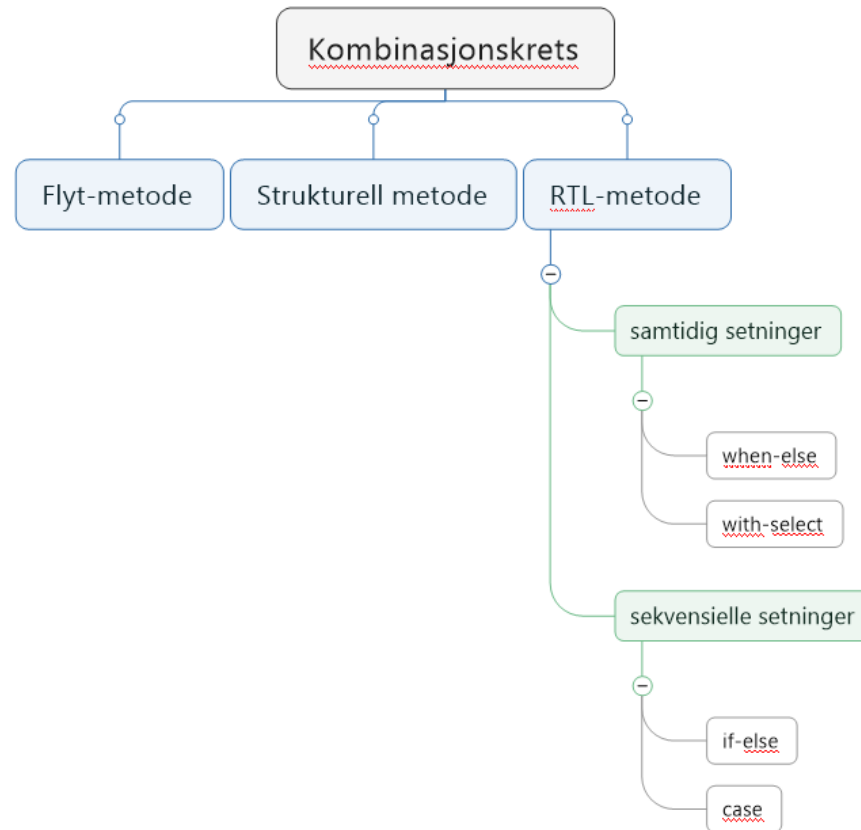


Kapittel 04: Modelling av kombinasjonskretser ved bruk av process

Hieu Nguyen

Oversikt om å Modellere Kombinasjonskretser



Innledning

- VHDL kan inneholde en rekke sekvensielle setninger
- Forskjellen mellom en samtidig setning og en sekvensiell setning er at alle sekvensielle setninger blir **innkapslet innenfor «process»**
- **En process** selve er en samtidig setning
- Sekvensielle setninger har mange konstruksjoner, men mange av dem har ikke klar tilsvarende **maskinvare**
- Vi begrenser oss med bruk av prosess for to oppgaver til å
 - Beskrive **ruting-struktur** med **if** og **case** setninger
 - Konstruere mal for **minneelementer/lagringselementer**

Syntaks for Process

- **Sensitivity_list**: en liste av signaler som prosess responderer til
- For kombinasjonskreter, må **alle** inngangssignaler inkluderes i sensitivity-listen
- Kroppen av en prosess består **ubegrenset** antall sekvensielle setninger

```
process (sensitivity_list)
```

```
begin
```

```
    sequential statement_1;
```

```
    sequential statement_2;
```

```
    sequential statement_3;
```

```
    .....
```

```
    sequential statement_n;
```

```
end process;
```

Syntaks for Process

```
process (sensitivity_list)
```

```
begin
```

```
    sequential statement_1;
```

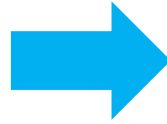
```
    sequential statement_2;
```

```
    sequential statement_3;
```

```
    .....
```

```
    sequential statement_n;
```

```
end process;
```



$$Eq = i0 \cdot i1 + i0' \cdot i1'$$



```
38 architecture Behavioral of one_bit_compare_process is
39 signal p0,p1 : std_logic;
40 begin
41
42     process (i0,i1)
43     begin
44         p0 <= i0 and i1;
45         p1 <= (not i0) and (not i1);
46         eq <= p0 or p1;
47     end process;
48
49 end Behavioral;
```

Syntaks for Process

Ubegrenset antall process innenfor architecture

Hver process betrakes som en samtidig setning


Samtidig setninger og process kam ligge sammen innenfor architecture

```
38 architecture Behavioral of one_bit_compare_process is
39 begin
40
41     process ()
42     begin
43         ----statement here
44     end process;
45
46     process ()
47     begin
48         ----statement here
49     end process;
50
51     process ()
52     begin
53         ----statement here
54     end process;
55
56 end Behavioral;
```


Signal-Tilordningssetning

- Syntaks

```
signal_name    <= value_expression;
```



- Eksempel



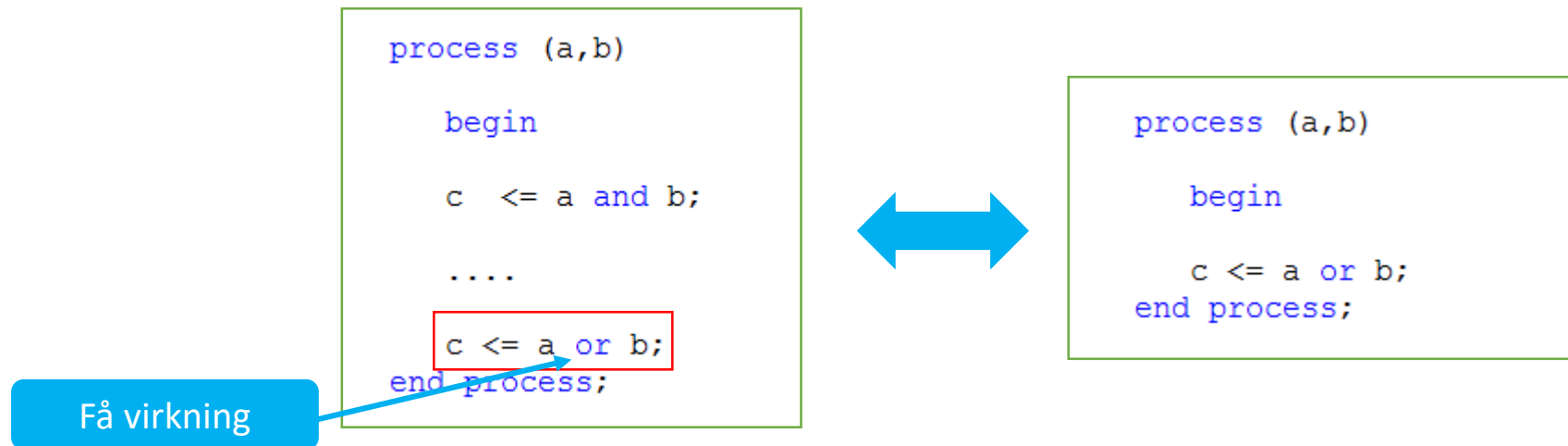
```
z    <= a xor b;
```

```
process (sensitivity_list)
begin
    sequential statement_1;
    sequential statement_2;
    sequential statement_3;
    .....
    sequential statement_n;
end process;
```

- Merk: den sekvensiell setningen må **innkapsles** innenfor **process**

Signal-Tilordningssetning

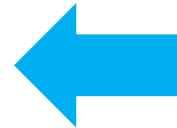
- Når et signal blir tilordnet **mange ganger** innenfor en prosess, har den **siste** tilordningen virkning
- Eksempel



Signal-Tilordningssetning

- Vi kan **ikke** ha flere tilordninger til **et signal utenfor process**
- Eksempel

```
----- utenfor process-----|  
  
c  <= a or b;  
  
c  <= a and b;
```



Feil

If-Else Setning med Prosess

- Syntaks
- Boolske uttrykk blir evaluert sekvensielt til et uttrykk er sant, ellers **else-gren** blir oppnådd.
- Setninger svarer til **logiske-sann** gren blir utført
- Sekvensiell setning (**if-else**) og (**when-else**) er tilsvarende

```
process (sensitivity_list)
begin
  if boolean_expr_1 then
    sequential_statements;
  elsif boolean_expr_2 then
    sequential_statements;
  elsif boolean_expr_2 then
    sequential_statements;
  else
    sequential_statements
  end if;
end process;
```

If-Else Setning med Proses

- Eksempel 1

```
process (sensitivity_list)
```

```
begin
```

```
if boolean_expr_1 then
```

```
sequential_statements;
```

```
elsif boolean_expr_2 then
```

```
sequential_statements;
```

```
elsif boolean_expr_2 then
```

```
sequential_statements;
```

```
else
```

```
sequential_statements
```

```
end if;
```

```
end process;
```

Samtidig setning:
when-else

```
r <= a+b+c    when m = n else  
a-b          when m > n else  
c+1;
```

Sekvensiell setning:
if-else

```
process (m,n,a,b,c)  
begin  
  
    if (m = n) else  
        r <= a+b+c;  
    elsif (m > n) else  
        r <= a-b;  
    else  
        r <= c+1;  
    end if;  
end process;
```

If-Else Setning med Proses

• Eksempel 2

```
process (sensitivity_list)
```

```
begin
```

```
  if boolean_expr_1 then
```

```
    sequential_statements;
```

```
  elsif boolean_expr_2 then
```

```
    sequential_statements;
```

```
  elsif boolean_expr_2 then
```

```
    sequential_statements;
```

```
  else
```

```
    sequential_statements
```

```
  end if;
```

```
end process;
```

Table 3.4 Function table of a four-request priority encoder

input r	output pcode
1 ---	100
0 1 --	011
0 0 1 -	010
0 0 0 1	001
0 0 0 0	000



Deres koder?

If-Else Setning med Proses

• Eksempel 3:

```
process (sensitivity_list)
begin
    if boolean_expr_1 then
        sequential_statements;
    elsif boolean_expr_2 then
        sequential_statements;
    elsif boolean_expr_2 then
        sequential_statements;
    else
        sequential_statements
    end if;
end process;
```

Table 3.5 Truth table of a 2-to-4 decoder with enable

en	input		output
	a(1)	a(0)	y
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000



Deres koder?

Case-Setning med Proses

- **Syntaks**

- Bruk **sel-signal** for å velge et sett av sekvensiell setninger for utførelse
- Choice_i må være den **gyldige verdien** til sel-signalet eller **et sett av gyldige verdiene** til sel-signalet
- Utvalg må være **gjensidig eksklusive** og **alle-eksklusive**
- **Others** på slutten dekker ubrukte verdier
- **Case-setningen** og **with-select** setningen er tilsvarende

```
case sel is
```

```
when choice_1 =>
```

```
sequential statements ;
```

```
when choice_2 =>
```

```
sequential statements;
```

```
.....
```

```
when choice_n =>
```

```
sequential statements;
```

```
when others =>
```

```
sequential statements;
```

```
end case;
```

Case-Setning med Proses

- Eksempel

```
case sel is
```

```
when choice_1 =>
```

```
sequential statements ;
```

```
when choice_2 =>
```

```
sequential statements;  
.....
```

```
when choice_n =>
```

```
sequential statements;
```

```
when others =>
```

```
sequential statements;
```

```
end case;
```

```
signal sel : std_logic_vector(1 downto 0);  
|  
with sel select  
    r  <= a+b+c when "00",  
      a-b   when "10",  
      c+1   when others;
```



```
process (a,b,c,sel)  
begin  
  
    case sel is  
        when "00"  =>  
            r <= a+b+c;  
        when "10"  =>  
            r <= a-b;  
        when others =>  
            r <= c+1;  
    end case;  
  
end process;
```



Case-Setning med Proses

- Eksempel

```
case sel is
```

```
when choice_1 =>
```

```
sequential statements ;
```

```
when choice_2 =>
```

```
sequential statements;
```

```
.....
```

```
when choice_n =>
```

```
sequential statements;
```

```
when others =>
```

```
sequential statements;
```

```
end case;
```

Table 3.4 Function table of a four-request priority encoder

input r	output pcode
1 - - -	100
0 1 - -	011
0 0 1 -	010
0 0 0 1	001
0 0 0 0	000



Deres koder

Case-Setning med Proses

• Eksempel

```
case sel is
  when choice_1 =>
    sequential statements ;
  when choice_2 =>
    sequential statements;
    .....
  when choice_n =>
    sequential statements;
  when others =>
    sequential statements;
end case;
```

Table 3.5 Truth table of a 2-to-4 decoder with enable

en	input		output
	a(1)	a(0)	y
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000



Deres koder

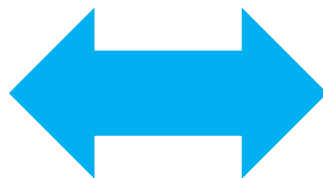


Samtidig Setninger vs. Sekvensiell Setninger

- **If-then-else** og **case** setninger tillater **hvilket som helst antall** og type sekvensielle setninger under hver gren. Derfor er det mye mer **fleksibilitet**
- Eksempel: En krets som **sorterer** verdier av inngangssignaler og avgir større verdi og mindre verdi

Samtidig setning when-else

```
større_verdi <= a when a > b else  
                b;  
  
mindre_verdi <= b when a > b else  
                a;
```



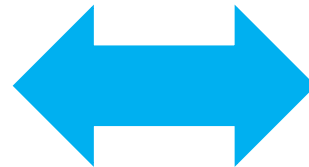
Sekvensiell setning if-else

```
process (a,b)  
begin  
    if (a > b) then  
        større_verdi <= a;  
        mindre_verdi <= b;  
    else  
        større_verdi <= a;  
        mindre_verdi <= b;  
    end if;  
end process;
```

Samtidig Setninger vs. Sekvensiell Setninger

Samtidig setning when-else

```
max_verdi <= a when ((a > b) and (a > c)) else  
              c when (a > b) else  
              b when (b > c) else  
              c;
```



Sekvensiell setning if-else

```
process (a,b)  
begin  
  if (a > b) then  
    if(a > c) then  
      max_verdi <= a;  
    else  
      max_verdi <= c;  
    end if;  
  else  
    if(b > c) then  
      max_verdi <= b;  
    else  
      max_verdi <= c;  
    end if;  
  end if;  
end process;
```

Utilsiktet minne/Unintended memory

- Et mest-oppstått problem er å inkludere **lagringselementer** i en kombinasjonskrets
- VHDL standard spesifiserer at et signal skal beholde sin tidligere verdi hvis det ikke blir tilordnet en verdi i en prosess. Dette fører til **intern tilstand** eller **lagringselement**
- Hvordan kan vi **unngå** utilsiktet minneelementer:
 - Regel 1: Inkluder **alle** inngangssignaler i **sensitivity-listen**
 - Regel 2: Inkluder **else-gren** i hver **if-then-else** setning og **others i case-setning**
 - Regel 3: Tilordne en verdi til **hvert** eneste signal i **hver** eneste gren

Utilsiktet minne/Unintended memory

- Eksempel

Ikke utilsiktet minne

```
process (a,b)
begin
  if (a > b) then
    gt <= '1';
    eq <= '0';
  elsif (a = b) then
    gt <= '0';
    eq <= '1';
  else
    gt <= '0';
    eq <= '0';
  end if;
end process;
end process;
```

Utilsiktet minne

```
process (a) -- missing b in sensitivity list
begin
  if (a > b) then -- eq not assigned
    gt <= '1';
  elsif (a = b) then -- gt not assigned
    eq <= '1';
  end if; -- else-branch is omitted
end process
```

Best Metode å Unngå Utilsiktet Minne

- Utfør tilordning standardverdi/default-value på begynnelsen

Ikke Utilsiktet minne ved bruk av standardverdi

```
process (a,b)
begin
  gt <= '0';-- assign default value
  eq <= '0';
  if( a > b) then
    gt <= '1';
  elsif (a = b) then
    eq <= '1';
  end if;
end process;
```

Utilsiktet minne

```
process (a) -- missing b in sensitivity list
begin
  if (a > b) then -- eq not assigned
    gt <= '1';
  elsif (a = b) then -- gt not assigned
    eq <= '1';
  end if; -- else-branch is omitted
end process
```

Konstant

- HDL-koder bruker ofte konstante verdier i **uttrykk** og **grenser** til datatabell
- Syntaks for konstant-deklarasjon

```
constant constant_name : datatype := value_expression;
```

- Eksempel

```
constant DATA_BIT : integer := 8;  
constant DATA_RANGE : integer := 2**DATA_BIT -1;
```

- Merk:
 - Konstant-uttrykk blir evaluert under pre-processing, og derfor trenger ikke fysisk maskinvare for det
 - Vanligvis bruker vi **STORE BOKSTAVER** for konstant sitt navn

Konstant

- Eksempel: Design en adderer med mente-bit

Bruk av direkte tall

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity add_w_carry is
  port(
    a, b: in std_logic_vector(3 downto 0);
    cout: out std_logic;
    sum: out std_logic_vector(3 downto 0)
  );
end add_w_carry;

architecture hard_arch of add_w_carry is
  signal a_ext, b_ext, sum_ext: unsigned(4 downto 0);
begin

  a_ext <= unsigned('0' & a);
  b_ext <= unsigned('0' & b);
  sum_ext <= a_ext + b_ext;
  sum <= std_logic_vector(sum_ext(3 downto 0));
  cout <= sum_ext(4);

end hard_arch;
```

Bruk av Konstant: N

```
architecture const_arch of add_w_carry is

  constant N: integer := 4;
  signal a_ext, b_ext, sum_ext: unsigned(N downto 0);

begin

  a_ext <= unsigned('0' & a);
  b_ext <= unsigned('0' & b);
  sum_ext <= a_ext + b_ext;
  sum <= std_logic_vector(sum_ext(N-1 downto 0));
  cout <= sum_ext(N);

end const_arch;
```


Generics

- VHDL gir oss en konstruksjon som blir kjent «**GENERIC**» for å passere informasjon til en entity og component
- Generic kan ikke modifiseres innenfor architecture
- Generics **funksjonalitet** er i noen grad samme «**constant**»
- Generic blir deklarerert **innenfor** en entity-deklarasjon, **rett før** port-deklarasjon

Generics

- Syntaks

```
entity entity_name is
  generic(
    generic_name_1 : datatype := default_values;
    generic_name_2 : datatype := default_values;
    ....
    generic_name_n : datatype := default_values
  );
  port (
    port_name_1 : mode datatype;
    port_name_2 : mode datatype;
    ....
    port_name_n : mode datatype);
end entity_name;
```

Generics

- Eksampel: Adderer med mente

```
entity entity_name is
  generic(
    generic_name_1 : datatype := default_values;
    generic_name_2 : datatype := default_values;
    ....
    generic_name_n : datatype := default_values
  );
  port (
    port_name_1 : mode datatype;
    port_name_2 : mode datatype;
    ....
    port_name_n : mode datatype);
end entity_name;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gen_add_w_carry is
  generic(N: integer:=4);
  port(
    a, b: in std_logic_vector(N-1 downto 0);
    cout: out std_logic;
    sum: out std_logic_vector(N-1 downto 0)
  );
end gen_add_w_carry;
```

Generics

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gen_add_w_carry is
    generic(N: integer:=4);
    port(
        a, b: in std_logic_vector(N-1 downto 0);
        cout: out std_logic;
        sum: out std_logic_vector(N-1 downto 0)
    );
end gen_add_w_carry;

architecture arch of gen_add_w_carry is
    signal a_ext, b_ext, sum_ext: unsigned(N downto 0);
begin
    a_ext <= unsigned('0' & a);
    b_ext <= unsigned('0' & b);
    sum_ext <= a_ext + b_ext;
    sum <= std_logic_vector(sum_ext(N-1 downto 0));
    cout <= sum_ext(N);
end arch;
```

Signal-deklarasjon

```
signal a4, b4, sum4      : unsigned(3 downto 0);
signal a8, b8, sum8      : unsigned(7 downto 0);
signal a16, b16, sum16   : unsigned(15 downto 0);
signal c4, c8, c16       : std_logic;
```

Instantiere adderer ved å passerer N = 8

```
----- instantiate 8-bit adder-----

adder_8_unit: work.gen_add_w_carry(arch)
    generic map (N => 8)
    port map (a => a8, b => b8, cout => c8, sum => sum8);
-----
```

Generics

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gen_add_w_carry is
  generic(N: integer:=4);
  port(
    a, b: in std_logic_vector(N-1 downto 0);
    cout: out std_logic;
    sum: out std_logic_vector(N-1 downto 0)
  );
end gen_add_w_carry;

architecture arch of gen_add_w_carry is
  signal a_ext, b_ext, sum_ext: unsigned(N downto 0);
begin
  a_ext <= unsigned('0' & a);
  b_ext <= unsigned('0' & b);
  sum_ext <= a_ext + b_ext;
  sum <= std_logic_vector(sum_ext(N-1 downto 0));
  cout <= sum_ext(N);
end arch;
```

Signal-deklarasjon

```
signal a4, b4, sum4      : unsigned(3 downto 0);
signal a8, b8, sum8      : unsigned(7 downto 0);
signal a16, b16, sum16   : unsigned(15 downto 0);
signal c4, c8, c16       : std_logic;
```

Instantiere adderer ved å passerer N = 16

```
----- instantiate 16-bit adder-----

adder_16_unit: work.gen_add_w_carry(arch)
  generic map (N => 16)
  port map (a => a16, b => b16, cout => c16, sum => sum16);
-----
```

Generics

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gen_add_w_carry is
  generic(N: integer:=4);
  port(
    a, b: in std_logic_vector(N-1 downto 0);
    cout: out std_logic;
    sum: out std_logic_vector(N-1 downto 0)
  );
end gen_add_w_carry;

architecture arch of gen_add_w_carry is
  signal a_ext, b_ext, sum_ext: unsigned(N downto 0);
begin
  a_ext <= unsigned('0' & a);
  b_ext <= unsigned('0' & b);
  sum_ext <= a_ext + b_ext;
  sum <= std_logic_vector(sum_ext(N-1 downto 0));
  cout <= sum_ext(N);
end arch;
```

Signal-deklarasjon

```
signal a4, b4, sum4      : unsigned(3 downto 0);
signal a8, b8, sum8      : unsigned(7 downto 0);
signal a16, b16, sum16   : unsigned(15 downto 0);
signal c4, c8, c16       : std_logic;
```

Instantiere adderer ved å passerer N = 4

```
----- instantiate 4-bit adder-----

adder_4_unit: work.gen_add_w_carry(arch)
  generic map (N => 4)
  port map (a => a4, b => b4, cout => c4, sum => sum4);
-----
```

Generics

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gen_add_w_carry is
  generic(N: integer:=4);
  port(
    a, b: in std_logic_vector(N-1 downto 0);
    cout: out std_logic;
    sum: out std_logic_vector(N-1 downto 0)
  );
end gen_add_w_carry;

architecture arch of gen_add_w_carry is
  signal a_ext, b_ext, sum_ext: unsigned(N downto 0);
begin
  a_ext <= unsigned('0' & a);
  b_ext <= unsigned('0' & b);
  sum_ext <= a_ext + b_ext;
  sum <= std_logic_vector(sum_ext(N-1 downto 0));
  cout <= sum_ext(N);
end arch;
```



```
----- instantiate 4-bit adder-----

adder_4_unit: work.gen_add_w_carry(arch)
  generic map (N => 4)
  port map (a => a4, b => b4, cout => c4, sum => sum4);

-----
```



```
----- instantiate 8-bit adder-----

adder_8_unit: work.gen_add_w_carry(arch)
  generic map (N => 8)
  port map (a => a8, b => b8, cout => c8, sum => sum8);

-----
```



```
----- instantiate 16-bit adder-----

adder_16_unit: work.gen_add_w_carry(arch)
  generic map (N => 16)
  port map (a => a16, b => b16, cout => c16, sum => sum16);

-----
```

Designeksempel

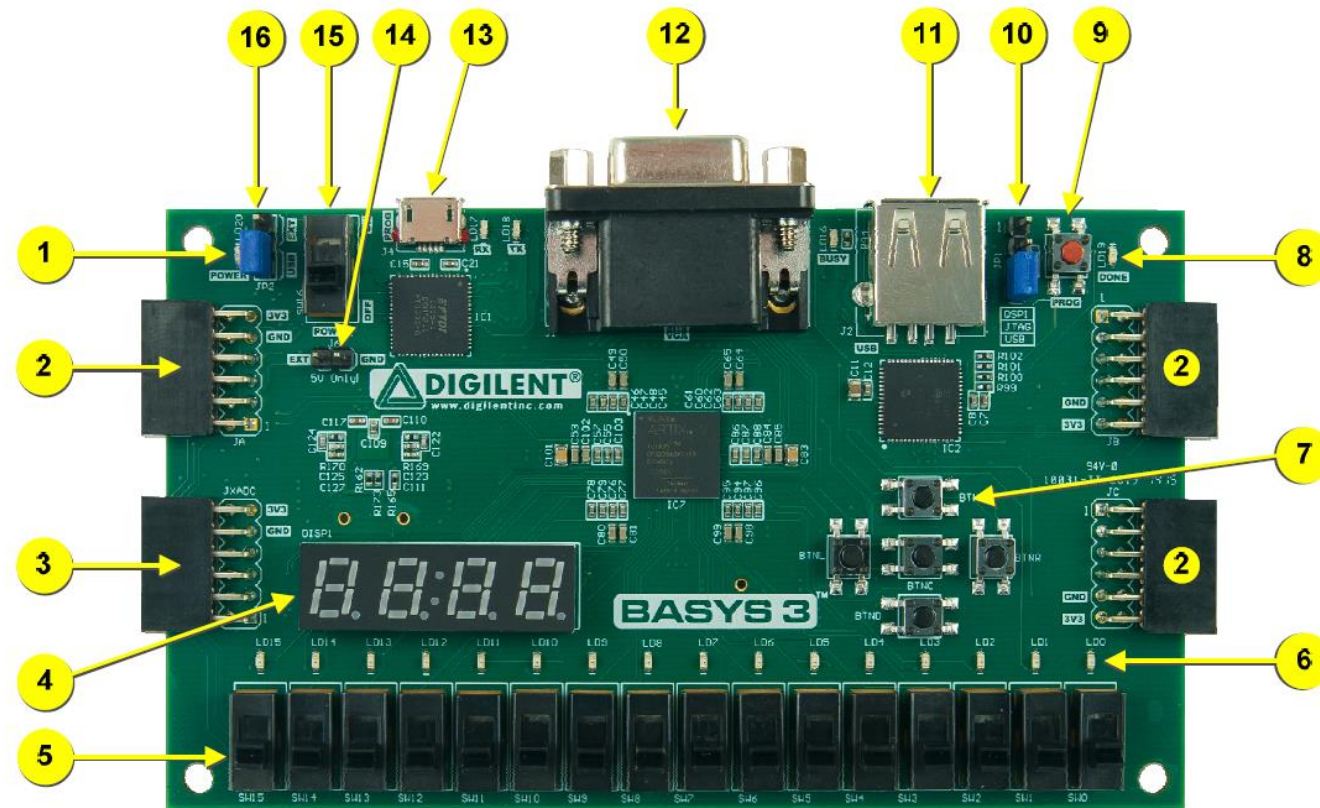
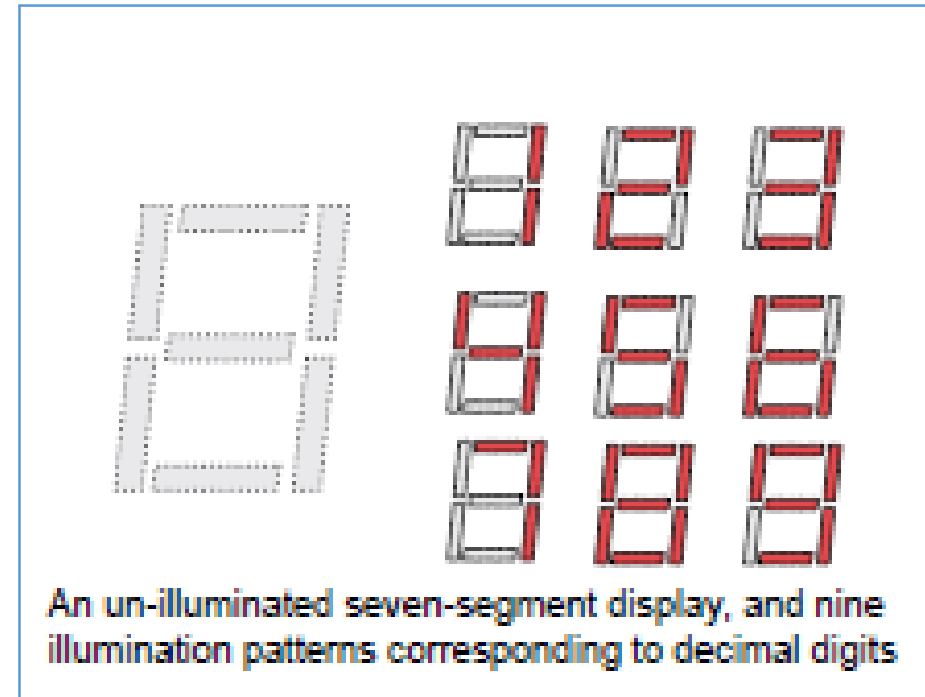


Figure 1. Basys3 board features

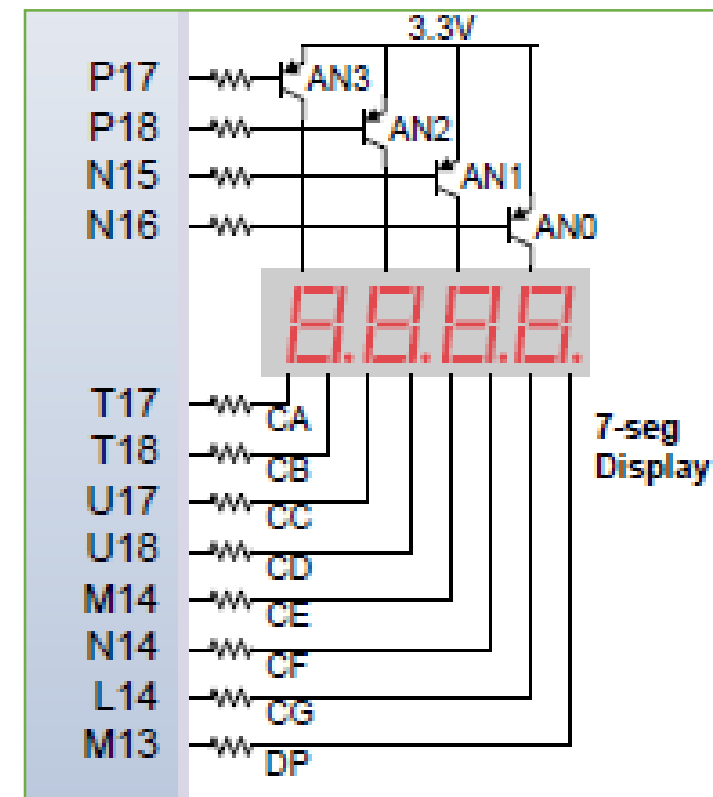
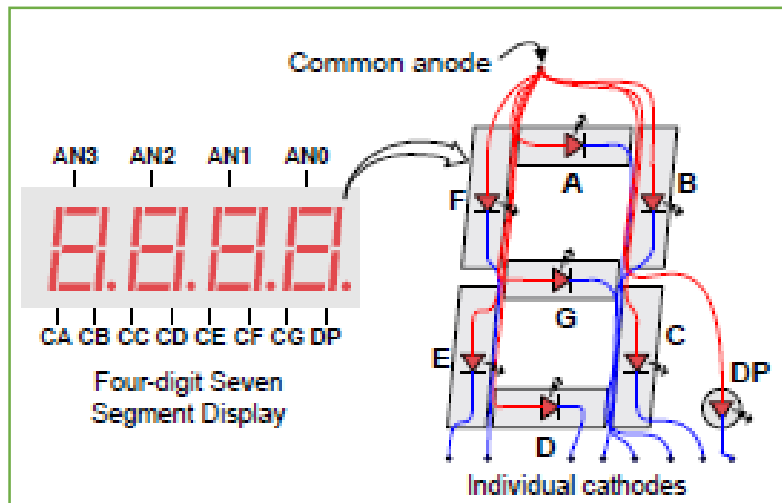
Designeksempel

- Et heksadesimal sifre for sju-segment LED (SSEG)-dekoder
 - 7-LED linjer og et desimalpunkt
 - På BASYS-3, blir SSEG konfigurert «active low», betyr at LED blir **lyst/tent** hvis tilsvarende reguleringssignal er **'0'**
 - Heksadesimal sifre til SSEG dekode har **4 biter** på inngangen, og genererer LED-mønster som vist



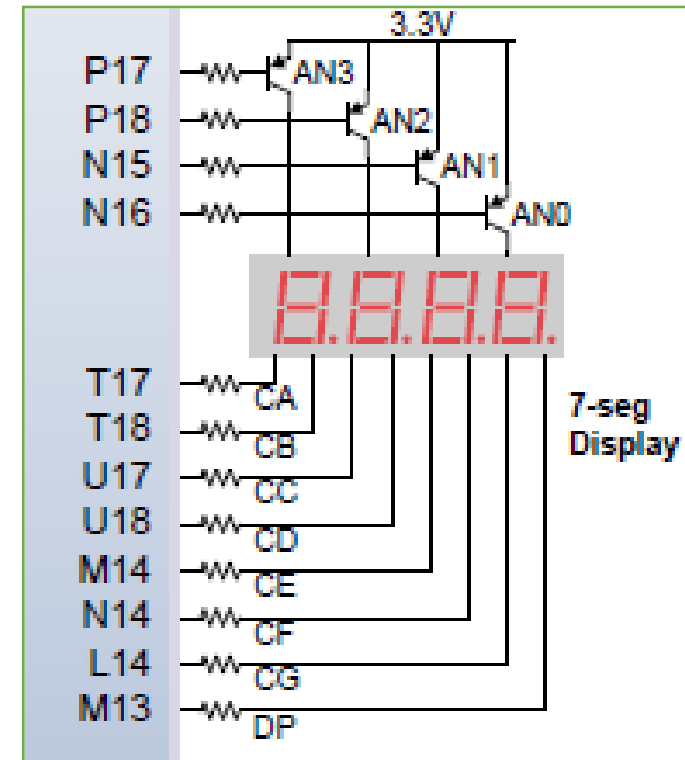
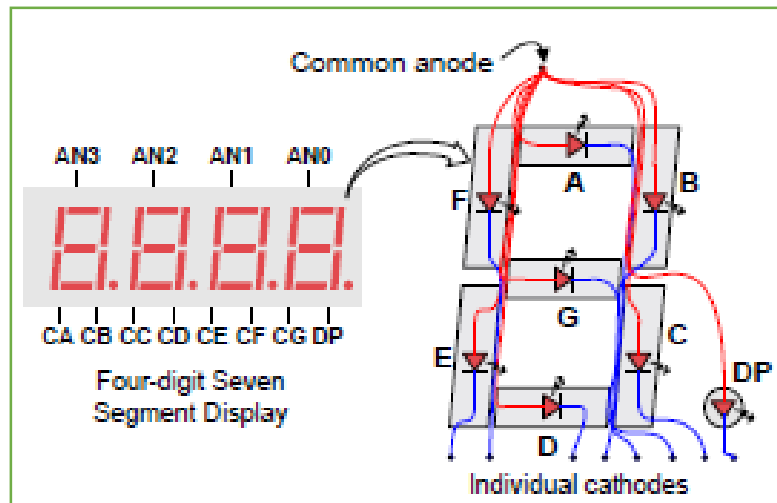
BASYS-3 SSEG LED

- BASYS-3 kort har 4- SSEG for fremvisning
- Katoder av SSEG display blir knyttet sammen og navngitt CA, CB, CG og DP



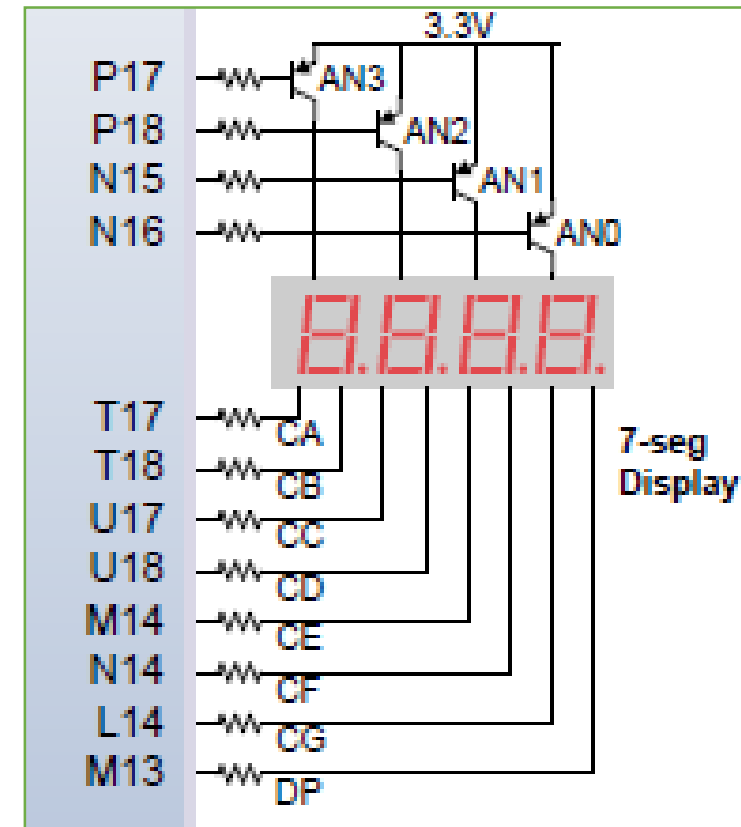
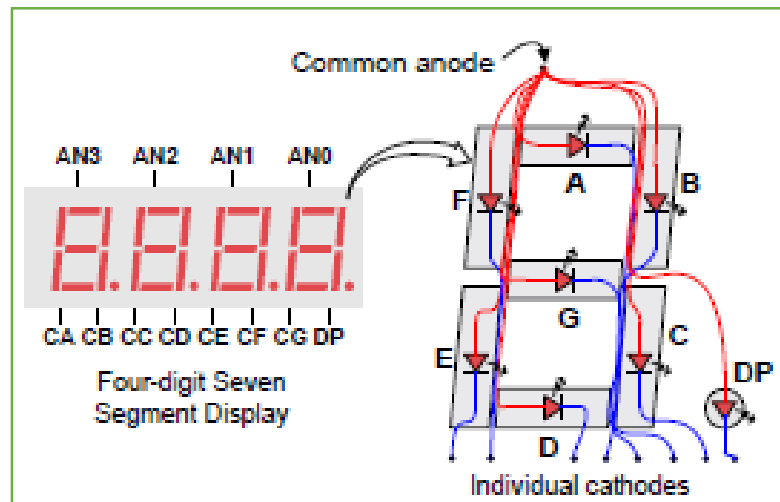
BASYS-3 SSEG LED

- Katodesignaler CA, CB, .. CG er felles for all 4-sifre
- Vi bruker **Anode** signal som **kontrollsignal** for å **slå på eller av et siffer**.



BASYS-3 SSEG LED

- Oppsummering: Både Anodene (AN0, AN1, AN3) og felles Katodene (CA,CB,..., CG, DP) blir aktive når de blir drevet til «lav»-nivå eller logikk '0'



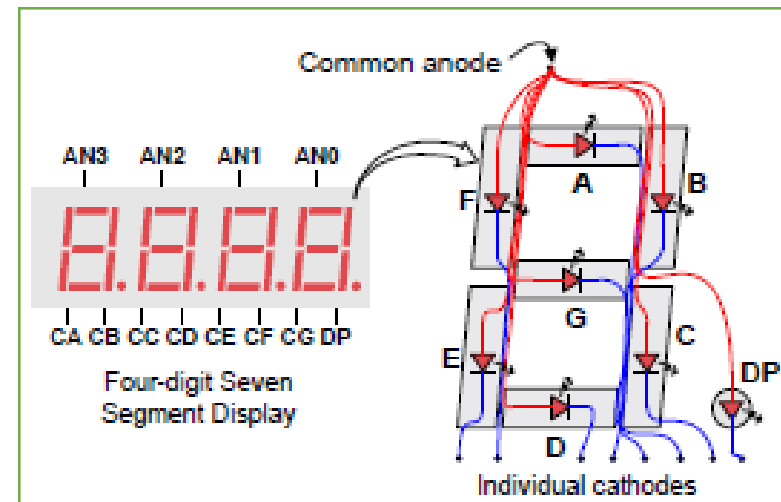
Heksadesimal SSEG LED Modul

- Modulbeskrivelse



Heksadesimal SSEG LED Modul

- Hex = «0000»
- AN0,....,AN3 = '0'
- CA på => CA = '0' | SEG(0) = '0'
- CB på => CB = '0' | SEG(1) = '0'
- CC på => CC = '0' | SEG(2) = '0'
- CD på => CD = '0' | SEG(3) = '0'
- CE på => CE = '0' | SEG(4) = '0'
- CF på => CF = '0' | SEG(5) = '0'
- CG av => CG = '1' | SEG(6) = '1'

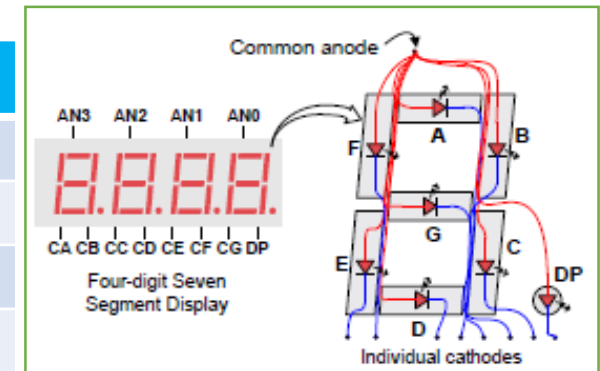


Heksadesimal SSEG LED Modul

- Sannhetstabell [0-7]

Hex (3 downto 0)	Hex character	SEG(6 downto 0)	Notes
0000	0	1 0 0 0 0 0 0	Turn off CG
0001	1	1 1 1 1 0 0 1	Off: CA, CD, CE, CF, CG
0010	2	0 1 0 0 1 0 0	off:CC,CF
0011	3	0 1 1 0 0 0 0	Off: CE, CF
0100	4	0 0 1 1 0 0 1	Off: CA,CD, CE
0101	5	0 0 1 0 0 1 0	Off: CB,CE
0110	6	0 0 0 0 0 1 0	Off: CB
0111	7	1 1 1 1 0 0 0	Off: CD, CE, CF, CG

Mapping	
SEG(0)	CA
SEG(1)	CB
SEG(2)	CC
SEG(3)	CD
SEG(4)	CE
SEG(5)	CF
SEG(6)	CG
SEG(7)	DP

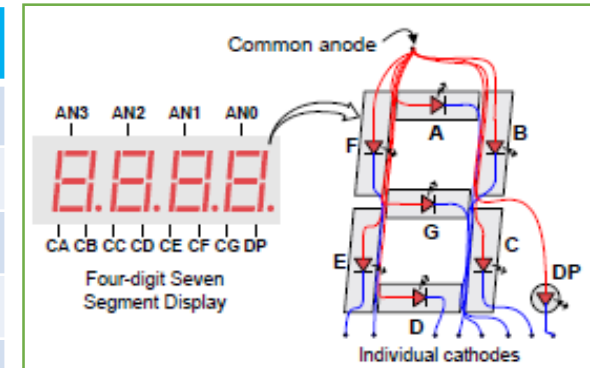


Heksadesimal SSEG LED Modul

• Sannhetstabell [8-15]

Hex (3 downto 0)	Hex character	Sseg(6 downto 0)	Notes
1000	8	0 0 0 0 0 0 0	Off: none
1001	9	0 0 1 0 0 0 0	Off: CE
1010	A	0 0 0 1 0 0 0	Off: CD
1011	B	0 0 0 0 0 1 1	Off: CA, CB
1100	C	1 0 0 0 1 1 0	Off: CB, CC, CG
1101	D	0 1 0 0 0 0 1	Off: CA, CF
1110	E	0 0 0 0 1 1 0	Off: CB, CC
1111	F	0 0 0 1 1 1 0	Off: CB, CC, CD

Mapping	
SEG(0)	CA
SEG(1)	CB
SEG(2)	CC
SEG(3)	CD
SEG(4)	CE
SEG(5)	CF
SEG(6)	CG
SEG(7)	DP



Heksadesimal SSEG LED Modul

- VHDL-Kode ved bruk av **case-setning**

Hex (3 downto 0)	Hex character	SEG(6 downto 0)	Notes
0000	0	1 0 0 0 0 0 0	Turn off CG
0001	1	1 1 1 1 0 0 1	Off: CA, CD, CE, CF, CG
0010	2	0 1 0 0 1 0 0	off:CC,CF
0011	3	0 1 1 0 0 0 0	Off: CE, CF
0100	4	0 0 1 1 0 0 1	Off: CA,CD, CE
0101	5	0 0 1 0 0 1 0	Off: CB,CE
0110	6	0 0 0 0 0 1 0	Off: CB
0111	7	1 1 1 1 0 0 0	Off: CD, CE, CF, CG

```
case sel is
    when choice_1 =>
        sequential statements ;
    when choice_2 =>
        sequential statements;
        .....
    when choice_n =>
        sequential statements;
    when others =>
        sequential statements;
end case;
```

Heksadesimal SSEG LED Modul

- VHDL-Kode ved bruk av **case-setning**

Hex (3 downto 0)	Hex character	Sseg(6 downto 0)	Notes
1000	8	0 0 0 0 0 0 0	Off: none
1001	9	0 0 1 0 0 0 0	Off:CE
1010	A	0 0 0 1 0 0 0	Off:CD
1011	B	0 0 0 0 0 1 1	Off:CA,CB
1100	C	1 0 0 0 1 1 0	Off: CB,CC, CG
1101	D	0 1 0 0 0 0 1	Off: CA,CF
1110	E	0 0 0 0 1 1 0	Off: CB, CC
1111	F	0 0 0 1 1 1 0	Off: CB,CC, CD

```
case sel is
```

```
when choice_1 =>
```

```
    sequential statements ;
```

```
when choice_2 =>
```

```
    sequential statements;  
    .....
```

```
when choice_n =>
```

```
    sequential statements;
```

```
when others =>
```

```
    sequential statements;
```

```
end case;
```

Heksadesimal SSEG LED Modul

- VHDL-Kode ved bruk av **if-setning**

Hex (3 downto 0)	Hex character	SEG(6 downto 0)	Notes
0000	0	1 0 0 0 0 0	Turn off CG
0001	1	1 1 1 1 0 0	Off: CA, CD, CE, CF, CG
0010	2	0 1 0 0 1 0	off:CC,CF
0011	3	0 1 1 0 0 0	Off: CE, CF
0100	4	0 0 1 1 0 0	Off: CA,CD, CE
0101	5	0 0 1 0 0 1	Off: CB,CE
0110	6	0 0 0 0 1 0	Off: CB
0111	7	1 1 1 1 0 0	Off: CD, CE, CF, CG

```
process (sensitivity_list)
begin
    if boolean_expr_1 then
        sequential_statements;
    elsif boolean_expr_2 then
        sequential_statements;
    elsif boolean_expr_2 then
        sequential_statements;
    else
        sequential_statements
    end if;
end process;
```

Heksadesimal SSEG LED Modul

- VHDL-Kode ved bruk av **if-setning**

Hex (3 downto 0)	Hex character	Sseg(6 downto 0)	Notes
1000	8	0 0 0 0 0 0 0	Off: none
1001	9	0 0 1 0 0 0 0	Off:CE
1010	A	0 0 0 1 0 0 0	Off:CD
1011	B	0 0 0 0 0 1 1	Off:CA,CB
1100	C	1 0 0 0 1 1 0	Off: CB,CC, CG
1101	D	0 1 0 0 0 0 1	Off: CA,CF
1110	E	0 0 0 0 1 1 0	Off: CB, CC
1111	F	0 0 0 1 1 1 0	Off: CB,CC, CD

```
process (sensitivity_list)
begin
    if boolean_expr_1 then
        sequential_statements;
    elsif boolean_expr_2 then
        sequential_statements;
    elsif boolean_expr_2 then
        sequential_statements;
    else
        sequential_statements
    end if;
end process;
```

Heksadesimal SSEG LED Modul

- Implementering

Pin Mapping	
SEG(0)	CA
SEG(1)	CB
SEG(2)	CC
SEG(3)	CD
SEG(4)	CE
SEG(5)	CF
SEG(6)	CG
SEG(7)	DP

Pin Mapping	
AN(3 downto 0)	An(3 downto 0)
DP	Push button center
Hex(3 downto 0)	SW(3 downto 0)