IN2060, Høst 2026

Kapittel 5: Digitale Byggeblokker

Philipp Häfliger <hafliger@ifi.uio.no>

24. september 2026

Innhold

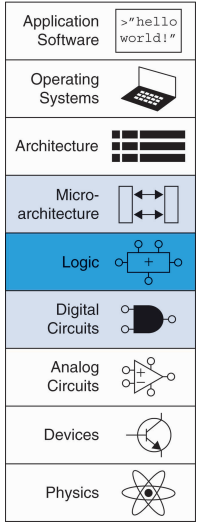
1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
   - Carry Propagate Addere
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

Innhold

1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

Kapittel 5 Delmål

- Forstå hvordan oppbygningen av kretser kan føre til tidsforsinkelser (propagation delay)
- Kjenne til og kunne beskrive virkemåte til noen digitale byggeblokker (som er vesentlige innen prosessorarkitekturer og programmerbar logikk):
  - Addere
  - Shiftregistere
  - Tellere
  - Minneelementer
  - ALU
  - Programmerbar logikk (PL)



Ressursser for delmålet

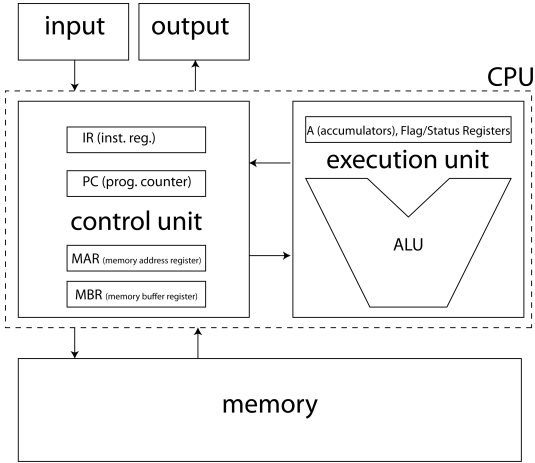
- Forelesning & selvstudium
  - Aritmetiske kretser
    - Addere, ALU, Shiftere
  - Sekvensielle byggeblokker
    - Tellere, shiftregistere
  - Minne
    - RAM, ROM
  - Programmerbar logikk
    - ROM logikk (LUTer), PLA, Logiske blokker
- Ukeoppgaver
  - Øvelse i å tolke og beskrive virkemåte til digitale byggeblokker i form av skjema, HDL og tekst.
- Oblig
  - CLA-adder funksjon og oppbygging
- Gruppetimer
  - HDL eksempler
  - Gjenomgang av oppgaver

Innhold

1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

Von Neumann Data-Arkitektur

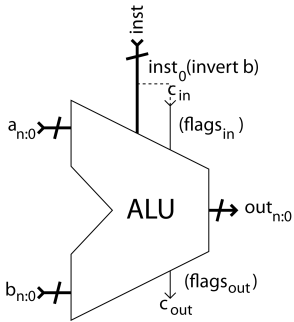
Hvor vi skal med alt dette:

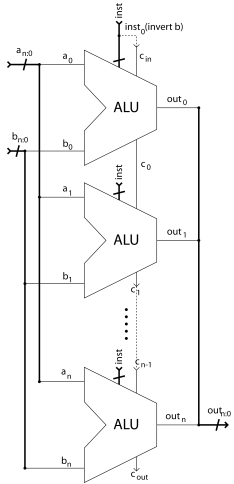


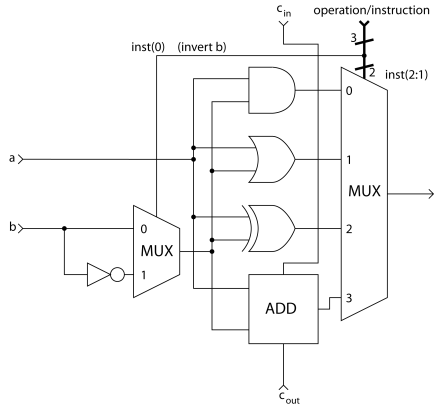
Innhold

1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
   - Carry Propagate Addere
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

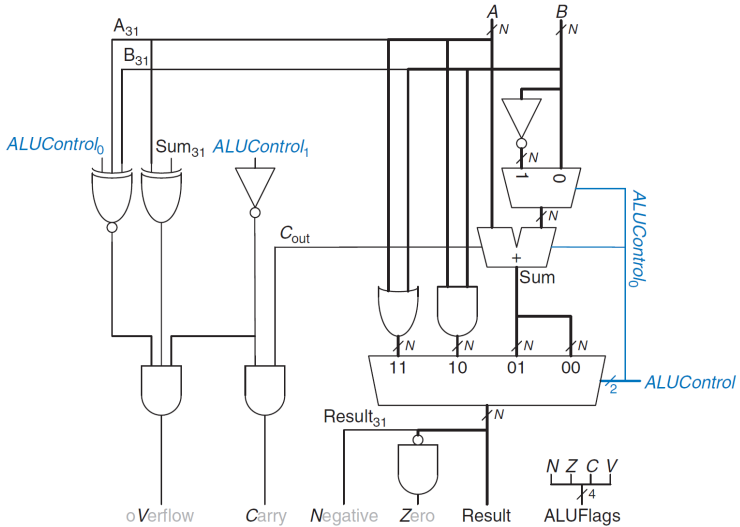
Arithmetic logical unit (ALU)





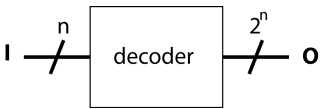


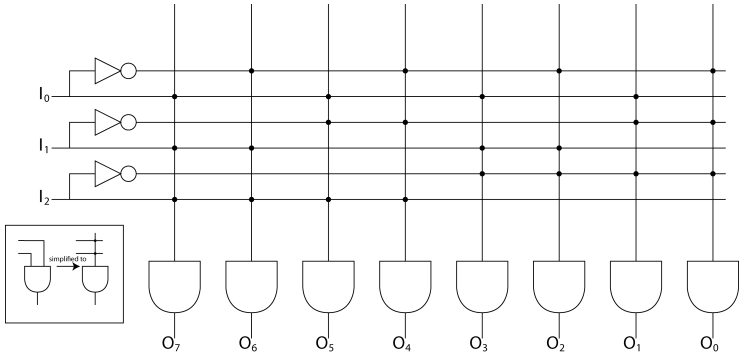
ALU med flagger



Decoder

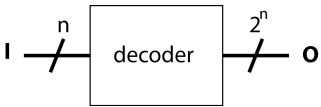
Binary (unsigned) to one-hot

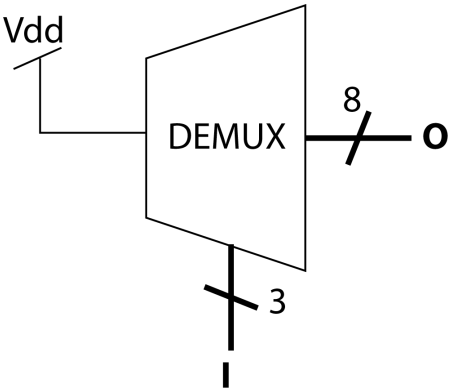




Decoder

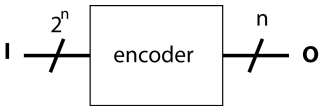
Binary (unsigned) to one-hot

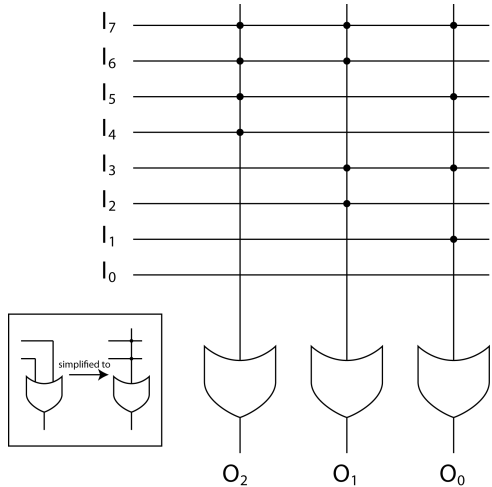




Encoder

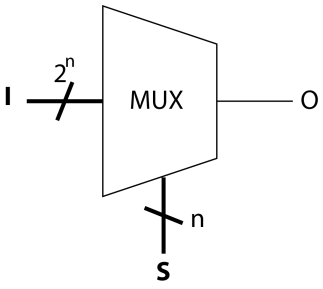
One-hot to binary (unsigned)

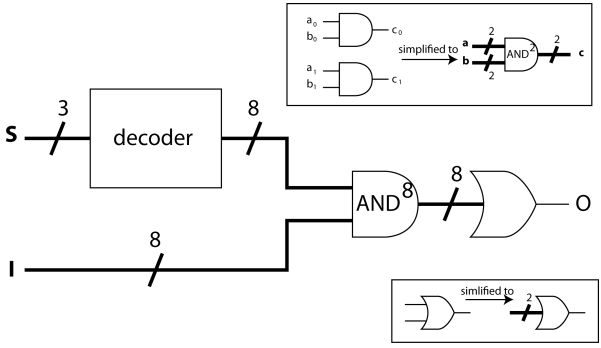




Multiplexer

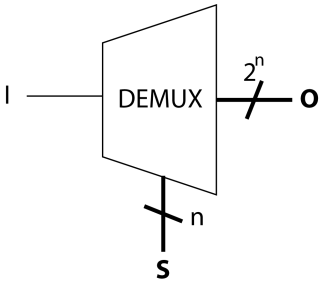
Routing one of 2^n inputs to an output

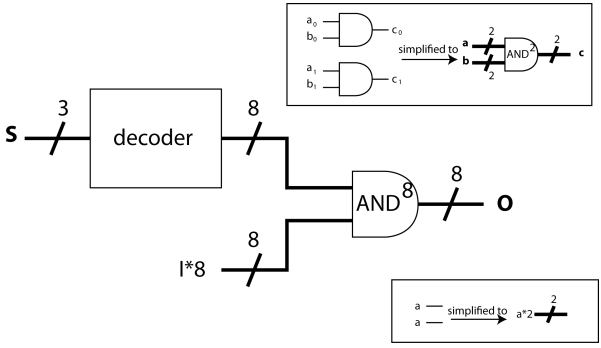




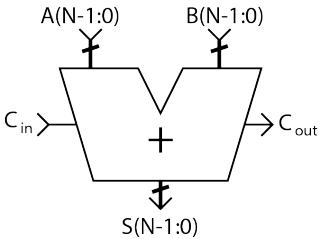
Demultiplexer

Routing an input to one of 2^n outputs.





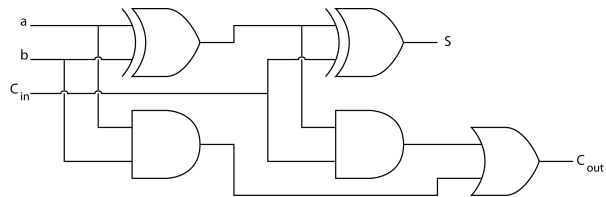
Carry propagate addere




```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD_UNSIGNED.ALL;
entity adder is
  generic(N: integer := 8);
  port(a, b: in STD_ULOGIC_VECTOR(N-1 downto 0);
     cin: in STD_ULOGIC;
     s: out STD_ULOGIC_VECTOR(N-1 downto 0);
     cout: out STD_LOGIC);
end entity adder;

architecture synth of adder is
  signal result: STD_LOGIC_VECTOR(N downto 0);
begin
  result <= ("0" & a) + ("0" & b) + cin;
  s <= result(N-1 downto 0);
  cout <= result(N);
end architecture synth;
```

fulladder



$S = A \oplus B \oplus C_{in}$
$C_{out} = ((A \oplus B) \wedge) \vee (A \wedge B)$

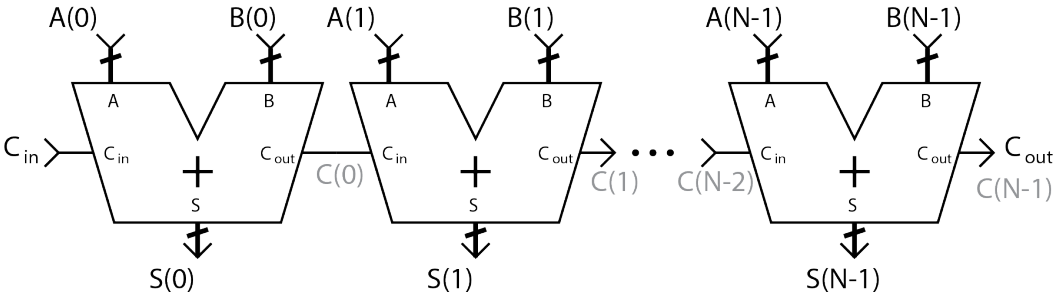
VHDL:

```
S <= A xor B xor Cin;
Cout <= ((A xor B) and Cin) or (A and B);
```

A	B	C_{in}	S	C_{out}
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

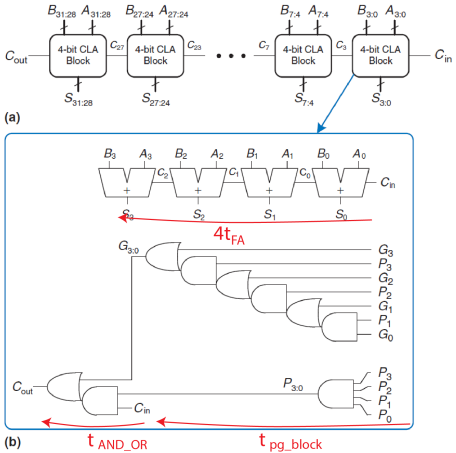
Delay:
$t_{FA} \approx 2t_{gate}$

Ripple carry adder



Delay:
Nt_{FA}

Carry Look Ahead (CLA) adder



$G_i = A_i \wedge B_i$

$P_i = A_i \vee B_i$

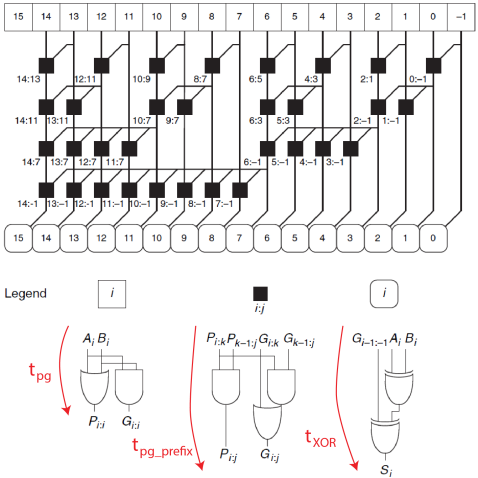
Delay for that: t_{pg}

Delay total (N bits and k length of CLA blocks):

$t_{pg} + t_{pg_block} + \frac{N-1}{k} t_{AND_OR} + kt_{FA}$

Merk: raskere enn ripple carry men skalerer fortsatt lineært med N

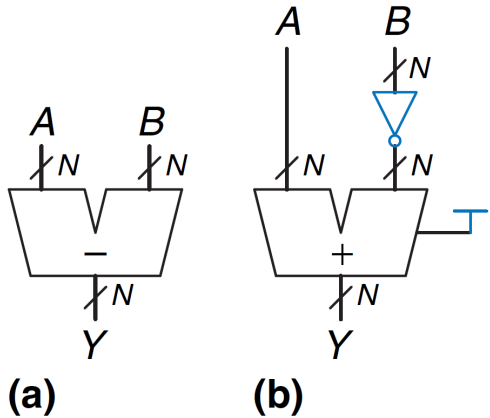
Prefix adder



Beregn C_{in} for alle ledd så fort som mulig:
$C_{in,i} = G_{i:-1}$

Delay total (N bits):
$t_{pg} + \log_2(N)\ t_{pg_prefix} + t_{XOR}$
Skalerer nå logaritmisk med N

Subtraktor

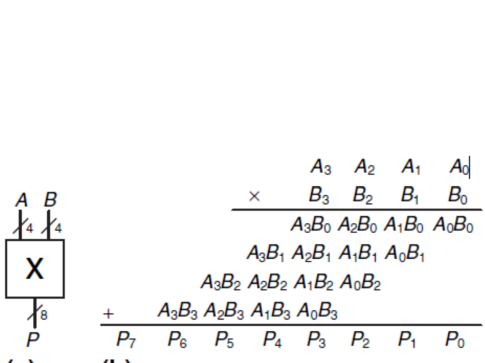


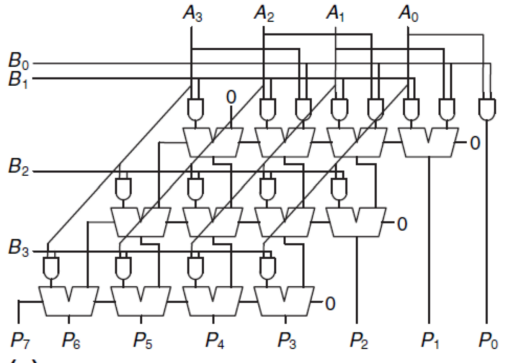
Shiftere

- Skyver bits på en bus (typisk mellom registere) til høyre eller venstre.
- Merk: her snakker vi om de kombinatoriske kretsene, altså ikke skift-registere
- De kan ha noen kontroll-inngangssignaler som bestemmer type shift og antall posisjoner, mest vanligvis med bruk av multipleksere.

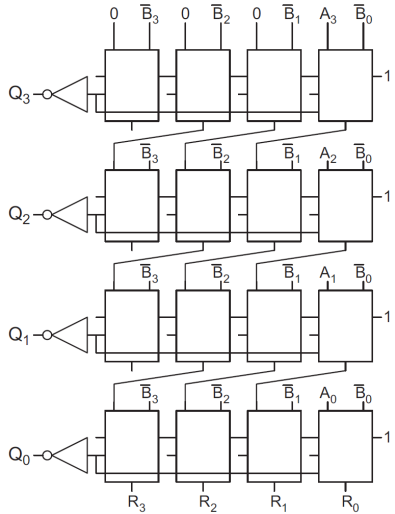
1. **logiske shiftere** fyller bits som 'kommer inn' med '0'.
2. **aritmetiske shiftere** ved et høyreshift fyller bits som kommer inn med verdien som står i MSB. For to-komplemnt-tall tilsvarer det en deling med 2 per 1-posisjons-shift. For venstre-shift fylles det på med '0', som i logisk shift. For to-komplemnt-tall tilsvarer det en multiplikasjon med to per 1-posisjons-shift.
3. **rotatorer** fyller bits som kommer inn med bits som har blitt skyvet ut i andre enden.

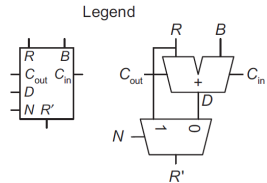
Unsigned Multiplikator





Unsigned Deler

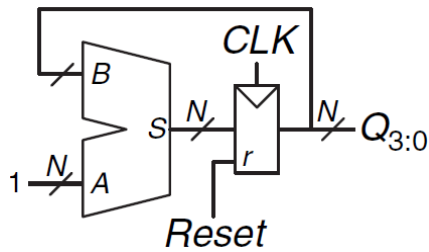




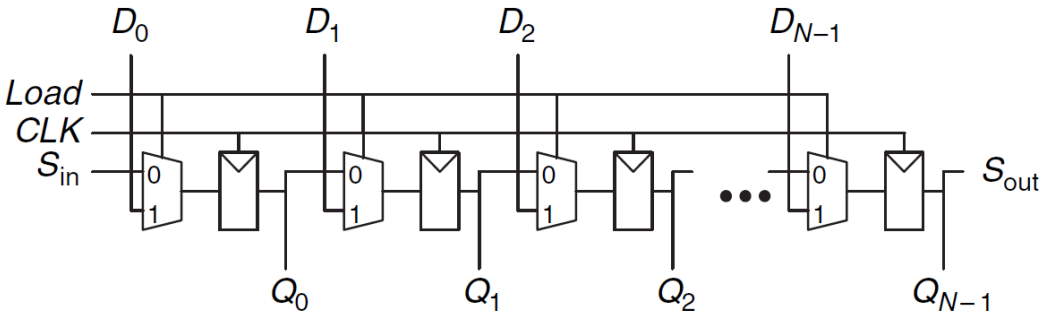
Innhold

1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

Tellere



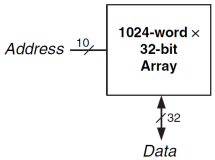
Shift-Registere

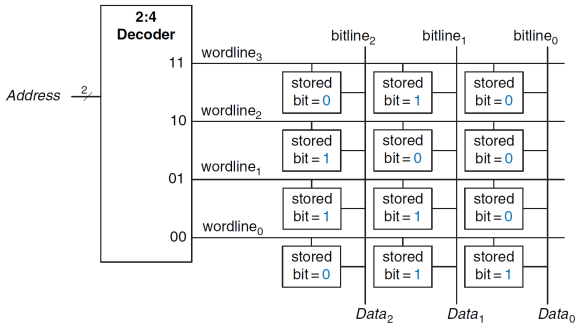


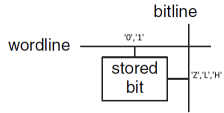
Innhold

1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

Minne



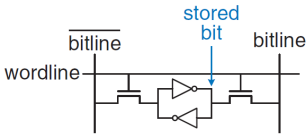




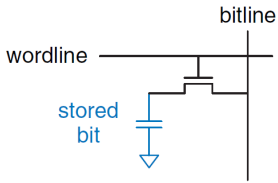
1024 words (depth/address space) and 32 bit word width.

Minnetyper

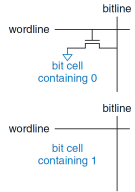
SRAM



DRAM



ROM (PROM, EPROM, EEPROM)



Memory Type	Transistors per Bit Cell	Latency
flip-flop	~20	fast
SRAM	6	medium
DRAM	1	slow

Også: DRAM har ulempen av 'destructive read' og at den lagrete ladningen lekker: det trengs 'refresh', både ved lesing og syklisk.
Også: ROM er 'non-volatile', dvs beholder tilstand også når forsyningen slåss av.

VHDL RAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD_UNSIGNED.ALL;

entity ram_array is
 generic(N: integer := 6; M: integer := 32);
 port(clk,
 we: in STD_LOGIC;
 adr: in STD_LOGIC_VECTOR(N-1 downto 0);
 din: in STD_LOGIC_VECTOR(M-1 downto 0);
 dout: out STD_LOGIC_VECTOR(M-1 downto 0));
end entity ram_array;
```

```
architecture RTL of ram_array is
 type mem_array is array ((2**N-1) downto 0)
  of STD_LOGIC_VECTOR (M-1 downto 0);
 signal mem: mem_array;
begin
 process(clk) begin
  if rising_edge(clk) then
   if we then mem(TO_INTEGER(adr)) <= din;
   end if;
  end if;
 end process;
 dout <= mem(TO_INTEGER(adr));
end architecture RTL;
```

Innhold

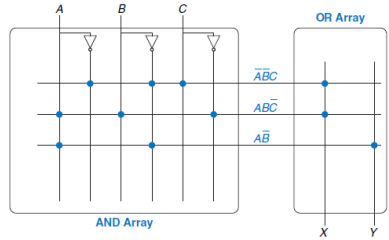
1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

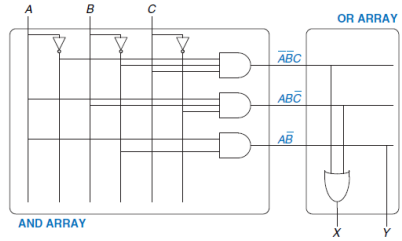
Logic Arrays

Variants: PLA (just combinational), SPLD, CPLD, and most versatile and popular : FPGA

Programmable Logic Array (PLA)

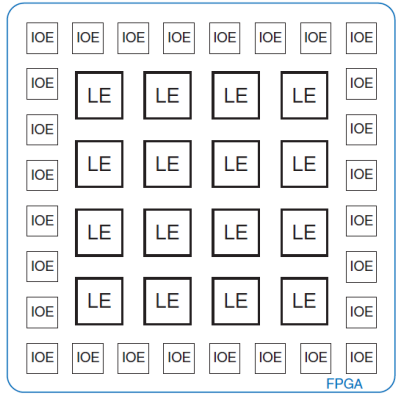
Implementrer programerbare 'sums of products' (SOM).
Her er det: 3 inputs, 3 implicants, 2 outputs





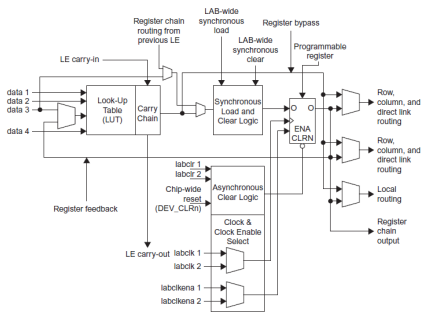
Field Programmable Gate Array (FPGA)

Veldig omfattende programmerbar logikk og kan konfigureres til alt innen kombinatoriske og sekvensielle kretser. De nyere kan ha mange ferdige tilleggselementer, som CPU kjerner og analog digital- og digital analog omformere (ADC/DAC) og selges da som 'system on chip' (SOC). Hvilken som helst VHDL-kode kan implementeres på en (stor nok) FPGA.



Field Programmable Gate Array Logic Element (LE)

(Et inntrykk uten utfyllende forklaring). I senteret er det en *look-up table* (LUT) og en register/flip-flop. LUT direkte implementerer en 'truth table', dvs den er den mest generelle logiske gaten som kan tenkes. Den kan ses på som en mini-mini-minne med 1-bit word width. LUT, Multiplekserne og signal routing mellom LE-ene er *konfigurert* gjennom registere/SRAM som lastes in ved oppstart, enten fra datamaskin, ekstern (EEP)ROM, eller intern EEPROM på FPGA selv. Konfigurasjonen er generert av en HDL-kompilator.

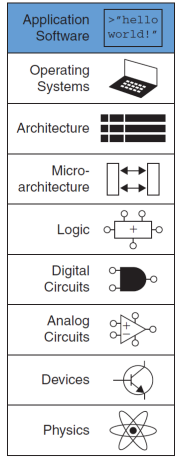


Innhold

1. Kapittel 5 Delmål
2. Von Neumann Data-arkitektur
3. Aritmetiske og andre Vanlige Kombinatoriske Kretser
4. Vanlige Sekvensielle Kretser
5. Minne
6. Logic Arrays
7. Litt om C-Syntaks

Hello World

```
// Write "Hello world!" to the console
#include <stdio.h>
// subroutine libraries' header files ("*.h")
int main(void){
// 'main' takes no parameters ('void') and returns an integer ('int')
 printf("Hello world!\n");
/*
 printf is specified in stdio.h and writes text output.
 "\n" is an 'escape sequence' that stands for the 'newline' character
*/
}
```



Bruk 'gcc filename.c' til å kompilere en C-programkode som inneholder `main` og standard messig blir resultatet en fil `a.exe`

Primitive Datatyper

Type	Size (bits)	Minimum	Maximum
`char`	8	$-2^{-7} = -128$	$2^7 - 1 = 127$
`unsigned char`	8	0	$2^8 - 1 = 255$
`short`	16	$-2^{15} = -32{,}768$	$2^{15} - 1 = 32{,}767$
`unsigned short`	16	0	$2^{16} - 1 = 65{,}535$
`long`	32	$-2^{31} = -2{,}147{,}483{,}648$	$2^{31} - 1 = 2{,}147{,}483{,}647$
`unsigned long`	32	0	$2^{32} - 1 = 4{,}294{,}967{,}295$
`long long`	64	-2^{63}	$2^{63} - 1$
`unsigned long`	64	0	$2^{64} - 1$
`int`	machine-dependent		
`unsigned int`	machine-dependent		
`float`	32	$\pm 2^{-126}$	$\pm 2^{127}$
`double`	64	$\pm 2^{-1023}$	$\pm 2^{1022}$

Kommandolinje argumenter

```
// Convert inches to centimeters
#include <stdio.h>
#include <stdlib.h> // for str2f
#define INCH2CM 2.54
int main(int argc, char** argv) {
 /*
 passing command line arguments as strings
 argc -> number of arguments
 (first is the command itself)
 argv -> array of arguments
 */
 float inch=5.5; // local variable
 float cm; // local variable
 if (argc > 1)
 {
  // sscanf(argv[1],"%f",&inch);
  inch=strtof(argv[1],NULL);
 }
 cm = inch * INCH2CM;
 printf("%f inches = %f cm\n", inch, cm);
}
```

Funksjoner og globale og lokale variabler

```
#include <stdio.h>
/*
Use local variables to find maximum
of 3 numbers and to transfer
it to a global variable and print
*/
int max;
// global variable for maximum value

int getMax(int a, int b, int c) {
 int result = a;
 // local variable for maximum value
 if (b > result) {
  if (c > b) result = c;
  else result = b;
 } else if (c > result) result = c;
 return result;
}

void printMax(int m) {
 printf("The maximum number is: %d\n", m);
}

int main(void) {
 max = getMax(4, 3, 7);
 printMax(max);
}
```

Operators (1/3)

Category	Operator	Description	Example
Unary	++	post-increment	`a++; // a = a+1`
	--	post-decrement	`x--; // x = x–1`
	&	memory address of a variable	`x = &y; // x = the memory // address of y`
	~	bitwise NOT	`z = ~a;`
	!	Boolean NOT	`!x`
	–	negation	`y = -a;`
	++	pre-increment	`++a; // a = a+1`
	--	pre-decrement	`--x; // x = x–1`
	(type)	casts a variable to (type)	`x = (int)c; // cast c to an // int and assign it to x`
	sizeof()	size of a variable or type in bytes	`long int y; x = sizeof(y); // x = 4`

Operators (2/3)

Multiplicative	*	multiplication	`y = x * 12;`
	/	division	`z = 9 / 3; // z = 3`
	%	modulo	`z = 5 % 2; // z = 1`
Additive	+	addition	`y = a + 2;`
	−	subtraction	`y = a - 2;`
Bitwise Shift	<<	bitshift left	`z = 5 << 2; // z = 0b00010100`
	>>	bitshift right	`x = 9 >> 3; // x = 0b00000001`
Relational	==	equals	`y == 2`
	!=	not equals	`x != 7`
	<	less than	`y < 12`
	>	greater than	`val > max`
	<=	less than or equal	`z <= 2`
	>=	greater than or equal	`y >= 10`

Operators (3/3)

Bitwise	&	bitwise AND	`y = a & 15;`
	^	bitwise XOR	`y = 2 ^ 3;`
	|	bitwise OR	`y = a | b;`
Logical	&&	Boolean AND	`x && y`
	||	Boolean OR	`x || y`
Ternary	? :	ternary operator	`y = x ? a : b; // if x is TRUE, // y=a, else y=b`
Assignment	=	assignment	`x = 22;`
	+=	addition and assignment	`y += 3;` `// y = y + 3`
	-=	subtraction and assignment	`z -= 10;` `// z = z - 10`
	*=	multiplication and assignment	`x *= 4;` `// x = x * 4`
	/=	division and assignment	`y /= 10;` `// y = y / 10`
	%=	modulo and assignment	`x %= 4;` `// x = x % 4`
	>>=	bitwise right-shift and assignment	`x >>= 5;` `// x = x>>5`
	<<=	bitwise left-shift and assignment	`x <<= 2;` `// x = x<<2`
	&=	bitwise AND and assignment	`y &= 15;` `// y = y & 15`
	|=	bitwise OR and assignment	`x |= y;` `// x = x | y`
	^=	bitwise XOR and assignment	`x ^= y;` `// x = x ^ y`