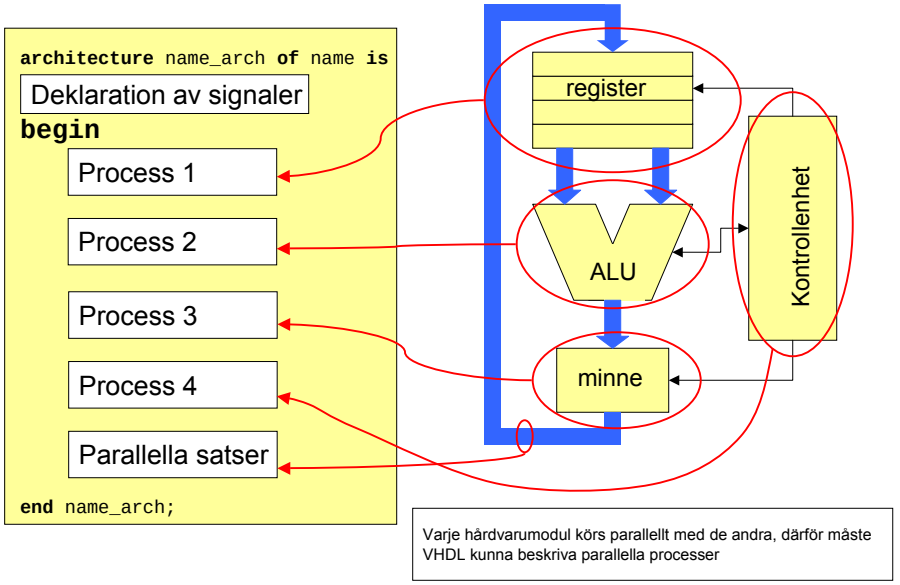
Parallell och sekventiell databehandling i VHDL

Innehåll

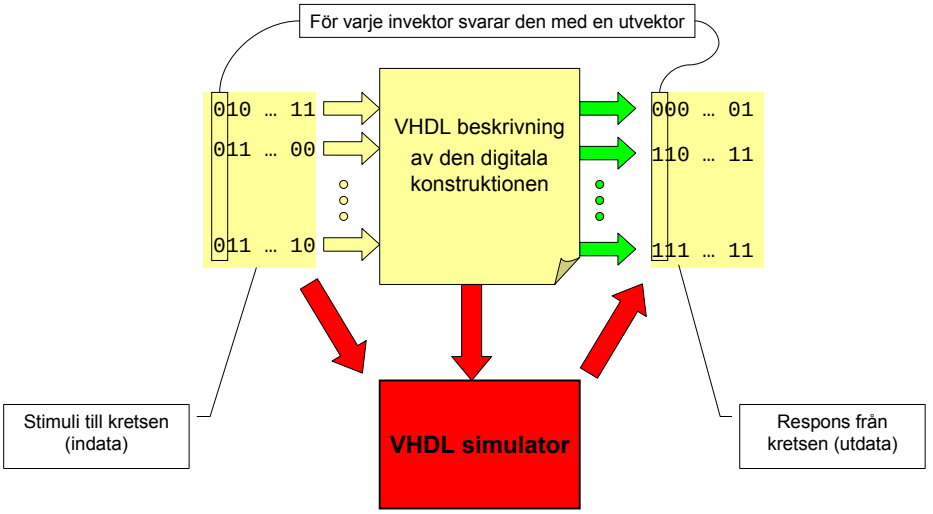
- Parallellitet i VHDL
- Fördröjningar
- Simuleringscykeln
- Aktivering av processer
- Parallella och sekventiella satser

Parallellitet i VHDL



Varje hårdvarumodul körs parallellt med de andra, därför måste VHDL kunna beskriva parallella processer

Simulering av VHDL program

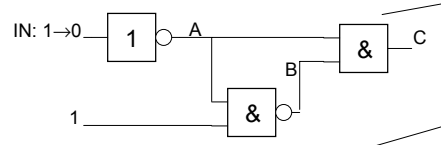


Simulering av sekventiella händelser

- Simuleringstid
  - Är den tid kretsen har simulerats (ej verklig tid)
- Delta-fördröjning
  - Används internt i simulatorn för att köa sekventiella händelser
  - Om man lägger till ett antal delta-fördröjningar till simuleringstiden så blir simuleringstiden densamma

Simulering av sekvens av händelser

- Exempel: Vad händer på utgången C ?



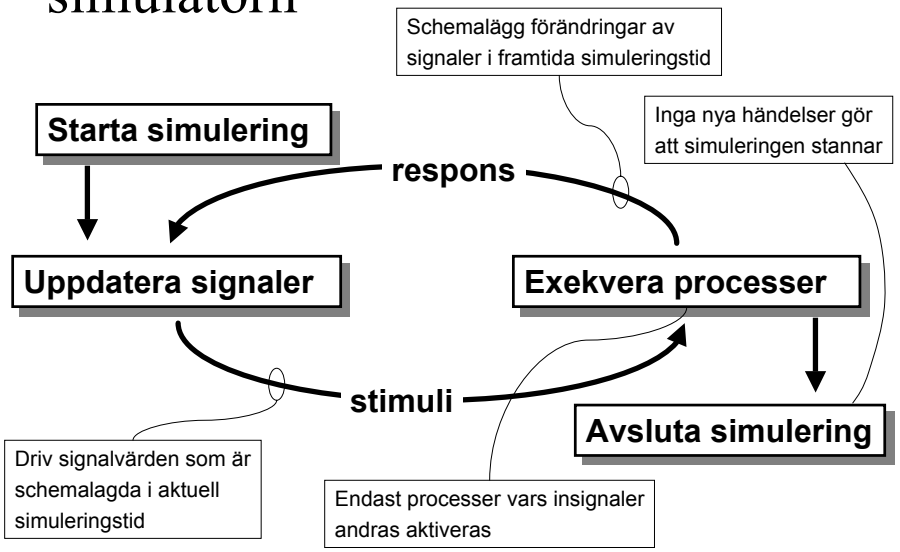
Antag att grindarna inte har någon fördröjning

Delta fördröjning används för sekvensering av händelser inom samma simuleringstid

simuleringstid	delta	händelse
0 ns	1	IN: 1 → 0 *inv* beräknar nytt värde för A
0 ns	2	A: 0 → 1 *nand* och *and* beräknar nya värden
0 ns	3	B: 1 → 0 C: 0 → 1 *and* beräknar nytt värde på C
0 ns	4	C: 1 → 0 inga nya händelser

Eftersom grindarna inte har någon fördröjning så är simuleringstiden 0 ns då nya värdet för C är beräknat

Simuleringscykeln för VHDL simulatorn



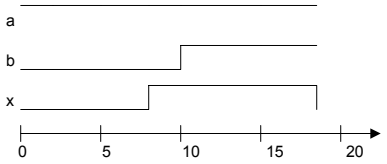
Schemaläggning av signaler

- Exempel #1, ”ändliga fördröjningar anges”

Tid	a, b, x	Drivare (av utgången *x*)	Kommentarer
0ns	1, 0, 0	(0, 0ns) (0, 5ns) (1, 8ns)	p1 körs p.g.a initiering
5ns	1, 0, 0	(0, 5ns) (1, 8ns)	
8ns	1, 0, 1	(1, 8 ns)	
10ns	1, 1, 1	(1, 10ns) (1, 15ns) (0, 18ns)	p1 körs p.g.a att *b* ändras
15ns	1, 1, 1	(1, 15ns) (0, 18ns)	
18ns	1, 1, 0		

Vid t=0 antar vi att:
a = ’1’; *b* = ’0’; *x* = ’0’
vid t=10 blir sätts *b*=’1’

```
p1: process(a, b)
begin
  x <= a and b after 5ns,
  not b after 8 ns;
end process p1;
```



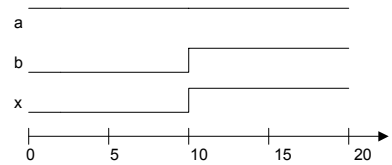
Schemaläggning av signaler

- Exempel #2, ”inga fördröjningar anges i koden”

Tid	a, b, x	Drivare (av utgången *x*)	Kommentarer
0ns	1, 0, 0	(0, 0ns) (0, 0ns+Δ)	p2 körs p.g.a initiering
0ns +Δ	1, 0, 0	(0, 0ns+Δ)	
10ns	1, 1, 0	(1,10 ns + Δ)	p2 körs p.g.a att *b* ändras
10ns+Δ	1, 1, 1	(1,10 ns + Δ)	

Vid t=0 antar vi att:
a = ’1’; *b* = ’0’; *x* = ’0’
vid t=10 blir sätts *b*=’1’

```
p2: process(a, b)
begin
  x <= a and b;
end process p2;
```



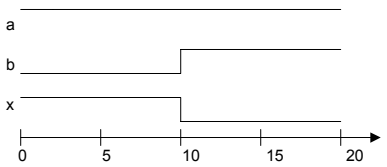
Schemaläggning av signaler

- Exempel #3, ”två parallella tilldelningar av signal”

Tid	a, b, x	Drivare (av utgången *x*)	Kommentarer
0ns	1, 0, 0	(0, 0ns) (1, 0ns+Δ)	p3 körs p.g.a initiering
0ns +Δ	1, 0, 1	(1, 0ns+Δ)	
10ns	1, 1, 1	(1,10ns)~~(1,10ns+Δ)~~(0,10ns+ Δ)	p3 körs p.g.a att *b* ändras
10ns+Δ	1, 1, 0	(0,10 ns + Δ)	

Vid t=0 antar vi att:
a = ’1’; *b* = ’0’; *x* = ’0’
vid t=10 blir sätts *b*=’1’

```
p3: process(a, b)
begin
  x <= '1';
  if b = '1' then
    x <= '0';
  end if;
end process p3;
```



Ex: VHDL kod för 3-ing. AND

- Exempel #4, ”en 3-ingångars AND-grind”
  - x = a(2)• a(1)• a(0)

```
and3: process(a)
begin
  x <= '1';
  for i in 2 downto 0 loop
    x <= a(i) and x;
  end loop;
end process p3;
```

Det nya värdet för *x* schemaläggs till en tidpunkt efter for-loopen har körts

X är alltid ’0’ i for-loopen
X <= a(i) and ’0’ → x = ’0’

Tid	a_2,a_1,a_0,x	Drivare (av utgången *x*)	Kommentarer
0ns	1,1,1,0	(0,0ns) ~~(1,0ns+Δ)~~ ~~(0,0ns+ Δ)~~ ~~(0,0ns+ Δ)~~ (0,0ns+ Δ)	and3 körs p.g.a initiering
0ns +Δ	1,1,1,0	(0, 0ns+Δ)	

Signaltilledelningar ger upphov till parallella händelser
Ordningen på signaltilldelningar i koden spelar ingen roll

Det fungerar inte!!

Ex: VHDL kod för 3-ing. AND

- ”en 3-ingångars AND” – använd variabler

```
and3: process(a)
  variable temp: bit;
begin
  temp := '1';
  for i in 2 downto 0 loop
    tmp := a(i) and temp;
  end loop;
end process p3;
```

Variabeltilldelning sker direkt utan att schemaläggas i framtiden

Genom att använda variabler blir koden sekventiell där ordningen på satserna spelar roll

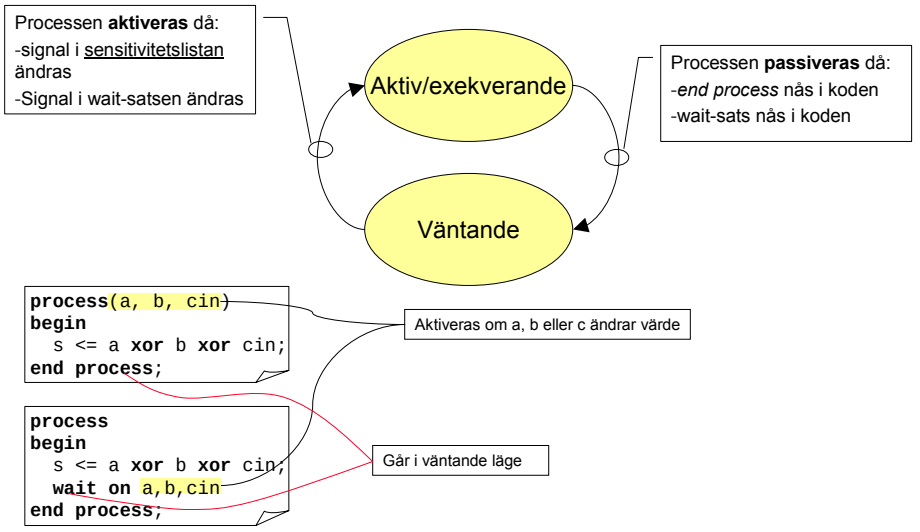
Variabler kan endast användas i processer

Processer

- En process är ett sekventiellt program
- Flera processer i en arkitektur exekveras parallellt
- Kommunikation mellan processer sker med signaler
- Syntax:

```
[<process_namn>:] process [(sensitivitetslista>)]
  [<deklarationer i processen>]
begin
  <sekventiella satser>
end process [<process namn>];
```

Aktivering av processer



Sekventiella satser

- Sekventiella satser finns endast inne i processer
- För sekventiella satser gäller
  - Att deras inbördes ordning i koden spelar roll
  - Att de exekveras i noll simuleringstid
  - Att de har absolut inget att göra med sekventiell logik eller tillståndsmaskiner
- De sekventiella satserna är
  - IF-THEN-ELSE
  - CASE-WHEN

IF-THEN-ELSE

- Exekverar första sektionen med satser som följer ett sant villkor
- Ingen annan sektion exekveras

```
label:                -- ej nödvändigt
if villkor1 then
  sats1;
elsif villkor2 then -- ej nödvändig sektion
  sats2;
else                  -- ej nödvändig sektion
  sats3;
end if;
```


Logiskt uttryck med IF-THEN …

- En serie villkor bildar ett logiskt uttryck

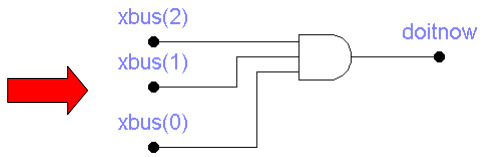
```
process (x, y, z, a, b)
begin
  if x = '1' then
    f <= a;
  elsif y = '1' then
    f <= b;
  elsif z = '1' then
    f <= c;
  else
    f <= d;
  end if;
end process;
```

f = xa + x'yb + x'y'zc + x'y'z'd

Logiskt uttryck med IF-THEN …

- Exempel:

```
process (xbus)
begin
  if xbus = "111"
then
    doitnow <= '1';
  else
    doitnow <= '0';
  end if;
end process;
```

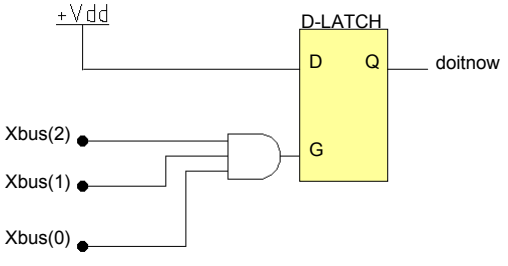


Felaktig användning av IF-THEN …

- Exempel:

```
process (xbus)
begin
  if xbus = "111" then
    doitnow <= '1';       -- en latch kommer att föras in
  end if;
end process;
```

Det finns inget i koden som säger att Utgången ska gå till '0'



CASE-WHEN satsen

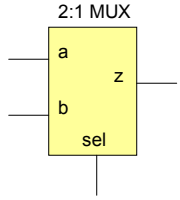
- Exekverar endast en sektion som följs av ett giltigt val eller det som följer efter *OTHERS*
- Valen måste vara ömsesidigt uteslutande
- Samtliga möjliga val måste anges
- Resulterar i en multiplexer-baserad struktur

```
label:
case selector_expression is
  when val1 => sats1;
  when val2 => sats2;
  when val3 => sats3;
  when OTHERS  => sats4;
end case;
```


CASE-WHEN – exempel

- 2:1 Multiplexer

```
case sel is
  when '0' => z <= a;
  when '1' => z <= b;
end case;
```

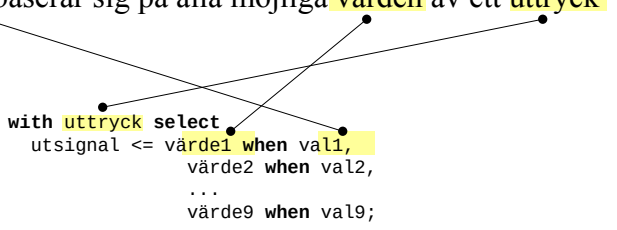


Parallella satser

- Parallella satser finns på arkitekturnivå
- För parallella satser gäller
  - Att deras inbördes ordning i koden spelar ingen roll
  - Att resultatet av den schemaläggs som en händelse framtida simuleringstid
- De parallella satserna är
  - With-select-when
  - When-else

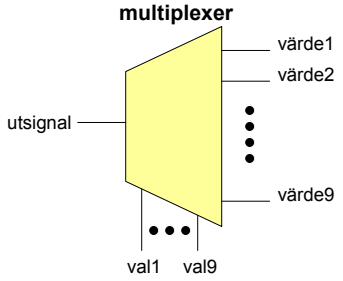
WITH-SELECT-WHEN

- Kallas för ”selected signal assignment”
- Parallella satser – alltså gäller det signaler
- Väljer en av flera värden som ska driva utsignalen
  - Valet baserar sig på alla möjliga värden av ett uttryck



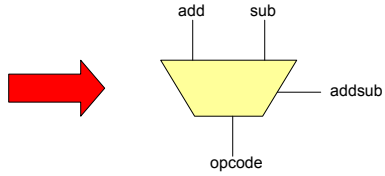
Resultat från en *with-select-when*

```
with uttryck select
  utsignal <= värde1 when val1,
              värde2 when val2,
              ...
              värde9 when val9;
```



Exempel på *with-select-when*

```
-- 2-till-1 multiplexer
with addsub select
  opcode <= add when '0',
            sub when '1';
```




```
with (a and b) select
  out <= "1011" when "00"
         "11--" when "1Z",
         x OR y when "11",
         "0000" when others;
```

Logiska uttryck tillåtna

others blir valet då ingen av de Ovanstående är sanna

```
with min_integer select
  outvec <= X"1F" when 35,
            X"27" when 2 TO 5,
            X"FF" when OTHERS;
```

Ett område av värden kan anges

When-else

- Kallas också för ”conditional signal assignment”
- Parallella satser – alltså gäller det signaler
- Väljer en av flera värden för att driva utsignalen
  - Valet baserar sig på det första villkoret som blir sant

```
utsignal <= värde1 when villkor1 else
            värde2 when villkor2 else
            ...
            värde9;
```


Exempel på *when-else*

```
opcode <= add when (addsub = '0') else
          sub when (addsub = '1') else
          nop;
```

```
out <= "1011" when (a = '0') else
       "11--" when (b = '0') else
       x OR y when (a AND b) = '1' else
       "0000";
```

```
-- 4-till-2 prioritetsavkodare
outcode <= "11" when in3 = '1' else
           "10" when in2 = '1' else
           "01" when in1 = '1' else
           "00" when in0 = '1' else
           "00";
```

Slut på föreläsning 3

Innehåll

- Parallellitet i VHDL
- Fördröjningar
- Simuleringscykeln
- Aktivering av processer
- Parallella och sekventiella satser