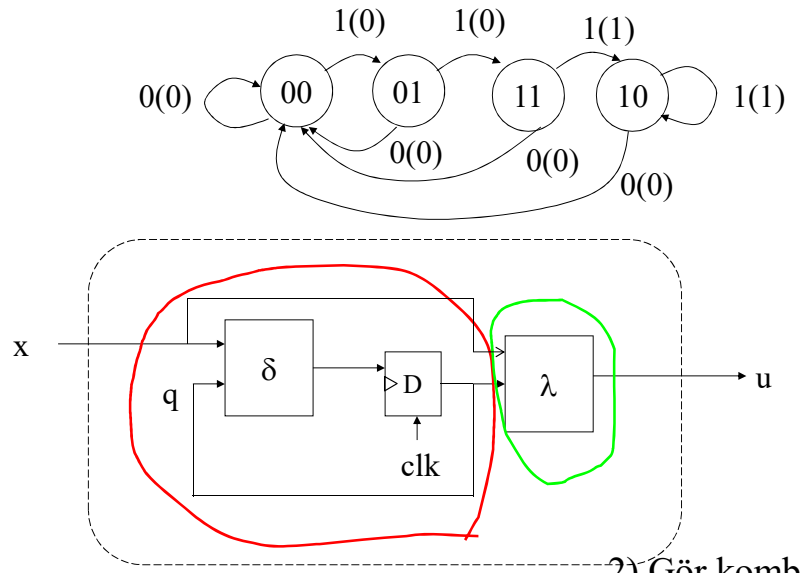
VHDL2

- Moder – portsatsen
- Datatyper
- Ett exempel, stegmotorstyrning
- Hierarkisk konstruktion, instantiering
- Kombinatorisk process
- Record, loop
- Lab3 : UART

1

Sekvensnätsexemplet



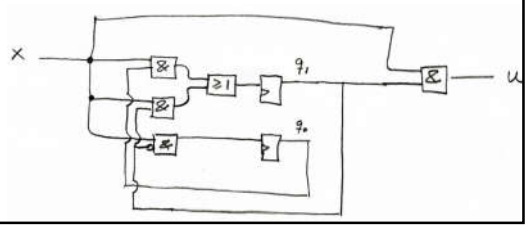
1) Gör en `process(clk)` av detta

2) Gör kombinatorik av det här!

2

2

```
entity snet is
  port(x,clk: in std_logic;
       u: out std_logic);
end entity;

architecture booleq2 of snet is
  signal q: std_logic_vector(1 downto 0);
begin
-- delta
process(clk)
begin
 if rising_edge(clk)
  case q is
    when "00" => if x='1' then q <= "01";
                 end if;
    -- överhoppade rader
    when "10" => if x='0' then q <= "00";
                 end if;
    when others => q <= "00";
  end case;
 end if;
end process;

-- lambda
u <= x and q(1);
end architecture;
```



3

VHDL beskriver hårdvara!

1. En VHDL-modul består av två delar
   - **`entity`**, som beskriver gränssnittet
   - **`architecture`**, som beskriver innehållet
2. För att göra **kombinatorik** används
   1. Booleska satser **`z <= x and y;`**
   2. **`with-select-when`**-satser
   3. **`when-else`**-satser

   } Samtidiga satser
3. För att göra sekvensnät används (en eller flera) **`process(clk)`**-satser
   1. enn **`if rising_edge(clk)`** … **`end if;`**
   2. booleska satser **`z <= x and y;`**
   3. **`case-when-satser`**
   4. **`if-then-else-satser`**

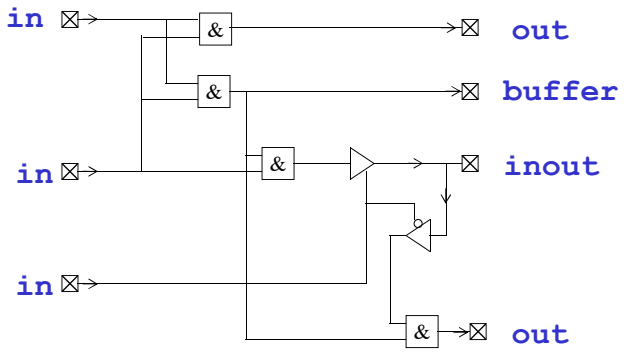
   } VL får vippa på sig
   Alla klockas samtidigt

4

4

Moder : portsatsen

OBS, om en utsignal också används inuti nätet, så ska den deklareras som `buffer`.



5

5

Datatyper

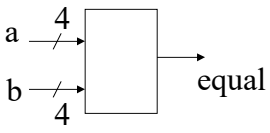
VHDL är ett **mycket starkt** typat språk! *Inbyggda* typer är bl a:

Typ	Möjliga värden	operatorer
integer	-2147483648 – 2147483647	**ABS** ** * / **MOD REM** + - (tecken) + - = /= < <= > >=
bit **boolean**	'0','1' 'false','true'	**NOT** = /= < <= > >= **AND NAND OR NOR XOR XNOR**
bit_vector	Obegränsad vektor av bit	**NOT** & **SLL SRL SLA SRA ROL ROR** = /= < <= > >= **AND NAND OR NOR XOR XNOR**

6

6

Exempel: 4-bits komparator




```
entity comp4 is
  port(a,b: in BIT_VECTOR(3 downto 0);
       equal: out BIT);
end entity;


architecture func of comp4 is
begin
  equal <= '1' when a=b else '0';
end architecture;
```

7

7

std_logic istället för bit

```
signal reset: std_logic;
```

Reset kan nu anta följande värden:

simulering
- 'U' : Uninitialized
- 'X' : Forcing Unknown

'0' : Forcing 0
'1' : Forcing 1
'Z' : High impedance ← tristate

ej i denna kurs
- 'W' : Weak Unknown
- 'L' : Weak 0
- 'H' : Weak 1

'-' : Don't care

t ex för att speca sanningstabell

8

8

std_logic forts.

För att få tillgång till std_logic skriver man följande i början av filen

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

(IEEE = International Electrical & Electronics Engineers)

Nu får du datatyperna **std_logic** och **std_logic_vector** och kan använda dem pss **bit** och **bit_vector**.

Man *kan* (men bör inte) dessutom lägga till raden

```
use IEEE.STD_LOGIC_UNSIGNED.ALL
```

Då kan man göra aritmetik med **std_logic_vector**

Användning av STD_LOGIC_UNSIGNED REKOMMENDERAS INTE!

9

9

~~STD_LOGIC_UNSIGNED~~

Nu kan du göra följande:

```
signal q: std_logic_vector(3 downto 0);  -- 4 bit ctr
...
q <= q + 1;
if q=10 ...
q <= "0011";
q(0) <= '1';
```

Dvs vi kan hantera q både som en boolesk vektor och som ett tal på intervallet [0,15];

VHDL är ett språk som har *operator overloading*.

10

10

VARNING!

Undvik om möjligt användning av:

STD_LOGIC_UNSIGNED
STD_LOGIC_ARITH
STD_LOGIC_SIGNED

Dessa bibliotek är inte IEEE standard-bibliotek,
och orsakar dessutom lätt problem vid typkonvertingar.

Använd istället:

NUMERIC_STD

11

11

NUMERIC_STD

NUMERIC_STD ger datatyperna **signed** och **unsigned** samt aritmetik (och diverse funktioner) på dessa, och vi kan skriva:

```
signal q: unsigned(3 downto 0);  -- 4 bit ctr
...
q <= q + 1;
if q=10 ...
q <= "0011";
q(0) <= '1';
```

Dvs **unsigned** är ett tal [0, 15] och kan även hanteras som en boolesk vektor.

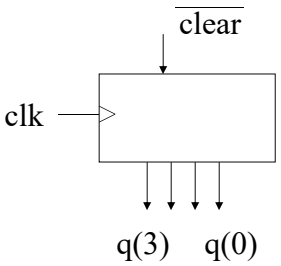
Googla på "why numeric_std is preferred" för mer info.

12

12

4-bits dekadräknare med **synkron** clear

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter is
port(clk, clear: in std_logic;
     q: buffer unsigned(3 downto 0));
end entity;

architecture func of counter is
begin
  process(clk)
  begin
    if rising_edge(clk) then
       if clear='0' then
          q <= "0000";
       elsif q=9 then
          q <= "0000";
       else
          q <= q + 1;
       end if;
    end if;
  end process;
end architecture;
```



13

13

Vi rekommenderar ang. typer

- Använd endast
  - **std_logic** och
  - **std_logic_vector (använd bara som vektor)**
  - **unsigned (använd som vektor + aritmetik)**
- Vill ni räkna inkludera
  - **NUMERIC_STD**
- Skippa **integer**. Går ej att indexera på bit-nivå.
- Skippa **bit**. Tristate och don't care saknas.

14

14

Uppräknade datatyper

Vi kan skapa egna datatyper. Det kan vara händigt vid konstruktion av sekvensnät med symboliska namn på tillstånden:

```
type state is   (odd,even);
...
signal q: state;
...
-- ett delta-nät
case q is
  when odd => if x='1' then q <= even;
                       else q <= odd;
              end if;
  when even => if x='1' then q <= odd;
                        else q <= even;
               end if;
end case;
```

15

15

Några praktiska småsaker

Konstanter

```
signal bus: unsigned(3 downto 0);
constant max: unsigned(3 downto 0) := "1111";
...
if bus = max then ...
```

Alias

```
signal address: unsigned(31 downto 0);
alias top_ad: unsigned (3 downto 0)
                        is address(31 downto 28);
```

Concatenation

```
signal bus: unsigned(1 downto 0);
signal a,b: std_logic;
bus <= a & b;
```

16

16

En adderare

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity adder is
port(a,b: in UNSIGNED(3 downto 0);
     s: out UNSIGNED(4 downto 0));
end entity;

architecture func of adder is
begin
-- zero extension
  s <= ('0'& a) + ('0'& b);
-- sign extension
-- s <= (a(3) & a) + (b(3) & b);
end architecture;
```

17

17

Asynkron/synkron reset?

```
process(clk,rst)
begin
 if rst='1' then
  q <= '0';
 elsif rising_edge(clk) then
  q <= q and x;
 end if;
end process;
```

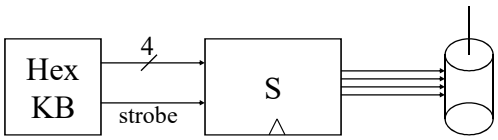
```
process(clk)
begin
 if rising_edge(clk) then
  if rst='1' then
    q <= '0';
  else
    q <= q and x;
  end if;
 end if;
end process;
```

18

18

Exempel: Stegmotorstyrning

(ur Lennart Bengtsson: *Digitala system*)

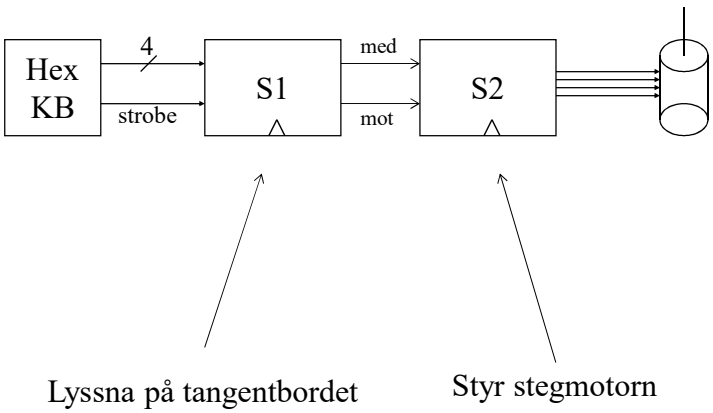


- För varje ändring enligt :
  0001->1001->1000->1010->0010->0110->0100->0101->
  rör sig motorn ett steg medurs.
- Första knapptryckningen anger riktning
  (0=medurs,1=moturs),
  andra tryckningen anger antal steg.

19

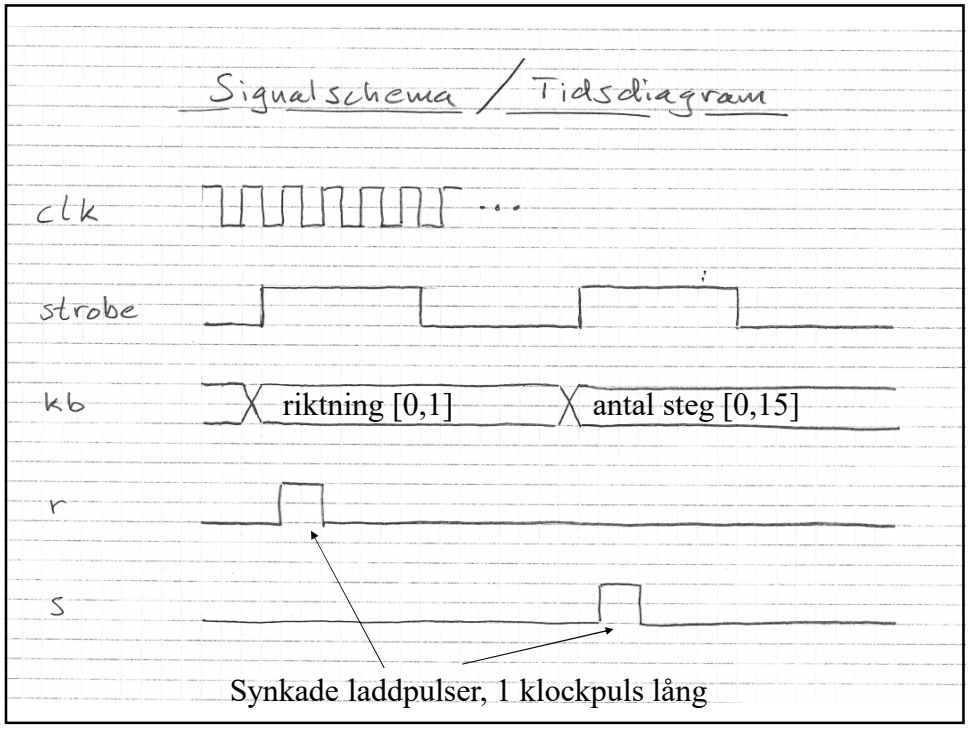
19

Dela upp nätet

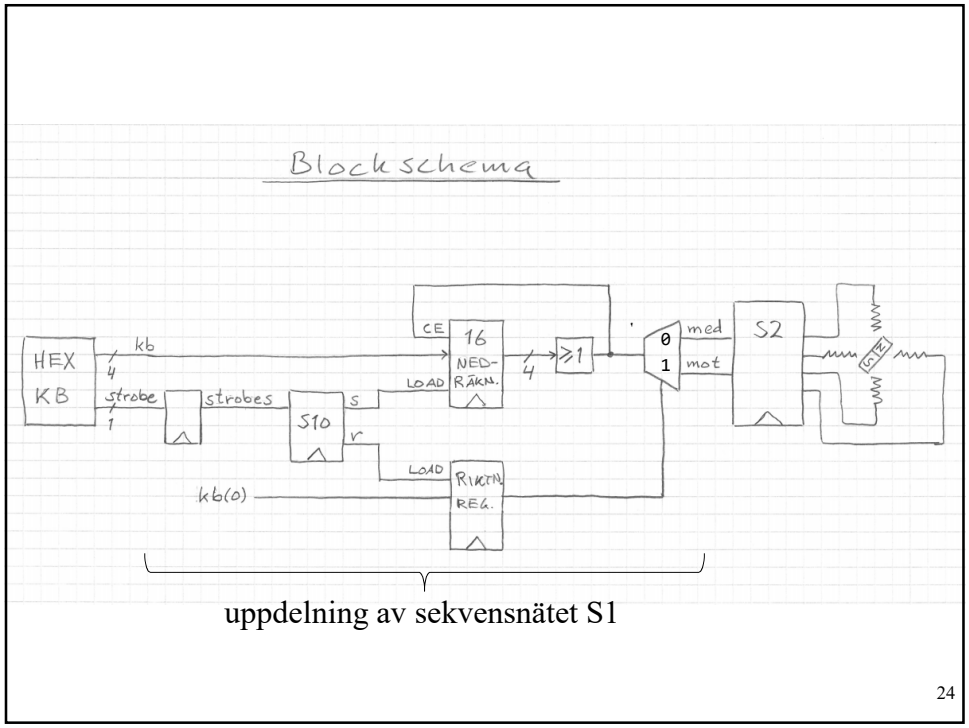


20

20

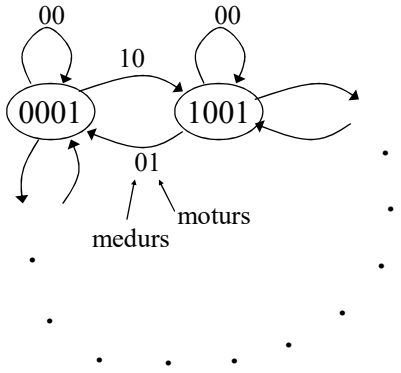

Signalschema / Tidsdiagram
clk
strobe
kb
riktning [0,1]
antal steg [0,15]
r
s
Synkade laddpulser, 1 klockpuls lång

22

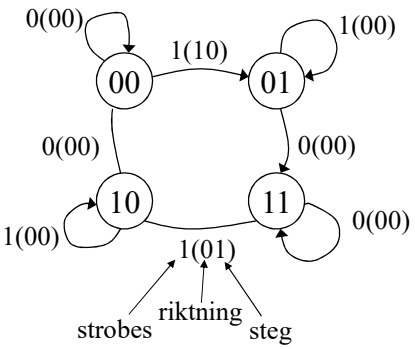

Blockschema
HEX
KB
kb
4
strobe
1
strobes
S10
s
r
CE
16
NED-
RÄKN.
LOAD
4
≥1
0
1
med
mot
S2
LOAD
RIKTN
REG.
kb(0)
uppdelning av sekvensnätet S1
24

24

Nu kan vi konstruera S2 direkt. Utgångarna ska vara hasardfria, så vi gör en Mooremaskin med U=Q.



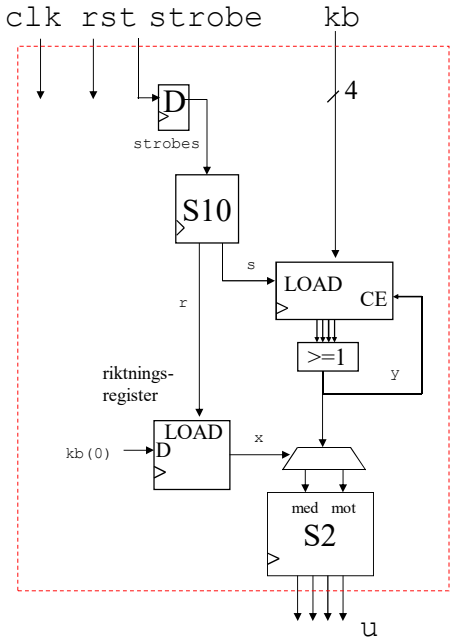
Strobeseparatorn S10



25

25

Blockschema => VHDL

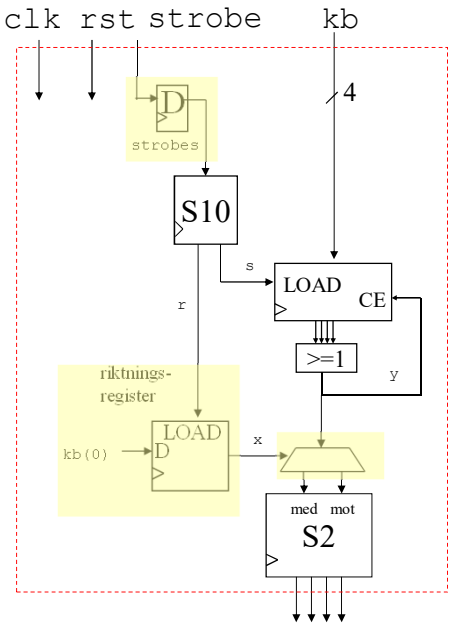



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity stegmotor is
  port(kb: in unsigned(3 downto 0);
       clk,strobe,rst: in std_logic;
       u: out unsigned(3 downto 0));
end entity;


architecture func of stegmotor is
  signal s,r,x,y,strobes: std_logic;
  signal s10: unsigned(1 downto 0);
  signal q: unsigned(3 downto 0);
  signal: med,mot: std_logic;
begin

-- hela vår konstruktion

end architecure;
```

26

26

Blockschema => VHDL

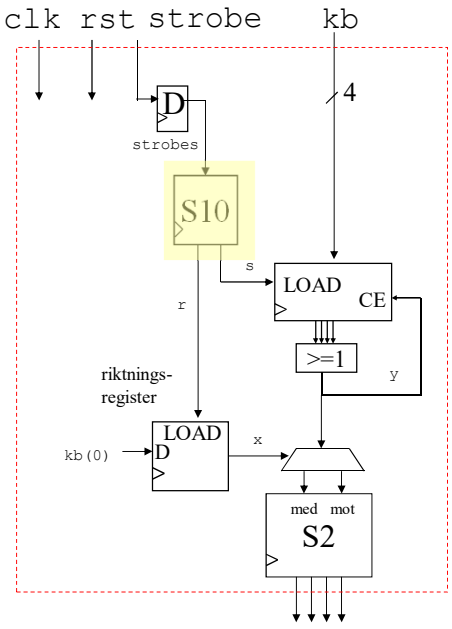


Vi börjar med dom små blocken...

Blockschema => VHDL

```
-- synkvippa
  sync: process(clk)
  begin
    if rising_edge(clk) then
      strobes <= strobe;
    end if;
  end process sync;


  -- riktningsregister
  riktning: process(clk)
  begin
    if rising_edge(clk) then
      if rst='1' then
        x <= '0';
      elsif r = '1' then
        x <= kb(0);
      end if;
    end if;
  end process riktning;


-- motorstyrning
  med <= not x and y;
  mot <= x and y;
```

Blockschema => VHDL



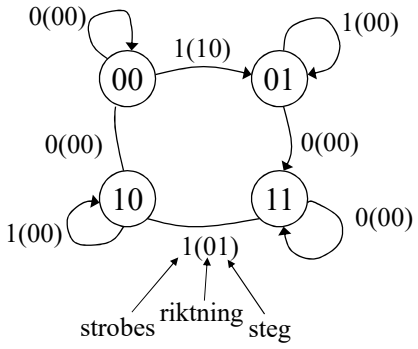
Sekvensnätet S10 ...

29

29

```
-- strobeseparator S10
  kbfix: process(clk)
  begin
    if rising_edge(clk) then
    case s10 is
      when "00"  => if strobes='1' then
                      s10 <= "01";
                    end if;
      when "01"  => if strobes='0' then
                      s10 <= "11";
                    end if;
      when "11"  => if strobes='1' then
                      s10 <= "10";
                    end if;
      when "10"  => if strobes='0' then
                      s10 <= "00";
                    end if;
      when others => null;
    end case;
    end if;

    if rst='1' then
      s10 <= "00";
    end if;
  end process kbfix;


-- ut från S10
r <= '1' when (s10=0 and strobes='1') else '0';
s <= '1' when (s10=3 and strobes='1') else '0';
```

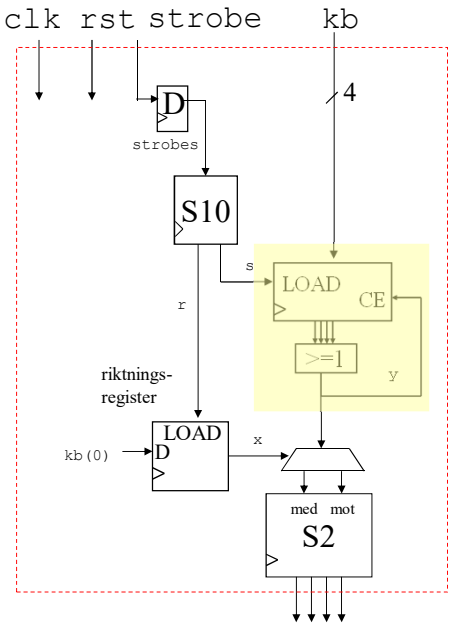
Strobeseparatorn S10



30

30

Blockschema => VHDL



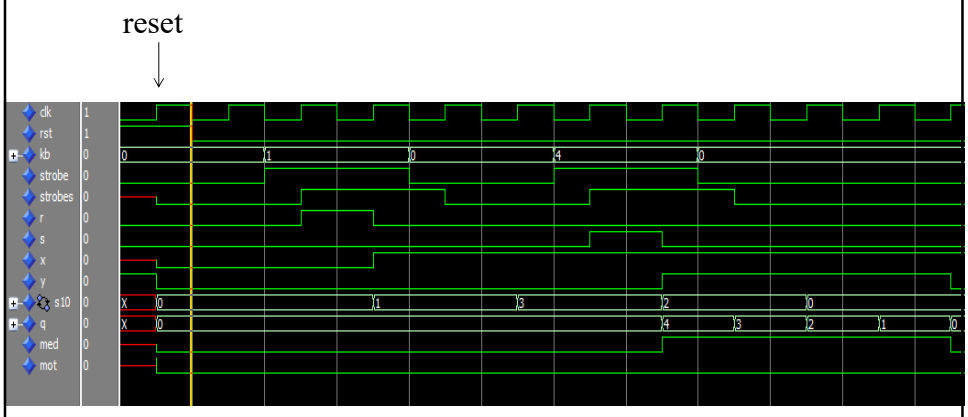
Räkneverket ...

```
-- räknaren
  ctr16: process(clk)
    begin
      if rising_edge(clk) then
        if rst='1' then
          q <= "0000";
        elsif s='1' then
          q <= kb;
        elsif q>0 then
          q <= q-1;
        end if;
      end if;
    end process ctr16;

  -- ut från räknaren
  y <= '0' when q=0 else '1';

 -- S2 hemuppgift
end architecture;
```

31

31

Modelsim



http://www.isy.liu.se/edu/kurs/TSEA22/lektion/Modelsim.pdf
http://www.isy.liu.se/edu/kurs/TSEA56/Manualer/SE_tutor.pdf

32

32

VHDL-kod + testbänk

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Decoder is
  Port (clk : in  STD_LOGIC;
        strobe: in  STD_LOGIC;
        q : out  UNSIGNED (3 downto 0));
end Decoder;

architecture func of Decoder is

signal x, y :STD_LOGIC;

begin

  process(clk) begin
    if rising_edge(clk) then
      code
      code
    end if;
  end process;

  code
  code

end architecture;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Decoder_tb is
end Decoder_tb;

architecture func of Decoder_tb is

  component Decoder
    Port (clk : in STD_LOGIC;
          strobe : in STD_LOGIC;
          q : out UNSIGNED (3 downto 0));
  end component;

  -- Testsignaler
  signal clk : STD_LOGIC;
  signal strobe : STD_LOGIC;
  signal q : UNSIGNED(3 downto 0);

begin

  uut: Decoder PORT MAP(
    clk => clk, strobe => strobe, q => q);

  -- Klocksignal 10MHz
  clk <= not clk after 50 ns;
  strobe <= '0', '1' after 1 us, '0' after 2 us;
end;
```

33

33

VHDL-kod + testbänk

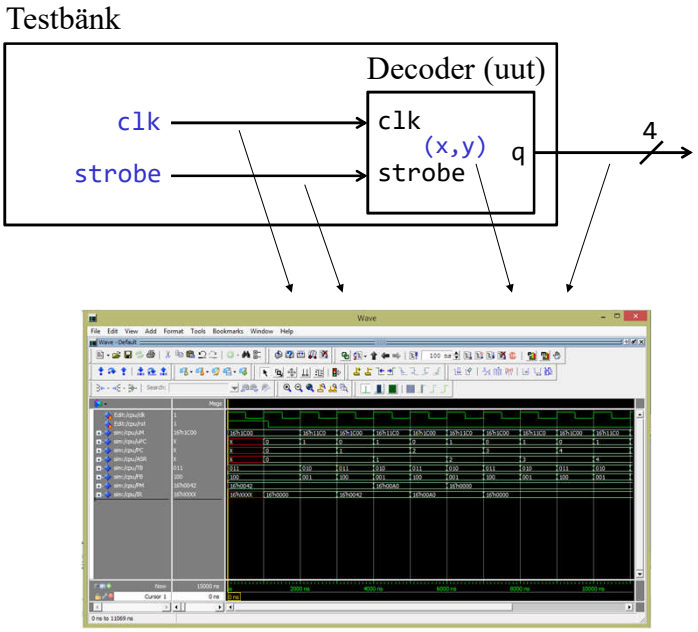
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Decoder is
  Port (clk : in  STD_LOGIC;
        strobe: in  STD_LOGIC;
        q : out  UNSIGNED (3 downto 0));
end Decoder;

architecture func of Decoder is

signal x, y :STD_LOGIC;

begin

  process(clk) begin
    if rising_edge(clk) then
      code
      code
    end if;
  end process;

  code
  code

end architecture;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Decoder_tb is
end Decoder_tb;

architecture func of Decoder_tb is

  component Decoder
    Port (clk : in STD_LOGIC;
          strobe : in STD_LOGIC;
          q : out UNSIGNED (3 downto 0));
  end component;

  -- Testsignaler
  signal clk : STD_LOGIC;
  signal strobe : STD_LOGIC;
  signal q : UNSIGNED(3 downto 0);

begin

  uut: Decoder PORT MAP(
    clk => clk, strobe => strobe, q => q);

  -- Klocksignal 10MHz
  clk <= not clk after 50 ns;
  strobe <= '0', '1' after 1 us, '0' after 2 us;
end;
```

34

34

VHDL-kod + testbänk



35

35

Slutsatser

- Vi ritade blockschema först!
- Vi har samma struktur på koden som på blockschemat!
  - Alltså: små processer som precis motsvarar ett block.
  - Vi har bra koll på mängden hårdvara

För CPLD

Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
10/108 (10%)	40/540 (8%)	8/108 (8%)	9/69 (14%)	19/216 (9%)

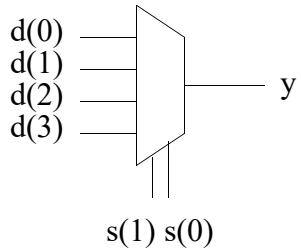
36

36

Hierarkisk konstruktion

Exempel: vi bygger en 8/1-mux mha 2 st 4/1-muxar.

Vi använder 4-1-muxen som komponent:



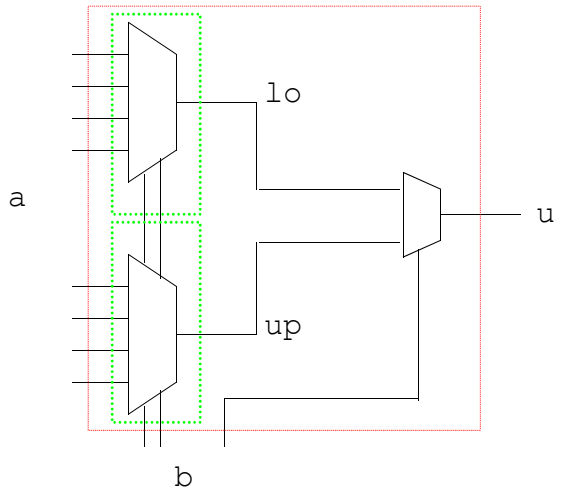
- Kod finns i fö VHDL1
- Flera olika architectures

37

37

En 8-1 mux

```
architecture func of mux4 is
begin
    y <= d(0) when s = "00" else
         d(1) when s = "01" else
         d(2) when s = "10" else
         d(3);
end architecture;
```



38

38

En 8-1 mux

```
entity mux8 is
  port( a: in unsigned(7 downto 0);
        b: in unsigned (2 downto 0);
        u: out std_logic);
end entity;

architecture func of mux8 is
  -- deklarera komponenten mux4
  -- deklarera lokala signaler
begin
  -- instantiera en mux4
  -- instantiera en mux4 till
  -- och några grindar och koppla ihop mux4:orna
end architecture;
```

39

39

8-1 mux

```
architecture func of mux8 is
  entity mux8 is port( ...
        ...);
  end entity;
  -- deklarera 4-1 mux-komponenten
  component mux4 is
    port( d: in unsigned (3 downto 0);
          s: in unsigned (1 downto 0);
          y: out unsigned);
  end component;
  -- och några lokala signaler
  signal lo,up: std_logic;
begin
  -- instantiera två mux4:or
  m1: mux4 port map(a(3 downto 0), b(1 downto 0),lo);
  m2: mux4 port map (a(7 downto 4), b(1 downto 0),up);

  -- och koppla ihop dem
  u <= lo when b(2)='0' else hi;
end architecture;
```

40

40

Vi rekommenderar

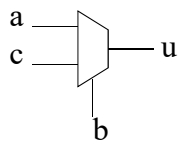
- Föregående exempel lönar sig knappast. Det är ju enklare att skriva om 4-muxen till en 8-mux.
- Hierarkisk konstruktion blir dock lämplig för stora byggen, tex processorer.
- **component**-satsen används även i testbänkar.
- om man vill simulera ett bygge bestående av flera CPLD-er är **component**-satsen mycket bra att ha!

41

41

Kombinatoriska processer

använd bara om du måste



OBS! Alla insignaler!!!
Ej klockan!

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

```
u <= c when b='1' else a;
```

Den Booleska funktionen måste vara specad för alla insignalkombinationer!

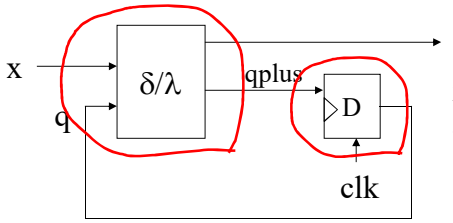
u = f(a,b,c)

42

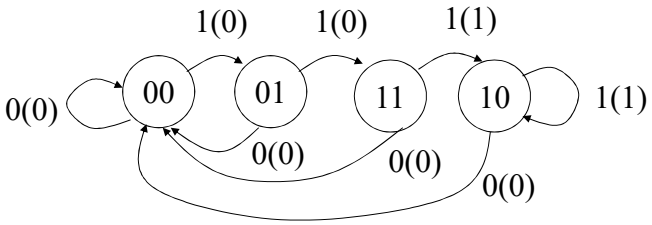
42

Sekvensnät2

1) Gör en `process(x,q)` av detta



2) Gör en `process(clk)` av detta



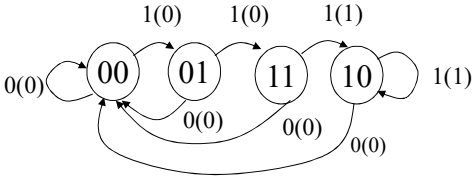
43

43

Sekvensnät2

Samma skal som förra gången

```
-- tillståndsvipporna
process(clk)
begin
 if rising_edge(clk)
   q <= qplus;
 end if;
end process;
```




```
-- delta och lambda
process(x,q)
begin
  u <= '0';   -- default-värden
  qplus <= "00";
  if x='1' then
    if q="00" then
      qplus <= "01";
    elsif q="01" then
      qplus <= "11";
    elsif q="11" then
      qplus <= "10";
      u <= '1';
    elsif q="10" then
      qplus <= "10";
      u <= '1';
    end if;
  end if;
end process;
```

Då vi förutsätter att u <= '0' och qplus <= "00" så behöver endast avvikelser från det anges i if-satsen i den kombinatoriska processen.

44

44

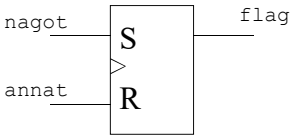
Ett varningsord 1: Multipel tilldelning

fel: mjukvarutänk

```
One: process(clk)
...
if nagot then
  flag <= '1';
end if;
end process one;


Two: process(clk)
...
if annat then
  flag <= '0';
end if;
end process two;
```

rätt: hårdvarutänk

```
Three: process(clk)
...
if nagot then
  flag <= '1';
elsif annat then
  flag <= '0';
end if;
end process three;
```

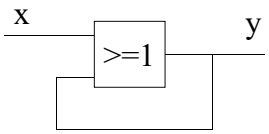


45

45

Latch

- Oönskat (oklockat, asynkront) minneselement pga kombinatorisk loop
- Kan uppstå pga:
  - Ofullständigt specad kombinatorik
  - Ihopkoppling av Mealynät



x	y
0	0
1	1
-	1

46

46

Ett varningsord 2: Oönskade latchar

Vid **select**-sats och **case**-sats kräver VHDL att alla fall täcks!

Det är inte nödvändigt vid **if**-sats och **when**-sats!
Ibland är detta bra och ibland är det förskräckligt dåligt.
För de fall som inte täcks bibehålls föregående utsignal.

	Sekvensnät (inuti klockad process)	Kombinatorik?

Ofullst.	`if count='1' then`
	`  q <= q+1;`
	`end if;`
		`u <= y when s(1) = '1' else`
		`  x when s(0) = '1';`

Fullst.	`if count='1' then`
	`  q <= q+1;`
	`else`
	`  q <= q;`
	`end if;`
		`u <= y when s(1) = '1' else`
		`  x when s(0) = '1' else`
		`  '0' when others;`

47

47

record

```
type controlword is record
     alu: unsigned(3 downto 0);
     tobus: unsigned(2 downto 0);
     halt: std_logic;
end record;
type styrminne is array(0 to 31) of controlword;

signal styr1, styr2: controlword;
signal mm: styrminne;
--
styr1.halt <= '0';
styr1.alu <= "1011";
styr1.tobus <= styr2.tobus;
--
mm(3) <= ("1011","111",'0');
```

49

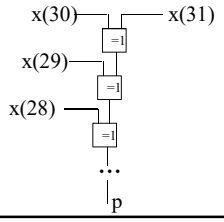
49

Lite överkurs - loop

Vi har en buss x, med 32 ledningar. Vi vill bilda paritet mellan alla ledningarna. Loopen beskriver på ett kompakt sätt det kombinatoriska nätet!

```
entity parity is
    port ( x : in  UNSIGNED (31 downto 0);
           pout : out  STD_LOGIC);
end entity;

architecture func of parity is
begin
  -- kombinatoriskt nät
  process(x)
    variable p: std_logic := '0';
  begin
    for i in 31 downto 0 loop
        p := p xor x(i);
    end loop;
    if p='1' then
      pout <= '1';
    else
      pout <= '0';
    end if;
  end process;

end architecture;
```

p är en variabel
⇒Ingen HW
⇒ vanlig tilldelning

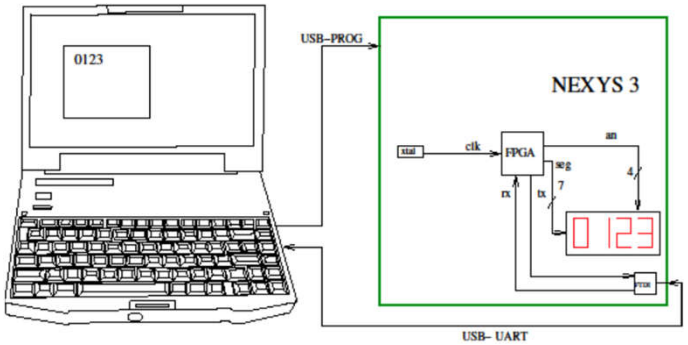


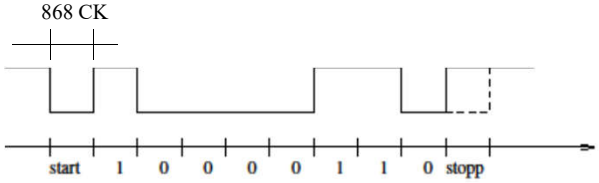
50

50

Lab3: UART

clk = 100 MHz ← enda tillåtna klockan





52

52