EE1410: Digitale Systemen

BSc. EE, 1e jaar, 2012-2013, 8e hoorcollege

Arjan van Genderen, Stephan Wong, Computer Engineering

3-5-2013

Hoorcollege 8

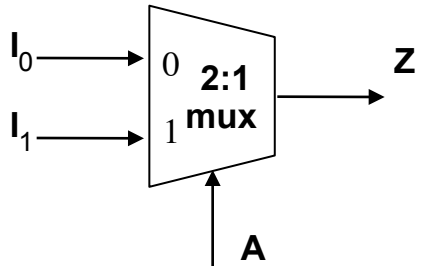
- Combinatorische modules
- Sequentiële modules
- Getalsystemen
- Arithmetische modules

Corresponderende stof in boek “Digital Logic”:
6 – 6.3 (niet 6.1.2), 7.8 – 7.9, 5.2 – 5.3, 5.6 – 5.6.1

Combinatorische modules

Multiplexers

2^n input bits naar 1 output bit, geselecteerd met *n* control bits

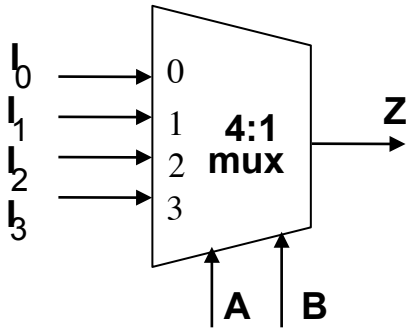


I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$Z = A' I_0 + A I_1$

Logische vorm:

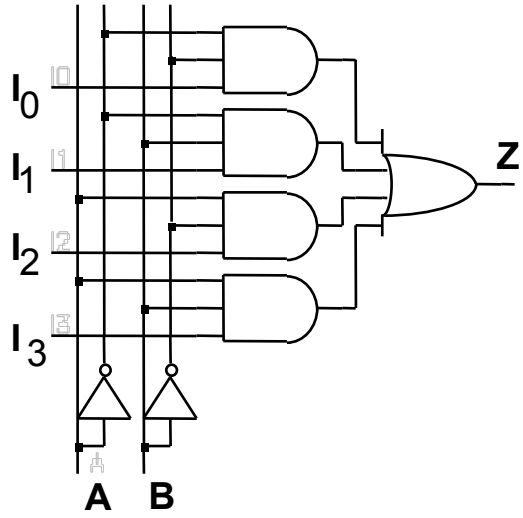
A	Z
0	I_0
1	I_1



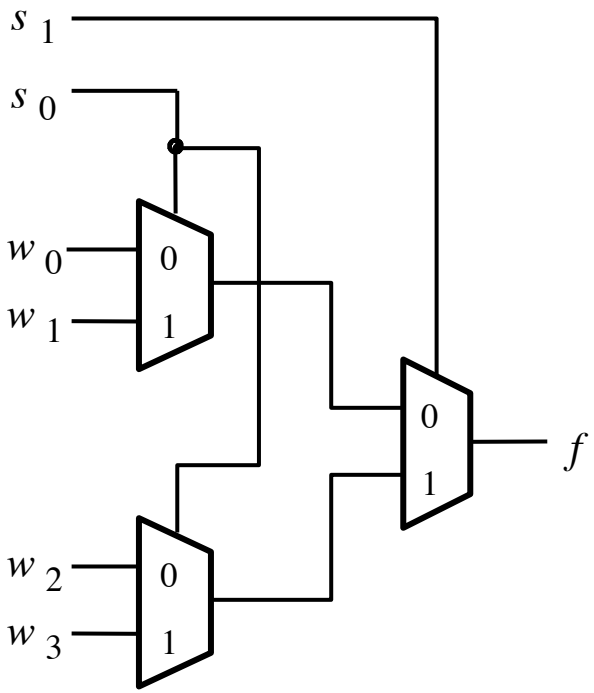
A	B	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$Z = A' B' I_0 + A' B I_1 + A B' I_2 + A B I_3$

Mogelijke implementatie



Multiplexers



Een 4-to-1 multiplexer geimplementeerd met twee 2-to-1 multiplexers

Multiplexers

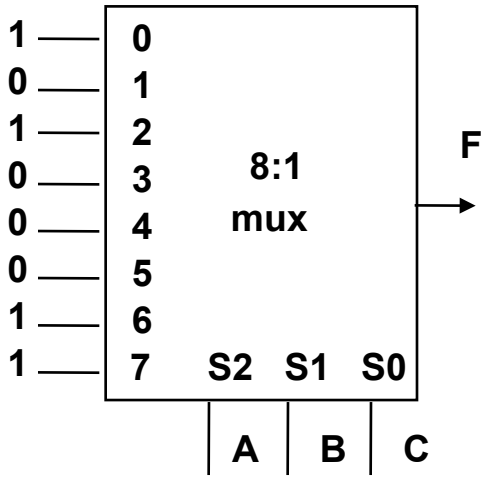
2^{n-1}:1 MUX kan elke functie van *n* variabelen implementeren:

Voorbeeld: $F(A,B,C) = m_0 + m_2 + m_6 + m_7$

$= A'\,B'\,C' + A'\,B\,C' + A\,B\,C' + A\,B\,C$

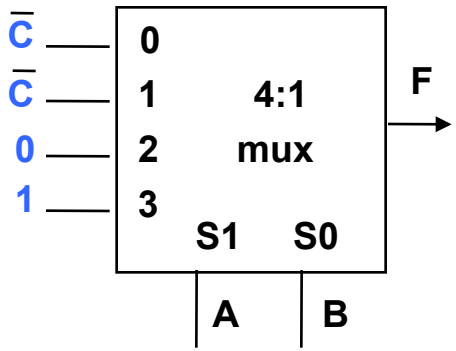
$= A'\,B'\,C' + A'\,B\,C' + A\,B$

$= A'\,B'\,(C') + A'\,B\,(C') + A\,B'\,(0) + A\,B\,(1)$



"Lookup Table"

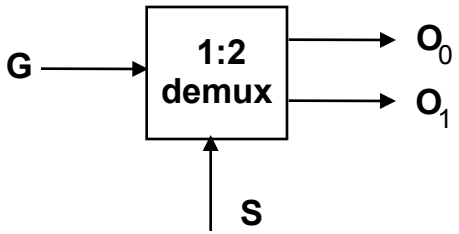
A	B	C	F	
0	0	0	1	$\overline{C}$
0	0	1	0	
0	1	0	1	$\overline{C}$
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



Demultiplexer / Decoder

Demultiplexer: 1 input bit naar 2^n output bits, geselecteerd met *n* control bits

1:2 Demux



$O_0 = G\ S'$
$O_1 = G\ S$

S_0	O_1	O_0
0	0	G
1	G	0

1:4 Demux

$O_0 = G\ S_1'\ S_0'$

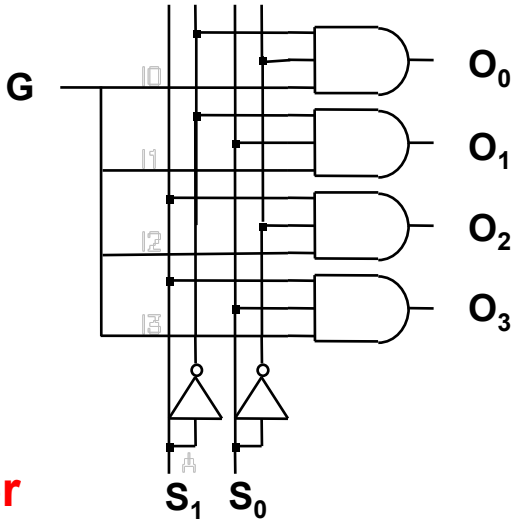
$O_1 = G\ S_1'\ S_0$

$O_2 = G\ S_1\ S_0'$

$O_3 = G\ S_1\ S_0$

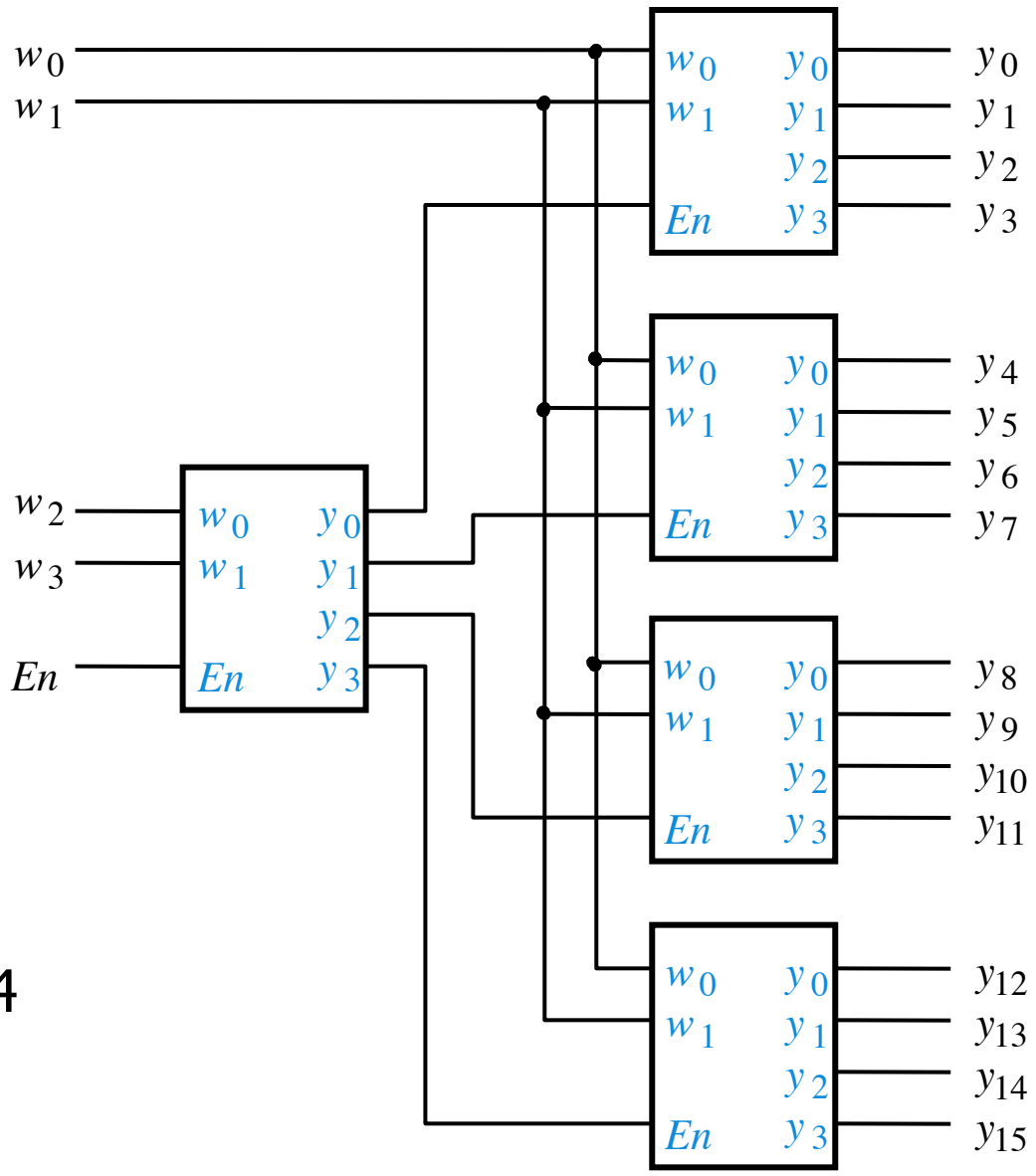
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	G
0	1	0	0	G	0
1	0	0	G	0	0
1	1	G	0	0	0

Mogelijke implementatie



Indien G geen data voorstelt maar enable signaal (of altijd 1 is), dan heet demultiplexer een decoder

Decoder

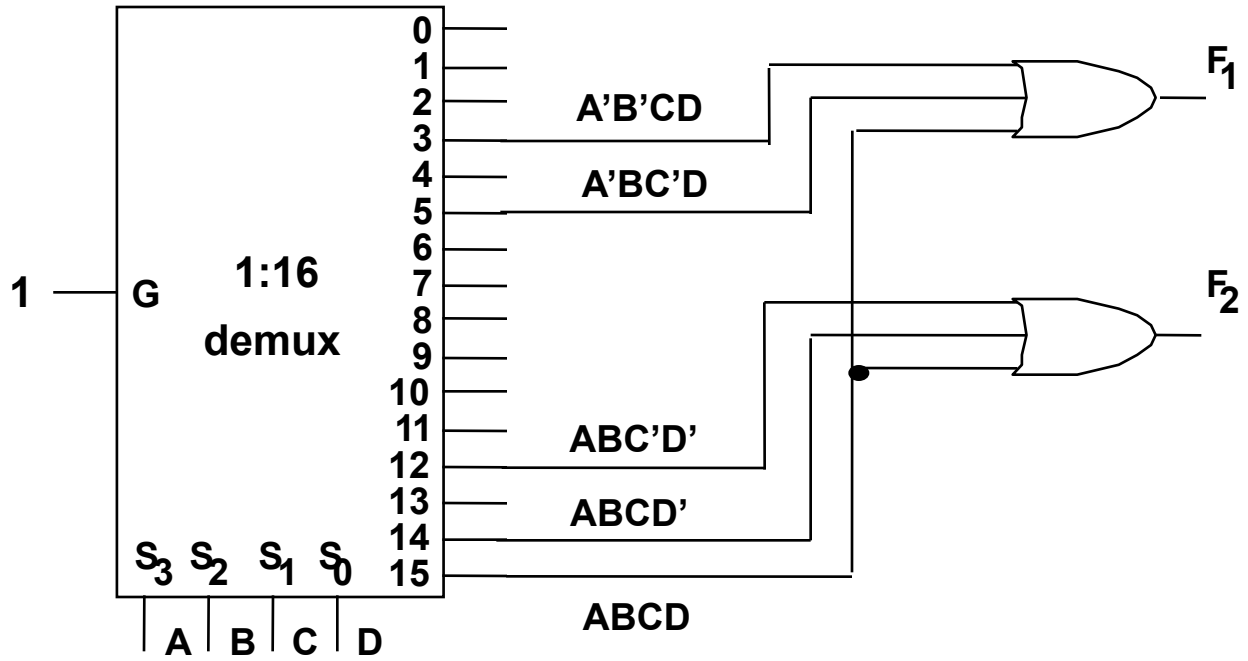


Een 1-to-16 decoder opgebouwd uit 1-to-4 decoders met enable ingang

Decoder

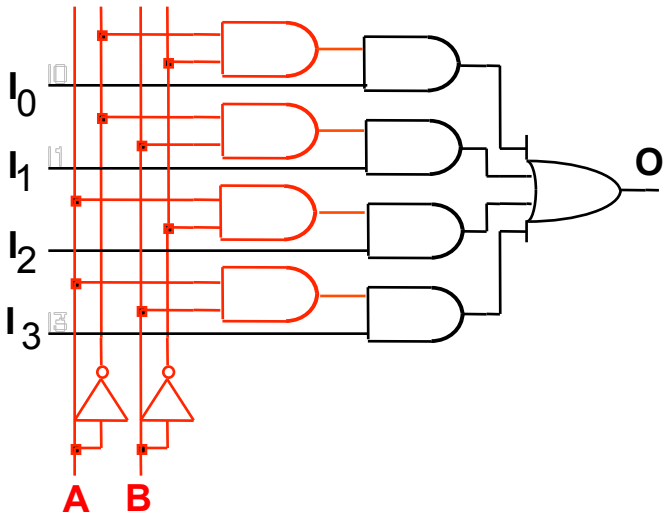
Decoder makkelijke manier om mintermen te genereren:

Voorbeeld: $F_1 = A'\,B\,C'\,D + A'\,B'\,C\,D + A\,B\,C\,D$
$F_2 = A\,B\,C'\,D' + A\,B\,C$

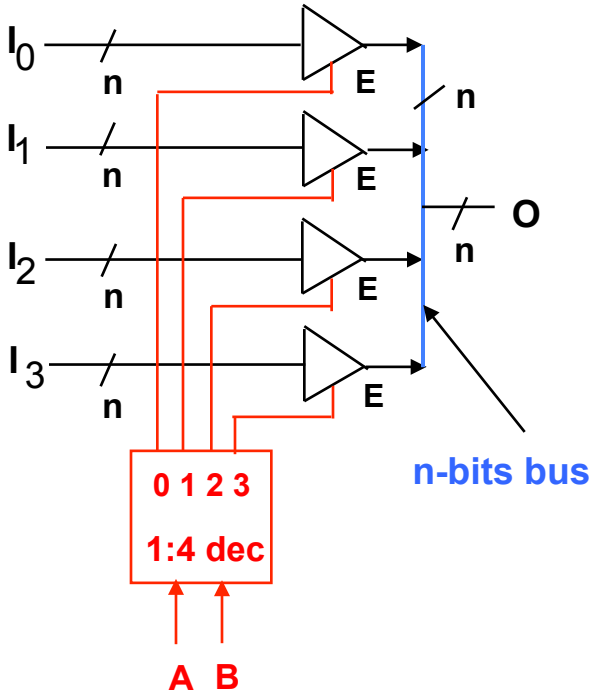


Multiplexer m.b.v. decoder

MUX opgebouwd met decoder en AND-OR schakeling



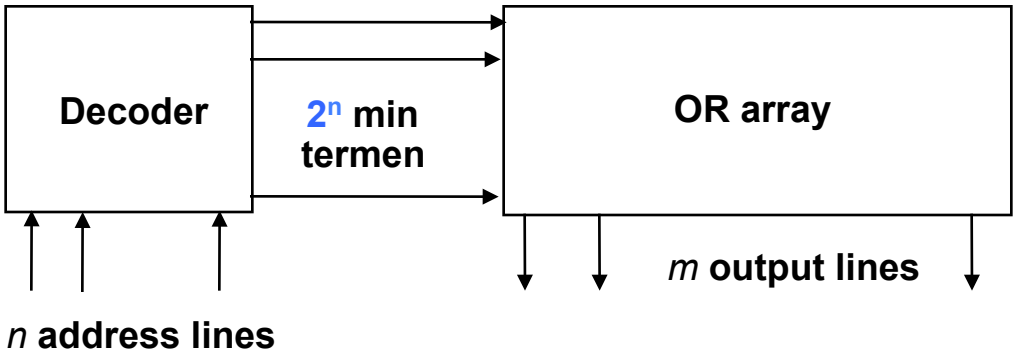
AND-OR schakeling vervangen door *bus* met *tri-state* buffers:



***n*-bit multiplexing**

Read-Only Memories

ROM: een PLA waarbij het AND array een volledige decoder is (2^n mintermen!) en waarbij alleen het OR array wordt geprogrammeerd



Oftwel het OR array:

- **levert de waarheidstabel voor alle m uitgangen**
- **is een tabel van 2^n bitvectoren (index = "adres"; bitvector = "woord")**
- **is een memory array voor 2^n woorden van m bits**

Logische Functies m.b.v. ROM

Implementatie van logische functies m.b.v. een ROM

Voorbeeld:

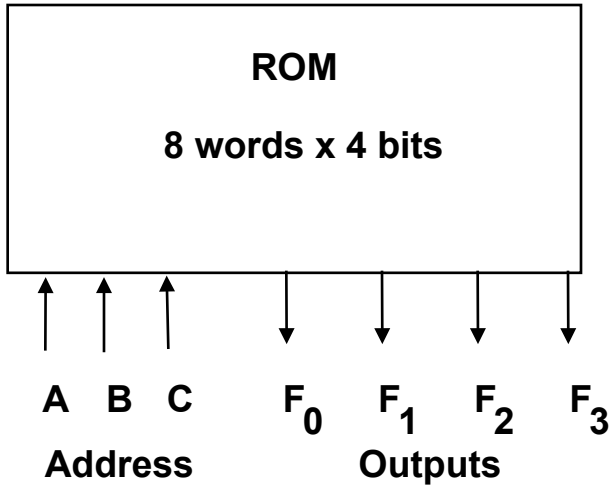
$F_0 = A'\,B'\,C + A\,B'\,C' + A\,B'\,C$

$F_1 = A'\,B'\,C + A'\,B\,C' + A\,B\,C$

$F_2 = A'\,B'\,C' + A'\,B'\,C + A\,B'\,C'$

$F_3 = A'\,B\,C + A\,B'\,C' + A\,B\,C'$

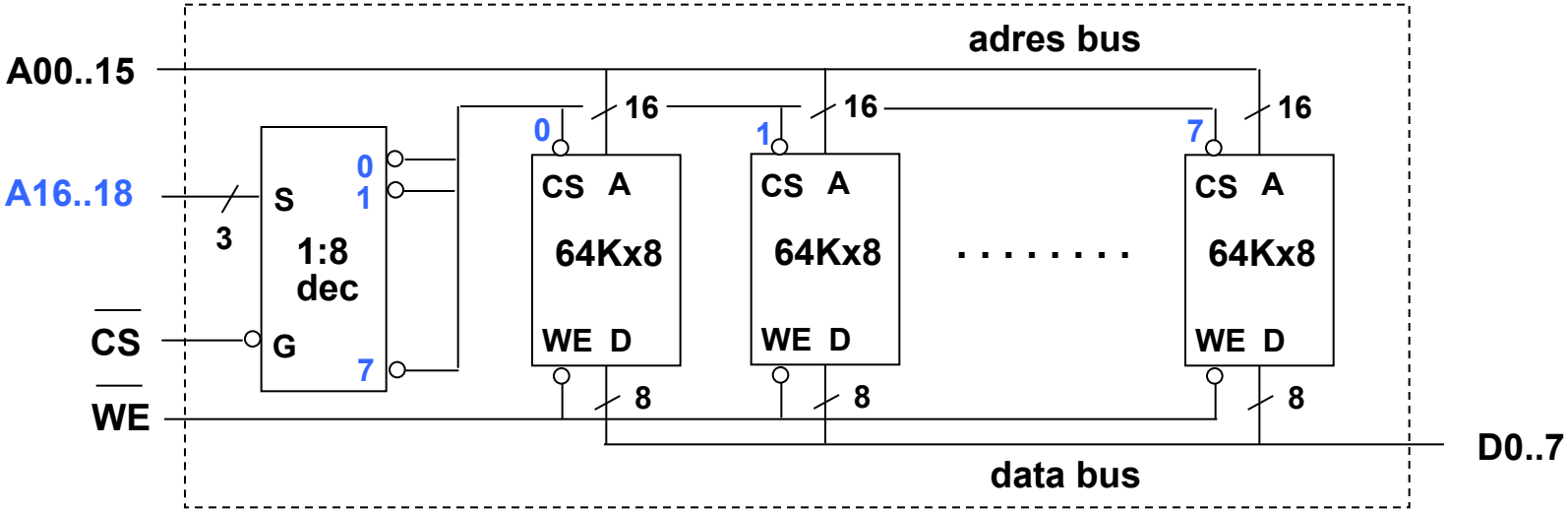
Address			Word Contents			
A	B	C	F_0	F_1	F_2	F_3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0



Geheugenuitbreiding met decoder

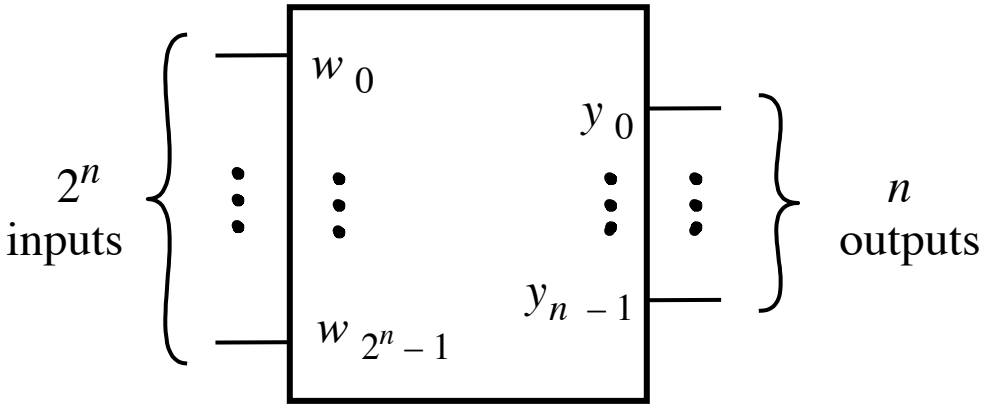
Grotere RAM = parallelschakeling kleinere RAM banken + decoder:

512Kx8 = 8 x 64Kx8 (met 1:8 DEC)



D uitgang NIET in tri-state (d.w.z. bank geselecteerd om uitgelezen te worden) als CS = 1 en WE = 0 (oftewel CS' = 0 en WE' = 1)

Encoder



Een 2^n-to-n binary encoder
(inverse functie van een decoder)

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Waarheidstabel (veronderstelt one-hot encoding aan ingang)

Priority Encoder

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Waarheidstabel van een "priority encoder" (z geeft aan of er sowieso wel een 1 in de ingang zit)

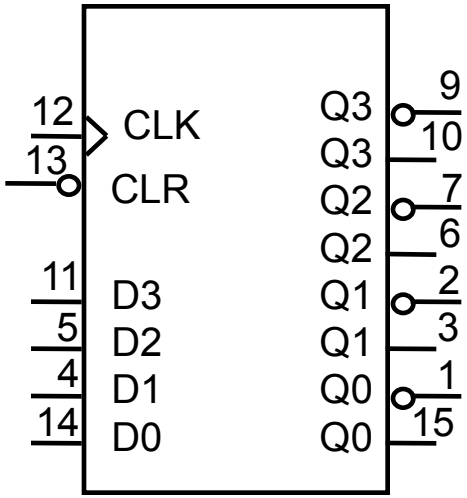
Sequentiële modules

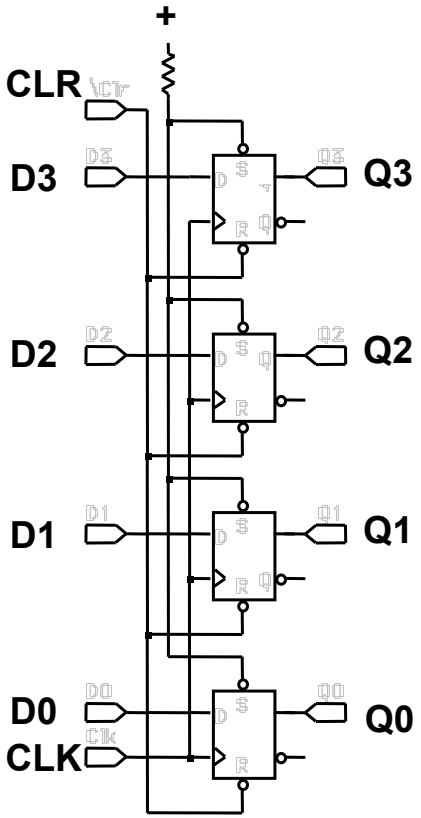
Data-register

Register = groep D-FFs die gezamenlijk worden bestuurd (CLK, CLR, ...)

Voorbeeld:

4-bit register (“quad D reg”) met clock en asynchrone clear (CLR, active low):

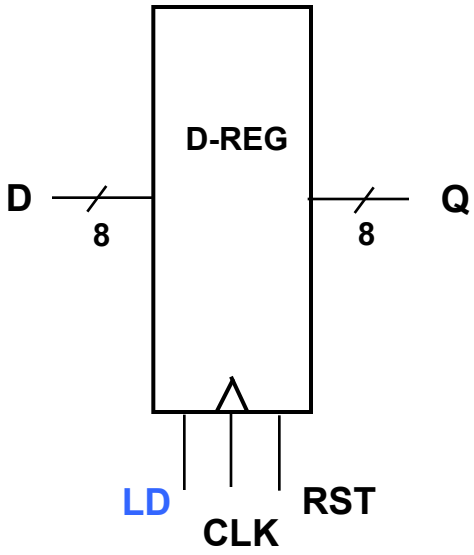


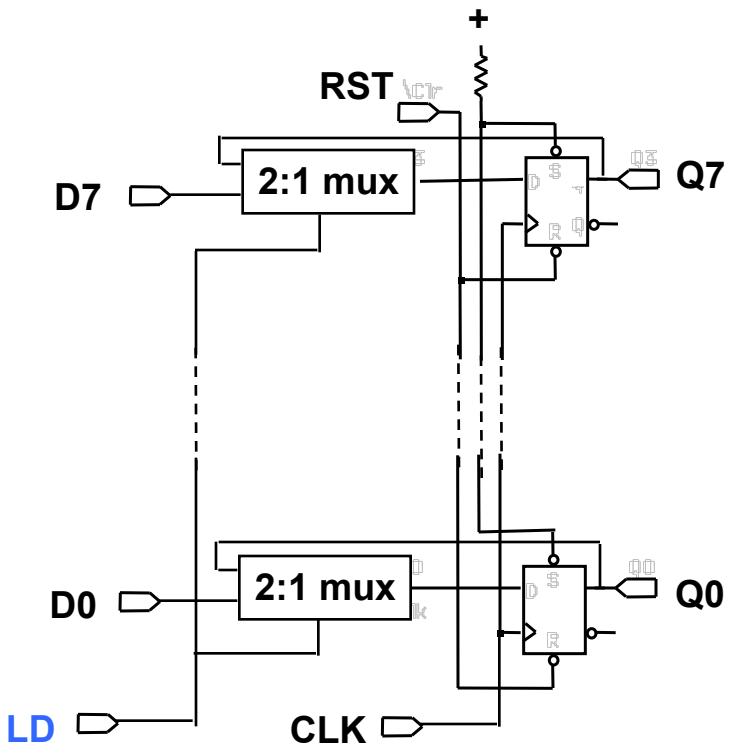


Data-register met load/hold mode

Ander voorbeeld:

8-bit D-register met asynchrone reset (RST) en synchrone load (LD):





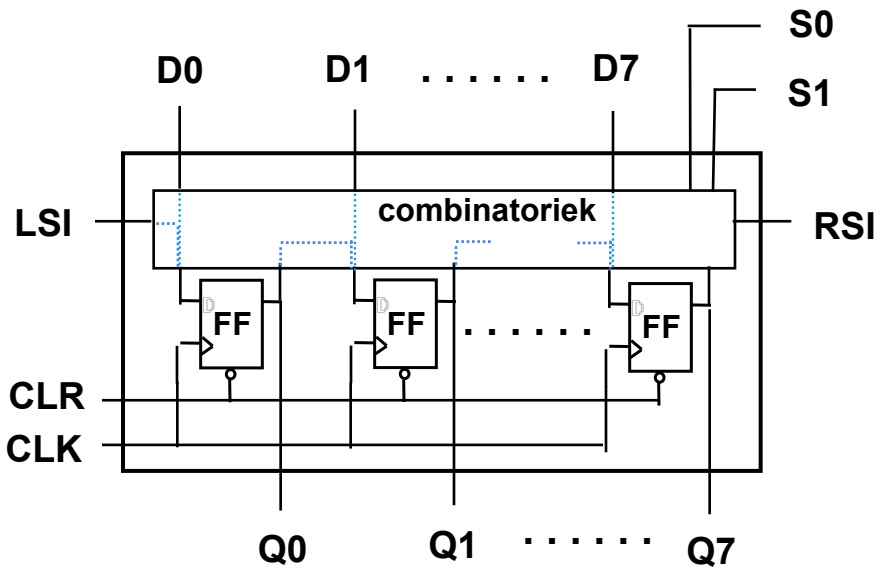
LD = 0: hold mode ($Q^+ = Q$)
LD = 1: load mode ($Q^+ = D$)

Schuif-register

Schuif-register = D-reg met data permutatie-mogelijkheden

voorbeeld:

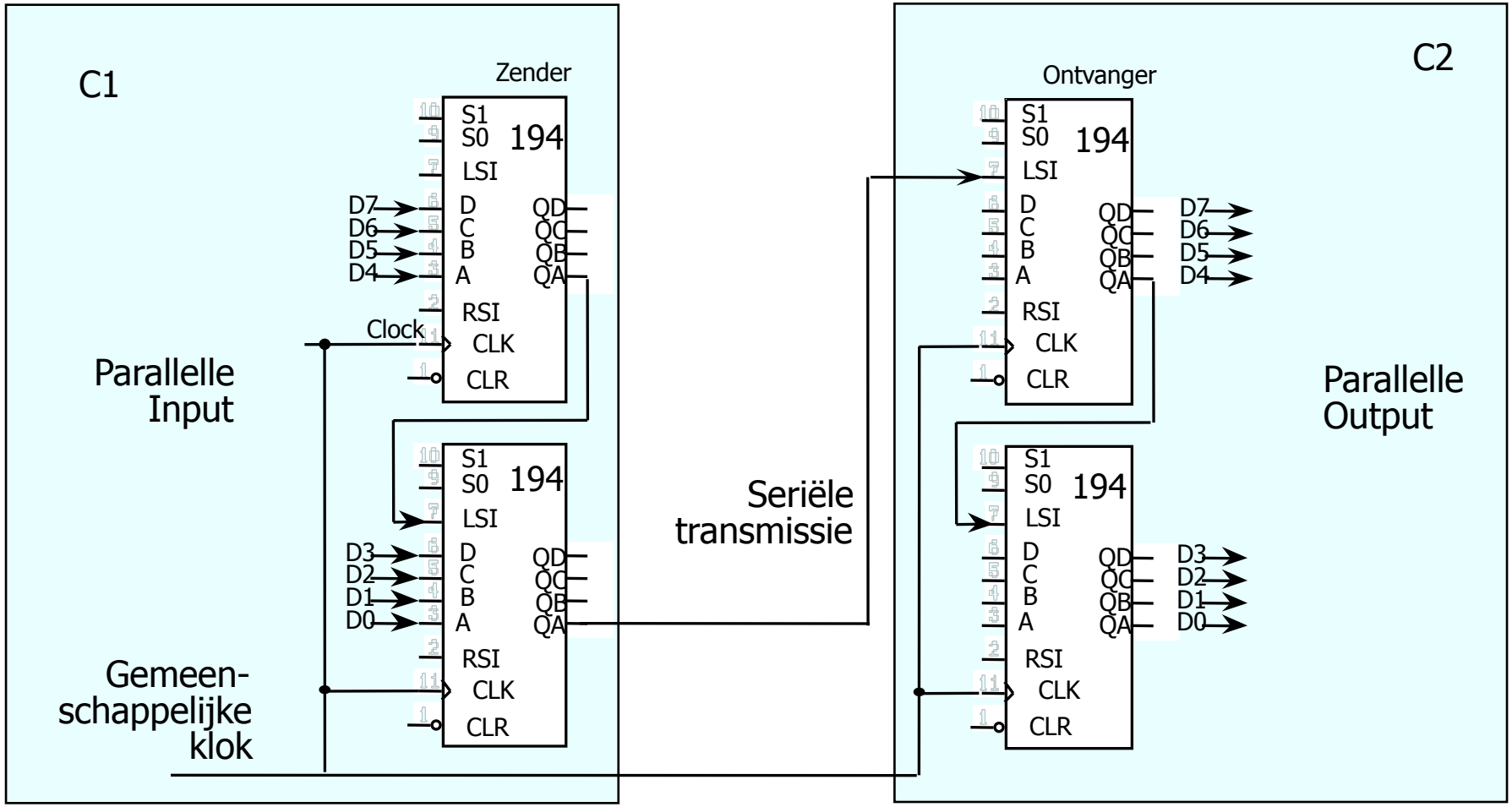
8-bit shuif-register met asynchrone clear (CLR) en vier synchrone functie-modes (S1 S0):



S1 S0:	Functie:
0 0	hold
0 1	shift right
1 0	shift left
1 1	load

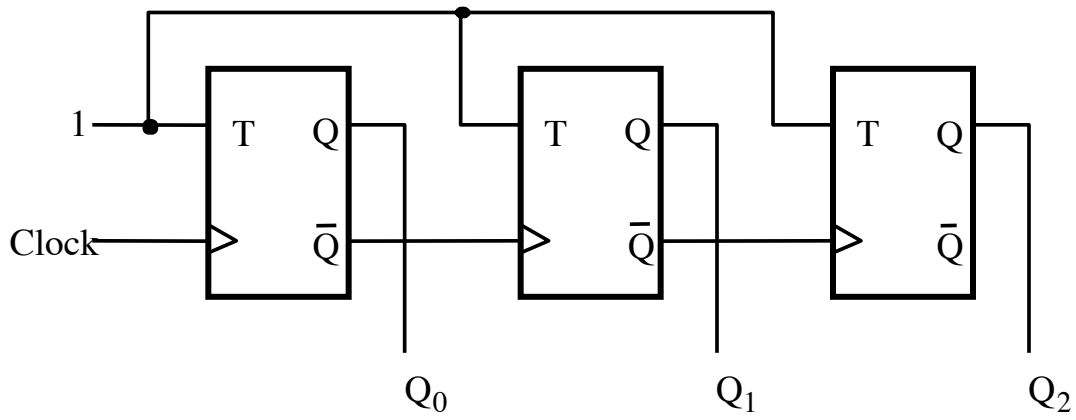
Schuifregisters geschikt voor serie-parallel convertors, zoals bij terminal-computer communicatie-verbindingen

Seriële communicatie met schuifregisters

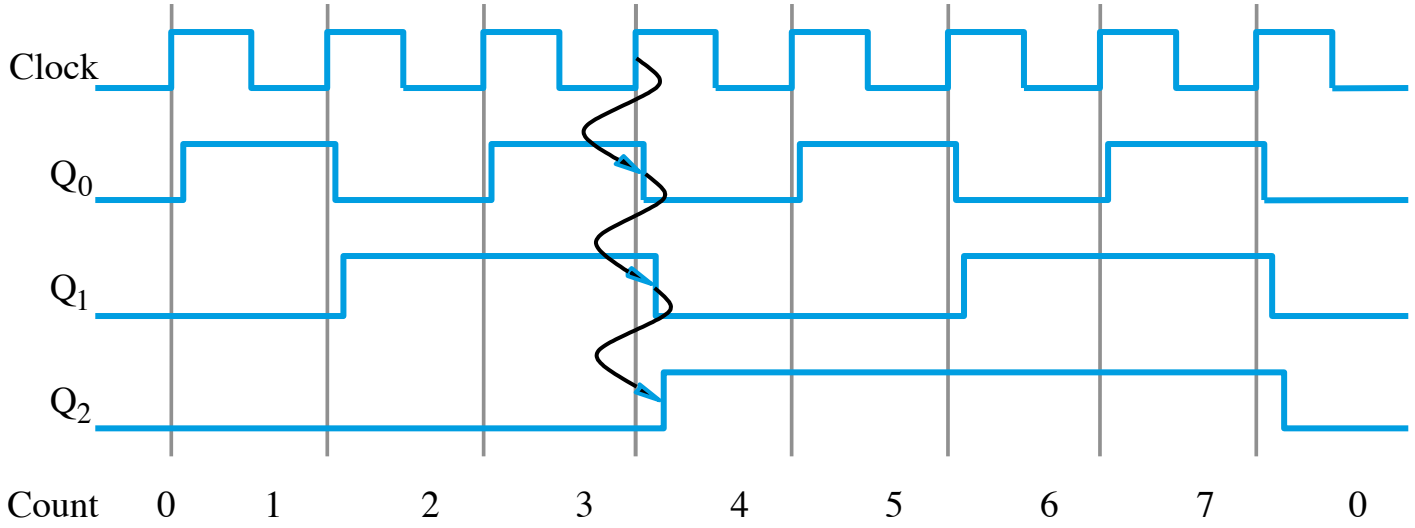


1 klokpuls voor links parallel inladen, daarna worden met 8 klokpulsen 8 bits serieel overgestuurd, daarna rechts parallel uit te lezen.

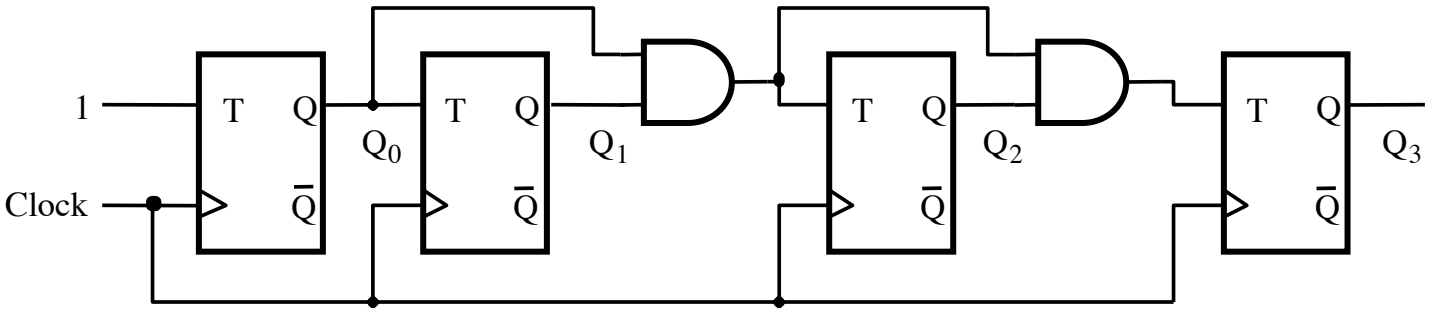
Asynchrone tellers



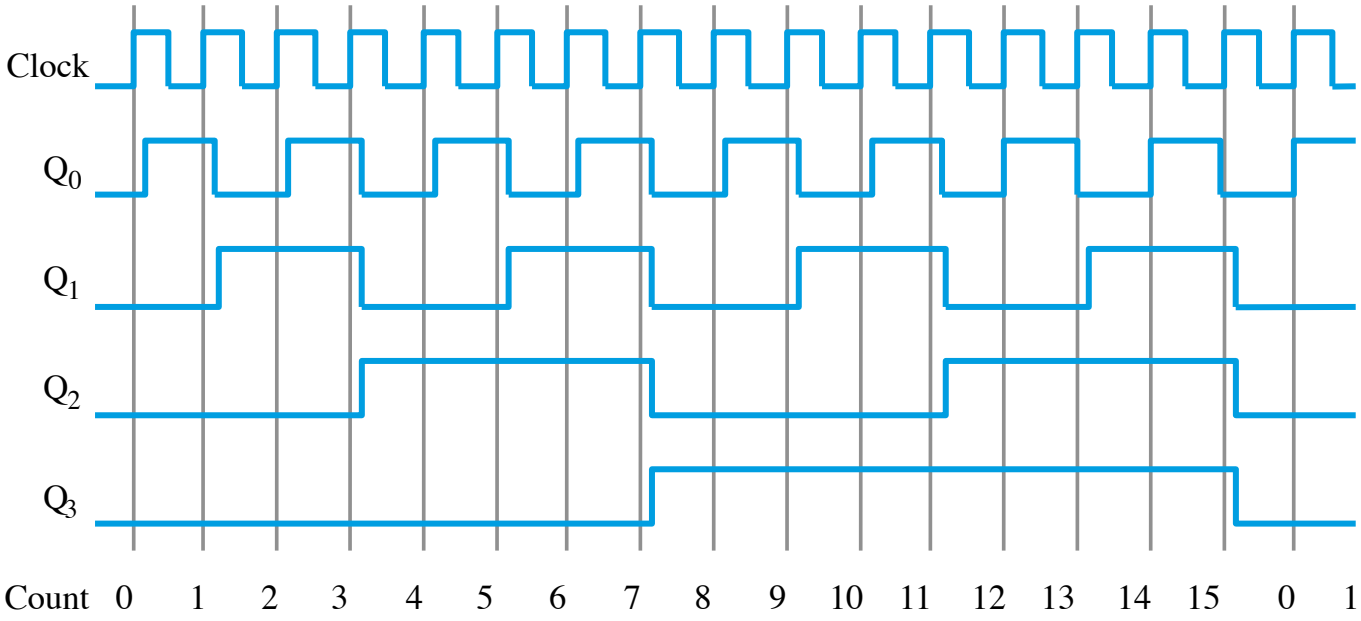
Uitgang flip-flop is kloksignaal voor buurman.



Synchrone tellers



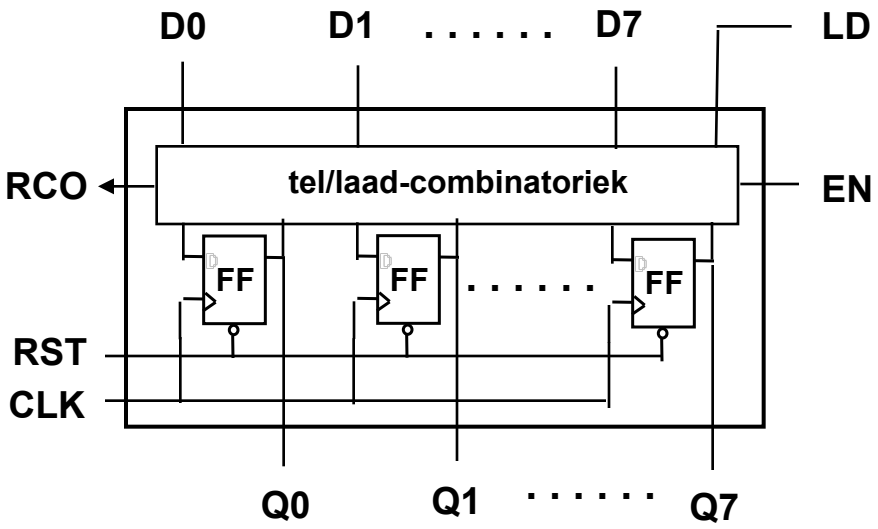
Alle flip-flops op hetzelfde kloksignaal.



Teller modules combineren

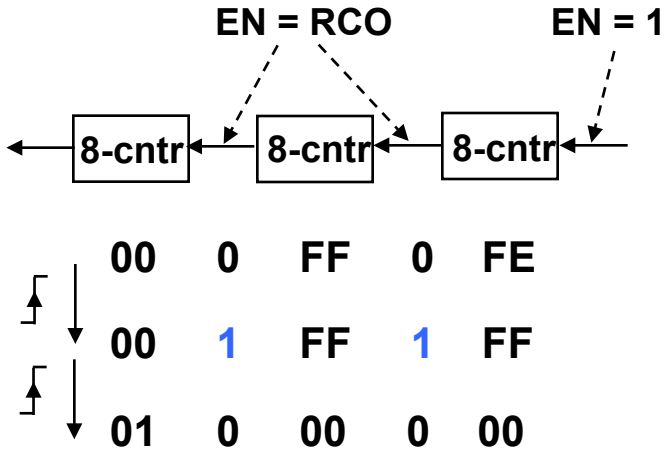
Voorbeeld:

8-bit teller met asynchrone reset (RST) en synchrone load (LD) + enable (EN), en met ripple carry output (RCO)



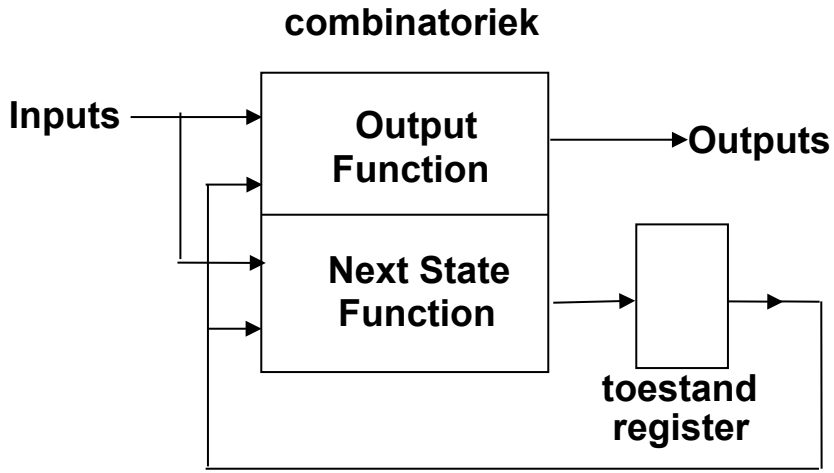
RCO = EN . [Q0..Q7 = 1..1]

24-bits teller mbv. RCO:



FSM realisaties mbv. modules

Finite State Machine model



Implementatie mogelijkheden:

- **combinatoriek: Losse poorten, Multiplexer, Decoder, PLA, ROM**
- **toestandregister: Losse flipflops, Dataregister, Teller met parallel load**
- **gehele FSM: registered PAL, CPLD of FPGA**

Getalsystemen

Getalsystemen

Representatie van positieve getallen is eenduidig (en reeds behandeld)

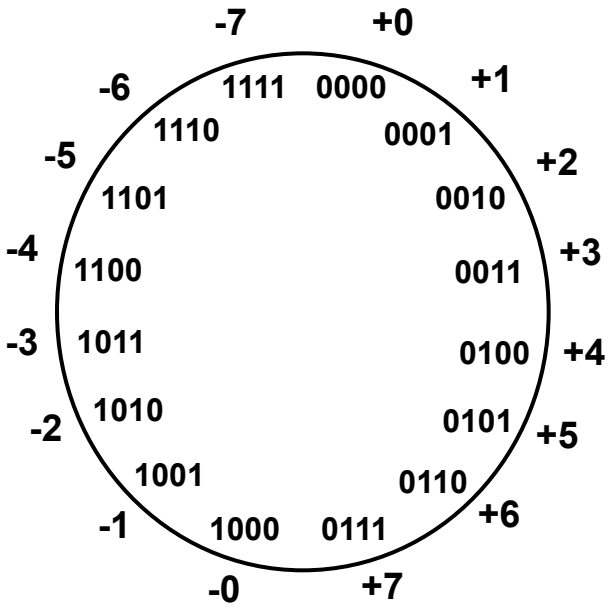
bv. 5 = 0101

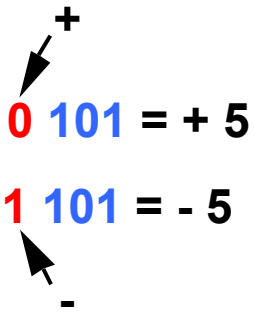
Met betrekking tot *negatieve* getallen bestaan er de volgende systemen:

- sign-magnitude (bv. 5 = 0101, -5 = 1101)
- one's complement (bv. 5 = 0101, -5 = 1010)
- two's complement (bv. 5 = 0101, -5 = 1011)

In de volgende sheets geven we voorbeelden voor 4 bits

Sign-Magnitude systeem



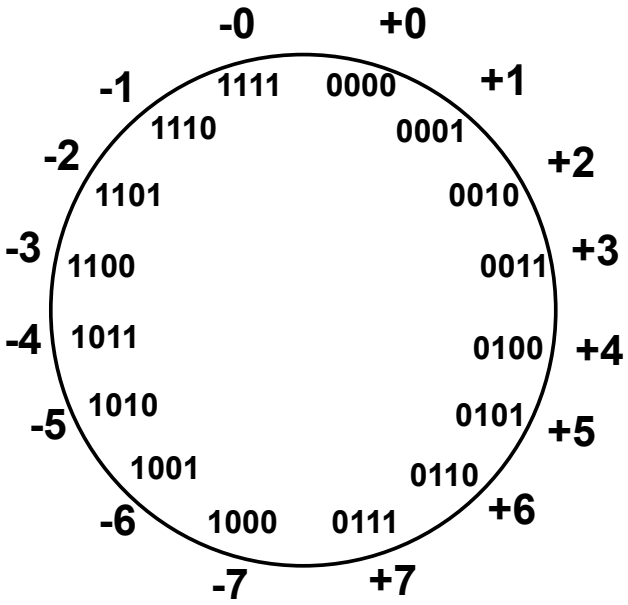


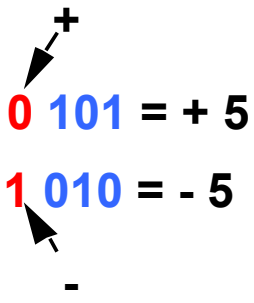
Hoogste bit geeft teken (sign):
0 = positief (of nul),
1 = negatief

Lagere bits geven de modulus (magnitude):
0 (000) - 7 (111)

- **Getalsbereik voor *n* bits: $[-2^{n-1} + 1, 2^{n-1} - 1]$**
- **Twee verschillende representaties voor 0 !**
- **Zeer lastige optelling/aftrekking**

One's Complement systeem



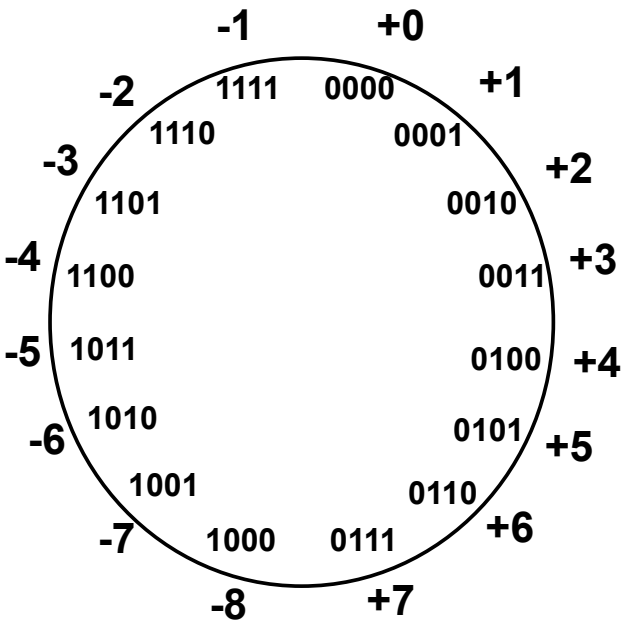


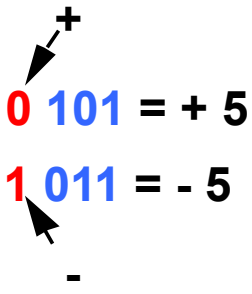
bitrepr (-X) = $\overline{\text{bitrepr (X)}}$

(-X = "one's complement" van X)

- **Getalsbereik voor *n* bits: $[-2^{n-1} + 1, 2^{n-1} - 1]$**
- **Twee verschillende representaties voor 0 !**
- **Lastige optelling/aftrekking**

Two's Complement systeem





1's compl, maar 1 getal *verschoven*:

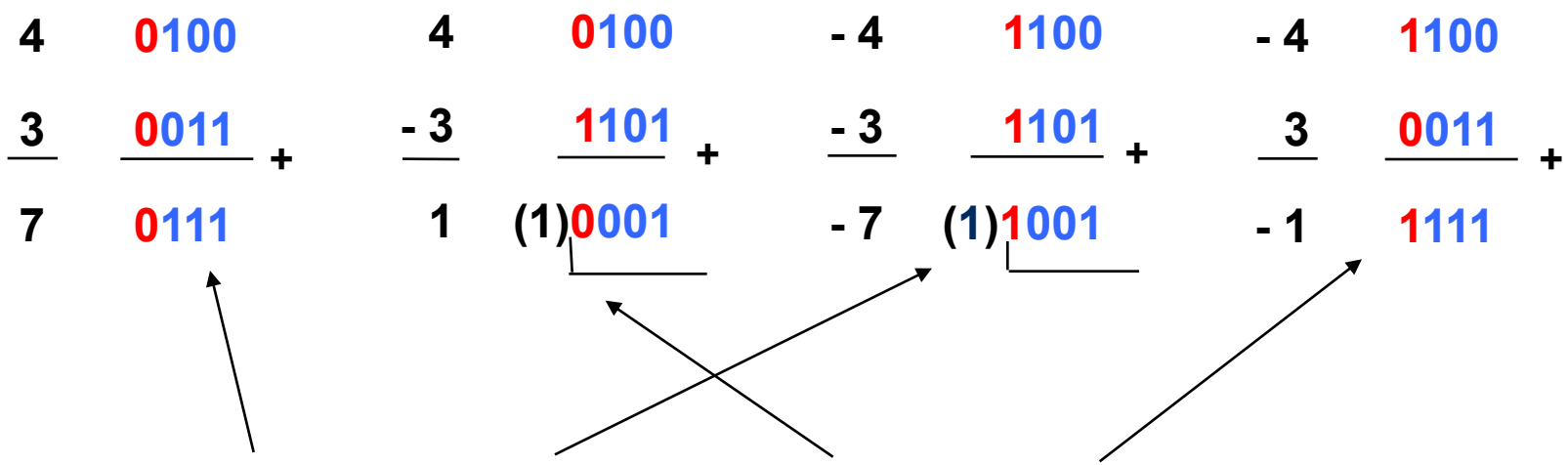
bitrepr (-X) = $\overline{\text{bitrepr (X)}}$ + 1

(ook geldt: bitrepr (X) = $\overline{\text{bitrepr (-X)}}$ + 1)

=> optelling/aftrekking kan eenvoudig door representaties (van positieve en negatieve getallen) altijd op te tellen (zie volgende slide)

- **Getalsbereik voor n bits: $[-2^{n-1}, 2^{n-1} - 1]$**
- **Slechts één representatie voor 0 !**
- **Verreweg meest populaire systeem**

optellen/aftrekken in 2's-complement



Toevallig geen *overflow*:
De carry mag dan verwaarloosd worden. Het resultaat kan echter buiten het getalsbereik liggen (zie volgende slide).

Nooit last van *overflow*:
resultaat altijd binnen getalsbereik (pos + neg of neg + pos)
De carry mag dan verwaarloosd worden

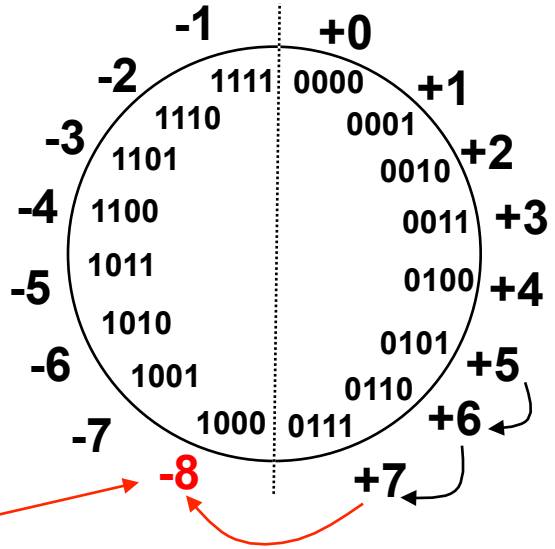
Overflow situaties

geen overflow

	0 0 0 0
5	0 1 0 1
2	0 0 1 0
7	0 1 1 1

geen overflow

	1 1 1 1
- 3	1 1 0 1
- 5	1 0 1 1
- 8	1 1 0 0 0

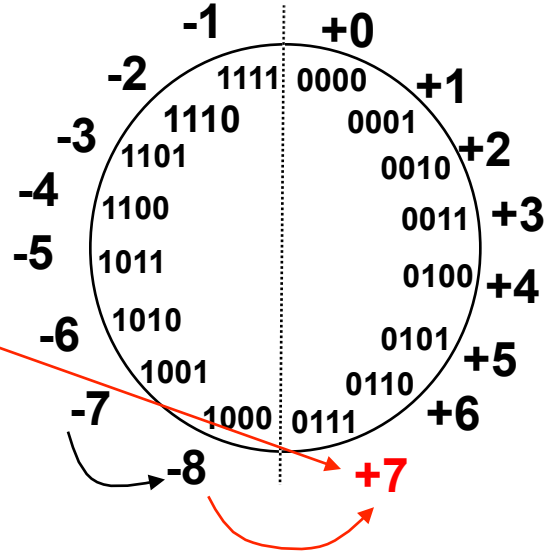


overflow!

	0 1 1 1
5	0 1 0 1
3	0 0 1 1
- 8	1 0 0 0

overflow!

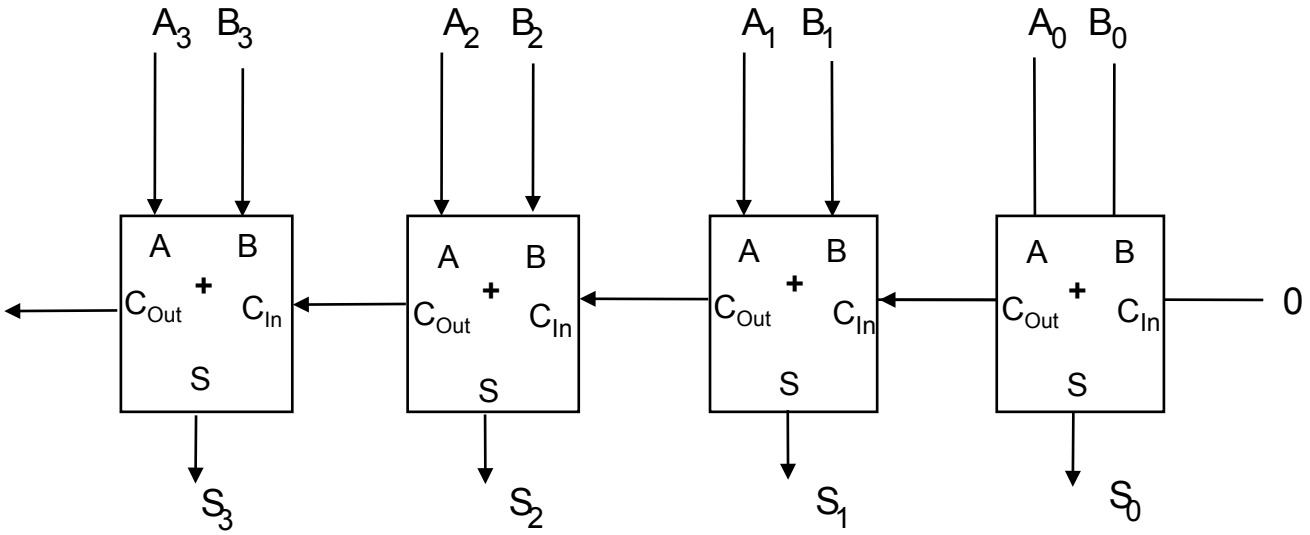
	1 0 0 0
- 7	1 0 0 1
- 2	1 1 0 0
7	1 0 1 1 1



Overflow indien de 2 hoogste carry's niet gelijk

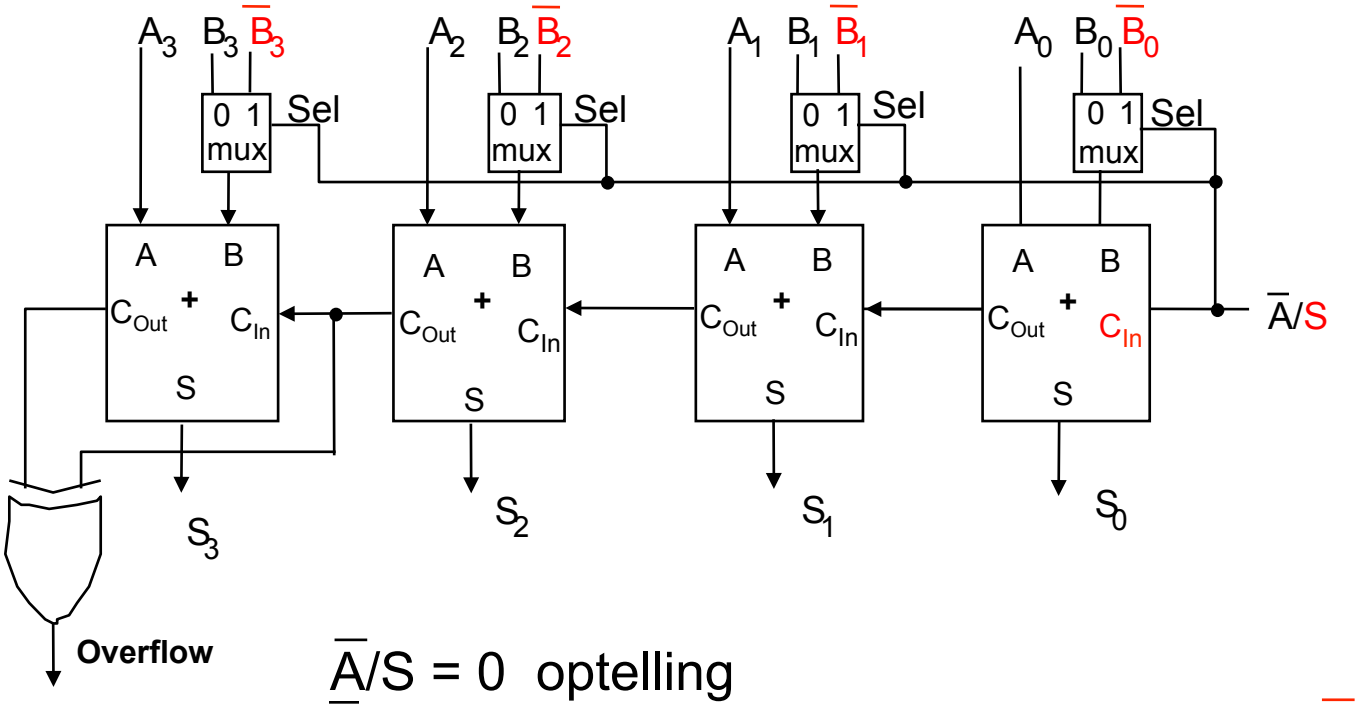
Aritmetische Circuits

4-bit opteller met Full-Adders



S = A + B

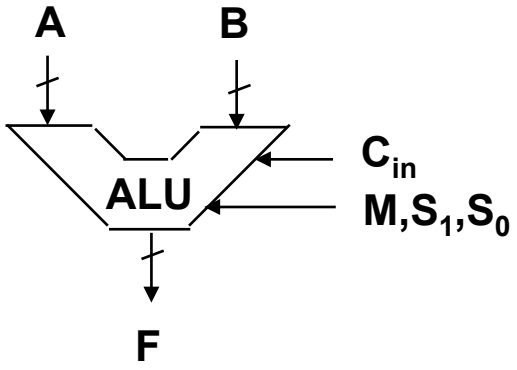
4-bit opteller/aftrekker



$\overline{A}/S = 0$ optelling

$\overline{A}/S = 1$ aftrekking $(A - B = A + (-B) = A + \overline{B} + 1)$

Arithmetic Logic Unit



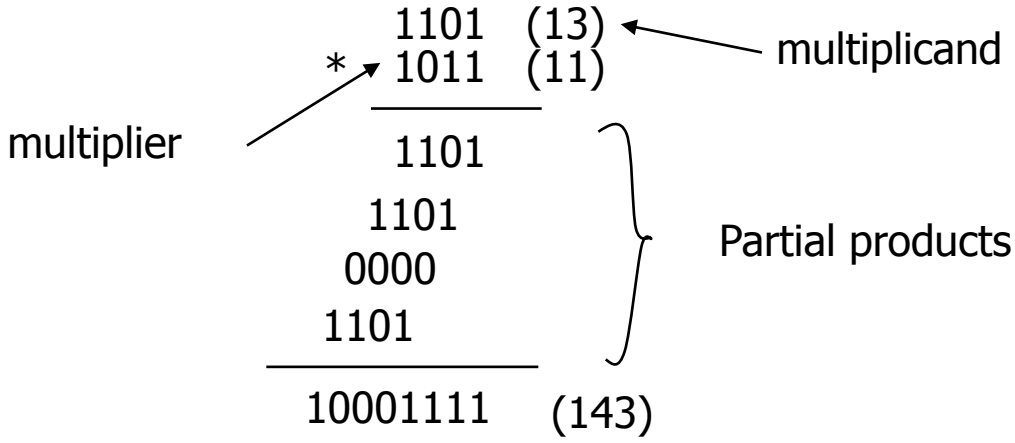
S_1	S_0	Functie	Opmerking
M = 0, Logische Operaties (bitwise)			
0	0	$F_i = A_i$	F = A
0	1	F_i = not A_i	Complement van A
1	0	$F_i = A_i$ xor B_i	XOR van A_i, B_i
1	1	$F_i = A_i$ xnor B_i	XNOR van A_i, B_i
M = 1, Aritmetische Operaties			
0	0	$F = A + C_{In}$	A of A+1
0	1	$F = (\text{not } A) + C_{In}$	F = 1's- of 2's compl. A
1	0	$F = A + B + C_{In}$	Som van A en B
1	1	$F = (\text{not } A) + B + C_{In}$	B + compl. A (= B - A als C_{in} =1)

Vermenigvuldiger (principe)

Voorbeeld: bereken 13 * 11

Vergelijk met:

```
  13
  11
  13
 13
 143
```

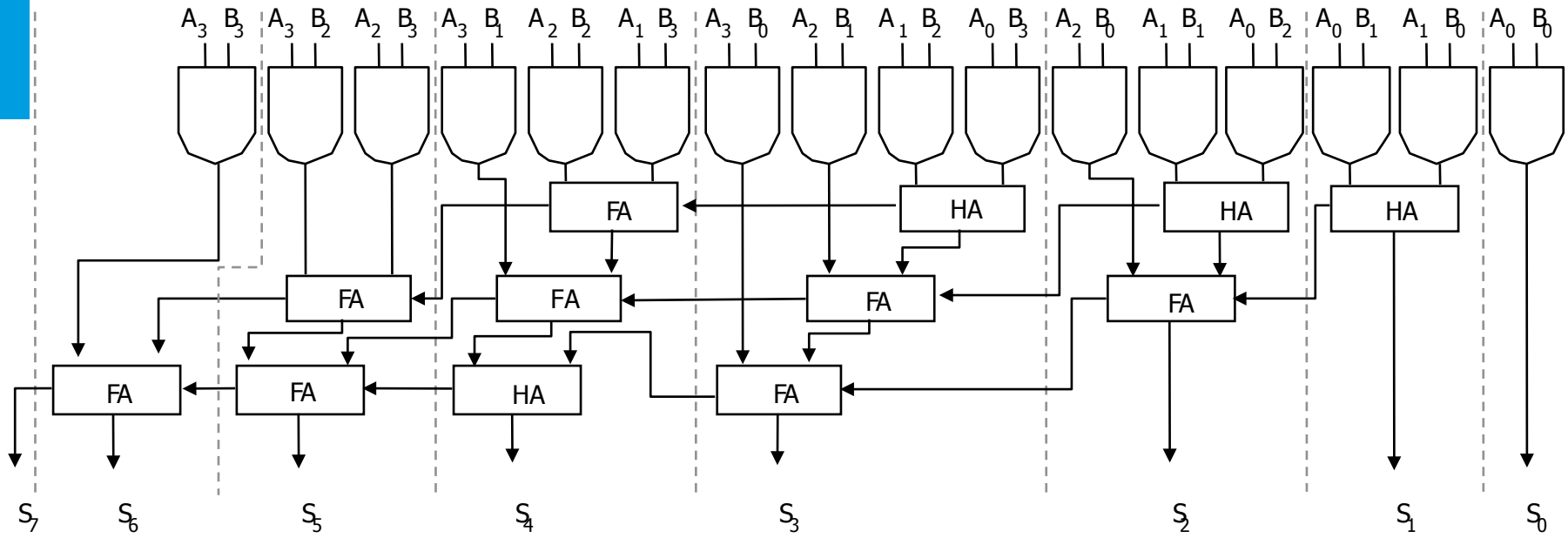


Optellen Partial Products

				A_3 B_3	A_2 B_2	A_1 B_1	A_0 B_0
				A_3 B_0	A_2 B_0	A_1 B_0	A_0 B_0
			A_3 B_1	A_2 B_1	A_1 B_1	A_0 B_1	
		A_3 B_2	A_2 B_2	A_1 B_2	A_0 B_2		
	A_3 B_3	A_2 B_3	A_1 B_3	A_0 B_3			
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Het product van twee n-bit getallen is een 2n-bit getal!

Vermenigvuldiger (implementatie)



Let op het gebruik van de parallel carry-outs t.b.v. de higher order sums!
De AND poorten zitten in de praktijk bij de adders in het array.

Samenvatting

- Combinatorische modules
- Sequentiële modules
- Getalsystemen
- Arithmetische modules

Volgende college: VHDL