INF3430

VHDL byggeblokker og testbenker

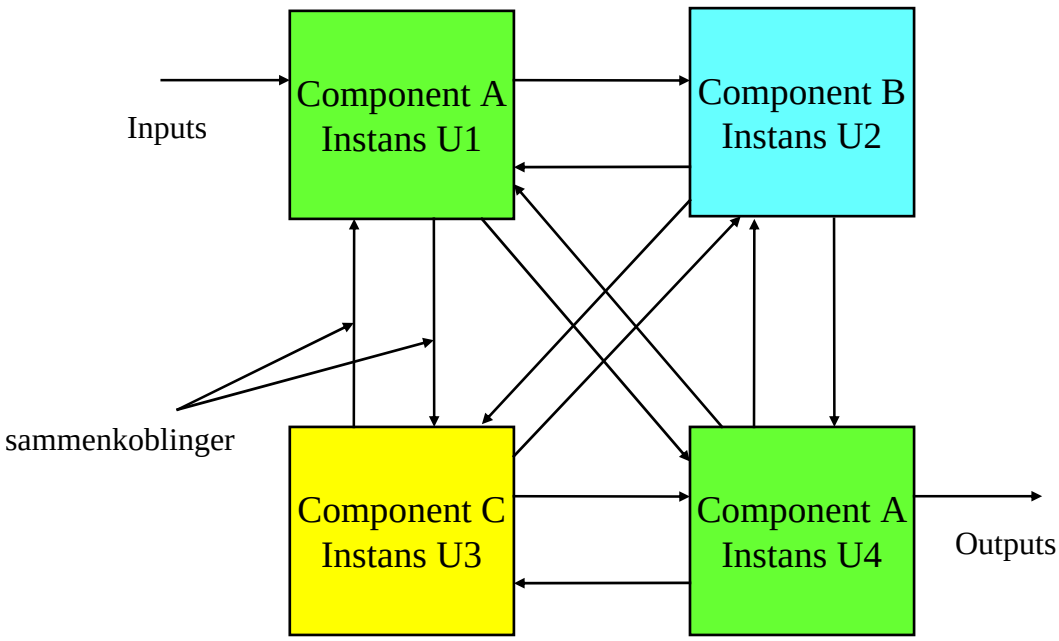
Innhold

- Entity/architecture
- Strukturelle design (nettliste)
- Generics
- Configurations
- Operatorer-Operator prioritet (precedence)
- Datatyper
  - Bit / IEEE1164 std_ulogic /std_logic
- Sekvensiell VHDL (Process)
- Testbenk for kombinatorisk logikk

Entity/architecture

- Entity og architecture er de to mest grunnleggende byggeklossene i VHDL.
- Entity
  - Tilkobling mot omverdenen i
  - Portbeskrivelsen
    - Input/output/bi-directional signaler
- Architecture
  - Beskriver virkemåte
  - En entity kan ha mange architectures
  - Kan benyttes til å beskrive kretsen for flere abstraksjonsnivåer
    - Behavorial (algoritmer)
    - RTL (Register Transfer Level)
    - Post synthesis (nettliste)
    - Post Place & Route (nettliste + timing)

Strukturelle design (netlist)



- Hver Component instans har et underliggende Entity/architecture par
- Vi kan lett gjenbruke “entities”
- Vi kan lage et hierarkisk design med så mange undernivåer vi måtte ønske
- Ikke overdriv
  - Å benytte flere prosesser innenfor en arkitektur er et godt alternativ til et strukturelt design

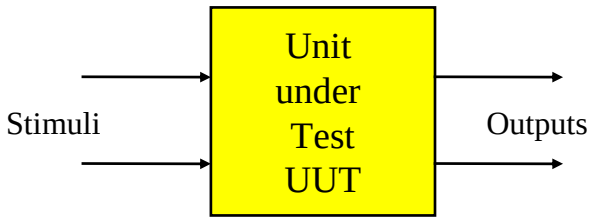
Strukturelle design (netlist)

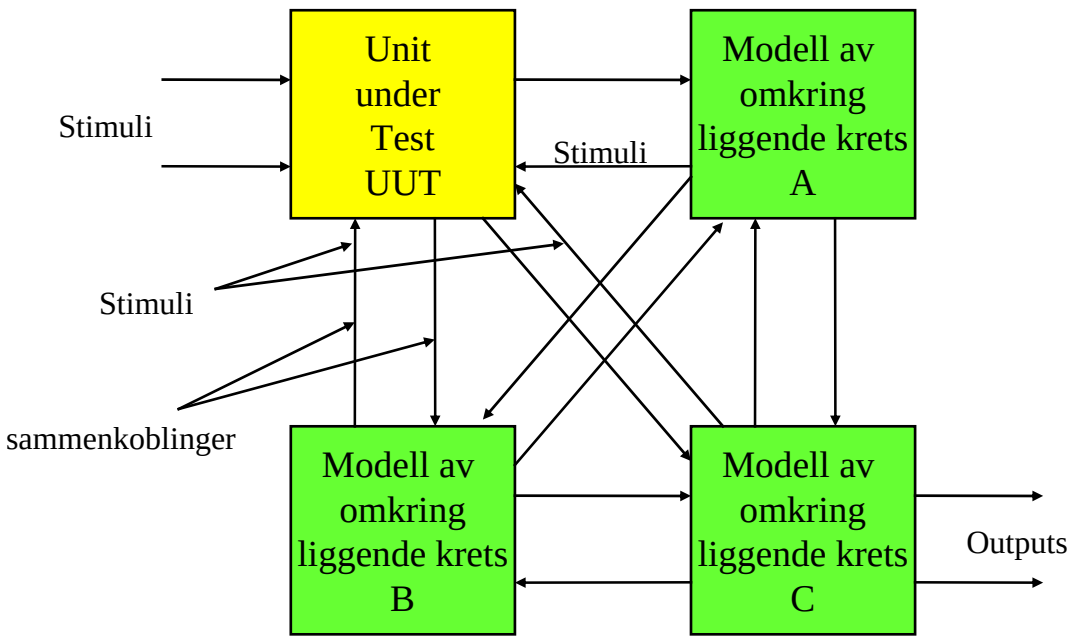
- Gjenbruk av moduler (entities og architectures)
- Generiske moduler(generics)
  - F.eks. skalèrbare bussbredder
  - Konfigurerbar funksjonalitet
- Bryter opp store design til mindre og håndterlige byggeklosser
  - Tenk funksjonelle blokker
  - Sammenkobling av funksjonelle blokker (entities/components/modules)
- Lettere samarbeid mellom flere i et designteam
  - Viktig med veldefinerte grensesnitt mellom moduler
- Ethvert entity/architecture-par kan benyttes som en byggekloss i en strukturell beskrivelse
  - Sammenkobling av komponenter

Testbenk

- Lager stimuli/testomgivelser i VHDL
- Meget kraftige simuleringsmuligheter
  - Fil I/O
    - Leser testmønstre fra fil
    - Skriver resultater til fil og sammenligner med fasitfil.
    - Fasitfil kan også leses inn og sammenligning mellom resultat og fasit kan utføres innenfor testbenken.
  - Kan bygge inn modeller for omkringliggende kretser
    - Særlig viktig dersom man har toveis kommunikasjon mellom UUT (Unit under test) og omkringliggende kretser. Jfr. Handshake signaler.
- Gir simulatoruavhengig test

Testbenk


Unit
under
Test
UUT
Stimuli
Outputs


Unit
under
Test
UUT
Modell av
omkring
liggende krets
A
Stimuli
Stimuli
Stimuli
sammenkoblinger
Modell av
omkring
liggende krets
B
Modell av
omkring
liggende krets
C
Outputs

Strukturelle design (netlist)

```
25: architecture netlist2 of comb_function is
26:
27:   component And2 is
28:     port (x, y : in BIT; z: out BIT);
29:   end component And2;
30:
31:   component Or2 is
32:     port (x, y : in BIT; z: out BIT);
33:   end component Or2;
34:
35:   component Not1 is
36:     port (x : in BIT; z: out BIT);
37:   end component Not1;
38:
39:   signal p, q, r : BIT;
40:
41: begin
42:   g1: Not1 port map (a, p);
43:   g2: And2 port map (p, b, q);
44:   g3: And2 port map (a, c, r);
45:   g4: Or2 port map (q, r, z);
46: end architecture netlist2;
```

- Her hentes entitetene fra ”working” library
- Sist kompilerte arkitektur benyttes (default binding)

Strukturelle design (netlist)

```
24:
25: architecture netlist of comb_function is
26:
27:   signal p, q, r : BIT;
28:
29: begin
30:   g1: entity WORK.Not1(ex1) port map (a, p);
31:   g2: entity WORK.And2(ex1) port map (p, b, q);
32:   g3: entity WORK.And2(ex1) port map (a, c, r);
33:   g4: entity WORK.Or2(ex1) port map (q, r, z);
34: end architecture netlist;
```

- Her spesifiseres eksplisitt hvilke bibliotek entitetene hentes fra.
- Det spesifiseres også hvilke arkitektur som benyttes
- Det benyttes en kombinasjon av eksterne (via entiteten) og interne signaler for å lage sammenkoblinger

Generics

- I tillegg til portbeskrivelsen kan en entity ha en generic beskrivelse
- Generics benyttes for å kunne lage parametriserbare komponenter (generiske)
- Generics benyttes oftest til å overføre timing informasjon eller strukturell informasjon
  - Eksempel 1:
    - Tidsforsinkelser kan variere fra krets til krets, mens virkemåten ellers er lik
  - Eksempel 2:
    - Antall bit kan variere fra krets til krets mens virkemåten ellers er lik.

Generics

```
24: entity And2 is
25:   generic (delay : DELAY_LENGTH := 10 ns);
26:   port (x, y : in BIT; z: out BIT);
27: end entity And2;
28:
29: architecture ex2 of And2 is
30: begin
31:   z <= x and y after delay;
32: end architecture ex2;
```

- Delay_length er en subtype av typen time
  - Ikke VHDL 1987 revisjon

Configurations

- Configurations er en mekanisme for å velge ut en gitt arkitektur til en entitet (component)
- Nyttig når vi skal simulere RTL, postsyntese og post place and route
- Benyttes til å binde sammen hierarki
- Uten bruk av configurations må man passe på hvilke arkitektur som er sist kompilert og dette kan bli komplisert etter hvert som designet vokser.

Configurations

- Antar at post syntese modellen er kompilert til biblioteket POSTSYNTH og
- Post place and route modellen er kompilert til biblioteket POSTROUTE

```
65: --Configuration for å simulere RTL designet
66: configuration CFG_FIRST_RTL of TEST_FIRST is
67:     for TESTBENCH
68:       for UUT : FIRST use entity WORK.FIRST(MY_FIRST_ARCH);
69:       end for;
70:     end for;
71: end CFG_FIRST_RTL;
72:
73:
74: --Configuration for å simulere etter syntese
75: --Vi må gjøre POSTSYNTH biblioteket synlig
76: library POSTSYNTH;
77: configuration CFG_FIRST_SYNTH of TEST_FIRST is
78:     for TESTBENCH
79:       for UUT : FIRST use entity POSTSYNTH.FIRST(Postsynth);
80:       end for;
81:     end for;
82: end CFG_FIRST_SYNTH;
83:
84: --Configuration for å simulere etter place and route
85: --Vi må gjøre POSTROUTE biblioteket synlig
86: library POSTROUTE;
87: configuration CFG_FIRST_POSTROUTE of TEST_FIRST is
88:     for TESTBENCH
89:       for all : FIRST use entity POSTROUTE.FIRST(Postroute);
90:       end for;
91:     end for;
92: end CFG FIRST POSTROUTE;
```

Sekvensiell VHDL (Processer)

- En process har en sensitivitetsliste
  - Processen evalueres når et av signalene på sensitivitetslisten endrer verdi
- I en process dannes èn *signaldriver* pr.signal.
  - Detter betyr at det er ikke noen konflikt med flere tilordninger pr. signal innenfor èn process.
- Oppdatering av et signal skjer når processen terminerer.

- Oppgave:
  Anta B=0, A skifter fra 0->1. Hva skjer med F og hvorfor?

```
112: process (A,B)
113: begin
114:    if A = '1' then
115:      B <= '1';
116:    else
117:      B <= '0';
118:    end if;
119:
120:    if B = '1' then
121:      F <= '1';
122:    else
123:      F <= '0';
124:    end if;
125: end process;
```

Sekvensiell VHDL (Processer)

- For å beskrive logikk ved bruk av processer er det vanlig å benytte
  - If – tester
  - Case – tester
- If og case benyttes som i andre programmeringsspråk f.eks. C
  - I if tester kan man teste på forskjellige signaler/variabler og
    - if tester har innebygd prioriet
  - I case setninger tester man kun på et signal (med en eller flere bit)
    - Ingen innebygd prioritet fordi samme signal benyttes overalt i testen

Operatorer

- Aritmetikk

  +, -, *, /
- Sammenligning

  >, >=, <, <=, =, /=
- Logiske operatorer

  and, nand, or, nor, not, xor, xnor

Operator prioriet (precedence)

- I VHDL er det kun not-operatoren som blir evaluert før andre
- Alle andre operatorer blir evaluert på lik linje
- Uttrykkene blir evaluert ettersom de er skrevet (venstre mot høyre)
- For å tvinge fram at et uttrykk evalueres før et annet benyttes paranteser
- Eksempler:

```
a <= a or b and c = (a or b) and c
z <= a and not b and c = a and (not b) and c
     = c and (a and (not b))
```

Datatyper

- Meget streng datatypesjekking i VHDL
- Scalare typer
  - Kan holde bare en verdi om gangen
  - Eksempel:
    - Integer, Real, Enumerated (bit), Physical (time)
- Composite typer
  - Kan holde mer enn en verdi om gangen. Dette er tilfelle for array og record typer
  - Eksempel:
    - Bit_vector

- Typen bit kan ha verdiene
  - 0,1
  - Dette er veldig begrensende i forhold til å modellere digital hardware
  - Dette medførte en ny standard, IEEE 1164 std_logic

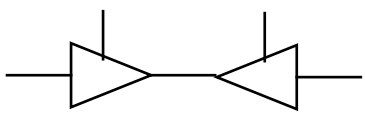
Datatypen std_logic IEEE 1164

```
TYPE std_ulogic IS
(
        '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
);
```

```
 SUBTYPE std_logic IS resolved std_ulogic;
```

std_ulogic/std_logic

- std_ulogic
  - u – Unresolved
  - Drivere kan ikke kobles sammen
- std_logic
  - Resolved
    - Verdier definert gjennom en såkalt “resolution table”
  - Drivere kan kobles sammen og kan modellere:
    - Wired logic
    - Tri-state drivere og busser
    - Pull up/pull down motstander



```
 CONSTANT resolution_table : stdlogic_table := (
-------------------------------------------------------------
|  U    X    0    1    Z    W    L    H    -        |   |
-------------------------------------------------------------

( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
);
```

Tri-state buffer

```
A <= B when EN = '1' else 'Z';

-- ekvivalent med
TRISTATE:
process (B,EN)
begin
  if EN = '1' then
    A <= B;
  else
    A <= 'Z';
  end if;
end process;
```