Programmerbara kretsar och VHDL

Föreläsning 10
Digitalteknik, TSEA22
Mattias Krysander
Institutionen för systemteknik

LINKÖPINGS UNIVERSITET

Dagens föreläsning

- Programmerbara kretsar
- Kombinationskretsar i VHDL

  with-select-when, when-else
- Sekvenskretsar i VHDL

  process, case-when, if-then-else
- In-/ut-signaler, datatyper, mm
- Räknare i VHDL
- Exempel

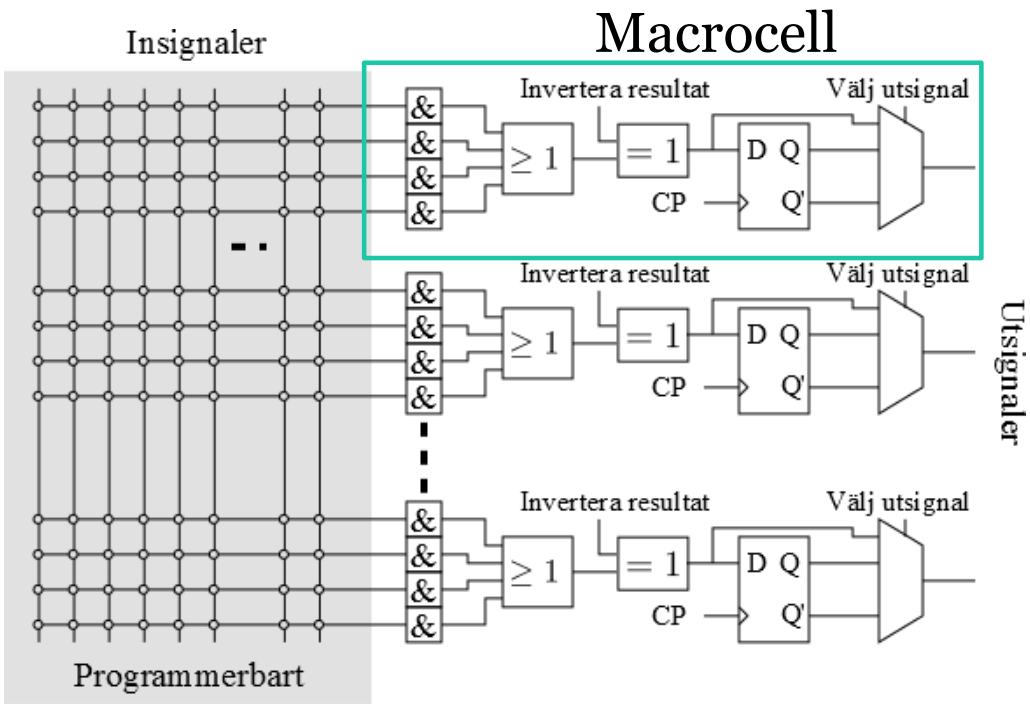
  Blockschema -> VHDL, kodstandard, simulering, programmering, verifiering
- Lab 4

Programmerbara kretsar

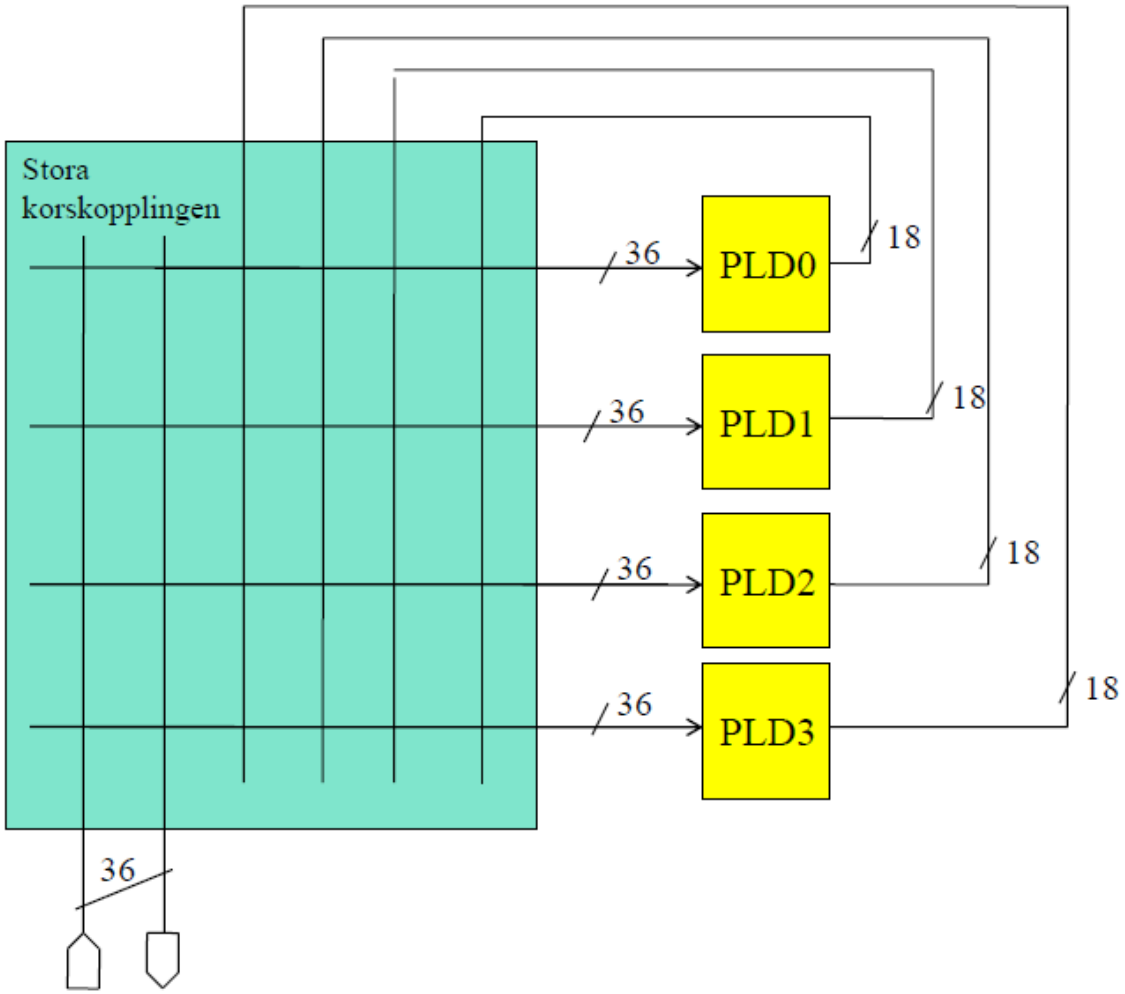
- PLD = programmable logic device
- CPLD = complex PLD,
  i princip flera PLD-er på ett chip
  ex: 108 vippor + 540 produkttermer
- FPGA = field programmable gate array
  - 5 000 000 vippor
  - 2 000 000 Look-up-tables (LUT)
  - 500 Mb RAM
  - Digital Signal Processing (DSP)
  - Processor

CPLD - konstruktion

- Grundblocket i en CPLD består oftast av ett AND-OR-nät
- AND-grindarnas ingångar är programmerbara
- Kallas Programmable Logic Array (PLA)



CPLD Xilinx 9572 - blockschema



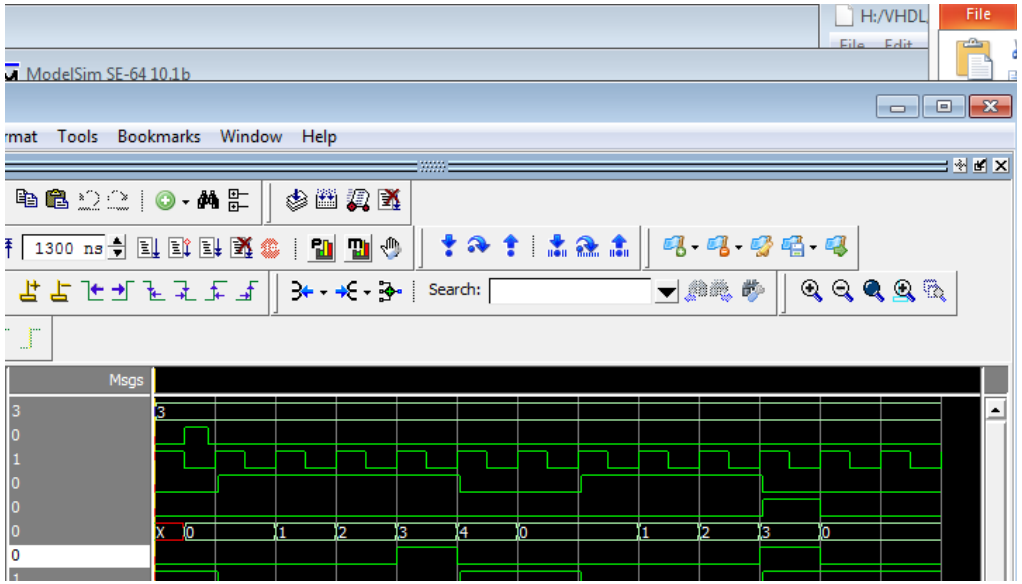
VHDL

Ett programspråk för att:

Syntetisera
(Xilinx)

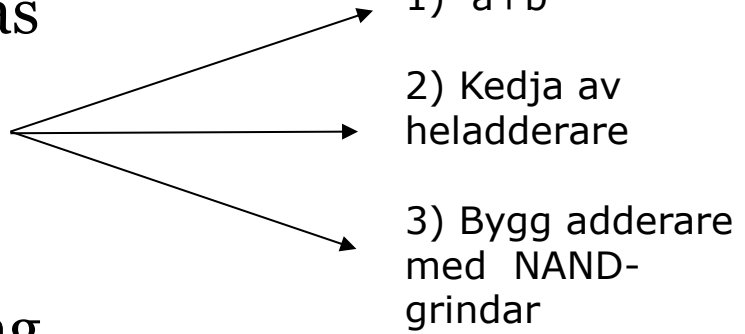
Simulera
(ModelSim)

Hårdvara



Varför VHDL?

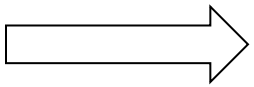
- Hantera komplexitet
  - VHDL-koden kan simuleras
  - Beskrivning på flera olika abstraktionsnivåer
- Ökad produktivitet
  - Snabbare än schemaritning
  - Återanvändbar kod
- Modernt programmeringsspråk
  - Rikt, kraftfullt
  - Parallellt, starkt typat, overloading
  - Ej objektorienterat



VHDL nackdelar?

- Svårt att lära sig?
  - Delmängd för syntes 1-2 dagar!
  - Avancerade simuleringar 1-2 månader
- Nytt sätt att tänka
  - Lätt att hamna i mjukvarutänkande!
  - FPGA-n, CPLD-n är inte en processor för VHDL
  - VHDL är inte sekvensiellt utan parallellt
  - Tilldelning, variabler betyder inte samma sak som i andra prog.språk
  - Gör så här:

Tänk hårdvara och gör ett blockschema



Översätt till VHDL

Hur ser ett VHDL-program ut?

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

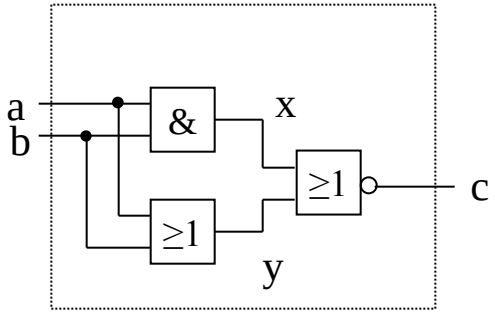
Gränssnitt mot omvärlden

```
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.

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 <= x nor y` körs.
Ordningen spelar ingen roll.

Vad betyder ett VHDL-program?

Syntetisering (Xilinx)

- `x <=  a and b;`
  betyder att en OCH-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 körs om a eller b ändras

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

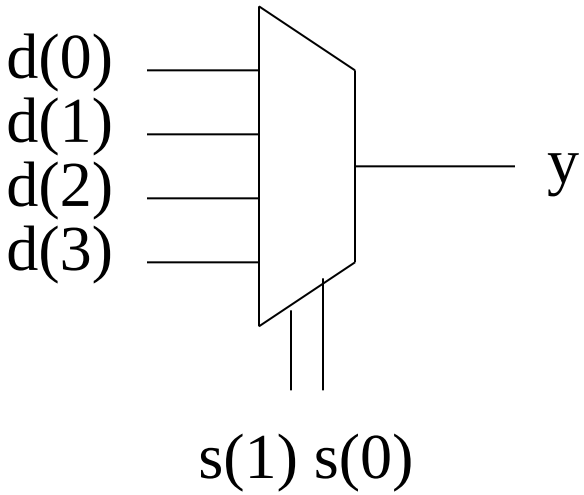
VHDL för kombinatoriska kretsar

Kombinationskretsar implementeras med

- ”vanlig” signaltilldelning `c <= a and b;`
- `with-select-when` är en mux (använt för minne i lab 2).
- `when-else` är en generaliserad mux.

En multiplexer

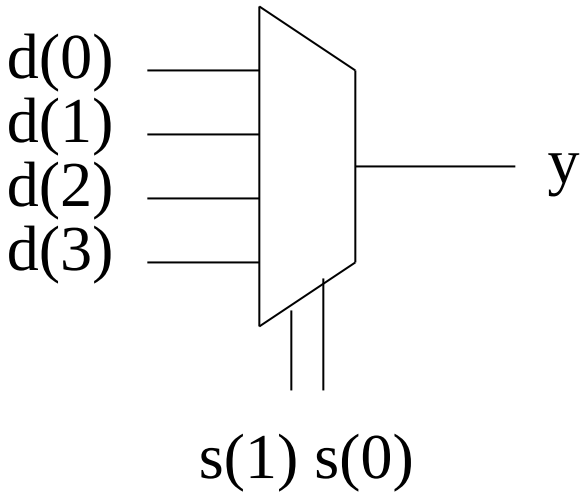
```
entity mux is
  port(  d: in std_logic_vector(0 to 3);
         s: in std_logic_vector (1 downto 0);
         y: out std_logic);
end entity mux;
```



Multiplexern, forts

VHDL har en sats som precis motsvarar en mux:

```
architecture behavior1 of mux is
begin
  with s select
    y <= d(0) when "00",
         d(1) when "01",
         d(2) when "10",
         d(3) when others;
end architecture behavior1;
```



Lägg märke till:
- det finns enn <= i satsen.
- enn rad är sann

with-select-when

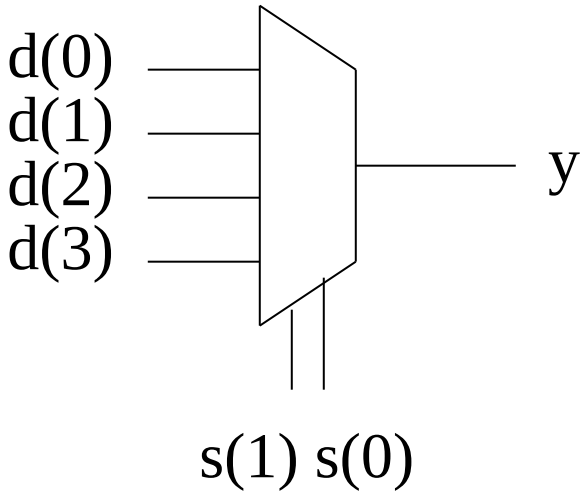
- Är en parallell sats, concurrent statement
- Endast utanför **process**

```
with (styrsignal) select
 (utsignal) <= (uttryck 1)    when (signalvärde 1),
               (uttryck 2)    when (signalvärde 2),
               ...
               (uttryck n-1) when (signalvärde n-1),
               (uttryck n)    when others;
```

OBS: samtliga värden på styrsignal **måste täckas!**

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;
```



when-else

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

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

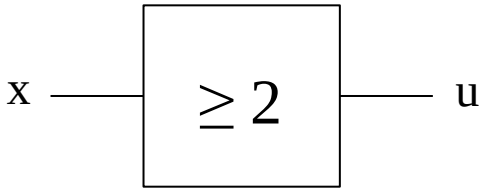
- Lägg märke till:
  - Det finns enn <= i satsen.
  - Noll eller flera villkor är sanna

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

Exempel

u = 1 omm x = (x(2), x(1), x(0)) innehåller 2 eller 3 ettor.

```
signal x: std_logic_vector(2 downto 0);
signal u: std_logic;

-- tavla
```



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;**
  - **with-select-when** är en mux.
  - **when-else** är en generaliserad mux.
  - Ovanstående används för kombinatorik utanför **process**-satsen

VHDL för sekvenskretsar

- process-satsen
  - case-when
  - if-then-else

} Endast inuti process-sats!

process

Process-satsen exekveras sekventiellt. Här exempel på D-vippa:

```
entity de is
  port(d,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 <= d;
    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.

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`?

case-when

- Endast inuti **process**
- Måste beskriva vad som händer för alla värden på styrsignal
- Motsvarar **with-select-when**

```
case (styrsignal) is
  when (värde 1)    => (sats 1);
  when (värde 2)    => (sats 2);
    ...
  when (värde n-1) => (sats n-1);
  when others       => (sats n);
end case;
```

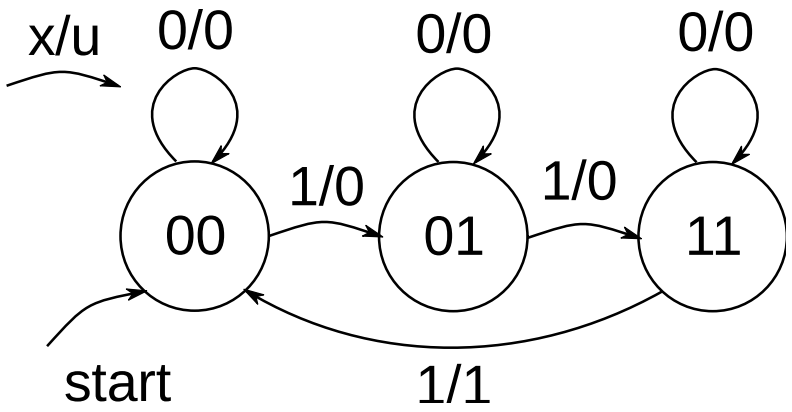
if-then-else

```
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

Bygg en sekvenskrets som ger utsignalen 1 i samma klockintervall som var tredje 1:a.

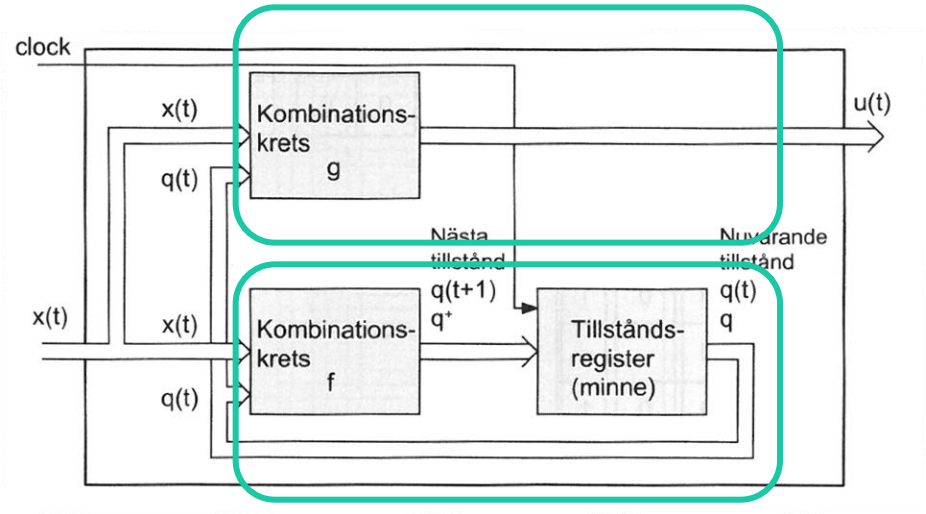


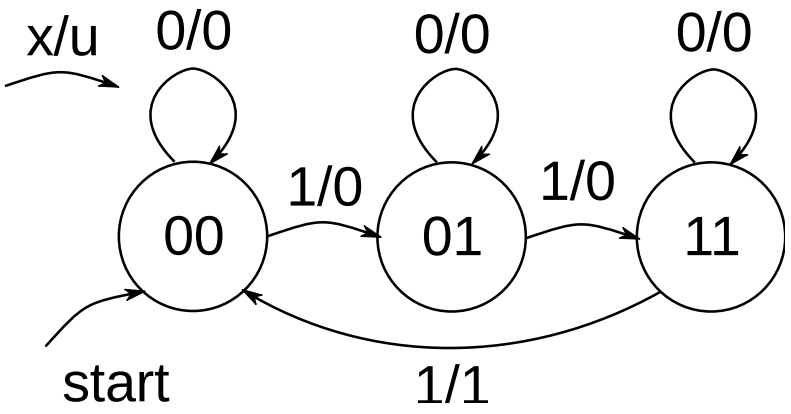
VHDL-beskrivning som liknar grafen.

Sekvenskretsar

Utsignal

Gör kombinatorik av detta



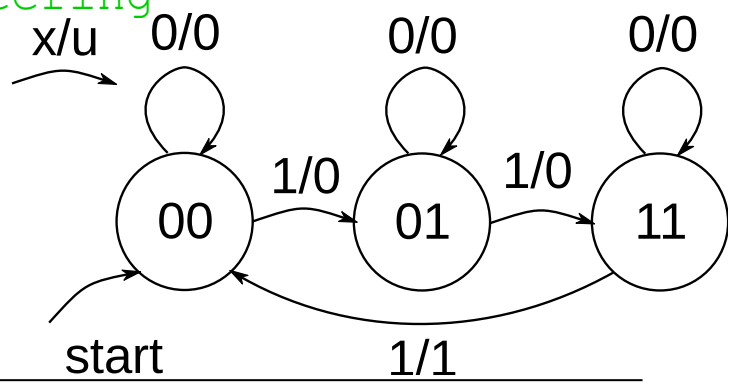


Tillståndsuppdatering

Gör en `process(clk)` av detta.

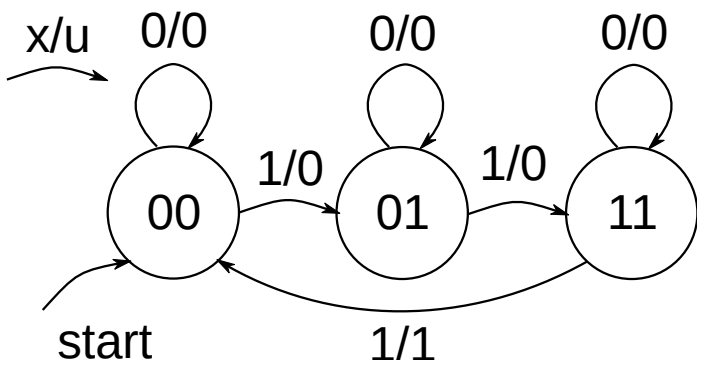
Sekvenskrets

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

architecture graf of skrets is
  signal q: std_logic_vector(1 downto 0);
begin
  -- q+ = f(q,x): tillståndsuppdatering

  -- u = g(q,x) :  utsignal

end graf;
```



Tillståndsuppdatering

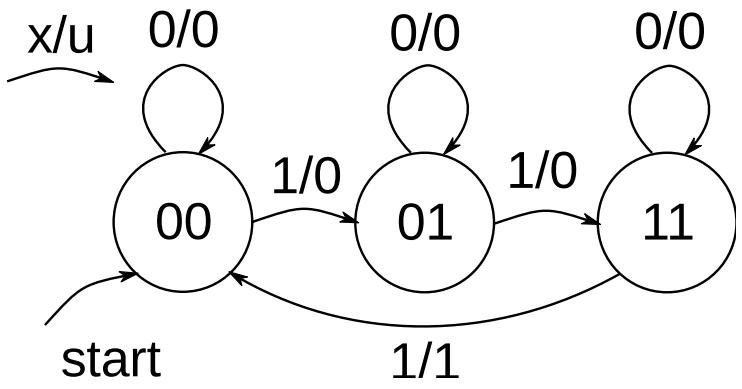
```
-- q+ = f(q,x): tillståndsuppdatering
process(clk)
begin
 if rising_edge(clk)then
  case q is
    when ”00” => if x=’1’ then q <= ”01”;
                   else q <= ”00”;
                 end if;
    when ”01” => if x=’1’ then q <= ”11”;
                   else q <= ”01”;
                 end if;
    when ”11” => if x=’1’ then q <= ”00”;
                   else q <= “11”;
                 end if;
    when others => q <= ”00”;
  end case;
 end if;
end process;
```



Utsignal

- K-krets utanför processen

```
u <= '1' when q="11" and x='1' else
     '0';

-- alternativt
u <= x and q(1) and q(0);
```



Komplett kod

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

architecture graf of skrets is
  signal q: std_logic_vector(1 downto 0);
begin
  -- q+ = f(q,x): tillståndsuppdatering
  process(clk)
  begin
    if rising_edge(clk)then
      case q is
        when "00" => if x='1' then q <= "01";
                       else q <= "00";
                     end if;
        when "01" => if x='1' then q <= "11";
                       else q <= "01";
                     end if;
        when "11" => if x='1' then q <= "00";
                       else q <= "00";
             end if;
        when others => q <= "00";
      end case;
    end if;
  end process;
  -- u = g(q,x) : utsignal
  u <= x and q(1) and q(0);
end graf;
```

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) **`with-select-when`**-satser
   c) **`when-else`**-satser

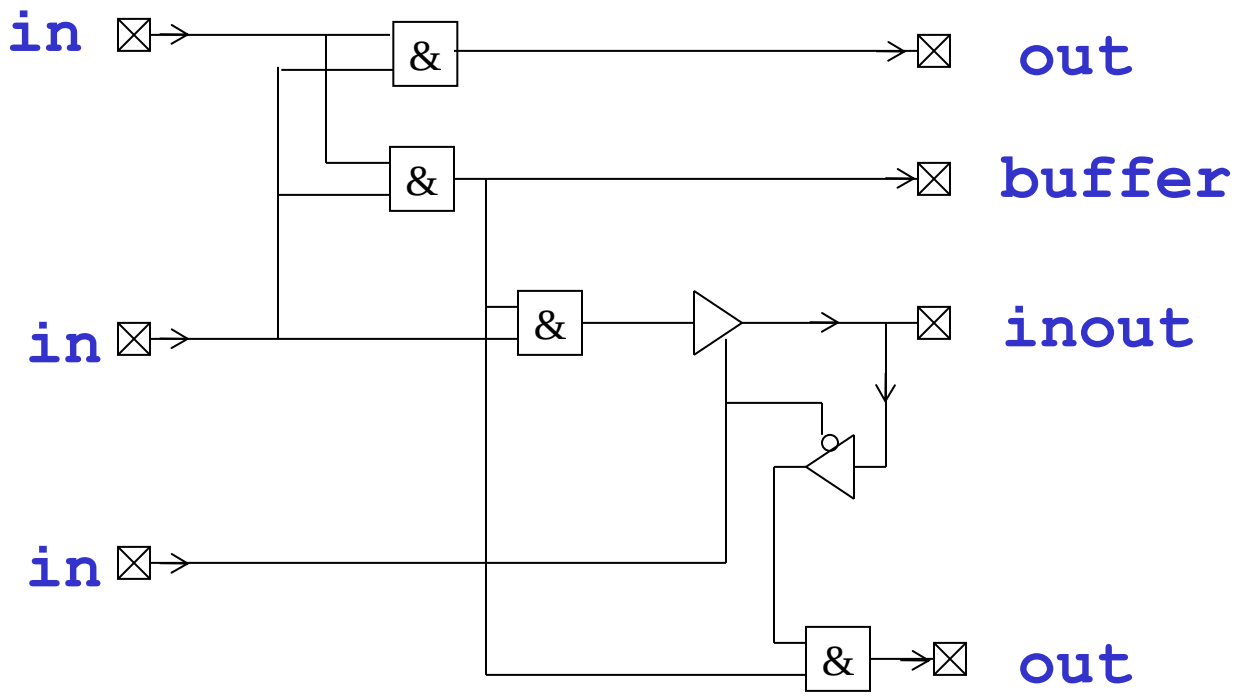
   } Samtidiga satser

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) **`case-when`**-satser
   d) **`if-then-else`**-satser

   } VL får vippa på sig
   Alla klockas samtidigt

In/utsignaler

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



std_logic

```
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

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];

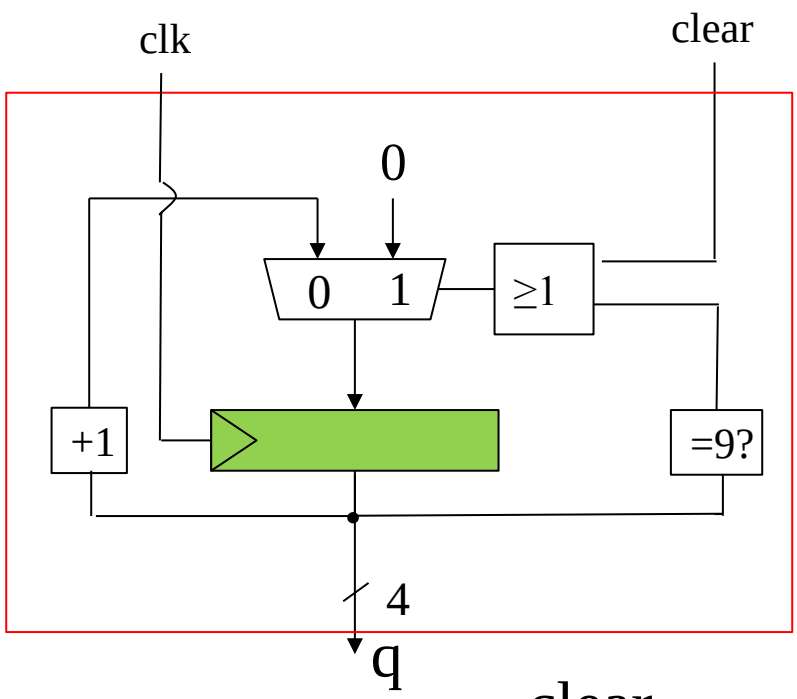
Vi rekommenderar

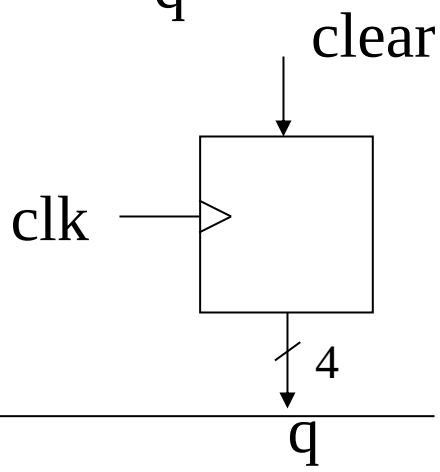
- Använd endast
  `std_logic` och **`std_logic_vector`**
- Vill ni räkna inkludera
  `NUMERIC_STD` -biblioteket

- Skippa **`integer`**. Går ej att indexera på bit-nivå.
- Skippa **`bit`**. Tristate och don't care saknas.

4-bits dekadräknare med synkron clear

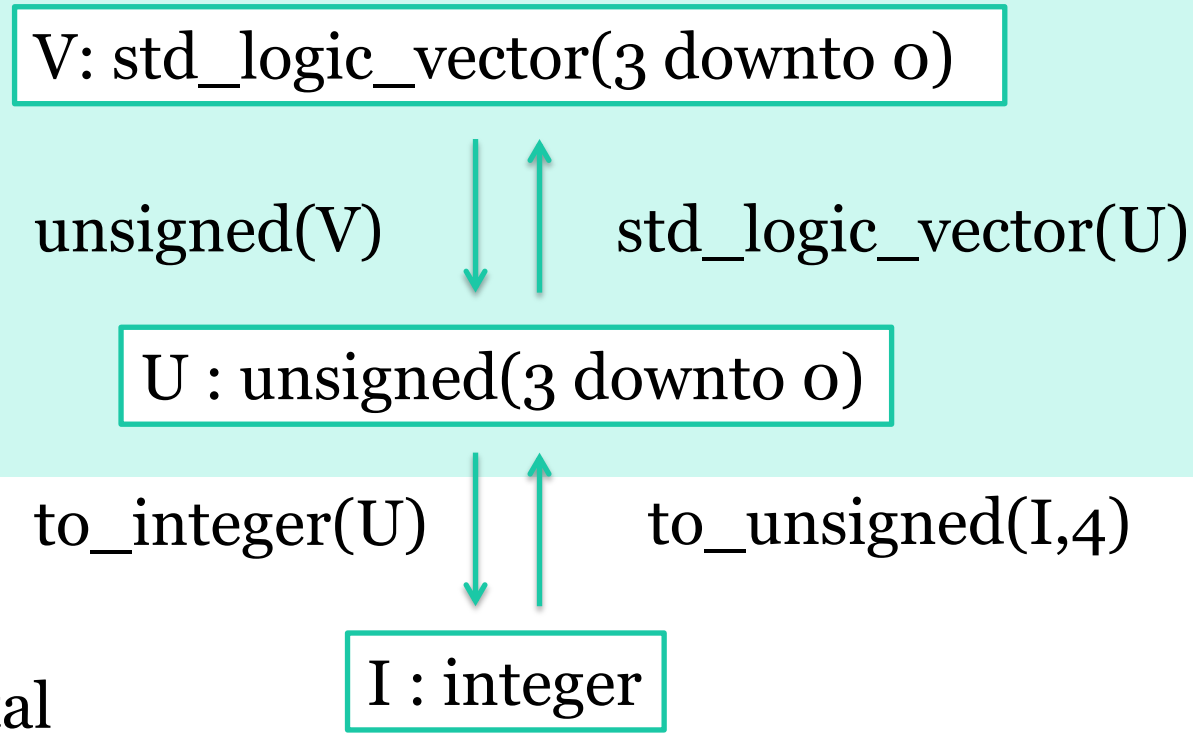

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

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

architecture simple of counter is
    signal q_int: unsigned (3 downto 0));
begin
  process(clk)
  begin
    if rising_edge(clk) then
        if clear='1' then
          q_int <= "0000";
        elsif q_int = 9 then
          q_int <= to_unsigned(0,4);
        else
          q_int <= q_int + 1;
        end if;
    end if;
  end process;
  q <= std_logic_vector(q_int);
end simple;
```





Typkonvertering

Bitvektorer med ett specificerat antal bitar

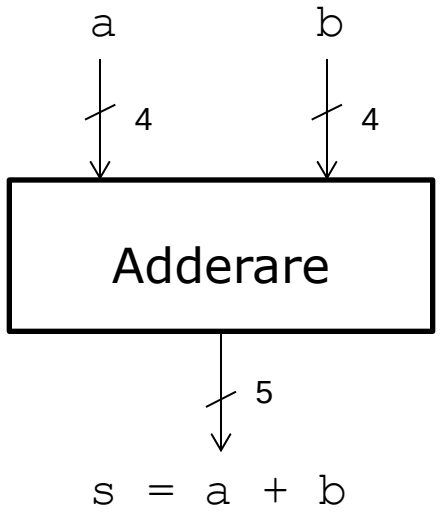


Konkatenering

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

4-bitsadderare

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


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


architecture simple of adder is
begin

-- zero extension
  s <= ('0'& a) + ('0'& b);
end simple;
```



Asynkron reset

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

Synkron reset

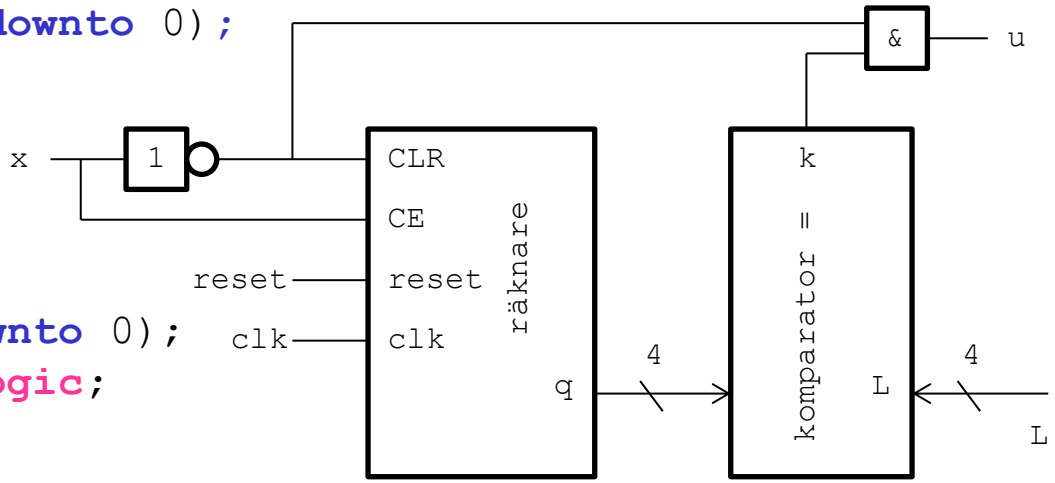
```
process(clk)
begin
 if rising_edge(clk) then
  if reset='1' then
    q <= '0';
  else
    q <= q and x;
  end if;
 end if;
end process;
```

Känslighetslistan:

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

Logiskt blockschema => VHDL

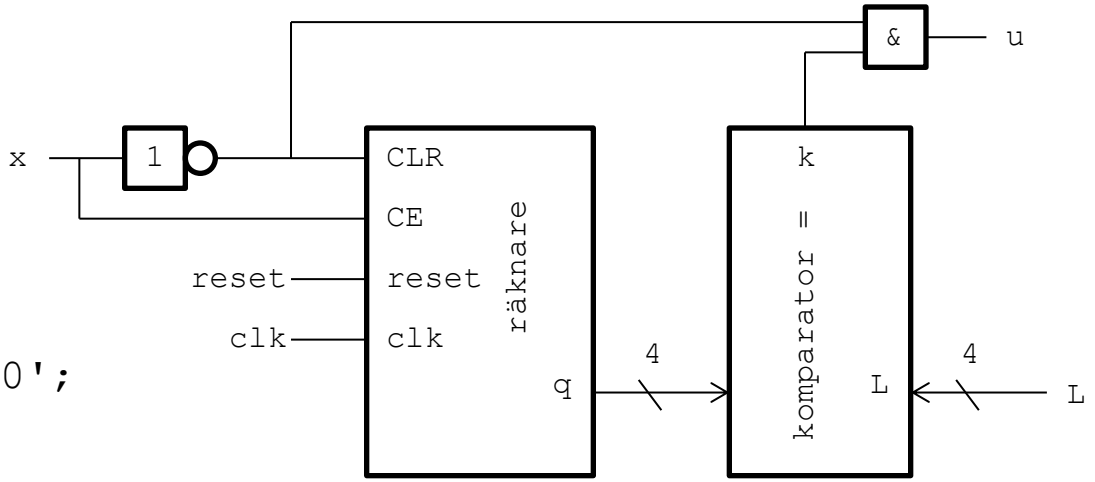

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pulsdetektor is
  port( clk, x : in  std_logic;
        reset : in  std_logic;
        L : in unsigned(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;
```



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 L = q else '0';

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

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



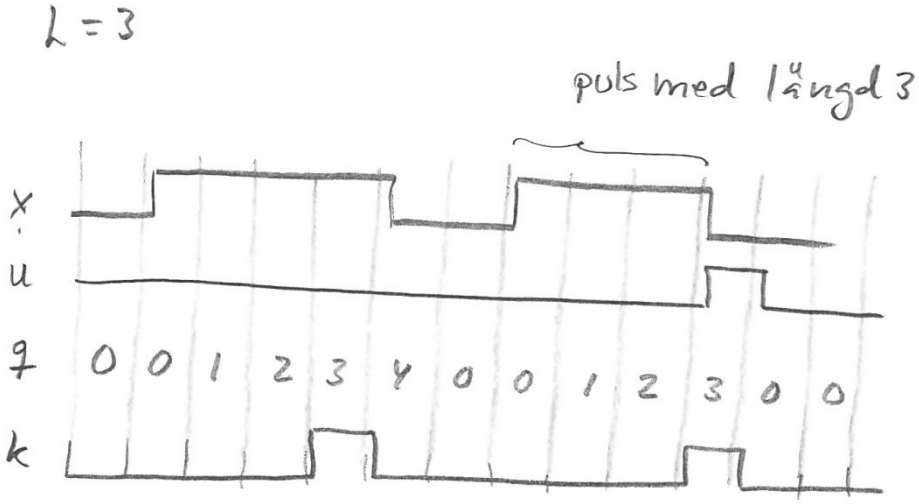
Simulering i ModelSim

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

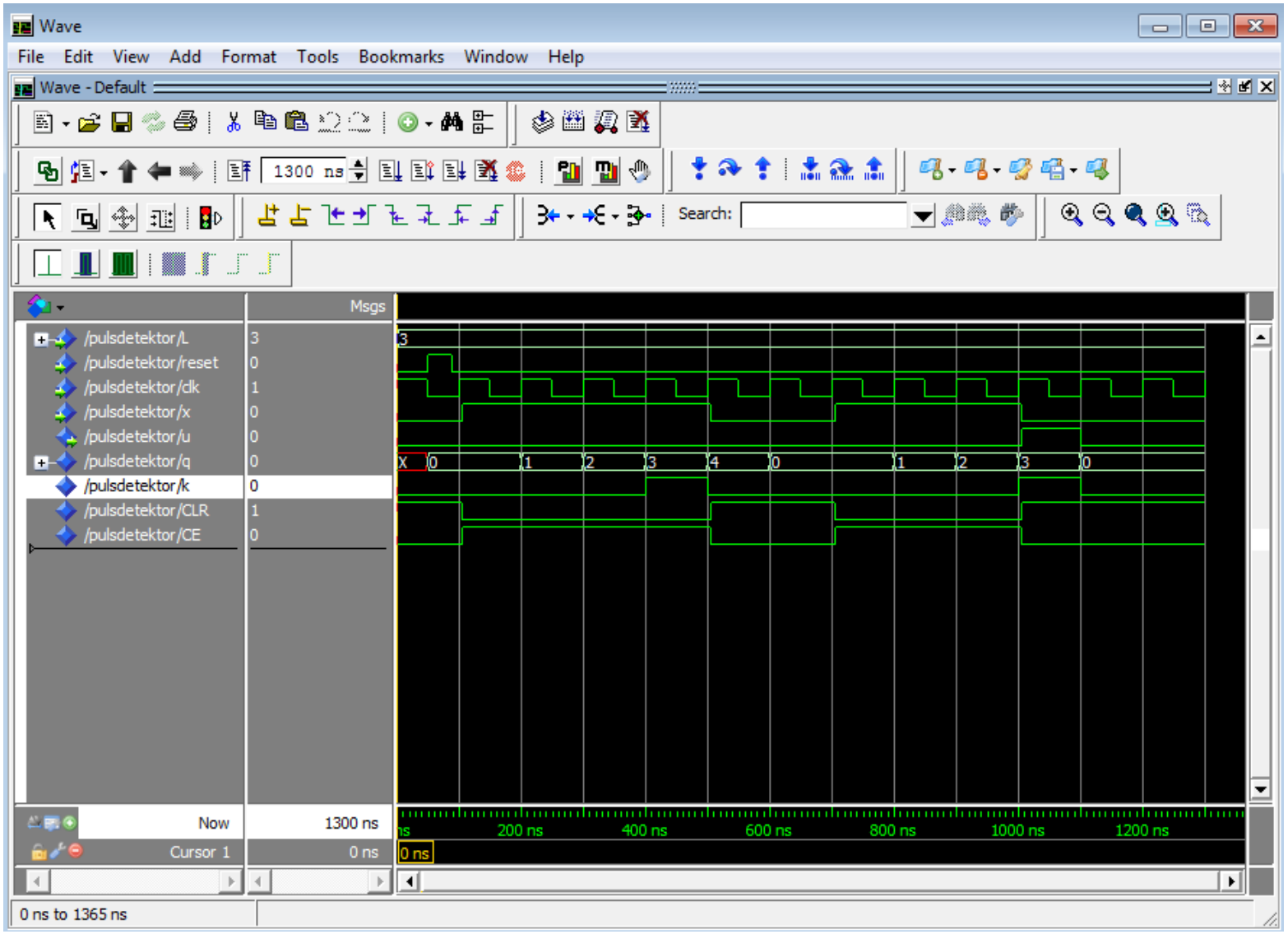
```
# sim.do
...
# clk Periodtid 100 ns
force -freeze sim:/pulsdetektor/clk 1 0, 0 50 -r 100
# reset
force -freeze sim:/pulsdetektor/reset 0 0, 1 50, 0 90
# x
force -freeze sim:/pulsdetektor/x 0 0, 1 105, 0 505, 1 705, 0 1005
# L
force -freeze sim:/pulsdetektor/L 0011 0
run 1300
```

Starta simulering i transcriptfönstret

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



ModelSim


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
1300 ns
Search:
Msgs
/pulsdetektor/L
/pulsdetektor/reset
/pulsdetektor/clk
/pulsdetektor/x
/pulsdetektor/u
/pulsdetektor/q
/pulsdetektor/k
/pulsdetektor/CLR
/pulsdetektor/CE
3
0
1
0
0
0
0
1
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 to 1365 ns

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

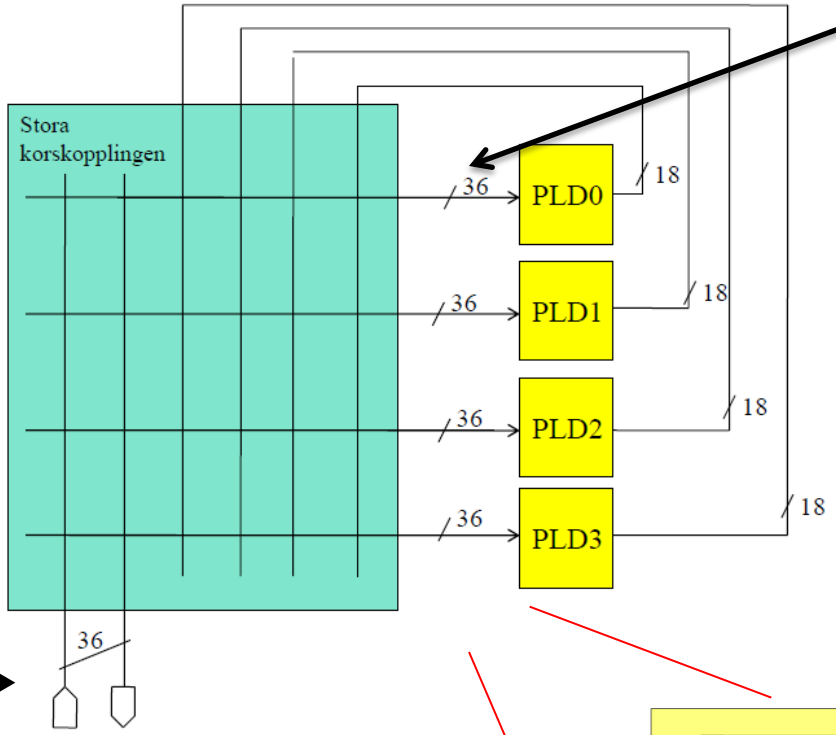
L3-L0, x, u, clk, reset

Beräkning av 4 tillstånd + 1 utsignal

4 D-vippor i 4-bitsräknaren

L3-L0, q3-q0, x

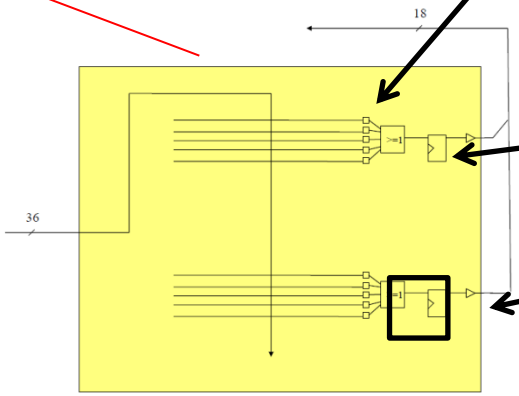
Får det plats?



functional block inputs!

produkttermer!
5 AND/vippa

pinnar!



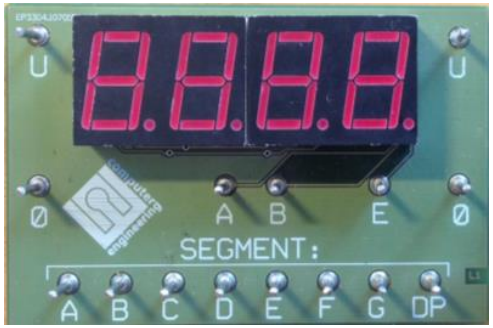
vippor!

makro-cell!

Vad kan ta slut?

- Pinnar
- Vippor
- Produkttermer

Lab 4



- Bland annat konstruera ett digitalt tidtagarur.
- Olika klockor
  - Systemklocka 8 MHz
  - Klocka som styr uppdateringsfrekvens på display (variabel frekvens)
  - Klocka som styr tidräkning 100 Hz
- Synkronisera klockpulser till systemklockan:
  - synkronisering + enpulsning

Inför Lab 4

- Obligatorisk lektion i ModelSim.
- Det finns ett antal valfria uppgifter.
  - Om ni vill göra VGA-uppgift (mer omfattande) så kontakta: olov.andersson@liu.se
- Förberedelseuppgifter:
  - Logiska blockscheman till uppgifterna
  - ModelSim-simulering av två kaskadkopplade BCD-räknare för sekundvisning. Ska redovisas vid laborationens start.
- ModelSim finns installerat på datorerna i Freja (och i Grinden).

Examination

- Vid examination ska logiskt blockschema, VHDL-kod samt korrekt fungerande krets uppvisas.
- Logiskt blockschema
  - Block ska bestå av väldefinierad hårdvara:
    - räknare, register, enpulsare, synkroniseringsvippor, sekvenskretsar (med tillståndsdiagram)
    - MUX, DMUX, adderare, komparatorer, kombinatoriska kretsar (minne, Booleska funktioner)
- VHDL-kod
  - Varje block ska motsvara ett kodavsnitt, t ex en process-sats/sekvenskrets
  - Överensstämmande signalnamn i blockschema och kod.

Digitalteknik

Mattias Krysander

www.liu.se

LINKÖPINGS UNIVERSITET