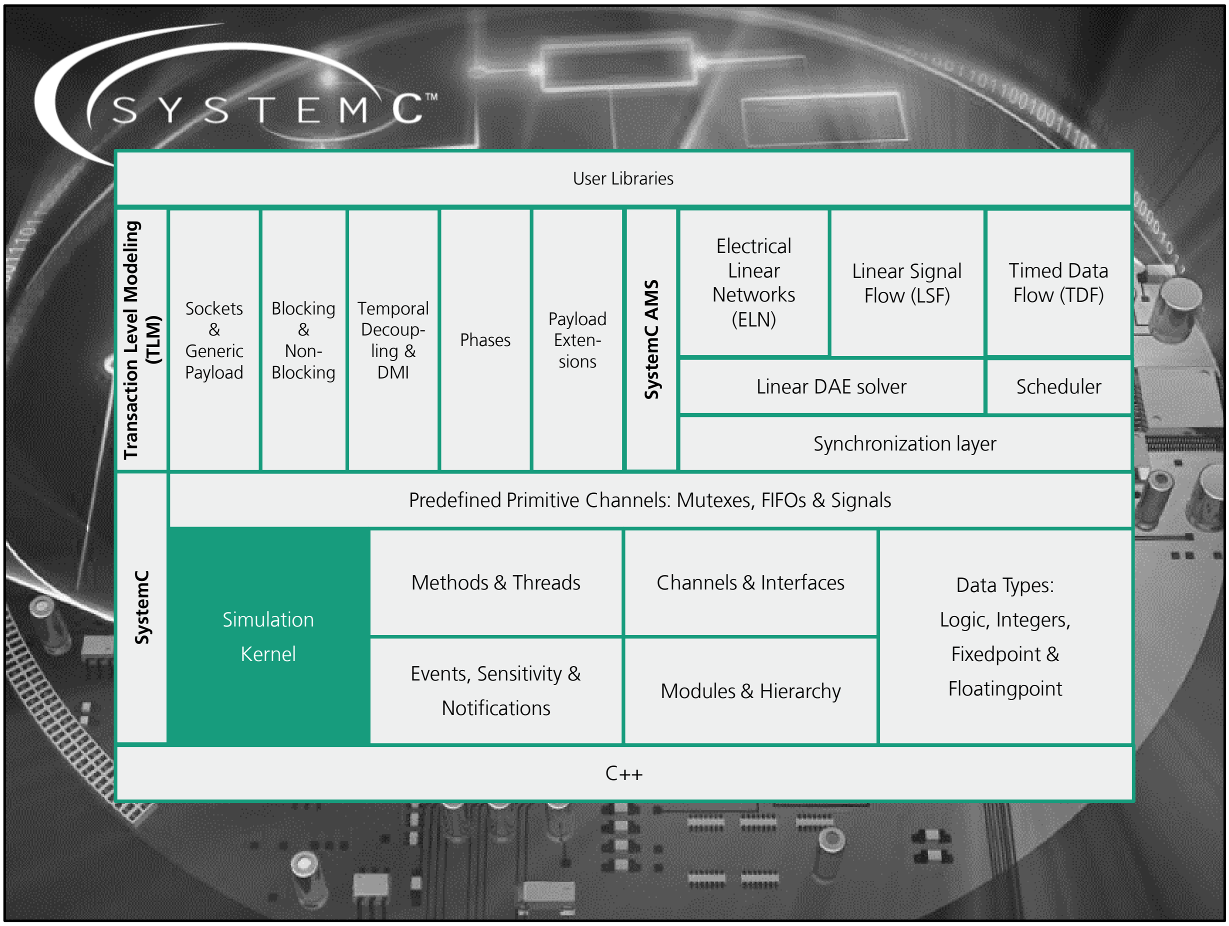
If you want to have constructor arguments for your SystemC module it is preferable not to use `SC_CTOR`, declare the normal constructor and use the `SC_HASPROCESS` instread.

- Not be confused with a process, its just that SystemC needs the class name for internal declarations for example in `SC_METHOD` or `SC_THREAD`. SystemC cannot know beforehand how you will call your module.
(What is typedef?: `typedef unsinged long ul;`)

21

Fraunhofer IESE

Your notes:

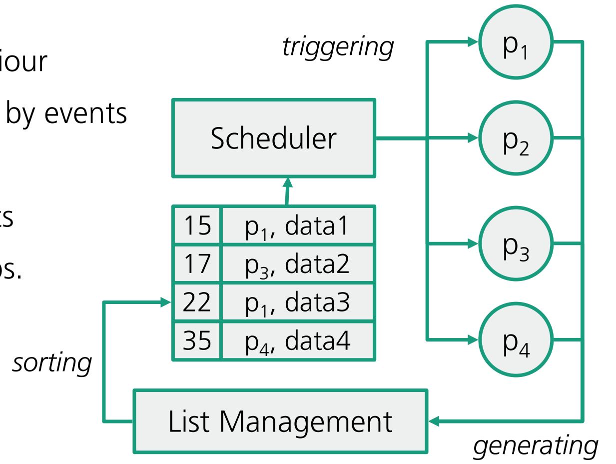

SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

Your notes:

Discrete Event Models (DEM) – General Concept

Evaluation of state changes only at occurrence of events!

- Process describes functional behaviour
- Execution of processes is triggered by events
- Processes are deterministic
- Processes may generate new events
- Events are sorted w.r.t. time stamps.

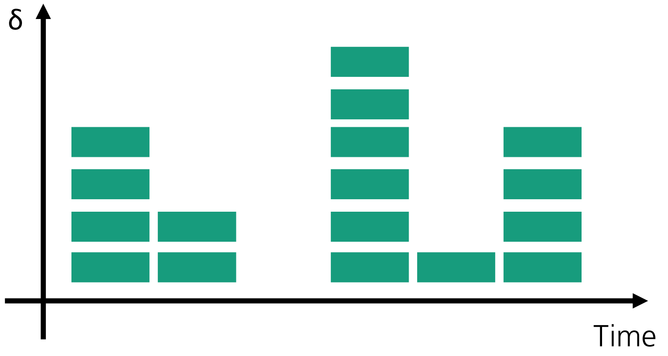


24

Fraunhofer IESE

Your notes:

The δ-Delay – A Concept of a Two-Dimensional Time



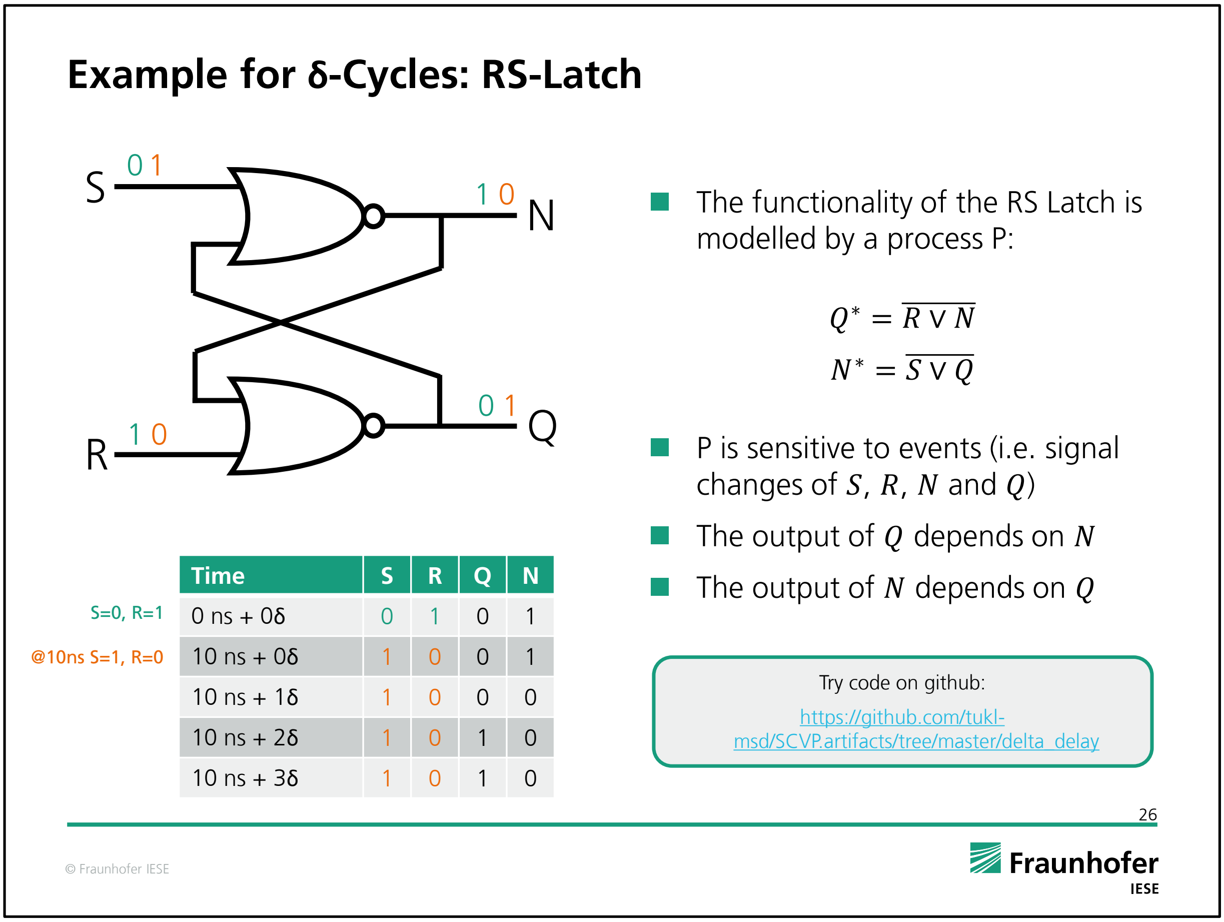
- The δ-Delay enables the simulation of concurrency in a sequential simulator
- The δ-Delay is an infinitesimally small abstract time unit
- The δ-Delay guarantees a deterministic signal assignment
- The δ-Delay is used, if a statement with 0 ns or SC_ZERO_TIME is called.

BUT not just in this case... more like, in this case as well

25

Fraunhofer IESE

Your notes:



The internal functionality of the RS latch is described by the following two statements:

$$Q^* = \overline{R \vee N}$$
$$N^* = \overline{S \vee Q}$$

In general, the signal on the left side (*) of the = depends on all the signals appearing on the right side. Therefore, the output of Q depends on N and the output of N depends on Q. If a signal depends on another signal, which has changed previously, then the expression in the signal assignment is re-evaluated in a so called δ-cycle. If the result of the evaluation is different than the current value of the signal, an event will be scheduled (added to the list of events to be processed) to update the signal with the new value and re-evaluate dependent equations. For example, if a change occurs on R or N, then the nor operator is evaluated, and if the result (Q^*) is different than the current value of Q, an event will be scheduled in order to update Q and re-evaluate the equation $N^* = \overline{S \vee Q}$. For the example in the slides: at simulation time 0ns the signals are set to $S = 0$, $R = 1$, $Q = 0$, and $N = 1$. At the time of 10 ns two values change to $S = 1$ and $R = 0$. Since we have dependencies we have to evaluate the process again:

$$Q^* = \overline{R \vee N} = \overline{0 \vee 1} = \overline{1} = 0$$
$$\boldsymbol{N^* = \overline{S \vee Q} = \overline{1 \vee 0} = \overline{1} = 0}$$

Apparently, the value of N has changed from 1 to 0. Therefore, we have to re-evaluate again the process in a next round (δ-cycle), in particular the equation $Q^* = \overline{R \vee N}$ because it depends on N:

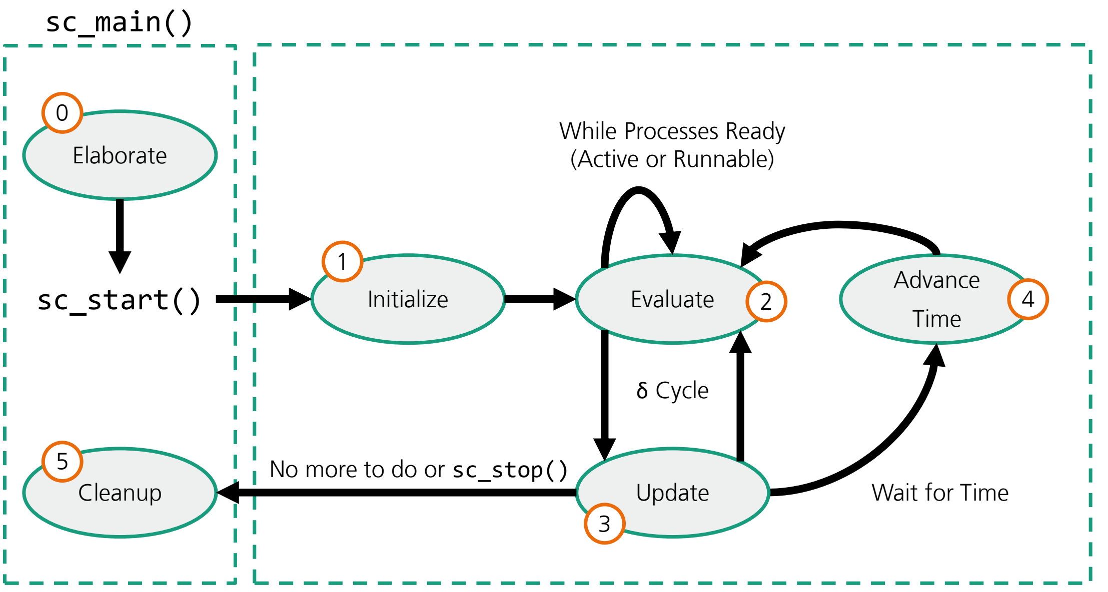
$$\boldsymbol{Q^* = \overline{R \vee N} = \overline{0 \vee 0} = \overline{0} = 1}$$
$$N^* = \overline{S \vee Q} = \overline{1 \vee 0} = \overline{1} = 0$$

Now, the value of Q has changed from 0 to 1. Therefore, we have to re-evaluate again the process (δ-cycle):

$$Q^* = \overline{R \vee N} = \overline{0 \vee 0} = \overline{0} = 1$$
$$N^* = \overline{S \vee Q} = \overline{1 \vee 1} = \overline{1} = 0$$

Now, there is no value change and therefore we are finished with δ-cycling. All the signals have reached their stable values.

SystemC Simulation Kernel



Fraunhofer IESE

Your notes:

SystemC Simulation Kernel

0 ***Elaborate***: Execution of all states prior to the `sc_start()` call are known as the elaboration phase. All constructors of all `SC_MODULE`s are called, the connections (bindings) between the different modules is checked. If for example a port is not bound the simulation will complain here in the beginning.

1 ***Initialize:*** During Initialization, each process is executed once (for `SC_METHOD`) or until a synchronization point (i.e. `wait()`) is reached (for `SC_THREAD`). In some circumstances it may not be desired for all processes to be executed in this phase. To turn off initialization for a process, we may call `dont_initialize()` after its `SC_METHOD` or `SC_THREAD` declaration inside the constructor. The order in which these processes are executed is unspecified, however, it is deterministic (for every simulation run with the same SystemC version it will behave the same way).

Fraunhofer IESE

Your notes:

SystemC Simulation Kernel

2 ***Evaluate***: From the set of processes marked as executable, all processes are executed successively and in an undefined order, and the marking is removed. An `SC_METHOD` is executed until the return, an `SC_THREAD` is suspended by calling a `wait(…)` statement. A process can not be interrupted during execution. By writing to `sc_signals` or `sc_fifos` etc., so-called update requests will be created in this phase for assignments to be made in the update phase 3. These update requests are noted by the scheduler. Furthermore, the execution of a `wait(…)` may result in a "timeout". This means that this process should be continued at a later time and they are stored in the event queue.

```
mySignal.write(true);
```

```
template< class T, sc_writer_policy POL > inline void
sc_signal<T,POL>::write( const T& value_ ) {
    bool value_changed = !( m_cur_val == value_ );
    [...]
    m_new_val = value_;
    if( value_changed ) {
        request_update();
    }
}
```

A look into the SystemC Kernel

Fraunhofer IESE

Your notes:

SystemC Simulation Kernel

3 ***Update***: In this phase, the previously requested updates are performed. The scheduler estimates if processes are sensitive to updates of these signals and mark them as executable. Then the scheduler goes again to the evaluation phase 2 (This looping is called a δ Cycle). If there are no new processes marked for execution we proceed to 4

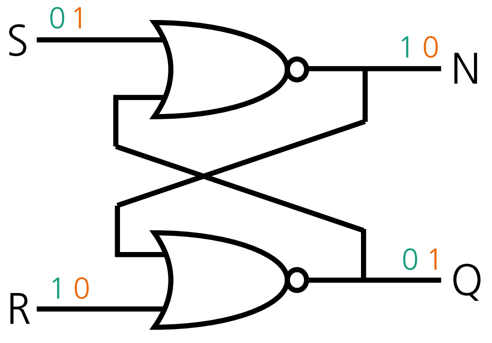
4 ***Advance Time***: Processes sensitive to events in the event queue with the smallest time are marked for execution and the scheduler proceeds to the evaluation phase 2 and thus, the simulation time is advanced. If there are no events in the event queue the simulation is finished. Then the scheduler proceeds to the cleanup phase 5 where all destructors are called.

- Note that calling `sc_stop()` in a process will directly lead to phase 5.

Fraunhofer
IESE

Your notes:

Remember: Example for Delta Delay: RS-Latch



In SystemC Code:

```
SC_MODULE(rslatch)
{
    sc_in<bool> S;
    sc_in<bool> R;
    sc_out<bool> Q;
    sc_out<bool> N;

    SC_CTOR(rslatch) : S("S"), R("R"), Q("Q"), N("N")
    {
        SC_METHOD(process);
        sensitive << S << R << Q << N;
    }

    void process()
    {
        Q.write(!(R.read()||N.read())); // NOR Gate
        N.write(!(S.read()||Q.read())); // NOR Gate
    }
};
```

	Time	S	R	Q	N
S=0, R=1	0 ns + 0δ	0	1	0	1
@10ns S=1, R=0	10 ns + 0δ	1	0	0	1
	10 ns + 1δ	1	0	0	0
	10 ns + 2δ	1	0	1	0
	10 ns + 3δ	1	0	1	0

Try code on github:
https://github.com/tukl-msd/SCVP.artifacts/tree/master/delta_delay

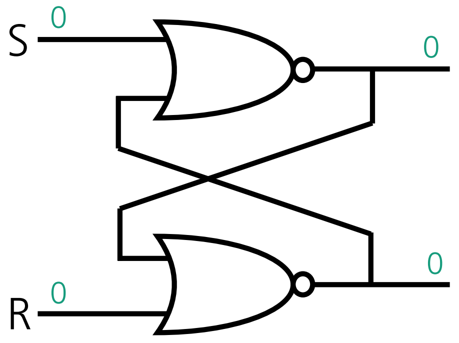
31

Fraunhofer IESE

See explanation on System Model Chapter …

Your notes:

Problem: Feedback Loops



Time	S	R	Q	N
0 ns + 0δ	0	0	0	0
0 ns + 0δ	0	0	1	1
0 ns + 1δ	0	0	0	0
0 ns + 2δ	0	0	1	1
0 ns + 3δ	0	0	0	0
…	…	…	…	…
0 ns + ∞δ	0	0	?	?

- In some rare occasions circuit can oscillate
- Infinite loop of δ-cycles – i.e. waiting forever
- Simulation time will never advance

Try code on github:
https://github.com/tukl-msd/SCVP.artifacts/tree/master/feedback_loop

32

Fraunhofer IESE

Your notes:

Order of Execution

Using Normal Variables:

```
void process() // int E=5 F=6
{
    E = F;
    F = E;
}
```

- Result is E = 6 and F = 6
- Swapping is not possible without a temporary variable

Using sc_signals etc.:

```
void process() // sc_signal<int> C=3 D=4
{
    C = D;
    D = C;
}
```

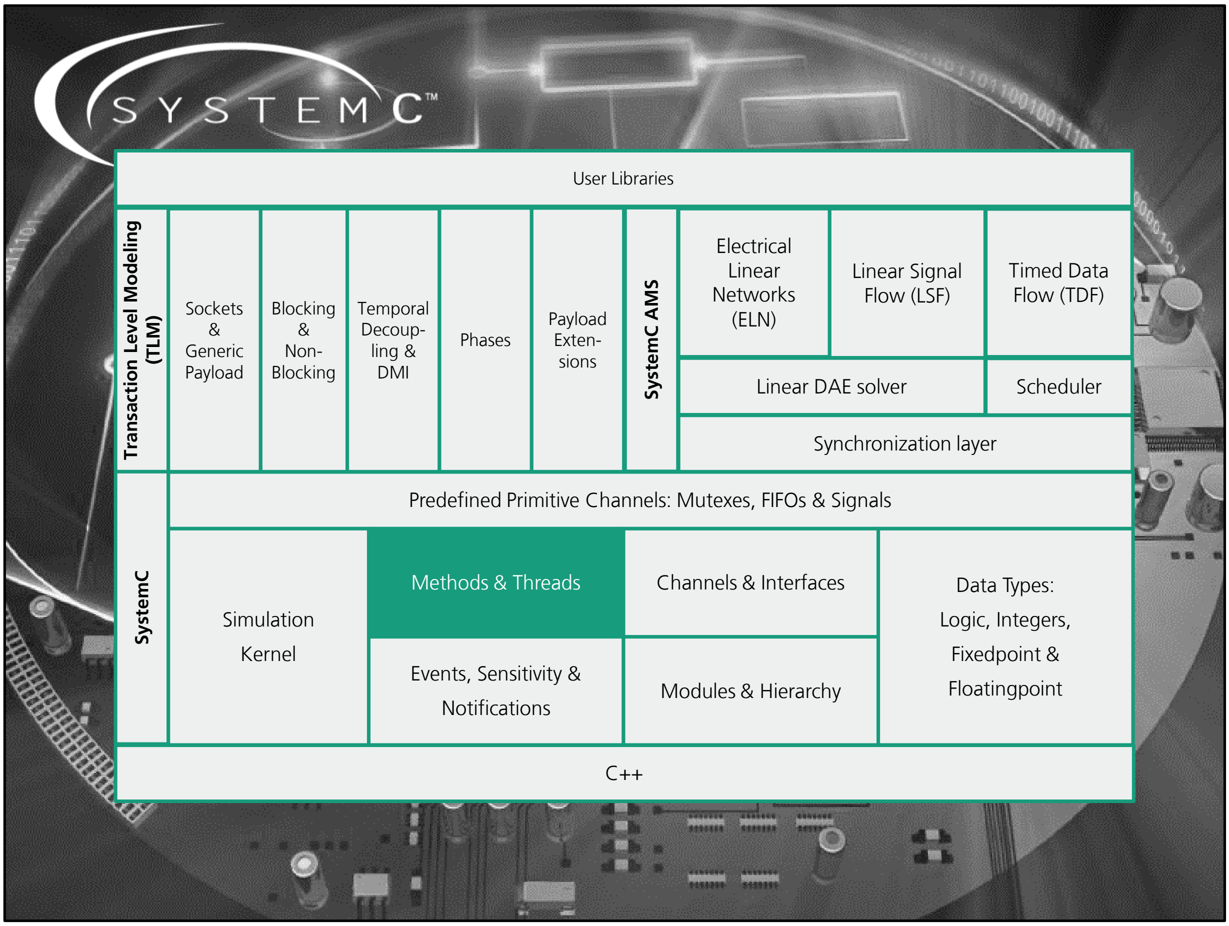
- Result is C = 4 and D = 3
- "Concurrent" execution of the statements

Try this as code on GitHub:
https://github.com/tukl-msd/SCVP.artifacts/tree/master/swapping_example

33

Fraunhofer IESE

Your notes:


SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

Your notes:

Methods and Threads

In SystemC there exist two ways of representing processes, called:

Methods (SC_METHOD)

- Similar to Verilog's always and VHDL's process
- Atomic execution of method, with no preemption – i.e. complete scope is executed {}

  there is no way out
- Therefore, infinite loops must be avoided otherwise simulation will get stuck
- Methods are usually sensitive to signals and events in the sensitivity list
- Methods can be called as often as possible – e.g. a signal change may trigger process again (δ cycle)

Threads (SC_THREAD)

- Threads are only started once at the begin of the simulation – i.e. if end of the scope is reached the thread dies.
- Threads can be suspended using the `wait(…)` statement
- Infinite loops are allowed and even needed
- Threads have much more overhead because of context switches this is why Methods are recommended for RTL design and Threads for more abstract designs
- Threads are good for test benches and TLM

36

Fraunhofer IESE

Your notes:

SC_METHOD Example:

Example: the RS Latch

- A change on the S or R input triggers the method
- However, the method changes Q and N such that the method is again triggered in the next delta cycle
- This 'fakes' concurrency within the method

Try code on github:
https://github.com/tukl-msd/SCVP.artifacts/tree/master/delta_delay

```
SC_MODULE(rslatch)
{
    sc_in<bool> S;
    sc_in<bool> R;
    sc_out<bool> Q;
    sc_out<bool> N;

    SC_CTOR(rslatch) : S("S"), R("R"), Q("Q"), N("N")
    {
        SC_METHOD(process);
        sensitive << S << R << Q << N;
    }

    void process()
    {
        Q.write(!(R.read()||N.read())); // NOR Gate
        N.write(!(S.read()||Q.read())); // NOR Gate
    }
};
```

Each instance of an `sc_module` needs a name.

Your notes:

SC_THREAD Example:

Example: the RS Latch

- For `SC_THREAD`s it is important that they have loops and `wait` statements otherwise they die.
- `SC_THREAD`s can be suspended by `wait` statements, `SC_METHOD`s can not!

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/thread_example

```
#include "systemc.h"

SC_MODULE(rslatch) {
    sc_in<bool> S;
    sc_in<bool> R;
    sc_out<bool> Q;
    sc_out<bool> N;

    SC_CTOR(rslatch) : S("S"), R("R"), Q("Q"), N("N") {
        SC_THREAD(process);
        sensitive << S << R << Q << N;
    }

    void process() {
        while(true) {
            wait();
            Q.write(!(R.read()||N.read())); // Nor Gate
            N.write(!(S.read()||Q.read())); // Nor Gate
        }
    }
};
```

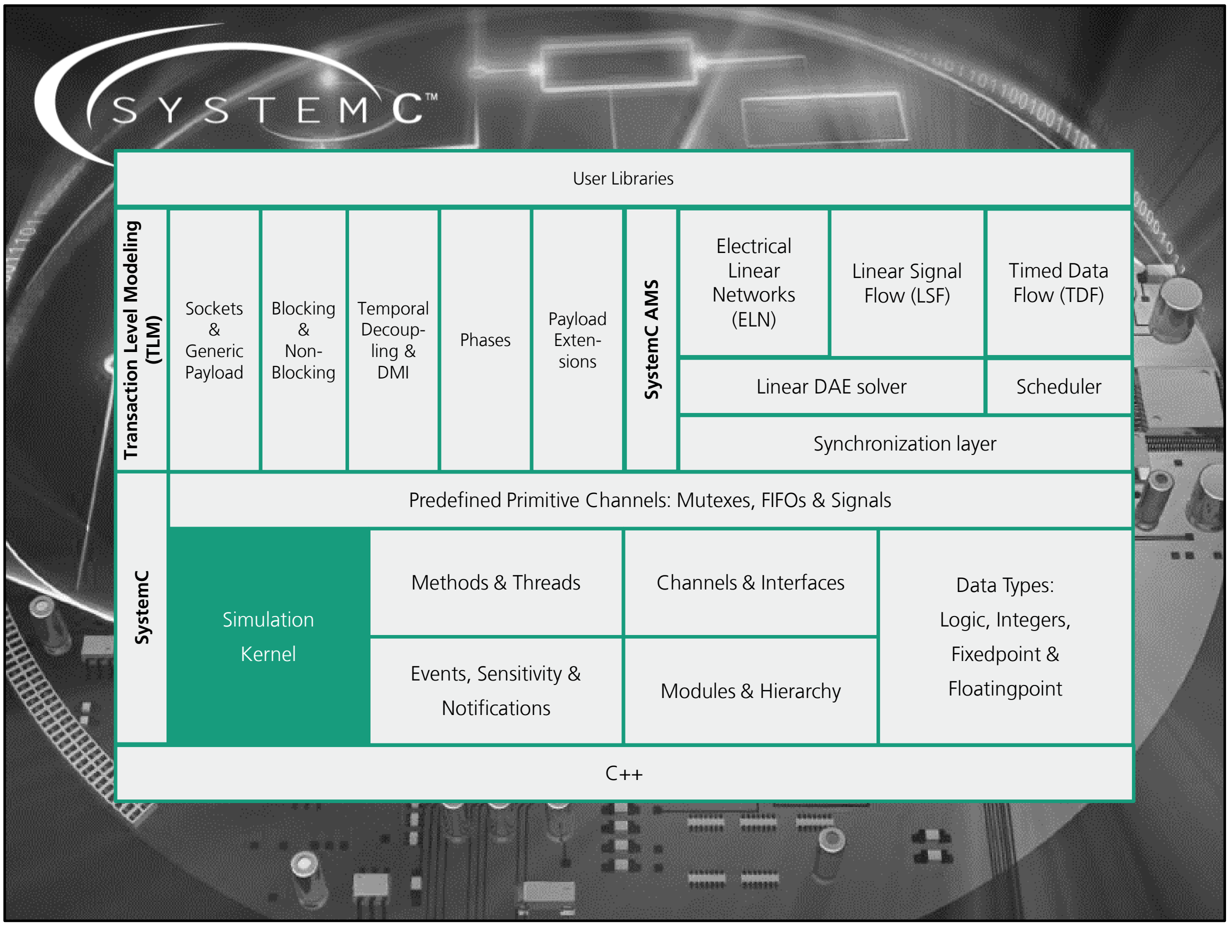
this is the part that differs in the implementation of SC_METHOD and SC_THREAD - the rest is much the same

38

the wait() statements can also be made conditional - if wait() statement is put in a SystemC method, the simulation will crash

Fraunhofer IESE

Your notes:


SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

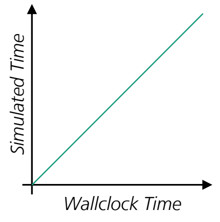
Your notes:

Notion of Time in Simulations

- ***Wall-Clock Time:*** the time from the start of execution to completion of the simulation for a human observer.
- ***Simulated Time:*** is the time being modeled by the simulation which may be less than or greater than the simulation's wall-clock time.

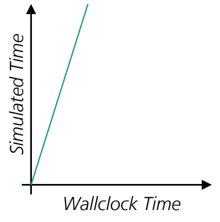
"Real-Time" (HiL)

WCT = ST

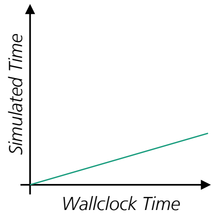


"As Fast as Possible" (vHiL)

WCT < ST



WCT > ST



Fraunhofer IESE

Your notes:

SystemC's Notion of Time

- ***Wall-Clock Time:*** the time from the start of execution to completion of the simulation for a human observer.
- ***Simulated Time:*** is the time being modeled by the simulation which may be less than or greater than the simulation's wall-clock time.

- SystemC tracks time with 64 bits of resolution using a class known as `sc_time`
- The global time is advanced within the kernel

42

Fraunhofer IESE

Your notes:

SystemC's Notion of Time

- `sc_time` is usually declared as: `sc_time name(double, sc_time_unit);`
- `sc_time` Provides all typical operands `+`, `-`, `*`, `/`, `==`, `!=`, `>`, `<`, …
- The time resolution can be set with by the function `sc_set_time_resolution(double, sc_time_unit)` (standard 1 PS)
- Special constant `SC_ZERO_TIME` (= `sc_time(0,SC_SEC)`)

```
sc_time name(1.5, SC_NS);
sc_time name2(name);
...
sc_start();
sc_start(name);
sc_start(sc_time(100,SC_US));
sc_stop();
...
sc_time name3 = sc_time_stamp();
...
```

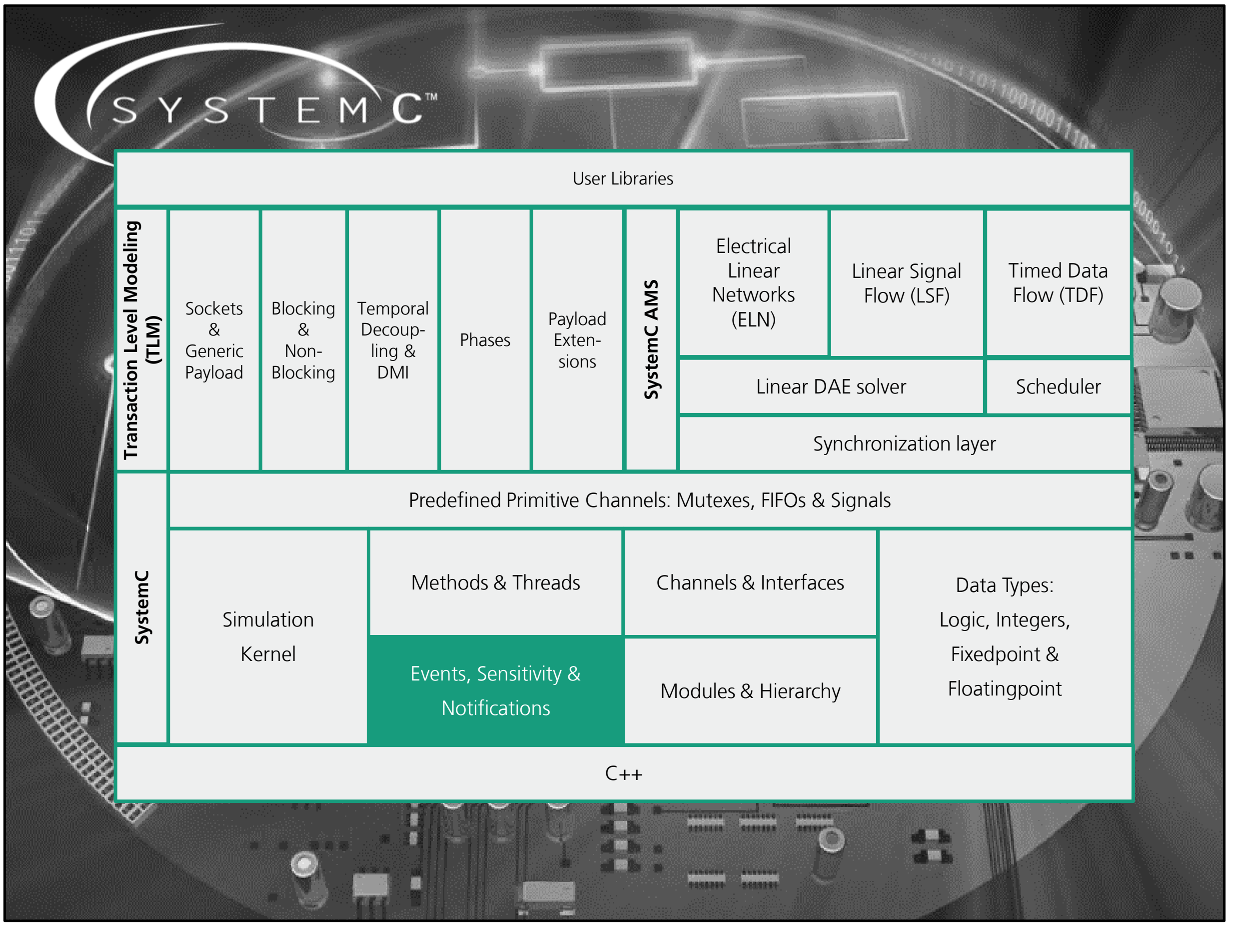
Simulation can run until there are no events, to a limited time, or unitl a call of `sc_stop()` in a process

The function `sc_time_stamp()` returns the current simulation time

enum	Units	Magnitude
SC_FS	Femtoseconds	10^{-15}
SC_PS	**Picoseconds**	$\mathbf{10^{-12}}$
SC_NS	Nanoseconds	10^{-9}
SC_US	Microseconds	10^{-6}
SC_MS	Milliseconds	10^{-3}
SC_SEC	Seconds	10^{0}

Fraunhofer IESE

Your notes:


SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

Your notes:

SystemC Events: `sc_event`

- Events are implemented with the `sc_event` class.
  - `sc_event myEvent;`
- Events are caused or fired through the event class member function `notify()`:
  - `myEvent.notify();`
    Avoid: events can be missed, non-determinism!
    Event is notified in the current evaluation phase
  - `myEvent.notify(SC_ZERO_TIME);`
  - `myEvent.notify(time);`
  - `myEvent.notify(10,SC_NS);`
  - `myEvent.cancel();`
- Only the first notification is noted

```
void triggerProcess() {
  wait(SC_ZERO_TIME);
  triggerEvent.notify(10,SC_NS);
  triggerEvent.notify(20,SC_NS); // Will be ignored
  triggerEvent.notify(30,SC_NS); // Will be ignored
}
```

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/sc_event_and_queue

Fraunhofer IESE

Your notes:

SystemC Events: sc_event_queue

```
SC_MODULE(eventQueueTester) {
    sc_event_queue triggerEventQueue;

    SC_CTOR(eventQueueTester) {
        SC_THREAD(triggerProcess);
        SC_METHOD(sensitiveProcess);
        sensitive << triggerEventQueue;
        dont_initialize();
    }

    void triggerProcess() {
       wait(100,SC_NS);
       triggerEventQueue.notify(10,SC_NS);
       triggerEventQueue.notify(20,SC_NS);
       triggerEventQueue.notify(40,SC_NS);
       triggerEventQueue.notify(30,SC_NS);
    }
    void sensitiveProcess() {
        cout << "@" << sc_time_stamp() << endl;
    }
};
```

- The class `sc_event_queue` notes all notifications
- Orders events w.r.t ascending time
- Provides also interface `sc_event_queue_if` for using as a port

Output:
@110ns
@120ns
@130ns
@140ns

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/sc_event_and_queue

47

Fraunhofer IESE

Your notes:

SystemC Events: Sensitivity

- Static Sensitivity (RTL Style):
  - Is Specified in the constructor of the model (elaboration) for both, `SC_METHOD`s and `SC_THREAD`s
  - `sensitive << mySignal << myClock.pos() << myAwesomeEvent;`
  - Static sensitivity cannot be changed!
- Dynamic Sensitivity (TLM Style):
  - Dynamic Sensitivity lets a simulation process change its sensitivity on the fly by calling different functions within the process.
    - `SC_THREAD` uses `wait(myAwesomeEvent);`
    - `SC_METHOD` uses `next_trigger(myAwesomeEvent);`
  - The static sensitivity is overwritten temporarily.

48

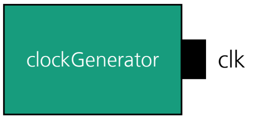
Your notes:

SystemC's Wait Statement

```
SC_MODULE(clockGenerator) {
    public:
    sc_out<bool> clk;
    bool value;
    sc_time period;

    SC_HAS_PROCESS(clockGenerator);
    clockGenerator(const sc_module_name &name, sc_time period) :
       sc_module(name), period(period), value(true)
    {
        SC_THREAD(generation);
    }
    void generation() {
        while(true) {
            value = !value;
            clk.write(value);
            wait(period/2);
        }
    }
};
```

```
wait();
wait(3);
wait(myEvent);
wait(sc_time(10,SC_NS));
wait(10,SC_NS);
wait(SC_ZERO_TIME);
```

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/clock_generator



- The wait function provides a syntax to allow to model delays within `SC_THREAD` processes.
- When a wait is invoked, the `SC_THREAD` process is suspended
- Waiting for integer e.g. 3 will wait 3 times
- Waiting for `SC_ZERO_TIME` will wait for one δ Cycle

49

Your notes:

SystemC Events: Sensitivity

Static Sensitivity		Dynamic Sensitivity	
THREAD	METHOD	THREAD	METHOD

Static Sensitivity – THREAD

```
SC_MODULE (Module)
{
  sc_in<int> a;

  void process() {
    while(true) {
      wait();
      // do something
    }
  }

  SC_CTOR (adder)
  {
    SC_THREAD (process);
    sensitive << a;
  }
};
```

Static Sensitivity – METHOD

```
SC_MODULE (Module)
{
  sc_in<int> a;

  void process()
  {
    // do something
  }

  SC_CTOR (adder)
  {
    SC_METHOD (process);
    sensitive << a;
  }
};
```

Dynamic Sensitivity – THREAD

```
SC_MODULE (Module)
{
  sc_in<int> a;

  void process() {
    while(true) {
      wait(
        a.valueChangedEvent
      );
      // do something
    }
  }

  SC_CTOR (adder)
  {
    SC_THREAD (process);
  }
};
```

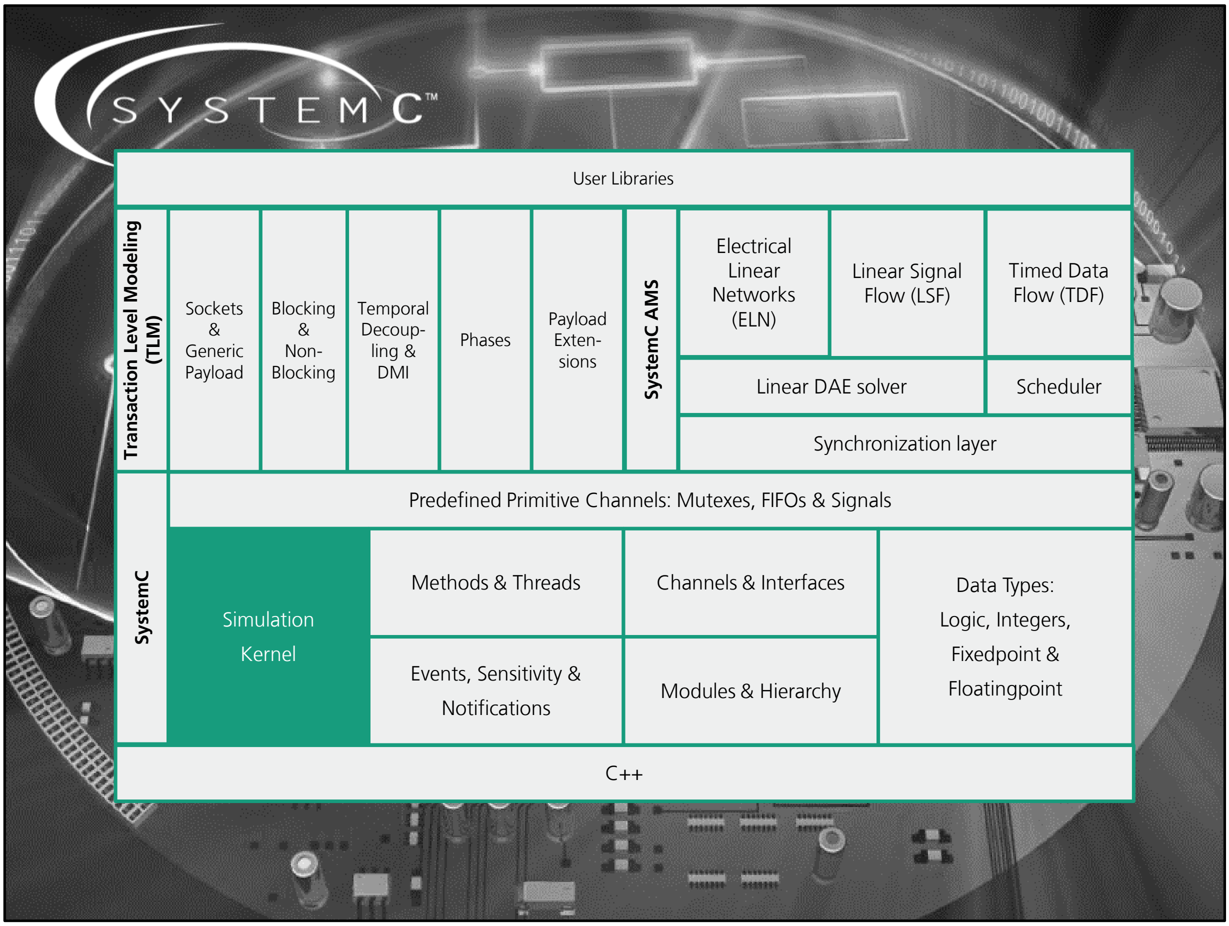
Dynamic Sensitivity – METHOD

```
SC_MODULE (Module)
{
  sc_in<int> a;

  void process(
  {
    // do something
    next_trigger(
      a.valueChangedEvent
    );
  }

  SC_CTOR (adder)
  {
    SC_METHOD (process);
  }
};
```

50

Fraunhofer IESE

Your notes:

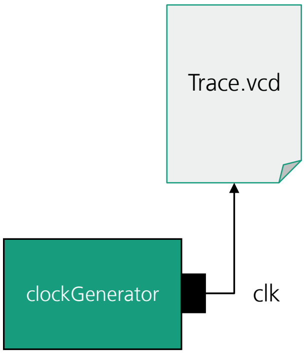

SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

Your notes:

SystemC Waveform Tracing

```
int sc_main ()
{
    clockGenerator g("clock_1GHz", sc_time(1,SC_NS));
    sc_signal<bool> clk;

    // Bind Signals
    g.clk.bind(clk);

    // Setup Waveform Tracing:
    sc_trace_file *wf = sc_create_vcd_trace_file("trace");
    sc_trace(wf, clk, "clk");

    // Start Simulation
    sc_start(10, SC_NS);

    // Close Trace File:
    sc_close_vcd_trace_file(wf);

    return 0;
}
```

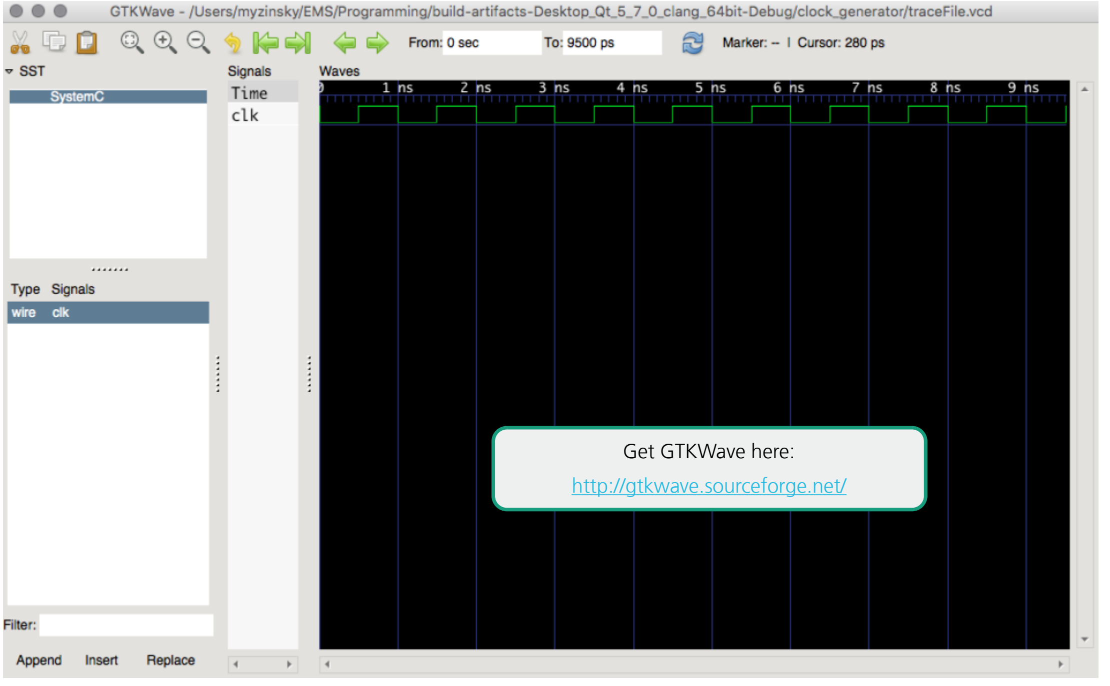
- Like VHDL or Verilog, SystemC allows the non-intrusive recording of siganls into a waveform vcd file
- cout is printed in every delta cycle -> confusing



Fraunhofer IESE

Your notes:

SystemC Waveform Tracing



54

Fraunhofer IESE

Your notes:

SystemC’s `sc_clock`

From SystemC Specification:

```
sc_clock(const char* name_,
        const sc_time& period_,
        double         duty_cycle_ = 0.5,
        const sc_time& start_time_ = SC_ZERO_TIME,
        bool           posedge_first_ = true );

sc_clock(const char* name_,
        double         period_v_,
        sc_time_unit   period_tu_,
        double         duty_cycle_ = 0.5 );

sc_clock(const char* name_,
        double         period_v_,
        sc_time_unit   period_tu_,
        double         duty_cycle_,
        double         start_time_v_,
        sc_time_unit   start_time_tu_,
        bool           posedge_first_ = true );
```

- For easy creation of clock generators
- Example:

```
sc_clock clock("Clk", 10, SC_NS, 0.5, 10, SC_NS);
sc_clock clock("Clk2", sc_time(10, SC_NS));
sc_clock clock("Clk3", 10, SC_NS, 0.5);
```

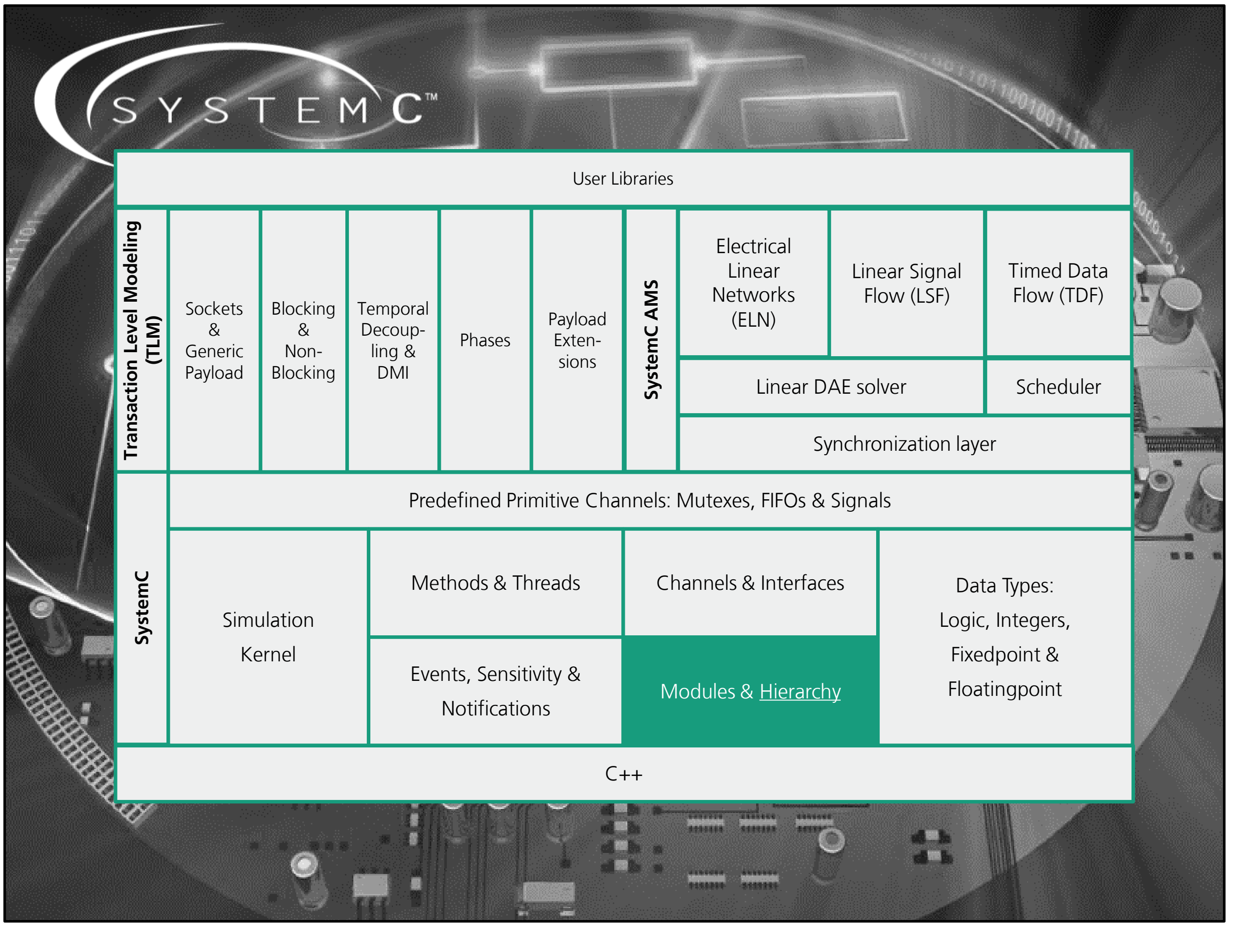
- Processes can be sensitive to clocks:

```
SC_METHOD(monitor);
sensitive << clk.pos();
```

55

Fraunhofer IESE

Your notes:

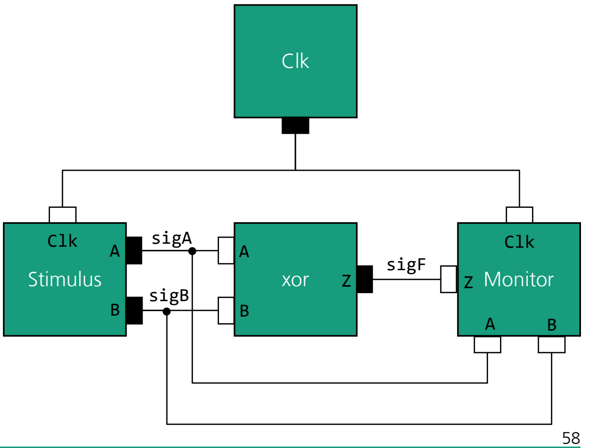

SYSTEMC™
User Libraries
Transaction Level Modeling (TLM)
Sockets & Generic Payload
Blocking & Non-Blocking
Temporal Decoup-ling & DMI
Phases
Payload Exten-sions
SystemC AMS
Electrical Linear Networks (ELN)
Linear Signal Flow (LSF)
Timed Data Flow (TDF)
Linear DAE solver
Scheduler
Synchronization layer
SystemC
Predefined Primitive Channels: Mutexes, FIFOs & Signals
Simulation Kernel
Methods & Threads
Channels & Interfaces
Data Types: Logic, Integers, Fixedpoint & Floatingpoint
Events, Sensitivity & Notifications
Modules & Hierarchy
C++

Your notes:

Connecting Modules (Binding)

```
int sc_main(int argc, char* argv[]) {
    sc_signal<bool> sigA, sigB, sigF;

    sc_clock clock("Clk", 10, SC_NS, 0.5);

    stim Stim1("Stimulus");
    Stim1.A.bind(sigA);
    Stim1.B.bind(sigB);
    Stim1.Clk.bind(clock);

    exor2 DUT("xor");
    DUT.A(sigA);
    DUT.B(sigB);
    DUT.Z(sigF);

    Monitor mon("Monitor");
    mon.A(sigA);
    mon.B(sigB);
    mon.Z(sigF);
    mon.Clk(clock);

    sc_start();  // run forever

    return 0;
}
```

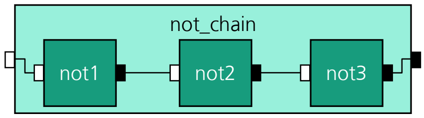
- Connecting `SC_MODULE`s in `sc_main` or in a toplevel module
- Binding of components with signals
- Keyword bind can be used or not



Fraunhofer IESE

Your notes:

Connecting Modules in Modules (Hierarchical Binding)




```
SC_MODULE(NOT)
{
  public:
  sc_in<bool> in;
  sc_out<bool> out;

  SC_CTOR(NOT) : in("in"), out("out")
  {
    SC_METHOD(process);
  }

  void process()
  {
    out.write(!in.read());
  }
};
```

```
SC_MODULE(not_chain) {
  sc_in<bool> A;
  sc_out<bool> Z;
  NOT not1, not2, not3;
  sc_signal<bool> h1,h2;

  SC_CTOR(not_chain):
  not1("not1"), not2("not2"),
  not3("not3"), A("A"), Z("Z"),
  h1("h1"), h2("h2")
  {
        not1.in.bind(A);
        not1.out.bind(h1);
        not2.in(h1);
        not2.out(h2);
        not3.in(h2);
        not3.out(Z);
    }
};
```

```
int sc_main ()
{
  sc_signal<bool> foo;
  sc_signal<bool> bar;

  not_chain c("not_chain");

  foo.write(0);
  c.A.bind(foo);
  c.Z(bar);

  sc_start();

  cout << bar.read(); //1
}
```

Try code on github: https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/not_chain

Fraunhofer IESE

Your notes:

Next Topics

- SystemC Data Types
- More on Modules and Hierarchy
- Ports (Exports, Multiports), Interfaces and Channels
- Event Queues, Event Finders
- Differences to VHDL
- Dynamic Processes
- Primitive Channels (FIFOs, Mutex …)
- Report Handling
- Callbacks (Elaboration…)
- Synthesis Subset / HLS

…

- Transaction Level Modelling (TLM)

Fraunhofer IESE

Your notes: