F4: Kombinatorisk logik i VHDL

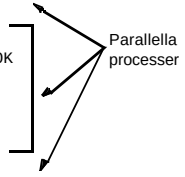
- **Innehåll:**
  - **Kombinatorisk logik**
    - **Parallella satser**
    - **Sekventiella satser**
  - **Operatorer för relationer**
  - **Instantiering av komponenter**

Kombinatorisk och sekventiell logik

- **Kombinatorisk logik**
  - **Utgångarna är funktion av värdena som för ögonblicket ligger på ingångarna**
  - **Inget minne**
  - **Exempel: grindar, multiplexer, avkodare, ALU**
- **Sekventiell logik**
  - **Utgångarna är funktion av nuvarande och tidigare värden på ingångarna**
  - **Använder minne (vippor, RAM)**
  - **Exempel: tillståndsmaskiner, räknare, skiftregister**
- **Båda implementeras med parallella och/eller sekventiella VHDL satser**
- **Parallella satser inne i *architecture*, utanför processer**
  - **dataflöde och strukturell stil**
- **Sekventiella satser inne i processer**
  - **Beteende stil**

Parallella och sekventiella satser i VHDL

```
ENTITY ename IS
  Ports( a, b, c: IN bit;
         y, z, w: OUT bit;
  Declarations   -- no variables
END ename

ARCHITECTURE first OF ename IS
  Declarations   -- no variables, but signals are OK

BEGIN
  y <= a AND b;

  PROCESS (a,b,c)
    Declarations   -- no signals, but variables are OK
    VARIABLE v: bit;
  BEGIN
    v := (a OR b) AND c;
    w <= a XOR v;
  END PROCESS;

  z <= c XOR b;

END first;
```



Kombinatorisk logik - parallella satser

- **Parallella satser**
  - **Booleska ekvationer**
  - **Logiska- och relationsoperatorer**
  - ***With-Select-When*, *When-Else***
  - **Komponentinstantiering**
- **Booleska ekvationer**
  - **Används i både parallella och sekventiella signaltilldelningar (<=)**
  - **Används enbart i sekventiella variabel tilldelningar (:=)**
  - **Ofta bra för enkla till komplexa skalära operationer**
  - **Exempel:**

```
asignal <= bsig OR csig;
avariable := bvar AND cvar;
```

Logiska operatorer

- **logiska operatorer som används i booleska ekvationer**
  - **AND, OR, XOR, NAND, NOR, XNOR, NOT**
  - **För typerna: bit, boolean, std_logic, std_ulogic, och i 1-dimensionella arrayer**
  - **Operatorn NOT har högre prioritet (*precendence*); övriga har lika lägre prioritet**
  - **Paranteser behövs vanligtvis för logiska uttryck med många nivåer**
  - **Exempel:**

```
z <= a AND b AND c OR d NAND e OR NOT f;
```

är ekvivalent med

```
z <= ((((a AND b) AND c) OR d) NAND e) OR (NOT f);
```


With-Select-When sats

- **Kallas för *"selected signal assignment"***
- **parallella satser, alltså gäller det signaler**
- **liknar den sekventiella *CASE*-satsen**
- **Väljer en av flera värden som ska driva utsignalen**
  - **valet baseras på alla möjliga värden av ett uttryck (*selector_expression*)**
- **Syntax:**

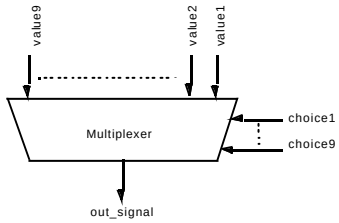
```
WITH selector_expression SELECT
  out_signal <= value1 WHEN choice1,
                value2 WHEN choice2,
                ...
                value9 WHEN choice9;
```


With-Select-When sats

- ***selector_expression (väljare)***
  - **signalnamn eller uttryck med signalnamn**
- ***choices (val)***
  - **typen måste matcha *selector_expression***
  - **ett set med val specificeras som måste vara ömsesidigt uteslutna och inkludera alla**
  - **tal, strängar, uttryck, OTHERS**
- ***Values* (värden)**
  - **alla normalt gilitga värden på höger sida i en signaltilldelningssats**
- **Syntesresultat**
  - **satsen syntetiseras till en N-bit M-till-1 multiplexer**
    - ***values* är dataingångar**
    - ***selector* och *choices* bildar ingångskoder för styrsignalen**
    - ***out_signal* är datautgången**

Syntesresultat (*with-select-when sats*)

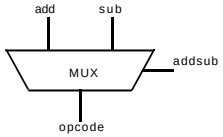
```
WITH selector_expression SELECT
  out_signal <= value1 WHEN choice1,
                value2 WHEN choice2,
                ...
                value9 WHEN choice9;
```



With-Select-When sats

- **Exempel**

```
-- 2-to-1 multiplexer
WITH addsub SELECT
  opcode <= add WHEN '0',
            sub WHEN '1';
```




```
WITH (a AND b) SELECT
  out <= "1011" WHEN "00" | "01",
         "11--" WHEN "1Z",
         x OR y WHEN "11",
         "0000" WHEN OTHERS;
```

```
WITH myint SELECT
  outvec <= X"1F" WHEN 35,
            X"27" WHEN 2 TO 5,
            X"FF" WHEN OTHERS;
```


When-Else sats

- **Kallas också för *"conditional signal assignment"***
- **Parallella satser, alltså gäller det signaler**
- **liknar den sekventiella IF-THEN-ELSE satsen**
- **väljer en av flera värden för att driva en utgångssignal**
  - **valet baserar sig på det första villkoret som blir sant (TRUE)**
- **Syntax:**

```
out_signal <= value1 WHEN condition1 ELSE
              value2 WHEN condition2 ELSE
              ...
              value9;
```


When-Else sats

- **Villkor (*conditions*)**
  - **booleskt uttryck**
  - **ett set med villkor som inte nödvändigtvis är ömsesidigt uteslutna, inget krav att alla ska vara inkluderade**
  - **första sanna villkoret bestämmer värdet som ska tilldelas signalen**
  - **sista värdet tilldelas om alla villkor är falska**
- **Syntesresultat**
  - **om villkoren är ömsesidigt uteslutna så blir resultatet en multiplexer, samma som för *WITH-SELECT-WHEN* satsen**
  - **I annat fall blir resultatet en mer komplex prioritets avkodare där det första villkoret har högsta prioritet**

When-Else sats

- **Exempel**

```
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-to-2 priority encoder
outcode <= "11" WHEN in3 = '1' ELSE
           "10" WHEN in2 = '1' ELSE
           "01" WHEN in1 = '1' ELSE
           "00" WHEN in0 = '1' ELSE
             "00";
```

Kombinera bitar in till en vektor

- **gruppera**
  - **en lista av typ-kompabilitet kan tilldelas till en vektor**
  - **kan finnas på höger sida i en signaltilldelningssats**
  - **Exempel:** `xvector <= (a, b, c, d);`
- **sammanfogning**
  - **sammanfoga kompatibla bitar till en vektor**
  - **kan finnas på höger sida i en signaltilldelningssats**
  - **kan användas i ett WHEN-villkor eller i ett IF-villkor**
  - **Exempel:** `xvector <= (a & b & c & d);`

"*don't cares*" i villkor

- **Villkor med *don't care* värden '-' ska inte användas i IF och WHEN-ELSE satser**
  - **De jämförs bokstavligen med '-' i simuleringen**
  - **De tolkas alltid som FALSE vid syntes**
  - **Exempel:** `IF a = "0--" THEN ...`
- **men don't cares kan ändå användas med sdt_match funktionen**
  - ***std_match* finns i *numeric_std* och *std_arith packages***
  - **Format:** `std_match (name, bitstring);`
  - **returnerar TRUE för antingen 0 eller 1 in en position med '-'**
  - **Exempel:** `IF std_match(a, "0--") THEN ...`

operatorer för relationer i villkor

- **testar likhet och olikhet: (= och /=)**
- **testar relativ ordning och storlek: (<, <=, >, >=)**
- **returnerar booleska värden TRUE eller FALSE**
- **finns definierad för samtliga typer, även för användar-definierade**
- **De två operandernas typer måste matcha, exempel:**

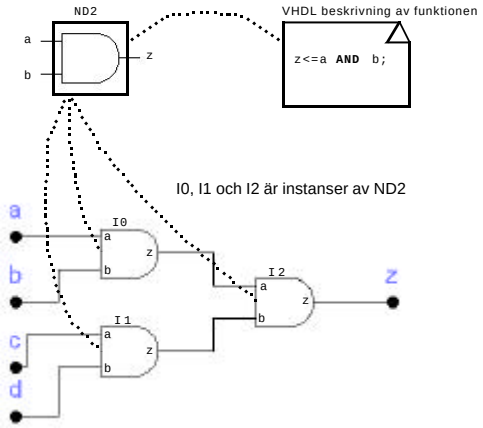
```
IF a_std_logic_vector = 125 THEN ...     FEL!
IF a_bit = "101" THEN ...                FEL!
IF a_integer = X"2F" THEN ...            FEL!


IF a_std_logic_vector = X"2F" ...
IF a_bit = '1' THEN ...
IF a_integer = 125 THEN ...
```


operatorer för relationer i villkor

- **genom "overloading" av operatorer kan operander av olika typer jämföras**
- **Numeric_std package (IEEE 1076.3) tillåter**
  - **jämförelse av signed med integer och unsigned med integer**
- **Std_arith package tillåter**
  - **jämförelse av std_logic_vector (som unsigned) med integer**
  - **för detta krävs:**

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE work.std_arith.ALL;
```

Instantiering av komponenter



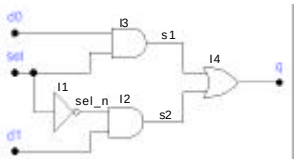
Instantiering av komponenter

- **parallella satser som kan användas för att implementera kombinatorisk logik**
- **representerar sammankopplingen av andra *entities* som kallas *components***
- **man måste specificera en eller flera *packages* före arktiekturen som innehåller de komponenter som man vill koppla ihop (t.ex.** `USE work.mypkg.ALL;`**)**
- **Syntax:**

```
instance_label: component_name PORT MAP
  (ordered_list_of_signals);
```

- **Exempel:**

```
USE work.some_library.ALL;
ARCHITECTURE struct_mux OF mux IS
SIGNAL s1, s2, sel_n: std_logic;
BEGIN
  I1: inv PORT MAP(sel, sel_n);
  I2: and PORT MAP(sel_n, d1, s2);
  I3: and PORT MAP(d0, sel, s1);
  I4: or  PORT MAP(s1, s2, q);
END;
```



Kombinatorisk logik - sekventiella satser

- **Beteende stil inne i en process eller subprogram (funktion eller procedur)**
- **en process är en parallell sats**
- **den innehåller sekventiella satser**
  - **satsernas inbördes ordning spelar roll**
  - **exekveras i noll simuleringstid**
  - **har absolut inget att göra med sekventiell logik eller FSM**
- **sekventiella satser**
  - **booleska ekvationer (tilldelningssatser till variabler)**
  - **IF-THEN-ELSE**
  - **CASE-WHEN**
- **Temporära variabler kan användas inne i processer**
  - **men signaler används alltid för alla processens in- och utgångar**

IF-THEN-ELSE satsen

- **Logiskt sett ekvivalent med den parallella satsen för signaltilldelning (*WHEN-ELSE*), men mer användbar**
- **Syntax:**

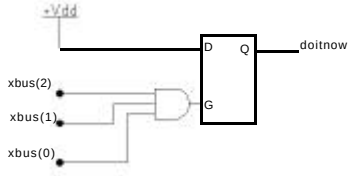
```
label:                              -- ej nödvändigt
IF condition1 THEN
  statement1;
ELSIF condition2 THEN               -- ej nödvändig sektion
  statement2;
ELSE                                -- ej nödvändig sektion
  statement3;
END IF;
```

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

IF-THEN-ELSE satsen

- **VAR FÖRSIKTIG - en icke önskvärd latch införs om en signal inte tilldelas ett värde varje gång processen körs.**
- **Exempel: Önskad funktio är** `doitnow = xbus(2) AND xbus(1) AND xbus(0)`

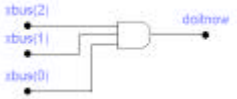
```
PROCESS (xbus)
BEGIN
  IF xbus = "111" THEN
    doitnow <= '1';       -- en latch kommer att föras in
  END IF;
END PROCESS;
```



IF-THEN-ELSE satsen

- **Exempel:**

```
PROCESS (xbus)
BEGIN
  IF xbus = "111" THEN
    doitnow <= '1';
  ELSE
    doitnow <= '0';
  END IF;
```



IF-THEN-ELSE satsen

- **En serie av villkor som bildar en prioritet avkodare**

```
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;
```

- **Resultat**
  - `f = xa + x'yb + x'y'zc + x'y'z'd`

CASE-WHEN satsen

- **logiskt sett är den ekvivalent med *selected signal assignment* (*WITH-SELECT-WHEN*) men mer användbar**
- **Syntax:**

```
label:
CASE selector_expression IS
  WHEN choice1 => statement1;
  WHEN choice2 => statement2;
  WHEN choice3 => statement3;
  WHEN OTHERS  => statement4;
END CASE;
```

- **Exekverar endast en sektion som följs av ett giltigt val (*choice*) eller det som följer efter OTHERS**
- **Valen måste vara ömsesidigt uteslutande och inkludera alla**
- **Ger en multiplexer-baserad struktur**

Aritmetiska operatorer

- **Standard aritmetiska operatorer**

Add	+
Subtract	-
Multiply	*
Divide	/
Mod	**MOD** (endast integers)
Remainder	**REM** (endast integers)
Exponential	**
Absolutvärde	**ABS**
Sammanfogning	&
Tecken	+ eller -