LUNDS UNIVERSITET
Lunds Tekniska Högskola

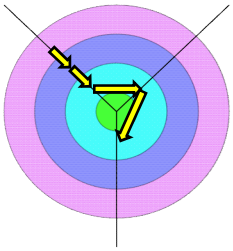
EITF65
Digitalteknik

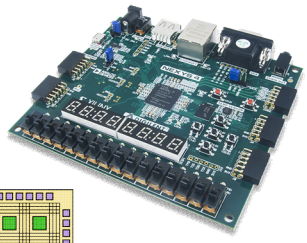
FÖRELÄSNING 16 – KOMBINATORISKA NÄT MED VHDL

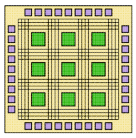
Dagens föreläsning

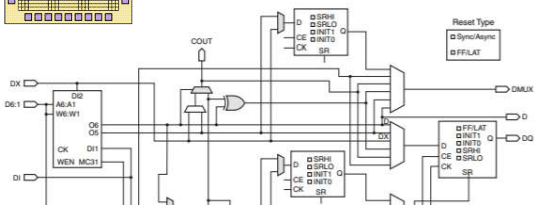
- Signaler och konstanter
- Signalbuss
- Tilldelning av signalvärde
- Process
- Port map
- Testbench

Förra gången







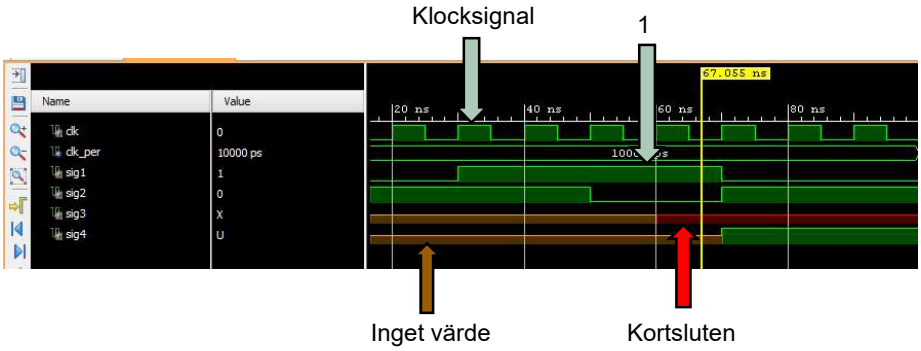


VIVADO

Förra gången

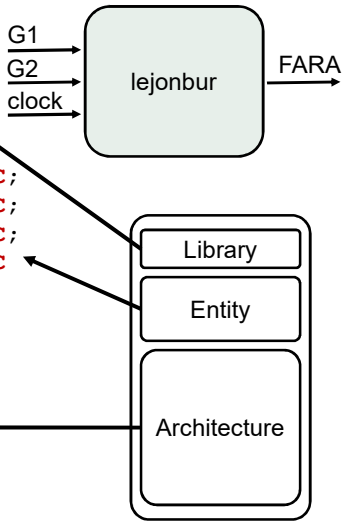
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

STD_LOGIC
STD_LOGIC_VECTOR



VHDL - exempel

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

entity lejonbur is
        port (
                G1      : in  STD_LOGIC;
                G2      : in  STD_LOGIC;
                FARA    : out STD_LOGIC;
                clock   : in  STD_LOGIC
        );
end lejonbur;

architecture behav of lejonbur is
begin
        FARA <= G1 and G2;
end behav;
```



Deklaration av signaler

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

entity lejonbur is
        port (
                G1      : in  STD_LOGIC;
                G2      : in  STD_LOGIC;
                FARA    : out STD_LOGIC;
                clock   : in  STD_LOGIC
        );
end lejonbur;

architecture behav of lejonbur is
        signal s1 : STD_LOGIC;
begin
        s1 <= G2;
        FARA <= G1 and s1;
end behav;
```

Endast läsbar i kretsen

Endast skrivbar i kretsen

Deklaration av signaler

Signalen finns endast inuti kretsen

Läs- & skrivbar

Deklaration av konstanter

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

entity lejonbur is
        port (
                G1      : in  STD_LOGIC;
                G2      : in  STD_LOGIC;
                FARA    : out STD_LOGIC;
                clock   : in  STD_LOGIC
        );
end lejonbur;

architecture behav of lejonbur is
        constant c1 : STD_LOGIC := '1';
        signal s1 : STD_LOGIC;
begin
        s1 <= G2 and c1;
        FARA <= G1 and s1;
end behav;
```

Deklaration av konstant

Signaler och konstanter

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

entity lejonbur is
        port (
                G1      : in  STD_LOGIC;
                G2      : in  STD_LOGIC;
                FARA    : out STD_LOGIC;
                clock   : in  STD_LOGIC
        );
end lejonbur;

architecture behav of lejonbur is
        constant c1 : STD_LOGIC := '1';
        signal s1 : STD_LOGIC;
begin
        s1 <= G2 and c1;
        FARA <= G1 and s1;
end behav;
```

Objekttyp
STD_LOGIC
STD_LOGIC_VECTOR
INTEGER

Tilldelning av värde för constant

'1' logisk 1:a
1 heltalet 1

Tilldelning av värde för signal

Egendefinierade typer

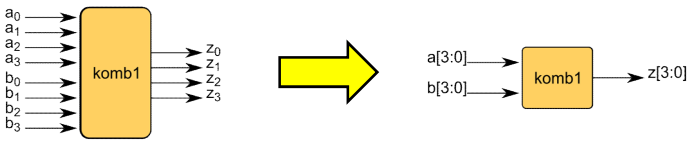
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lejonbur is
        port (
                inp     : in  STD_LOGIC;
                o       : out STD_LOGIC);
end lejonbur;

architecture behav of lejonbur is
        type my_type is (on, off);
        signal s1 : my_type;
begin
        if inp = '1'
                s1 <= on;
        else
                s1 <= off;
        end if;
        ...
```

Deklaration av ny signaltyp

Möjliga värden för typen

Användbart vid beskrivning av FSM

Signalbuss




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

entity lejonbur is
   port (
      a0 : in STD_LOGIC;
      a1 : in STD_LOGIC;
      a2 : in STD_LOGIC;
      a3 : in STD_LOGIC;
      b0 : in STD_LOGIC;
      ...
      z0 : out STD_LOGIC;
      ...);
end lejonbur;
```

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

entity lejonbur is
   port (
      a : in STD_LOGIC_VECTOR(3 downto 0);
      b : in STD_LOGIC_VECTOR(3 downto 0);
      z : out STD_LOGIC_VECTOR(3 downto 0)
    );
end lejonbur;
```

Användning av signalbuss

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

entity sig_vector is
   port (
      a : in STD_LOGIC_VECTOR(3 downto 0);
      b : in STD_LOGIC_VECTOR(3 downto 0);
      z : out STD_LOGIC_VECTOR(7 downto 0)
    );
end sig_vector;

architecture behav of lejonbur is
      signal s1, s2 : STD_LOGIC_VECTOR(3 downto 0);
begin
      s1 <= "0010";
      s2(2 downto 0) <= a(1 downto 0) & b(1);
      s2(3) <= '1';

      z <= s2 & s4;
end behav;
```

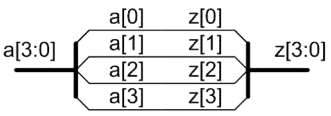
För att sätt alla bitar till ett visst värde
`z <= (others => '0');`

Konkatenera signaler

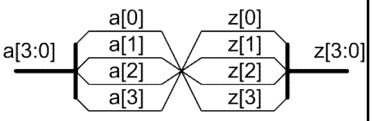
Observera skillnaden
`'1'` en bit
`"0010"` flera bitar

3 downto 0 eller 0 to 3?

```
entity komb3 is
        port(
        a : in STD_LOGIC_VECTOR(3 downto 0);
        z : out STD_LOGIC_VECTOR(3 downto 0));
end entity;
```




```
entity komb3 is
        port(
        a : in STD_LOGIC_VECTOR(3 downto 0);
        z : out STD_LOGIC_VECTOR(3 to 0));
end entity;
```

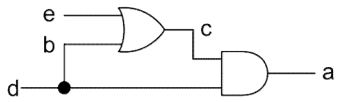


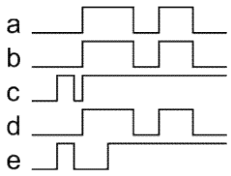

```
architecture data_flow of komb2 is
begin
        z <= a;
end architecture;
```

Concurrent Signal Assignment

Signalvärden uttrycks med Booleska operationer

```
and, or, xor, nor, not, ...
```






```
architecture data_flow of comb2 is
begin
   a <= d and c;
   b <= d;
   c <= b or e;
end architecture;
```

Använd parenteser!

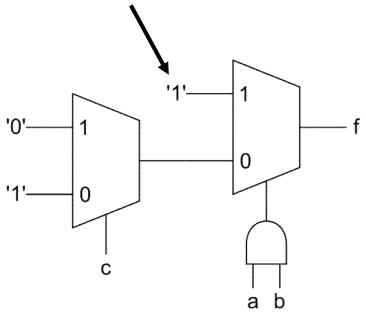
```
a <= (b or e) and d;
```


Conditional Signal Assignment

En signal får värde beroende på villkor likt if-else

```
architecture data_flow of comb2 is
begin
   f <= '1' when (a='1' and b='1') else
     '1' when c='0' else
     '0';
end architecture;
```

Villkoret `(a=' 1' and b='1')` har högst prioritet

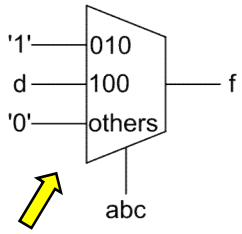


Selected Signal Assignment

En signal får värde beroende på villkor likt case/switch

```
architecture data_flow of comb2 is
begin
   with abc select
      f <= '1' when "010",
      d   when "100",
      '0' when others;
end architecture;
```

Alla andra fall ex.
"000", "111", ...



Ingen prioritet mellan olika signaler

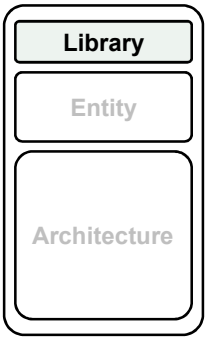
Heladderare

```
library IEEE;
use IEEE.std_logic_1164.all;
entity full_adder is
        port(a, b, c_in : in STD_LOGIC;
             z, c_out : out STD_LOGIC);
end entity;
architecture data_flow of full_adder is
        signal temp : STD_LOGIC;
begin
        temp <= a xor b;
        z <= temp xor c_in;
        c_out <= (a and b) or (c_in and temp);
end architecture;
```

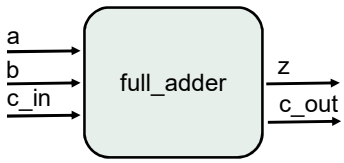
Heladderare - Library

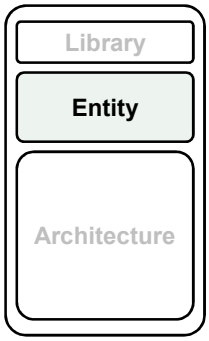
För att använda
STD_LOGIC och **STD_LOGIC_VECTOR**

```
library IEEE;
use IEEE.std_logic_1164.all;
```



Heladderare - Entity






```
entity full_adder is
        port(a, b, c_in : in STD_LOGIC;
             z, c_out : out STD_LOGIC);
end entity;
```

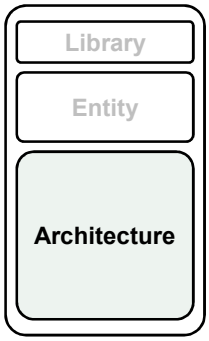
Heladderare - Architecture

Arkitekturens namn

```
architecture data_flow of full_adder is
        signal temp : STD_LOGIC;
begin
        temp <= a xor b;
        z <= temp xor c_in;
        c_out <= (a and b) or (c_in and temp);
end architecture;
```

Alla tilldelning sker parallellt (samtidigt)

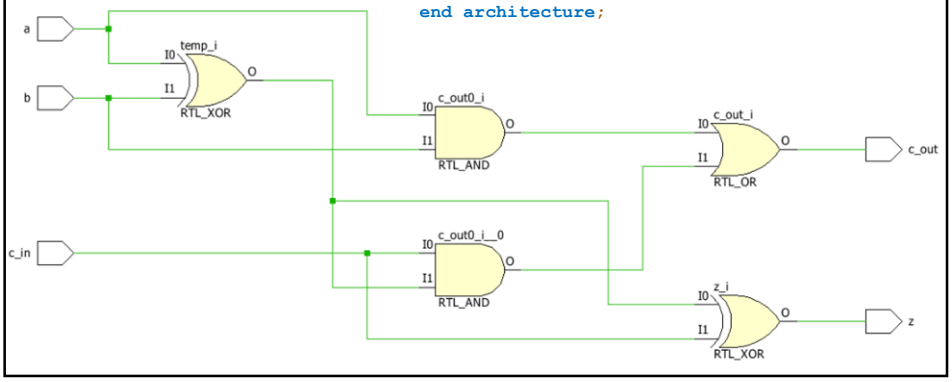
Deklaration av interna signaler



Heladderare – Vivado

```
entity full_adder is
        port(
        a, b, c_in : in STD_LOGIC;
        z, c_out : out STD_LOGIC);
end entity;
```

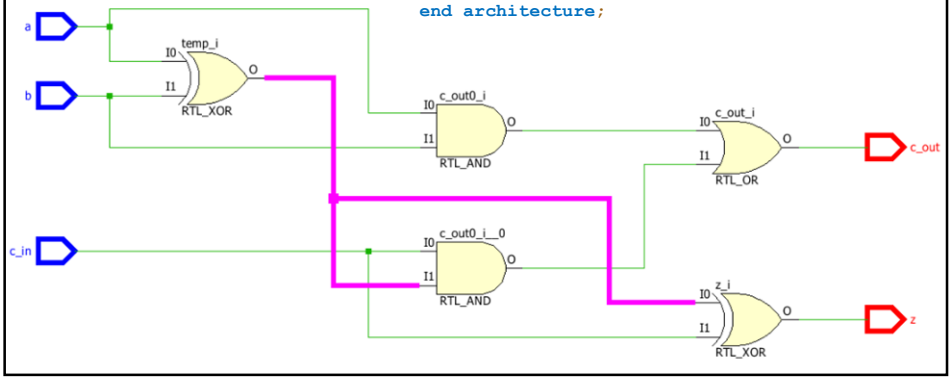
```
architecture data_flow of full_adder is
        signal temp : STD_LOGIC;
begin
        temp <= a xor b;
        z <= temp xor c_in;
        c_out <= (a and b) or (c_in and temp);
end architecture;
```



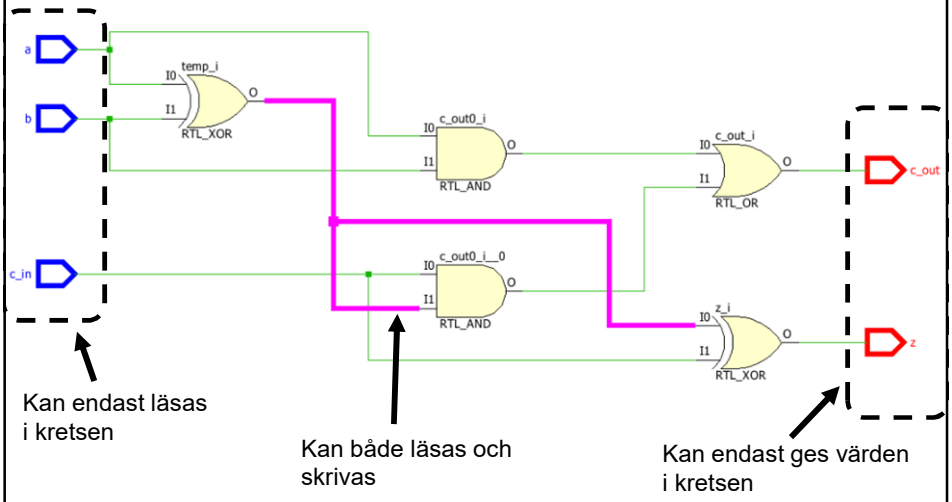
Heladderare - Signaler

```
entity full_adder is
        port(
        a, b, c_in : in STD_LOGIC;
        z, c_out : out STD_LOGIC);
end entity;
```

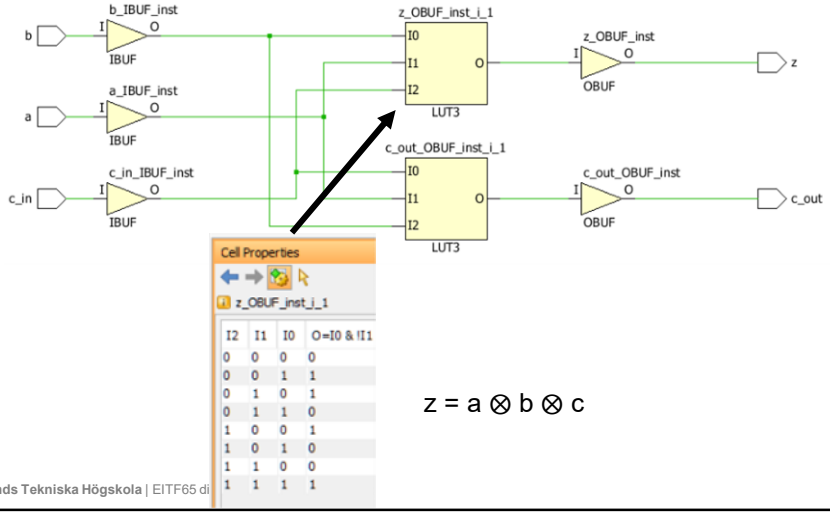
```
architecture data_flow of full_adder is
        signal temp : STD_LOGIC;
begin
        temp <= a xor b;
        z <= temp xor c_in;
        c_out <= (a and b) or (c_in and temp);
end architecture;
```



In- och utsignaler, interna signaler



Heladderare - FPGA



$z = a \otimes b \otimes c$

Ge alltid signaler ett värde

```
a <= '1' when b = '0';
```

`a` blir ett minneselement

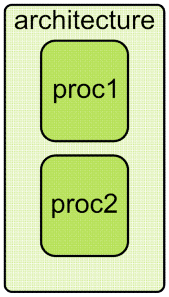
```
a <= '1' when b = '0' else '0';
```

`a` blir en ledning med värdet `b'`

Process

- Beskriva beteendet av en digital krets på högre abstraktionsnivå
- LIKNAR traditionell sekventiell programmering
- IF-ELSE och CASE satser kan användas

```
..
architecture behav of system1 is
          ..
begin
  proc1: process(..)
  begin
    ..
  end process;

  proc2: process(..)
  begin
    ..
  end process;
end architecture;
```



Alla processer utförs parallellt

Process - Syntax

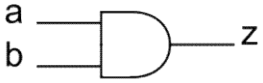
sensitivity list

Etikett för processen

```
proc1: process(..)
begin
  ..
end process;
```

```
and2: process(a, b)
begin
  if a and b then
    z <= '1';
  else
    z <= '0';
  end if;
end process;
```

Signalerna a och b triggar processen

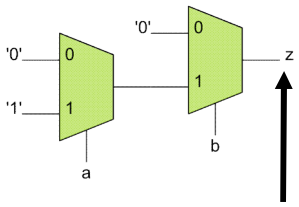


Viktigt att utsignalen alltid får ett värde

Process - Signaltilldelning

I en process kan en signal tilldelas värden på flera ställen utan att det blir en kortslutning

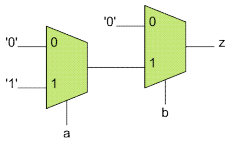
```
proc2: process(a, b)
begin
  z <= '0';
  if a = '1' then
   z <= '1';
  end if;
  if b = '0' then
   z <= '0';
  end if;
end process;
```



z kommer ENDAST att anta det värde som den tilldelas sist i processen

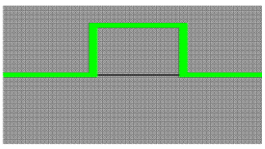
Process – Sekventiellt?

```
proc2: process(a, b)
begin
  z <= '0';
  if a = '1' then
   z <= '1';
  end if;
  if b = '0' then
   z <= '0';
  end if;
end process;
```



om a = '1' och b='0'

Programmering

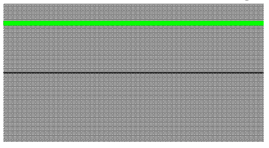


Sekventiella strukturer underlättar implementering av algoritmer

MEN

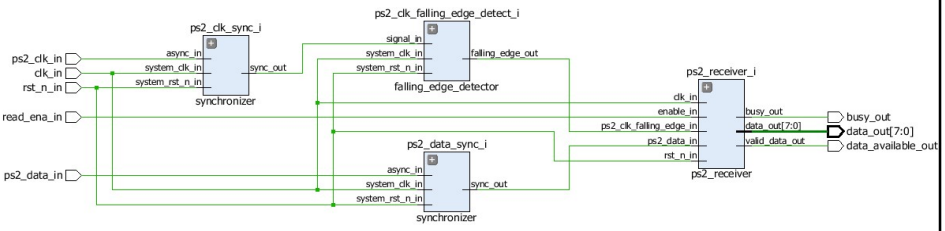
glöm inte att det är hårdvara som genereras

Hårdvarubeskrivning



Konstruktion av större system

- Dela upp i mindre komponenter
- Varje komponent konstrueras och testas fristående
- Komponenter sätts ihop till större



Koppla samman delkretsar i VHDL

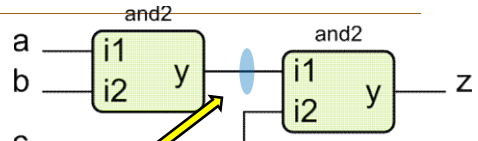
```
architecture struct of example is

  component full_adder is
    port(
        a, b, c_in : in STD_LOGIC;
        z, c_out : out STD_LOGIC);
  end component;

begin
  comp1:full_adder
    port map(...);
  ...
end architecture;
```

Deklaration av komponent

Instansiera och koppla ihop signaler

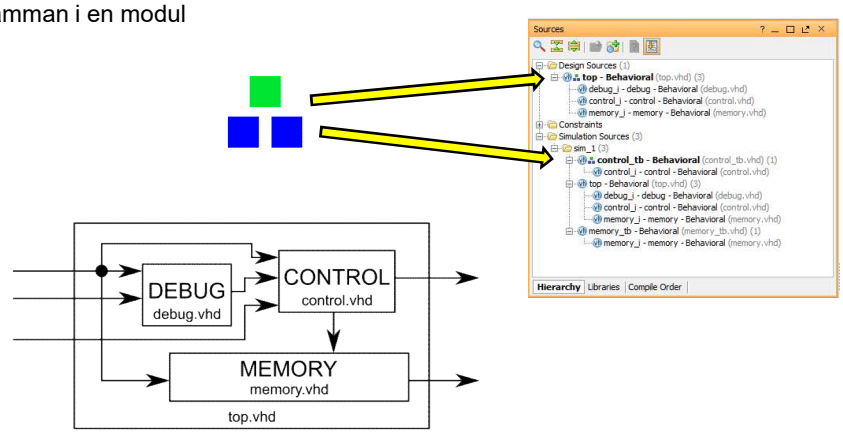
Port Map - exempel

```
entity ex_port_map is
port(a, b, c : in STD_LOGIC;
     z : out STD_LOGIC);
architecture struct of ex_port_map is

  component and2 is
    port(i1, i2 : in STD_LOGIC;
         y : out STD_LOGIC);
  end component;

  signal intern1 : STD_LOGIC;
begin
  comp1:and2 port map(
      i1 => a,
      i2 => b,
      y => intern1);

  comp1:and2 port map(
        i1 => intern1,
        i2 => c,
        y => z);
end architecture;
```



Topp-modul

Vid design av ett system med mer än en komponent måste dessa kopplas samman i en modul

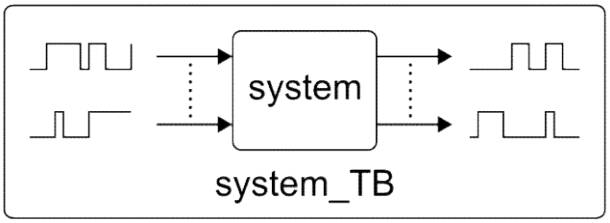


Testbench

Inga in- eller utsignaler

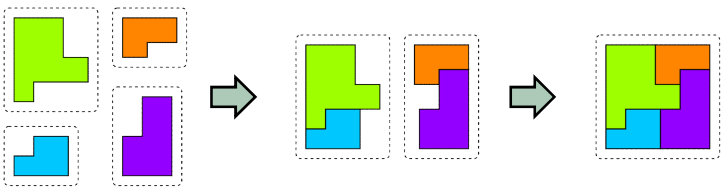
Innehåller en instans av modulen som ska testas

Genererar insignalsmönster



Verifiering

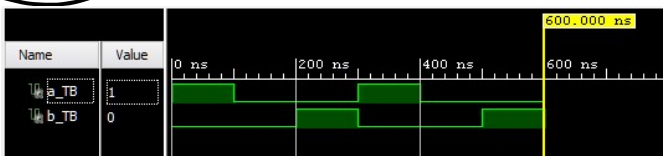
En testbench för varje komponent och delsystem



Generera testsignaler

```
library ieee;
use ieee.STD_LOGIC_1164.all;

entity system_TB is
end entity;

architecture arch of system_TB is
    component system is
        port(a, b : in STD_logic;
             f : out STD_LOGIC);
    end component;

    signal a_TB, b_TB, f_TB : STD_LOGIC
begin
    DUT:system port map(
        a => a_TB,
        b = b_TB,
        f => f_TB
    );
```

```
signal_gen:process
begin
    a_TB <= '1';
    b_TB <= '0';
    wait for 100 ns;
    a_TB <= '0';
    wait for 100 ns;
    b_TB <= '1';
    wait for 100 ns;
end process;
end architecture;
```



Sammanfattning

- Olika typer av signaler
  - användardefinierade typer
- Flera signaler kan sättas ihop till en signalbuss
- Olika sätt att tilldela signalvärden
  - prioriterad tilldelning kan ge långa logiska kedjor
- I process kan en signal tilldelas värden på flera ställen
- Komponenter kan kopplas ihop med port map
- Testbench
  - Generera insignaler till modulen som ska testas
  - Hela VHDL-språket kan användas


LUNDS UNIVERSITET
Lunds Tekniska Högskola