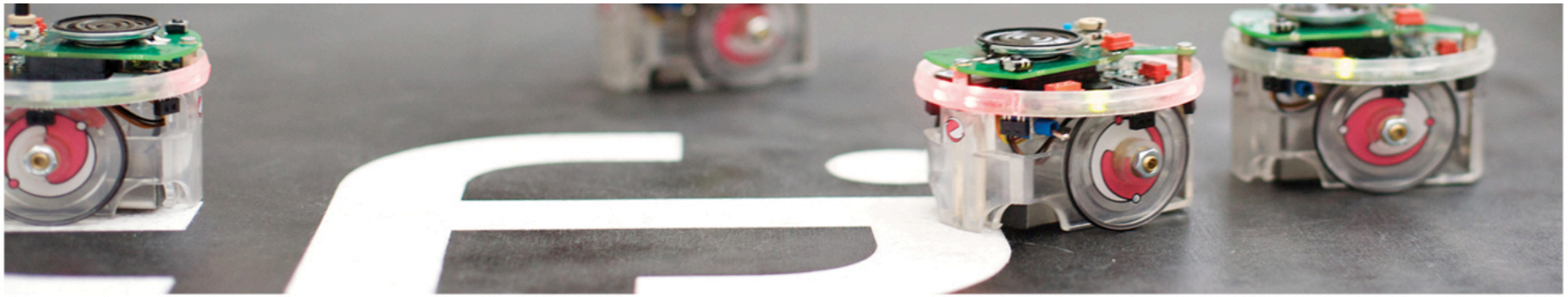


UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

IN1020

Logiske porter om forenkling til ALU

Hovedpunkter

- Utlesing av sannhetsverdi-tabell; Max og Min-termer
- Forenkling av uttrykk med Karnaugh diagram
- Portimplementasjon
- Kretsanalyse
- Adder og subtraktor design
- Generell Black-box design/tankesett
- Enkoder og Dekoder
- Multiplexer og DeMultiplexer
- ALU
- Datapath

Minterm

I en funksjon kan en binær variabel x opptre som x eller x´

En funksjon kan være gitt på ***"sum av produkt"*** form

Eksempel:

$$F = xy + xy' + x$$

Hvert "produktledd" som inneholder **alle** variablene kalles en **minterm**.

For to variable finnes det 4 forskjellige mintermer:

$$xy + xy' + x'y + x'y'$$

For 3 variable finnes det 2^3 forskjellige mintermer

Maksterm

En funksjon kan være gitt på "produkt av sum" form

Eksempel:

$$F = (x+y)(x+y´)y$$

Hvert "summeledd" som inneholder alle variablene kalles en **maksterm**

For to variable finnes det 4 forskjellige makstermer:

$$(x+y)\ (x+y´)\ (x´+y)\ (x´+y´)$$

For n variable finnes det 2^n forskjellige makstermer

Sannhetstabell / mintermer

Hvis man genererer en funksjon ut i fra sannhetstabellen får man en sum av mintermer

Eksempel:

$$F = x´y´z + xy´z´ + xyz´$$

En sannhetstabell kan sees på som en liste av mintermer

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Notasjon

Mintermer har notasjon m_x

Maxtermer har notasjon M_x

Funksjoner som bare består av sum/produkt av min/maxtermer (kanonisk form) har følengde notasjon:

$$F(x,y,z) = S(m_3, m_6) = S(3, 6) = x'yz+xyz'$$

$$F(x,y,z) = \Pi(M_3, M_6) = \Pi(3, 6) = (x+y'+z')(x'+y'+z)$$

Karnaughdiagram

Grafisk metode for forenkling av Boolske uttrykk

- Uttrykket må være representert ved sum av mintermer (m_x).
  Disse leser vi direkte ut av sannhetstabellen
- Metoden egner seg for funksjoner med 2-4(5) variable

Eksempel, 2 variable:

$$F = m_1 + m_3 = a\acute{}b + ab$$

Eksempel, 4 variable:

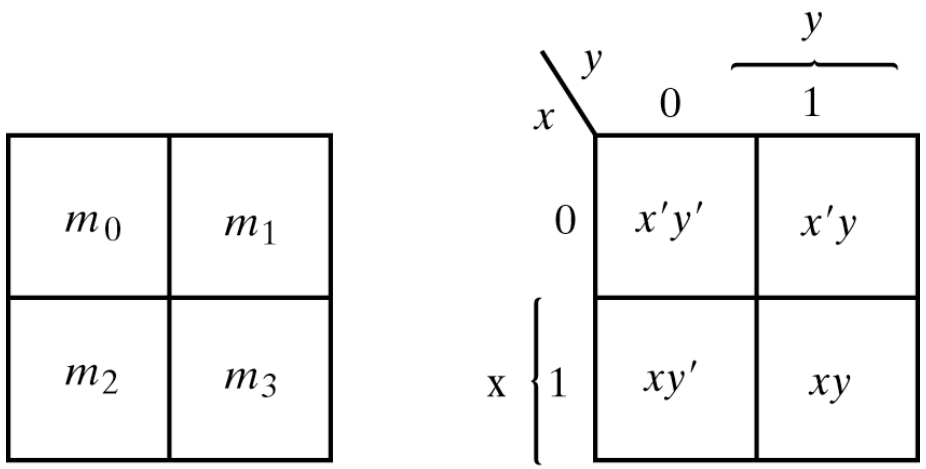
$$F = m_0 + m_1 + m_{15} = A\acute{}B\acute{}C\acute{}D\acute{} + A\acute{}B\acute{}C\acute{}D + ABCD$$

Prosedyre, 2 variable

Setter inn mintermene i diagram

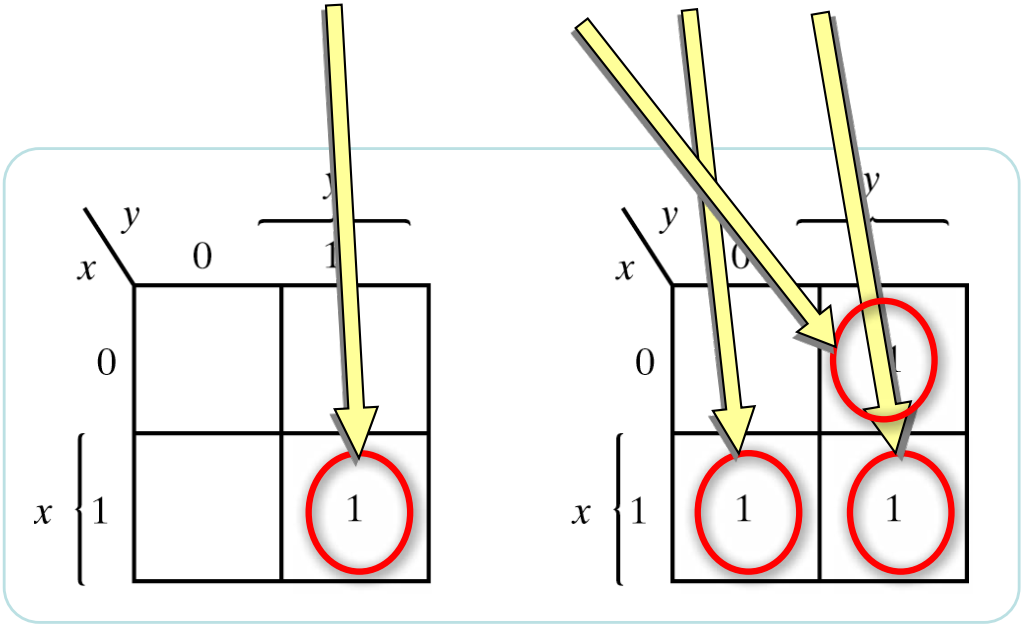
- Eksempel: generell funksjon - 2 variable

$$F = m_0 + m_1 + m_2 + m_3$$



Prosedyre, 2 variable

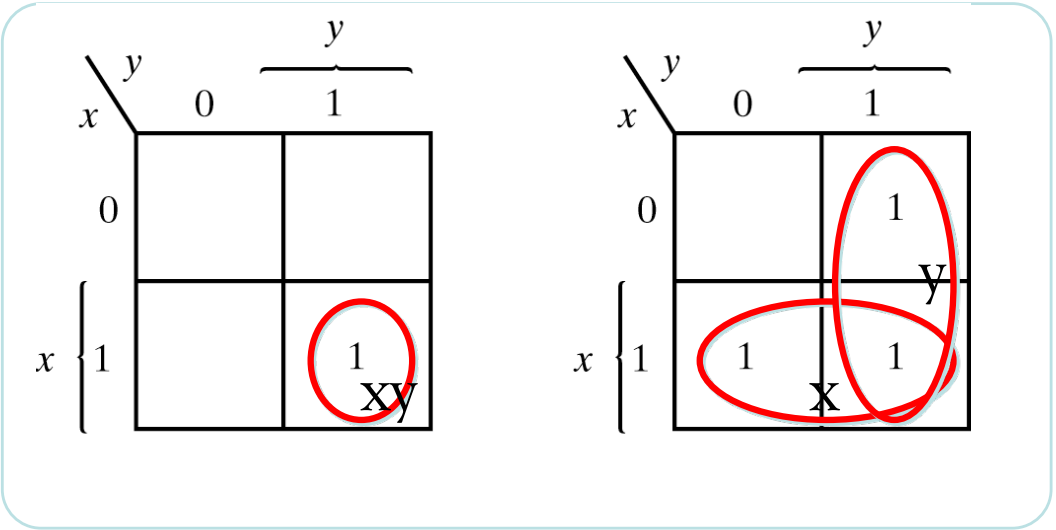
$F_1 = xy$ og $F_2 = x´y + xy´ + xy$



Prosedyre, utlesning

- Grupperer naboruter som inneholder ”1” slik at vi får sammenhengende rektangler, Velg så store grupper som mulig. Antall element må være en potens av 2

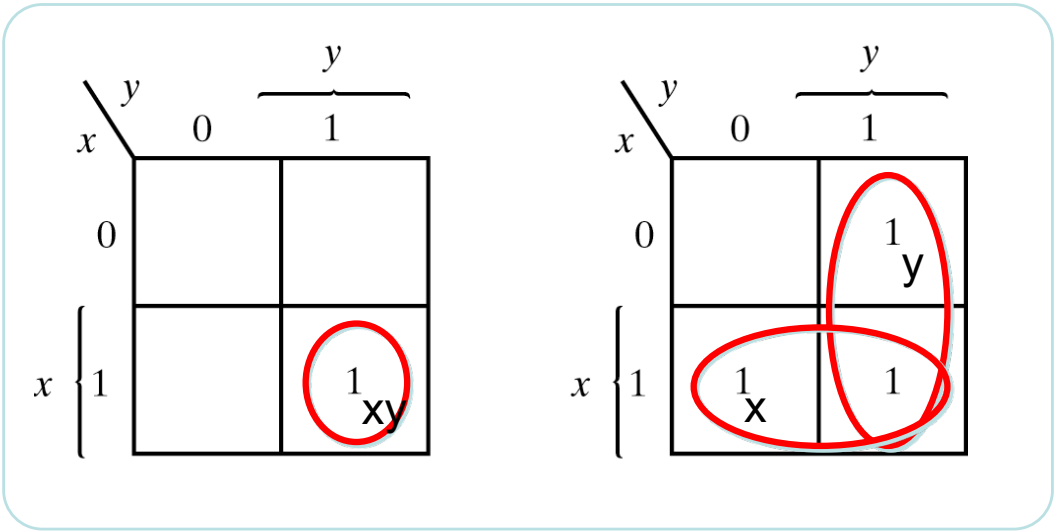
- Representerer gruppene ved de variablene i gruppen som ikke varierer



Prosedyre, utlesning

Funksjonene som diagrammene beskriver er nå gitt av summen av uttrykkene som representerer hver gruppe

$F_1 = xy$ $F_2 = x + y$



Karnaugh - 3 variable

Plassering av mintermer for 3-variable funksjoner:

- Mintermene plasseres slik at kun 1 variabel varierer i mellom hver vannrette/loddrette naborute

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

x \ yz	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

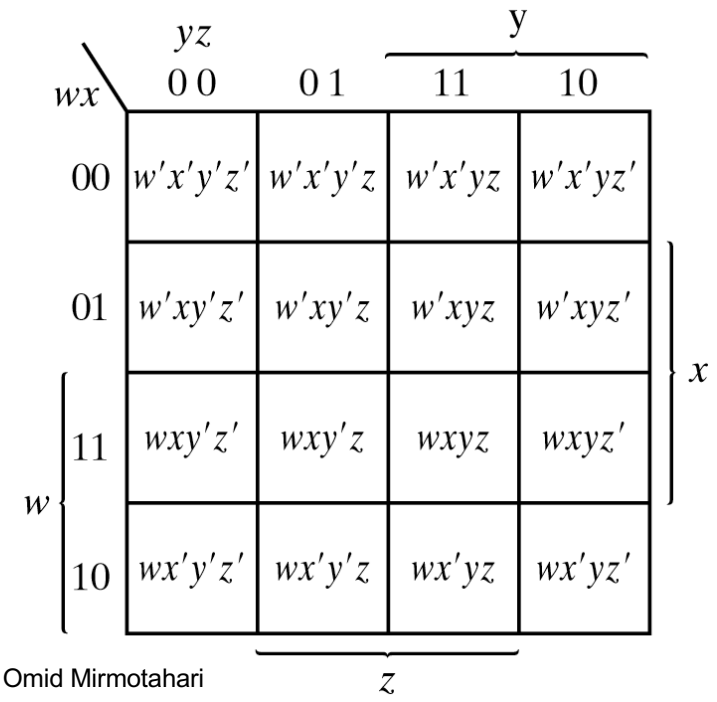
y spans 11 and 10; z spans 01 and 11; x spans the 1 row.

Karnaugh - 4 variable

Plassering av mintermer for 4-variable funksjoner

- Mintermene plasseres slik at kun 1 variabel varierer i mellom hver vannrette/loddrette naborute

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

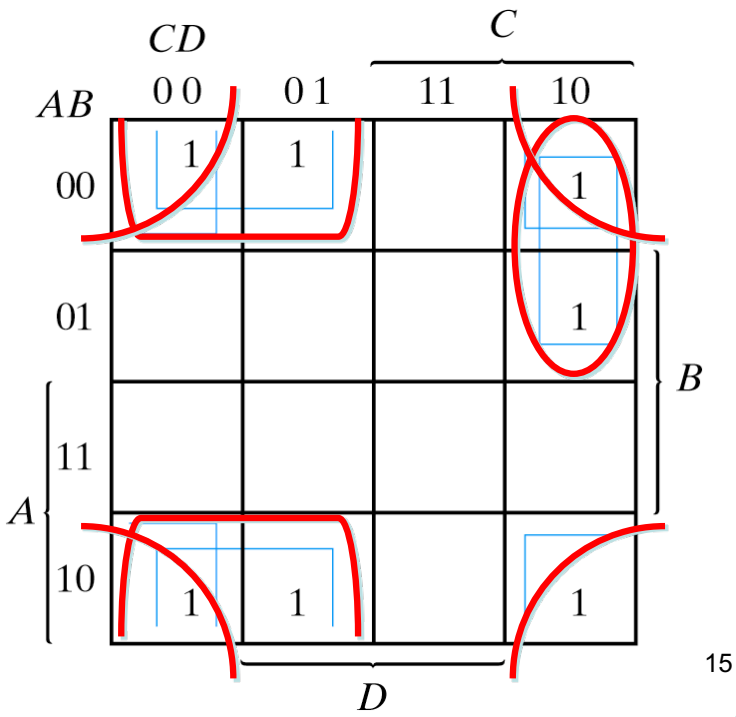


Grupperingsregler for diagram med 2-4 variable

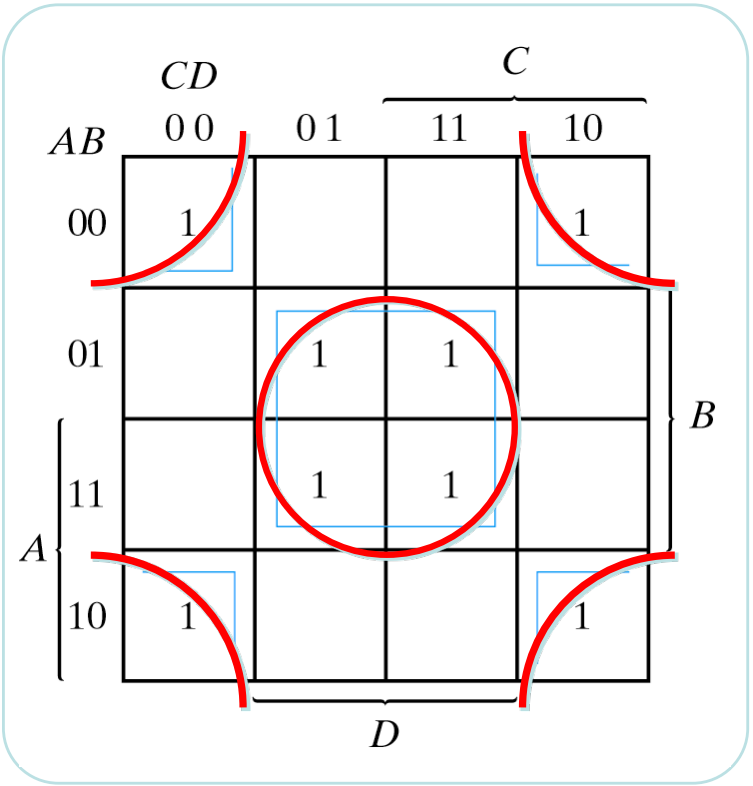
Grupperer naboruter som inneholder ”1” slik at vi får sammenhengende rektangler

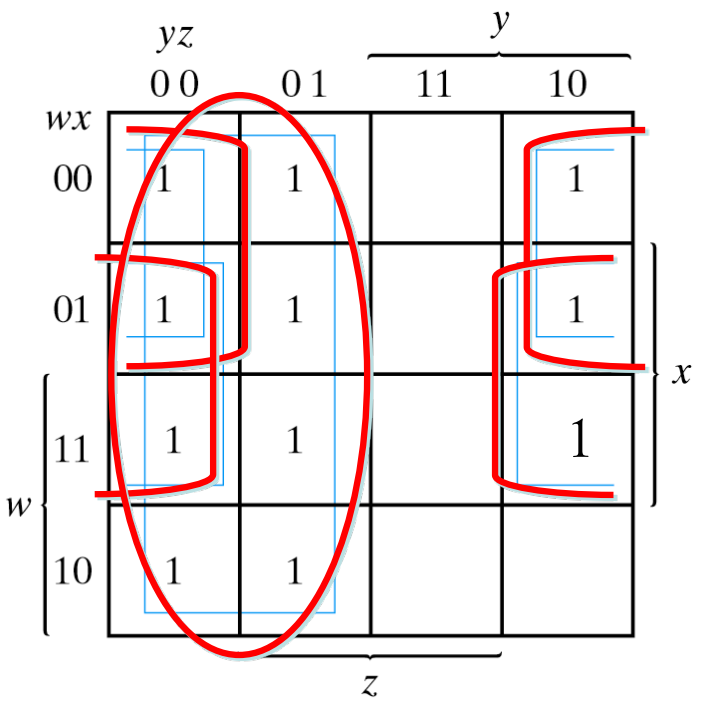
Ytterkantene av diagrammet kan også være naboruter

Eksempel

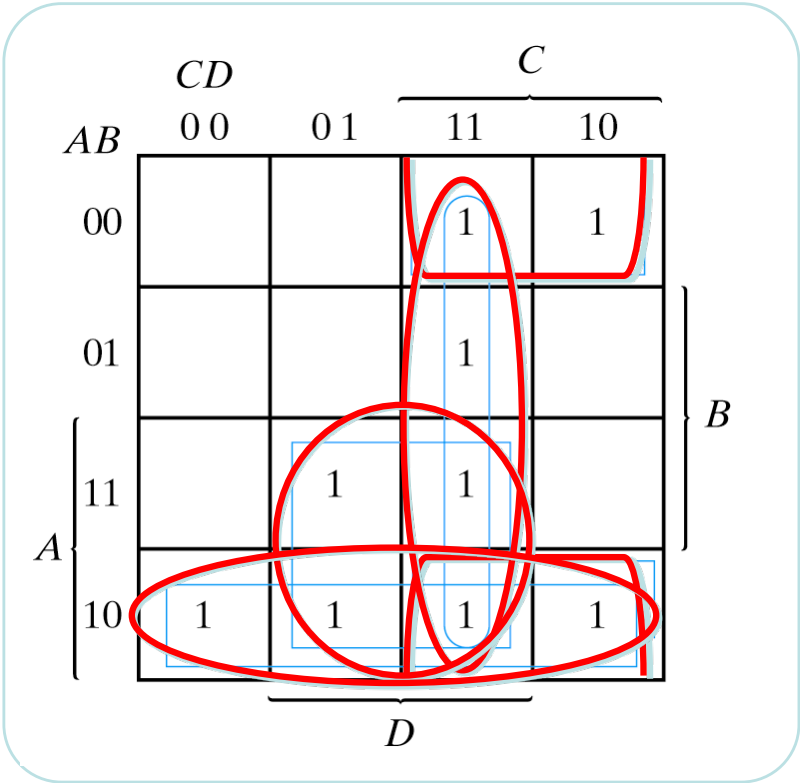


Grupperingsregler for diagram med 2-4 variable





Grupperingsregler for diagram med 2-4 variable



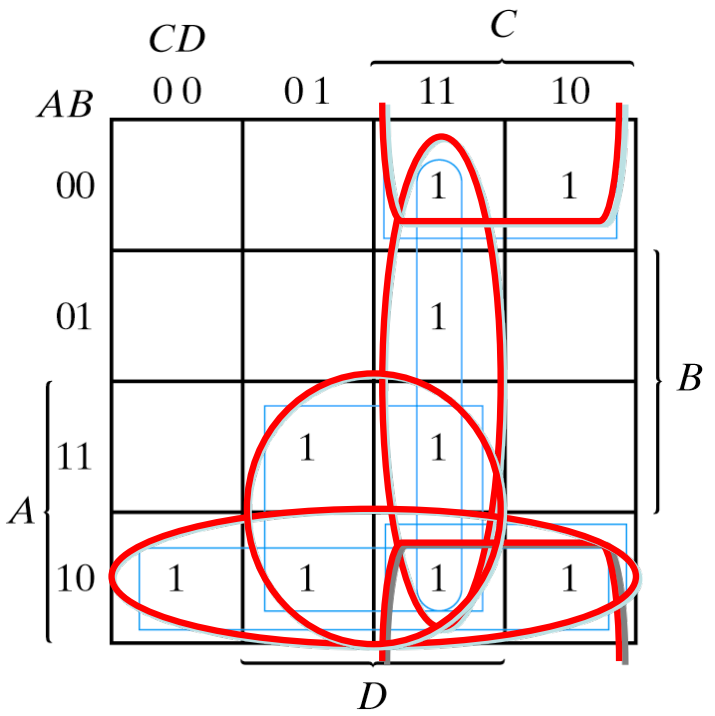
Utlesningssregler for diagram med 2-4 variable

Representerer hver gruppe ved de variablene i gruppen som ikke varierer.

Diagrammets funksjon blir summen av hvert gruppeledd:

Eksempel

$$F = AD + CD + B´C + AB´$$

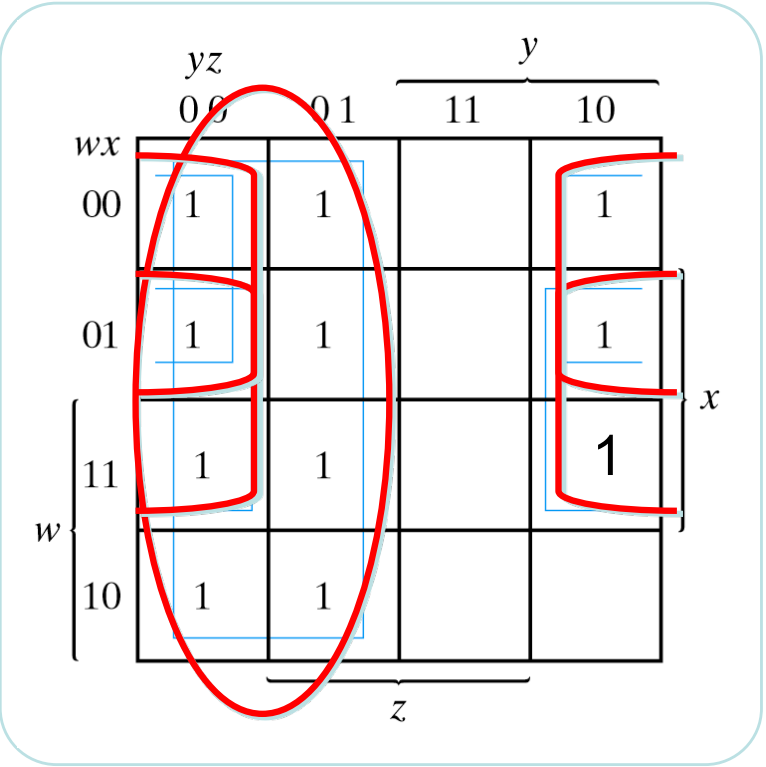


Utlesningsregler for diagram med 2-4 variable

Eksempel:

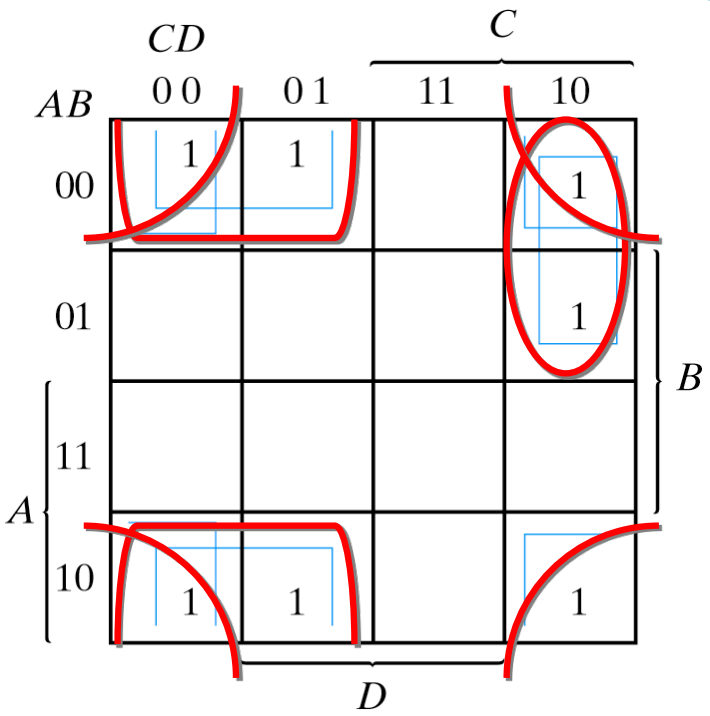
$$F = y' + w'z' + xz'$$

Merker oss at jo større ruter desto enklere uttrykk



Utlesningssregler for diagram med 2-4 variable

F = B´D´+ C´B´+ A´CD´



Utlesning av ”0”ere

Ved å lese ut de tomme rutene (”0”erne) fra diagrammet får man F´

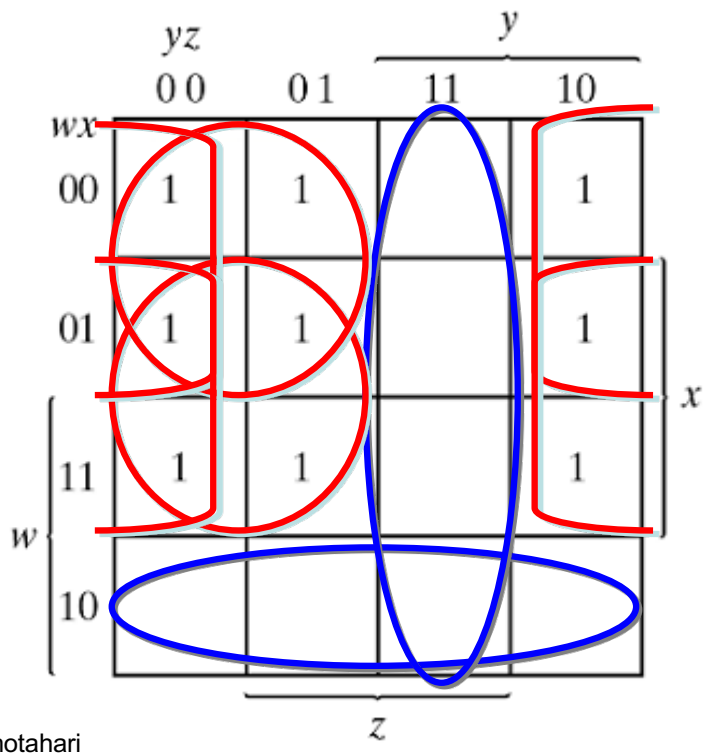
Dette kan noen ganger gi en enklere funksjon, eksempel:

$F' = yz + wx'$

$F = (yz + wx')'$

Hadde vi lest ut ”1”ere ville vi fått

$F = xy' + w'y' + w'z' + xz'$



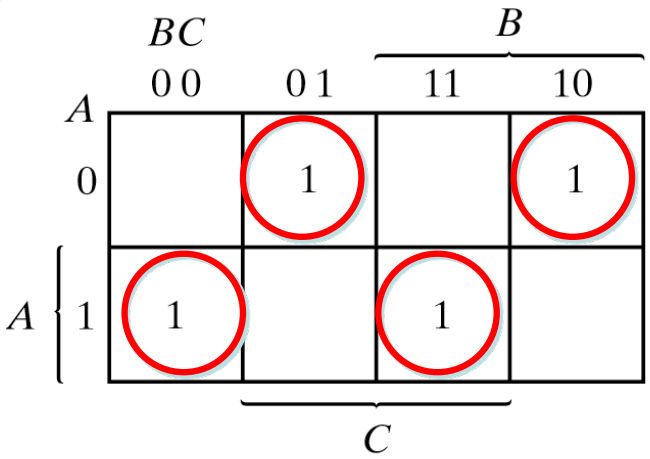
Spesialteknikker

XOR

XOR funksjonen dekker maksimalt ”uheldige” ”1”er plasseringer i diagrammet

Har man XOR porter til rådighet bruker man disse

I dette eksemplet kan
1 stk. 3-inputs XOR realisere F



(a) Odd function

$F = A \oplus B \oplus C$

XOR

XOR representerer den odde funksjonen, dvs de kombinasjoner av inngangen hvor det er odde antall 1´ere

Man kan også bruke Karnaugh diagram til å forenkle enkelte XOR funksjoner.

XNOR er den inverterte funksjonen av XOR

XNOR

ab \ cd				
	1		1	
		1		1
	1		1	
		1		1

XOR

ab \ cd				
		1		1
	1		1	
		1		1
	1		1	

ab \ cd				
	1		1	
	1		1	
		1		1
		1		1

ab \ cd				
			1	1
	1	1		
			1	1
	1	1		

ab \ cd				
	1	1		
	1	1		
			1	1
			1	1

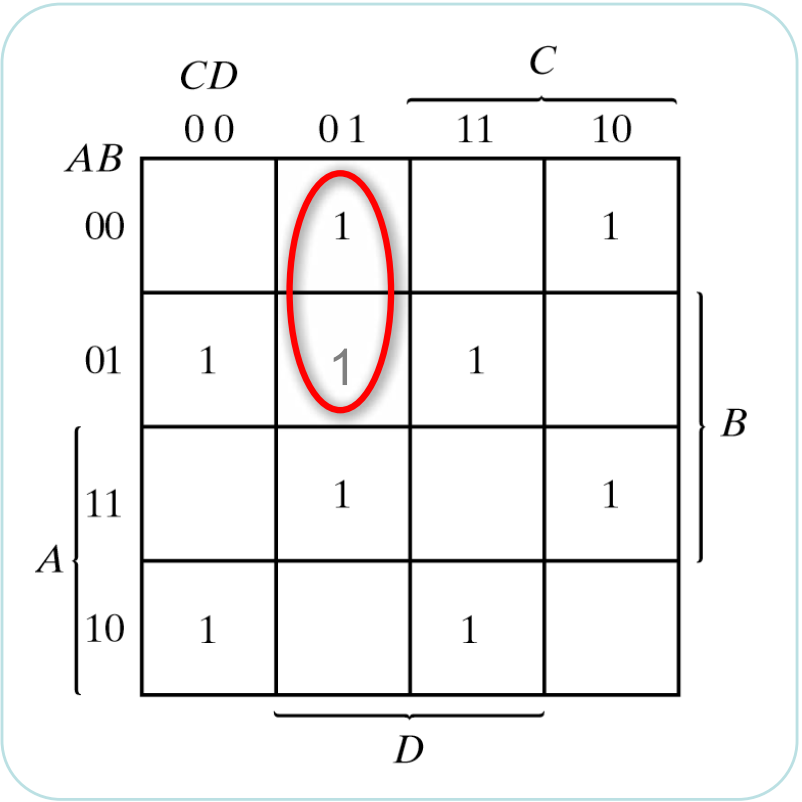
ab \ cd				
			1	1
			1	1
	1	1		
	1	1		

XOR

XOR funksjonen kan kombineres med andre ledd

Eksempel:

$F = A \oplus B \oplus C \oplus D + A´C´D$



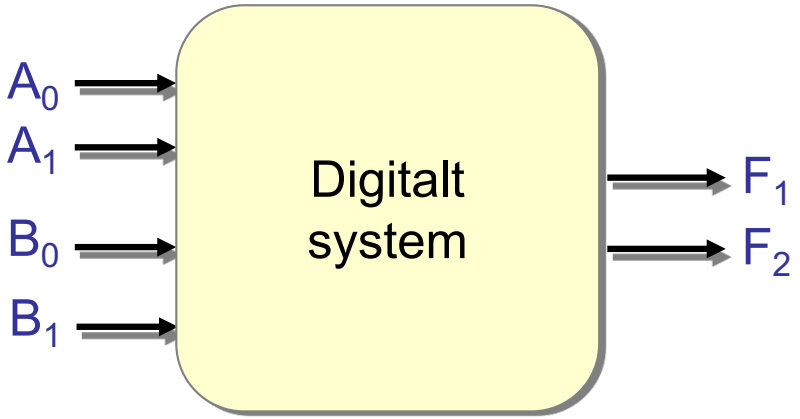
Designeksempel

Vi ønsker å designe en krets som kan sammenligne to tall A og B. Hvert tall er representert ved to bit.

Kretsen skal finne A>B samt A=B

Vi har dermed $2 \cdot 2 = 4$ innganger, og 2 utganger

Setter navn på utgangene:
F_1 for A>B og F_2 for A=B



Designeksempel

Vi trenger en oversikt over alle mulige inngangs/utgangs kombinasjoner, derfor:

- Setter opp en sannhetstabell for hver utgang (slår sammen til en dobbel tabell)
- Leser ut mintermer

$$F_1 = A_1´A_0B_1´B_0´ + A_1A_0´B_1´B_0´ + A_1A_0´B_1´B_0 + A_1A_0B_1´B_0´ + A_1A_0B_1´B_0 + A_1A_0B_1B_0´$$

$$F_2 = A_1´A_0´B_1´B_0´ + A_1´A_0B_1´B_0 + A_1A_0´B_1B_0´ + A_1A_0B_1B_0$$

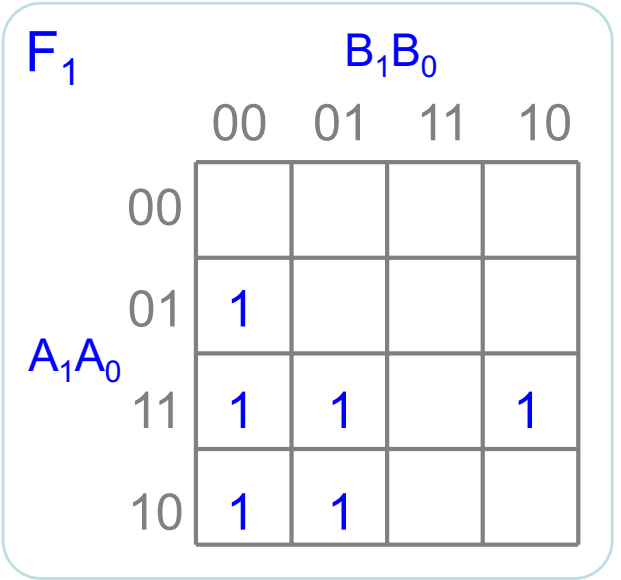
Innganger				Utganger	
A_1	A_0	B_1	B_0	F_1	F_2
0	0	0	0	0	1
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	0	0
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

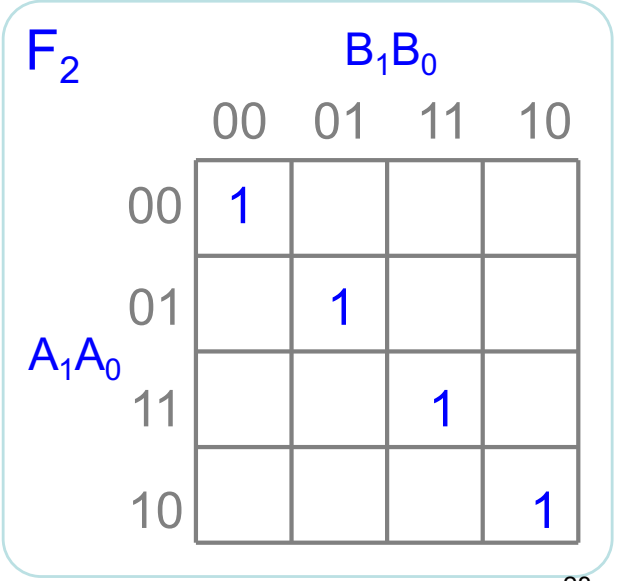
Forenkler uttrykket for å spare porter

Setter inn i Karnaughdiagram

$$F_1 = A_1´A_0B_1´B_0´ + A_1A_0´B_1´B_0´ + A_1A_0´B_1´B_0 + A_1A_0B_1´B_0´ + A_1A_0B_1´B_0 + A_1A_0B_1B_0´$$

$$F_2 = A_1´A_0´B_1´B_0´ + A_1´A_0B_1´B_0 + A_1A_0´B_1B_0´ + A_1A_0B_1B_0$$





Leser ut av diagrammene

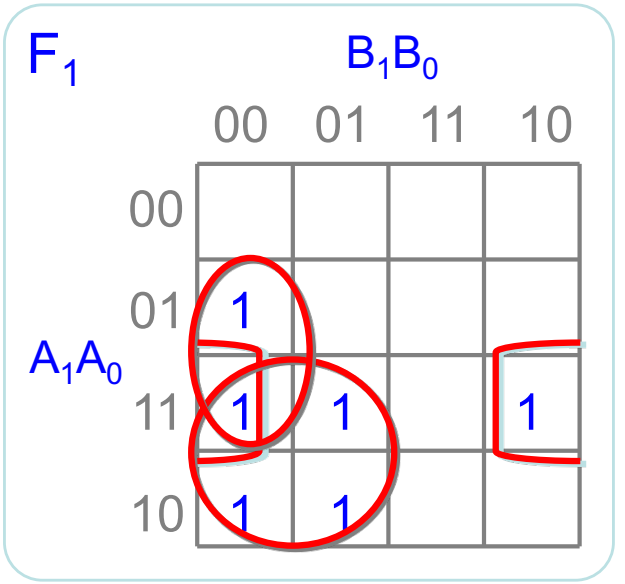
$$F_1 = A_1B_1' + A_0B_1'B_0' + A_1A_0B_0'$$

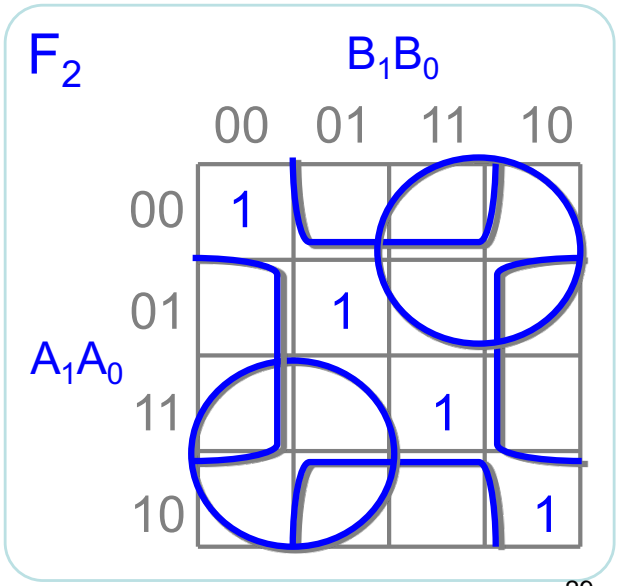
F_2: Ingen forenkling mulig ved utlesning av ”1”ere, leser derfor ut ”0”ere

$$F_2' = A_1B_1' + A_0B_0' + A_0'B_0 + A_1'B_1$$

Inverterer begge sider

$$F_2 = (A_1B_1' + A_0B_0' + A_0'B_0 + A_1'B_1)'$$



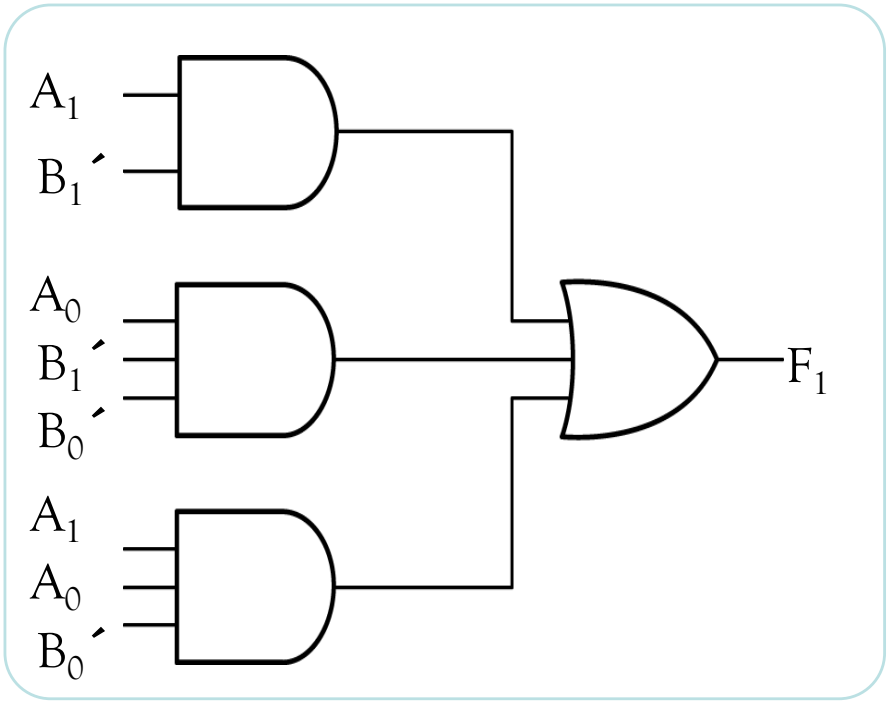


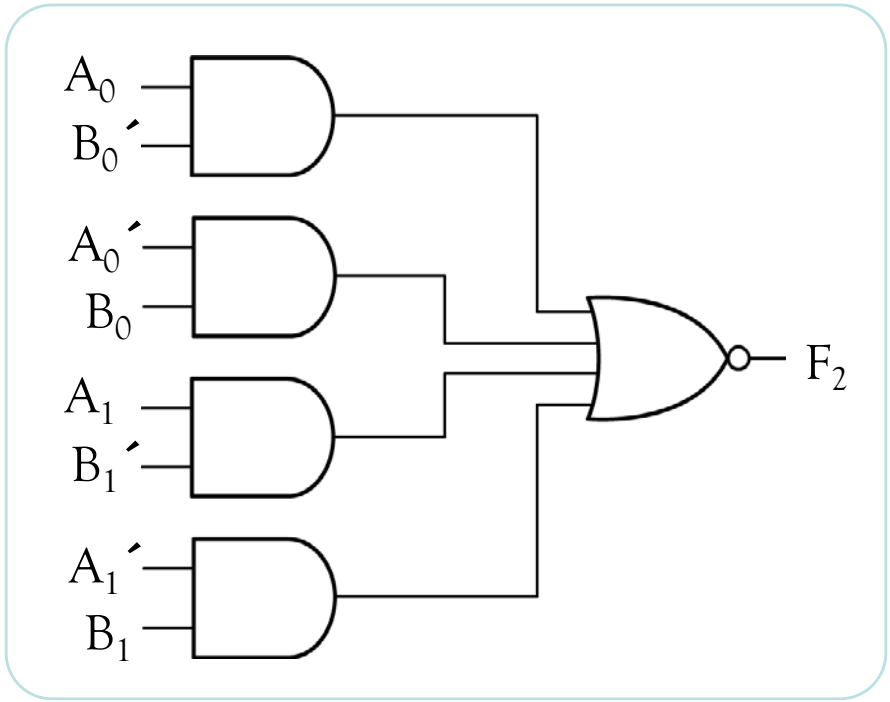
Implementerer uttrykkene

$$F_1 = A_1B_1' + A_0B_1'B_0' + A_1A_0B_0'$$

$$F_2 = (A_0B_0' + A_0'B_0 + A_1B_1' + A_1'B_1)'$$

(Hva med XOR?)





Black box tankesett!

- Styring av heis

Designeksempel 1

- Styring av heis

- Inngangssignaler fra sensor
  - K = Knapp trykket inn: 0/1
  - V = Overvekt: 0/1
  - D = Dør lukket: 0/1

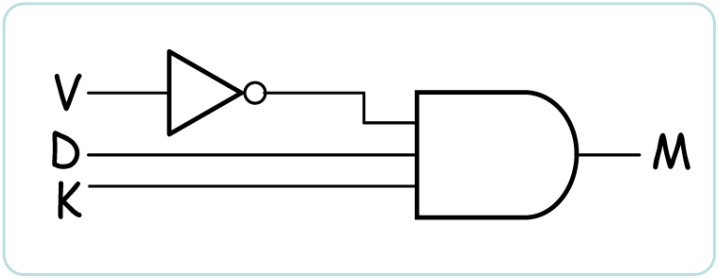
- Utgangssignal til aktuator:
  - M = Motor på: 0/1

Sannhetstabell for designet

M = KV'D

K	V	D	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

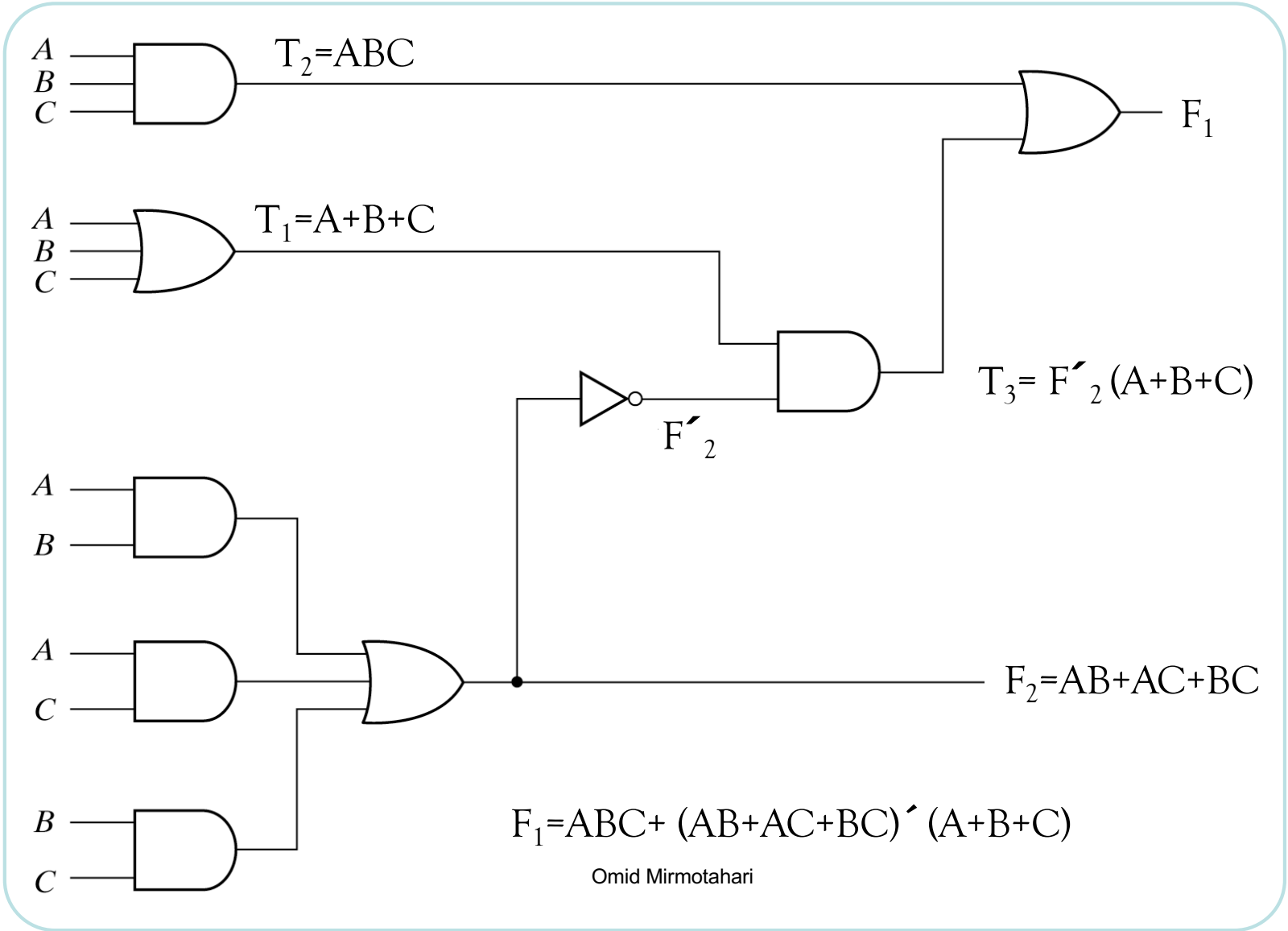
Implementasjon på portnivå



Generell analyseprosedyre for digitale kretser

1) Sett funksjonsnavn på ledningene
2) Finn funksjonene
3) Kombiner funksjonsuttrykkene

Eksempel


A
B
C
T2=ABC
F1
A
B
C
T1=A+B+C
T3= F´2 (A+B+C)
F´2
A
B
A
C
F2=AB+AC+BC
B
C
F1=ABC+ (AB+AC+BC)´(A+B+C)

Binær adder

En av de mest brukte digitale kretser

Vanlige anvendelser:

Mikroprosessor ALU / Xbox / mikserbord / digitalt kommunikasjonsutstyr / AD-DA omformere osv...

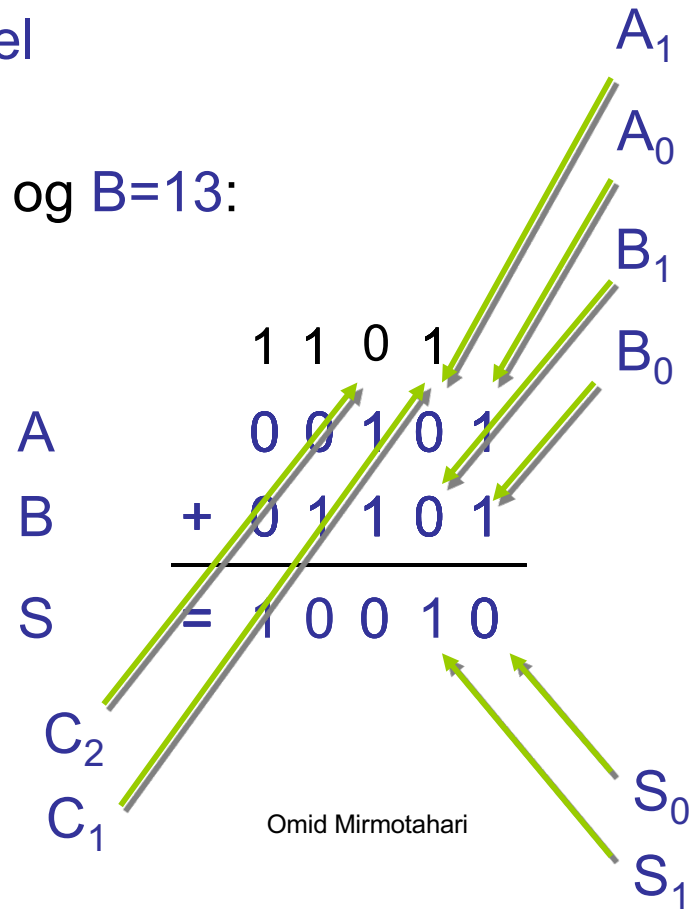
- Basis for addisjon / subtraksjon / multiplikasjon / divisjon og mange andre matematiske operasjoner
- All form for filtrering / signalbehandling

Binær adder

Ønsker å designe en generell binær adder

Funksjonelt eksempel

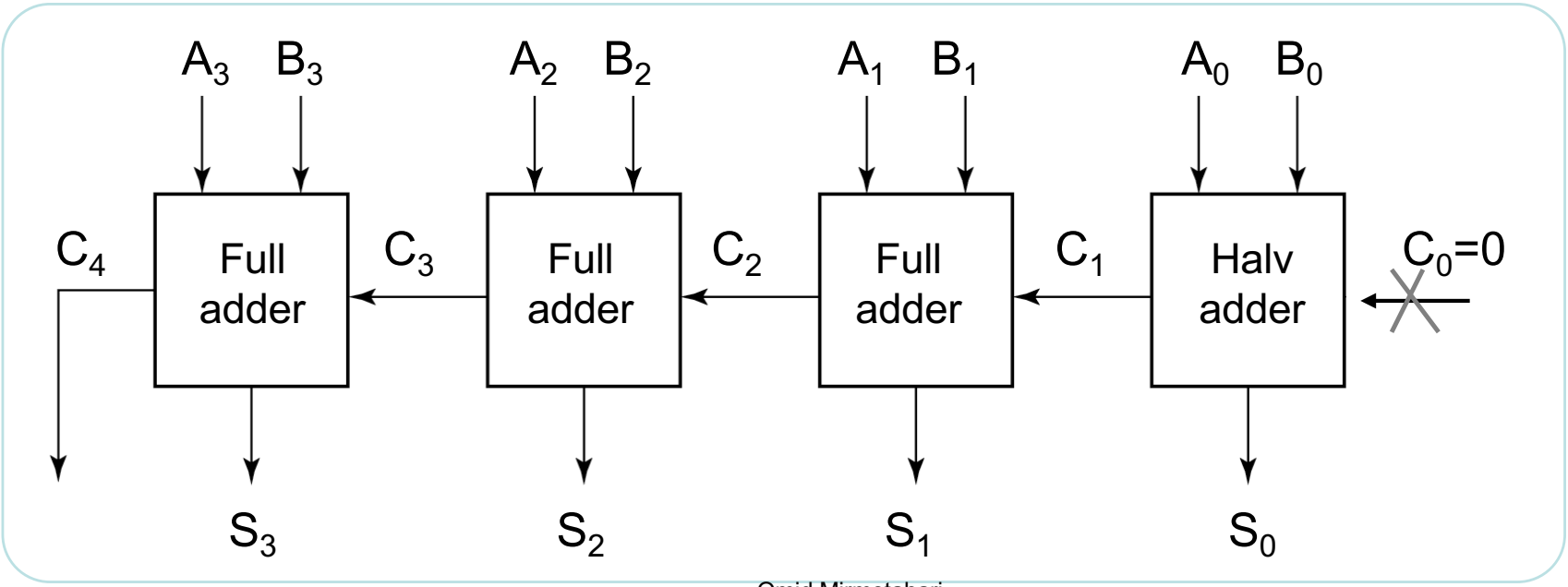
Adder to tall A=5 og B=13:



Et adder system

Systemelementer:

- Halvadder: Tar ikke mente inn
- Fulladder: Tar mente inn



Halvadder (ingen mente inn)

Adderer sammen de to minst signifikante bittene A_0 og B_0.

Elementet har 2 innganger og 2 utganger

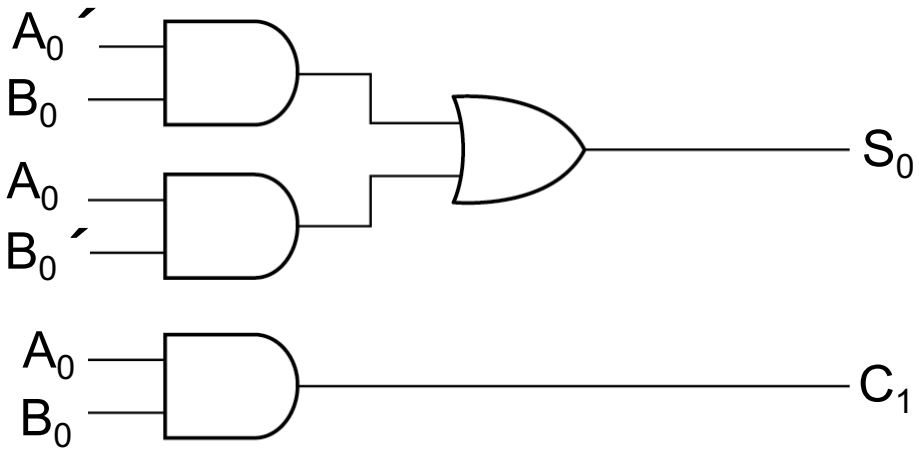
$S_0 = A_0 \acute{} B_0 + A_0 B_0 \acute{} = A_0 \oplus B_0$

$C_1 = A_0 B_0$

Sannhetstabell

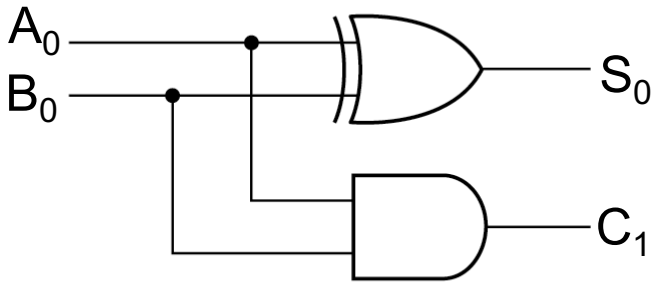
A_0	B_0	S_0	C_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Halvadder implementasjon



$S_0 = A_0´B_0 + A_0B_0´$

$C_1 = A_0B_0$



$S_0 = A_0 \oplus B_0$

$C_1 = A_0B_0$

Fulladder (mente inn)

Adderer sammen bit A_n , B_n med evt. mente inn

Elementet har 3 innganger og 2 utganger

Sannhetstabell

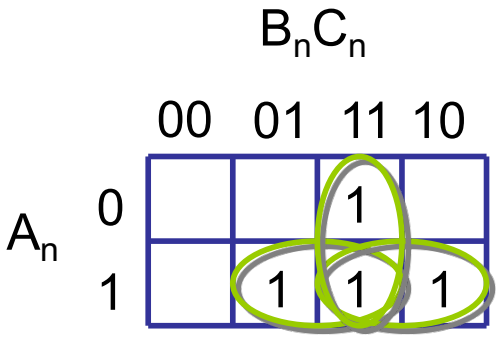
A_n	B_n	C_n	S_n	C_{n+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S_n = A_n \oplus B_n \oplus C_n$ (oddefunksjon)

$C_{n+1} = A_n´B_nC_n + A_nB_n´C_n + A_nB_nC_n´ + A_nB_nC_n$

Forenkling

Forenkler C_{n+1} ved Karnaughdiagram



$$C_{n+1} = A_n´B_nC_n + A_nB_n´C_n + A_nB_nC_n´ + A_nB_nC_n$$

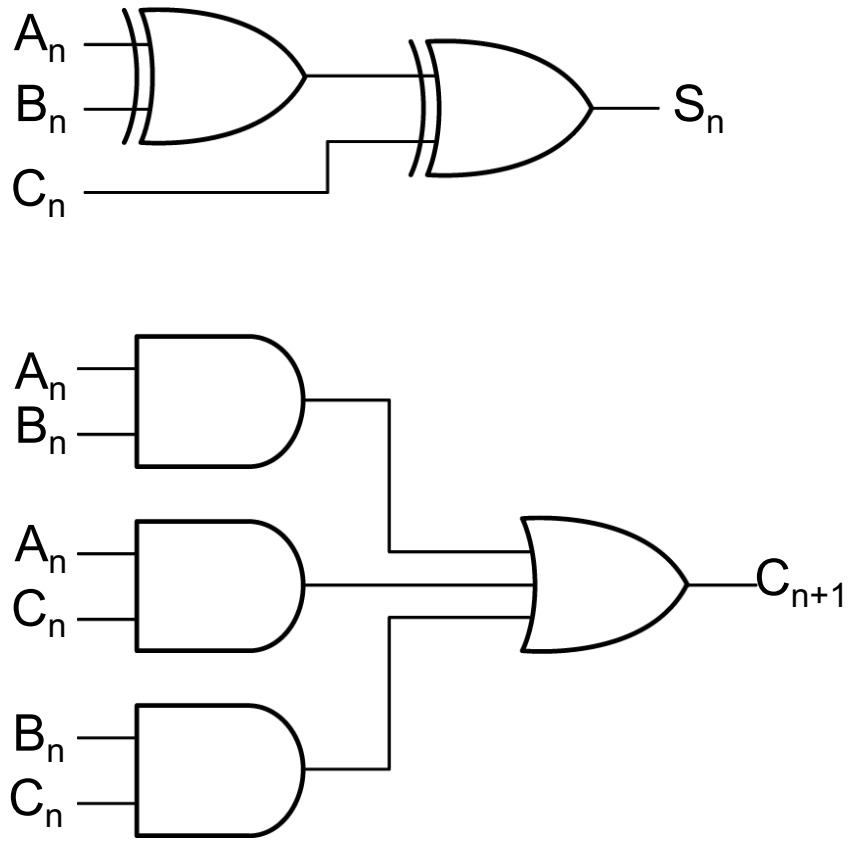
$$C_{n+1} = A_nB_n + A_nC_n + B_nC_n$$

Implementasjon I

Rett fram implementasjon

$$S_n = A_n \oplus B_n \oplus C_n$$

$$C_{n+1} = A_nB_n + A_nC_n + B_nC_n$$

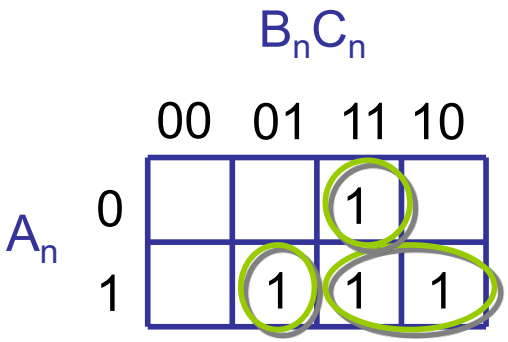


Implementasjon II

Forenklet implementasjon av C_{n+1} basert på gjenbruk av porter fra S_n

$$S_n = (A_n \oplus B_n) \oplus C_n$$

Leser ut C_{n+1} fra karnaughdiagram på nytt



$$C_{n+1} = A_nB_n + A_nB_n´C_n + A_n´B_nC_n$$

$$C_{n+1} = A_nB_n + (A_nB_n´ + A_n´B_n)\,C_n$$

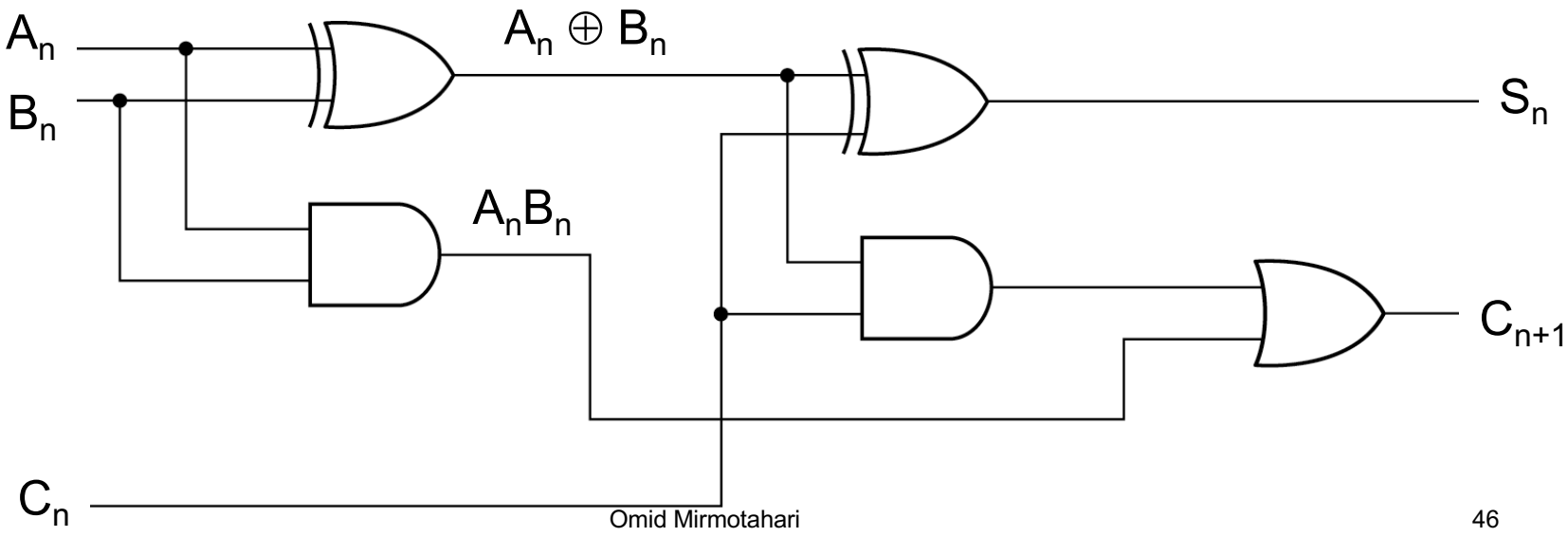
$$C_{n+1} = A_nB_n + (A_n \oplus B_n)\,C_n$$

Implementasjon II

Vanlig implementasjon av en-bits fulladder

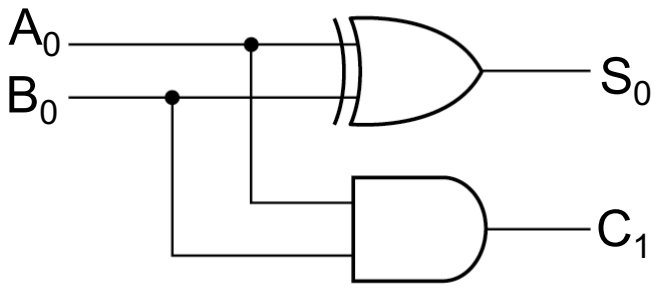
$$S_n = (A_n \oplus B_n) \oplus C_n$$

$$C_{n+1} = A_nB_n + (A_n \oplus B_n)\, C_n$$

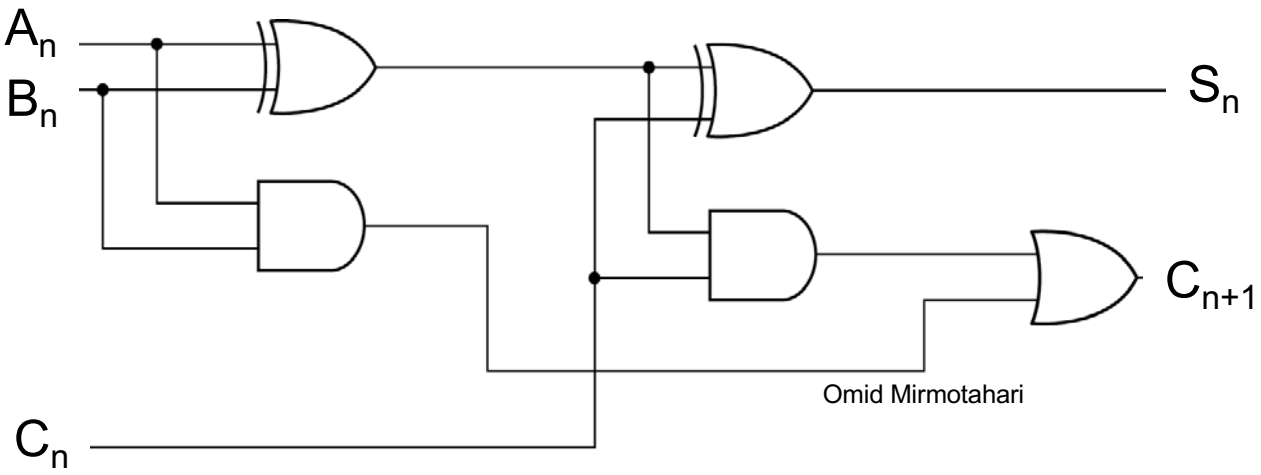


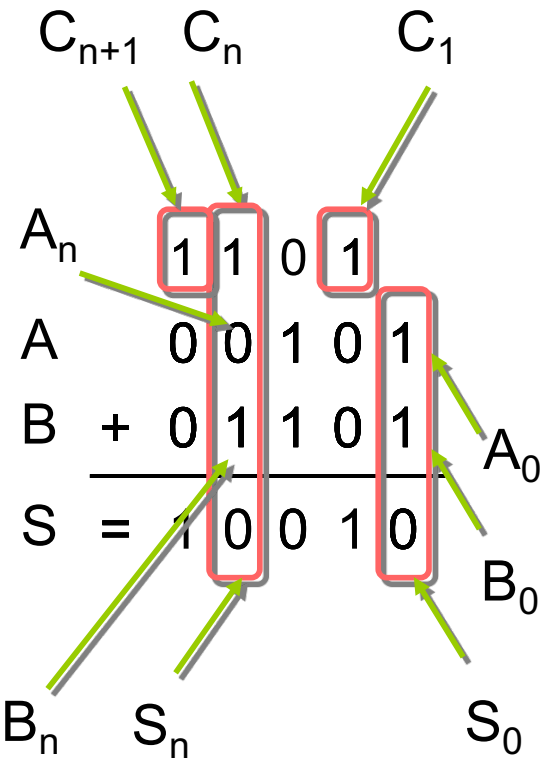
Binær adder

Halvadder (ikke mente inn)



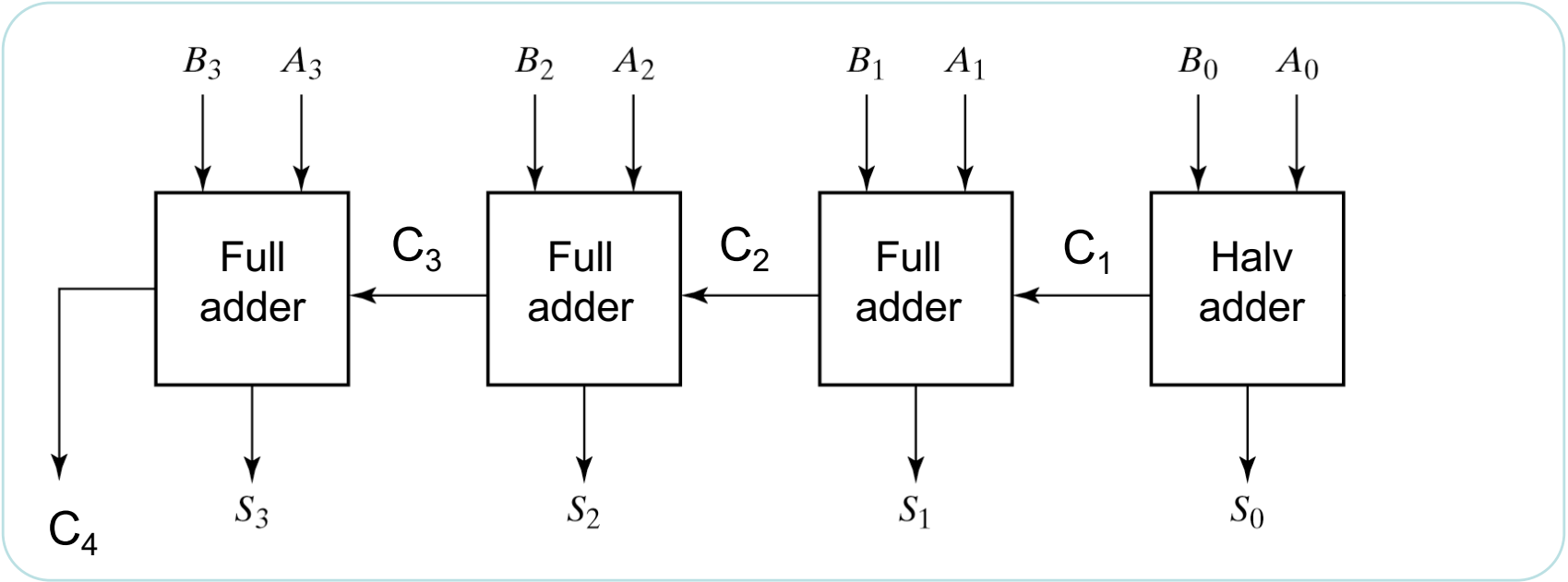
Fulladder (evt. mente inn)





Menteforplantning

4-bits binær adder

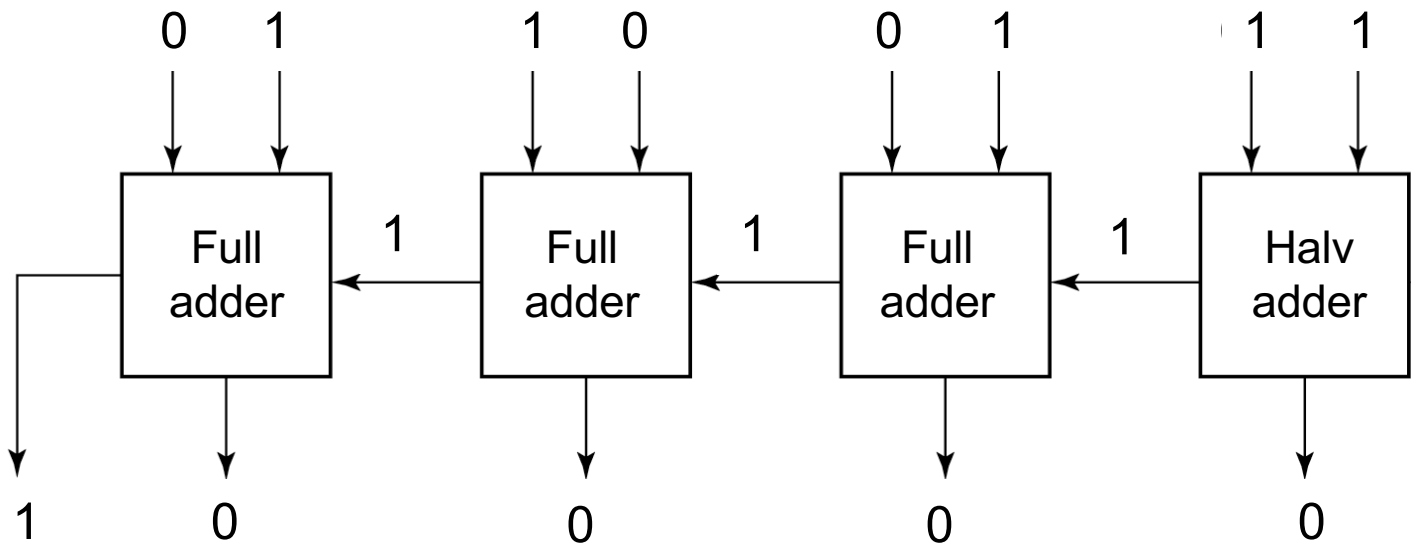


Menteforplantning

Portforsinkelse gir menteforplantning (rippeladder)

Eksempel

Adderer 0101 og 1011



Komparator

Komparator – sammenligner to tall A og B

- 3 utganger: A=B, A>B og A<B

Eksempel: 4-bits komparator

Utgang A=B

Slår til hvis $A_0=B_0$ og $A_1=B_1$ og $A_2=B_2$ og $A_3=B_3$

Kan skrives: $(A_0 \oplus B_0)´(A_1 \oplus B_1)´(A_2 \oplus B_2)´(A_3 \oplus B_3)´$

Komparator - eksempel

Utgang $A>B$ slår til hvis:

($A_3>B_3$) eller

($A_2>B_2$ og $A_3=B_3$) eller

($A_1>B_1$ og $A_2=B_2$ og $A_3=B_3$) eller

($A_0>B_0$ og $A_1=B_1$ og $A_2=B_2$ og $A_3=B_3$)

Kan skrives:

$(A_3B_3´) + (A_2B_2´)\,(A_3 \oplus B_3)´ + (A_1B_1´)\,(A_2 \oplus B_2)´(A_3 \oplus B_3)´ +$

$(A_0B_0´)(A_1 \oplus B_1)´(A_2 \oplus B_2)´(A_3 \oplus B_3)´$

Komparator - eksempel

Utgang $A<B$ slår til hvis:

$(A_3<B_3)$ eller

$(A_2<B_2$ og $A_3=B_3)$ eller

$(A_1<B_1$ og $A_2=B_2$ og $A_3=B_3)$ eller

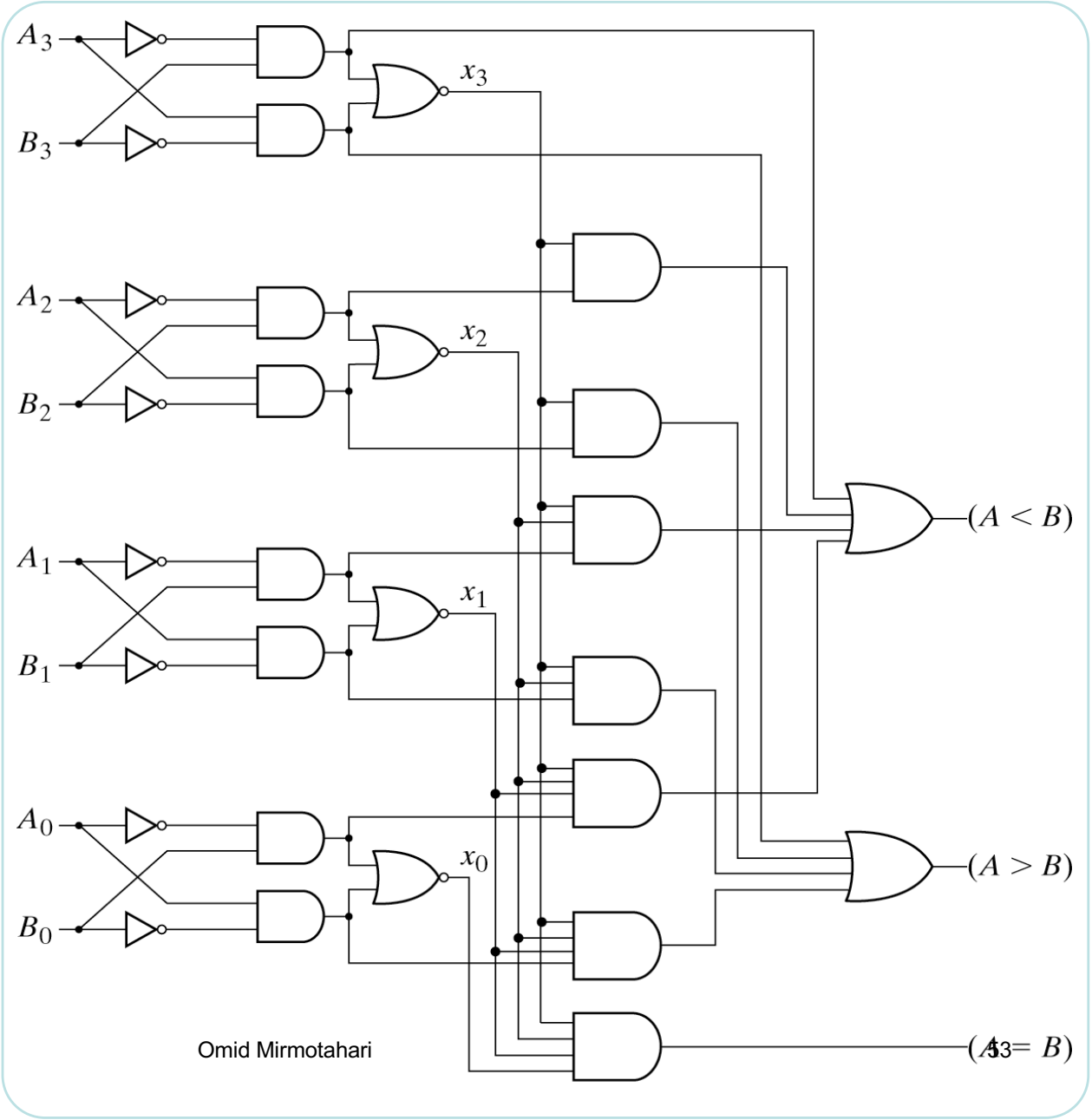
$(A_0<B_0$ og $A_1=B_1$ og $A_2=B_2$ og $A_3=B_3)$

Kan skrives:

$$(A_3 \acute{} B_3) + (A_2 \acute{} B_2)\,(A_3 \oplus B_3) \acute{} + (A_1 \acute{} B_1)\,(A_2 \oplus B_2) \acute{} (A_3 \oplus B_3) \acute{} +$$

$$(A_0 \acute{} B_0)(A_1 \oplus B_1) \acute{} (A_2 \oplus B_2) \acute{} (A_3 \oplus B_3) \acute{}$$

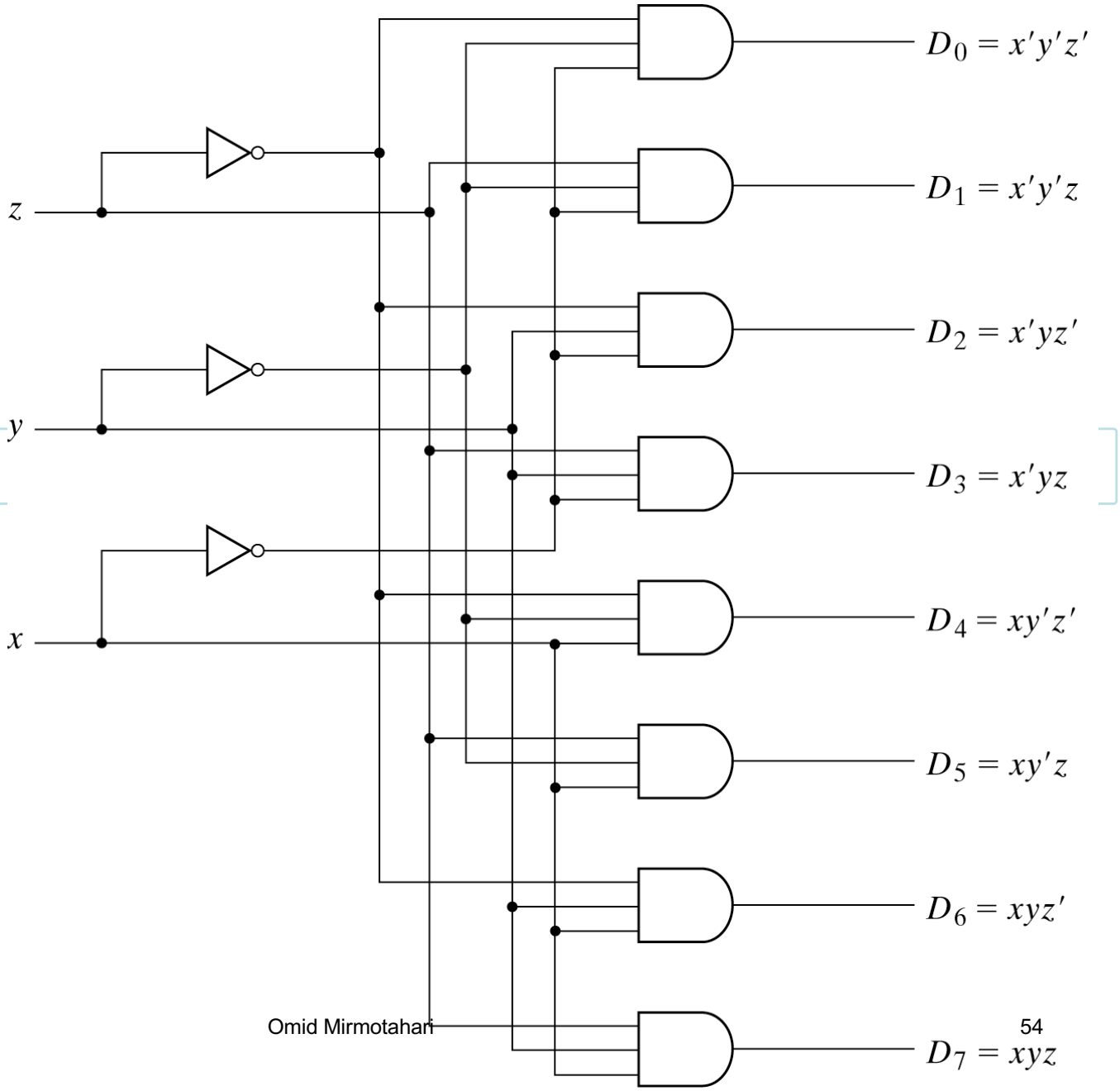
Komparator - eksempel



Dekoder

Dekoder – tar inn et binært ord, gir ut alle mintermer

Eksempel: 3bit inn / 8bit ut



Dekoder - sannhetstabell

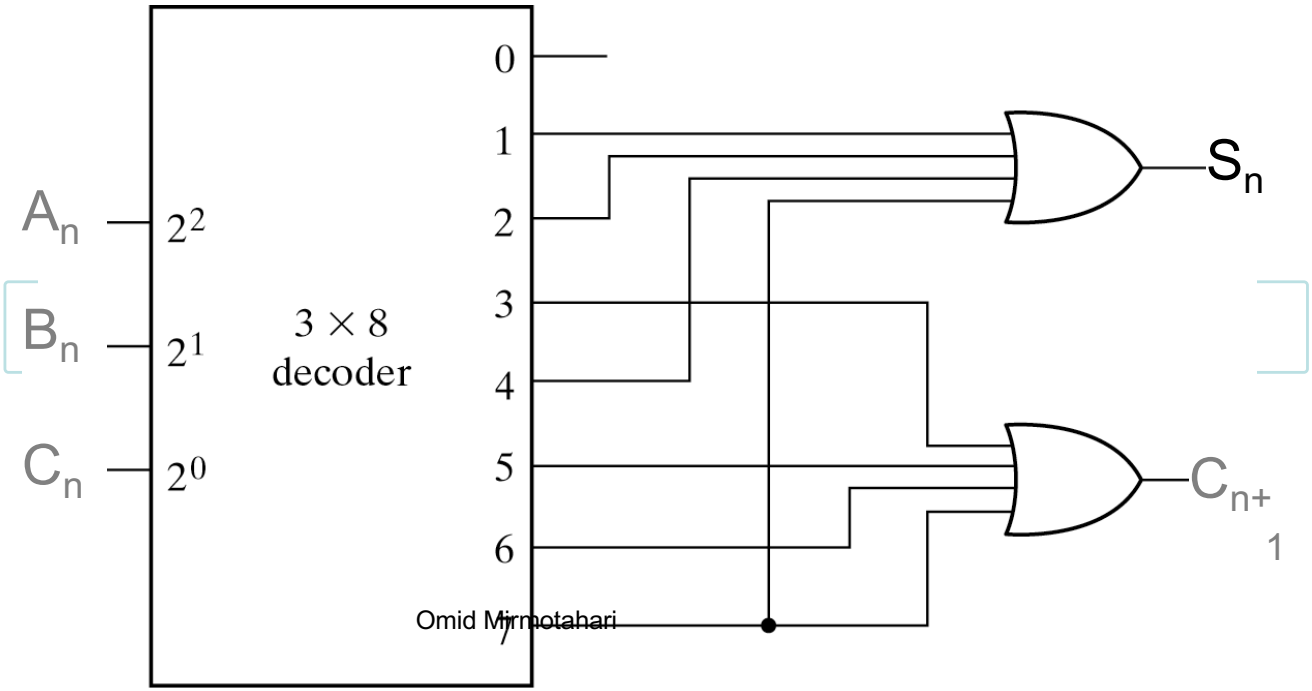
Eksempel: 3bit inn

Innganger			Utganger							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Dekoder – generering av logiske funksjoner

Dekoder - elektrisk sannhetstabell. Kan generere generelle logiske funksjoner direkte fra mintermene på utgangen

Eksempel:
Fulladder



Enkoder

Enkoder – motsatt av dekoder

Eksempel: 8x3 enkoder

Innganger								Utganger		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$x = D_4 + D_5 + D_6 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$z = D_1 + D_3 + D_5 + D_7$$

Antar at det ikke eksisterer andre inngangskombinasjoner

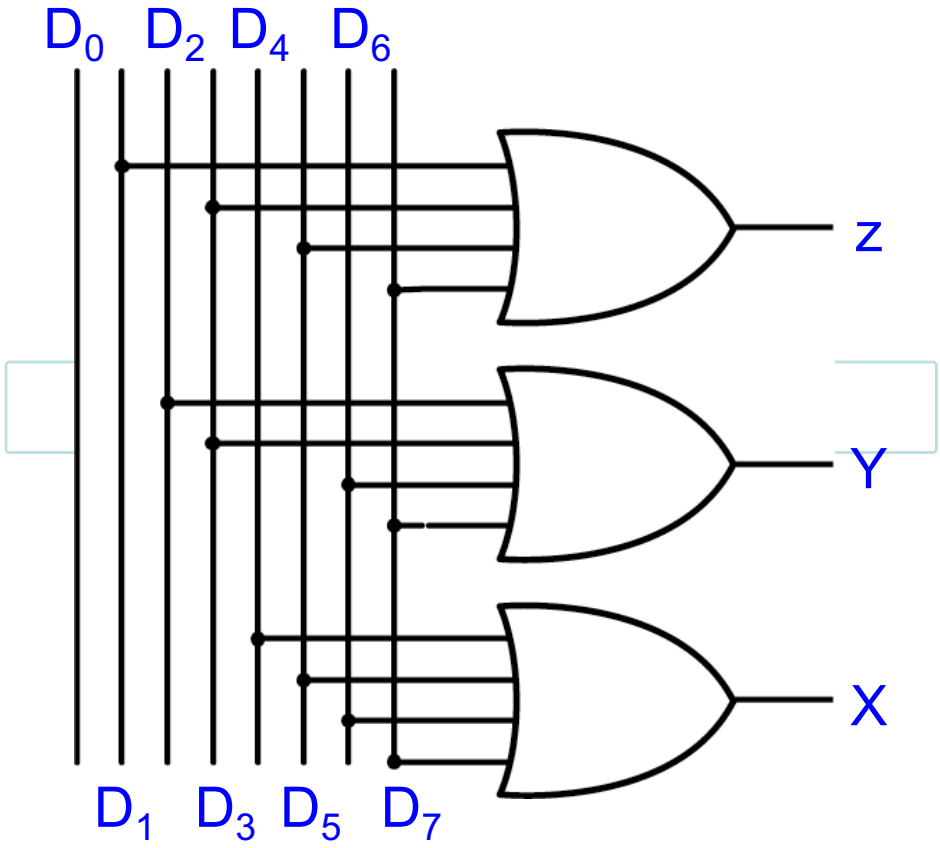
Enkoder

Eksempel

$x = D_4 + D_5 + D_6 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

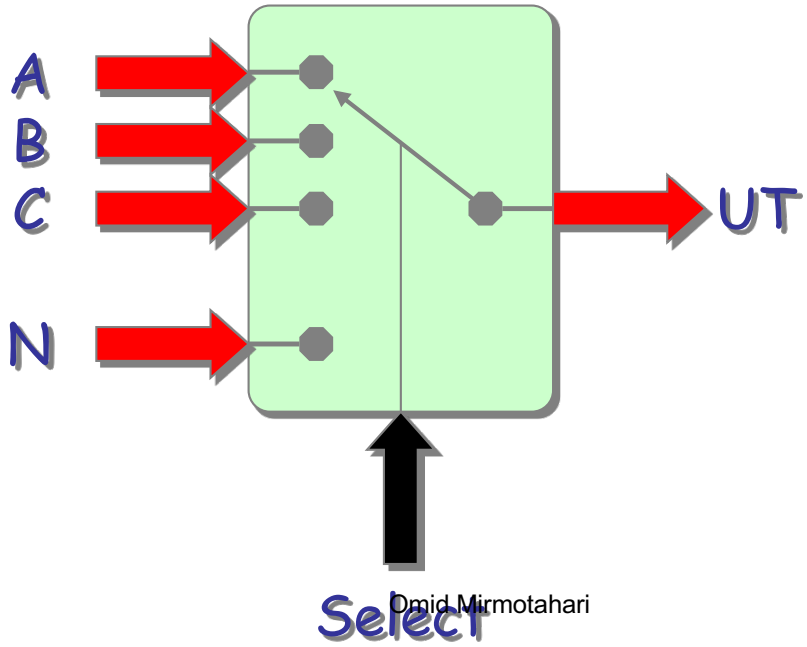
$z = D_1 + D_3 + D_5 + D_7$



Multiplekser

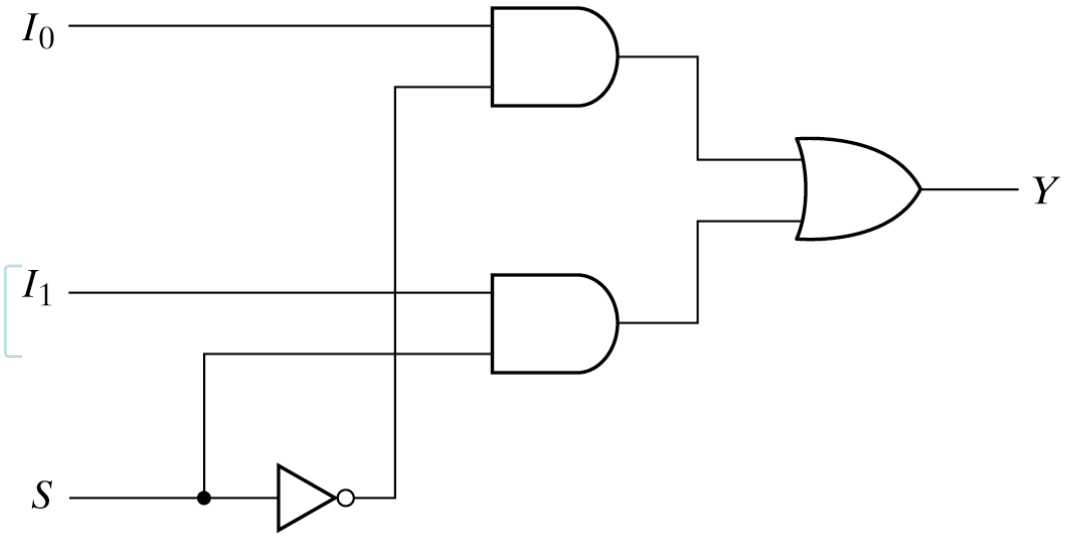
Multiplekser (MUX) – velger hvilke innganger som slippes ut

Hver inngang kan bestå av ett eller flere bit

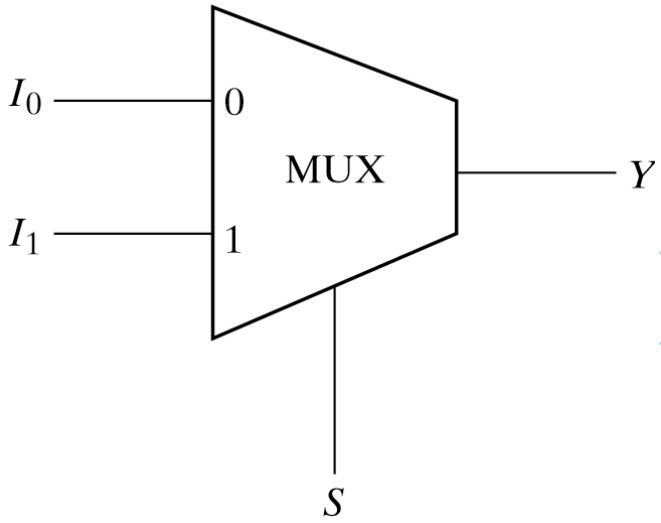


MUX

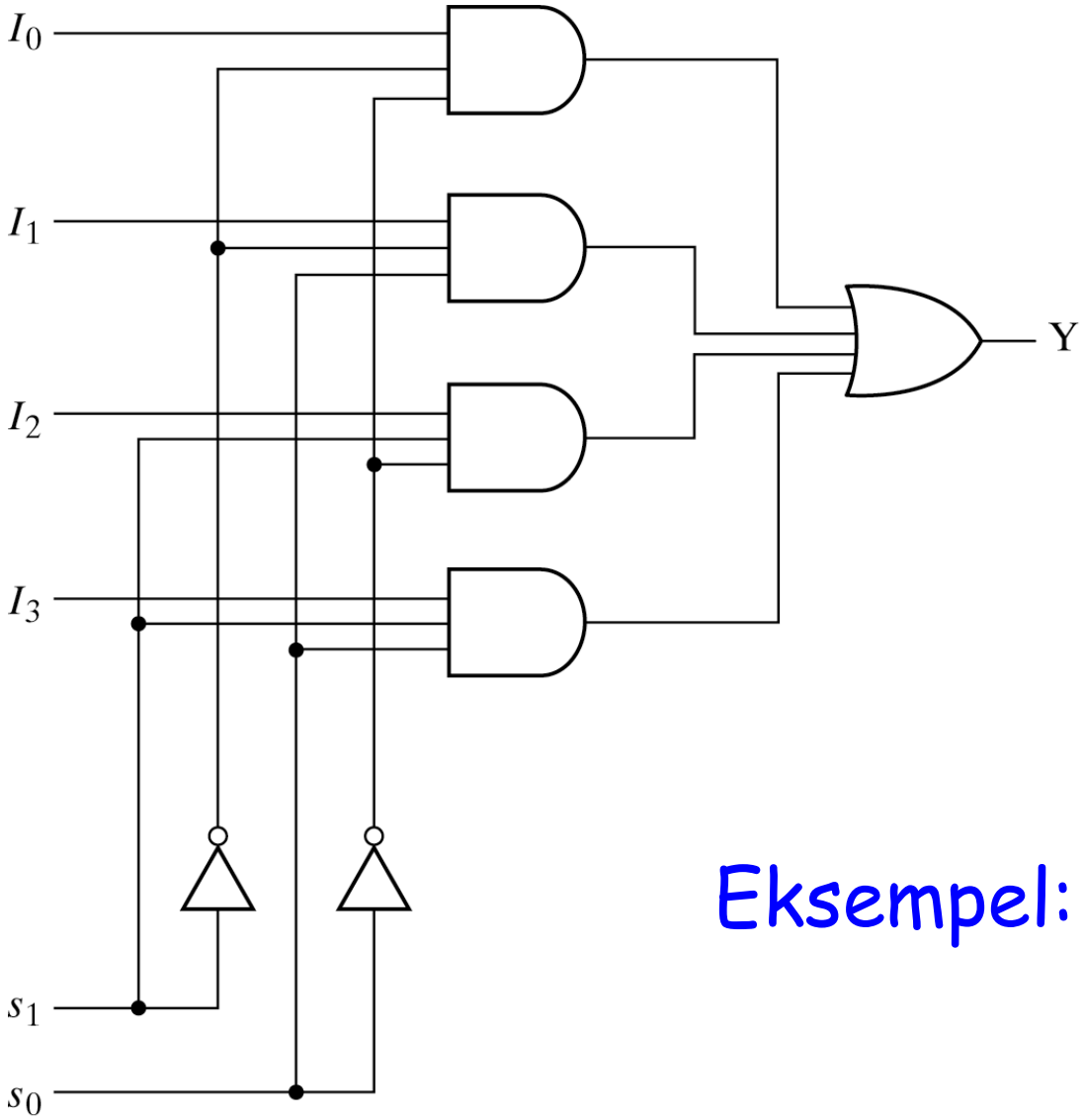
Eksempel: 2-1 MUX



Implementasjon



Symbol



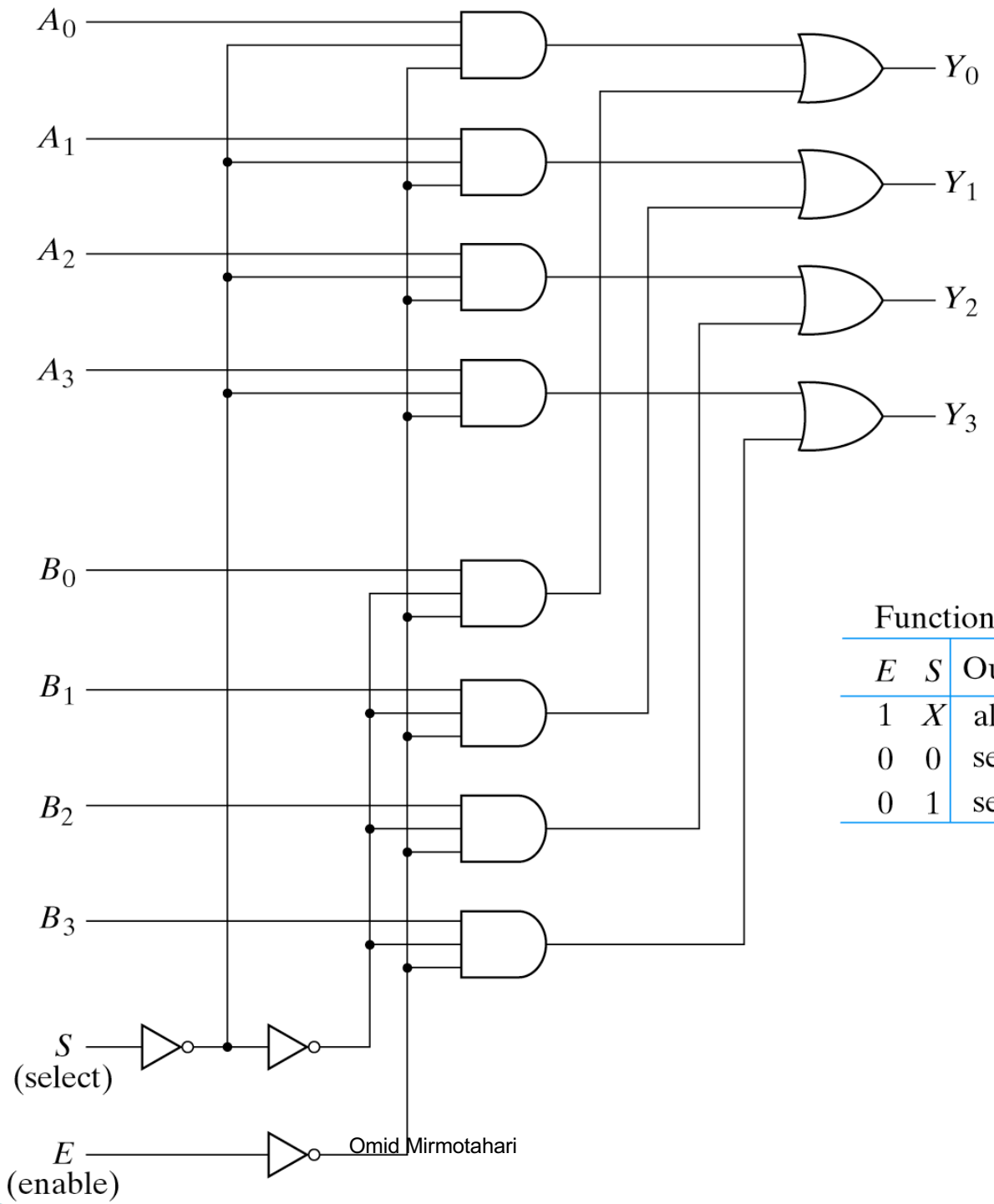
s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Eksempel: 4-1 MUX

MUX

Eksempel: 2-1 MUX

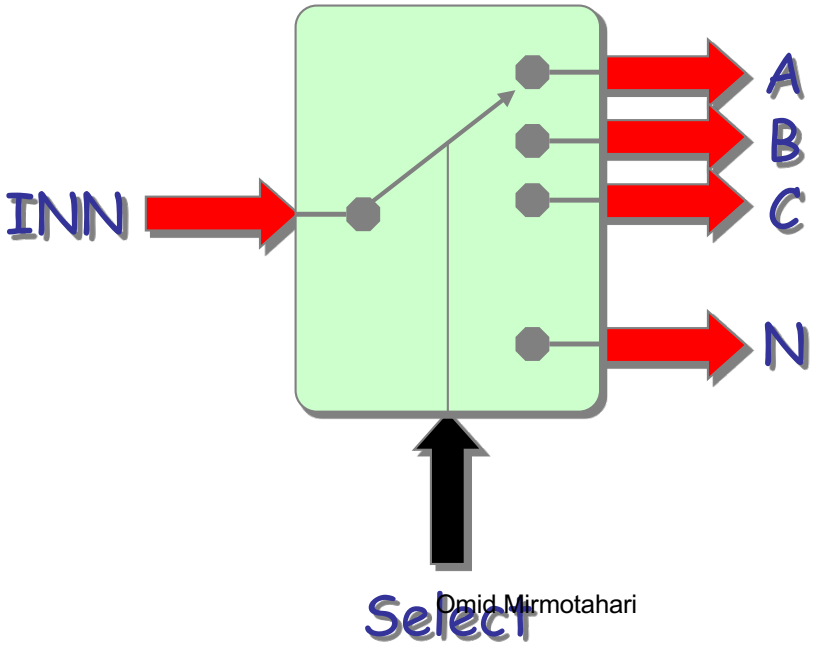


Function table

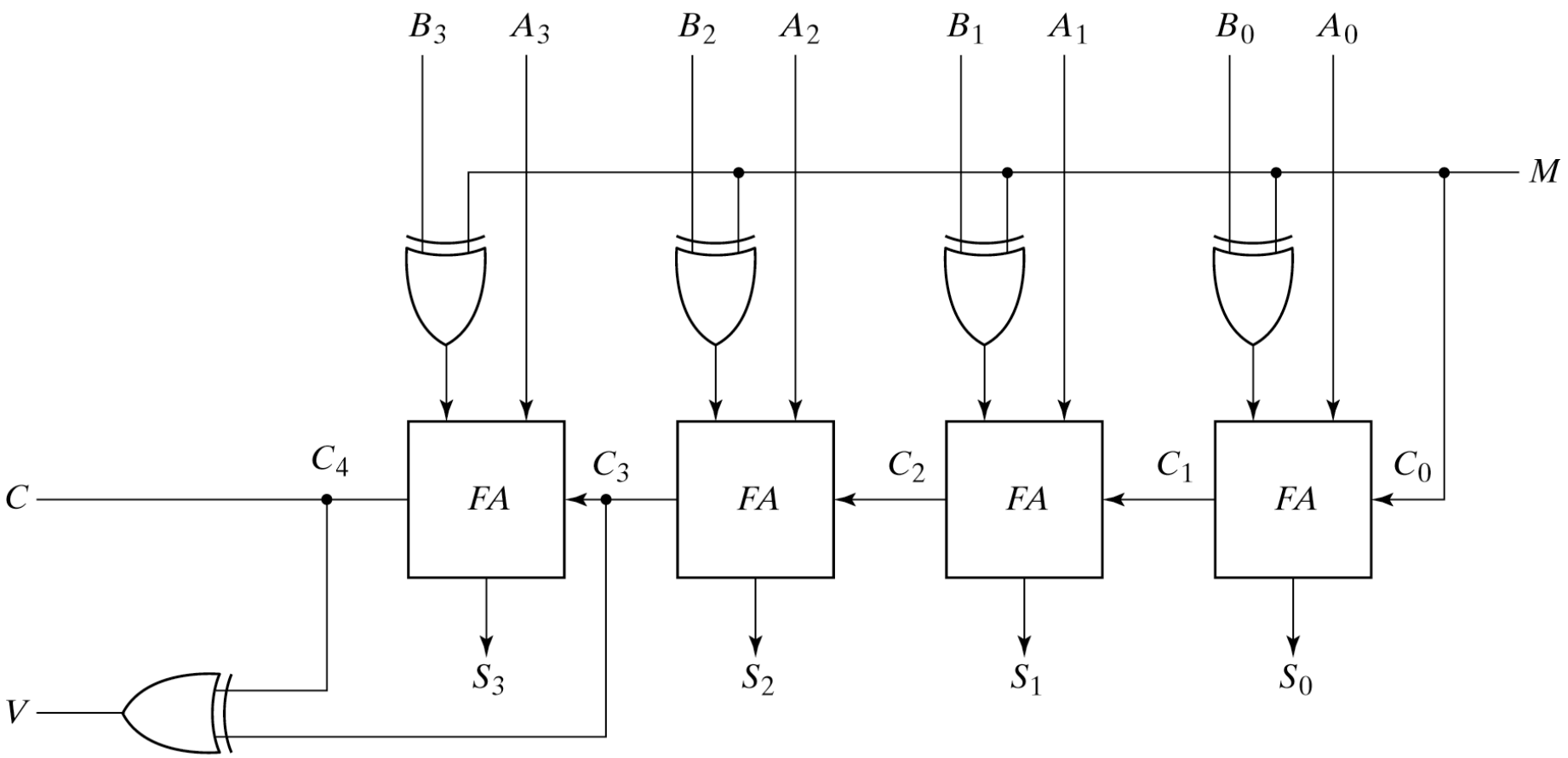
E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Demultiplekser

Demultiplekser – motsatt av multiplekser



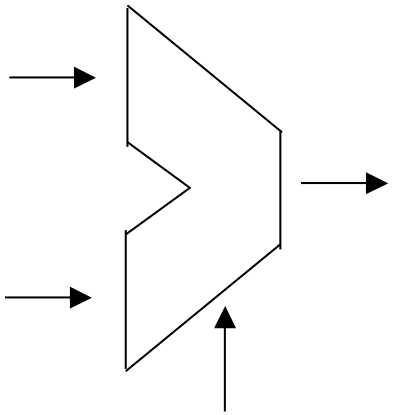
Kombinert adder/subtraktor



M=0: adder / M=1: subtraktor / V: overflow bit

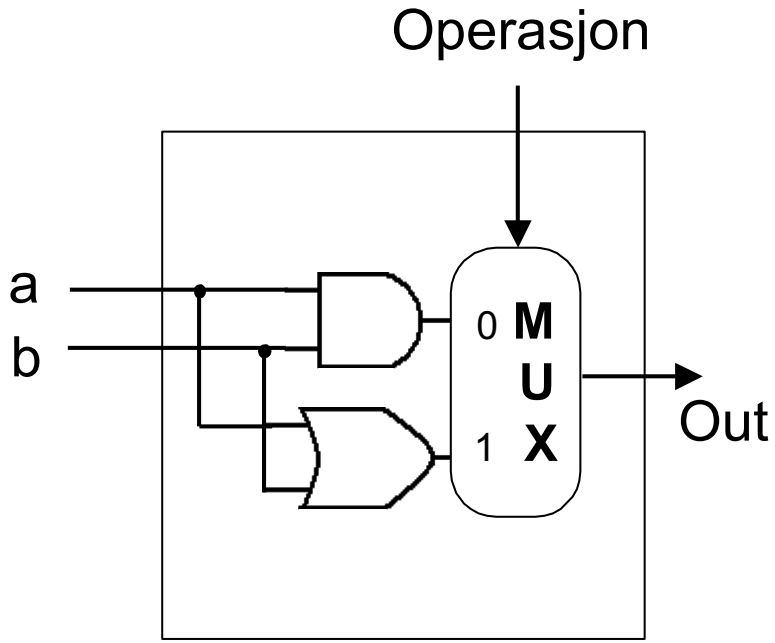
Aritmetisk logisk enhet (ALU)

- Den delen av CPU hvor logiske og aritmetiske beregninger utføre, f.eks addisjon, subtraksjon, AND, OR osv…

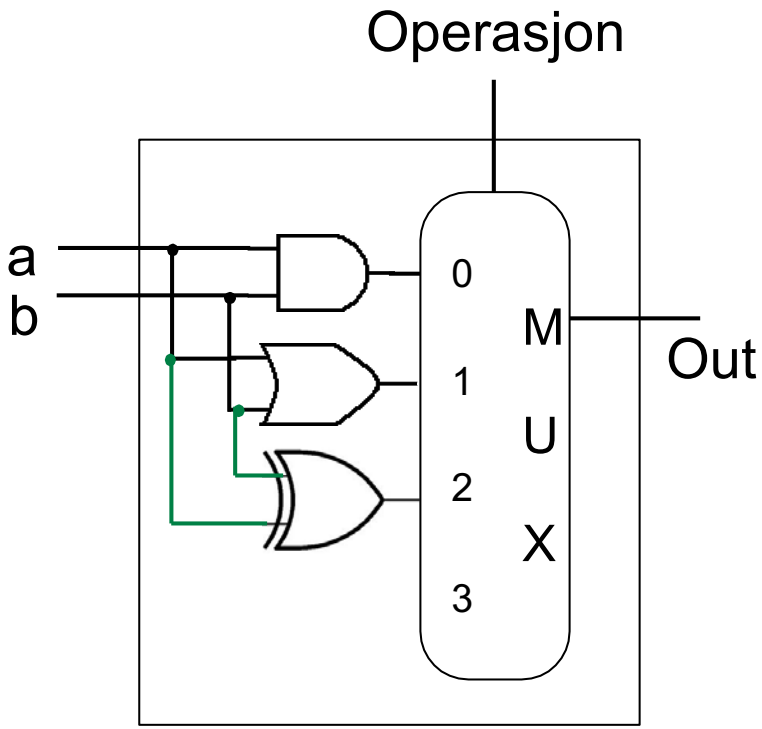


- Operasjoner som ALU skal utføre: (eksempelvis)
  - AND
  - OR
  - XOR
  - Addisjon
  - Subtraksjon

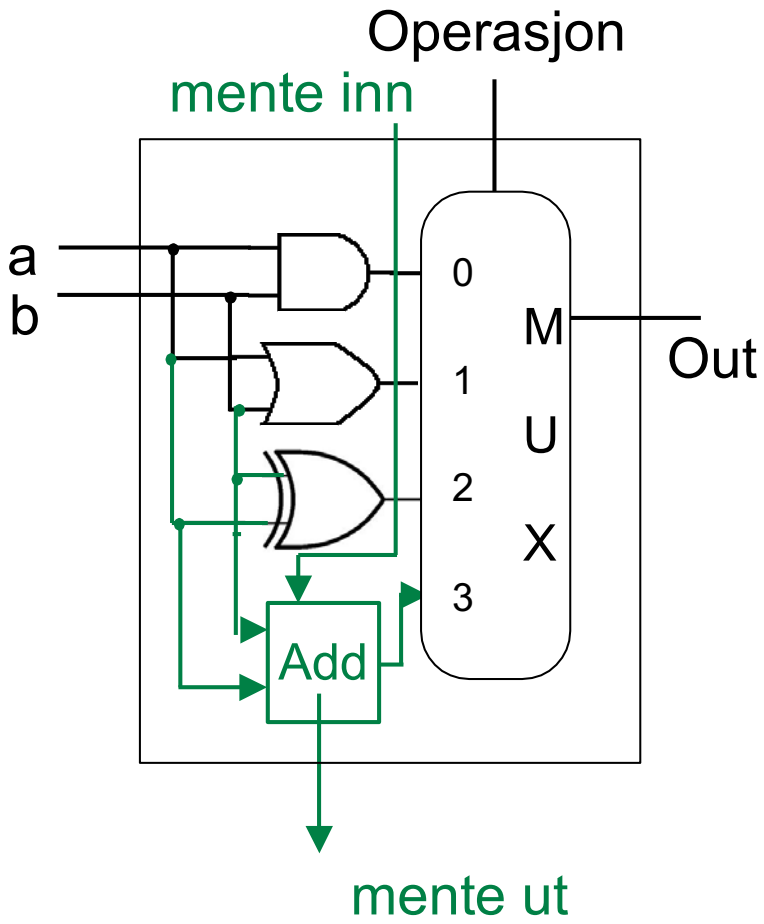
- Følgende byggeblokker trenger vi:
  - Fulladder
  - Multiplekser
  - AND,OR og NOT



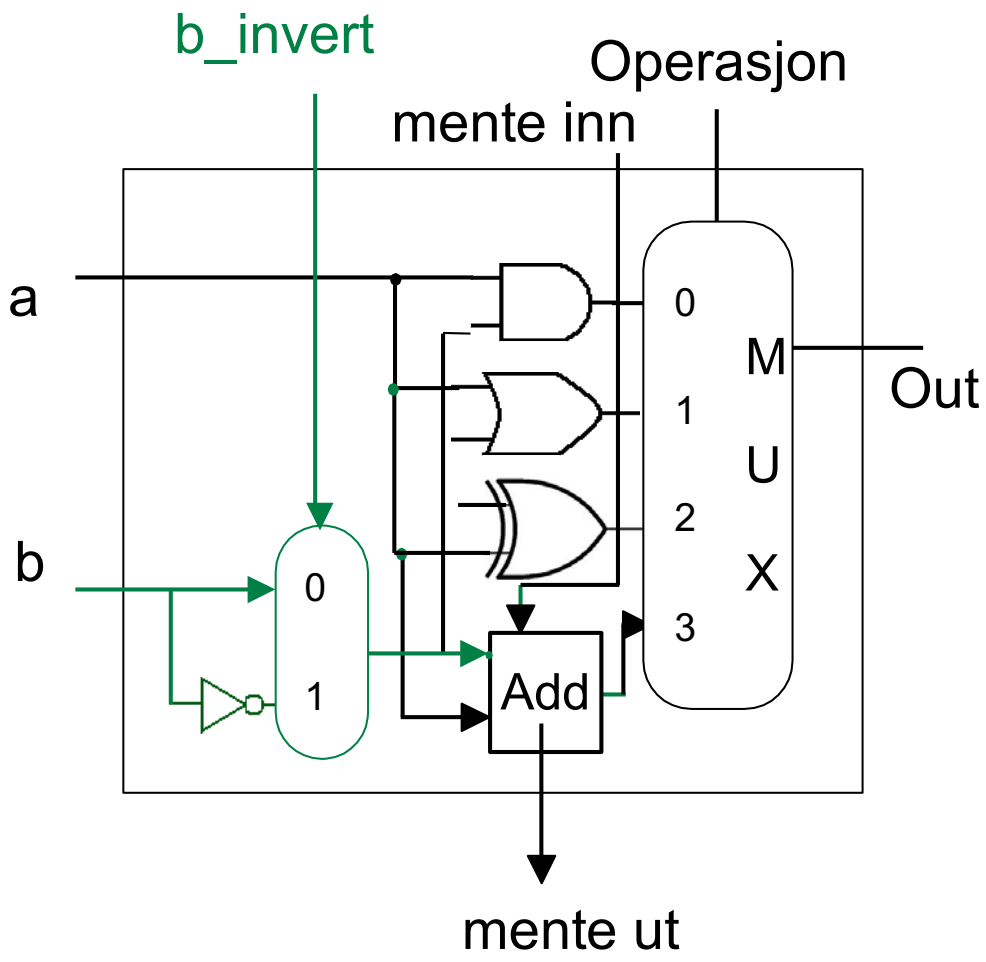
Operasjon	Out
0	AND (ab)
1	OR (a+b)



Operasjon		Out
0	0	AND
0	1	OR
1	0	XOR
1	1	?



Operasjon		Out
0	0	AND
0	1	OR
1	0	XOR
1	1	SUM

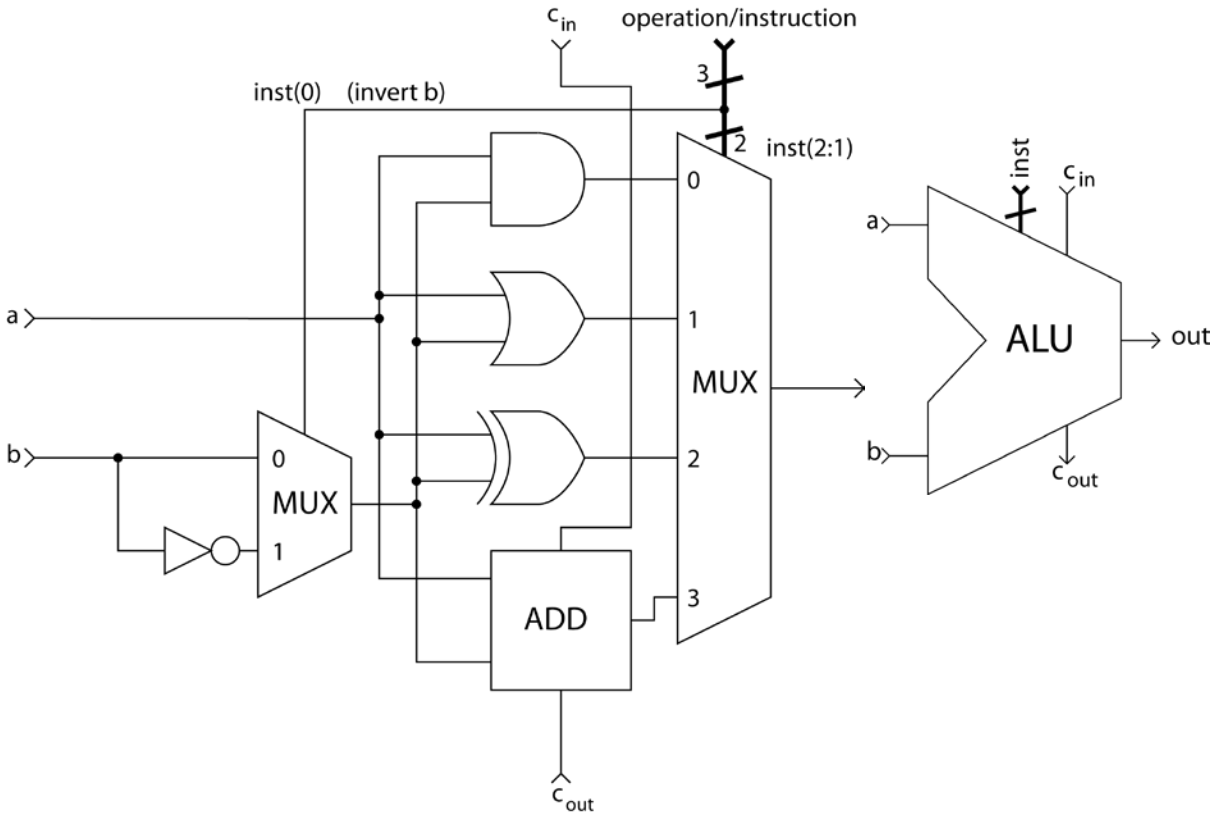


Operasjon = a2a1a0
a2 = b_invert
a1 = MSB MUX
a0 = LSB MUX

Operasjon	Out
000	ab
001	a+b
010	a XOR b
011	a SUM b
100	ab´
101	a+b´
110	a XOR b´
111	a SUB b

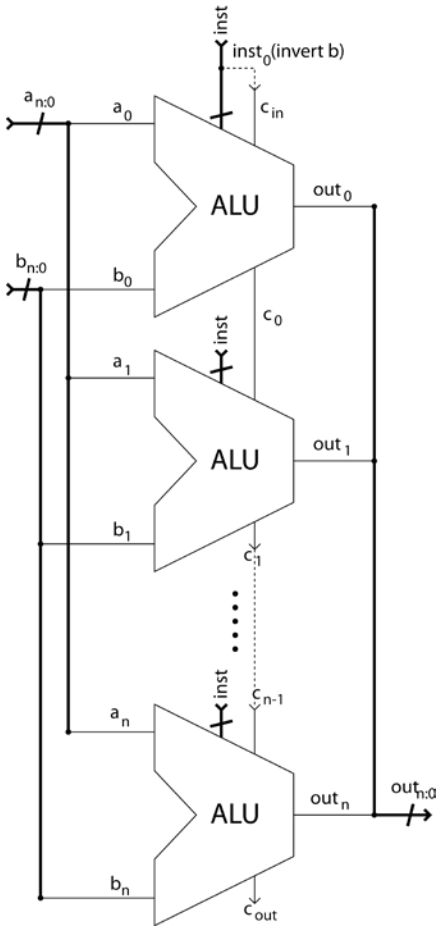
1-bit ALU

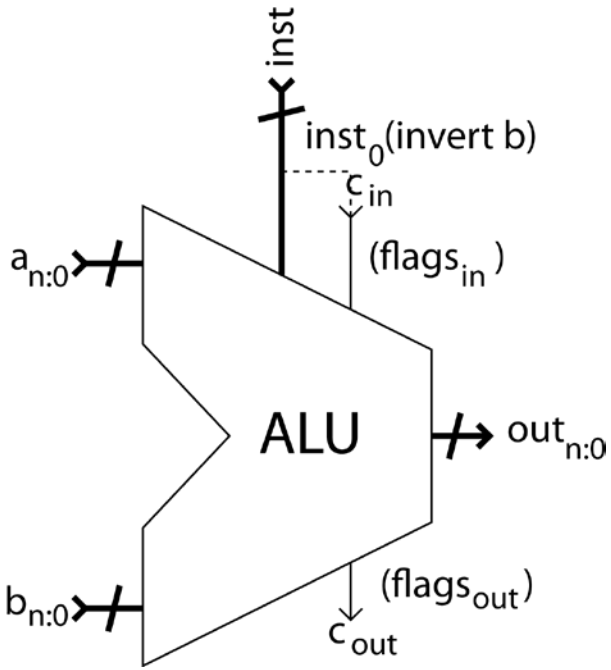
inst = a2a1a0
Inst(2) = a2 invert b
Inst(1) = a1 MSB MUX
Inst(0) = a0 LSB MUX



inst	computation
000	a · b
001	a · b´
010	a + b
011	a + b´
100	a ⊕ b
101	a ⊕ b´
110	a + b
111	a - b

N-bit ALU


inst
inst0(invert b)
an:0
a0
cin
ALU
out0
bn:0
b0
c0
inst
a1
ALU
out1
b1
c1
inst
cn-1
an
ALU
outn
outn:0
bn
cout


inst
inst0(invert b)
cin
an:0
(flagsin)
ALU
outn:0
bn:0
(flagsout)
cout

ALUer i CPU

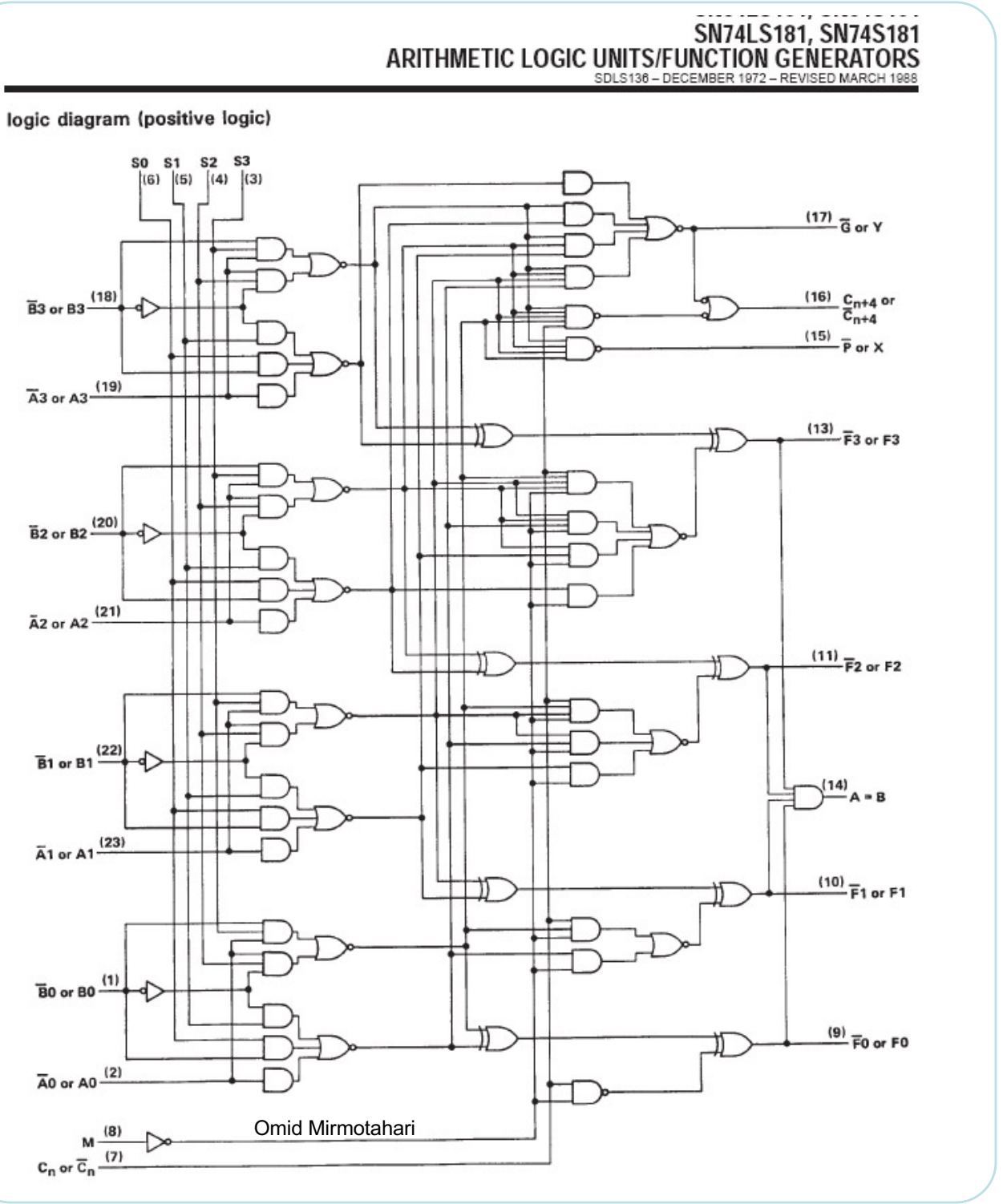
- Moderne CPU kan inneholde flere ALUer
- ALUer kan designes til å håndtere flere funksjoner per trinn og mer komplekse.
- Alternativt kan man bruke software aktivt til å designe slik at en ALU kan utføre komplekse operasjoner over flere trinn.
- TRADE-OFF: hvor skal man plassere kompleksiteten, i hardware eller software
- Ulemper med å sette kompleksitet i hardware er høyere kostnader i fbm strømforbruk, areal og produksjonskost.
- ALU designet spiller en stor rolle med hensyn på CPU sin ytelse, da det mest komplekse operasjonen setter maksimal klokkefrekvens.

ALU

ALU –Arithmetic Logic Unit

Generell regneenhet

Eksempel:
SN74LS181
4bit utbyggbar ALU
30 forskjellige operasjoner



ALU - SN74LS181

SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C_n}$ = H (no carry)	$\overline{C_n}$ = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A + B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$ PLUS 1
L	H	H	L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
H	L	L	L	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS $A^{\dagger}$	F = A PLUS A PLUS 1
H	H	L	H	$F = A + \overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

Komplett CPU: 4-bit databuss / 3bit adressebuss