Föreläsning 2
Grunderna i VHDL, ModelSim

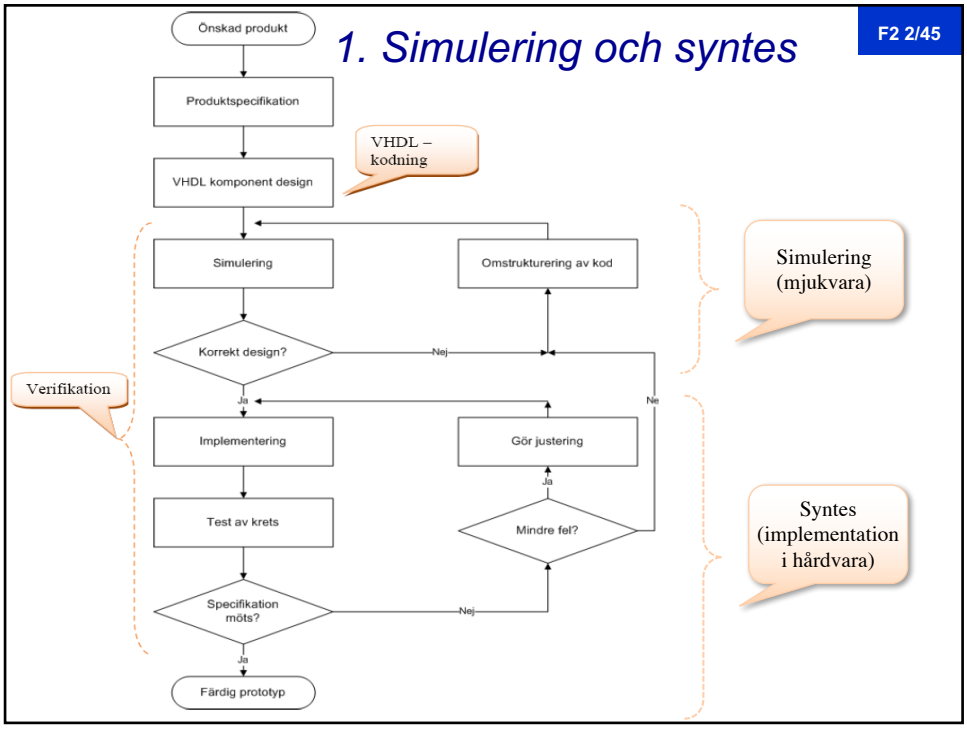
Erik Agrell 2017-08-30

Innehåll:

1. Simulering och syntes
2. VHDLs byggstenar
3. Parallell VHDL
4. Sekventiell VHDL

Bilder av E. Agrell, A. Linde, A. Crayvenn, S. Farinam

1. Simulering och syntes



Arbetsgång i ModelSim

1. Skapa projektmapp
2. Starta ModelSim
3. Create Project, ange namn och mapp
4. New file, ange namn (samma som entiteten)
5. Dubbelklicka på filnamnet
6. Skriv (eller kopiera) VHDL-kod, spara 
7. Kompilera 
8. Felmeddelande? Debugga!
9. Simulera 
10. Markera rätt entitet, under "work"
11. Högerklicka signaler, Add Wave
12. Skriv kommandon: FORCE, RUN

Simuleringskommandon

Sätt x till 1 hela tiden:

```
force x 1
```

Sätt x först till 0, sedan till 1 efter 50 ns, till 0 igen efter 150ns, etc.:

```
force x 0 0ns, 1 50ns, 0 150ns, ...
```

Gör x periodisk med periodtiden 200ns (bra för klocksignaler):

```
force x ... –repeat 200ns
```

Simulera i 500ns:

```
run 500ns
```

(eller bara `run 500` eftersom `ns` är default)

Radera kurvorna och börja om från 0ns vid nästa `run` (som annars fortsätter föregående simulering):

```
restart -force
```

Licensproblem i ModelSim?

```
# vsim -gui
# Start time: 10:05:16 on Sep 02,2014
# ** FATAL ERROR: ModelSim PE Student Edition licensing failure
due to one or more problems with the license key such as:
# - it is not found
# - it has expired
# - it is not for this user
# - it is not for this computer
# - it is not for this version of ModelSim PE Student Edition.
#
# Please go to http://www.model.com and download an updated copy
of the ModelSim PE Student Edition.
# Error loading design

ModelSim>
```

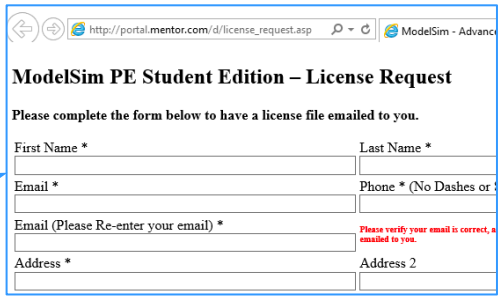

Orsaker och lösningar

```
# Please go to http://www.model.com and download an updated copy
of the ModelSim PE Student Edition.
# Error loading design
```

Orsak 1: Giltig licens saknas

Lösning:

- Beställ en gratislicens i samband med installationen och lägg filen på rätt ställe



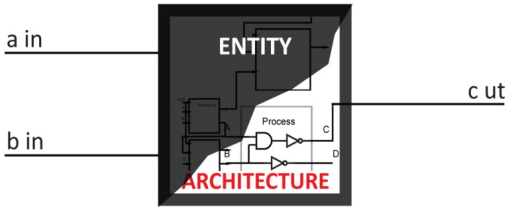
Orsak 2: MentorGraphics tror att du försöker köra två ModelSim på samma licens

Lösningar:

- Avsluta alla andra ModelSim med samma licens
- Om det inte hjälper: Stäng av internet (behövs t.ex. om ModelSim nyligen kraschade)

2. VHDLs byggstenar

- Entiteten beskriver kopplingen till omgivningen
- Arkitekturen beskriver funktionen



- Entiteter och arkitekturer har namn (identifierare)

VHDL-komponent

Vi återvänder till exemplet med och-grinden:

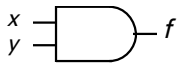
```
-- Simple AND gate
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is port (
  x,y: in std_logic;
  f: out std_logic);
end entity;

architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```

kommentar

bibliotek

entitet

arkitektur



Namn i VHDL-kod

- Namn består av bokstäver, siffror och understreck. Måste börja med bokstav.
- Ingen skillnad mellan stora och små bokstäver
- Två bindestreck (--) betyder att resten av raden är kommentar
- Använd inte åääö, inte ens i kommentarer
- Inga mellanslag i filnamn eller sökvägar (path)
- Undvik mycket långa sökvägar

Bibliotek

Ger tillgång till inbyggda definitioner och funktioner

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

Entitet

Definierar gränssnittet: in- och utsignaler

```
entity and_gate is port (
  x,y: in std_logic;
  f: out std_logic);
end entity;
```


Entitetens syntax

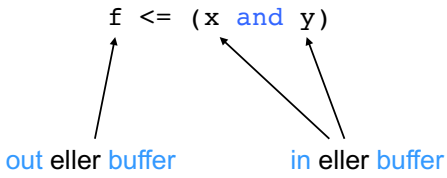
```
entity <komponentnamn> is port (
   <signalnamn>: <riktning> <datatyp>;
   ...
   <signalnamn>: <riktning> <datatyp>);
end entity;
```

Obs parentesens placering!

- Riktning kan vara in, out eller buffer.
- Datatyp kan vara std_logic eller std_logic_vector (Även bit, boolean och integer förekommer.)

in, out och buffer

- in är en ingång. Den kan *inte* tilldelas värden i arkitekturen.
- out är en utgång. Dess värde kan *inte* läsas i arkitekturen.
- buffer är en utgång som är kopplad till ett minneselement. Dess värde *kan* läsas i arkitekturen.



std_logic

Datatypen std_logic kan anta följande värden i simulering:

```
'U' - Uninitialized,        'X' - Forcing Unknown,
'0' - Forcing 0,            '1' - Forcing 1,
'Z' - High Impedance,       'W' - Weak Unknown
'L' - Weak 0                'H' - Weak '1'
'-' - Don't care
```

Bra att känna till att de finns – men vi behöver egentligen bara 0, 1 och U

Arkitektur

Beskriver den interna funktionen

```
architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```


Arkitekturens syntax

Arkitekturnamnet är inte viktigt – syns inte utåt

```
architecture <arkitekturnamn> of <komponentnamn> is

   signal <signal_namn>: <datatyp>;
   ...

begin

...

end architecture;
```

Samma som entitetens namn

Alla satser här exekveras samtidigt! Ordningen spelar ingen roll.

Filhuvud

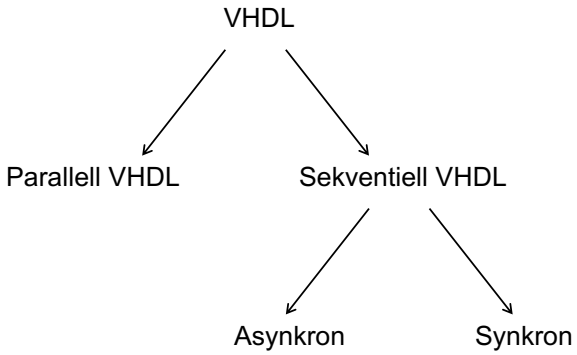
```
----------------------------------------------
-- Company:           Company X
-- Engineer:          Alexander Scott Crayvenn
-- Create Date:       19:38:20 03/30/2009
-- Design Name:       NOR.vhd
-- Module Name:       nor - structure
-- Project Name:      Exempel
-- Target Devices:    Cyclone II

-- Tool versions:     Quartus II, ModelSim PE 6.5
-- Description:       Synthesizable model for
--                    an nor gate
-- Errors:            None
-- Additional Comments:
----------------------------------------------
```


Konventioner

- En filkatalog innehåller ett projekt
- Ett projekt kan innehålla flera VHDL-filer
- En VHDL-fil innehåller en entitet (en komponent)
- En VHDL-fil har samma namn som sin entitet

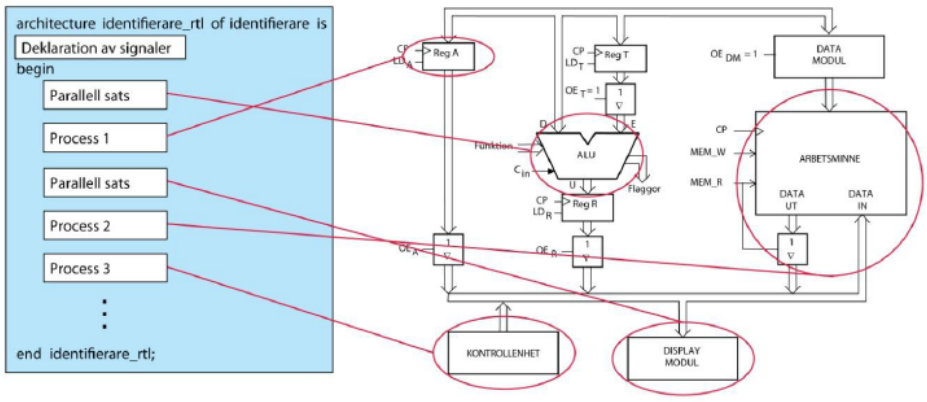
Viktigt!

Olika sorters VHDL-kod



Sorterna kan kombineras i samma komponent

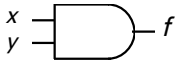
Kombination av parallell och sekventiell VHDL



- Alla satser och processer i architecture exekveras samtidigt (parallellt)

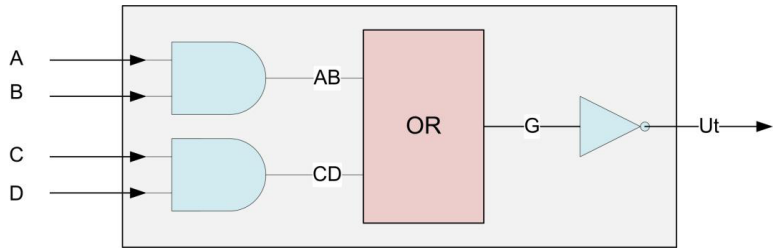
3. Parallell VHDL

Det tidigare AND-exemplet är parallell VHDL (ingen process):

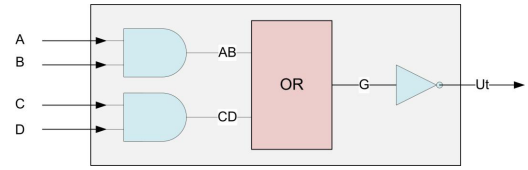



```
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is port (
  x,y: in std_logic;
  f: out std_logic);
end entity;

architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```


Exempel med interna signaler



Interna signaler

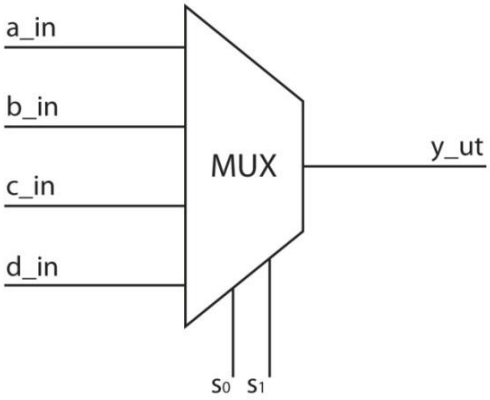



```
library ieee;
use ieee.std_logic_1164.all;

entity AOI is port (
  A,B,C,D: in std_logic;
  Ut: out std_logic);
end entity;

architecture rtl of AOI is
  signal AB,CD,G: std_logic;
begin
  AB <= A and B;
  CD <= C and D;
  G <= AB or CD;
  Ut <= not G;
end architecture;
```

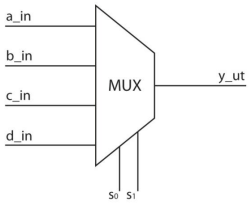
Man kan deklarera flera signaler på samma rad

Finns endast inuti arkitekturen

Två sätt att realisera en multiplexer



Bibliotek och entitet:




```
-- F2: multiplexer
library ieee;
use ieee.std_logic_1164.all;

entity mux is port (
  a_in, b_in, c_in, d_in: in std_logic;
  s0, s1: in std_logic;
  y_ut: out std_logic);
end entity;
```


Metod 1: Grindnivå

Signaltilldelning med tilldelningsoperatorn <= och logiska operatorer:

```
architecture arch of mux is
begin
  y_ut <= (a_in and not s1 and not s0) or
          (b_in and not s1 and s0) or
          (c_in and s1 and not s0) or
          (d_in and s1 and s0);
end architecture;
```

Vektorer

- I stället för en lista av `std_logic`, är det bekvämt att använda vektorer
- Deklareras med `std_logic_vector(... downto 0)` ⇨ MSB först
- Enkla citattecken för bitar ('0', '1') och dubbla för vektorer ("0000").
  OBS: Typografiska citattecken (’0’, “00”) funkar inte. Varning för att kopiera kod från ”smarta” ordbehandlare!
- En tilldelning kan gälla en hel vektor:

```
f <= "11111110"
f <= x"FE"
```

eller delar av den:

```
f(0) <= '1'
f(2 downto 0) <= "010"
```


Metod 2: WITH

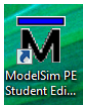
```
architecture arch of mux is
  signal s: std_logic_vector(1 downto 0);
begin
  s <= s1 & s0;
  with s select
    y_ut <= a_in when "00",
            b_in when "01",
            c_in when "10",
            d_in when "11";
end architecture;
```

&: konkatenering

with...select:
s jämförs med
flera värden

Koden ovan innehåller ett vanligt fel. Vilket?

Demo

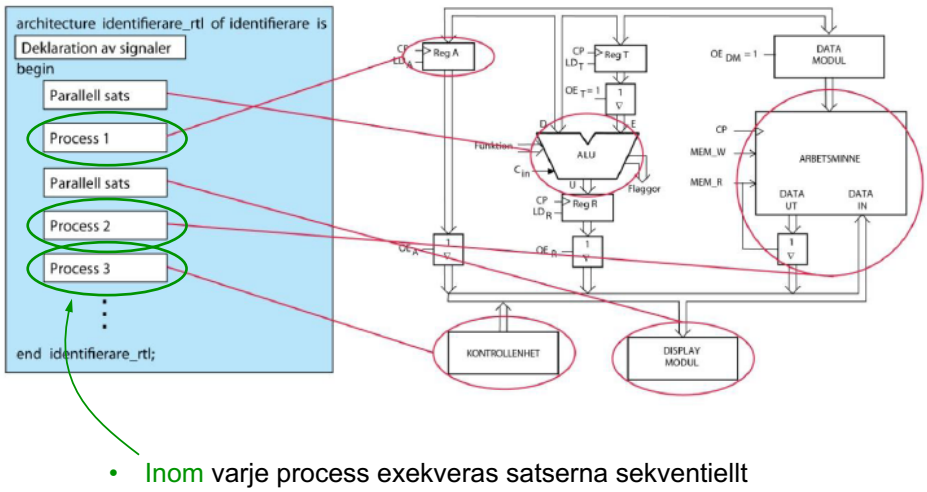


Rättad kod

```
architecture arch of mux is
  signal s: std_logic_vector(1 downto 0);
begin
  s <= s1 & s0;
  with s select
    y_ut <= a_in when "00",
            b_in when "01",
            c_in when "10",
            d_in when others;
end architecture;
```

Vi använder `others` i stället för `"11"` för att täcka alla alternativ

4. Sekventiell VHDL



- Inom varje process exekveras satserna sekventiellt

När exekveras en process?

- I syntes (hårdvara) är komponenterna påslagna hela tiden och anpassar utsignaler efter insignaler
- I simulering är det omöjligt att exekvera hela tiden. Man vill exekvera en process när det behövs och inte annars.

⇒ Användaren behöver tala om för simulatorn när det är dags att exekvera processen

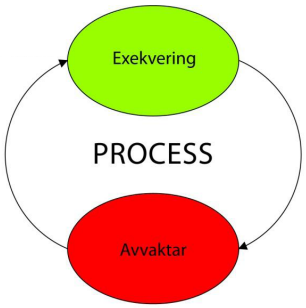
Syntaxen för en process

```
[process_namn:] process (sensitivitetslista)
<variabel namn :data_typ {:= initial_värde;}
begin
<sekventiella satser>
end process [process_namn];
```

- I processens sensitivitetslista ingår de signaler som skall påverka processens igångsättning vid simulering
- Sensitivitetslistan ignoreras vid syntes

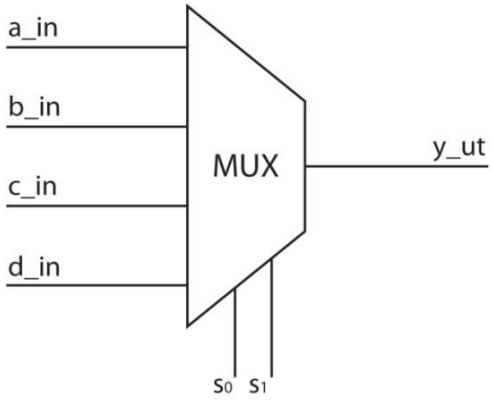
Process och sensitivitetslista

En process simuleras då någon signal i sensitivitetslistan ändras



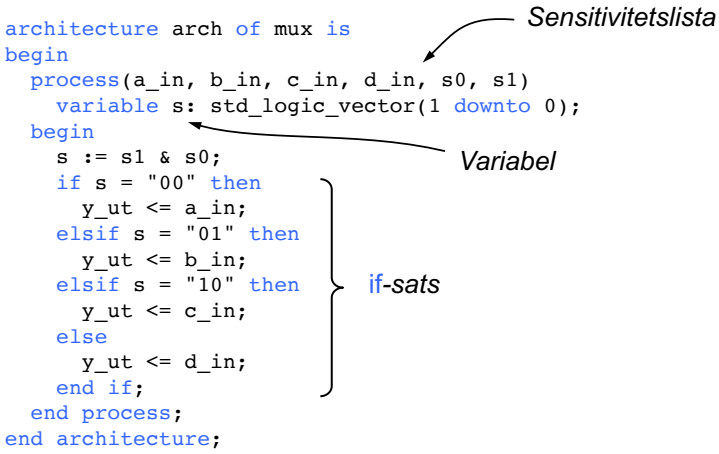
När signalerna i sensitivitetslistan är oförändrade, simuleras inte processen. Utsignalerna ligger kvar oförändrade.

Två sekventiella sätt att realisera MUXen



Metod 3: IF

Vi kan använda samma bibliotek och entitet som förut




```
architecture arch of mux is
begin
  process(a_in, b_in, c_in, d_in, s0, s1)
    variable s: std_logic_vector(1 downto 0);
  begin
    s := s1 & s0;
    if s = "00" then
      y_ut <= a_in;
    elsif s = "01" then
      y_ut <= b_in;
    elsif s = "10" then
      y_ut <= c_in;
    else
      y_ut <= d_in;
    end if;
  end process;
end architecture;
```


Syntaxen för en if-sats

```
if villkor then uttryck;
     else if nytt villkor
          then något annat;
     else nytt uttryck;
end if;
```

- Varje if avslutas med end if
- else if kan sammanskrivas som elsif
- if-satser är sekventiella och får inte placeras i den parallella delen av VHDL-koden

Signaler och variabler

Signaler

- deklareras i den parallella delen
- används både i parallella och sekventiella delen
- tilldelas med <=
- *i parallell kod:* uppdateras hela tiden
- *i sekventiell kod:* uppdateras först då processen avslutas
- I hårdvara realiseras signaler som ledningar

Variabler

- deklareras i den sekventiella delen (process)
- används i den sekventiella delen
- tilldelas med :=
- uppdateras när den tilldelas värde
- existerar inte utanför processen
- I hårdvara realiseras variabler som register

Metod 4: CASE

```
architecture arch of mux is
begin
  process(a_in, b_in, c_in, d_in, s0, s1)
    variable s: std_logic_vector(1 downto 0);
  begin
    s := s1 & s0;
    case s is
      when "00" =>
        y_ut <= a_in;
      when "01" =>
        y_ut <= b_in;
      when "10" =>
        y_ut <= c_in;
      when others =>
        y_ut <= d_in;
    end case;
  end process;
end architecture;
```

case-*sats*

Syntaxen för en case-sats

case är den sekventiella motsvarigheten till with

```
CASE selectorSignal IS
   WHEN value1 =>
      sequential code;
   WHEN value1 =>
      sequential code;
  [WHEN value2 =>
      sequential code;]
  [WHEN others =>
      sequential code;]
END CASE [Case label];
```

Om when*-satserna inte täcker alla möjliga värden för* `selectorSignal` *använder vi* others

Alla when-satser i samma case har samma prioritet

Exekvering i en process

- Inom varje process exekveras koden sekventiellt
- Endast den sista tilldelningen av varje signal i en process räknas
- Använd variabler i processer och tilldela motsvarande signal ett värde på slutet (”skuggvariabler”)

Exempel: NAND

```
architecture arch of nand_gate is
begin
  process(x,y)
  begin
    f <= x and y;
    f <= not f;
  end process;
end architecture;
```

Vi vill invertera resultatet. Varför fungerar inte detta?

Svar: `f` *har här samma värde som när processen startade – oavsett vad som hände tidigare i processen!*

Rättad kod

```
architecture arch of nand_gate is
begin
  process(x,y)
  variable z: std_logic;
  begin
    z := x and y;
    f <= not z;
  end process;
end architecture;
```

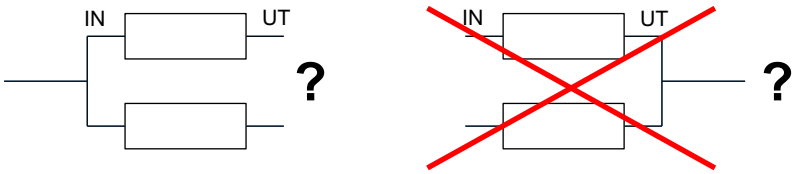
`z` *är en variabel, ingen signal*

Signalen `f` *tilldelas endast en gång. OK!*

Fast det finns minst två enklare sätt att realisera en NAND-grind

Koppla ihop utgångar?

Exempel: Om man bygger elektronik i hårdvara, är det OK att kombinera kretsar så här?



OK **Fel**

- Man kopplar inte ihop utgångarna från två kretsar!
- Det gäller i VHDL också: två satser/processer får inte ha samma utsignaler.

Sammanfattning

Parallell VHDL:

- I architecture, utanför processer
- All kod exekveras samtidigt, ordningen är oväsentlig
- Viktigast är att kunna tilldelningar (<=)

Sekventiell VHDL:

- Sker i processer
- Allt exekveras sekventiellt inom processer – om man använder variabler
- Alla processer exeveras samtidigt

Läs & lös

Läs:

- Lab-PM: inledning och kapitel 1
- Kretskonstruktion med VHDL: avsnitt 1–3.4 och 10.1–10.3

Lös:

- Simulera en XOR-grind i ModelSim genom att följa arbetsgången på bild 3 och modifiera koden på bild 20.
- Inlämningsuppgift 1 (deadline 8 sept)
- Lab-förberedelseuppgift 1.1

Anmälan till labgrupper öppnar idag kl 13:00 i Pingpong