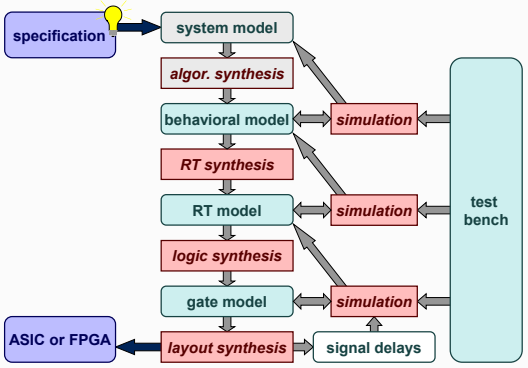
VHDL

0.1. Introduction

VHSIC (Very High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage. Standardized as IEEE 1076.

- Common programming language with extensions to describe hardware: concurrent processes, timing model
- Hierarchical Modeling
- Support of different design styles
- Standardized by IEEE and ANSI since 1987. In this lab VHDL-87 is used (VHDL-93 contains extensions).
- Verilog is another HDL widely used in the U.S.A.
- Extension for describing analog and mixed-signal circuits: VHDL-AMS

0.2. Design Flow



1. Keywords

1.1. Valid Expressions

1.1.1 Comments

```
-- This line is a comment
-- There are no block comments
```

1.1.2 Numbers

```
11   13.1415   13e8   11.6e-19
```

1.1.3 Characters

```
'A'   ' '   '''
```

1.1.4 Strings

1.2. Entity

1.3. Generic Module

```
entity genCntr is
    generic ( N : positive := 8 );
    port ( CLOCK, RESET : in bit;
    COUNTER : out integer range 0 to 2**N-1 );
end genCntr;
```

1.4. architecture

behavioral function: algorithmic or dataflow: simple transformation of the input into the output signals.

structural circuit components and connections

1.5. Dataflow Models

Concurrent signal assignments (i.e. outside processes) are activated independently of their written sequence. Selected signal assignment statement (e.g. 4-input multiplexer):

```
architecture BEHAV2_MUX4 of MUX4 is
begin
with SEL select  -- selector signal
    Z <=    X(0) after 5 ns when "00";
            X(1) after 5 ns when "01";
            X(2) after 5 ns when "10";
            X(3) after 5 ns when "11";
end BEHAV2_MUX4;
```

This statement corresponds to a if-then-else construct.

1.6. Structural Model

```
architecture STRUCT_NAND3 of NAND3 is
    component AND2
        port(E1,E2: in bit; A: out bit);
    end component;
    component INV
        port(A: in bit; Z: out bit);
    end component;
    signal AUX1, AUX2: bit;
begin
    M1: AND2 port map(I1, I2,AUX1);
    M2: AND2 port map(AUX1,I3,AUX2);
    M3: INV port map(AUX2,Z);
end STRUCT_NAND3;
```

1.7. Variables and Signals

Signals correspond to wires between hardware blocks and provide a means to communicate between parallel processes. They hold values like variables, but have a timing behavior. Signals may not be declared inside processes.

```
signal A,B,C : bit_vector(0 to 3) := "1111";
variable EN : bit := 0;
constant Q_delay: time:=5 ns;
```

1.7.1 Attributes of Signal S and variable T : time

S'event returns true or false in current simulation cycle
S'last_event returns elapsed time since last event
S'last_value returns value of S before last event
S'stable(T) returns true if no event for time T
S'delayed(T) the signal S delayed by time T

1.7.2 Edge Detection

STANDARD: `if (SIGNAL'event and SIGNAL='1') then`
IEEE.STD_LOGIC_1164: `if (rising_edge(SIGNAL)) then`

STANDARD: `if (SIGNAL'event and SIGNAL='0') then`
IEEE.STD_LOGIC_1164: `if (falling_edge(SIGNAL)) then`

1.8. Process

A process describes the behavior (the function) of a circuit (component) as a sequence of statements. Processes are identified by names (process labels) and can have a *sensitivity list*. A process is executed, when the value of a signal in its sensitivity list changes. To store intermediate states of a process variables can be used. They can only be declared and used inside of processes.

```
LABEL : process( < SENSITIVE SIGNALS > )
    < VARIABL DEFINITIONS >
begin
    < CODE >
end process;
```

Sensitivity list must be complete and contain all signals that are read within the process.

1.9. Testbenches

```
entity TB is -- empty
end TB; -- no inputs, no outputs

architecture STIMULI of TB is
    component DEC2X4
        port(A,B,EN: in bit;
        D: out bit_vector(0 to 3));
    end component;
    signal X,Y,E: bit:='0';
    R: bit_vector(0 to 3):="1111";
begin
    UUT: DEC2X4 port map(X,Y,E,R);
    ...
end STIMULI;
```

1.10. Module

```
entity multiplexer is
    Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
           B : in  STD_LOGIC_VECTOR (15 downto 0);
           S : in  STD_LOGIC;
           O : out  STD_LOGIC_VECTOR (15 downto 0));
end multiplexer;

architecture Behavioral of multiplexer is

begin
    mux_process: process(A,B,S)
    begin
            case S is
                when'0' => O <= A;
                when'1' => O <= B;
                when others => O <= "XXXXXXXXXXXXXXXX";
            end case;
    end process mux_process;

end Behavioral;
```

2. Delays

Cannot be synthesized

2.1. Wait

sensitive: `wait on S1, S2;`
causal: `wait until SIGNAL = '1'`
temporal: `wait for 1ms`
stop execution: `wait;`

2.2. After

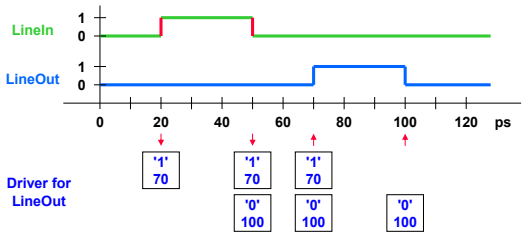
The term `DATA <= X"00" after DELAY;` is executed at simulation time T, but the value is not updated at time T. All active processes continue execution. After all processes have been suspended, the next simulation cycle is started. When the simulator reaches simulation time $T+$DELAY the value `X"00"` is assigned to DATA and all processes sensitive to DATA are started.
Even with a delay of 0, no immediate change of the signal value occurs and the process does not notice the value change until its next execution

B <= A; Signal assignment are scheduled (delayed),
C := B; Variable assignment are immediate!

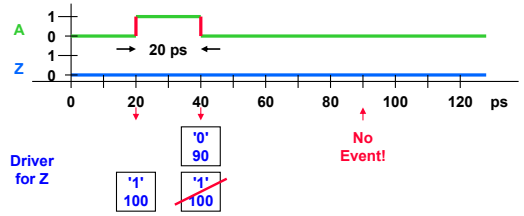
2.3. transmission

Modeling the signal delay of an ideal circuit with unlimited bandwidth, even shortest pulses are transferred. Only the delay influences the result, the signal value does not.

```
signal LineIn, LineOut: bit := '0';
...
TransmissionLine: process(LineIn)
begin
    LineOut <= transport LineIn after 50 ps;
end process TransmissionLine;
```



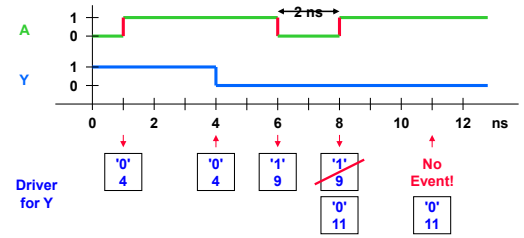

```
signal A, Z: bit := '0';
...
AsymDelay: process(A)
begin
    if A='1' then
        Z <= transport A after 80 ps;
    else -- A='0'
        Z <= transport A after 50 ps;
    end if;
end process AsymDelay;
```



2.4. Inertia

Modeling of real circuits where electronic charge is moved around

```
signal A: bit := '0';
signal Y: bit := '1';
...
INV: process(A)
begin
    Y <= inertial not A after 3 ns;
end process INV;
```



3. Final State Machines (FSM)

3.1. Moore FSM

```
entity FSM_Moore is
    port(Clock, Reset, Enable : in bit;
         E : in bit_vector(1 downto 0);
         A : out bit));
end FSM_Moore;

architecture Sequence of FSM_Moore is
    type States is (Z0, Z1, Z2, Z3); -- enumeration type
    signal State, NSt: States;
begin

-- State Memory Logic
StateMem: process(Clock) -- synchronous reset
begin
    if Clock='1' and Clock'event then
        if Reset='1' then
            State <= Z0 after 20 ns;
        elsif Enable='1' then -- synchronous enable
            State <= NSt after 20 ns;
        end if;
    end if;
end process StateMem;

-- Next State Logic
NextStateLogic: process(E, State)
begin
    case State is
        when Z0 =>
            if E="01" then NSt <= Z1 after 20 ns;
            else NSt <= Z0 after 20 ns;
            end if;
        when Z1 =>
            if E="11" then NSt <= Z1 after 20 ns;
            elsif E="01" then NSt <= Z1 after 20 ns;
            else NSt <= Z0 after 20 ns;
            end if;
        -- ...
    end case;
end process NextStateLogic;

-- Output Logic
OutputLogic: process(E, State)
begin --no if State in Moore, only Mealy
    if (State=Z2 and E="10") then
        A <= '1' after 20 ns;
    else
        A <= '0' after 20 ns;
    end if;
end process OutputLogic;
```

4. Harware Debugging

4.1. Latches

- Latch wird generiert, wenn Signal/Variabler nicht in allen möglichen Verzweigungen (Pfade durch Prozeß) explizit ein Wert zugewiesen wird.
- Latch kann generiert werden, wenn Signal/Variable in einem Verzweigungspfad gelesen wird, bevor neuer Wert zugewiesen wird.
- Andernfalls werden Signale/Variablen zu kombinatorischer Logik synthetisiert bzw. herausoptimiert.

4.2. Variables and Signals

There are no variables in hardware: use signals

5. Packages

```
package STD_LOGIC_1164 is
    -- is stored in the library IEEE
    type STD_LOGIC
        is ('U', -- uninitialized (default)
        'X', -- signal conflict between '0' and '1'
        '0', -- active '0' (driven by a gate)
        '1', -- active '1' (driven by a gate)
        'Z', -- high impedance (tri-state output port)
        'W', -- signal conflict between 'L' and 'H'
        'L', -- weak '0' (pull-down resistor)
        'H', -- weak '1' (pull-up resistor)
        '-' -- don't care, the value is irrelevant and
    );  -- can be used for logic minimization
    type STD_LOGIC_VECTOR
    is array (natural range <>) of STD_LOGIC;
end STD_LOGIC_1164;
```