

Forelesning 03: RTL-Nivå Kombinasjonskrets

Hieu Nguyen

Innledning

- Vi undersøker HDL-beskrivelse for modul-nivå som består mellomstørrelseskomponenter slik: **adderer/adder, komparator/comparator, multiplekser/multiplexer**
- De tre komponentene er byggeblokker som blir brukt i **register-overføring-metode**. Designet basert på de tre blokkene blir derfor henvist som **RTL-design [Register Transfer Logic Design]**
- Vi først diskuterer om kompleks VHDL-operatorer og konstruksjoner for rutering, etterpå vi skal demonstrere design av en RTL-kombinasjonal kretse gjennom en rekke eksempler

RTL-nivå komponenter

- **Logiske** operatorer
- **Forhold**-operatorer
- **Aritmetiske** operatorer

RTL-nivå Komponenter

- IEEE std_logic_1164 pakke

Table 3.1 Operators and data types of VHDL-93 and IEEE std_logic_1164 package

Operator	Description	Data type of operands	Data type of result
a ** b	exponentiation	integer	integer
a * b	multiplication		
a / b	division	<i>integer type for constants and array boundaries, not synthesis</i>	
a + b	addition		
a - b	subtraction		
a & b	concatenation	1-D array, element	1-D array
a = b	equal to	any	boolean
a /= b	not equal to		
a < b	less than	scalar or 1-D array	boolean
a <= b	less than or equal to		
a > b	greater than		
a >= b	greater than or equal to		
not a	negation	boolean, std_logic, std_logic_vector	same as operand
a and b	and		
a or b	or		
a xor b	xor		



Aritmetiske operasjoner



Forhold-operasjoner



Logiske operasjoner

RTL-nivå Komponenter

- IEEE std_logic_1164 pakke

Table 3.1 Operators and data types of VHDL-93 and IEEE std_logic_1164 package

Operator	Description	Data type of operands	Data type of result
a ** b	exponentiation	integer	integer
a * b	multiplication	<i>integer type for constants and array boundaries, not synthesis</i>	
a / b	division		
a + b	addition		
a - b	subtraction		
a & b	concatenation	1-D array, element	1-D array
a = b	equal to	any	boolean
a /= b	not equal to	scalar or 1-D array	boolean
a < b	less than		
a <= b	less than or equal to		
a > b	greater than		
a >= b	greater than or equal to		
not a	negation	boolean, std_logic, std_logic_vector	same as operand
a and b	and		
a or b	or		
a xor b	xor		



Ny operasjon (se på denne senere)

RTL-nivå Komponenter


- **Aritmetiske operatører**

- I VHDL-standard, blir aritmetiske operasjoner definert for **heltall datatype** og for **naturlig datatype**
- Vi foretrekker å bruke datatypen som er **syntetiserbare** og vi bør definere **eksakt** antall biter og formatet til datatype (**dvs: signed, unsigned**).
 - Pakken **numeric_std** blir utviklet for denne hensikten
- Merk: Vi bruker datatype av heltall og naturlig tall for **Konstant** og **datatabell/array-begrensning**, men **ikke for syntese**.

RTL-nivå Komponenter

- **Numeric_std pakke**

- Tillegg nye datatyper: **unsigned** og **signed**
- Definer forhold- og **aritmetiske** operatorer for de nye datatypene
- Vi må legge en ekstra setning i **bibliotek-deklarasjon** til å anrope/invoke denne pakken



```
21  
22 library IEEE;  
23 use IEEE.STD_LOGIC_1164.ALL;  
24 use IEEE.numeric_std.ALL;  
25
```

RTL-nivå Komponenter

- Numeric_std pakke

Table 3.2 Overloaded operators and data types in the IEEE numeric_std package

Overloaded operator	Description	Data type of operands	Data type of result
a * b a + b a - b	arithmetic operation	unsigned, natural signed, integer	unsigned signed
a = b a /= b a < b a <= b a > b a >= b	relational operation	unsigned, natural signed, integer	boolean boolean



Aritmetiske operasjoner



Forhold-operasjoner

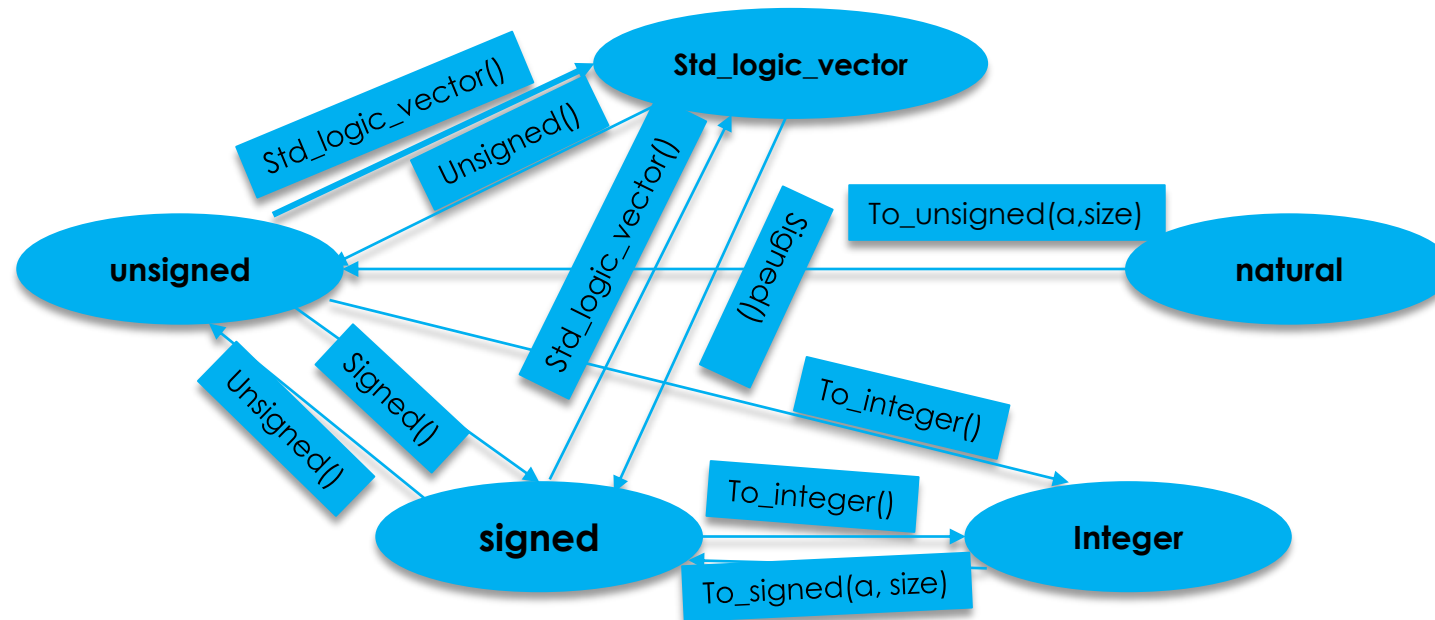
Datatype-Konvertering

- `Std_logic_vector`, `unsigned`, `signed` blir behandlet **forskjellig**
- Konverteringsfunksjon eller **typecasting** er nødt til å konvertere signaler med forskjellige datatyper

Data type of a	To data type	Conversion function/type casting
<code>unsigned, signed</code>	<code>std_logic_vector</code>	<code>std_logic_vector(a)</code>
<code>signed, std_logic_vector</code>	<code>unsigned</code>	<code>unsigned(a)</code>
<code>unsigned, std_logic_vector</code>	<code>signed</code>	<code>signed(a)</code>
<code>unsigned, signed</code>	<code>integer</code>	<code>to_integer(a)</code>
<code>natural</code>	<code>unsigned</code>	<code>to_unsigned(a, size)</code>
<code>integer</code>	<code>signed</code>	<code>to_signed(a, size)</code>

Datatype-Konvertering

- Diagram-presentasjon



Datatype-Konvertering

- Eksempel

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
...  
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);
```

Teste med Vivado-vhdl!!!!

```
signal u1, u2, u3, u4, u5, u6, u7: unsigned(3 downto 0);
```

```
u1 <= s1;  -- not ok, type mismatch  
u2 <= 5;   -- not ok, type mismatch  
s2 <= u3;  -- not ok, type mismatch  
s3 <= 5;   -- not ok, type mismatch
```

```
u1 <= unsigned(s1);    -- ok, type casting  
u2 <= to_unsigned(5,4); -- ok, conversion function  
s2 <= std_logic_vector(u3); -- ok, type casting  
s3 <= std_logic_vector(to_unsigned(5,4)); -- ok
```

Datatype-Konvertering

- Eksempel

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
...  
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);
```

Teste med Vivado-vhdl!!!!

```
signal u1, u2, u3, u4, u5, u6, u7: unsigned(3 downto 0);
```

```
u4 <= u2 + u1;  -- ok, both operands unsigned  
u5 <= u2 + 1;   -- ok, operands unsigned and natural
```

```
s5 <= s2 + s1;  -- not ok, + undefined over the types  
s6 <= s2 + 1;  -- not ok, + undefined over the types
```

Datatype-Konvertering

- Eksempel

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
...  
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);
```

```
signal u1, u2, u3, u4, u5, u6, u7: unsigned(3 downto 0);
```

```
s5 <= std_logic_vector(unsigned(s2) + unsigned(s1)); -- ok  
s6 <= std_logic_vector(unsigned(s2) + 1);           -- ok
```

Teste med Vivado-vhdl!!!!

Sammenkjedingsoperator Concatenation

- Sammenkjedingsoperator **kombinerer** segmenter av elementer eller **små** datatabeller for å danne en **stor** datatabell
- Eksempel**

```
signal a1: std_logic;  
signal a4: std_logic_vector(3 downto 0);  
signal b8, c8, d8: std_logic_vector(7 downto 0);
```

```
...  
b8 <= a4 & a4;  
c8 <= a1 & a1 & a4 & "00";  
d8 <= b8(3 downto 0) & c8(3 downto 0);
```



Hvis a1 = 1, a4 = 1100; da får vi
b8 = «1100-1100»
c8 = 11110000
d8 = 1100-0000

Sammenkjedingsoperator Concatenation

- En hoved anvendelse av sammenkjedningsoperatoren er å utføre **skift-operasjon**
- **Eksempel**

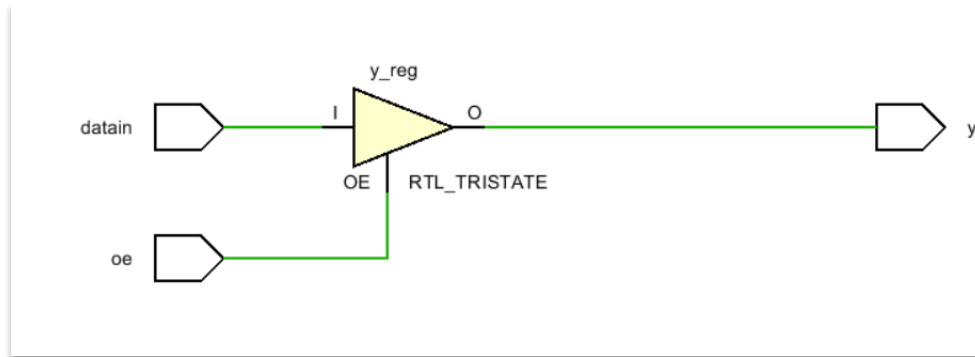
```
signal a : std_logic_vector(7 downto 0);  
signal rot, shl, sha : std_logic_vector(7 downto 0);  
  
-- rotate a to right 3 bits  
rot <= a(2 downto 0) & a(7 downto 3);  
  
--shift a to right 3 bits og insert 0  
shl <= "000" & a(7 downto 3);  
  
--shift s right 3 bits and insert MSB  
sha <= a(7) & a(7) & a(7) & a(7 downto 3);
```



For example: a := «11001010»
rot <= «010-11001»
shl <= «000-11001»
sha <= «11111001»

'Z' - Verdi i Std_Logic Datatype

- 'Z' verdi innebærer **høy impedans** eller **åpen krets**
- 'Z' verdi kan syntetiseres ved bruk av **tri-state buffer/tre-tilstanders** buffer

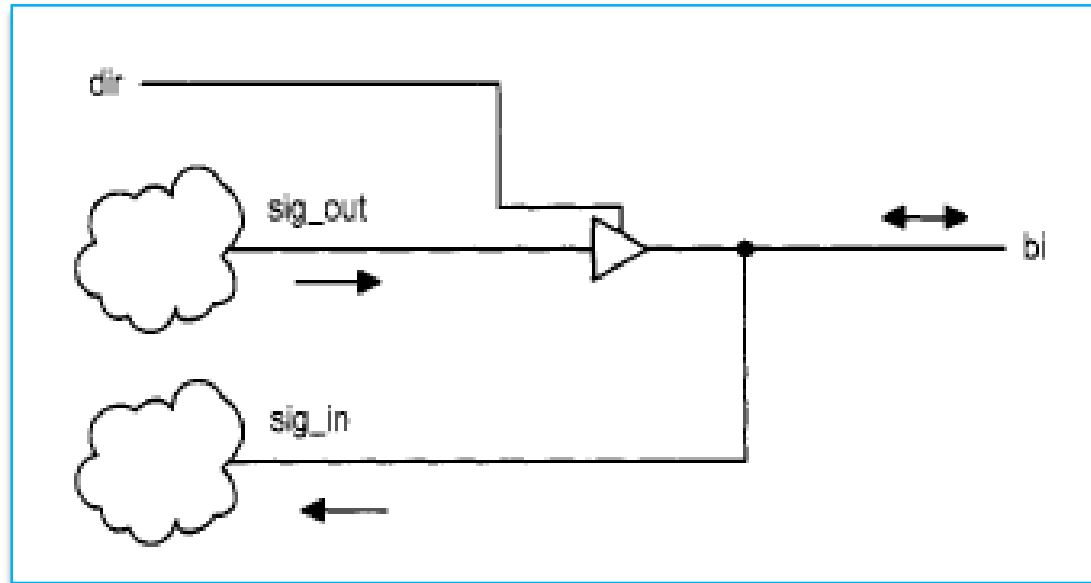


oe	y
1	Datain
0	'Z'

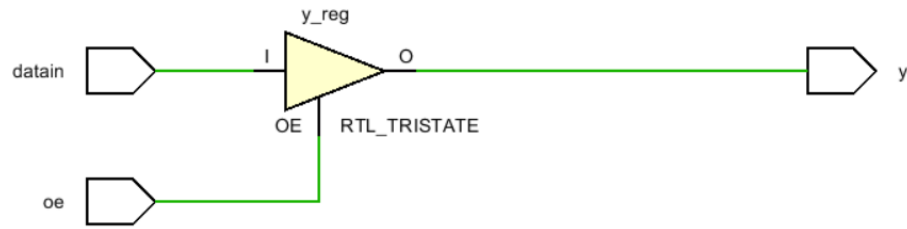
- Operasjon av bufferet blir kontrollert av **enable-signal** «oe»
 - Hvis oe = '1', inngangssignal slipper gjennom. Ellers utgangssignal fremstår som en åpen krets

'Z' - Verdi i Std_Logic Datatype

- Anvendelse av tri-state buffer



When-Else Setning for Tri-state Buffer



oe	y
1	Datain
0	'Z'

Modelling i VHDL

```
41 begin
42   y <= datain when oe = '1' else
43       'Z';
44
45 end Behavioral;
```

Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

Technical Specification

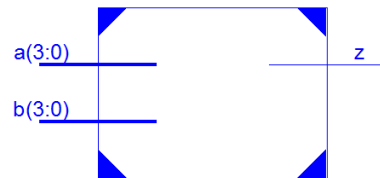
a (3 downto 0)	b (3 downto 0)	z
a = b		1
a /= b		0



RTL-kode med when-else setning

```
38 architecture Behavioral of four_bit_c
39
40 begin
41
42     z <= '1' when a = b else
43         '0';
44
45 end Behavioral;
```

four_bit_comparator_when_else



four_bit_comparator_when_else

Ruteringskrets med Samtidig Tilordningssetning

When-Else Setning

Syntaks

```
signal_name <= value_expr_1 when boolean_expr_1 else  
               value_expr_2 when boolean_expr_2 else  
               value_expr_3 when boolean_expr_3 else  
               .....  
               value_expr_n;
```

```
38 architecture Behavioral of four_bit_c  
39  
40 begin  
41  
42     z <= '1' when a = b else  
43         '0';  
44  
45 end Behavioral;
```

- Boolske uttrykk blir evaluert på tur til et uttrykk sann/true logikk, og tilsvarende **value_expr** verdiuttrykk blir tilordnet til **signal_name** på venstre signal.
- **Value_expr_n** blir tilordnet til **signal_name** hvis alle boolske uttrykk er usanne.

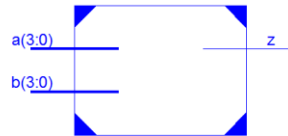
Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

Greater than comparator

Technical Specification

a (3 downto 0)	b (3 downto 0)	z
a > b		1
a <= b		0

four_bit_greater_than_comparator



four_bit_greater_than_comparator

Syntaks

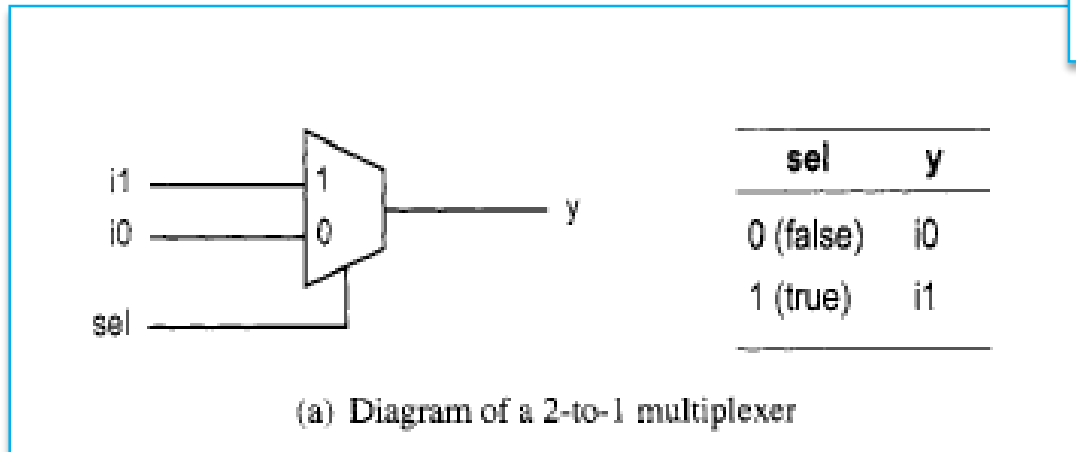
```
signal_name    <=  value_expr_1 when boolean_expr_1 else  
                   value_expr_2 when boolean_expr_2 else  
                   value_expr_3 when boolean_expr_3 else  
                   .....  
                   |value_expr_n;
```



Deres koder????

Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

- Ruting blir utført ved bruk av 2-til-1 multiplekser



```
signal_name  <=  value_expr_1 when boolean_expr_1 else  
                value_expr_2 when boolean_expr_2 else  
                value_expr_3 when boolean_expr_3 else  
                .....  
                |value_expr_n;
```

Deres koder???

sel signal er en bit, men *i0* og *i1* kan ha hvilket som helst antall biter. Vi velger 8 bit for *i0* og *i1*

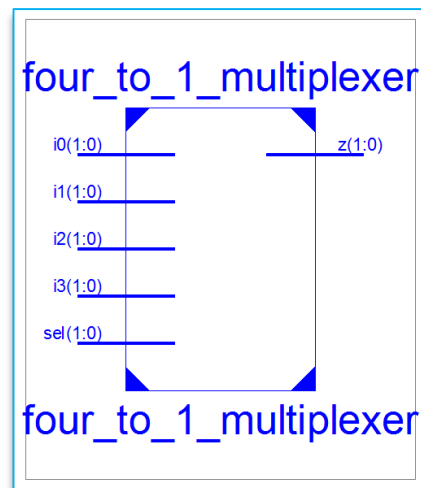
Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

4_to_1 multiplexer

Technical Specification

Sel	Z(1 downto 0)
00	I0(1 downto 0)
01	I1(1 downto 0)
10	I2(1 downto 0)
11	I3(1 downto 0)

```
signal_name    <=  value_expr_1 when boolean_expr_1 else  
                   value_expr_2 when boolean_expr_2 else  
                   value_expr_3 when boolean_expr_3 else  
                   .....  
                   |value_expr_n;
```



Deres koder???

Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

Test av four_to_one multiplexer on Basys 3

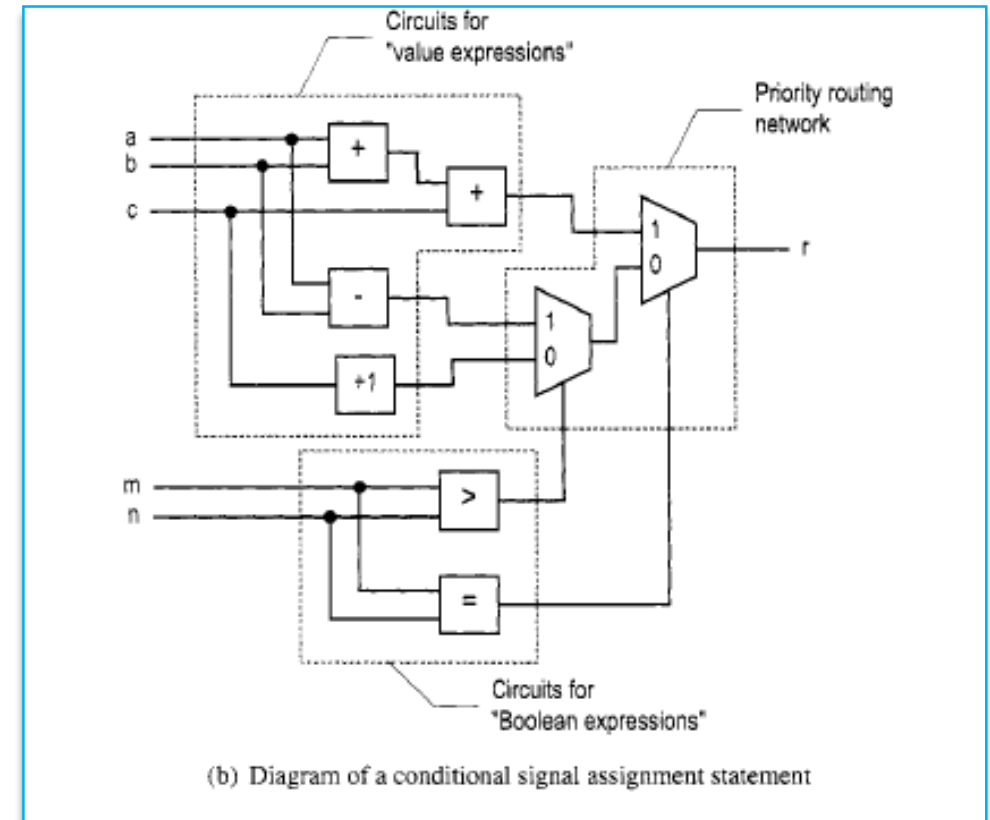
- Opprette en vhdl_modul med navn: four_to_one_multiplexer_top med ingangs og utgangssignal som vist
- Pin tilordning

Signal	Basys-3 pin	External leds
I0(1 downto 0)	Sw(1 down to 0)	
I1(1 downto 0)	Sw(3 down to 2)	
I2(1 downto 0)	Sw(5 down to 4)	
I3(1 downto 0)	Sw(7 down to 6)	
Z(1 downto 0)	Led(1 downto 0)	
Sel(1)	Push button Right	
Sel (0)	Push button Left	

Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

- Eksempel

```
83  
84 r <= a+b+c    when m = n else  
85               a-b    when m > n else  
86               c+1;  
87
```

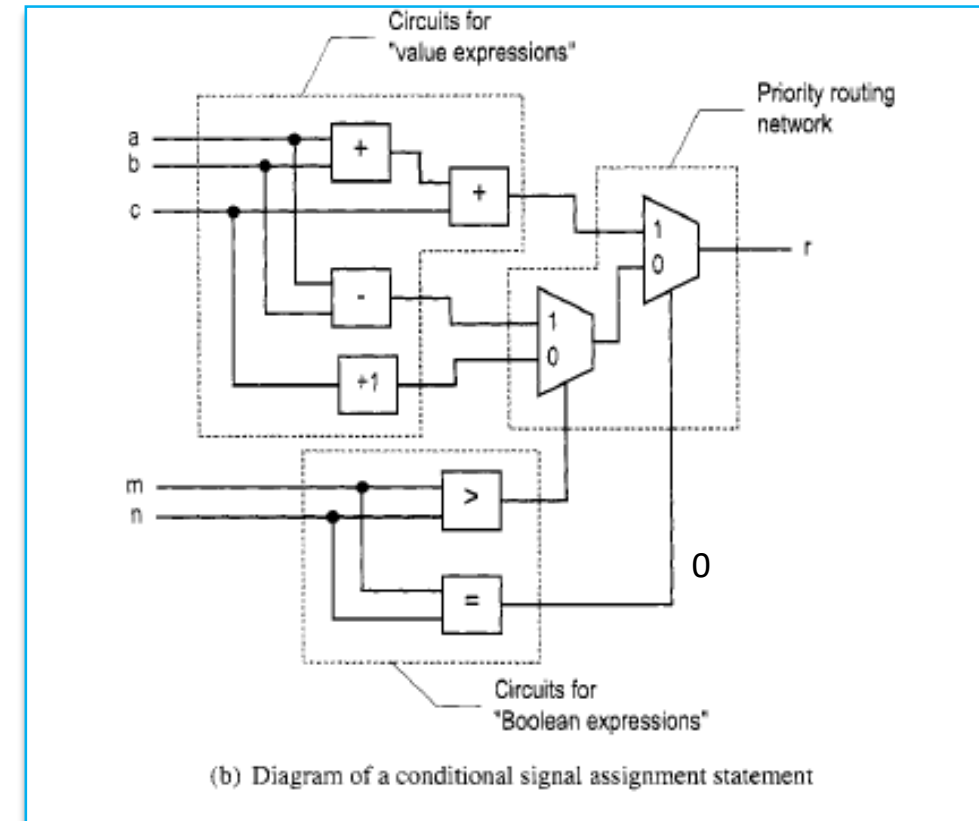


Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

- Eksempel

```
83  
84  r <= a+b+c    when m = n else  
85      a-b        when m > n else  
86      c+1;  
87
```

Tilfelle 1: $m > n$

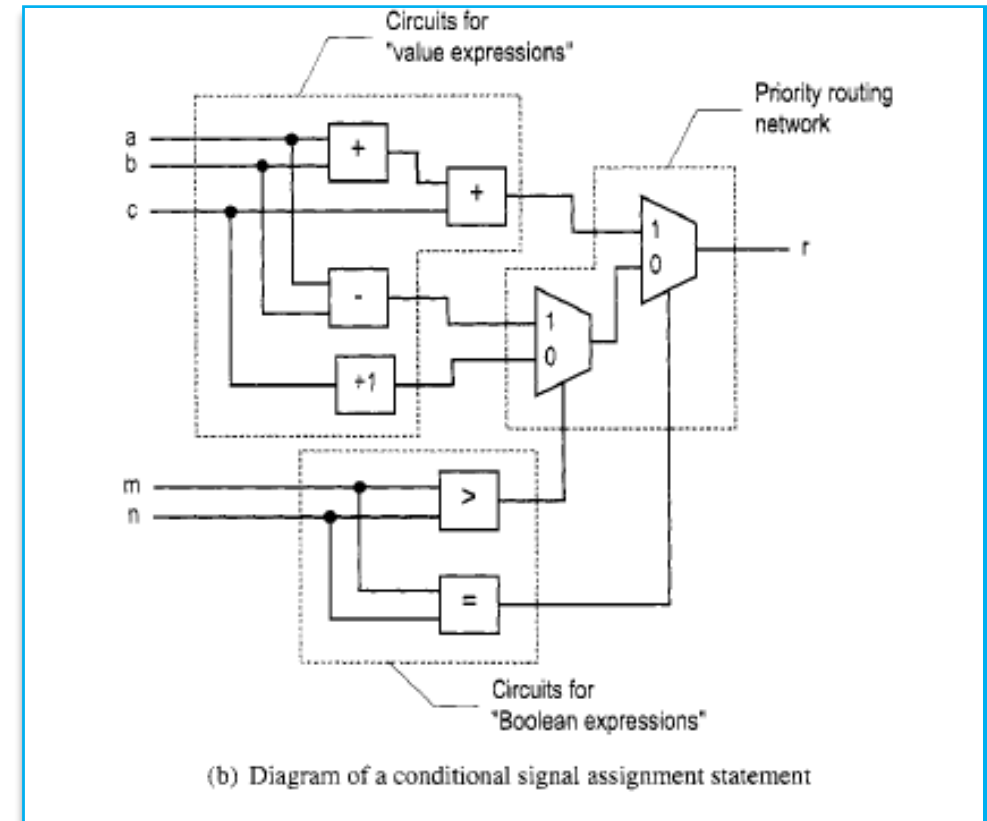


Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

- Eksempel

```
83  
84 r <= a+b+c    when m = n else  
85      a-b      when m > n else  
86      c+1;  
87
```

Tilfelle 2: $m = n$

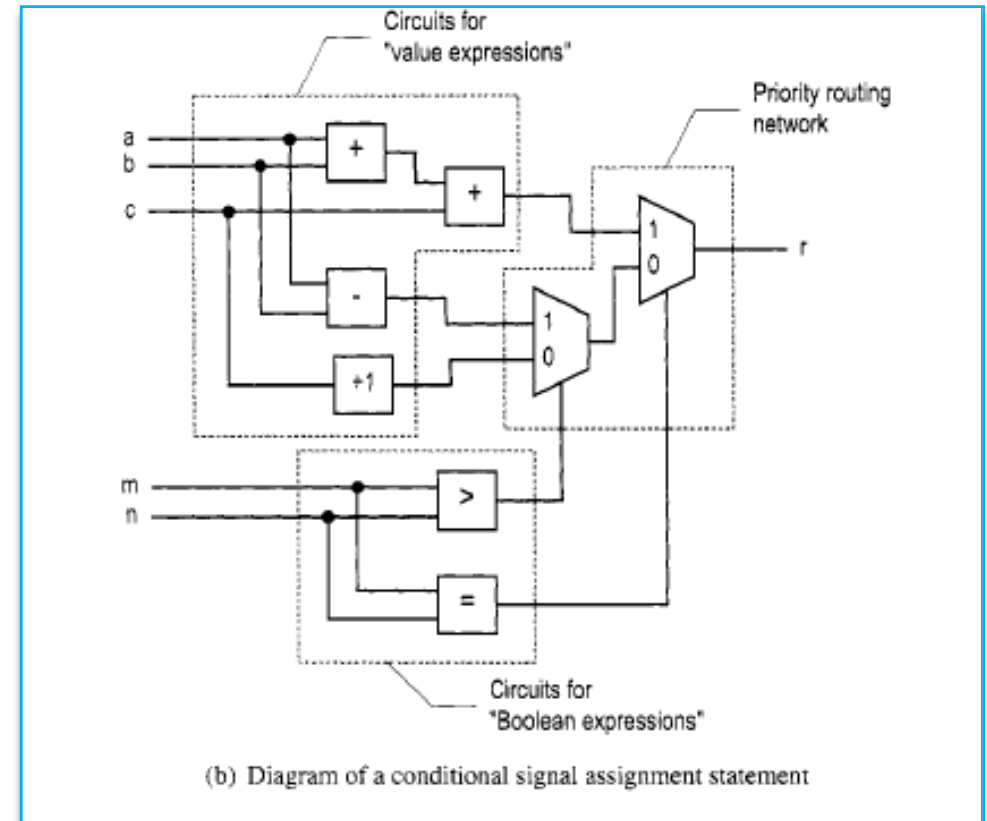


Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

- Eksempel

```
83  
84 r <= a+b+c   when m = n else  
85             a-b   when m > n else  
86             c+1;  
87
```

Tilfelle 3: $m < n$



Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

- Eksempel: Priority encoder/ Prioritetsenkoder
 - Merk: '-' betyr 'don't care'

Table 3.4 Function table of a four-request priority encoder

input r	output pcode
1---	100
01--	011
001-	010
0001	001
0000	000

```
1  --Listing 3.1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  entity prio_encoder is
5      port(
6          r: in std_logic_vector(4 downto 1);
7          pcode: out std_logic_vector(2 downto 0)
8      );
9  end prio_encoder;
10
11 architecture cond_arch of prio_encoder is
12 begin
13     pcode <= "100" when (r(4)='1') else
14             "011" when (r(3)='1') else
15             "010" when (r(2)='1') else
16             "001" when (r(1)='1') else
17             "000";
18 end cond_arch;
```

Ruteringskrets med Samtidig Tilordningssetning When-Else Setning

- **Eksempel** : n -til- 2^n binær dekode

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

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

```
2 library ieee;
3 use ieee.std_logic_1164.all;
4 entity decoder_2_4 is
5     port(
6         a: in std_logic_vector(1 downto 0);
7         en: in std_logic;
8         y: out std_logic_vector(3 downto 0)
9     );
10 end decoder_2_4;
11
12 architecture cond_arch of decoder_2_4 is
13 begin
14     y <= "0000" when (en='0') else
15         "0001" when (a="00") else
16         "0010" when (a="01") else
17         "0100" when (a="10") else
18         "1000"; -- a="11"
19 end cond_arch;
20
```

Ruteringskrets med samtidig tilordningssetning with-select setning

Four_to_one_muxplexer igjen

Technical Specification

Sel	Z(1 downto 0)
00	I0(1 downto 0)
01	I1(1 downto 0)
10	I2(1 downto 0)
11	I3(1 downto 0)



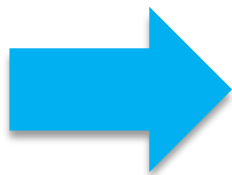
With-select koder

```
41 architecture Behavioral of four_to_
42 begin
43   with sel select
44     z <= i0 when "00",
45         i1  when "01",
46         i2  when "10",
47         i3  when others; -- "11"
48 end Behavioral;
49
```

Ruteringskrets med samtidig tilordningssetning with-select setning

Technical Specification 4-to-1 multiplexer

Sel	Z(1 downto 0)
00	I0(1 downto 0)
01	I1(1 downto 0)
10	I2(1 downto 0)
11	I3(1 downto 0)



With-select koder

```
41 architecture Behavioral of four_to_
42 begin
43   with sel select
44     z <= i0 when "00",
45         i1  when "01",
46         i2  when "10",
47         i3  when others; -- "11"
48 end Behavioral;
49
```


Ruteringskrets med samtidig tilordningssetning with-select setning

Syntaks

```
75  
76 with sel select  
77 signal_name <= value_expr_1 when choice_1,  
78                 value_expr_2 when choice_2,  
79                 value_expr_3 when choice_3,  
80                 .....  
81                 value_expr_n when others;
```



With-select koder

```
41 architecture Behavioral of four_to_2  
42 begin  
43 with sel select  
44     z <= i0 when "00",  
45         i1 when "01",  
46         i2 when "10",  
47         i3 when others; -- "11"  
48 end Behavioral;  
49
```

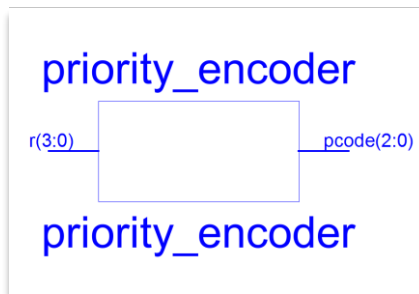
- Syntaks for **with-select** setning
- **Choice_i** må være en **gyldig** verdi eller et sett av gyldige verdier av «sel»
- **Choice_1, Choice_2, ... Choice_n** må **gjensidig eksklusive** (ingen verdi blir brukt mer enn en gang)
- **Alle** verdiene til «sel» må brukes
- Reservert ord: **others** blir brukt for å dekke alle **ubrukte** verdier

Ruteringskrets med samtidig tilordningssetning with-select setning

- Prioritet-dekoder

Table 3.4 Function table of a four-request priority encoder

input r	output pcode
1---	100
01--	011
001-	010
0001	001
0000	000



```
75  
76 with sel select  
77 signal_name  <= value_expr_1 when choice_1,  
78              value_expr_2 when choice_2,  
79              value_expr_3 when choice_3,  
80              .....  
81              value_expr_n when others;
```



Deres koder????

Ruteringskrets med samtidig tilordningssetning with-select setning

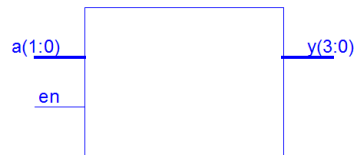
- Eksempel n -til- 2^n binær dekode

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

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

```
75
76 with sel select
77 signal_name  <= value_expr_1 when choice_1,
78               value_expr_2 when choice_2,
79               value_expr_3 when choice_3,
80               .....
81               value_expr_n when others;
```

two_to_four_encoder_with_select



two_to_four_encoder_with_select



Deres koder????

Ruteringskrets med samtidig tilordningssetning with-select setning

- Implementering 2-to-4 encoder

Signal	Basys-3 pin
En	Button center
a(1 downto 0)	Sw(1 downto 0)
Y(3 downto 0)	Led(3 downto 0)

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

en	a(1 downto 0)	Led
0	00	Ingen lyst
0	01	Ingen lyst
0	10	Ingen lyst
0	11	Ingen lyst
1	00	Led0 lyst
1	01	Led1 lyst
1	10	Led2 lyst
1	11	Led3 lyst