-SystemC Flow-

1. basic adder
2. SR latch is in the delta_delay
3. **SR latch** with **THREAD** is in thread_example
    a. toplevel is for testbench > connecting modules
        i. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/delta_delay

    b. giving names to the ports
        i. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/feedback_loop

    c. thread example
        i. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/thread_example

BU yukarıdaki 3lü a,b,c birbirlerinin aynısı gibi. Sadece 2. olan, yani feedback loop durmuyor, infinite loop.

4. switching values of variables
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/swapping_example

5. event queue
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/sc_event_and_queue

6. clock generator > not important
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/clock_generator

7. Waveform Tracing - sc_clock - Connecting Modules (Binding) = önemli
    a. file:///C:/Users/User/Desktop/SystemC/SystemC.pdf > page 101
8. connecting modules in hierarchical (not1 > not2 > not3)
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/not_chain

9. data types
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/datatypes

10. polymorphism
    a. page 120
11. custom channel = FIFO > can be only implemented by using SC_THREADS
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/custom_fifo

12. Khan Process Network (KPN) > not detailed
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/kpn_example

13. Mutex > not detailed
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/mutex_example

14. custom signal
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/custom_signal

15. port arrays > 1 module with multiple ports > static
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/portarrays

16. multiple ports > dynamic
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/multiports

17. multiple binding
    a. page 147
18. dynamic processes
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/dynamic_processes

19. printing out report > not important

    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/reporting

20. callbacks
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/callbacks

21. TLM design > has written on the course from scratch
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/custom_tlm

22. TLM Initiator (CPU) > Target (Memory) & Blocking Transport
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_target

    b. page 188
23. TLM interconnect components dynamically
    a. Initiator (CPU) > Interconnect (Bus) > Targets (Memory1 & Memory2)
    b. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_interconnect_target
    c. port arrays are used
24. Quantum keeper > for timing synchronization
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_quantum_keeper

25. DMI (Direct Memory Interface) = bypass the interconnect (ie Bus)
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_dmi

26. Blocking transport but without time
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_debug_transport

27. TLM basic memory manager
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_memory_manager

28. TLM 4 Phase Handshake

    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_target

29. TLM Sockets > no binding required
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_simple_sockets

30. TLM multi-pass-through sockets
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_multipasstrough_sockets

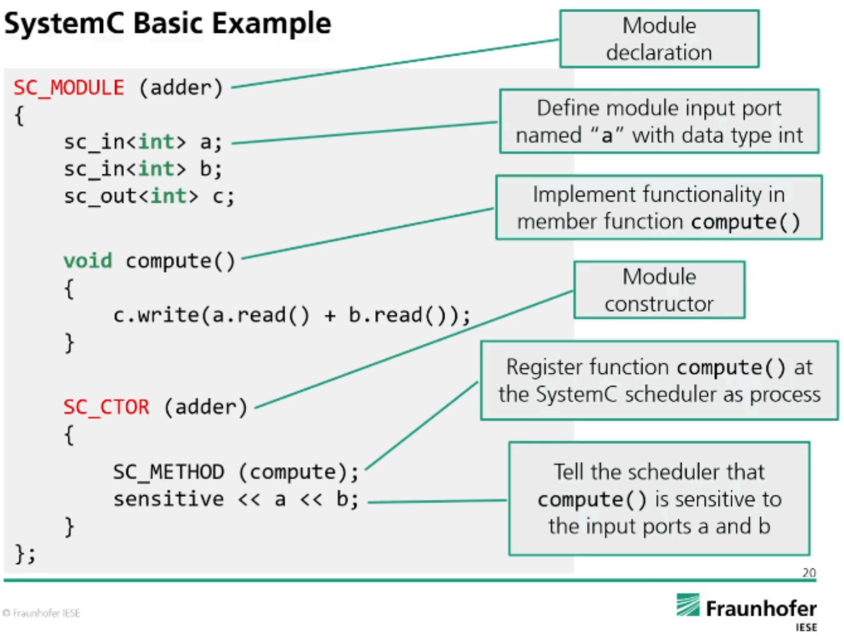
31. TLM backpressure
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_at_backpressure

32. TLM payload extension
    a. https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_payload_extensions

ADDER

- SC_MODULE(adder) {}; == class adder : public sc_module {};
- **<>** == means **template**
- sc_in<int>, sc_out<int>
- void **compute**() { the function}
- C = A + B ====> c.write(a.read() + b.read()) ====> This means these a b and c are not variables, but signals.
- SC_CTOR(adder) {
    - SC_METHOD(**compute**);
    - Sensitive << a << b;
- For an adder for example, we write SC_MODULE(adder){}; Then inside these {}, we write inputs and outputs.
- THEN, **again inside this module**, we define the **method**, which is the function that this module should do. => do it with **void**

- AGAIN, **inside the SC_MODULE**, we will add the constructor, which is **SC_CTOR(adder)**. The name inside should be the same as the name of the module.
    - Inside this SC_CTOR, add SC_METHOD(name of the void function), telling that computation will be done by this adder when it is called.
    - Then we add the sensitivity list by just writing sensitive and using << for each signal



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

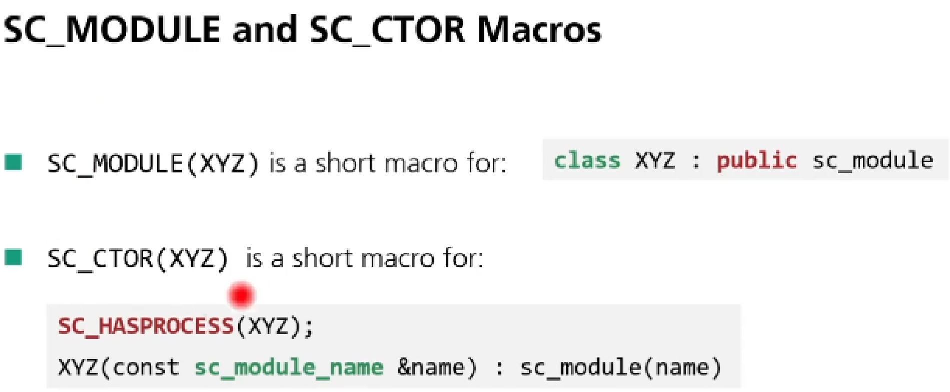
Tell the scheduler that compute() is sensitive to the input ports a and b

20

© Fraunhofer IESE

Fraunhofer IESE

-



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_CTOR yerine SC_HASPROCESS de yazılabilirmiş.

SR LATCH

- **In SC_CTOR, we define the PORTS.**
-