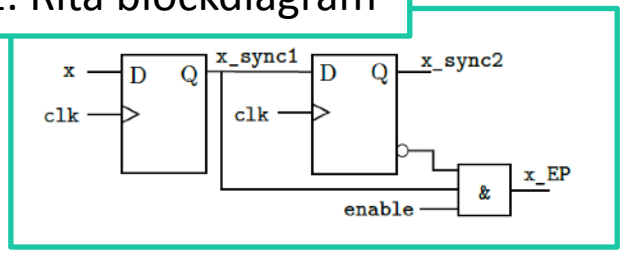
Programmerbara kretsar och VHDL

Föreläsning 9
Digitalteknik
Mattias Krysander
Institutionen för systemteknik

LINKÖPINGS UNIVERSITET

Programmerbara kretsar och VHDL

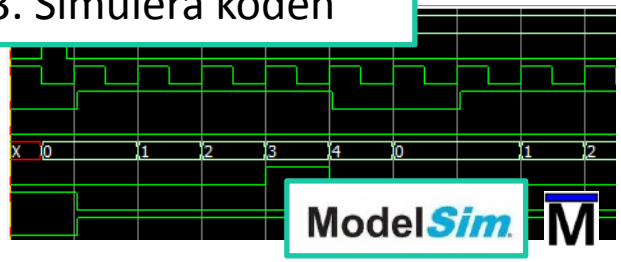
1. Rita blockdiagram



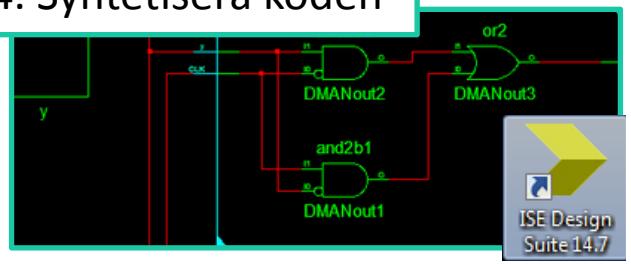
2. Skriv VHDL-kod

```
process(clk) begin
  if rising_edge(clk) then
    x_sync1 <= x;
    x_sync2 <= x_sync1;
  end if;
end process;
x_EP <= x_sync1 and (not x_sync2) and enable;
```

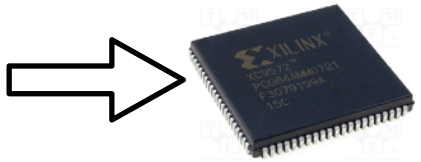
3. Simulera koden



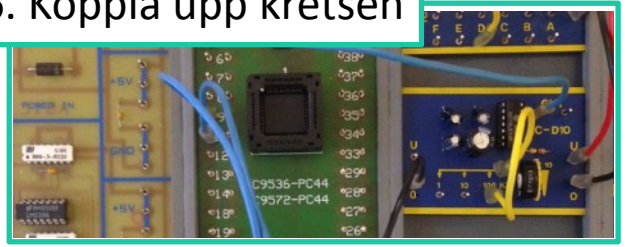
4. Syntetisera koden



5. Bränna CPLDn



6. Koppla upp kretsen



Kursinformation VHDL-delen

- Lektion 7 : Datorlektion visar arbetsgången och verktygskedjan för att skapa programmerbara kretsar.
  - Obligatoriskt att genomfört före lab 3.
- Lab 3 : Programmerbara kretsar VHDL+Modelsim+ Xilinx, Material:
  - Labkompendium (speciellt för lab3)
  - Kod, innehåller för varje uppgift:
    - Kodskelett `uppgift.vhd` där ni ska fylla i er kod.
    - Testbänkar `uppgift_tb.vhd` definierar simulering och tester av kretsens funktion.
    - Do-filer `first_time.do/rerun.do` använder ovanstående filer för att kompilera, simulera och testa er kod. Använd för att felsöka och verifiera er kod som ni skrivit i `uppgift.vhd`. Mer info på datorlektionen.
  - Video (konfigurera/bränna chip och uppkoppling)
  - Digitaltekniska byggblock [pdf]
- Program: finns på datorer i Grinden, Muxen3, Muxen4
  - Modelsim SE-64 10.1b -> Modelsim (simulering)
  - ISE Design Suite 14.7 -> 64-bit Project Navigator (syntetisering)

Fjärrinloggning (alternativ till öppet lab)

- RDP-instruktioner: https://rdpklienter.edu.liu.se
- Din dator behöver ha en RDP-klient
  - Windows: förinstallerat
  - Mac: Installera t ex Microsoft remote desktop
  - Linux: Installera t ex Remmina
- Viktigt: När man RDP:ar in till en dator låser man datorn för alla andra användare.
  - Ingen annan kan använda datorn varken i salen eller logga in remote
  - **Datorerna i GRIN, MUX3 och MUX4 får bara användas på icke-schemalagd tid.**
  - Kontrollera att ni loggar ut korrekt innan ni bryter RDP-uppkopplingen.

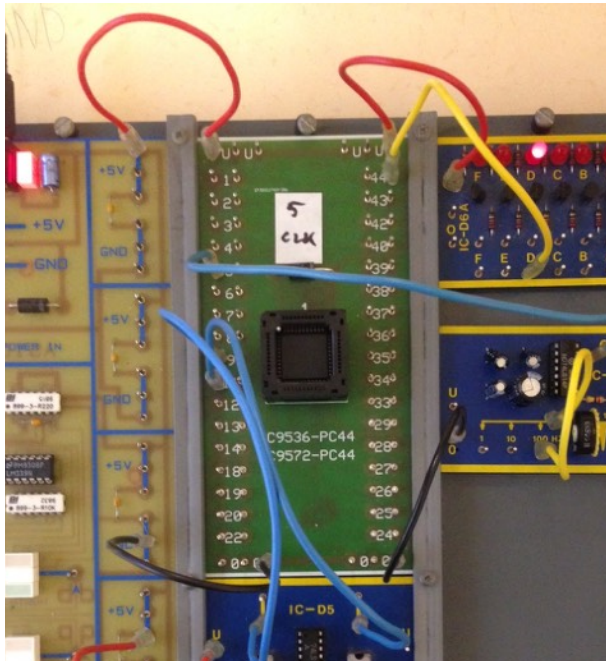
Dagens föreläsning

- Programmerbara kretsar och VHDL
- Introducerande VHDL-exempel
- Datatyper
- VHDL för Kombinationskretsar
  Grindar, Multiplexer, Komparator
- VHDL för sekvenskretsar
  D-vippor, Enpulsare, Räknare
- Exemplifierad arbetsgång
  Ett labliknande exempel

Programmerbara kretsar och VHDL

Programmerbara kretsar

- Istället för att koppla ihop grindar eller konstruera egna integrerade kretsar så finns det kretsar vars funktion kan programmeras
- Programmable logic device (PLD)
- Olika tekniker har använts
  - PROM kan användas för godtycklig funktion
  - Idag används primärt CPLD eller FPGA

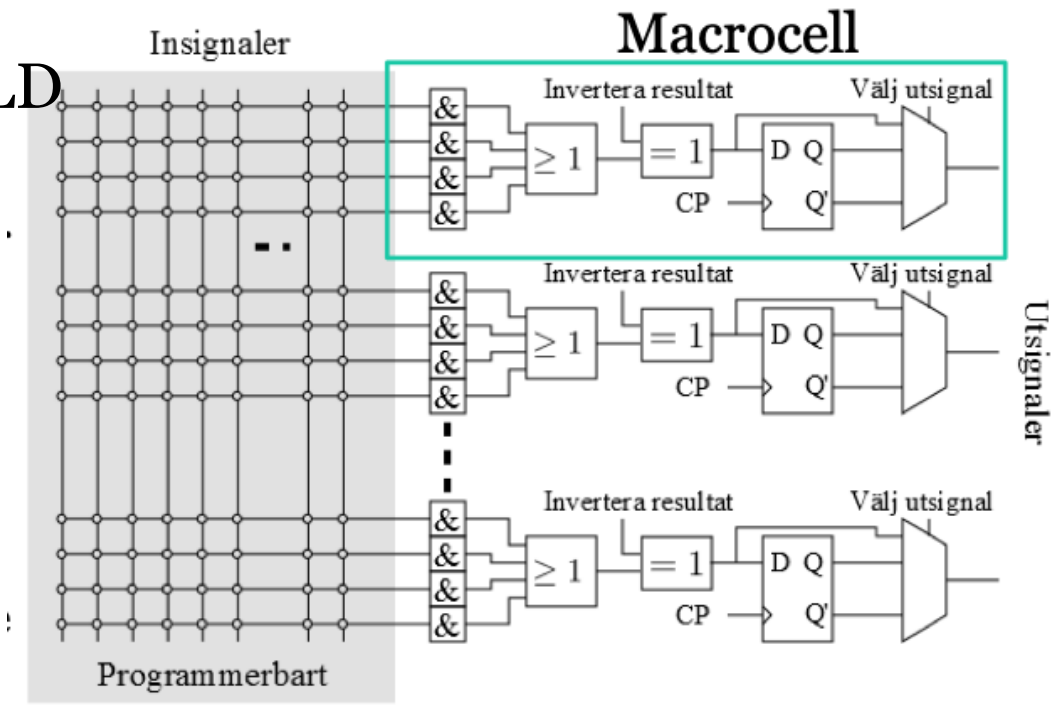


Programmerbara kretsar

- PLD = Programmable Logic Device
- CPLD = Complex PLD,
  i princip flera PLD-er på ett chip
  ex: 108 vippor + 540 produkttermer
- FPGA = Field Programmable Gate Array
  - 5 000 000 vippor
  - 2 000 000 Look-up-tables (LUT)
  - 500 Mb RAM
  - Digital Signal Processing (DSP)
  - Processor

CPLD - konstruktion

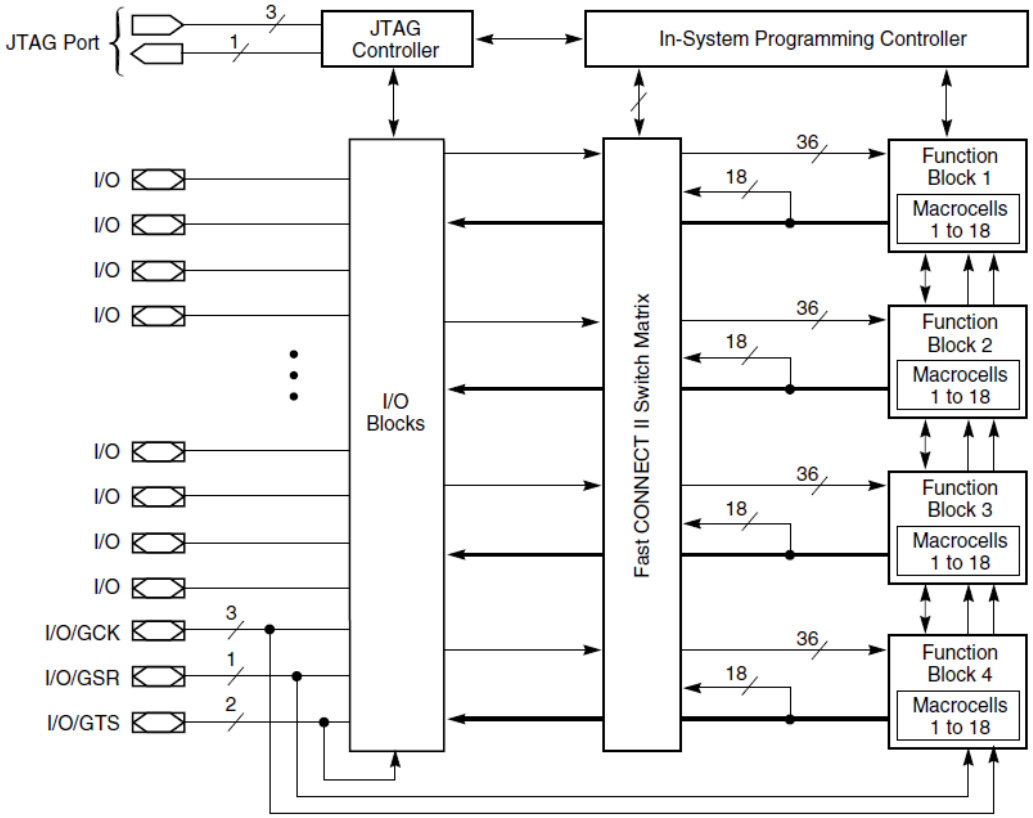
- Grundblocket i en CPLD består oftast av ett AND-OR-nät
- AND-grindarnas ingångar är programmerbara



CPLDn på labben (Xilinx XC9572)

Fyra 36V18-block

- 18 utsignaler från 36 insignaler
- Switchmatris för att koppla ihop in- och utgångar samt in- och utsignaler till/ från blocken
- Bild från datablad

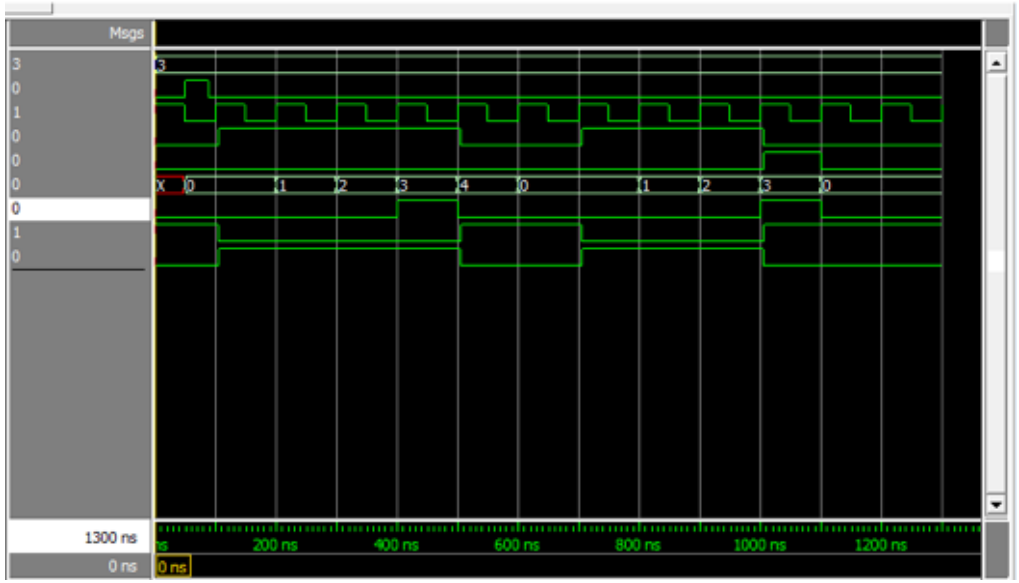


VHDL (Very high speed integrated circuit Hardware Description Language)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;              -- paket för att räkna

entity counter is
   port(clr, clk, ce, reset: in std_logic;
        q: out std_logic_vector(3 downto 0);
        rco: out std_logic);
end entity;

architecture rtl of counter is
  signal q_int : unsigned(3 downto 0); -- typ för att räkna

begin
-- tillståndsuppdatering
process(clk,reset) begin
  if (reset = '1') then                -- asynkron reset
     q_int <= to_unsigned(0,4);
  elsif rising_edge(clk) then
    if (clr = '1') then                -- synkron clear
      q_int <= to_unsigned(0,4);
    elsif (ce = '1') then              -- count enable
      if q_int = 9 then
        q_int <= "0000";
      else
        q_int <= q_int + 1;            -- aritmetisk operation
      end if;
    end if;
  end if;
end process;
```

VHDL beskriver hårdvara!

Simulera
(ModelSim)



Syntetisera
(Xilinx)

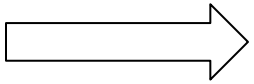
Hårdvara

VHDL beskriver hårdvara

Nytt sätt att tänka

- Lätt att hamna i mjukvarutänkande!
- FPGA-n, CPLD-n är inte en processor för VHDL
- Tilldelning, variabler betyder inte samma sak som i andra prog.språk
- Gör så här:

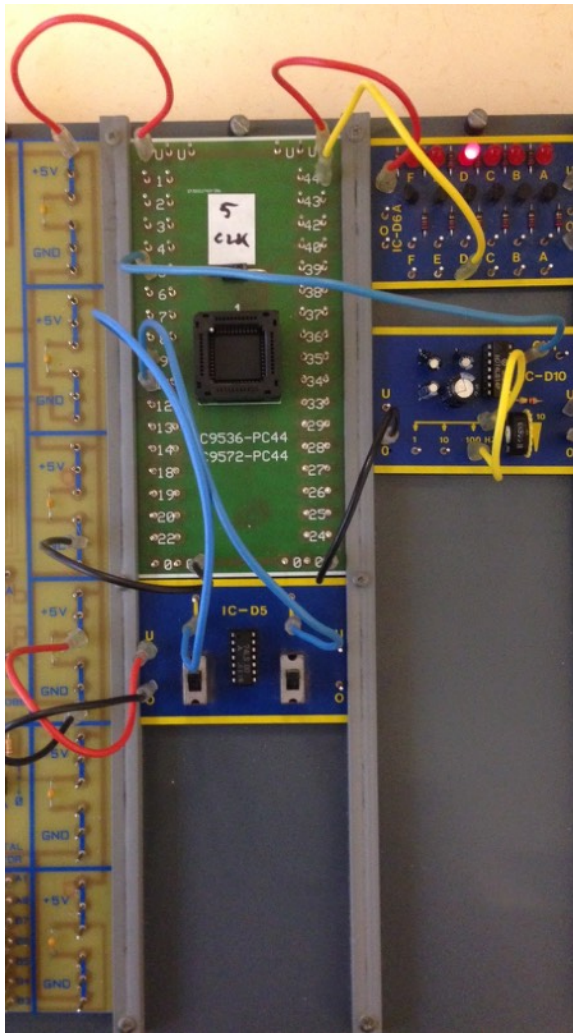
Tänk hårdvara och gör ett blockschema



Översätt till VHDL

Konstruktion med CPLD

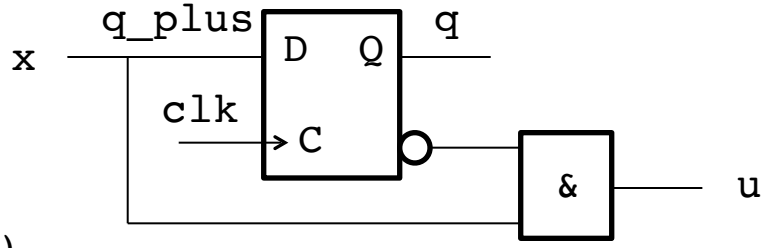
- Rita kretsschema
- Översätt till VHDL (vhd-fil)
- Simulera kretsen i Modelsim
- Syntetisera i Xilinx (skapa en jed-fil)
  - passar in (optimerar) kretsen på den aktuella CPLD:n
  - bestämmer vilka in och utgångar som kommer att användas
- Programmering (använd jed-filen)
  - speciell mjuk- och hård-vara används för att programmera CPLD:n



Introducerande VHDL-exempel

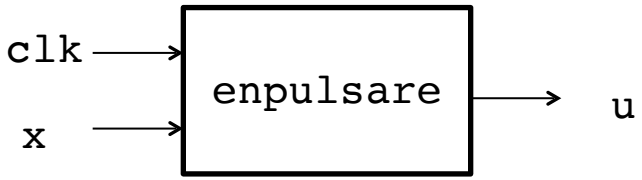
VHDL-exempel - enpulsaren

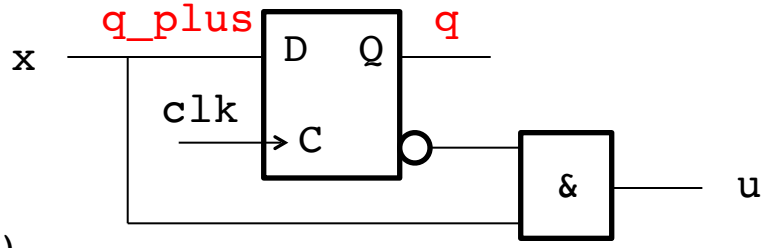
```
library ieee;
use ieee.std_logic_1164.all;

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

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```



VHDL-exempel - enpulsaren

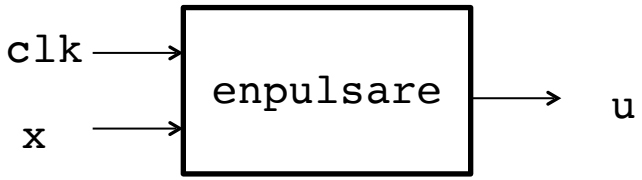
```
library ieee;
use ieee.std_logic_1164.all;

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

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```

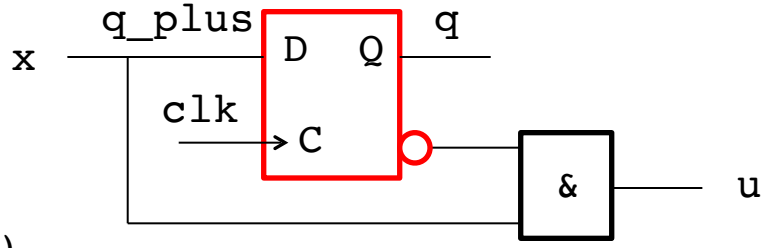




VHDL-exempel - enpulsaren

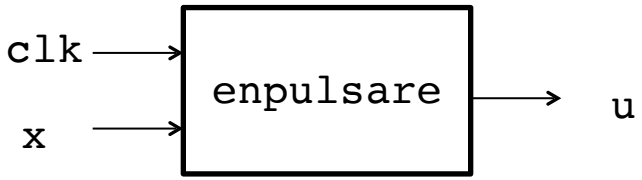
```
library ieee;
use ieee.std_logic_1164.all;

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

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```

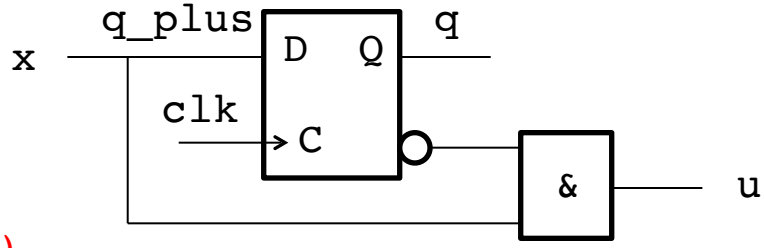




VHDL-exempel - enpulsaren

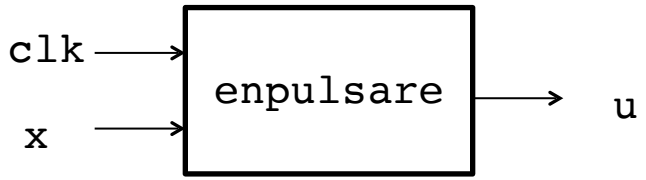
```
library ieee;
use ieee.std_logic_1164.all;

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

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                  -- q+ = f(q,x)
  u <= (not q) and x;           -- u  = g(q,x)
end ekvationer;
```

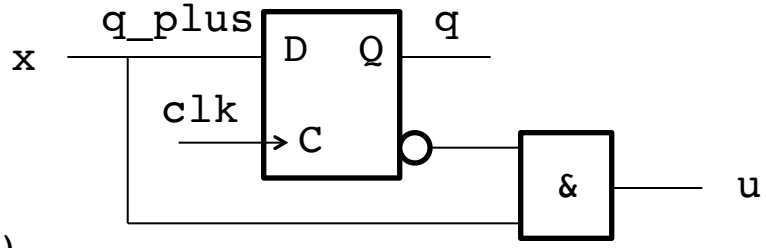




VHDL-exempel - enpulsaren

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

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

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```





Hur ser ett VHDL-program ut?

```
entity namn1 is
 -- beskrivning av in- och utgångar
end entity namn1;
```

Gränssnitt mot omvärlden

```
architecture namn2 of namn1 is
 -- beskrivning av interna signaler
begin
 -- beskrivning av funktion
end architecture namn2;
```

Innehåll

VHDL är inte case sensitive, små eller stora bokstäver spelar ingen roll, ej heller mellanslag (förutom i namn och nyckelord då ..).

Datatyper

Datatyper

- std_logic är datatypen som används för att representera bitar
- Kan anta följande värden: `01uxz-wlh`
  - `0`: Forcing 0
  - `1`: Forcing 1
  - `U`: Uninitalized, i simulering innan annat värde antagits
  - `X`: Forcing Unkown, i simulering när flera utgångar försöker driva signalen.
  - `Z`: High impedance, tri-state
  - `-`: Don't care, precis lika smidigt som när ni gör Karnaugdiagram

`std_logic_vector` repr. bitvektorer

- d = (d(0), d(1), d(2), d(3))
  - `d: in std_logic_vector(0 to 3);`
- s = (s(1), s(0))
  - `s: in std_logic_vector(1 downto 0);`

- Indexering
  - d(0)

Vi rekommenderar

- Använd endast
  - `IEEE.STD_LOGIC_1164.ALL`
  - **`std_logic`** och **`std_logic_vector`**
- Vill ni räkna inkludera
  - `IEEE.NUMERIC_STD.ALL` och använd
  - **`unsigned`**

Numeric_std

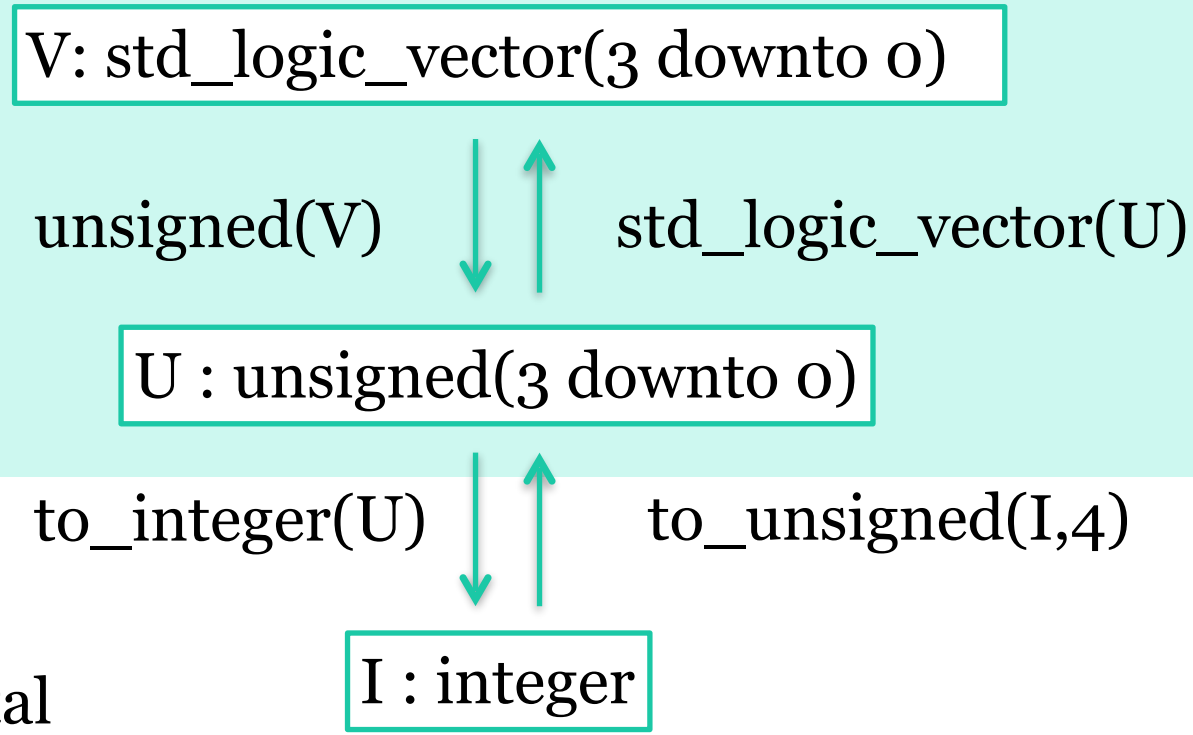
```
use IEEE.NUMERIC_STD.ALL -- lägger till paket

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

Notera hur värdet av heltal, bitvektor samt bit anges

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

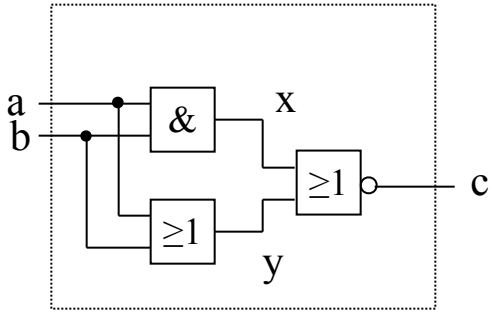
Typkonvertering

Bitvektorer med ett specificerat antal bitar



VHDL för kombinationskretsar

VHDL för kombinatoriska kretsar




```
entity knet is
  port(a,b: in std_logic;
       c:   out std_logic);
end entity knet;

architecture firsttry of knet is
  signal x,y : std_logic;
begin
  c <= not (x or y);
  x <= a and b;
  y <= a or b;
end architecture firsttry;
```

Parallellt exekverande satser.
Om `a` ändras så körs `x <= a and b` och `y <= a or b`,
vilket gör att `c <= not (x or y)` körs.
Ordningen spelar ingen roll.

Vad betyder ett VHDL-program?

Syntetisering (Xilinx)

- `x <=  a and b;`
  betyder att en AND-grind kopplas in mellan trådarna a, b och x

  Endast en tilldelning på x tillåten.

Simulering (ModelSim)

- `x <=  a and b;`
  är en parallellt exekverande sats som körs om a eller b ändras

Än så länge är ordningen mellan satserna oviktig
”Programmera” aldrig i VHDL!
Tänk hårdvara => översätt till VHDL

VHDL för kombinatoriska kretsar

Kombinationskretsar implementeras med

- ”vanlig” signaltilldelning `c <= a and b;`
- `when-else` är en generaliserad mux.

when-else

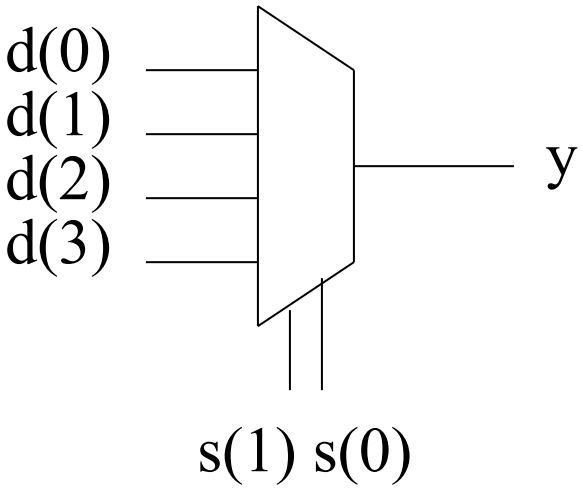
- Är en parallell sats, concurrent statement
- Endast utanför **process**

```
utsignal <= uttryck 1 when villkor 1 else
            uttryck 2 when villkor 2 else
            ...
            uttryck n-1 when villkor n-1 else
            uttryck n;
```

- Lägg märke till:
  - Det finns bara en <= i satsen.
  - Noll, ett eller flera villkor är sanna

    $s = u_1v_1 + u_2v_1'v_2 + \ldots + u_{n-1}v_1'v_2'\ldots v_{n-2}'v_{n-1} + u_nv_1'v_2'\ldots v_{n-2}'v_{n-1}'$
- **when-else** kan uttrycka vilken Boolesk funktion som helst!

En multiplexer



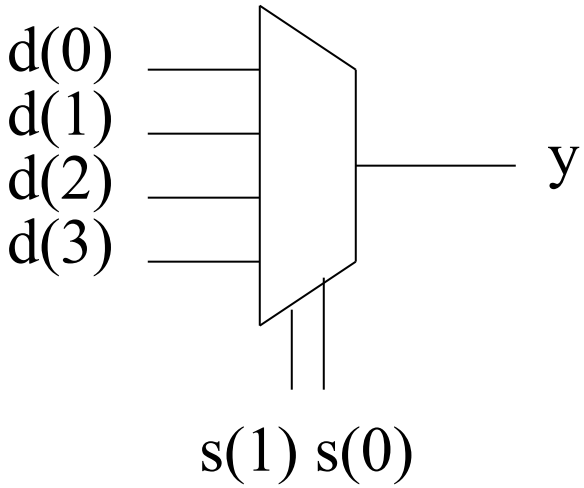
d = (d(0), d(1), d(2), d(3))

s = (s(1), s(0))

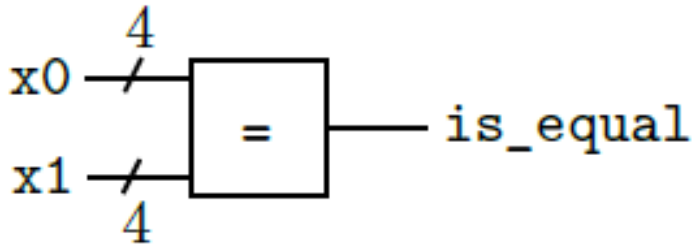
```
entity mux is
  port(  d: in std_logic_vector(0 to 3);
         s: in std_logic_vector(1 downto 0);
         y: out std_logic);
end entity mux;
```

Multiplexern, forts

```
architecture behavior2 of mux 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 behavior2;
```



Komparator




```
architecture beh of sys is

signal x0: std_logic_vector(3 downto 0);
signal x1: std_logic_vector(3 downto 0);
signal is_equal: std_logic;

begin

    is_equal <= '1' when (x1 = x0)
                else '0';

end architecture beh;
```

Vad har vi så långt?

- **entity** beskriver gränssnittet
- **architecture** beskriver innehållet
- Mellan **begin** och **end** har vi parallella satser.
  - ”vanlig” signaltilldelning **c <= a and b;**
  - **when-else** är en generaliserad mux.
  - Ovanstående används för kombinatorik utanför **process**-satsen

VHDL för sekvenskretsar

VHDL för sekvenskretsar

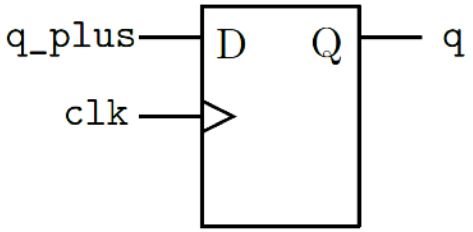
- process-satsen
  - if-then-else

Endast inuti process-sats!

process

Process-satsen exemplifierad på en D-vippa:

```
entity de is
  port(q_plus,clk: in STD_LOGIC;
               q: out STD_LOGIC);
end de;

architecture d_vippa of de is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      q <= q_plus;
    end if;
  end process;
end d_vippa;
```



Processen exekveras när `clk` ändras i känslighetslistan

`q` uppdateras på positiv `clk`-flank

Det gamla värdet på `q` ligger kvar om ett nytt ej specas.

Asynkron reset

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

Synkron reset

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

Känslighetslistan:

- `clk` ska alltid vara med
- Ev. asynkrona insignaler till vippa/räknare/register
- Andra signaler ska ej vara med.

Överraskande konsekvenser av processer

```
process(clk)
begin
  if rising_edge(clk) then
       y <= x;
       z <= y;
  end if;
end process;
```

- Låt `z = y = 0`
- Sätt `x = 1` och klocka en gång.
- Då blir väl `z = y = 1`?

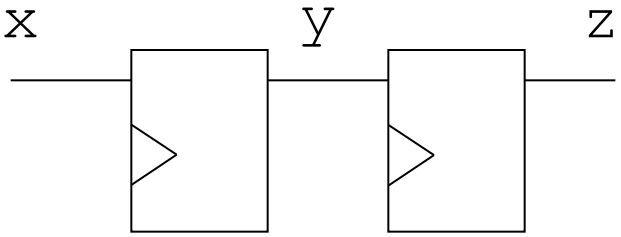
Så här blir det!

Inuti klockad process

```
y <= x;
z <= y;
```

synt

Alla VL får en vippa på sig



sim

Alla VL får ett + på sig

```
y+ = x;
z+ = y;

y = y+;
z = z+;
```

if-then-else

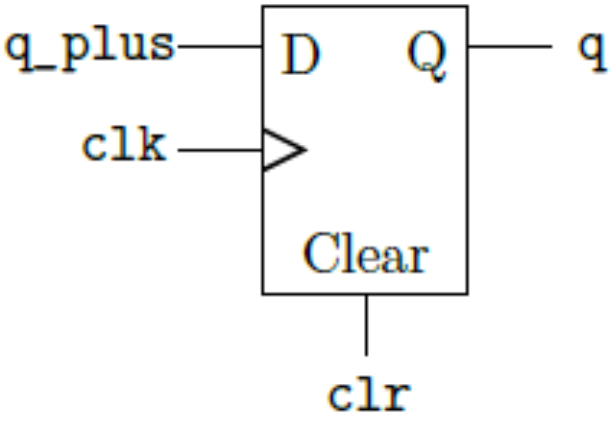
```
if (uttryck 1) then
   (sats 1)
elsif (uttryck 2) then
   (sats 2)
elsif (uttryck n-1) then
   (sats n-1)
else
   (sats n)
end if;
```

- Endast inuti **process**
- Motsvarar **when-else**
- Observera stavning på **elsif**

Exempel: D-vippa med asynkron clear

Lägg till asynkorn insignal i känslighetslistan

```
process(clk,clr) begin
    if clr = '1' then
        q <= '0';
    elsif rising_edge(clk) then
        q <= q_plus;
    end if;
end process;
```



Asynkron funktion

Synkron funktion

VHDL beskriver hårdvara!

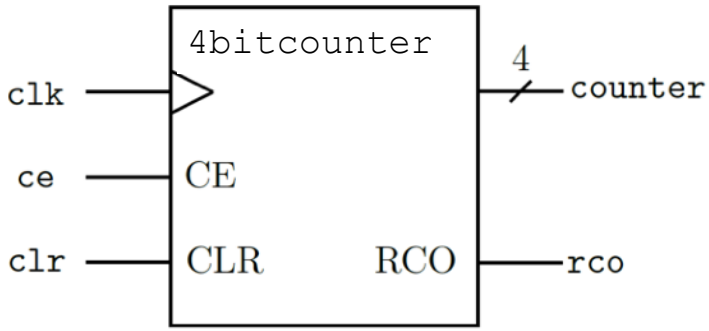
1. En VHDL-modul består av två delar
   a) **`entity`**, som beskriver gränssnittet
   b) **`architecture`**, som beskriver innehållet
2. För att göra kombinatorik används
   a) Booleska satser: **`z <= x and y;`**
   b) **`when-else`**-satser

   } Samtidiga satser
3. För att göra sekvensnät används (en eller flera) **`process(clk)`**-satser
   a) enn **`if rising_edge(clk)`** … **`end if;`**
   b) Booleska satser: **`z <= x and y;`**
   c) **`if-then-else`**-satser

   } VL får vippa på sig
   Alla klockas samtidigt

4-bitsräknare med asynkron clear

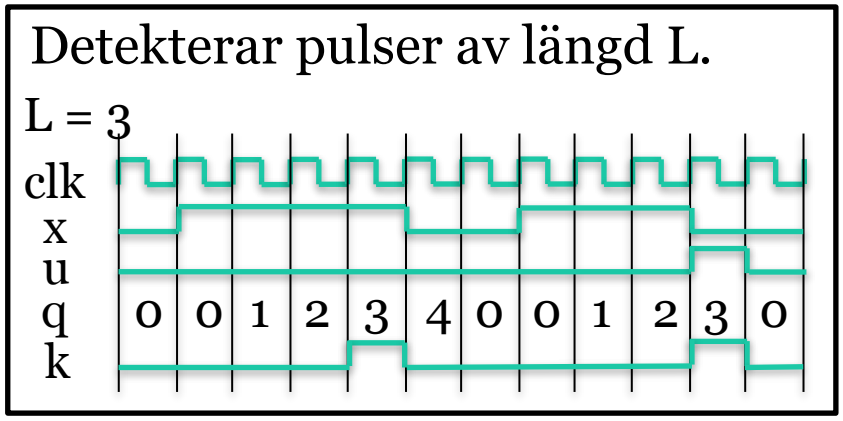

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity 4bitcounter is
port(clk, ce, clr : in std_logic;
      rco : out std_logic;
     counter: out std_logic_vector(3 downto 0));
end 4bitcounter;

architecture beh of 4bitcounter is
    signal q: unsigned(3 downto 0);
begin
  process(clk,clr) begin
    if clr = '1' then
           q <= "0000";
    elsif rising_edge(clk) then
        if ce = '1' then
           q <= q + 1; -- övergång 15->0 automatiskt
        end if;
    end if;
  end process;
  rco <= '1' when ((ce = '1') and (q = 15))
         else '0';
  counter <= std_logic_vector(q);    -- hårt typat

end beh;
```



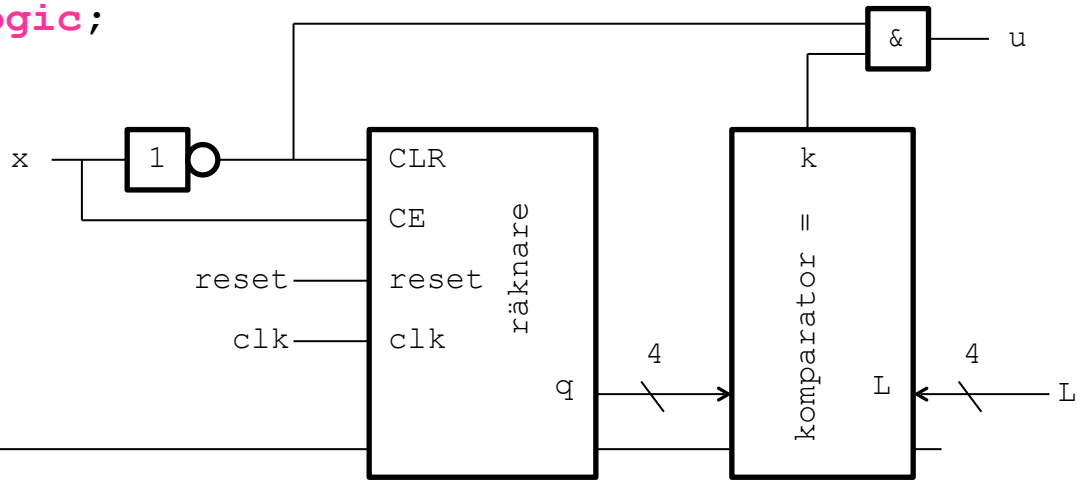
Alternativ: q <= to_unsigned(0,4);

Exemplifierad arbetsgång

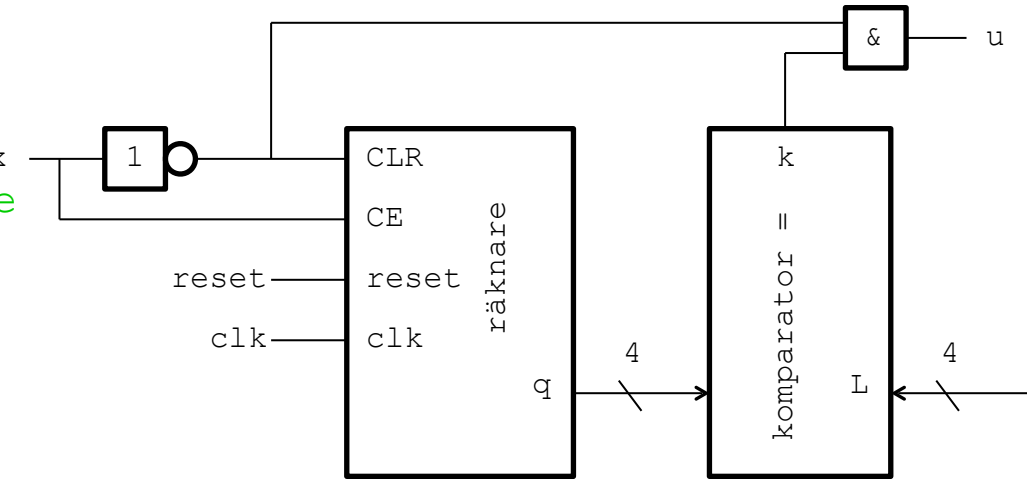
Ett labliknande exempel




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

entity pulsdetektor is
  port( clk, x : in  std_logic;
        reset : in  std_logic;
        L : in std_logic_vector(3 downto 0);
        u : out std_logic);
end pulsdetektor;

architecture Behavioral of pulsdetektor is
  signal q : unsigned(3 downto 0);
  signal CLR, CE, k: std_logic;
begin
  -- hela vår konstruktion

end Behavioral;
```



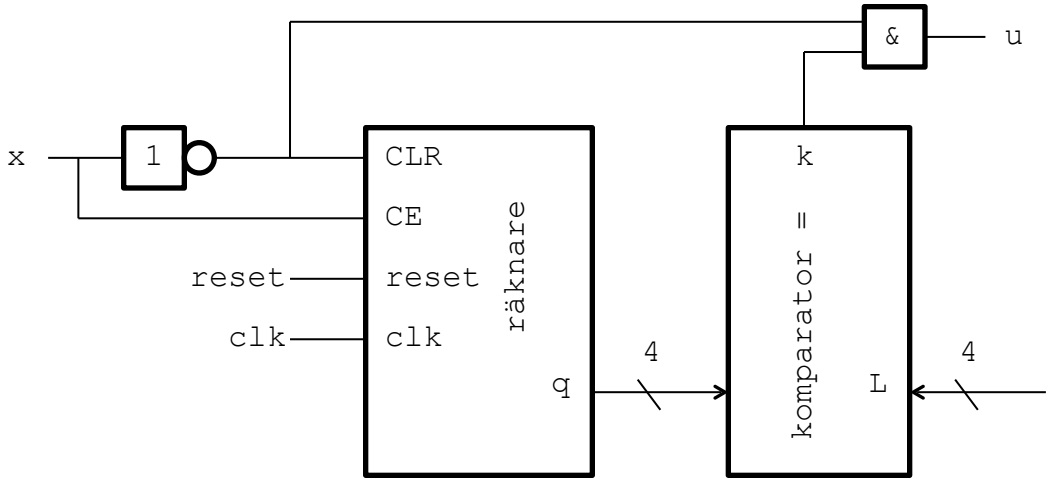
Räknaren




```
-- insignaler till räknare
CE <= x;
CLR <= not x;

-- räknare
ctr16: process(clk,reset)
begin
    if reset = '1' then       -- asynkron clear
       q <= "0000";
    elsif rising_edge(clk) then
       if CLR = '1' then      -- synkron clear
           q <= "0000";
       elsif CE = '1' then    -- count enable
           q <= q + 1; -- övergång från 15->0 automatiskt
       end if;
    end if;
 end process ctr16;
```

Komparator och utsignal

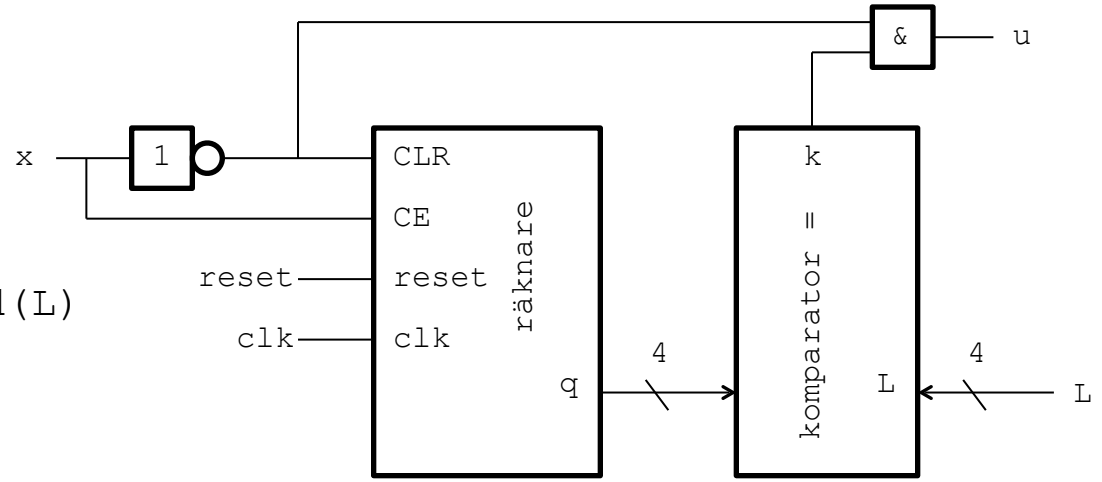
```
-- komparator
k <= '1' when q = unsigned(L)
    else '0';

-- utsignal
u <= CLR and k;
```



Blockschema ->VHDL

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

entity pulsdetektor is
   ...
end pulsdetektor;

architecture Behavioral of
pulsdetektor is
   ...
begin
   -- insignaler till räknare
   CE <= x;
   CLR <= not x;

   -- räknare
   ctr16: process(clk,reset)
      ...
   end process ctr16;

   -- komparator
   k <= '1' when q = unsigned(L)
            else '0';
   -- utsignal
   u <= CLR and k;
end Behavioral;
```

- Varje block har motsvarande kod.
- Överensstämmande signalnamn i blockschema och kod.



Simulering i ModelSim

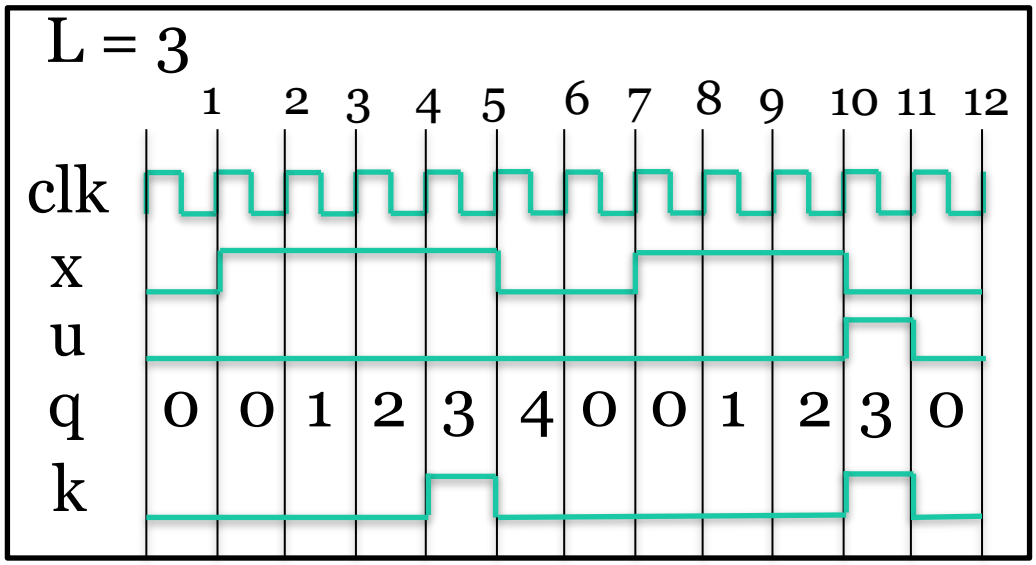
Utdrag ur fil för att definiera insignaler och köra en simulering.

```
# sim.do
...
# clk Periodtid 100 ns
force -freeze sim:/pulsdetektor/clk 1 0, 0 50 -r 100
# reset
force -freeze sim:/pulsdetektor/reset 0 0, 1 50, 0 90
# x
force -freeze sim:/pulsdetektor/x 0 0, 1 105, 0 505, 1 705, 0 1005
# L
force -freeze sim:/pulsdetektor/L 0011 0
run 1300
```

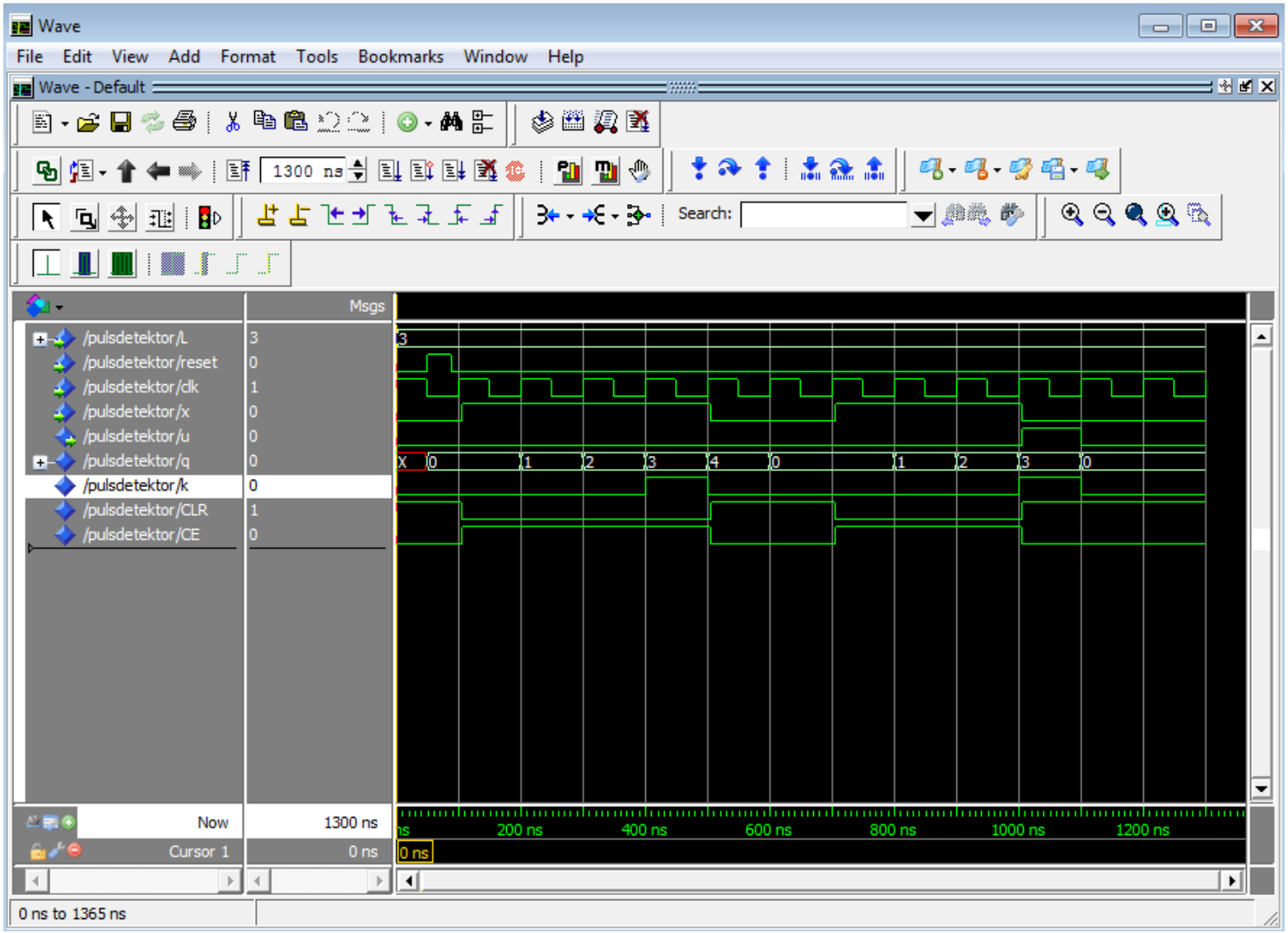
Starta simulering i Transcriptfönstret:

```
VSIM > do sim.do
```

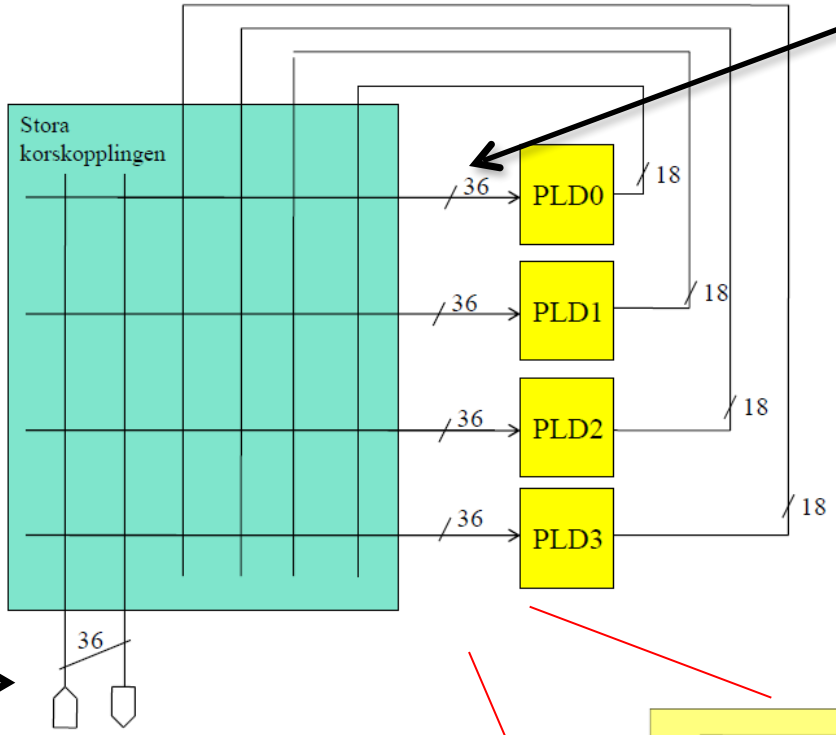
Ni behöver inte göra egna do-filer. Det finns färdig testbänkar med fördefinierade insignaler och testfall.



ModelSim


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
1300 ns
Search:
Msgs
/pulsdetektor/L 3
/pulsdetektor/reset 0
/pulsdetektor/clk 1
/pulsdetektor/x 0
/pulsdetektor/u 0
/pulsdetektor/q 0
/pulsdetektor/k 0
/pulsdetektor/CLR 1
/pulsdetektor/CE 0
X 0 1 2 3 4 0 1 2 3 0
Now 1300 ns
Cursor 1 0 ns
200 ns 400 ns 600 ns 800 ns 1000 ns 1200 ns
0 ns
0 ns to 1365 ns

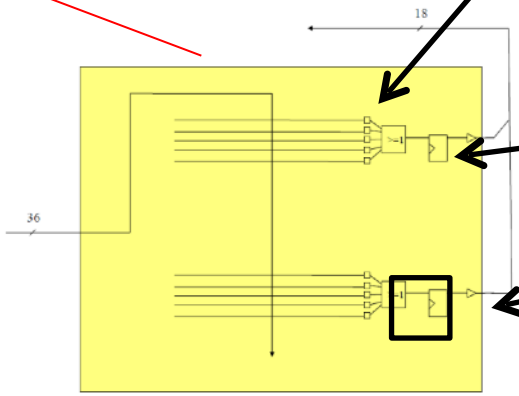
Får det plats?



functional block inputs!

produkttermer!
5 AND/vippa

pinnar!



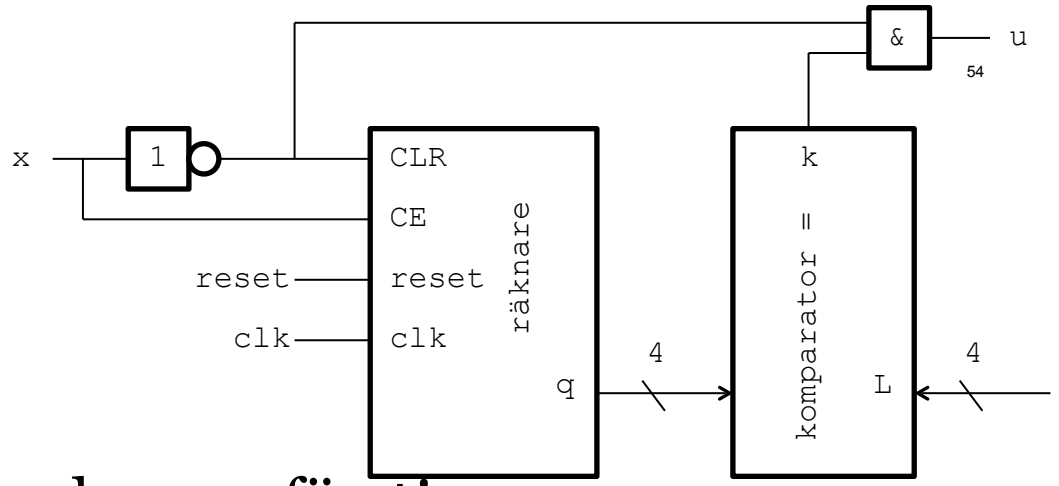
vippor!

makro-cell!

Vad kan ta slut?

- Pinnar
- Vippor
- Produkttermer

Slutsatser



- Vi ritar logiska blockscheman först!
- Vi har samma struktur på koden som på blockschemat!
  - Alltså: små process-satser som precis motsvarar ett block.
  - Vi har bra koll på mängden hårdvara.

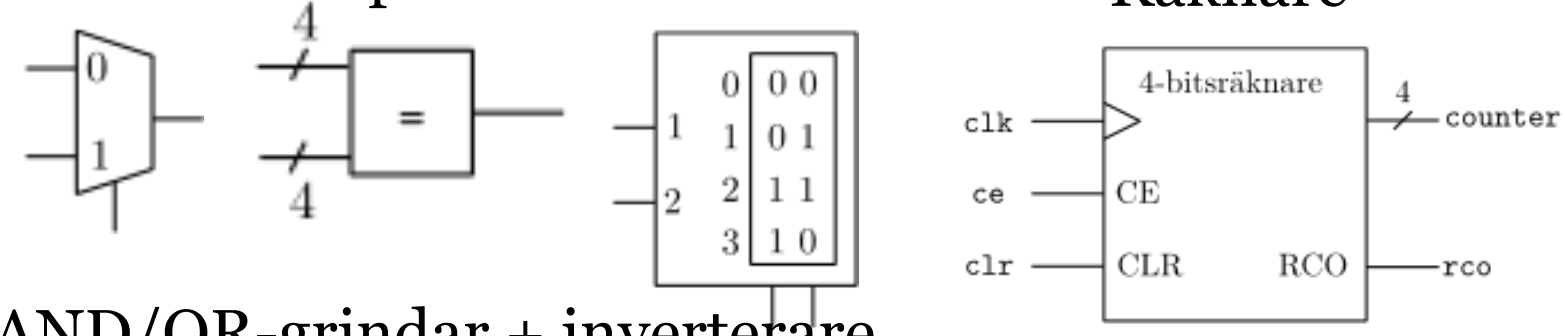
Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
5/36 (14%)	16/180 (9%)	4/36 (12%)	8/34 (24%)	9/72 (13%)

Använda produkttermer L3-L0, x, u, clk, reset

Beräkning av 4 tillstånd + 1 utsignal 4 D-vippor i 4-bitsräknaren L3-L0, q3-q0, x

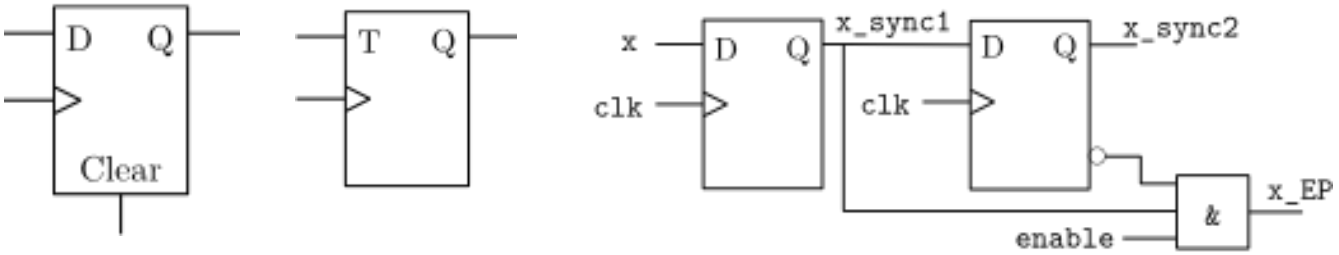
Digitaltekniska byggblock (använd dessa)

MUX Komparator ROM Räknare



AND/OR-grindar + inverterare

D-vippa T-vippa Enpulsare med synkronisering



Komponenternas kod i pdf:en Digitaltekniska byggblock.

Lab 3

- VHDL-kod
  - Varje block ska motsvara ett kodavsnitt
  - Överensstämmande signalnamn i blockschema och kod.
- Glöm inte synkroniseringsvippor på ingångarna!

Digitalteknik

Mattias Krysander

www.liu.se

LINKÖPINGS UNIVERSITET