

Digitaalsüsteemide modelleerimine – motivatsioon

- Erinevad abstraktsiooni-tasemed

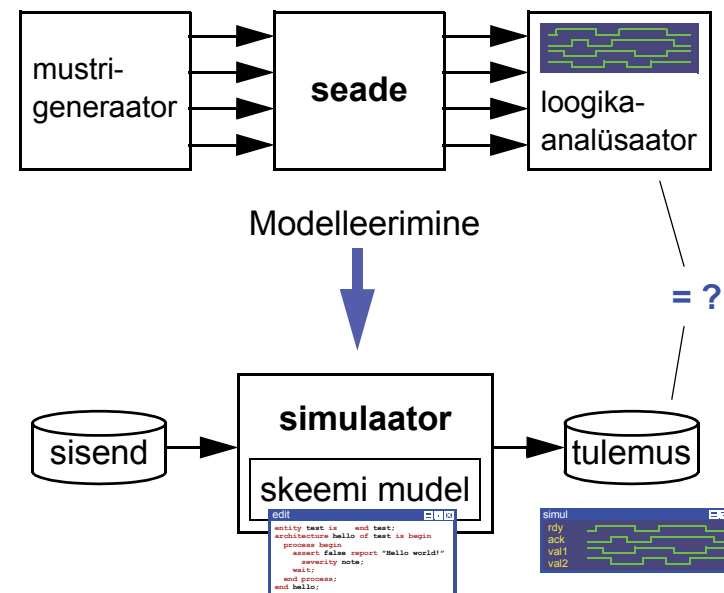
- Sama keel kõikidel projekteerimisetappidel
 - ... või vähemalt enamikel projekteerimisetappidel
- Sama mooduli erinevate kirjelduste võrreldavus
- Simuleerimine erinevatel kirjeldustasemetel

- Dokumenteerimine

- Modelleeritav spetsifikatsioon
 - “isedokumenteerv mudel”

- Süntees

- Lähtekirjedus kõrgel abstraktsioonitasemel
- Formaliseeritav teisendus kõrgemalt abstraktsioonitasemelt madalamale
- Automatiseeritavus
- Verifitseeritavus, st. formaalne erinevate kirjeldustasemete võrdlus
- Valideeritavus, st. käitumuslik erinevate kirjeldustasemete võrdlus



HDL – Hardware Description Language

- **Riistvara kirjelduskeel**
 - Tükeldamine
 - struktuurne hierarhia
 - Funktsionaalsuse kapseldatavus
 - funktsionaalne hierahia
 - Andmete kapseldatavus
 - abstraktsed andmetüübid
 - Paralleelsus / Sama-aegsus
- **VHDL – VHSIC Hardware Description Language**
 - VHSIC – Very High Speed Integrated Circuit
 - 1981 juuni – ajurünnak Massachusetts (riik, DoD, akadeemia)
 - 1986 – IEEE alustas standardiseerimist. IEEE-1076 (VHDL'87)
 - 1994 – versioon VHDL'93 (IEEE 1076-1993), täiendused
 - 1999 – VHDL-AMS (IEEE 1076.1-1999), analoog- ja segasignaaliid
 - 2002 – VHDL'2000 (IEEE 1076-2002), vigade parandused ja täpsustused
 - 2009 – VHDL 4.0 (IEEE 1076-2008), parendatud sünteesitavus, arendus jätkub...



VHDL kirjandus

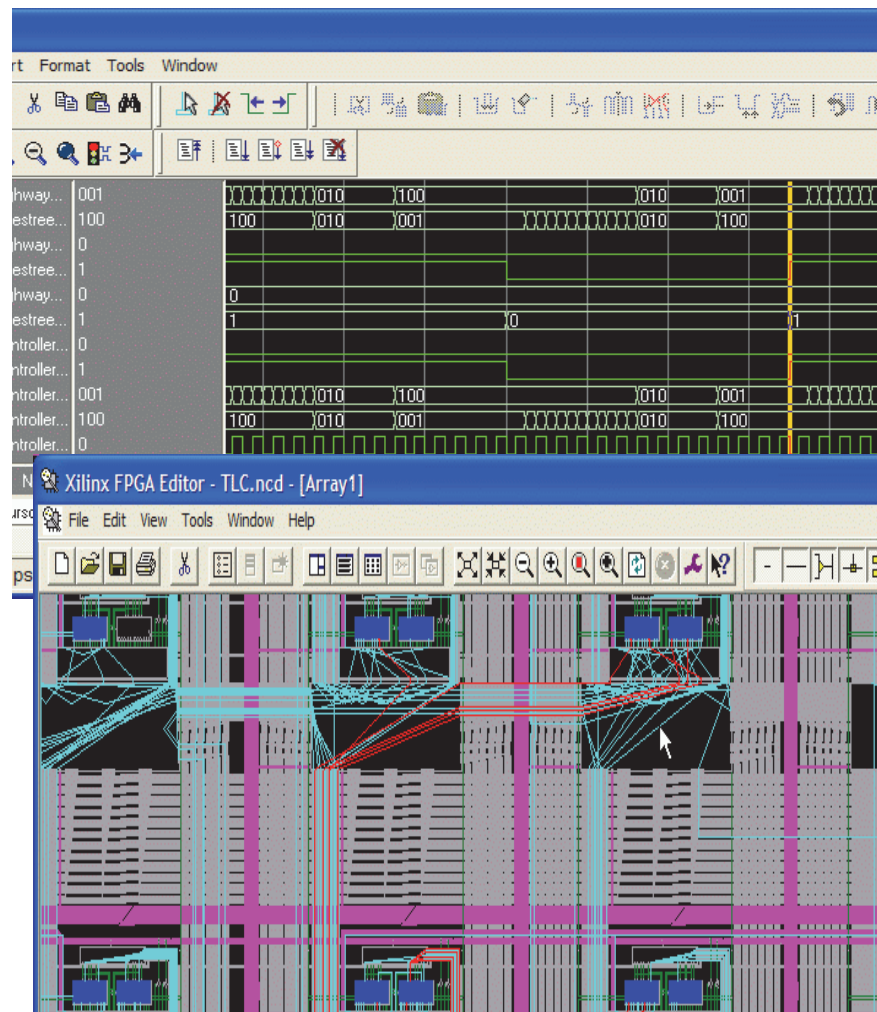
- **Peter J. Ashenden, “The Student's Guide to VHDL” & “The Designer's Guide to VHDL.”**
- **Michael John Sebastian Smith, “Application-Specific Integrated Circuits.”**
 - <http://www10.edacafe.com/book/ASIC/ASICs.php> vt. peatükk 10
- **Dirk Jansen et al. (editors), “The electronic design automation handbook.”**
- **Kalle Tammemäe, “Riistvara kirjeldamiskeel VHDL.”**
- **IAY0340 “Digitaalsüsteemide modelleerimine ja süntees”**
 - <http://mini.pld.ttu.ee/~lrv/IAY0340/>
 - **lisakirjandus & simulaatorid**
 - http://www.mentor.com/company/higher_ed/modelsim-student-edition
 - <http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm>
 - <http://zamiacad.sourceforge.net/web/>
 - <http://ghdl.free.fr/> & <http://sourceforge.net/projects/gtkwave/>

Riistvara disain tänapäeval

- **Model VHDL-s**

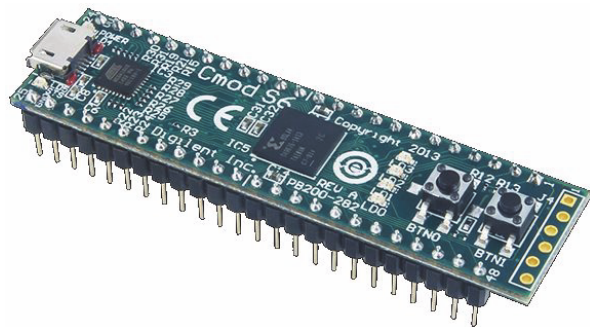
```
--
-- Highway is green, sidestreet is red.
--
if sidestreet_car = NoCar then
    wait until sidestreet_car = Car;
end if;
-- Waiting for no more than 25 seconds ...
if highway_car = Car then
    wait until highway_car = NoCar for 25 sec;
end if;
-- ... and changing lights
highway_light <= GreenBlink;
wait for 3 sec;
highway_light <= Yellow;
sidestreet_light <= Yellow;
wait for 2 sec;
highway_light <= Red;
sidestreet_light <= Green;
```

- **Samm-sammuline täiendamine**
- **Süntees FPGA-le**



Prototüüpimine

- Võimalus kontrollida süsteemi tööd realsusele lähedastes tingimustes ilma vajaduseta luua ülikallist spetsialiseeritud mikroskeemi



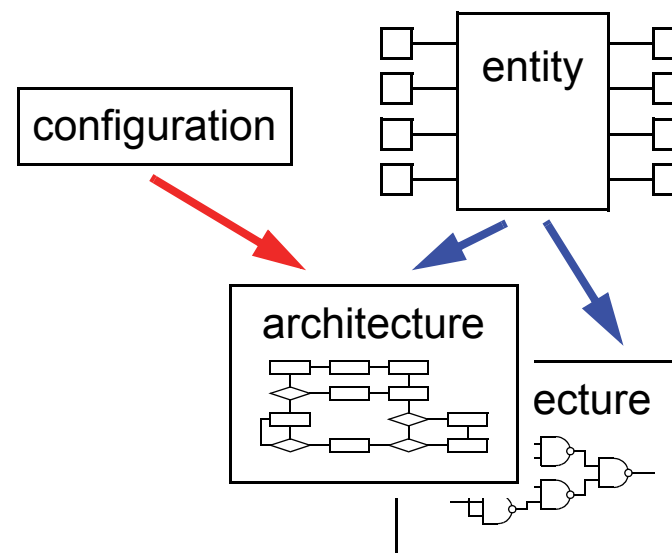
Digilent Inc., Cmod S6
www.digilentinc.com, \$69
 [Xilinx Inc., FPGA XC6SLX4]



Xilinx Inc., Zynq Ultrascale+ MPSoC ZCU106 Eval.Kit, XCZU7EV, \$1995

VHDL-i põhielemendid

- **entity (olem)**
 - liidese kirjeldus
- **architecture (arhitektuur)**
 - käitumise/struktuuri kirjeldus
- **configuration (konfiguratsioon)**
 - kirjelduse (abstr.taseme) määramine
- **package (paketid)**
 - andmetüübid jne.



```
entity test is      end test;
architecture hello of test is begin
    process begin
        assert false report "Hello world!" severity note;
        wait;
    end process;
end hello;
```

Funktsiooni esitamine – näide ja stiilid

- $$f1 = \overline{(\overline{b} \oplus \overline{c})} + a b \overline{c}$$

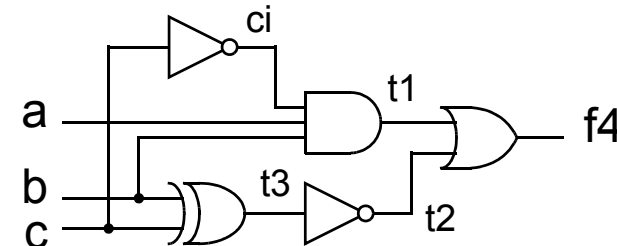
```
f1 <= (not (b xor c)) or (a and b and (not c));
```
- $$f2 = b(a + c) + \overline{b}\overline{c}$$

```
f2 <= (b and (a or c)) or ((not b) and (not c));
```
- $$f3 = (b + \overline{c})(a + \overline{b} + c)$$

```
f3 <= (b or (not c)) and (a or (not b) or c);
```
- f4 – skeem**

```
f4 <= t1 or t2;  t1 <= ci and a and b;
t2 <= not t3;  t3 <= b xor c;  ci <= not c;
```
- f0 – tõeväärtustabeli kirjeldus**

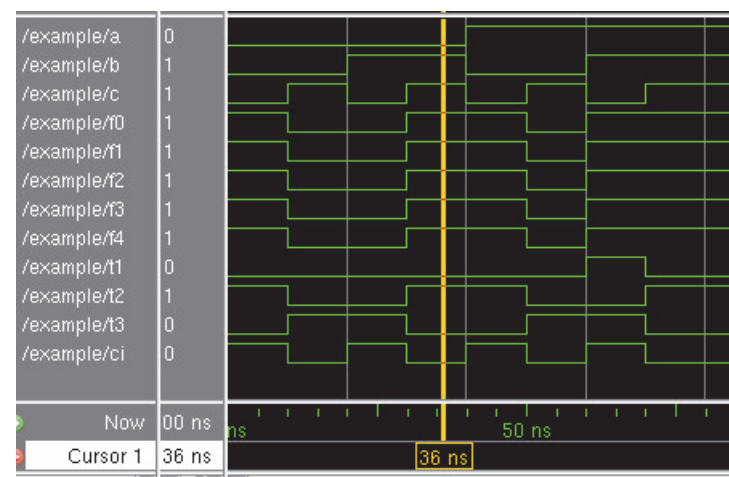
```
process (a, b, c) begin
    f0 <= '0';
    if a='0' then
        if b=c then f0 <= '1'; end if;
    else
        if b='1' or c='0' then f0 <= '1'; end if;
    end if;
end process;
```



abc	f0
000	1
001	0
010	0
011	1
100	1
101	0
110	1
111	1

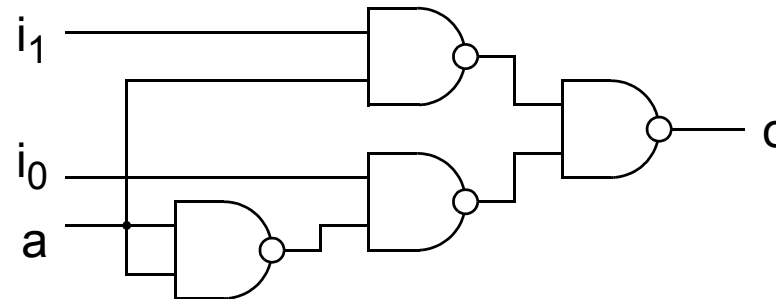
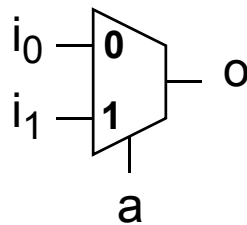
Kontroll ehk simuleerimine

```
entity example is
end entity example;
architecture allinone of example is
    signal a, b, c, f0, f1, f2, f3: bit;
    signal f4, t1, t2, t3, ci: bit;
begin
    a <= '0', '1' after 40 ns;
    b <= '0', '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
    c <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns,
        '0' after 40 ns, '1' after 50 ns, '0' after 60 ns, '1' after 70 ns;
    f1 <= (not (b xor c)) or (a and b and (not c));
    f2 <= (b and (a or c)) or ((not b) and (not c));
    f3 <= (b or (not c)) and (a or (not b) or c);
    f4 <= t1 or t2; t1 <= ci and a and b; t2 <= not t3;
    t3 <= b xor c; ci <= not c;
    process (a, b, c) begin
        f0 <= '0';
        if a='0' then
            if b=c then f0 <= '1'; end if;
        else
            if b='1' or c='0' then f0 <= '1'; end if;
        end if;
    end process;
end architecture allinone;
```



Multiplekser

- $o = \bar{a} i_0 + a i_1$
- De Morgan'i seadus
 - $\bar{a} \& \bar{b} = \overline{a + b}$ $\bar{a} + \bar{b} = \overline{a \& b}$
 - $a' \& b' = (a + b)'$ $a' + b' = (a \& b)'$
- $o = ((a' \& i_0)' \& (a \& i_1)')'$

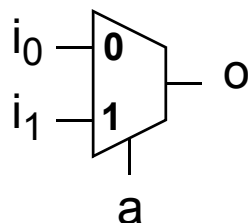




Multiplekser

olem

```
entity MUX is
  port ( a, i0, i1 : in bit;
        o : out bit );
end MUX;
```



käitumuslik

```
architecture behave of MUX is
begin
```

```
  process ( a, i0, i1 ) begin
    if a = '1' then
      o <= i1;
    else
      o <= i0;
    end if;
```

```
  end process;
end behave;
```

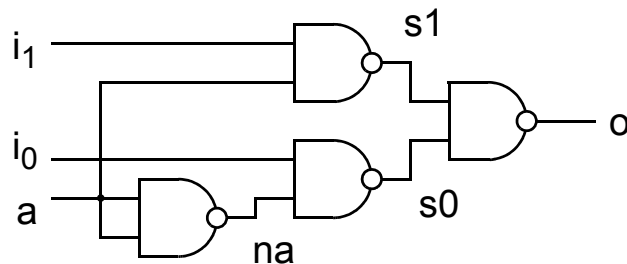
- $$o = \bar{a} i_0 + a i_1$$

Multiplekser

andmevoog

architecture dataflow of MUX is
begin
 o <= ((not a) and i0) or
 (a and i1);
end dataflow;

- $$o = \bar{a} i_0 + a i_1$$

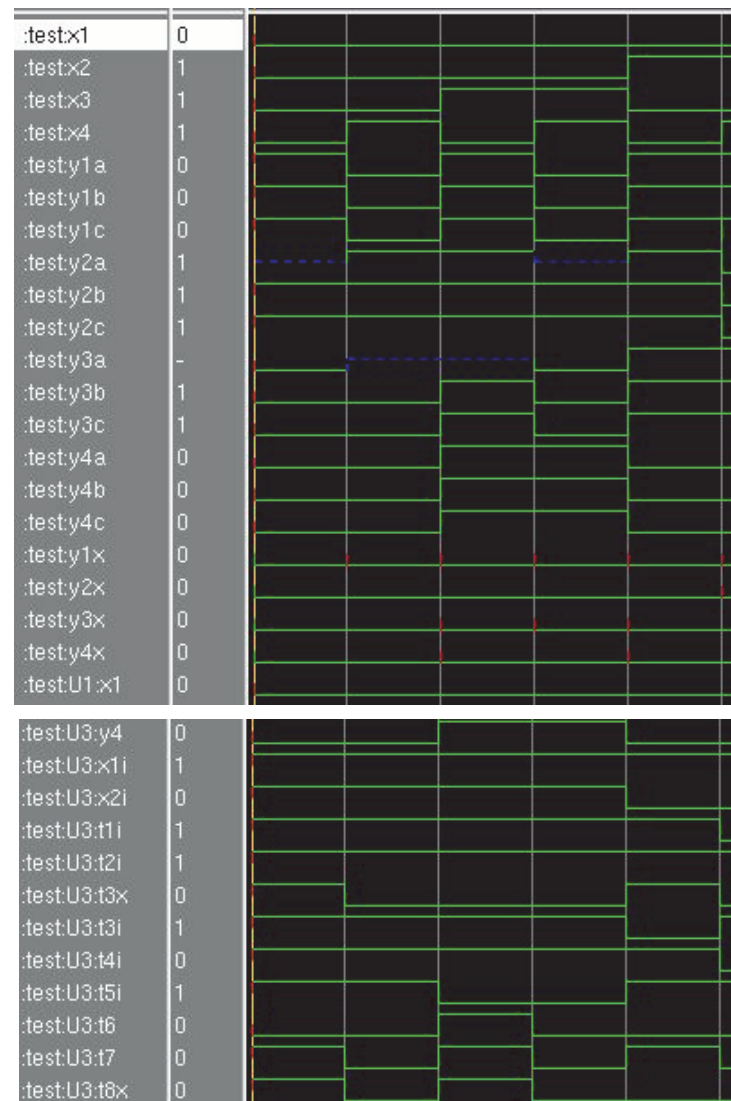
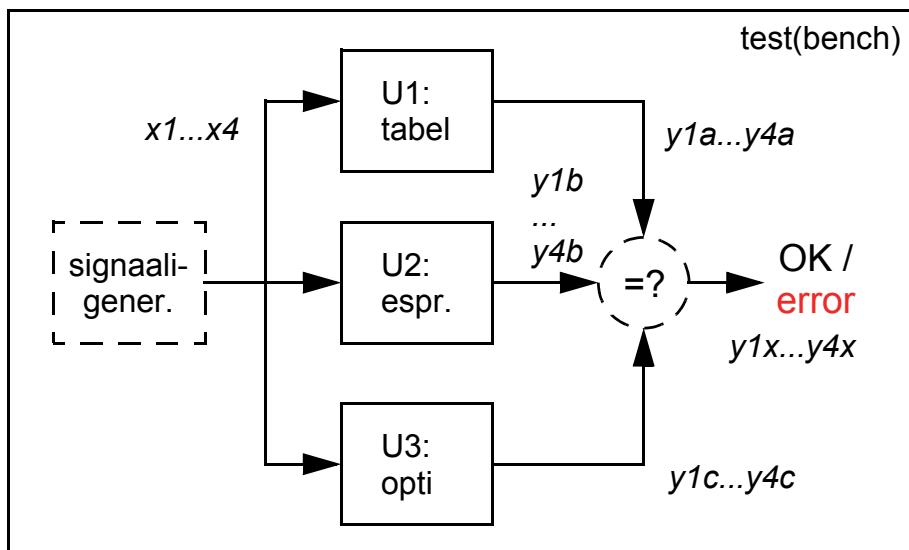


struktuur

architecture struct of MUX is
 component NANDg
 port (i0, i1 : in bit;
 c : out bit);
 end component;
 signal na, s1, s0 : bit;
begin
 U1: NANDg port map (a,a,na);
 U2: NANDg port map (i1,a,s1);
 U3: NANDg port map (i0,na,s0);
 U4: NANDg port map (s1,s0,o);
end struct;

VHDL & kodutöö #1

- Kolm etappi, kõiki vaja kontrollida
 - sisendkombinatsioonide genereerimine
 - kolm erinevat kirjeldust korraga
 - tulemuste automaatne võrdlemine



Testpink == testkeskkond

- Signaalide deklareerimine ja sisendite genereerimine

```
library IEEE; use IEEE.std_logic_1164.all;
entity test is
end entity test;
```

```
library IEEE; use IEEE.std_logic_1164.all;
architecture bench of test is
    signal x1, x2, x3, x4: std_logic;
    signal y1a, y1b, y1c, y2a, y2b, y2c: std_logic;
    signal y3a, y3b, y3c, y4a, y4b, y4c: std_logic;
    signal y1x, y2x, y3x, y4x: std_logic;
    ...
```

```
begin
```

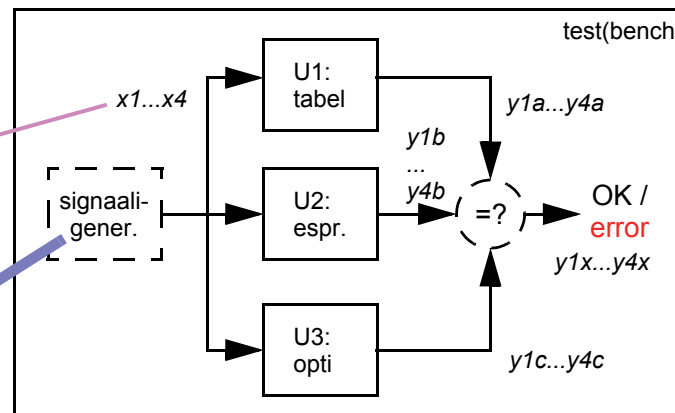
```
-- Input signals (after every 10 ns)
```

```
x1 <= '0' after 0 ns, '1' after 80 ns, '0' after 160 ns;
x2 <= '0' after 0 ns, '1' after 40 ns, '0' after 80 ns, '1' after 120 ns;
x3 <= '0' after 0 ns, '1' after 20 ns, '0' after 40 ns, '1' after 60 ns,
      '0' after 80 ns, '1' after 100 ns, '0' after 120 ns, '1' after 140 ns;
x4 <= '0' after 0 ns, '1' after 10 ns, '0' after 20 ns, '1' after 30 ns,
      '0' after 40 ns, '1' after 50 ns, '0' after 60 ns, '1' after 70 ns,
      '0' after 80 ns, '1' after 90 ns, '0' after 100 ns, '1' after 110 ns,
      '0' after 120 ns, '1' after 130 ns, '0' after 140 ns, '1' after 150 ns;
```

```
...
```

```
end architecture bench;
```

← deklareerimine
← laineaju genereerimine



Testpink == testkeskkond

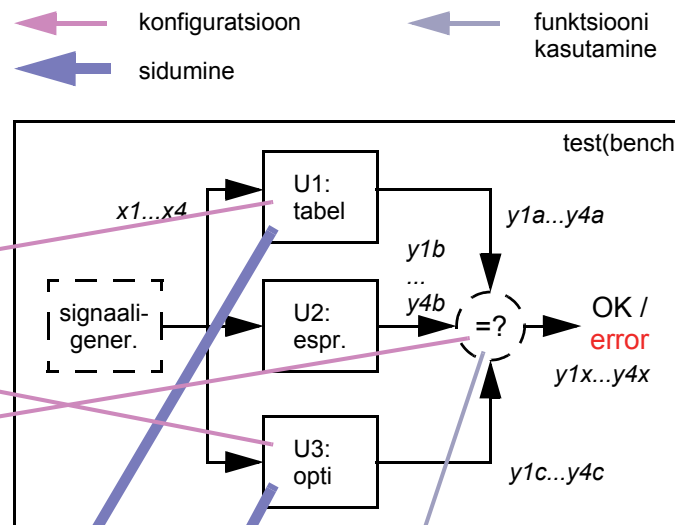
Mitu varianti korrage ja tulemuste võrdlemine

```

component f_system
  port ( x1, x2, x3, x4: in std_logic;
         y1, y2, y3, y4: out std_logic );
end component;

for U1: f_system use entity work.f_system(tabel);
for U2: f_system use entity work.f_system(espresso);
for U3: f_system use entity work.f_system(opti);

function compare_signals (s1, s2, s3: std_logic)
  return std_logic is
  begin
    if s1='-' then      if s2/=s3 then return 'X'; end if;
    else                if s1/=s2 or s1/=s3 then return 'X'; end if;
    end if;
    return '0';
  end function compare_signals;
...
U1: f_system port map (x1, x2, x3, x4, y1a, y2a, y3a, y4a);
U2: f_system port map (x1, x2, x3, x4, y1b, y2b, y3b, y4b);
U3: f_system port map (x1, x2, x3, x4, y1c, y2c, y3c, y4c);
y1x <= compare_signals (y1a, y1b, y1c);    y2x <= compare_signals (y2a, y2b, y2c);
y3x <= compare_signals (y3a, y3b, y3c);    y4x <= compare_signals (y4a, y4b, y4c);
end architecture bench;
  
```



Kodutöö #1 ja kirjeldus-stiilid

```
library IEEE; use IEEE.std_logic_1164.all;
entity f_system is
    port ( x1, x2, x3, x4: in std_logic;
           y1, y2, y3, y4: out std_logic );
end entity f_system;
-- behavioral
architecture tabel of f_system is
begin
    process (x1, x2, x3, x4)
        variable in_word, out_word:
            std_logic_vector (3 downto 0);
    begin
        in_word := x1 & x2 & x3 & x4;
        case in_word is
            when "0000" => out_word := "1-00";
            when "0001" => out_word := "01-0";
            when "0010" => out_word := "11-1";
            when "0011" => out_word := "0-01";
            when "0100" => out_word := "1110";
            ...
            when "1110" => out_word := "-000";
            when "1111" => out_word := "1011";
            when others => out_word := "----";
        end case;
        y1 <= out_word(3); y2 <= out_word(2);
        y3 <= out_word(1); y4 <= out_word(0);
    end process;
end architecture tabel;
```

```
-- data-flow
architecture espresso of f_system is
begin
    y1 <= ((not x1) and x2 and (not x3) and x4) or
           (x2 and (not x3) and (not x4)) or
           ((not x1) and (not x4)) or (x1 and x3);
    ...
    y4 <= (x1 and x3 and x4) or ((not x2) and x3) or
           ((not x1) and x3 and (not x4)) or
           (x1 and (not x2) and (not x4));
end architecture espresso;

-- data-flow (almost structural)
architecture opti of f_system is
    signal x1i, x2i, t1i, t2i, ... t6: std_logic;
    signal t7, t8x, t8, t68i, ... t197i: std_logic;
begin
    x1i <= not (x1 and x1);
    x2i <= not (x2 and x2);
    t1i <= t4i or x1 or x3;
    t2i <= not (x1 and x3 and x4);
    ...
    t197i <= not (t19 or t7);
    y1 <= not (t197i and t3i);
    y2 <= not (t19 or t8);
    y3 <= not (t3i and t4i and t68i);
    y4 <= not (t2i and t5i and t68i);
end architecture opti;
```



Andmete esitamine

- **Neli objektide klassi:**
 - konstant (constant) – fikseeritud väärtus
 - signaal (signal) – väärtustel on ajalugu (sündmused)
 - muutuja (variable) – ainult jooksev väärtus
 - fail (file) – simuleeriva arvuti failisüsteemi objekt
- **Tüüp (type)**
 - esitab objekti struktuuri, ülesehitust ja mäluvajadust
- **Klass kirjeldab objekti käitumist ja kuidas objekti mudelis kasutatakse**
 - jooksva väärtuse lugemine
 - uue väärtuse omistamine – nt. signaali värskendatakse simulatsioonitsükli lõpus
- **Indentifikaatorid**
 - “INTGR9”, “intgl_5” – legaalne
 - “Intgrl-5”, “Acquire_”, “8to3”, “Abc@adr” – illegaalne
- **Literaaliid**
 - arvud – 12 1.34E-12 8#1470_0220#
 - sümbolid/stringid – 'A' "string"
 - bitid/vektorid – '0' "1101" X"FFF"



Eraldajad

	Name	Example
&	jätkaja (konkatenaator)	FourB_v := TwoB_v & "10"
	"või"	when 'Y' 'y' =>
#	baasiga literaalid	Total_v := 16#2AF#
:	andmeobj. eraldaja	variable Sw_v : OnOff_Typ;
.	kirje element	OnOff_v := Message_v.Switch_v;
=>	"siis"	when On1 => Sun_v := 6;
=>	"saab"	Arr_v := (El1 => 5, others => 100);
:=	muutujale omistamine	Sum_v := Numb_s + 7;
<=	signaalile omistamine	Count_s <= 5 after 5 ns;
<>	"määramata"	type S_Typ is array (integer range <>) ...
--	kommentaar	-- this is definitely a comment;

Operaatorid

- **loogika:** *and, or, nand, nor, xor* (VHDL'87) *xnor* (VHDL'93)
- **võrdlus:** *=, /=, <, >, <=, >=*
- **nihutamine:** *sll, srl, sla, sra, rol, ror* (VHDL'93)
- **liitmine:** *+, -, &*
- **märk:** *+, -*
- **korrutamine:** **, /, mod, rem*
- **varia:** ***, abs, not*
- **Näited**
 - **&** : jätkamine, *'1' & "10" = "110"*
 - ****** : astendamine, *2**3 = 8*
 - **mod** : moodul, *7 mod (-2) = -1 – A=B*N+(A mod B)*
 - **rem** : jääk, *7 rem (-2) = 1 – A=(A/B)*B+(A rem B)*

Avaldised & käsud

- **Omistamine**
 - signaalile – $s \leq [\text{transport} \mid \text{inertial}] \text{ expression } [\text{after time}] ;$
 - muutujale – $v := \text{expression};$
 - avaldis – $\text{expression operation expression}$
 $\text{variable} \mid \text{signal}$
 function-call
- **Kontrollvoo käsud (järjestikulised)**
 - signaalile/muutujale omistamine
 - tingimuslikud – if-then-else, case
 - tsüklid – for-loop, while-loop
 - protseduuri välja kutsumine
 - ajakontroll (wait); muud käsud (nt. assert)
- **Andmevoo käsud (sama-aegsed)**
 - signaalile omistamine
 - tingimuslikud – when-else
 - sama-aegse protseduuri välja kutsumine
 - komponentide sidumine; muud käsud (nt. assert, generate)



Tüübid

- Skalaartüüp – diskreetne (integer, enumeration, physical) või reaaltüüp
- Komposiit-tüüp – massiivid ja kirjed
- Pöördustüüp (access) & failitüüp – (ainult simuleerimine)
- Alamtüübi (*subtype*) korral seatakse andmetüübile täiendavaid piiranguid
 - piirangutest kinni pidamist kontrollitakse simuleerimise käigus
- Näited:
 - type Bit_position is range 7 downto 0; -- uus tüüp
 - subtype Int0_5_Typ is integer range 0 to 5; -- alamtüüp
 - type Color is (Green,Yellow,Red); -- loendustüüp (enumeration)

Pakett STANDARD

```
package STANDARD is
  type BOOLEAN is (FALSE, TRUE);
  type BIT is ('0', '1');
  type CHARACTER is (NUL, SOH, ..., 'a', 'b', 'c', ..., DEL);
  type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
  type INTEGER is range -(2**31-1) to (2**31-1);
  type REAL is range ...;
  type TIME is range ...
    units fs; ps=1000 fs; ... hr=60 min; end units;
  function NOW return TIME;
  subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
  subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
  type STRING is array (POSITIVE range <>) of CHARACTER;
  type BIT_VECTOR is array (NATURAL range <>) of BIT;
end STANDARD;
```



Tüübiteisendus

- VHDL on range tüüpimisega
 - teisendused on vajalikud eri tüüpide vahel

```

use IEEE.std_logic_1164.all;      -- for std_logic_vector
use IEEE.std_logic_arith.all;    -- for signed and unsigned
...
signal k: std_logic_vector(7 downto 0) := "11110000";
signal a, b: signed(7 downto 0);
signal c: unsigned(15 downto 0);
...
a <= conv_signed(100,8);          -- conversion function
c <= conv_unsigned(65535,16);    -- conversion function
b <= signed' ("00001111");      -- type casting
a <= a + signed' (k);           -- type casting

```



Arhitektuur

- **Arhitektuur (architecture) – deklaratsioonid ja tegevused**
 - Protsessid (process) ja andmevoo käsud (concurrent signal assignment)
 - Alamkomponentide paigaldus (component instantiation)
 - Muud sama-aegselt täidetavad käsud
 - concurrent procedure, generate, concurrent assertion, block
- **Protsess (process) – deklaratsioonid ja mudeli käitumine**
 - peab sisaldama aja kontrolli – tundlikkuse nimistu või ootekäsud
- **Ekvivalentsed käitumised**
 - andmevoo käsk


```
x <= a and b after 5 ns;
```
 - protsess & tundlikkuse nimistu


```
process ( a, b ) begin
    x <= a and b after 5 ns;
end process;
```
 - protsess & ootekäsk


```
process begin
    wait on a, b;
    x <= a and b after 5 ns;
end process;
```



Ajakontroll

- Signaalile omistamise edasi lükkamine – “... after T;”
- Tundlikkuse nimistu
- Ootekäsud (wait)
 - oota signaali sündmust – wait on x;
 - oota tingimuse täitumist – wait until x='1';
 - oota kindel ajavahemik – wait for 20 us;
 - oota (igavesti) – wait;
 - kombineritud kasutamine – wait on clk until clk='1' and ready='1' for 1 us;
- ootekäskude tundlikkus
 - wait on a until a='1' and b='0'; – tundlik muutustele ainult signaalil *a*
 - wait until a='1' and b='0'; – tundlik muutustele signaalidel *a* ja *b*



Tingimuslikud käsud

- **if-then-else**

```
[label:] if conditional-expression then statements...
elsif conditional-expression then statements...
else statements...
end if [label];
```

- *conditional-expression* – avaldis, mis tagastab loogikaväärtuse (boolean)

- **case**

```
[label:] case expression is
when constant-value [| constant-value] => statements...
when others => null
end case [label];
```



Tsüklid

```
[label:] [iteration-method] loop
    statements...
end loop [label];
```

```
iteration-method ::=
    while conditional-expression | for counter in range
```

```
exit [label] [when conditional-expression];
next [label] [when conditional-expression];
```

```
range ::=    expression to expression |
              expression downto expression |
              type'range | ...
```



Tsüklid – näide

- **for-loop**

```
for I in my_array'range loop
  next when I<lower_limit;
  exit when I>upper_limit;
  sum := sum + my_array(I);
end loop;
```
- **while-loop**

```
while a<10 loop
  a := a + 1;
end loop;
```



Käitumuslik hierarhia

- **Funktsioonid ja protseduurid**
 - **function**
 - kasutatakse ainult avaldisena
 - ei tohi sisaldada ajakontrolli käske
 - ainult sisendparameetrid (sisuliselt konstandid)
 - operaatorite defineerimine
 - **procedure**
 - kasutatakse käsuna
 - võib sisaldada ajakontrolli käske
 - sisend- (konstandid) ja väljundparameetrid (muutujad/signaalid)

```
function "and" (l,r: signed) return signed is begin
    return signed(std_logic_vector(l) and std_logic_vector(r));
end;
-- ...
x <= a and b;
-- ...
```

Käitumuslik hierarhia – protseduurid

```
PACKAGE adder_elements IS
-- full_adder: 1-bit full adder (declaration)
PROCEDURE full_adder (
    CONSTANT a0, b0, c0: IN bit;
    VARIABLE o0, c1: OUT bit);
END adder_elements;
```

```
PACKAGE BODY adder_elements IS
PROCEDURE half_adder (
    CONSTANT a0, b0: IN bit;
    VARIABLE o0, c1: OUT bit)
IS
BEGIN
    o0 := a0 XOR b0;
    c1 := a0 AND b0;
END half_adder;
```

```
-- full_adder: 1-bit full adder
-- (body, i.e. implementation)
PROCEDURE full_adder (
    CONSTANT a0, b0, c0: IN bit;
    VARIABLE o0, c1: OUT bit)
IS
    VARIABLE c_1, c_2, o_1: bit;
BEGIN
    half_adder ( a0, b0, o_1, c_1 );
    half_adder ( o_1, c0, o0, c_2 );
    c1 := c_1 or c_2;
END full_adder;
END adder_elements;
```

```
PACKAGE BODY adder_elements IS
-- full_adder: 1-bit full adder
-- (body, i.e. implementation)
PROCEDURE full_adder (
    CONSTANT a0, b0, c0: IN bit;
    VARIABLE o0, c1: OUT bit)
IS
    -- Defining a 3-D array (hyper-cube)
    TYPE look_up_1 IS ARRAY (bit) OF bit;
    TYPE look_up_2 IS ARRAY (bit) OF look_up_1;
    TYPE look_up_3 IS ARRAY (bit) OF look_up_2;
    CONSTANT output: look_up_3 :=
-- -----
-- |    0    1        0    1    |    a0 /    |
-- |    0    0        1    1    |    b0 / c0 |
-- -----
    ( ( ( '0', '1' ), ( '1', '0' ) ), -- | 0 |
      ( ( '1', '0' ), ( '0', '1' ) ) ); -- | 1 |
    CONSTANT carry: look_up_3 :=
-- -----
-- |    0    1        0    1    |    a0 /    |
-- |    0    0        1    1    |    b0 / c0 |
-- -----
    ( ( ( '0', '0' ), ( '0', '1' ) ), -- | 0 |
      ( ( '0', '1' ), ( '1', '1' ) ) ); -- | 1 |
BEGIN
    o0 := output (a0) (b0) (c0);
    c1 := carry  (a0) (b0) (c0);
END full_adder;
END adder_elements;
```



Atribuudid

```
-- type myArray is array (9 downto 0) of any_type;
-- variable an_array: myArray;
-- type fourval is ('0', '1', 'Z', 'X');
-- signal sig: sigtype;
-- constant T: time := 10 ns;
```

Atribuut	Tüüp	Tulemus
myArray'high left low right	integer	9 9 0 0
myArray'ascending	boolean	false
an_array'length range reverse_range	integer	10 9 downto 0 0 to 9
fourval'leftof('0') rightof('1')	fourval	error 'Z'
fourval'pos('Z')	integer	2
fourval'pred('1') succ('Z') val(3)	fourval	'0' 'X' 'X'
sig'delayed(T)	sigtype	sig-i 10 ns hilistatud väärtus
sig'event	boolean	tõene kui sig-l on sündmus
sig'last_event	time	aeg eelmisest sündmusest
sig'last_value	sigtype	väärtus enne viimast sündmust
sig'stable(T)	boolean	sündmus (now-T)-st kuni now-ni
...	...	...



Atribuutide kasutamine

- **Fikseeritud kood**

```
signal s: bit_vector (7 downto 0);
...
for i in 0 to 7 loop ...
```

- **Paindlikum kood**

```
constant sz: integer := 16;
signal s: bit_vector (sz-1 downto 0);
...
for i in 0 to sz-1 loop ...
```

- **Kõige paindlikum kood (atribuudid ja üldistatud parameeter)**

```
entity ... is
  generic (sz: positive);
  port ...
  ...
  signal s: bit_vector (sz-1 downto 0);
  ...
  for i in s'low to s'high loop ...
  for i in s'reverse_range loop ...
```

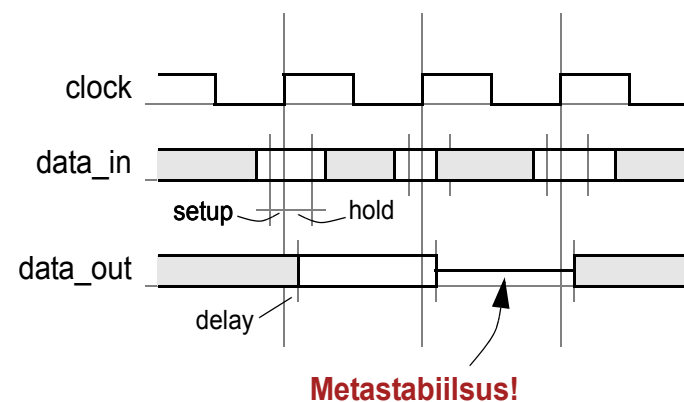
Atribuutide kasutamine – #2

- Seade- & hoide-ajad mäluelementides (trigerites)
 - seadeaeg (setup time) – sisend peab olema stabiilne mingi aeg enne taktsignaali aktiivset fronti
 - hoideaeg (hold time) – sisend peab olema stabiilne mingi aeg pärast taktsignaali aktiivset fronti
- Põhjuseks signaalide levimine eri teid pidi mäluelemendi sees
- Tagajärjeks võib olla metastabiilsus – väljundi '0' ja '1' vahel

```

process (clock,data_in) begin
  if clock'event and clock='1' then
    assert data_in'last_event >= 3 ns
    report "setup time violation" severity warning;
    data_out <= data_in after 3 ns;
  end if;
  if data_in'event and clock='1' then
    assert clock'last_event >= 5 ns
    report "hold time violation" severity warning;
  end if;
end process;

```





Struktuurne hierarhia

- Paketid, teegid
- Komponendid, konfiguratsioonid

Mäluelemendid

- Protsess, mis peab meeles (osade) signaalide/muutujate väärtuseid
- Protsessi peatamine kindlaks ajaks

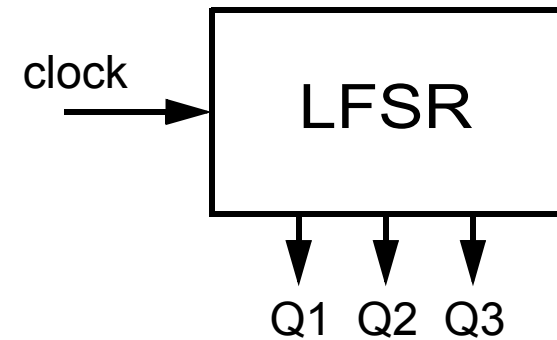
```
process begin
    wait on CLK until CLK='1';
    Q1 <= D1;
end process;

process (CLK) begin
    if CLK='1' and CLK'event then
        Q2 <= D2;
    end if;
end process;
```

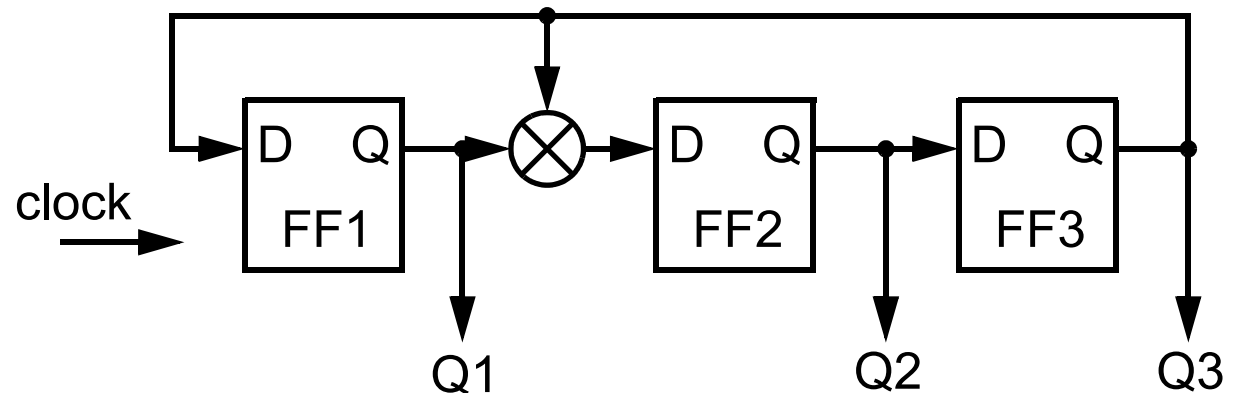
```
process (CLK) begin
    if CLK='1' and CLK'event then
        if RES='1' then
            Q3 <= '0';
        elsif ENA='1' then
            Q3 <= D3;
        end if;
    end if;
end process;
```

Näide #2 – LFSR

```
entity LFSR is
  port ( clock : in bit;
        Q1,Q2,Q3 : out bit );
end LFSR;
```



- $Q3 := Q2 \parallel$
 $Q2 := Q1 + Q3 \parallel$
 $Q1 := Q3$





LFSR – käitumuslik kirjeldus

architecture behavior of LFSR is

begin

 process

 variable Olek: bit_vector(3 downto 0) := "0111";

 begin

 Q3 <= Olek(2) after 5 ns;

 Q2 <= Olek(1) after 5 ns;

 Q1 <= Olek(0) after 5 ns;

 wait on clock until clock = '1';

 Olek := Olek(2 downto 0) & '0';

 if Olek(3) = '1'

 then Olek := Olek xor "1011";

 end if;

 end process;

end behavior;



LFSR – andmevoo kirjeldus

architecture dataflow of LFSR is

```

    signal FF1, FF2, FF3 : bit := '1';
begin
b1: block (clock = '1' and not clock'stable)
    begin
        FF3 <= guarded FF2 after 5 ns;
        FF2 <= guarded FF1 xor FF3 after 5 ns;
        FF1 <= guarded FF3 after 5 ns;
    end block;
    Q3 <= FF3;
    Q2 <= FF2;
    Q1 <= FF1;
end dataflow;
  
```

LFSR – struktuurne kirjeldus

architecture structure of LFSR is

```

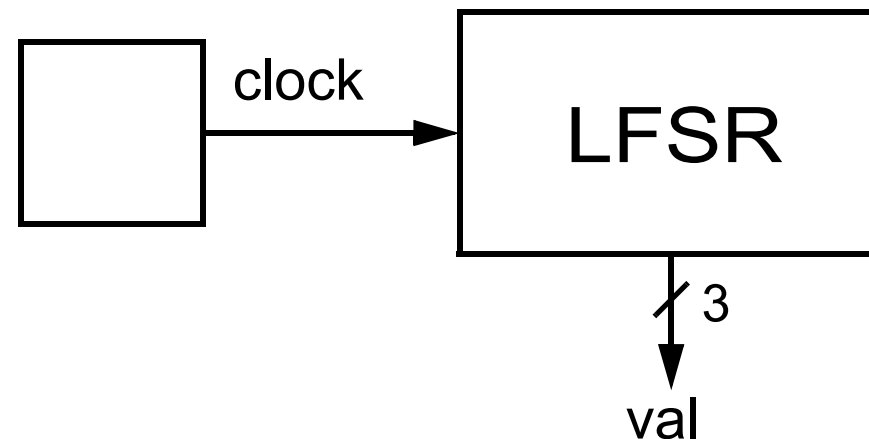
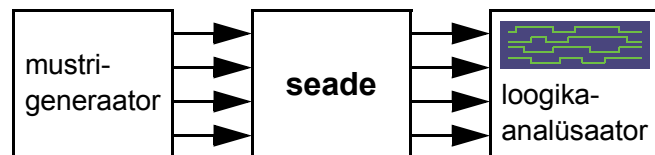
signal xor_out : bit;
signal SR1, SR2, SR3 : bit := '1';
component FF
    port ( clock, data : in bit; Q out bit );
end component;
component XORgate
    port ( a, b : in bit;  x : out bit );
end component;
begin
    FF1: FF port map ( clock, SR3, SR1 );
    FF2: FF port map ( clock, xor_out, SR2 );
    FF3: FF port map ( clock, SR2, SR3 );
    xor1: XORgate port map ( SR1, SR3, xor_out );
    Q3 <= SR3;    Q2 <= SR2;    Q1<= SR1;
end structure;

```

LFSR – testpink

- stiimul-signaali generaator → testitav moodul → tulemuste analüüs

```
entity LFSRstim is
end LFSRstim;
```





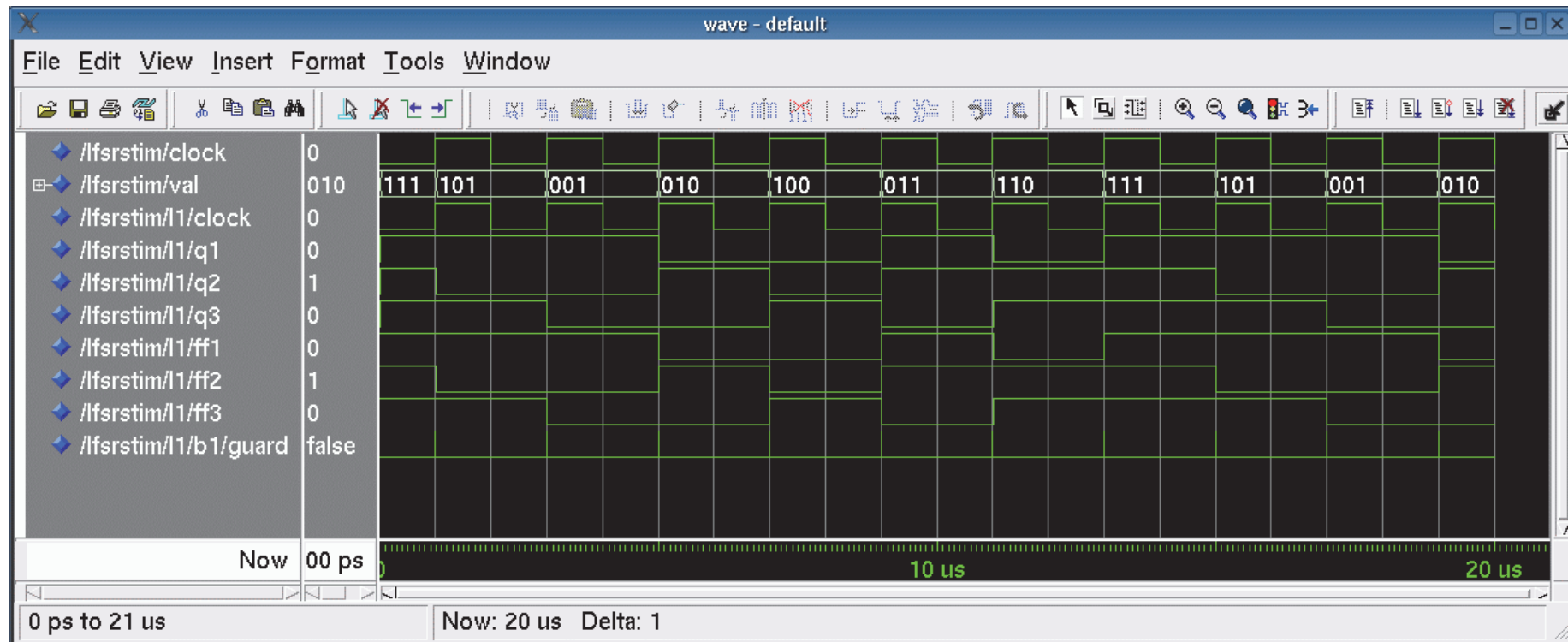
LFSR – testpink

```

architecture test of LFSRstim is
  component LFSR
    port ( clock : in bit;  Q1, Q2, Q3 : out bit );
  end component;
  signal clock : bit := '0';
  signal val : bit_vector (3 downto 1);
begin
  L1: LFSR port map ( clock, val(1), val(2), val(3) );
  process begin
    for I in 1 to 20 loop
      wait for 1 us;      clock <= not clock;
    end loop;
    wait;
  end process;
end test;

```

LFSR – simulatsiooni tulemused



Konfiguratsioon <http://www10.edacafe.com/book/ASIC/ASICs.php>

- **komponendid**

```
entity AD2 is port (A1, A2: in BIT; Y: out BIT); end;
architecture B of AD2 is begin Y <= A1 and A2; end;
entity XR2 is port (X1, X2: in BIT; Y: out BIT); end;
architecture B of XR2 is begin Y <= X1 xor X2; end;
```
- **komponentide deklaratsioonid & konfiguratsiooni spetsifikatsioon**

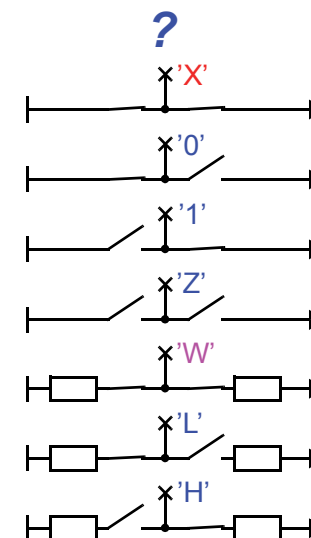
```
entity Half_Adder is port (X, Y: BIT; Sum, Cout: out BIT); end;
architecture Netlist of Half_Adder is use work.all;
    component MX port (A, B: BIT; Z:out BIT); end component;
    component MA port (A, B: BIT; Z:out BIT); end component;
    for G1:MX use entity XR2(B) port map(X1 => A,X2 => B,Y => Z);
begin
    G1:MX port map (X, Y, Sum); G2:MA port map (X, Y, Cout);
end;
```
- **konfiguratsiooni deklaratsioon, plokk-konfiguratsioon, komponendi konfiguratsioon**

```
configuration C1 of Half_Adder is
    use work.all;
    for Netlist
        for G2:MA
            use entity AD2(B) port map(A1 => A,A2 => B,Y => Z);
        end for;
    end for;
end;
```



IEEE 9-valentne loogika

Väärtus	Nimetus	Interpretatsioon
'U'	uninitialized	mudeli käitumine
'X'	forcing unknown	mudeli käitumine
'0'	forcing 0	loogikaväärtus ("transistor")
'1'	forcing 1	loogikaväärtus ("transistor")
'Z'	high impedance	ühendamata
'W'	weak unknown	mudeli käitumine
'L'	weak 0	loogikaväärtusl ("takisti")
'H'	weak 1	lloogikaväärtus ("takisti")
'_'	don't care	määramatusl



- **IEEE.std_logic_1164 – std_ulogic, std_logic ja std_logic_vector**
 - std_ulogic – baastüüp; std_logic – mitu signaali-allikat ja lahendus-funktsioon
- **IEEE.std_logic_arith – signed, unsigned**
 - std_logic_vector'i laiendamine märgiga või märgita täisarvuks

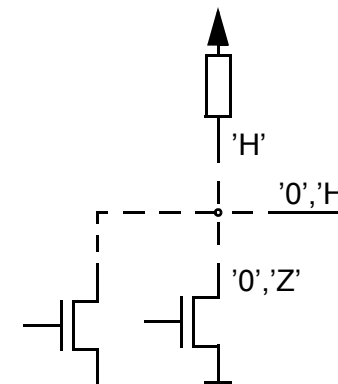
Lahendusfunktsioon (resolution function) – I²C näitel

```

package I2C_defs is
  type I2C_bit is ( '0', 'Z', 'H' );
  type I2C_bit_vector is array (integer range <>) of I2C_bit;
  function resolved ( v: I2C_bit_vector ) return I2C_bit;
  -- ...
end I2C_defs;

package body I2C_defs is
  function resolved ( v: I2C_bit_vector ) return I2C_bit is
    variable r: I2C_bit := 'Z';
    type I2C_1d is array ( I2C_bit ) of I2C_bit;
    type I2C_2d is array ( I2C_bit ) of I2C_1d;
    constant resolution_table: I2C_2d := (
      -----
      -- '0'   'Z'   'H'
      -----
      ( '0', '0', '0' ),    -- '0'
      ( '0', 'Z', 'H' ),    -- 'Z'
      ( '0', 'H', 'H' ) );  -- 'H'
    begin
      for i in v'range loop    r := resolution_table ( r ) ( v(i) );    end loop;
      return r;
    end resolved;
    -- ...
  end I2C_defs;

```



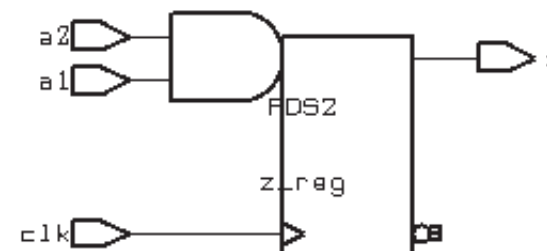
VHDL & süntees – “what you write is what you get...”

- Mitte kõik konstruktsioonid pole sünteesitavad
- Kombinatorne loogika ja mäluelemendid tuleks eraldi kirjeldada (mõtle riistvaras!)
- Ettevaatust signaalide ja muutujate segiläbi kasutamisel!

```

signal A1, A2: BIT;
-- . . .
process (CLOCK)
  variable A3: BIT;
begin
  if CLOCK='1' and CLOCK'event then
    A3 := A1 and A2;
    Z  <= A3;
  end if;
end process;

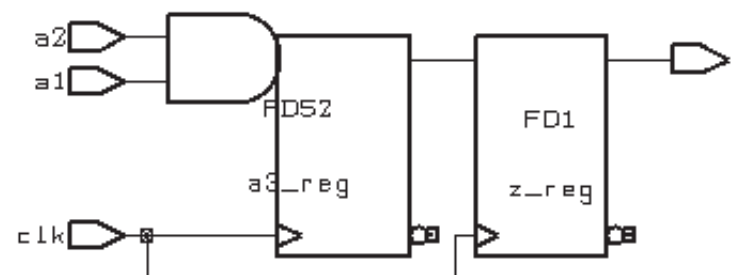
```



```

signal A1, A2, A3: BIT;
-- . . .
process (CLOCK)
begin
  if CLOCK='1' and CLOCK'event then
    A3 <= A1 and A2;
    Z  <= A3;
  end if;
end process;

```

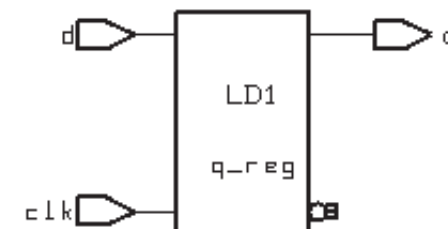


Lukk-register või frondist töötav (flip-flop)?

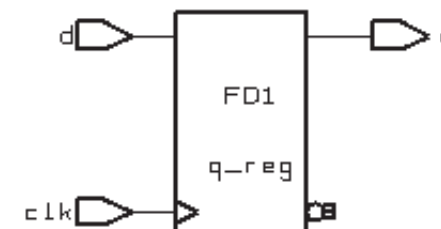
```
P1_L: process (CLK, D) begin
    if CLK='1' then    Q <= D;
    end if;
end process P1_L;
```

```
P2_FL: process (CLK) begin
    if CLK='1' then    Q<=D;
    end if;
end process P2_FL;
```

```
P1_FF: process (CLK) begin
    if CLK='1' and
        CLK'event then Q<=D;
    end if;
end process P1_FF;
```



- **Simulation OK kuid mitte süntees!**
-- Warning: Variable 'd' is being read
-- in routine .. line .. in file '..',
-- but is not in the process sensitivity
-- list of the block which begins
-- there. (HDL-179)
- **Tulemus – lukk-register**



VHDL & süntees – mõtle riistvaras!

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity test is
    port ( a, b, c: in unsigned(7 downto 0);
          x: in unsigned(2 downto 0);
          o: out unsigned(7 downto 0) );
```

```
end test;
```

```
architecture bhv2 of test is begin
```

```
process (a, b, c, x) begin
```

```
    case x is
```

```
        when "010" =>    o <= a+b;
```

```
        when "011" =>    o <= a+c;
```

```
        when "110" =>    o <= b+c;
```

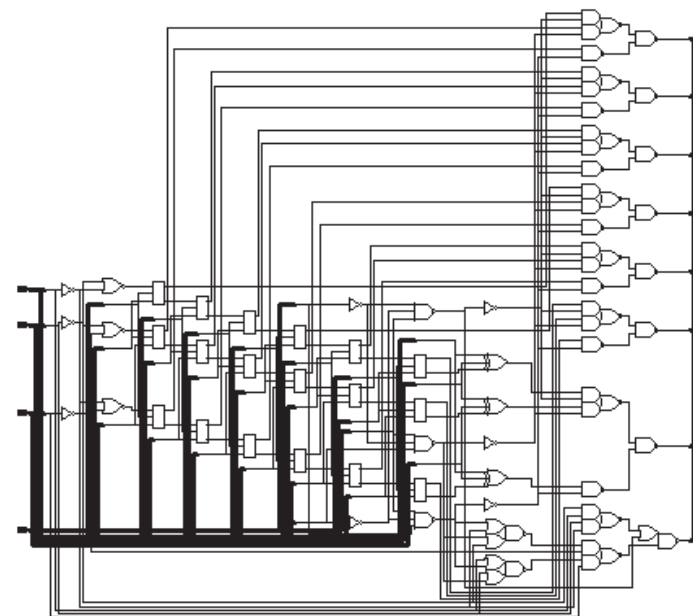
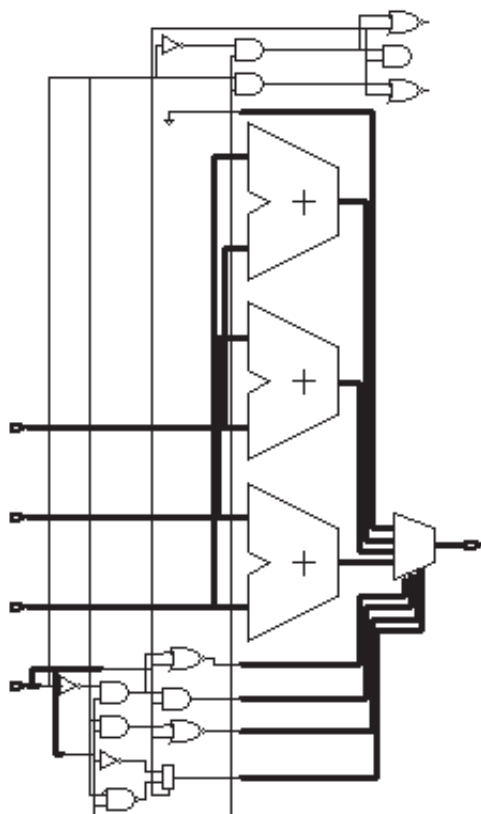
```
        when others =>
            o <= (others=>'0');
```

```
    end case;
```

```
end process;
```

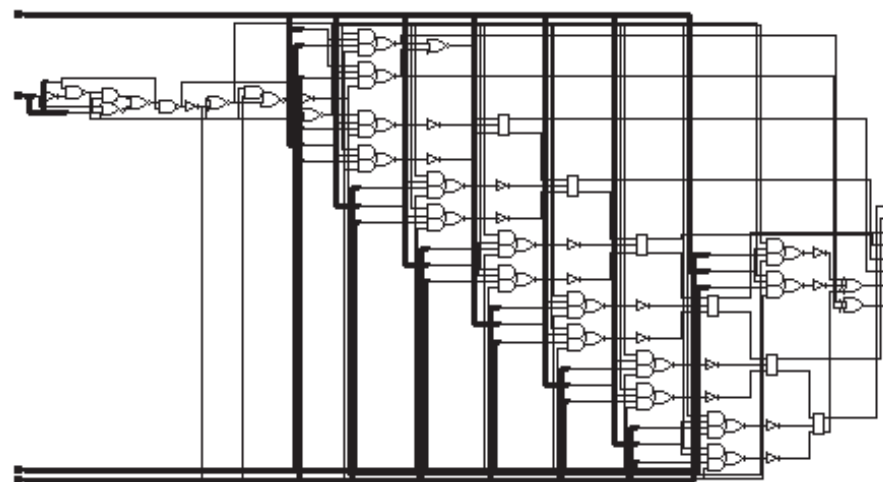
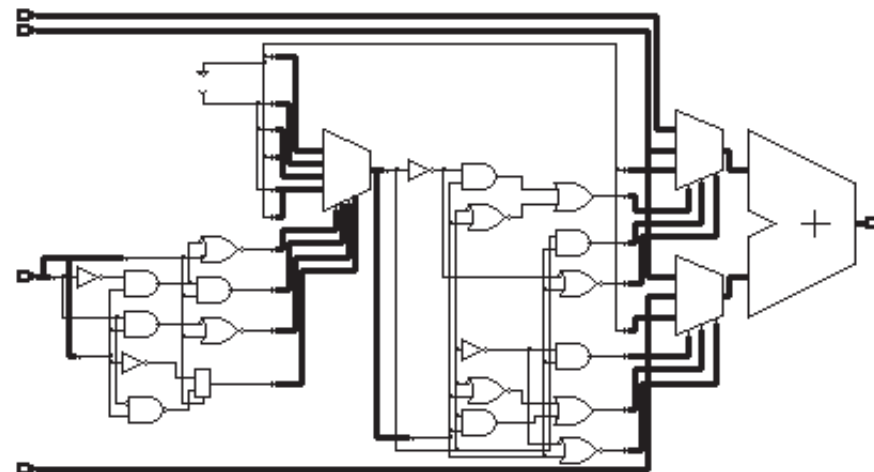
```
end architecture bhv2;
```

220 l.e. / 11.57 ns



VHDL & süntees – mõtle riistvaras!

```
architecture rtl of test is
    signal a1, a2: unsigned(7 downto 0);
    signal dc: unsigned(1 downto 0);
begin
    dec: process (x) begin
        case x is
            when "010" =>    dc <= "01";
            when "011" =>    dc <= "10";
            when "110" =>    dc <= "11";
            when others =>    dc <= "00";
        end case;
    end process dec;
    m1: process (a, b, dc) begin
        case dc is
            when "01" =>    a1 <= a;
            when "10" =>    a1 <= a;
            when "11" =>    a1 <= b;
            when others =>    a1 <= (others=>'0');
        end case;
    end process m1;
    m2: process (b, c, dc) begin
        -- ...
    end process m2;
    o <= a1 + a2;
end architecture rtl;
```



117 l.e. / 19.2 ns

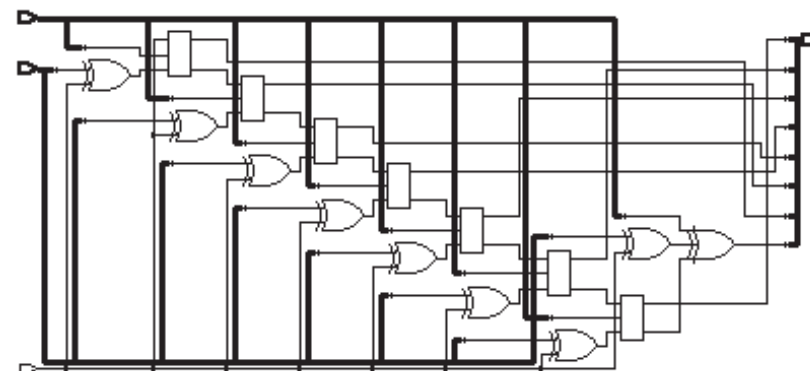
Universaalne liitja-lahutaja

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity add_sub is
    port ( a, b: in unsigned(7 downto 0);
          x: in std_logic;
          o: out unsigned(7 downto 0) );
end add_sub;
architecture bhv of add_sub is begin
process (a, b, x) begin
    if x='0' then o <= a+b;
    else          o <= a-b; end if;
end process;
end architecture bhv;
```

145 l.e. / 11.64 ns

```
architecture dfl of test5 is
    signal a1, b1, o1: unsigned(8 downto 0);
begin
    a1 <= a & '1';
    b1 <= b & '0' when x='0' else
        unsigned(not std_logic_vector(b)) &
        '1';
    o1 <= a1+b1;
    o <= o1(8 downto 1);
end architecture dfl;
```

87 l.e. / 12.45 ns



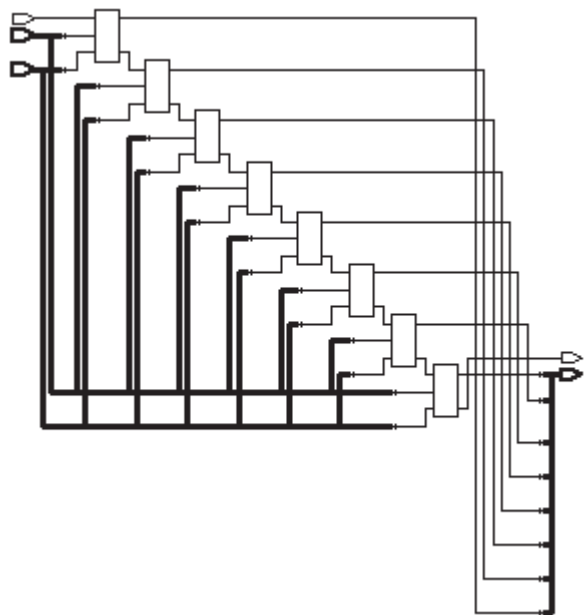
Liitjad & lahutajad

```

signal a, b, o: unsigned (7 downto 0);
signal a1,b1,o1: unsigned (9 downto 0);
-- ...
a1 <= '0' & a & '1';
b1 <= '0' & b & ci;
o1 <= a1 + b1;
o <= o1(8 downto 1);
co <= o1(9);

```

64 l.e. / 10.66 ns [60 l.e. / 10.08 ns ilma ci/co]

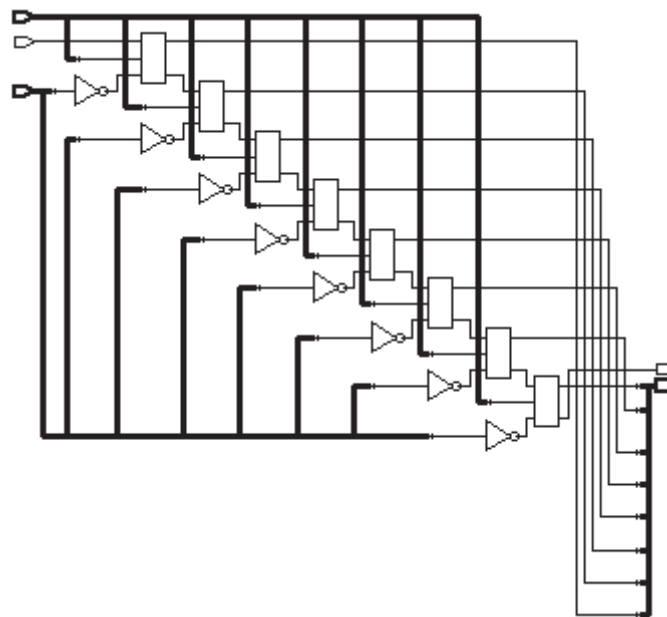


```

signal a, b, o: unsigned (7 downto 0);
signal a1,b1,o1: unsigned (9 downto 0);
-- ...
a1 <= '0' & a & '1';
b1 <= '0' &
    unsigned(not std_logic_vector(b)) & ci;
o1 <= a1 + b1;
o <= o1(8 downto 1);
co <= o1(9);

```

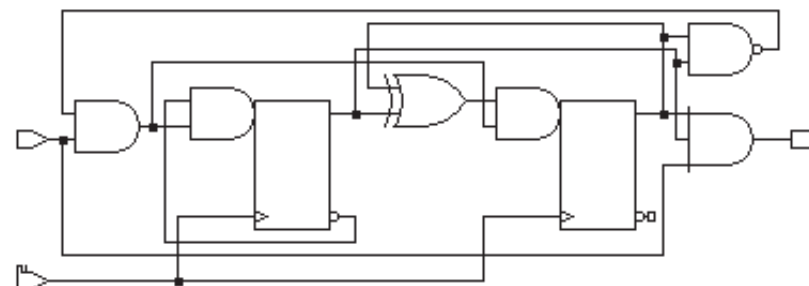
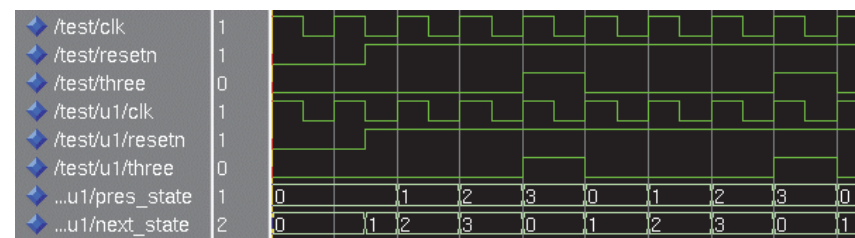
72 l.e. / 10.62 ns [66 l.e. / 10.35 ns ilma ci/co]



Automaat – kirjeldus-stiilid & süntees

Kaks protsessi (modulo-4 counter)

```
library IEEE; use IEEE.std_logic_1164.all;
entity counter03 is
    port ( clk: in bit;
          resetn: in std_logic;
          three: out std_logic );
end entity counter03;
architecture fsm2 of counter03 is
    subtype state_type is integer range 0 to 3;
    signal pres_state, next_state: state_type := 0;
begin
    process (clk) begin -- State memory
        if clk'event and clk = '1' then
            pres_state <= next_state;
        end if;
    end process;
    -- Next state & output functions
    process (resetn, pres_state) begin
        three <= '0';
        if resetn='0' then    next_state <= 0;
        else
            case pres_state is
                when 0 to 2 => next_state <= pres_state + 1;
                when 3 => next_state <= 0;    three <= '1';
            end case;
        end if;
    end process;
end architecture fsm2;
```



22 gates / 3.70 ns

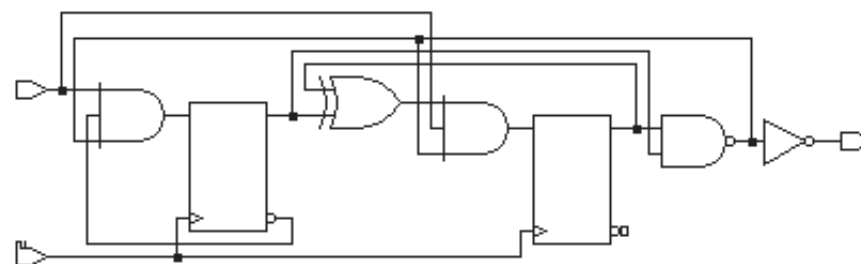
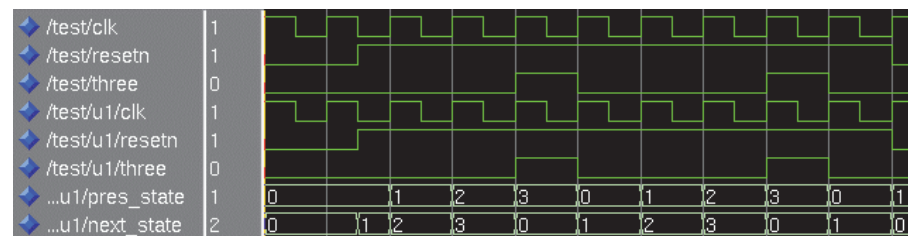
Automaat – kirjeldus-stiilid & süntees

Kolm protsessi (modulo-4 counter)

```
library IEEE; use IEEE.std_logic_1164.all;
architecture fsm3 of counter03 is
    subtype state_type is integer range 0 to 3;
    signal pres_state, next_state: state_type := 0;
begin
    process (clk) begin -- State memory
        if clk'event and clk = '1' then
            pres_state <= next_state;
        end if;
    end process;

    -- Next state function
    process (resetn, pres_state) begin
        if resetn='0' then next_state <= 0;
        else
            if pres_state=3 then next_state <= 0;
            else next_state <= pres_state + 1;
            end if;
        end if;
    end process;

    -- Output function
    process (resetn, pres_state) begin
        if pres_state=3 then three <= '1';
        else three <= '0';
        end if;
    end process;
end architecture fsm3;
```



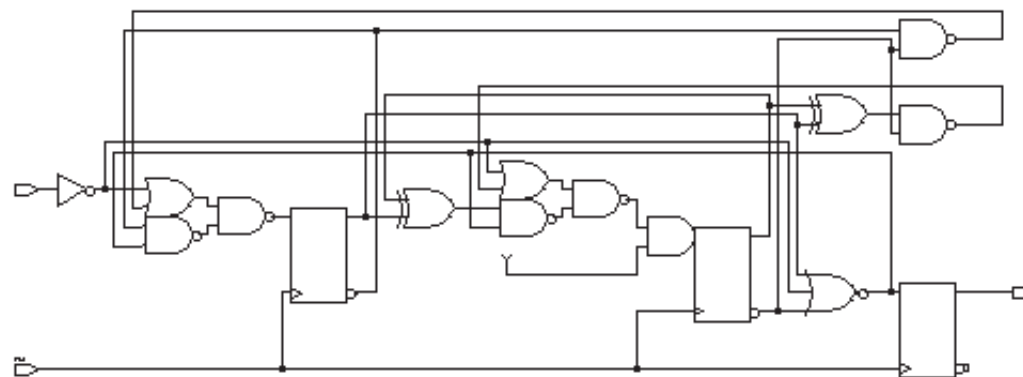
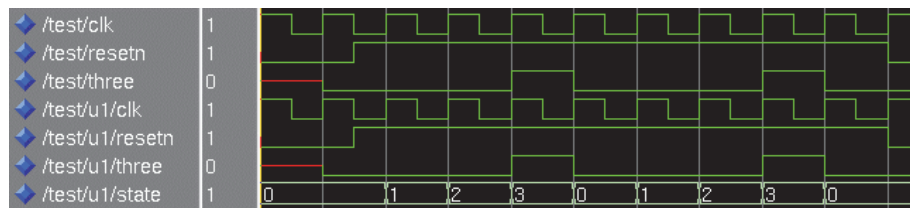
23 gates / 4.36 ns

Automaat – kirjeldus-stiilid & süntees

Üks protsess (modulo-4 counter)

```
library IEEE; use IEEE.std_logic_1164.all;
architecture fsm1 of counter03 is
    subtype state_type is integer range 0 to 3;
    signal state: state_type := 0;
begin
    process (clk) begin
        if clk'event and clk = '1' then
            three <= '0';
            if resetn='0' then      state <= 0;
            else
                case state is
                    when 0 | 1 => state <= state + 1;
                    when 2 => state <= state + 1;  three <= '1';
                    when 3 => state <= 0;
                end case;
            end if;
        end if;
    end process;
end architecture fsm1;
```

```
// Another version to build the process
process begin
    wait on clk until clk='1';
    three <= '0';
    if resetn='0' then      state <= 0;
    else
        -- and so on...
```



38 gates / 5.68 ns