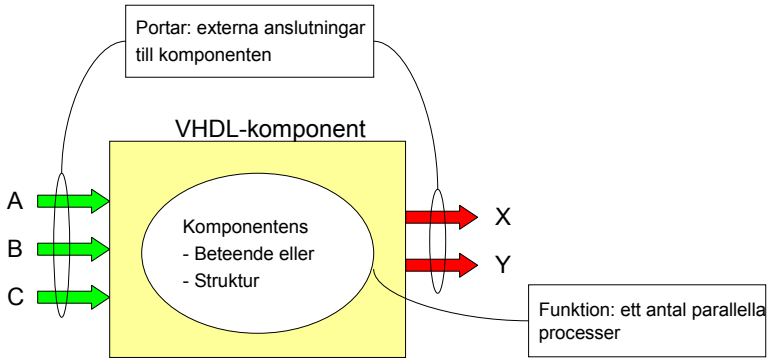
Grunder i VHDL

Innehåll

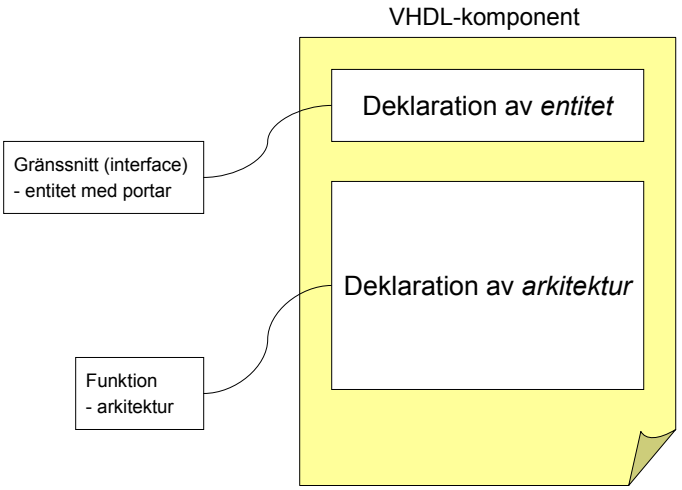
- Komponentmodell
- Kodmodell
- Entitet
- Arkitektur
- Identifierare och objekt
- Operationer för relationer

Komponentmodell

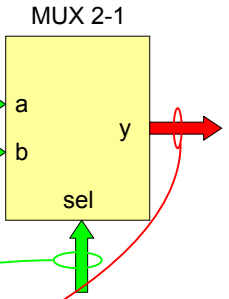
- Modell för att beskriva komponenter
  - Externt gränssnitt
  - Intern funktion



Kodmodell

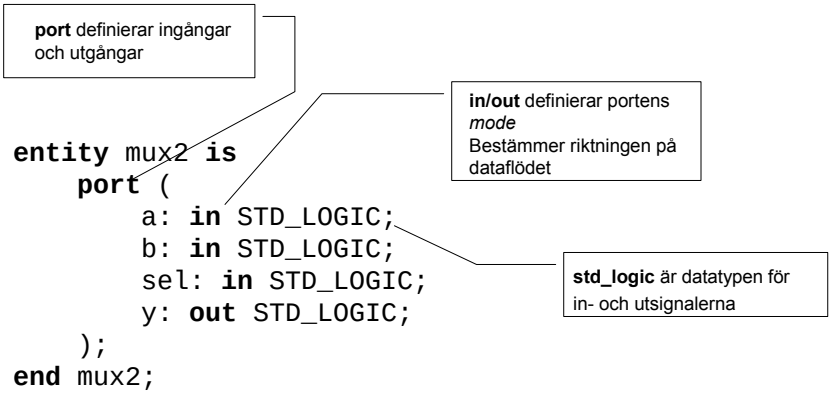


Deklarera VHDL-komponentens gränssnitt




```
entity mux2 is
    port (
        a: in STD_LOGIC;
        b: in STD_LOGIC;
        sel: in STD_LOGIC;
        y: out STD_LOGIC;
    );
end mux2;
```

VHDL-komponentens portar




```
entity mux2 is
    port (
        a: in STD_LOGIC;
        b: in STD_LOGIC;
        sel: in STD_LOGIC;
        y: out STD_LOGIC;
    );
end mux2;
```


Portar i VHDL

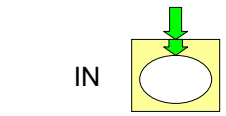
- Port-deklarationerna är det viktigaste i entitets-deklarationen
- Varje port representerar
  - Komponentens externa pinnar
- Varje port har
  - Port-namn
  - Mode
  - Datatyp

En identifierare som du skapar

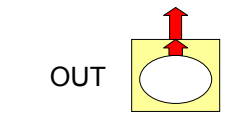
Riktning på data

Vilka värden porten kan anta

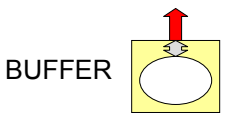
Portarnas olika *moder*



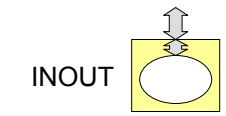
Signalen går enbart **in** till komponenten och drivs av någon annan komponent.
Insignalen används på höger sida av en tilldelning:
z <= a OR **`inport`**



Signalen går enbart ut från komponenten.
Det går ej att läsa utsignalens värde i komponenten
används på vänster sida i tilldelning: **`outport`** <= a OR b

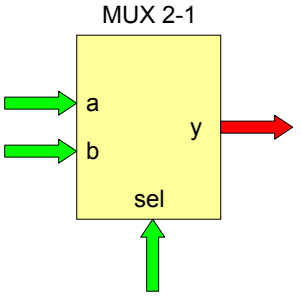


Signalen går enbart ut från komponenten.
Det går att läsa utsignalens värde i komponenten
Används på båda sidorna i tilldelning:
`buffer_port` <= a OR b;
z <= **`buffer_port`** OR c;



Signalen kan gå i båda riktningarna, antingen in eller ut
-signalens värde kan läsas av komponenten
-Signalen kan också drivas av andra komponenter
-Kan användas på båda sidor av en tilldelning

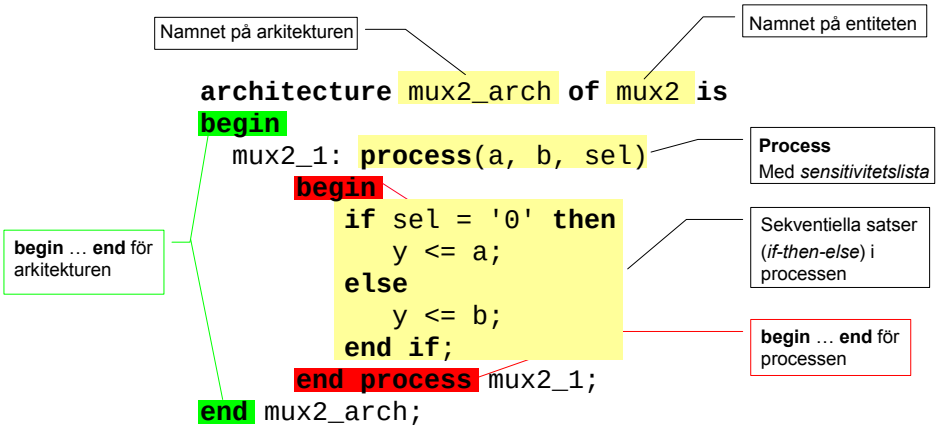
Beskrivning av funktionen i arkitekturen



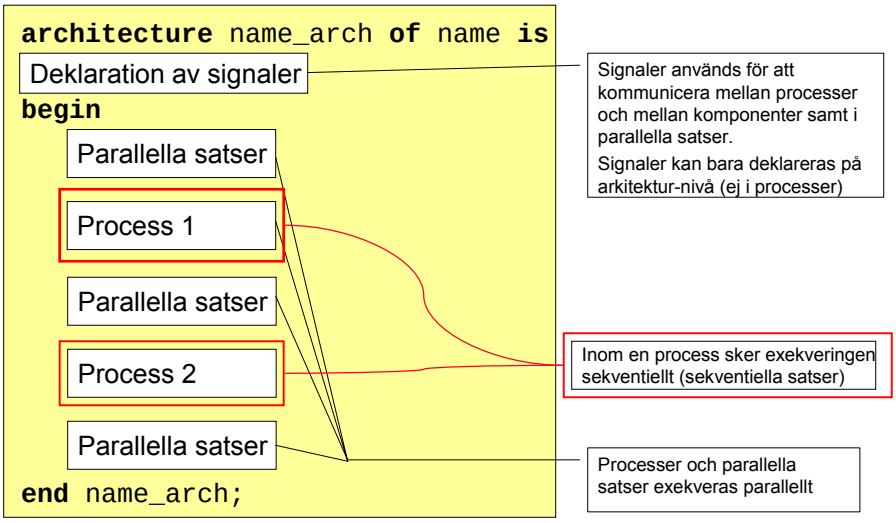

```
architecture mux2_arch of mux2 is
begin
  mux2_1: process(a, b, sel)
      begin
          if sel = '0' then
             y <= a;
          else
             y <= b;
          end if;
      end process mux2_1;
end mux2_arch;
```

I arkitekturen beskrivs funktionen:
Om *sel* är 0 så läggs värdet på insignalen *a* ut på utsignalen *y*.
I annat fall (*sel*=1) så läggs insignalen *b* ut på *y*.

Arkitektur-deklaration

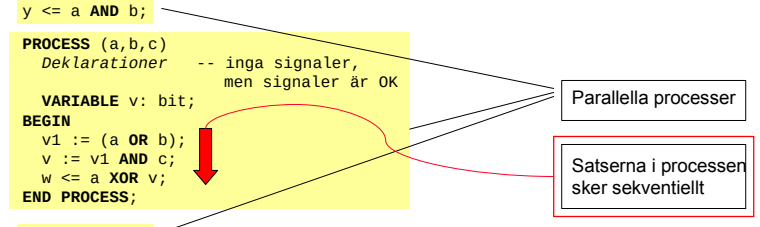


Struktur för arkitekturen



Exempel på parallella och sekventiella satser

```
ENTITY ename IS
  Ports( a, b, c: IN bit;
         y, z, w: OUT bit;
  Deklarationer   -- inga variabler tillåtna
END ename

ARCHITECTURE first OF ename IS
  Deklarationer   -- inga variabler,
                     men signaler är OK

BEGIN
  y <= a AND b;

  PROCESS (a,b,c)
    Deklarationer   -- inga signaler,
                       men signaler är OK
    VARIABLE v: bit;
  BEGIN
    v1 := (a OR b);
    v := v1 AND c;
    w <= a XOR v;
  END PROCESS;

  z <= c XOR b;

END first;
```



Identifierare i VHDL

- Identifierare
  - Är namn på saker DU skapar
  - T.ex. namn på arkitektur, entity, process, variabel, signal
  - Regler för namn
    - Får ej vara ett reserverat ord i VHDL (t.ex **for** och **if**)
    - Stora och små bokstäver tolkas lika (case-insensitive)
    - Första tecknet måste vara en bokstav
    - Sista tecknet får ej vara *underscore* (_)
    - Det får inte finnas två *underscore* efter varandra (__)

Dataobjekt i VHDL

- Objekt är saker som kan hålla ett värde
  - Objekten har klass och typ (*class*, *type*)
    - *Class* bestämmer vilka slags operationer man kan göra
    - *Type* bestämmer vilka värden som är giltiga
  - De kan tilldelas ett startvärde (endast för simulering)
  - De deklareras i *entity*, *architecture*, *process*, eller *package*

Klasser i VHDL

- *Signal*
  - Dess värde ändras som funktion av tiden
  - Den har en signaldrivare och kan ses som en fysisk ledning
- *Variable*
  - Dess värde ändras omedelbart vid tilldelning
  - Inga tidsbegrepp finns relaterade till den
- *Constant*
  - Dess värde kan inte ändras
- *File*
  - Värde från extern fil kan skrivas och läsas

Datatyper i VHDL

- VHDL har hårda krav på datatyper
  - Objekt av olika grundtyper kan inte tilldelas varandra direkt
  - Vid typkonvertering ska konverteringsfunktioner användas
- Två huvudkategorier av datatyper
  - Skalärer (*scalar*)
    - Kan anta ett enda värde
    - Exempel: enumeration, integer, float, physical
  - Sammansatta (*composite*)
    - Kan anta flera värden
    - Exempel: array, record

Skalärer

- Enumeration (uppräkningsbara)
  - En lista med distinkta värden en variabel kan anta
  - Ex: `type weekday = (mon, tue, wed, thu, fri, sat, sun);`
- Integer
  - En mängd heltal – positiva och negativa
  - En fördefinierad datatyp
  - Integer är av 32-bitar med tecken $-(2^{31}-1$ till $+(2^{31}-1)$
  - Ett begränsat område används vid beskrivning av hårdvara
    - Ex: `variable num: integer range -64 to 64`

Skalärer

- Flyttal
  - Datatypen för flyttal är den fördefinierade typen **real**
  - 32-bitars enkel precision
  - Används inte för hårdvarubeskrivningar
    - Resulterar i för komplex hårdvara
- Physical
  - Datatyp för fysikaliska storheter
    - Ex. time, mA, Volt
    - Har ingen betydelse vid beskrivning av hårdvara

Exempel på uppräkningsbara datatyper

- Fördefinierade datatyper (1076)
  - `type boolean is (FALSE, TRUE);`
  - `type bit is ('0','1');`
- Fördefinierade datatyper (1164)
  - Std_logic
  - Std_ulogic
  - Samt arrayer av dessa och dess under-typer
  - Tillgång till dessa får genom att inkludera biblioteket:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

Std_logic

Definition av std_logic

```
type std_ulogic is (       'U', -- Uninitialized
                           'X'  -- Forcing unknown
                           '0'  -- Forcing zero
                           '1'  -- Forcing one
                           'Z'  -- High impedance
                           'W'  -- Weak unknown
                           'L'  -- Weak zero
                           'H'  -- Weak one
                           '-');-- Don't care

subtype std_logic is resolved std_ulogic;
```

Skrivs först i VHDL-programkoden för att inkludera biblioteket

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


Sammansatta datatyper

- Arrayer
  - Exempel på deklaration av en bit-vektor om 8 bitar

```
signal s1: bit_vector(7 downto 0);
variable v1: bit_vector(7 downto 0);
```

  - Tilldela bit-vektor det binära talet 11010010

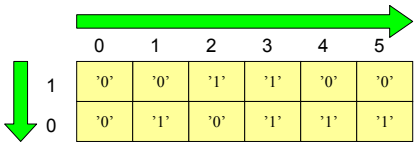
```
s1 <= "11010010";
v1 := "11010010";
```

Tolkas som den minst signifikanta biten

Tolkas som den mest signifikanta biten

Sammansatta datatyper

- Ex: två-dimensionell tabell
  - `type table6x2 is array (0 to 5, 1 downto 0) of bit;`
  - `constant mytable: table6x2 := ("00","01","10","11","01","01");`

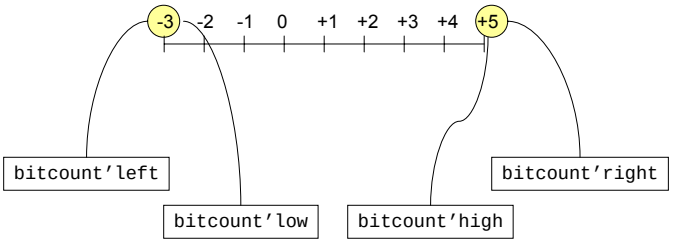


- Ex: Bitvektorer för binära, oktala och hexadecimala tal
  - `X"A3"  -- = B"1010_0011" för en 8-bitars vektor`
  - `O"27"  -- = B"010_111" för en 6-bitars vektor`

Attribut

- Attribut
  - Tillhandahåller information om en signal, variabel, datatyp, funktion, etc.
  - Exempel #1

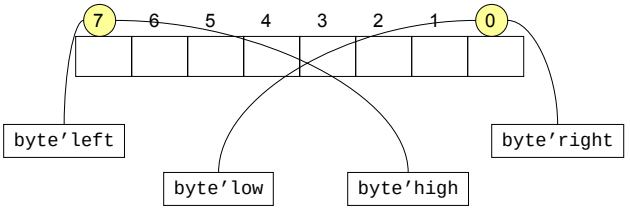
```
type bitcount is integer range –3 to +5;
```



Attribut

- Exempel #2

```
type byte is array (7 downto 0) of std_logic;
```



- Exempel #3

```
for i in byte'high downto byte'low loop
  v_byte(i) := '1';
end loop;
```


Operationer i VHDL

- Relationsoperatorer

Symbol	Operation
=	likhet
/=	olikhet
<	Mindre än
>	Större än
<=	Mindre än eller lika
>=	Större än eller lika

Aritmetiska operationer

Symbol	Operation
+	addidion
-	subtraktion
*	multiplikation
/	division
abs	absolutvärde
rem	rest
mod	modulus
**	exponent

Operationer som stöds av syntesverktyg

SLUT på Föreläsning 2

- Innehåll
  - Komponentmodell
  - Kodmodell
  - Entitet
  - Arkitektur
  - Identifierare och objekt
  - Operationer