Repetisjon

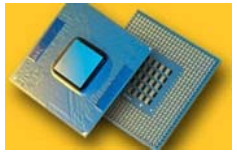
Sentrale temaer i kurset som er relevante for eksamen

(Eksamen kan inneholde stoff som ikke er nevnt her)

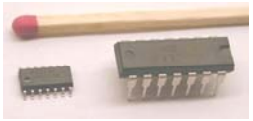
M.H

Teknologier

- VLSI (Very-Large-Scale-Integrated-Circuits)
  - Mer enn 100 000 porter på samme chip



- LSI (Large-Scale-Integrated-Circuits)
  - Noen få 1000 porter på samme chip
- SSI (Small- Scale-Integrated-Circuits)
  - Noen få porter på samme chip



M.H

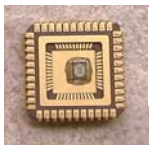
Digitale design metoder I

Hardvare basert design:

- SSI design
  - Setter sammen SSI pakker på kretskort

- FPGA (Field Programmable Gate Array)
  - Rekonfigurerbar logikk, eks. Xilinx, Altera, osv..

- ASIC (Application Specified Integrated Circuit)
  - Skreddersydd logikk (designer på transistor nivå)
  - Strømforbruk / hastighet / areal / spesialfunksjoner / pris / kombinert analog-digital / andre spesielle formål



M.H

Digitale design metoder II

Softvare / hardvare basert design:

- MicroController
  - Mikroprosessor med tilhørende I/O og memory på en brikke (datamaskin på en brikke)
- DSP (Digital Signal Processor)
  - Microcontroller spesialbygd for rask signalbehandling, eks. video, audio, osv..
- PC / arbeidsstasjon
  - Digitale operasjoner kan utføres på PC med I/O kort

M.H

NMOS transistoren

- NMOS (Negative doped Metal Oxide Silicon)
  - En 3 (4) terminals komponent

Symbol:

  - Spenningen på gate bestemmer om transistoren leder strøm mellom drain og source terminalene

M.H

PMOS transistoren

- PMOS (Positive doped Metal Oxide Silicon)
  - En 3 (4) terminals-komponent

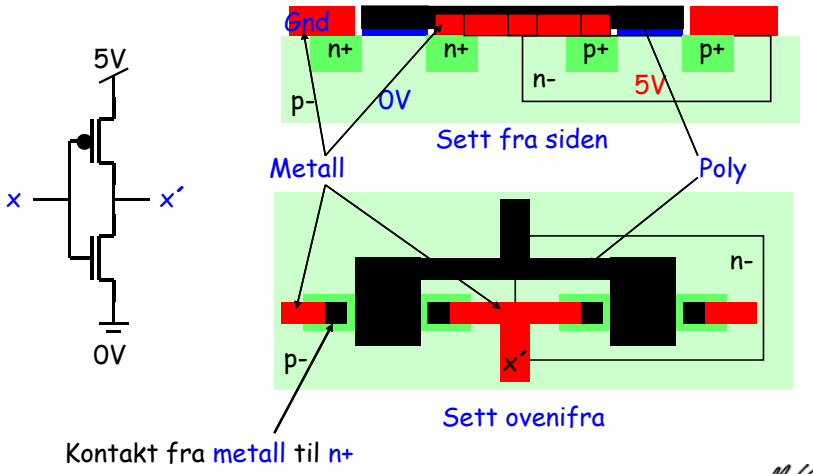
Symbol:

  - Spenningen på gate bestemmer om transistoren leder strøm i mellom drain og source terminalene

M.H

CMOS kretser

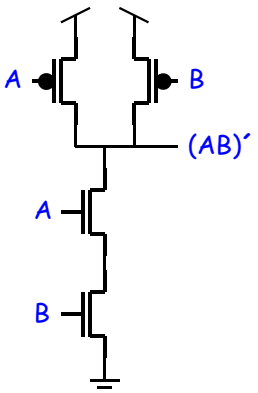
CMOS (Complementary MOS) inverter



M.H

CMOS NAND-krets

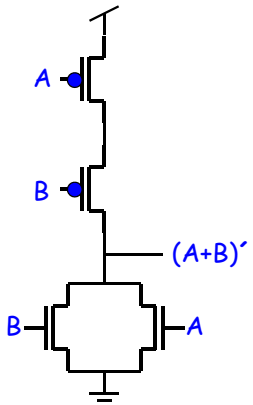
Både A og B må være 5V for å koble utgangen ned til 0V



M.H

CMOS NOR-krets

Det holder at enten A eller B er 5V for å koble utgangen ned til 0V



M.H

Portforsinkelse

Portforsinkelse (propagation delay) er et mål på hvor lang tid det tar før utgangen til en port reagerer på en logisk forandring på inngangen

Portforsinkelse skyldes parasittiske kapasitanser i systemet

Portforsinkelsen er gitt av driveregenskapene til porten samt hvor stor kapasitans det er på utgangen

Kapasitansen på en utgang er vanligvis dominert av kapasistansene til inngangene utgangen er koblet til

M.H

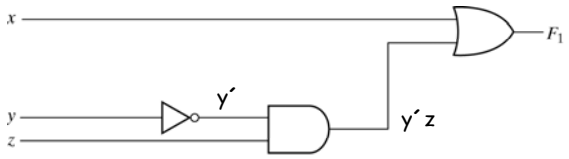
Boolske funksjoner

Tilordner en binær variabel en verdi bestemt av verdien på en eller flere andre binære variabler

Eksempel:

$F_1 = x + y'z$

Direkte port-implementasjon:



M.H

Sannhetstabell

En boolsk funksjon kan visualiseres i en sannhetstabell

Eksempel:

$F = x + y'z$

En gitt funksjon har kun en sannhetstabell

Men, en gitt sannhetstabell har uendelig mange funksjonsuttrykk

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	1
1 1 0	1
1 1 1	1

Stikkord: forenkling av funksjonsuttrykk

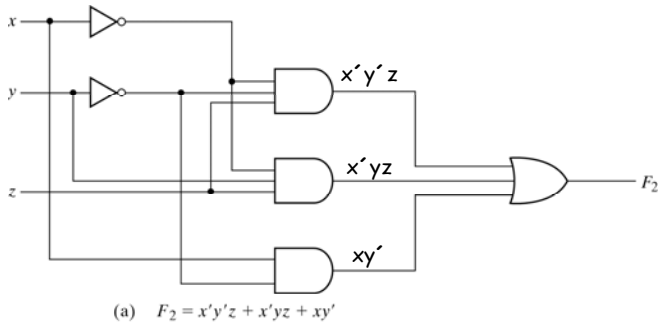
M.H

Teorem/postulatliste

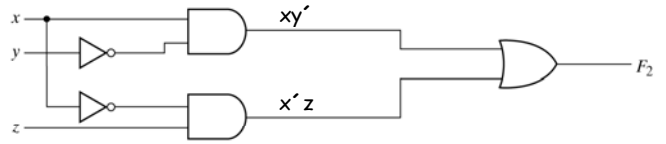
x + 0 = x	x•1 = x
x + x´ = 1	xx´ = 0
x + y = y+x	xy = yx
x + (y+z) = (x+y) + z	x (yz) = (xy)z
x(y+z) = xy + xz	x + (yz) = (x+y)(x+z)
x + x = x	x•x = x
x + 1 = 1	x•0 = 0
x + xy = x	x(x+y) = x
(x+y)´ = x´y´	(xy)´ = x´ + y´

M.H

Port-implementasjon av F_2



(a) $F_2 = x'y'z + x'yz + xy'$



(b) $F_2 = xy' + x'z$

M.H

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 fins det 4 forskjellige mintermer:

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

For 3 variable fins det 2^3 forskjellige mintermer:

xyz + xyz' + xy'z + xy'z'+x'yz + x'yz' + x'y'z + x'y'z'

M.H

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'

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

En sannhetstabell kan sees på som en liste av mintermer

M.H

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 fins det 4 forskjellige makstermer:

$$(x+y)(x+y')(x'+y)(x'+y')$$

For n variable fins det 2^n forskjellige makstermer:

M.H

Generell design prosedyre

1. Bestem hvilke signal som er innganger og utganger
2. Sett opp sannhetstabell for alle inngangskombinasjoner
3. Generer funksjonsuttrykket som sum av mintermer
4. Tilpass /forenkle funksjonsuttrykket mot aktuelle porter

M.H

Forenkling på portnivå

Det er ikke alltid at det enkleste funksjonsuttrykket resulterer i den enkleste port-implementasjonen

Ved forenkling på portnivå må man vite hvilke porter man har til rådighet, og så justere funksjonsuttrykket mot dette. (håndverk)

M.H

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}

wx \ yz	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

(w: 11, 10; x: 01, 11; y: 11, 10; z: 01, 11)

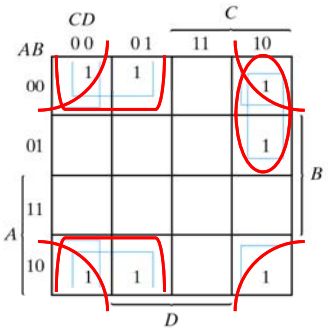
M.H

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

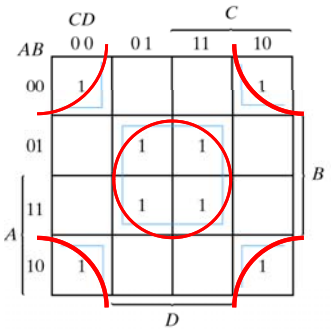
Eksempel

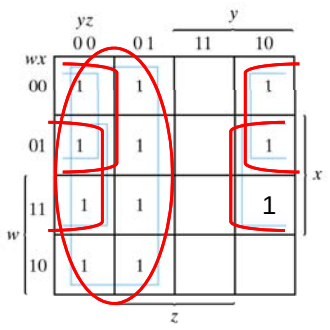


M.H

Grupperingsregler for diagram med 2-4 variable

Eksempel:





M.H

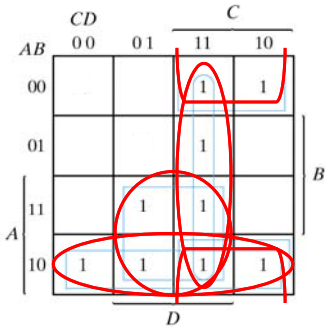
Utlesningsregler for diagram med 2-4 variable

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´



M.H

Utlesning av "0"ere

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

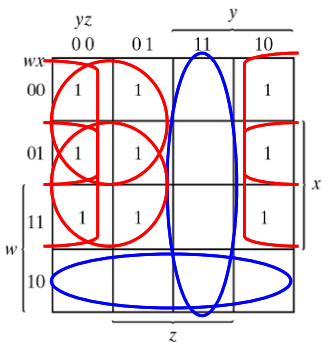
Dette kan noen ganger gi en enklere funksjon, eksempel:

F´ = yz + wx´

F = (yz + wx´)´

Hadde vi lest ut "1"ere ville vi fått

F = xy´ + w´y´ + w´z´ + xz´



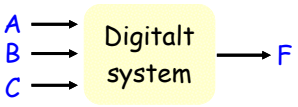
M.H

Don't care

I noen tilfeller har man inngangskombinasjoner som aldri dukker opp

I andre tilfeller bryr man seg ikke om utfallet for visse inngangskombinasjoner

Slike kombinasjoner kalles don't care kombinasjoner og markers med "X"



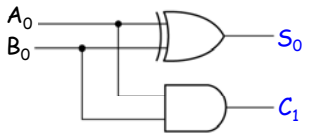
Innganger Utganger

A B C	F
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	X
1 1 0	1
1 1 1	X

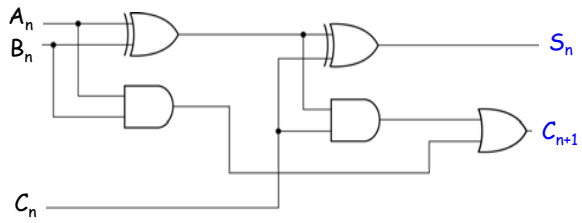
M.H

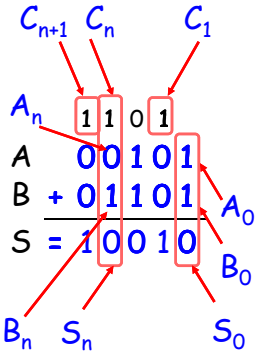
Binær adder

Halvadder (ikke mente inn)



Fulladder (evt. mente inn)



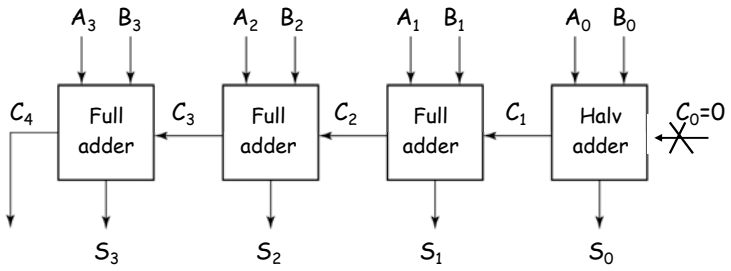


M.H

Et adder system

Systemelementer:

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



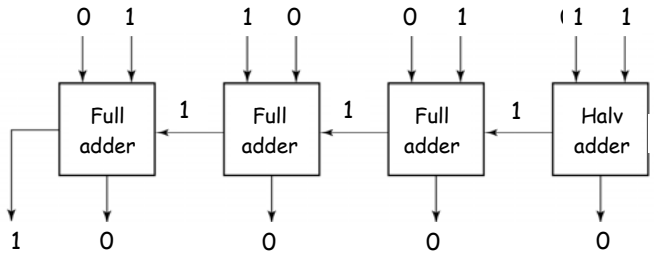
M.H

Menteforplantning

Portforsinkelse gir menteforplantning (rippeladder)

Eksempel

Adderer 0101 og 1011



M.H

Definisjoner

Kombinatorisk logikk

Utgangsverdiene er entydig gitt av nåværende kombinasjon av inngangsverdier

Sekvensiell logikk

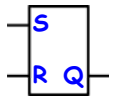
Inneholder hukommelse (låsekretser)

Utgangsverdiene er gitt av nåværende kombinasjon av inngangsverdier, samt sekvensen (tidligere inngangs-/utgangsverdier)

M.H

SR Latch - funksjonell beskrivelse

1) Kretsen skal sette Q til "1" hvis den får "1" på inngang S. Når inngang S går tilbake til "0" skal Q forbli på "1"
2) Kretsen skal resette Q til "0" når den får "1" på inngang R. Når inngang R går tilbake til "0" skal Q forbli på "0"
3) Tilstanden "1" på både S og R brukes normalt ikke



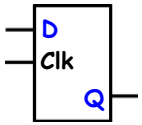
SR

S R	Q
0 0	låst
0 1	0
1 0	1
1 1	0

M.H

D Latch

Dataflyten gjennom en D latch kontrolleres av et klokkesignal



D latch:

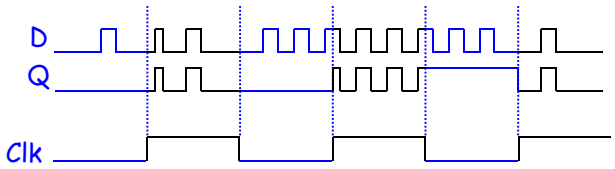
1) Slipper gjennom et digitalt signal så lenge klokkeinngangen er "1" (transparent)
2) I det øyeblikk klokkeinngangen går fra "1" til "0" låser utgangen seg på sin nåværende verdi. Forandringer på inngangen vil ikke påvirke utgangsverdien så lenge klokkesignalet er "0"

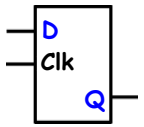
M.H

D Latch

Clk = 1 : kretsen slipper gjennom signalet

Clk = 0 : kretsen holder (låser) utgangssignalet





Logisk verdi på D i det øyeblikk Clk går i fra "1" til "0" bestemmer verdien som holdes på Q

M.H

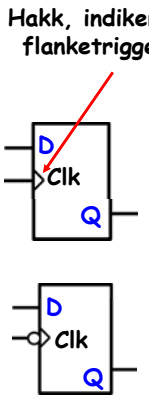
Flip-Flop'er

Flip-Flop'er kommer i to varianter:

- Positiv flanketrigget
- Negativ flanketrigget

På en positiv flanketrigget Flip-Flop kan utgangen kun skifte verdi i det øyeblikk klokkesignalet går fra "0" til "1".

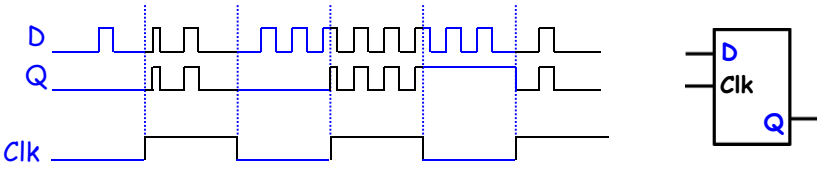
På en negativ flanketrigget Flip-Flop kan utgangen kun skifte verdi i det øyeblikk klokkesignalet går fra "1" til "0".



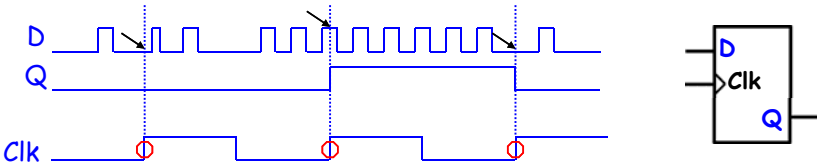
M.H

D Flip-Flop

En D latch er transparent for Clk=1



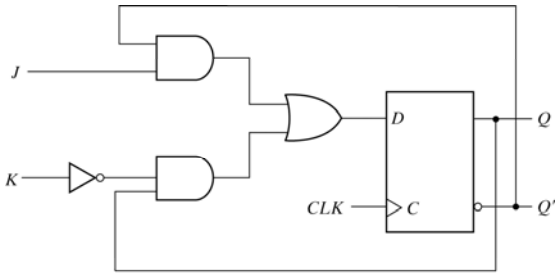
En positiv flanketrigget D flip-flop sampler verdien på D i det øyeblikk Clk går fra "0" til "1" (positiv flanke). Denne verdien holdes fast på utgangen helt til neste positive flanke



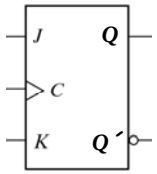
M.H

JK Flip-Flop

Kretsoppbygging



Grafisk symbol

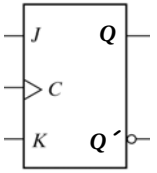


M.H

JK Flip-Flop

En JK flip-flop har følgende egenskaper

J=0, K=0:	Utgang låst
J=0, K=1:	Resetter utgang til "0"
J=1, K=0:	Setter utgang til "1"
J=1, K=1:	Inverterer utgang Q→Q´



Utgangen kan kun forandre verdi på stigende klokkeflanke

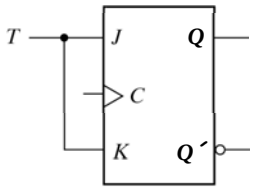
En JK flip-flop er den mest generelle flip-floppen vi har

J K	Q
0 0	låst
0 1	0
1 0	1
1 1	Q´

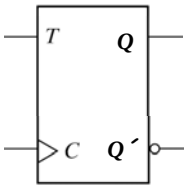
M.H

T Flip-Flop

Kretsoppbygging



Grafisk symbol



M.H

T Flip-Flop

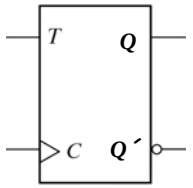
En T flip-flop har følgende egenskaper

T=0, Utgang låst

T=1, Inverterer utgang Q→Q´

Utgangen kan kun forandre verdi på stigende klokkeflanke

Det er lett å lage tellere av T flip-flop'er



T	Q
0	låst
1	Q´

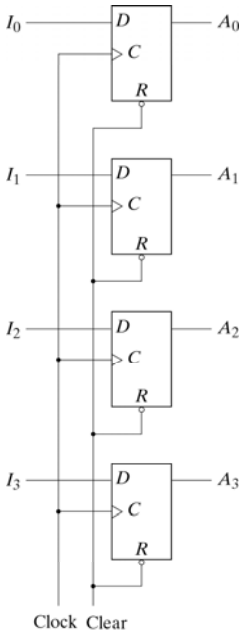
M.H

Register

N bits register - parallellkobling av N stk. D flip-flopper

Data I_0-I_3 slippes gjennom til utgang A_0-A_3 på stigende klokkeflanke

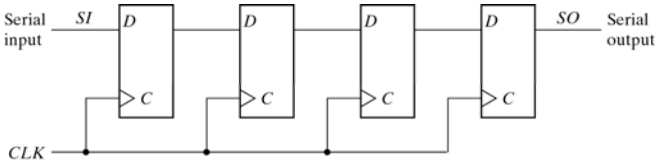
Anvendelse: Kontroll av dataflyt



M.H

Shift register

Shiftregister – Seriell inngangsdata klokkes gjennom registeret ett trinn per klokkeperiode



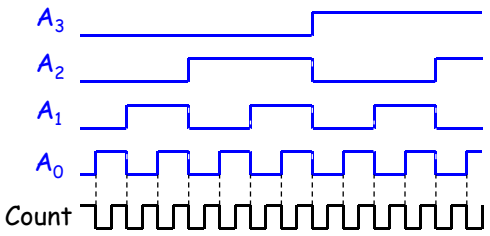
For animasjoner se
www.play-hookey.com

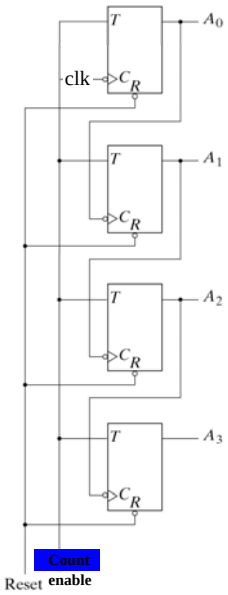
M.H

Rippeltellere

Shift register – T flip-flopper

Utgang "toggler" på negativ flanke





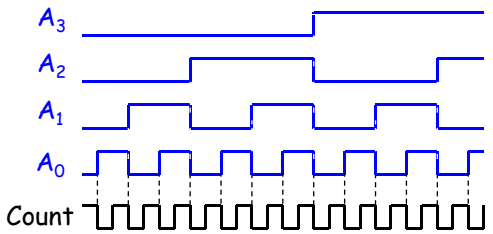
www.play-hookey.com

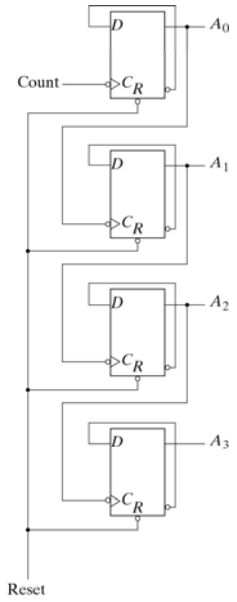
M.H

Rippeltellere

Shift register – D flip-flopper

Fører invertert utgang tilbake på negativ klokke





www.play-hookey.com

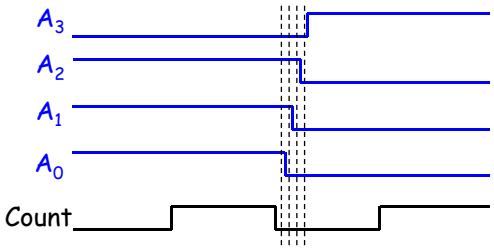
M.H

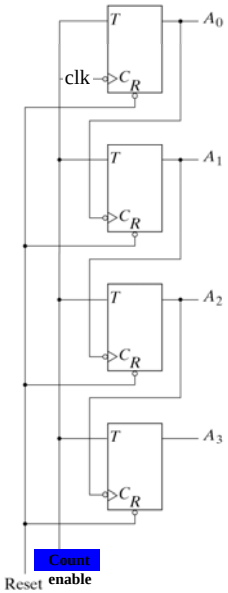
Rippeltellere

Signalet rippler/propagerer gjennom trinnene

Problem:

Portforsinkelse gir temporært gale utgangsverdier



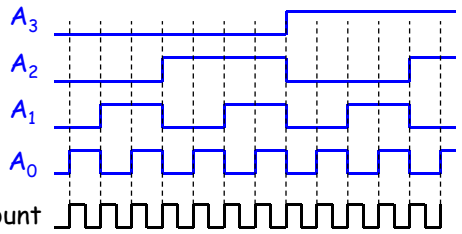


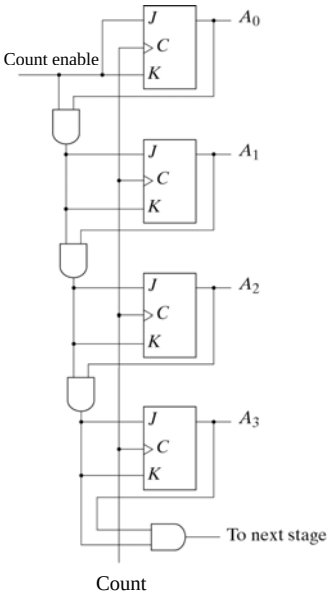
M.H

Synkronteller

Utganger skifter verdi samtidig / synkront

J	K	Q
0	0	uforandret
0	1	0
1	0	1
1	1	Q´

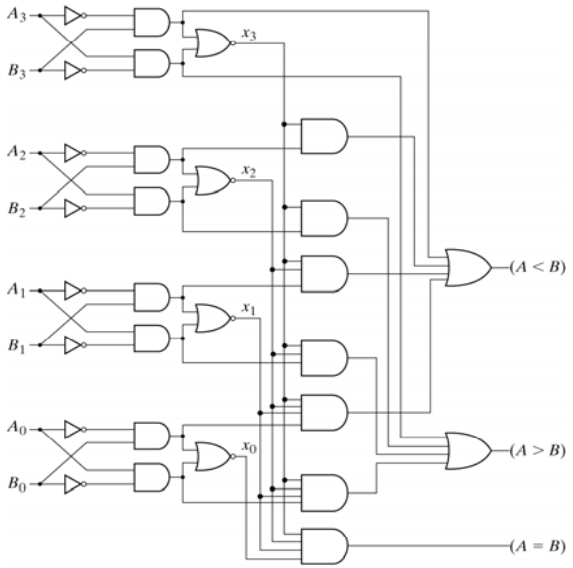




www.play-hookey.com

M.H

Komparator - eksempel

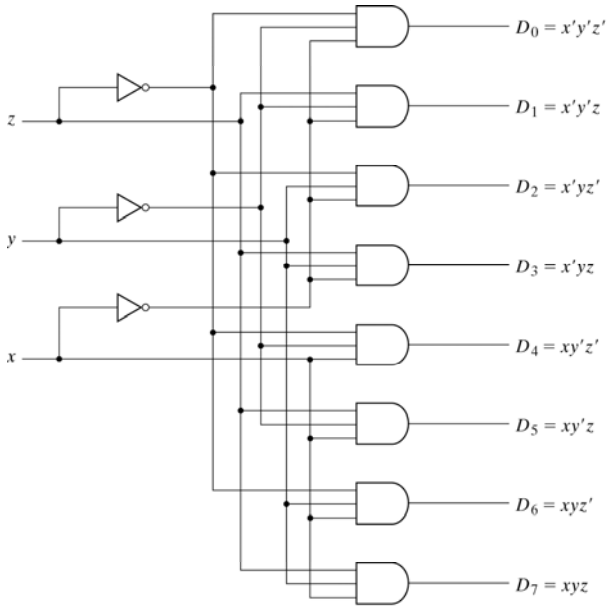


M.H

Dekoder

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

Eksempel: 3bit inn / 8bit ut



M.H

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

M.H

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

$x = D_4 + D_5 + D_6 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

$z = D_1 + D_3 + D_5 + D_7$

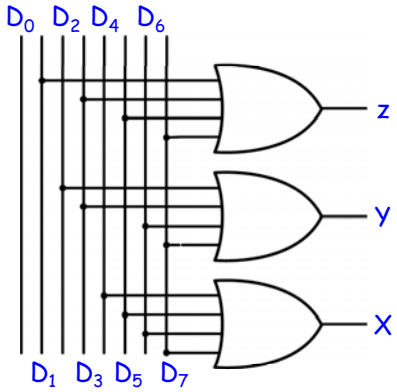
Antar at det ikke eksisterer andre inngangskombinasjoner

M.H

Enkoder

Eksempel

$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$

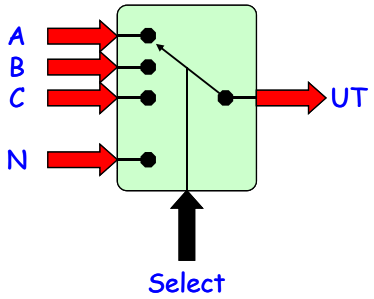


M.H

Multiplekser

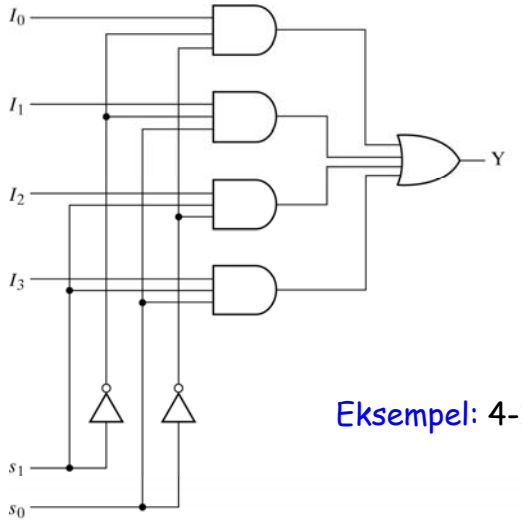
Multiplekser (MUX) – velger hvilke innganger som slippes ut

Hver inngang kan bestå av ett eller flere bit



M.H

MUX



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

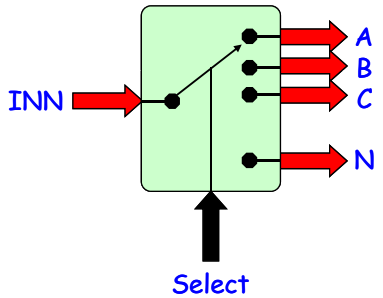
(b) Function table

Eksempel: 4-1 MUX

M.H

Demultiplekser

Demultiplekser – motsatt av multiplekser

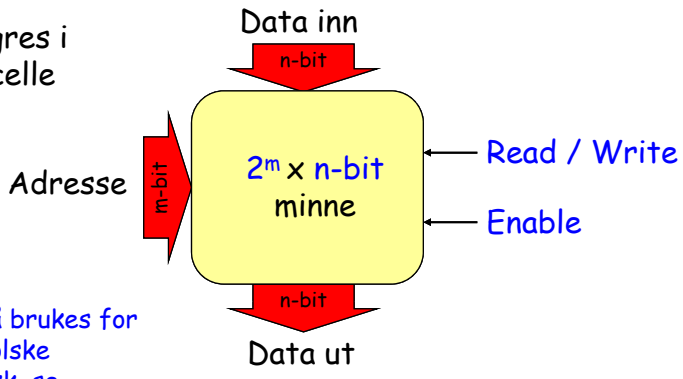


M.H

Minne - generelt

Generelt minne

Hvert bit lagres i egen minne-celle



NB:

Minne kan også brukes for å generere Boolske funksjonsutrykk, se tidligere eksamens-oppgaver

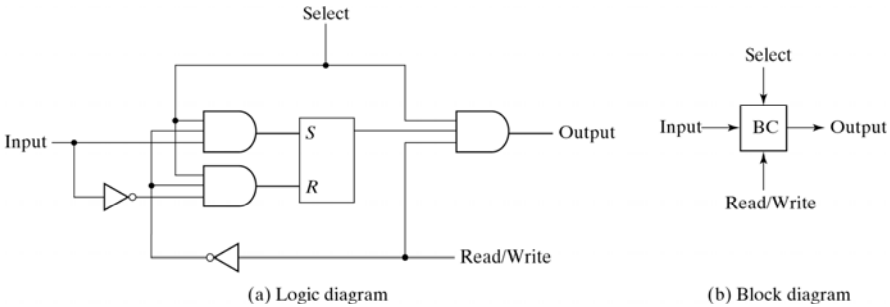
Data inn/data ut deler ofte samme buss i praksis

M.H

Minne – teoretisk cellestruktur

RAM celle laget av standardkomponenter

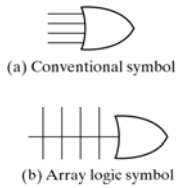
Separat datainngang/datautgang

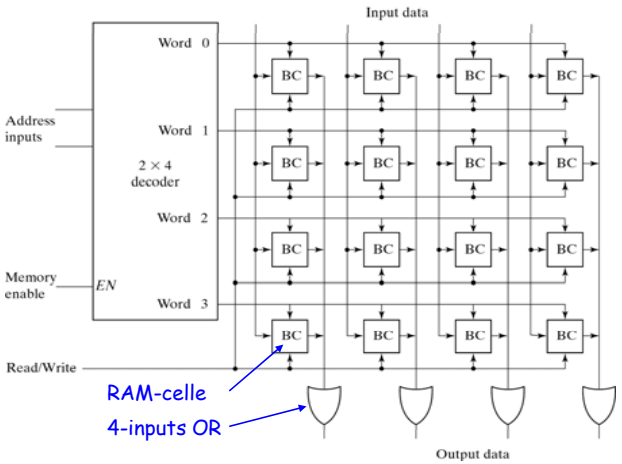


M.H

RAM – teoretisk system

System-eksempel:
4x4bit RAM





M.H

SRAM

SRAM (Static-Random-Access-Memory)

- \+ Raskt, <1ns – CPU cacheminne
- \+ Trenger ikke refresh, latch-basert

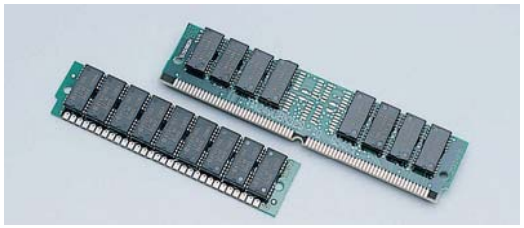
- \- Høyt strømforbruk
- \- Tar stor plass, 4 eller 6 transistorer / celle
- \- Flyktig (volatile), data forsvinner når strømforsyningen slås av

M.H

DRAM

DRAM - Dynamic Random Access Memory

+ Kompakt/billig struktur, 1 transistor / celle
- Trenger oppfriskning (refresh) hver 20-30ms
- Flyktig (volatile), data forsvinner når strømforsyningen slås av



M.H

ROM

ROM – Read Only Memory

+ Ikke-flyktig (non-volatile), data vedvarer når strømforsyningen slås av
+ Rask operasjon <1ns
+ Lavt strømforbruk
+ Kan lages meget kompakt, hver celle består kun av en transistor eller diode
- Må programmeres på fabrikken, aktuell for store produksjonskvanta

M.H

EPROM

EPROM, Erasable Programmable Read Only Memory

+ Ikke-flyktig (non-volatile), data vedvarer når strømforsyningen slås av
- Data kan legges inn av forbruker, data kan slettes ved bruk av ultrafiolett lys (eget lokk i pakken). Data må slettes (erase) før ny data kan legges inn

M.H

EEPROM

EEPROM, Elektrical Erasable Programmable Read Only Memory

+ Ikke-flyktig (non-volatile), data vedvarer når strømforsyningen slås av
- Data kan legges inn av forbruker, data kan slettes uten bruk av ultrafiolett lys. Data må slettes (erase) før ny data kan legges inn

M.H

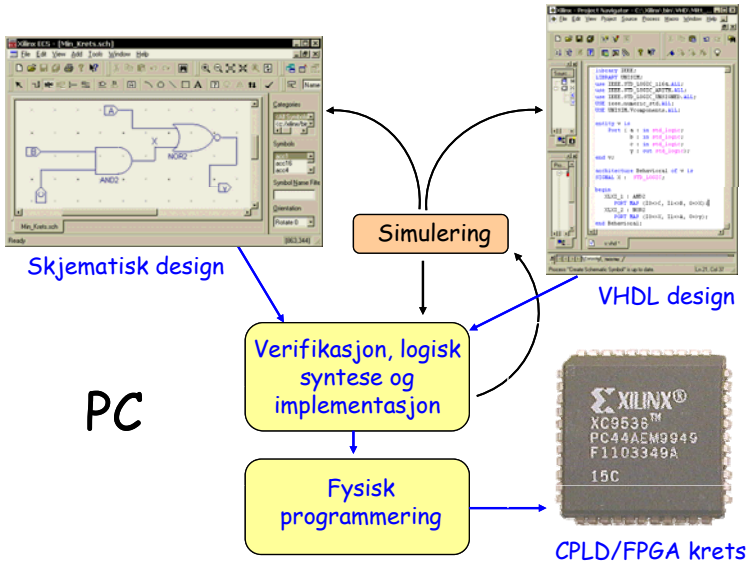
Flash

Flash - en billig og rask variant av EEPROM. Meget aktuell hukommelse for mange anvendelser

- Ikke-flyktig (non-volatile), data vedvarer når strømforsyningen slås av
- Kan lages meget kompakt, hver celle består av kun en transistor

M.H

CPLD/FPGA design



M.H

FPGA-oppbygning

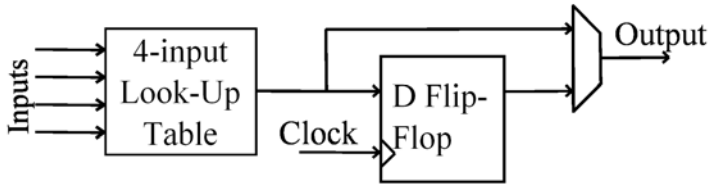
En FPGA-brikke inneholder et gitt antall rekonfigurerbare logikkenheter

Hver logikkenhet inneholder typisk en oppslagstabell (LUT) og en flip-flop

logikkenhetene kobles sammen slik softvaren finner hensiktmessig for å implementere designet vårt

M.H

Generell FPGA-logikkenhet



LUT-en kan brukes til å implementere en hvilken som helst 4-variabels boolsk funksjon / kombinatorisk krets med 4 innganger og 1 utgang

(en slags ROM som kan omprogrammeres)

M.H

Tilstandsmaskin

En tilstandsmaskin er et sekvensielt system som gjennomløper et sett med tilstander styrt av verdiene på inngangssignalene

Tilstanden systemet befinner seg i, pluss evt. inngangsverdier bestemmer utgangsverdiene

Tilstandsmaskins-konseptet gir en enkel og oversiktlig måte å designe avanserte system på

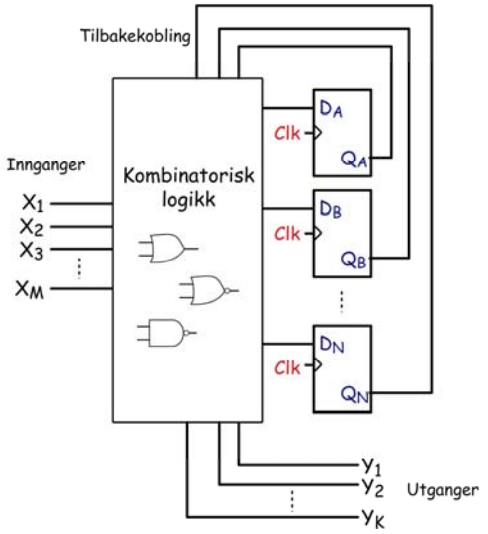
M.H

Tilstandsmaskin

Generell tilstands-maskin basert på D flip-flops

N-stk flip-flops gir 2^N forskjellige tilstander

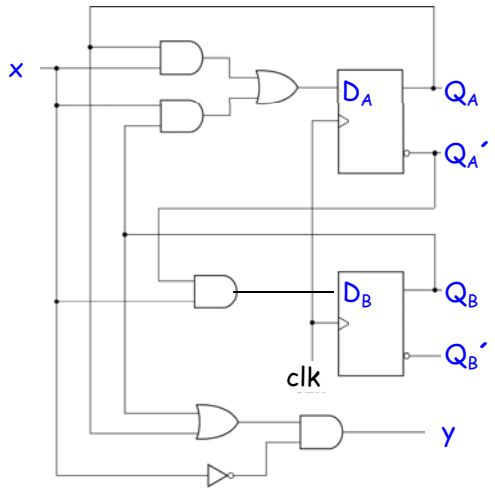
Utgangssignalene er en funksjon av nåværende tilstand pluss evt. inngangsverdier



M.H

Eksempel nr.1

Tilstandsmaskin der utgang y er en funksjon av tilstanden gitt av verdiene til Q_A og Q_B, samt inngangen x



M.H

Tilstandstabell

Tilstandstabell = sannhetstabell for tilstandsmaskin

Eksempel nr.1: En inngang, en utgang og 2 stk. D flip-flops

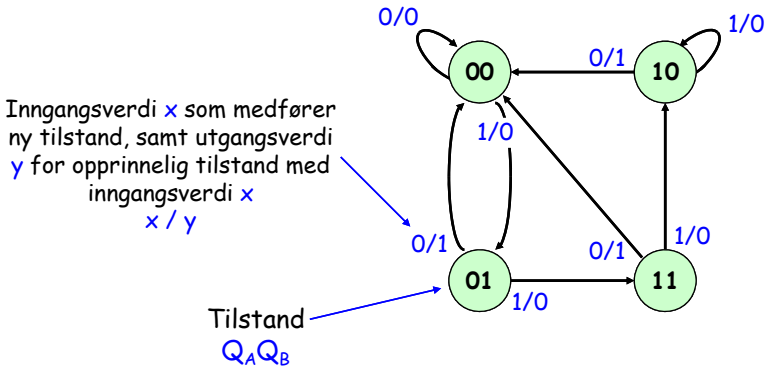
Nåværende tilstand Q_A Q_B	Inngang x	Neste tilstand Q_A Q_B	Utgang for nåværende tilstand y
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	0 0	1
0 1	1	1 1	0
1 0	0	0 0	1
1 0	1	1 0	0
1 1	0	0 0	1
1 1	1	1 0	0

M.H

Tilstandsdiagram

Tilstandsdiagram = grafisk illustrasjon av egenskapene til en tilstandsmaskin

Eksempel nr.1:

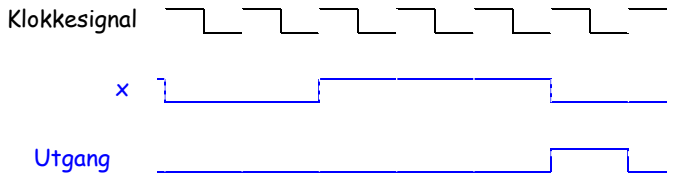


M.H

Eksempel nr.3 – design av sekvensdetektor

Ønsker å lage en krets som finner ut om det har forekommet tre eller flere "1"ere etter hverandre i en klokket bit-sekvens x

Klokket bit-sekvens: Binært signal som kun kan skifte verdi synkront med et klokkesignal



M.H

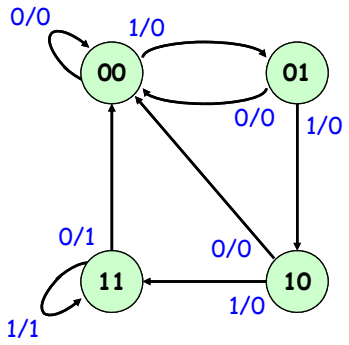
Eksempel nr.3 – design av sekvensdetektor

Tilstandsdiagram

Velger å ha 4 tilstander. Lar hver tilstand symbolisere antall "1"ere som ligger etter hverandre i bit-sekvensen.

Inngang: bit-sekvens x

Utgang: gitt av tilstanden, "0" for tilstand 0-2, "1" for tilstand 3



M.H

Eksempel nr.3

Bruker D flip-flops

D_A og D_B settes til de verdiene man ønsker at Q_A og Q_B skal ha i neste tilstand

$D_A = Q_A'Q_Bx + Q_AQ_B'x + Q_AQ_Bx$

$D_B = Q_A'Q_B'x + Q_AQ_B'x + Q_AQ_Bx$

$y = Q_AQ_B$

Nåværende tilstand		Inngang	Neste tilstand		Utgang for nåværende tilstand
Q_A	Q_B	x	Q_A	Q_B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

M.H

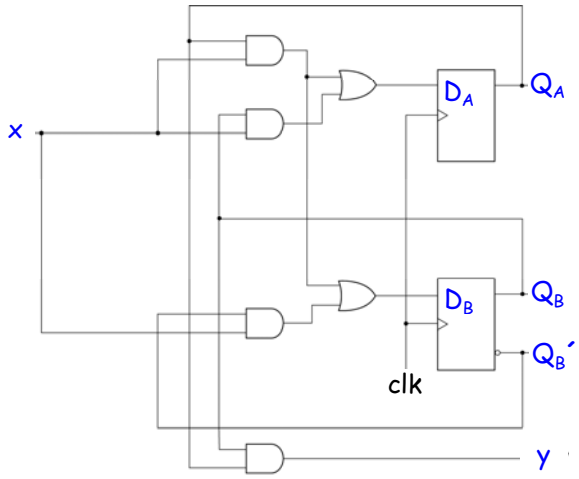
Eksempel nr.3

Forenkler uttrykkene med Karnaugh-diagram

$D_A = Q_A x + Q_B x$

$D_B = Q_A x + Q_B' x$

$y = Q_A Q_B$



Generell designprosedyre basert på D flip-flops

1) Definer tilstandene, inngangene og utgangene
2) Velg tilstandskoder, og tegn tilstandsdiagram
3) Tegn tilstandstabell
4) Reduser antall tilstander hvis nødvendig
5) Bytt tilstandskoder hvis nødvendig for å forenkle
6) Finn de kombinatoriske funksjonene
7) Sjekk at ubrukte tilstander leder til ønskede tilstander
8) Tegn opp kretsen

VHDL

VHDL (Very High Speed Integrated Circuits) Hardware **D**escription **L**anguage

VHDL er en tekstlig beskrivelse av et digitalt design

VHDL versus skjematisk design

+ Mulighet for å beskrive kretser på et høyere abstraksjonsnivå
+ Raskere design

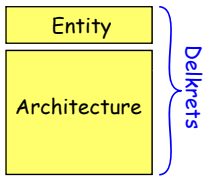
- Kan miste detaljkontroll på portnivå

Generell VHDL struktur - delkretser

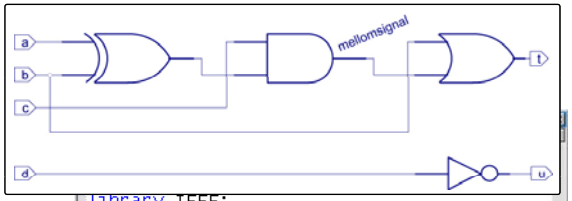
En VHDL beskrivelse består av en eller flere delkretser

Hver delkrets har:

1) Entity: Et grensesnitt som definerer delkretsens innganger og utganger (pinout)
2) Architecture: En beskrivelse av delkretsens interne virkemåte



Eksempel II



Interne signaler kan defineres lokalt inne i "architecture" blokken

Deklarasjon av internt signal

NB: Selv om program-linjene er sekvensielle definerer de et parallelt (konkurrent) system

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity delkrets2 is
    Port ( a : in std_logic;
           b : in std_logic;
           c : in std_logic;
           d : in std_logic;
           t : out std_logic;
           u : out std_logic);
end delkrets2;

architecture Behavioral of delkrets2 is

signal mellomsignal : std_logic;
begin

    mellomsignal <= (a xor b) and c;
               t <= mellomsignal or b;
               u <= not(d);

end Behavioral;
```


M.H

Kondisjonell beskrivelse

Kondisjonell beskrivelse – when/else:

Syntaks:

```
Utgang <= verdi1 when test1 else
          verdi2 when test2 else
          verdi3;
```

NB: alle tilstander bør defineres. Hvis ikke - har man mindre kontroll på det syntetiserte resultatet*

Hvis flere tester slår til vil første tilordning være gjeldende (prioritet)

*Noen synteseverktøy implementerer latcher (kan holde utgangsverdien), andre ikke

M.H

Kondisjonell beskrivelse

Eksempel 4-input MUX:

```
entity muxen is
    Port ( in1 : in std_logic;
           in2 : in std_logic;
           in3 : in std_logic;
           in4 : in std_logic;
           sel : in std_logic_vector(1 downto 0);
           y   : out std_logic);
end muxen;

architecture Behavioral of muxen is
begin
   y <= in1 when sel="00" else
        in2 when sel="01" else
        in3 when sel="10" else
        in4;

end Behavioral;
```


M.H

"Process"

Mekanismen "process" gir mulighet for sekvensiell beskrivelse

En "process" beskriver en konkurrent enhet med mulighet for intern sekvensiell oppførsel

En "process" kan spesifiseres med en sensitivitets-liste. Systemet processen beskriver (kan man tenke seg) starter opp hver gang en av variablene i listen forandrer verdi

Et større system kan inneholde flere parallelle processer - Analogi: "event driven" programmering

M.H

"Process" – D latch eksempel

Beskrivelse av en D latch

```
entity Dlatch is
    Port ( D   : in std_logic;
           clk : in std_logic;
           Q   : out std_logic);
end Dlatch;

architecture Behavioral of Dlatch is
begin

   process(clk,D)
   begin

      if (clk='1') then
         Q <= D;
      end if;

   end process;

end Behavioral;
```

Sensitivitets-liste

Processen starter opp hver gang clk eller D forandrer verdi (event)

Når clk går til '1' overføres verdien på D til Q

Når D forandres og clk='1' overføres verdien på D til Q (transparent latch).

M.H

"Process" – D flip-flop eksempel

Beskrivelse av en D flip-flop

```
entity Dflipp is
    Port ( D   : in std_logic;
           clk : in std_logic;
           Q   : out std_logic);
end Dflipp;

architecture Behavioral of Dflipp is
begin

   process(clk)
   begin

      if (clk='1' and clk'event) then
         Q <= D;
      end if;

   end process;

end Behavioral;
```

Processen starter opp hver gang clk forandrer verdi

clk'event slår til når clk forandrer verdi

clk'event kan synes unødvendig, men er viktig for å garantere syntese av flip-flop og ikke latch

Merk at på positiv clk-flanke overføres verdien på D til Q. Ellers skjer det ikke noe

M.H

"Process" – MUX eksempel

```
entity Mux2 is
    Port ( A   : in std_logic;
           B   : in std_logic;
           Sel : in std_logic;
           Q   : out std_logic);
end Mux2;

architecture Behavioral of Mux2 is
begin

   process(A,B,Sel)
   begin
      if Sel='0' then
         Q <= A;
      else
         Q <= B;
      end if;

   end process;

end Behavioral;
```

Beskrivelse av en 2-1 MUX

Når Sel går til '0' overføres verdien på A til Q.

Når Sel går til '1' overføres verdien på B til Q.

Når verdien på A eller B forandres overføres selektert verdi til Q

Generelt: Når alle input-signaler er spesifisert i sensitivitets-listen og alle utganger alltid gis en verdi, syntetiseres en kombinatorisk krets

M.H

Generelt om processer

En process kan beskrive både kombinatoriske kretser (med hukommelse) og sekvensielle kretser

Vil man beskrive en kombinatorisk krets:

- Ta med alle inngangssignaler i sensitivitets-listen
- Pass på at utgangssignalene alltid blir tilordnet en verdi

Vil man beskrive en sekvensiell krets:

- Latcher bør generelt unngås: bruk flip-flop's ved å spesifisere flanketrigging ('event)
- Forsøk å gjøre den sekvensielle delen så liten og oversiktlig som mulig ved å "trekke" kombinatoriske element ut av processen

M.H

Mer om kondisjonelle tester

For kondisjonell testing innen processer brukes ofte "case"

Syntaks:

```
case : signalet is
when  verdi1 =>
     uttrykk1;
when  verdi2 =>
     uttrykk2;
when  others =>
     uttrykk3;
end case;
```

Husk å spesifisere alle mulige verdier til signalet, evt. med "others"

Pass på at verdi1, verdi2 er forskjellige

M.H

Tidspunkt for signaltilordning i processer

Viktig å vite: Inne i en "process" vil alle signaltilordninger (<=) utføres først i det øyeblikket processen terminerer ("end process" linjen utføres)

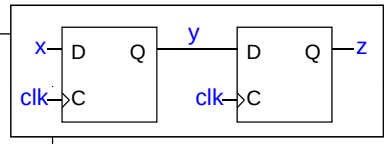
```
entity pipeline is
    Port ( x   : in std_logic;
           clk : in std_logic;
           z   : out std_logic);
end pipeline;

architecture Behavioral of pipeline is

signal y : std_logic;
begin

   process(clk)
   begin
      if clk='1' and clk'event then
      y <= x;
      z <= y;
      end if;
   end process;

end Behavioral;
```



Eksempel: signal z vil ikke nødvendigvis være lik signal x

M.H