technische universität
dortmund

Rechnerstrukturen WS 2012/13

- Synchrone Schaltwerke (Wiederholung)
  - Einleitung
  - Flip-Flops
- Schaltwerk-Entwurf
  - Einleitung
  - von Neumann-Addierwerk
- Speicher
  - SRAM-Realisierung

Hinweis: *Folien teilweise a. d. Basis von Materialien von Thomas Jansen*

19. November 2012

Synchrone Schaltwerke

ab jetzt getaktete Schaltwerke

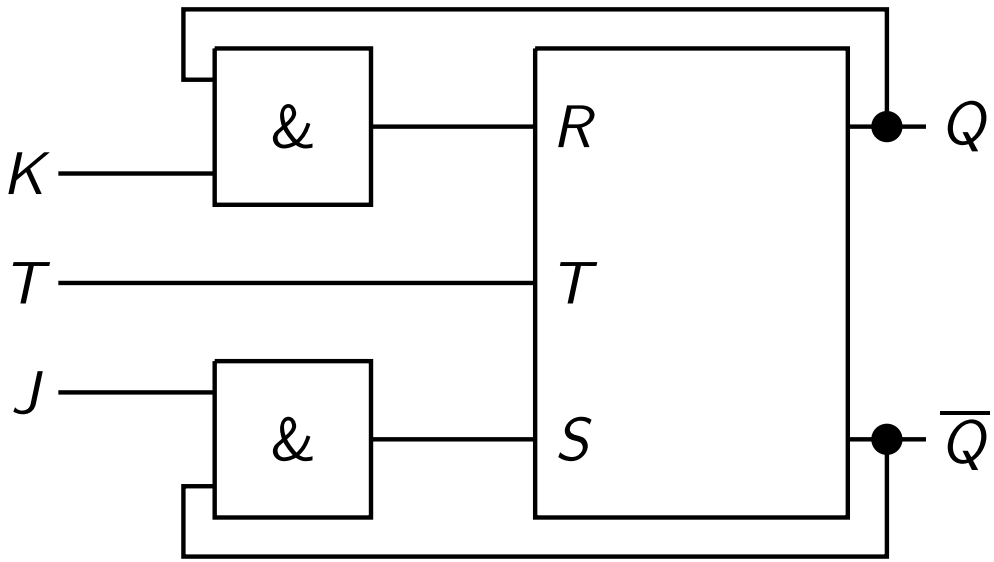
also Führe Taktsignal T ein

verschiedene technische Möglichkeiten

- Pegelsteuerung: aktiv, wenn 1 anliegt
- positive Flankensteuerung: aktiv, wenn Wechsel von 0 nach 1
- negative Flankensteuerung: aktiv, wenn Wechsel von 1 nach 0

digital-logische Ebene technisches Detail ignorieren

JK-Flip-Flop



R	S	Q
0	0	Q
0	1	1
1	0	0
1	1	nicht erlaubt

J	K	R	S	Q
0	0	0	0	Q
0	1	Q	0	0
1	0	0	$\overline{Q}$	1
1	1	Q	$\overline{Q}$	$\overline{Q}$

positiv alle Eingaben erlaubt, sinnvolle Funktionalität, vielseitig

Flip-Flops

Wir haben also hier 4 verschiedene Flip-Flop-Typen

Wozu brauchen wir eigentlich Flip-Flops?

klar Realisierung von Speicher

Was müssen wir für den Einsatz als Speicher wissen?

klar gezielte Änderung von Speicherinhalten

also Zustandstabellen eigentlich nicht interessant

besser Ansteuertabellen

etwas präziser

Zustandstabelle		Ansteuertabelle		
Eingabe	$\Rightarrow$ Zustand	Ist-Zustand, Soll-Zustand	$\Rightarrow$	Eingabe

Anmerkung Ansteuertabellen können „don't care“ enthalten

Ansteuertabelle JK-Flip-Flop

Zustandstabelle

J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Ansteuertabelle

Q_{alt}	Q_{neu}	J	K
0	0	0	$*$
0	1	1	$*$
1	0	$*$	1
1	1	$*$	0

Beobachtung immer **Freiheit** in der Ansteuerung

Unvollständig definierte Ansteuerfunktionen

Wir haben für RS-Flip-Flops und JK-Flip-Flops
nur partiell definierte Ansteuerfunktionen

Ist das ungünstig?

Nein!

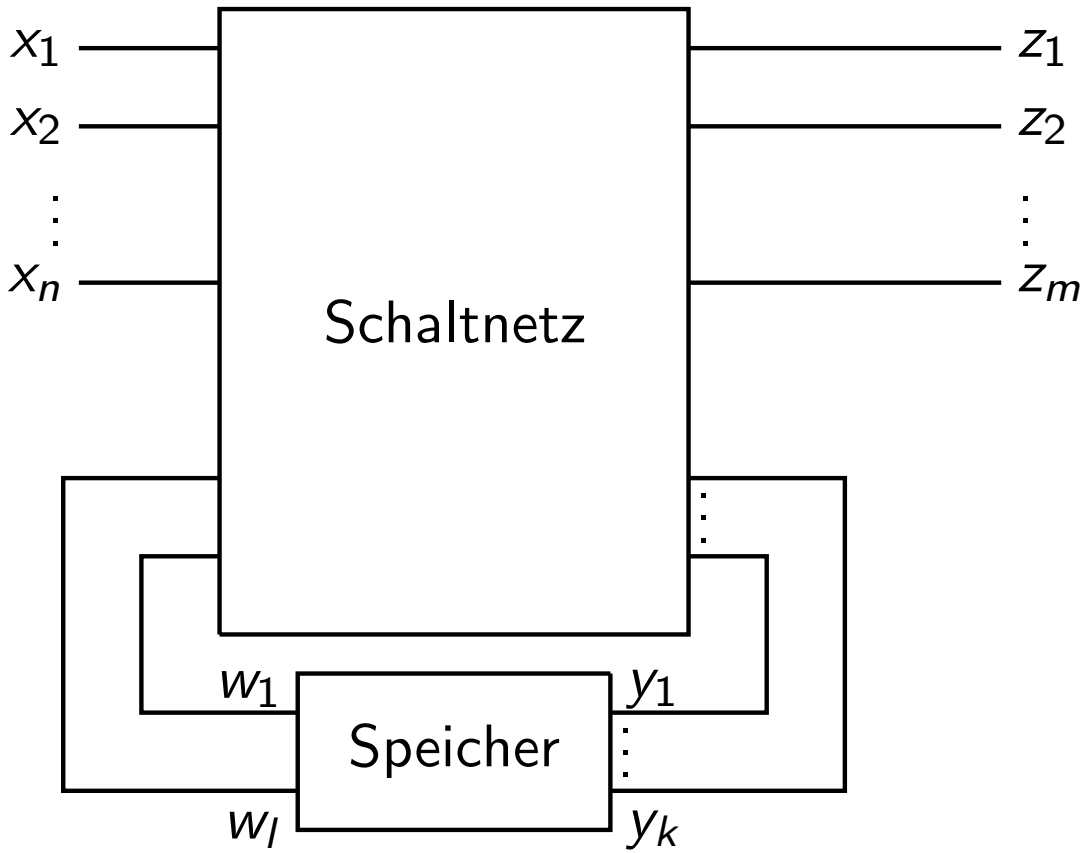
Erinnerung Minimalpolynome für partiell definierte Funktionen

Erinnerung Realisierungen können wesentlich einfacher sein

Erinnerung Minimalpolynom für partiell definiertes f
durch PI für f_1 und Überdeckung von f_0

Huffmann Schaltwerk-Modell

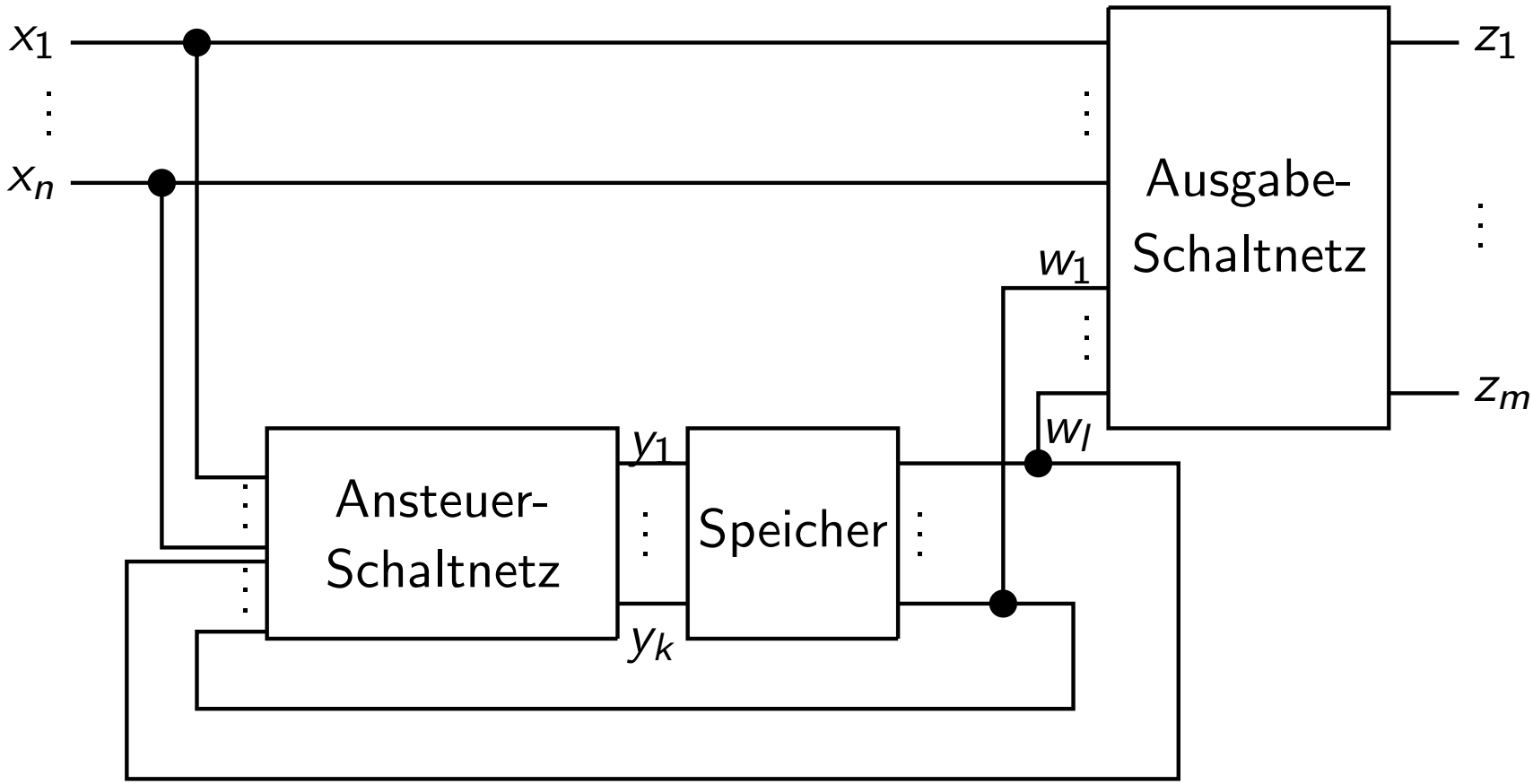
ein allgemeines, formales Schaltwerkmodell



Beobachtung führt Gelerntes über Schaltnetze, Flip-Flops und Automaten sinnvoll zusammen

Huffmann Schaltwerk-Modell

etwas detaillierter



Schaltwerk-Entwurf

Wunsch strukturierter Schaltwerk-Entwurf

Was können wir überhaupt als Schaltwerk realisieren?

klar alles, was als Automat beschrieben werden kann

zum Beispiel

- Getränke-Automat
- Ampelsteuerung
- Waschmaschinen-Steuerung
- ...

Anmerkung Heuristiken und Erfahrung sind wichtig

Schritte beim Schaltwerk-Entwurf

0. Verstehen der Aufgabe
1. Spezifikation des Verhaltens (z. B. als Mealy-Automat)
2. Wahl der Coderierung von Eingaben, Zuständen, Ausgaben
3. Wertetabelle mit Eingaben, Zustand, Ausgaben, neuem Zustand
4. Wahl der Flip-Flop-Typen
5. Ergänzung der Wertetabelle um die Flip-Flop-Ansteuerung
6. Entwurf passender boolescher Funktionen
7. Entwurf passender Schaltnetze
8. Entwurf vollständiges getaktetes Schaltwerk

Beispiel zum Schaltwerk-Entwurf

zur Einführung ein „praktisches“ Beispiel

hilfreich besonders gut verstandenes Problem

Aufgabe Addition von zwei Betragszahlen

Erinnerung wir haben

Verfahren	Größe	Tiefe
Schul-Methode	$\approx 5n$	$\approx 2n$
Carry-Look-Ahead	$\approx n^2$	$\approx 2\log_2 n$

jetzt klein und flach mit einem Schaltwerk
aber extrem langsam

Entwurf Addierwerk: Schritt 0

Schritt 0 verstehen der Aufgabe

Wunsch Summanden bitweise eingeben
Summe bitweise erhalten

klar geht nur von rechts nach links

Was muss man sich merken?

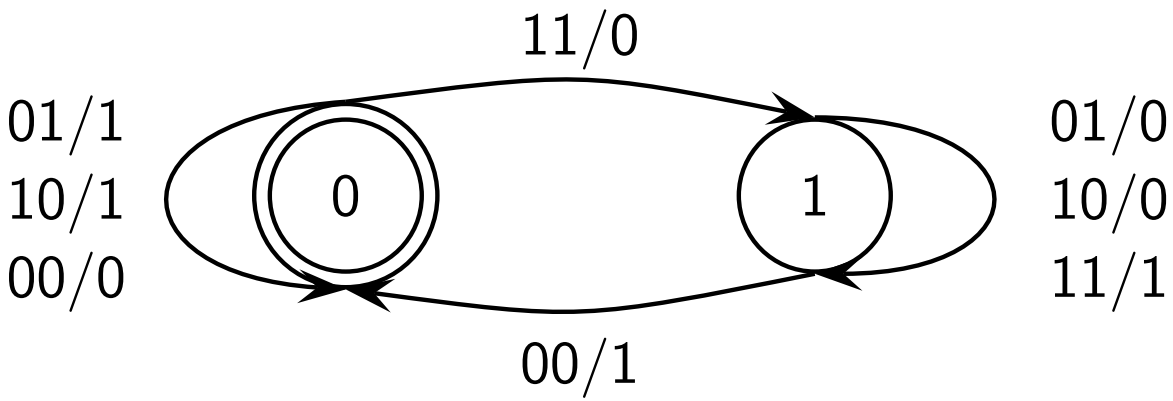
klar nur den aktuellen Übertrag
also 1 Bit

Entwurf Addierwerk: Schritt 1

Schritt 1 Spezifikation des Verhaltens

Entscheidung Beschreibung durch Mealy-Automat

Eingabealphabet $\Sigma = \{00, 01, 10, 11\}$
Ausgabealphabet $\Delta = \{0, 1\}$
Zustandsmenge $Q = \{0, 1\}$



Entwurf Addierwerk: Schritt 2

Schritt 2 Wahl der Codierung: Eingaben, Zuständen, Ausgaben

klar im Allgemeinen (fast) jede Freiheit

hier kanonische Codierung naheliegend

Codierung von Q

$q \in Q$	Codierung
0	0
1	1

Codierung von Σ

$w \in \Sigma$	Codierung
00	0 0
01	0 1
10	1 0
11	1 1

Codierung von Δ

$w \in \Delta$	Codierung
0	0
1	1

Entwurf Addierwerk: Schritt 3

Schritt 3 Wertetabelle Eingaben, Zustand, neuer Zustand, Ausgaben

naheliegend Eingaben heißen x, y
alter Zustand heißt c_{alt}
neuer Zustand heißt c_{neu}
Ausgabe heißt s

c_{alt}	x	y	c_{neu}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Entwurf Addierwerk: Schritt 4

Schritt 4 Wahl der Flip-Flop-Typen

Entscheidung JK-Flip-Flops

Anmerkung

- nicht viele überzeugende Gründe
- weil es besonders viele Freiheiten erlaubt
- weil der Zustandswechsel nichts nahelegt
- weil es oft benutzt wird

Enturf Addierwerk: Schritt 5

Schritt 5 Ergänzung der Wertetabelle um Flip-Flop-Ansteuerung

c_{alt}	x	y	c_{neu}	s	J	K
0	0	0	0	0	0	$*$
0	0	1	0	1	0	$*$
0	1	0	0	1	0	$*$
0	1	1	1	0	1	$*$
1	0	0	0	1	$*$	1
1	0	1	1	0	$*$	0
1	1	0	1	0	$*$	0
1	1	1	1	1	$*$	0

Ansteuertabelle JK-Flip-Flop

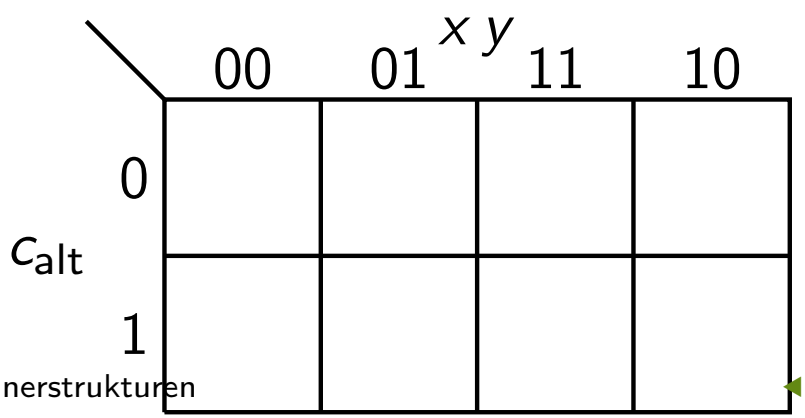
Q_{alt}	Q_{neu}	J	K
0	0	0	$*$
0	1	1	$*$
1	0	$*$	1
1	1	$*$	0

Entwurf Addierwerk: Schritt 6

Schritt 6 Entwurf passender boolescher Funktionen

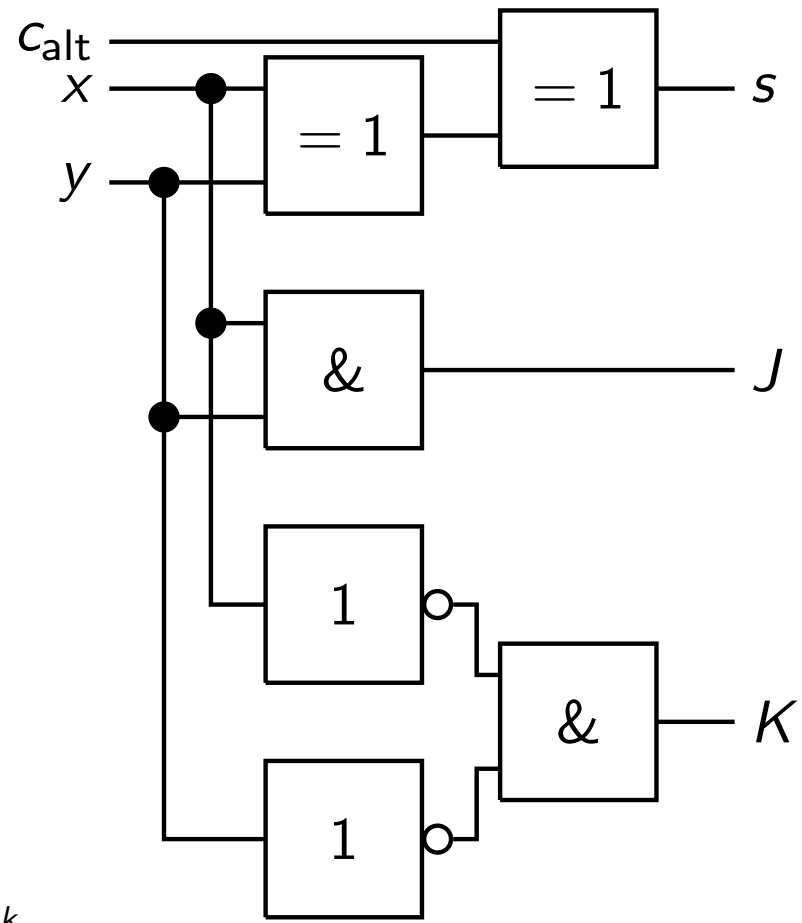
c_{alt}	x	y	c_{neu}	s	J	K
0	0	0	0	0	0	$*$
0	0	1	0	1	0	$*$
0	1	0	0	1	0	$*$
0	1	1	1	0	1	$*$
1	0	0	0	1	$*$	1
1	0	1	1	0	$*$	0
1	1	0	1	0	$*$	0
1	1	1	1	1	$*$	0

$$J(c_{alt}, x, y) = x\,y$$
$$K(c_{alt}, x, y) = \overline{x}\,\overline{y}$$
$$s(c_{alt}, x, y) = c_{alt} \oplus x \oplus y$$



Entwurf Addierwerk: Schritt 7

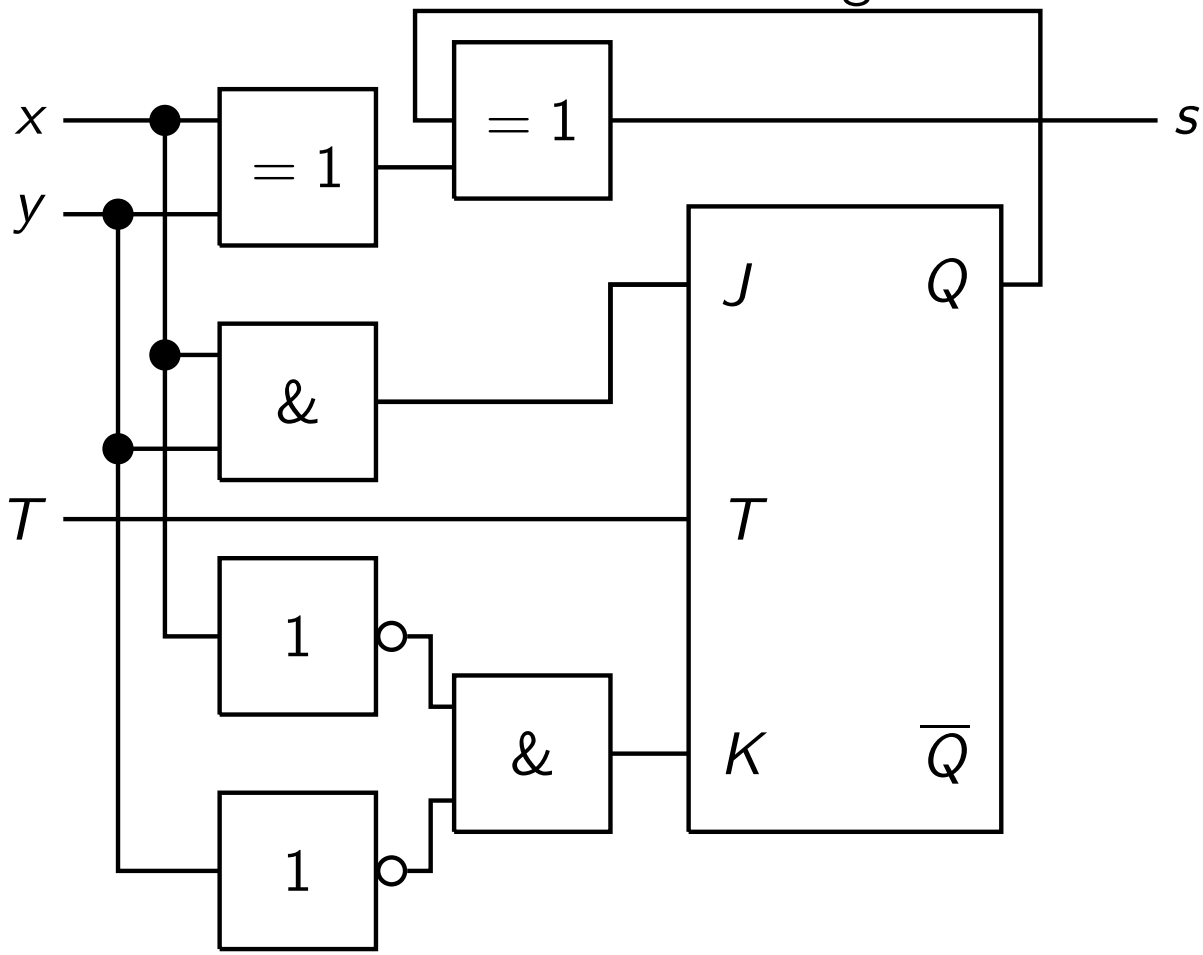
Schritt 7 Entwurf passender Schaltnetze



$$J(c_{alt}, x, y) = x\,y$$
$$K(c_{alt}, x, y) = \overline{x}\,\overline{y}$$
$$s(c_{alt}, x, y) = c_{alt} \oplus x \oplus y$$

Entwurf Addierwerk: Schritt 8

Schritt 8 Entwurf des vollständigen Schaltwerks



$$\begin{array}{r} 0\ 1\ 0\ 0\ 1\ 1 \\ +\ 0\ 1\ 0\ 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 0\ 0\ 0 \end{array}$$

Beobachung Eingabe $(0,0)$ im letzten Takt wichtig

Initialisierung? Eingabe $(0,0)$

Unser Addierwerk: Serien-Addierer

Fazit

- addiert beliebig lange Betragszahlen
- ist sehr klein und flach
- ist sehr leicht zu initialisieren
- ist extrem langsam

Warum ist das Schaltwerk so langsam?

klar und bekannt wegen der Überträge

Müssen Überträge so lange dauern?

bekannt im Allgemeinen nicht (siehe Addierer)

aber Bei bit-weiser Eingabe schon!

Über unsere Modelle

Wir wissen schon Überträge werden nur manchmal lange weitergereicht.

Kann man ein Schaltwerk bauen, dass nur manchmal langsam ist?

Problem unsere Automaten können das zunächst nicht

Beobachtung bei Mealy- und Moore-Automat bestimmt Länge der Eingabe die Anzahl der Rechentakte

darum Modifikation des Automatenmodells

neu Erlaube leere Eingabe ε und signalisiere Ende der Rechnung durch Rechenende-Zeichen

Beobachtung Das ist fundamental neu für uns. Rechenzeit kann von der Eingabe abhängen (nicht nur von der Eingabelänge).

Auf dem Weg zum besseren Addierwerk

Wie wollen wir vorgehen?

Beobachtung Eingaben müssen sofort ganz zur Verfügung stehen
sonst kann man nicht schneller sein

also Eingabealphabet $\Sigma = \{0,1\}^{2n}$

Eingabe $x_{n-1}x_{n-2}\cdots x_1x_0y_{n-1}y_{n-2}\cdots y_1y_0 \in \{0,1\}^{2n}$

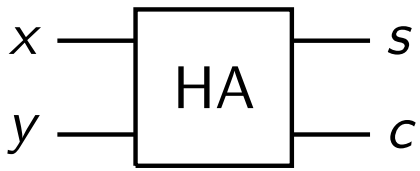
interpretieren als

$$\begin{array}{cccccc} & x_{n-1} & x_{n-2} & \cdots & x_0 \\ + & y_{n-1} & y_{n-2} & \cdots & y_0 \\ \hline \end{array}$$

bekannte Idee zur Lösung
Ersetze x und y durch x' und y' mit $x + y = x' + y'$
so lange, bis $y' = 0$ gilt.

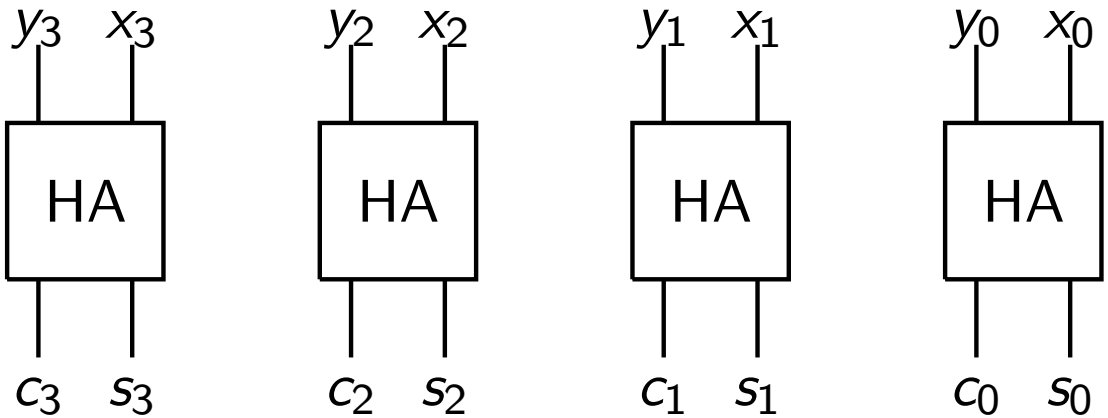
Eine gute Idee verallgemeinern

Erinnerung Wir kennen das schon vom Halbaddierer...



mit $x + y = 2c + s = (c0)_2 + s$

klar Das funktioniert auch für alle n Stellen parallel.



$$x_3' \, x_2' \, x_1' \, x_0' \quad = \quad s_3 \, s_2 \, s_1 \, s_0$$

Ü $y_3' \, y_2' \, y_1' \, y_0' \quad = \quad c_3 \, c_2 \, c_1 \, c_0 \, 0$

Fortschritt? in y hinten „neue“ 0

also nach $\leq n$ Takten $y = 0$

Noch offene Fragen

Was ist mit dem Ü?

klar potenziell kann in jedem Takt vorne ein Überlauf entstehen

Also bis zu n Überläufe speichern?

zum Glück nein

Wir wissen höchstens 1 Überlauf insgesamt

Wann ist die Rechnung fertig?

klar Rechnung fertig $\Leftrightarrow y = 0$

also „done“ $d = \overline{y_0 \vee y_1 \vee \cdots \vee y_{n-1}} = \neg \bigvee_{i=0}^{n-1} y_i$

jetzt unsere Ideen zusammensetzen

Das von Neumann-Addierwerk

John von Neumann (1903–1957)

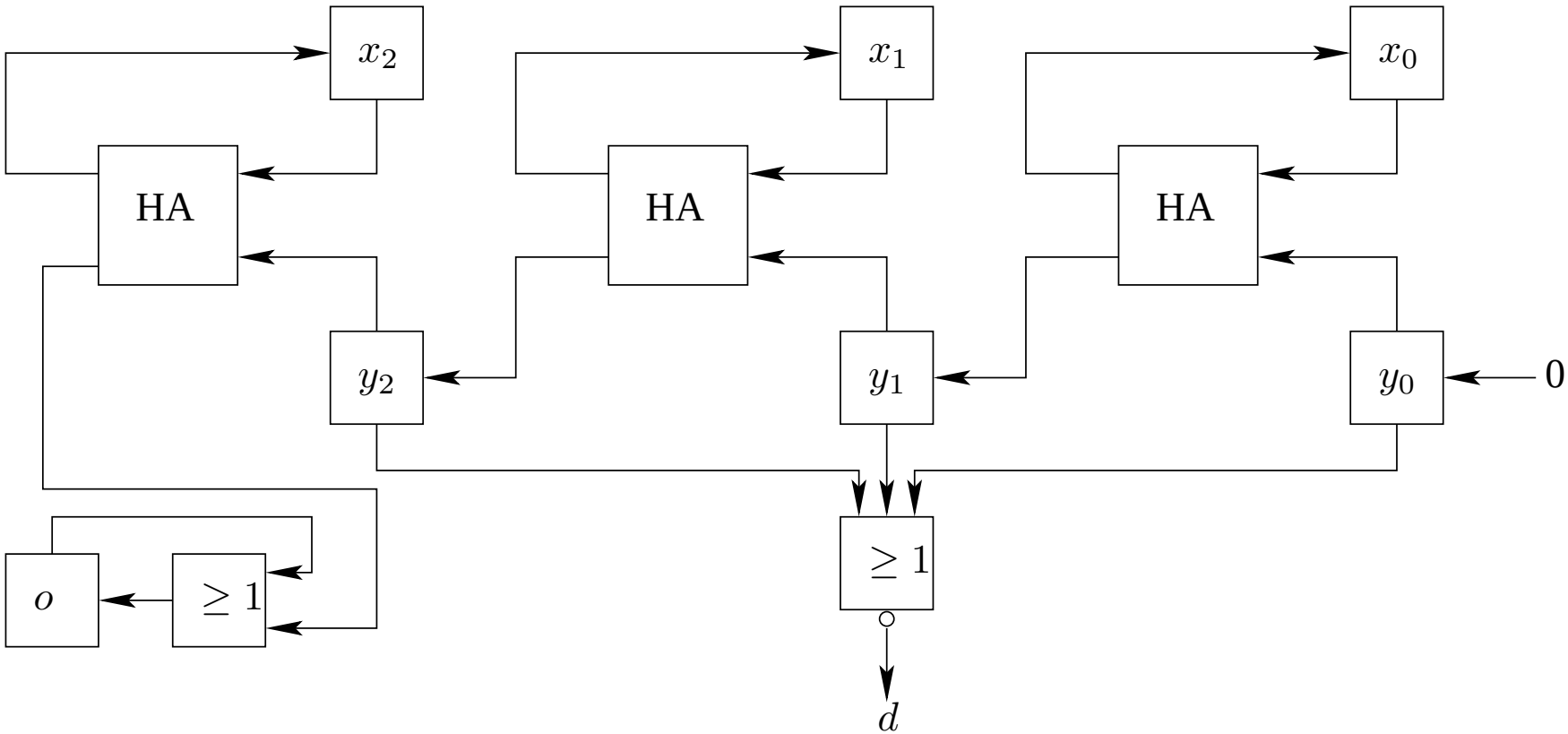
Ideen

- Schaltwerk bekommt direkt Summanden komplett
- Eingabe ε wird erlaubt
- Rechenende-Zeichen signalisiert Ende der Rechnung
- in jedem Takt ersetze x, y durch x', y' mit $x + y = x' + y'$
- Ersetzung stellenweise mit Halbaddierern

Hoffnung oft schnell

Schematische Darstellung von Neumann-Addierwerk

für $n = 3$



klar die x_i, y_i und Ü sind Speicher

$\rightsquigarrow$ mit Flip-Flops realisieren

Wieso Rückkopplung von 'Ü' mit Oder-Gatter?

Analyse des von Neumann-Addierwerks

Ist das Addierwerk denn jetzt wirklich schnell?

erstmal Welche Eingaben dauern lange?

1. Versuch viele Überträge

exemplarisch für $n = 6$ $111111 + 111111$

Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	1	1	1	1	1
	1	1	1	1	1	1
	y_5	y_4	y_3	y_2	y_1	y_0

$\rightsquigarrow$

Ü	x_5	x_4	x_3	x_2	x_1	x_0
1	0	0	0	0	0	0
	1	1	1	1	1	0
	y_5	y_4	y_3	y_2	y_1	y_0

$\rightsquigarrow$

Ü	x_5	x_4	x_3	x_2	x_1	x_0
1	1	1	1	1	1	0
	0	0	0	0	0	0
	y_5	y_4	y_3	y_2	y_1	y_0

also nach nur zwei Takten fertig schnell

Erinnerung bzgl. Übertrag schwierig: $0 + 1$, $1 + 0$

Analyse des von Neumann-Addierwerks II

2. Versuch lauter „W“-Stellen

exemplarisch für $n = 6$ $000000 + 111111$

Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	0	0	0	0	0	0
	1	1	1	1	1	1
	y_5	y_4	y_3	y_2	y_1	y_0

$\rightsquigarrow$

Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	1	1	1	1	1
	0	0	0	0	0	0
	y_5	y_4	y_3	y_2	y_1	y_0

also nach nur einem Takt fertig sehr schnell

klar $111111 + 000000$ wäre sofort fertig

Beobachtung Überträge mit langen Wegen dauern lange

Was ist die “worst case” Eingabe?

Worst-Case-Eingabe für das von Neumann-Addierwerk

Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	1	1	1	1	1
	0	0	0	0	0	1
	y_5	y_4	y_3	y_2	y_1	y_0
Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	1	1	1	0	0
	0	0	0	1	0	0
	y_5	y_4	y_3	y_2	y_1	y_0
Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	1	0	0	0	0
	0	1	0	0	0	0
	y_5	y_4	y_3	y_2	y_1	y_0
Ü	x_5	x_4	x_3	x_2	x_1	x_0
1	0	0	0	0	0	0
	0	0	0	0	0	0
	y_5	y_4	y_3	y_2	y_1	y_0

⇝

Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	1	1	1	1	0
	0	0	0	0	1	0
	y_5	y_4	y_3	y_2	y_1	y_0
Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	1	1	0	0	0
	0	0	1	0	0	0
	y_5	y_4	y_3	y_2	y_1	y_0
Ü	x_5	x_4	x_3	x_2	x_1	x_0
0	1	0	0	0	0	0
	1	0	0	0	0	0
	y_5	y_4	y_3	y_2	y_1	y_0

also extrem langsam

Analyse des von Neumann-Addierwerks

gesehen $111\cdots111 + 000\cdots000 \rightsquigarrow 0$ Takte
extrem schnell

gesehen $111\cdots111 + 000\cdots001 \rightsquigarrow n$ Takte
extrem langsam

Ist das von Neumann-Addierwerk schnell oder langsam?

Best Case	schnell	(0 Takte)
Worst Case	langsam	(n Takte)

0 Takte – Glücklicher Zufall? n Takte – Seltenes Unglück?
Was ist typisch?

Antwort Average-Case-Analyse
durchschnittliche Rechenzeit

Problem Welche Verteilung über Eingaben nehmen wir an?

Average-Case-Analyse

Average-Case-Rechenzeit durchschnittliche Rechenzeit bei Gewichtung der Eingaben nach ihrer W'keit bei gegebener W'keitsverteilung über den Eingaben

Problem Welche W'keitsverteilung ist angemessen?

meist betrachtet Gleichverteilung

Vorsicht reale Daten fast nie gleichverteilt

trotzdem hier Annahme Gleichverteilung

Gleichverteilung bedeutet

- alle Eingaben gleichwahrscheinlich
- an jeder Position 0 und 1 mit W'keit 1/2
- alle Positionen unabhängig

Average-Case-Analyse von Neumann-Addierwerk

Start: "Typische" Eingabe der Länge $2n$ bei Gleichverteilung:
je 50% Null- und Eins-Stellen (d.h. $n/2$ Einsen pro Summand)

Verarbeitung: nach 1 Takt ...

Überträge / y: ex. Eins an y_{i+1}, wenn vorher Eins sowohl an x_i als auch y_i (d.h. mit Wahrscheinlichkeit 1/4)
$\Rightarrow$ Erwarten nach 1 Takt $n/4$ Einsen in y

Ergebnis / x: ex. Eins in x_i, wenn vorher 01 oder 10 in x_i, y_i (d.h. in 50% der Fälle)
$\Rightarrow$ Erwarten nach 1 Takt *weiterhin* $n/2$ Einsen in x

$\Rightarrow$ Erwarten: Anzahl Einsen in y halbiert sich je Takt (in x unverändert)

Fazit: Erwarten Rechenende nach $log_2 n$ Takten

also Das von Neumann-Addierwerk ist im Durchschnitt schnell.

Moore-Automat zum von Neumann-Addierwerk

Ziel Beschreibung eines Moore-Automaten zum von Neumann-Addierwerk zur Addition von zwei n-Bit-Zahlen

Erinnerung Erweiterung des Automatenmodells um ε-Eingaben im folgenden Sinn

- bei Eingabe von $x\,y$ wird aktuelle Rechnung unterbrochen und mit Berechnung von $x + y$ begonnen
- bei Eingabe von ε wird aktuelle Rechnung fortgesetzt
- Ausgabe ist „–“, falls Rechnung noch nicht beendet
- Ausgabe ist Summe der Eingabezahlen, wenn fertig berechnet

Komponenten des Moore-Automaten

- Eingabealphabet
  alle möglichen Paare von n-Bit-Zahlen, außerdem ε
  $\Sigma := \{0,1\}^{2n} \cup \{\varepsilon\}$
- Ausgabealphabet
  alle möglichen Ergebnisse der Addition, außerdem $-$
  $\Delta := \{0,1\}^{n+1} \cup \{-\}$
- Zustandsmenge
  alle möglichen Zwischenergebnisse einschließlich Übertrag
  $Q := \{0,1\}^{2n+1}$
- Startzustand
  willkürlich $q_0 = \underbrace{00\cdots00}_{2n+1 \text{ Nullen}} = 0^{2n+1}$

Zustandsüberführungsfunktion δ

$\Sigma = \{0,1\}^{2n} \cup \{\varepsilon\}$, $\Delta = \{0,1\}^{n+1} \cup \{-\}$, $Q = \{0,1\}^{2n+1}$, $q_0 = 0^{2n+1}$

1. Fall Eingabe $w = x\,y \neq \varepsilon$

$\delta(q, w) := 0\,w = 0\,x\,y$

2. Fall Eingabe $w = \varepsilon$

Notation aktueller Zustand $q = \ddot{u}\,x\,y$

$\delta(q, \varepsilon) := q' = \ddot{u}'\,x'\,y' = \ddot{u}'\,x'_{n-1}\,x'_{n-2} \cdots x'_0\,y'_{n-1}\,y'_{n-2} \cdots y'_0$
mit

- $y'_0 := 0$
- $y'_i := x_{i-1} \wedge y_{i-1}$ für $i \in \{1, 2, \ldots, n-1\}$
- $x'_i := x_i \oplus y_i$ für $i \in \{0, 1, \ldots, n-1\}$
- $\ddot{u}' := \ddot{u} \vee (x_{n-1} \wedge y_{n-1})$

Ausgabefunktion λ

$\Sigma = \{0,1\}^{2n} \cup \{\varepsilon\}$, $\Delta = \{0,1\}^{n+1} \cup \{-\}$, $Q = \{0,1\}^{2n+1}$, $q_0 = 0^{2n+1}$

Notation neuer Zustand $\delta(q,w) = q' = \ddot{u}'\, x'\, y'$

1. Fall $y' \neq 0^n$

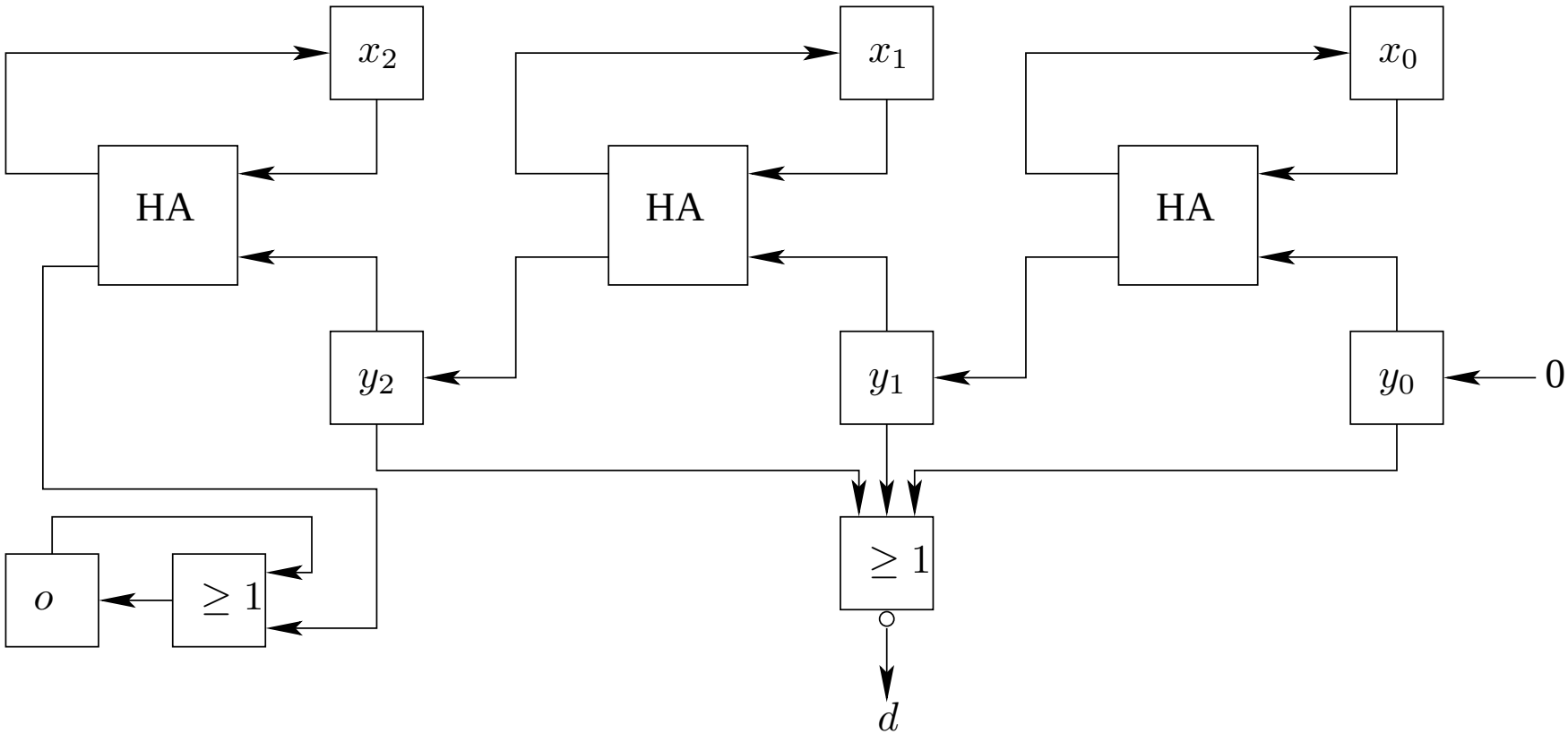
$\lambda(\delta(q,w)) := -$

2. Fall $y' = 0^n$

$\lambda(\delta(q,w)) := \ddot{u}'\, x'$

Schematische Darstellung von Neumann-Addierwerk

für $n = 3$



Speicher

Speicher

Erinnerung Wir können in einem Flip-Flop ein Bit speichern.

klar 1-Bit-Speicher reichen uns nicht.

klar Wir können in k Flips-Flops k Bits speichern.

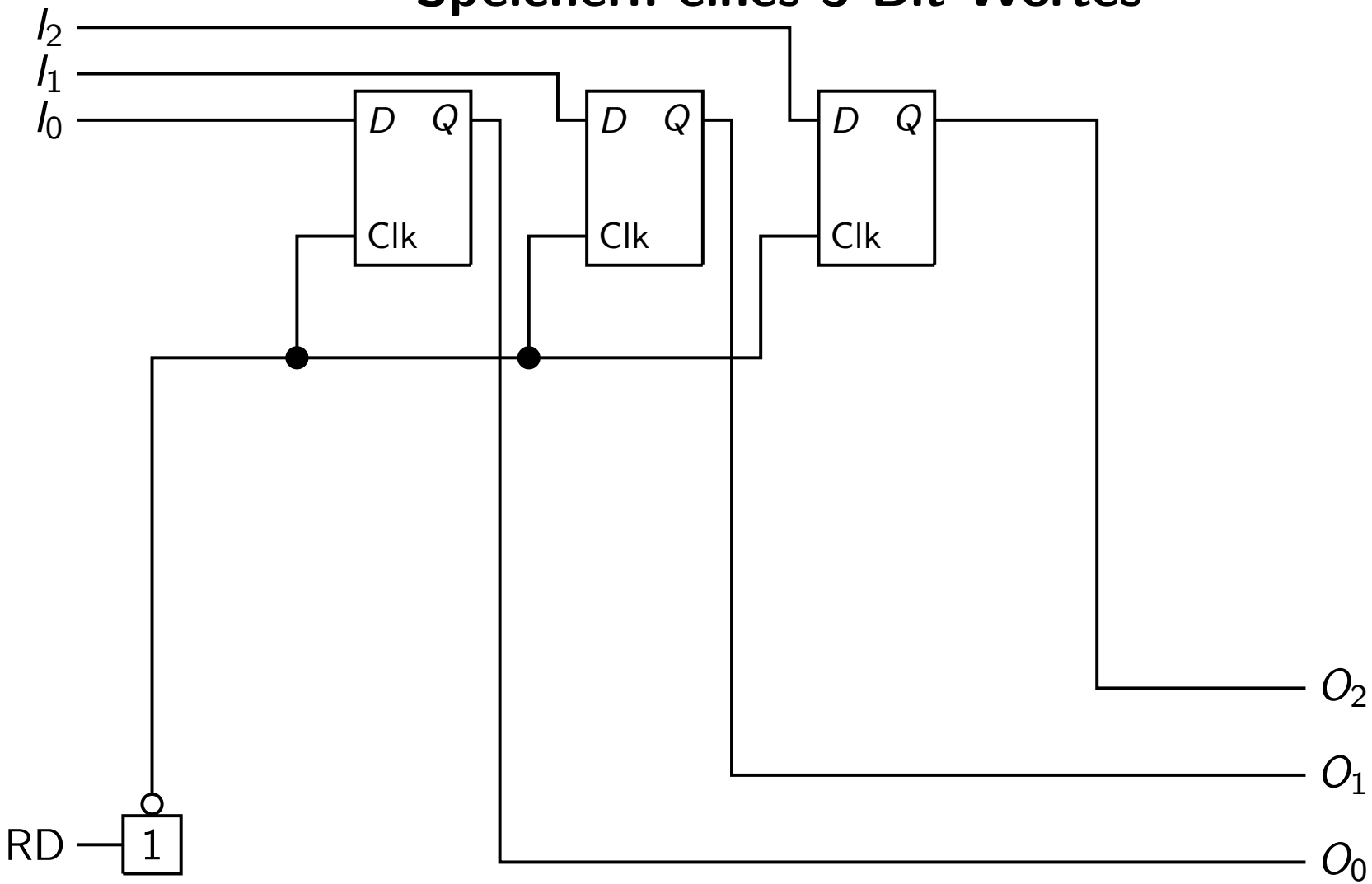
Wie organisieren wir das so, dass der Zugriff bequem ist?

exemplarisch Speicher für 3-Bit-Wörter

Warum 3? passt bequem auf eine Folie

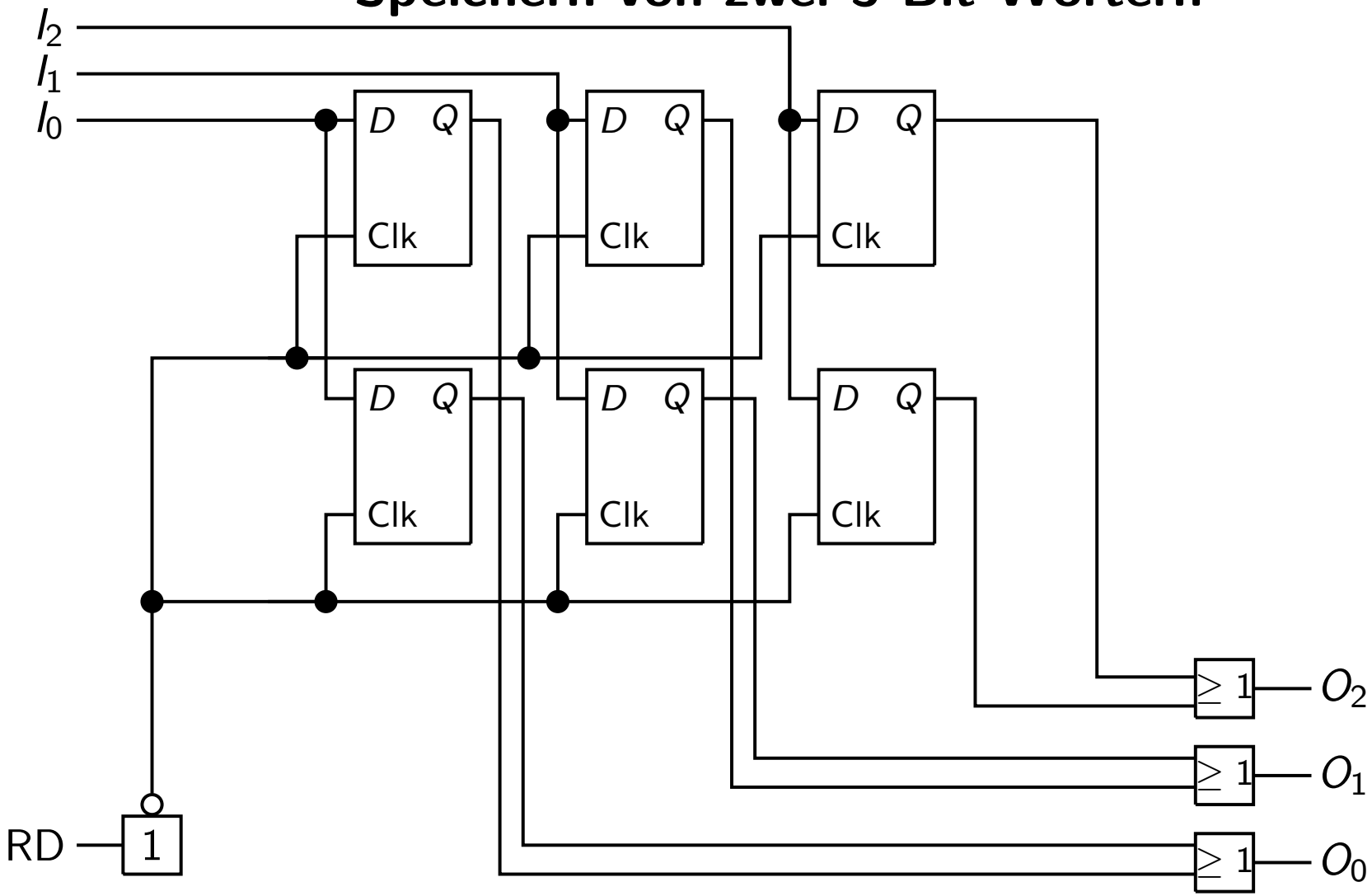
Anmerkung Auch 3-Bit-Speicher reicht in der Praxis nicht aus.

Speichern eines 3-Bit-Wortes



Wie können wir mehr als nur ein Wort speichern?

Speichern von zwei 3-Bit-Wörtern



Beobachtung Schaltung funktioniert so nicht

Speichern von zwei 3-Bit-Wörtern

klar Die Speicherwörter müssen getrennt ansprechbar sein.

also Wir wollen ein Wort wahlfrei addressieren.

klar Wir brauchen eine Adressleitung A_0.

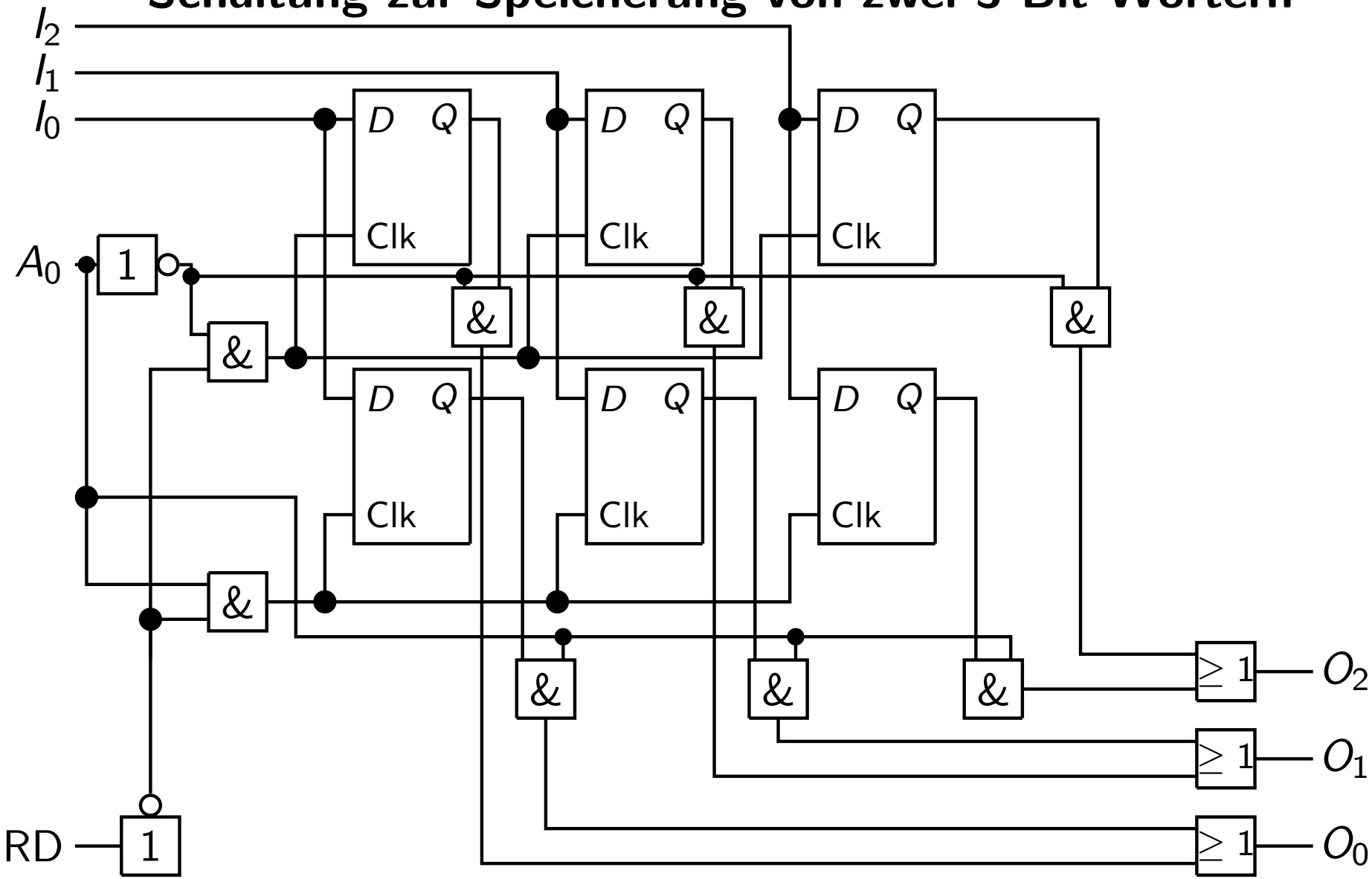
Interpretation

- $A_0 = 0$ Wort w_0 addressiert zum Lesen oder Schreiben
- $A_0 = 1$ Wort w_1 addressiert zum Lesen oder Schreiben

wie bisher RD bestimmt, ob gelesen oder geschrieben wird

- $\mathrm{RD} = 1$ Speicherinhalt lesen
- $\mathrm{RD} = 0$ Speicherinhalt schreiben

Schaltung zur Speicherung von zwei 3-Bit-Wörtern



Speichererweiterungen

angenommen Schaltung zur Speicherung von zwei 3-Bit-Wörtern im Einsatz

angenommen Speicher reicht im Betrieb nicht aus

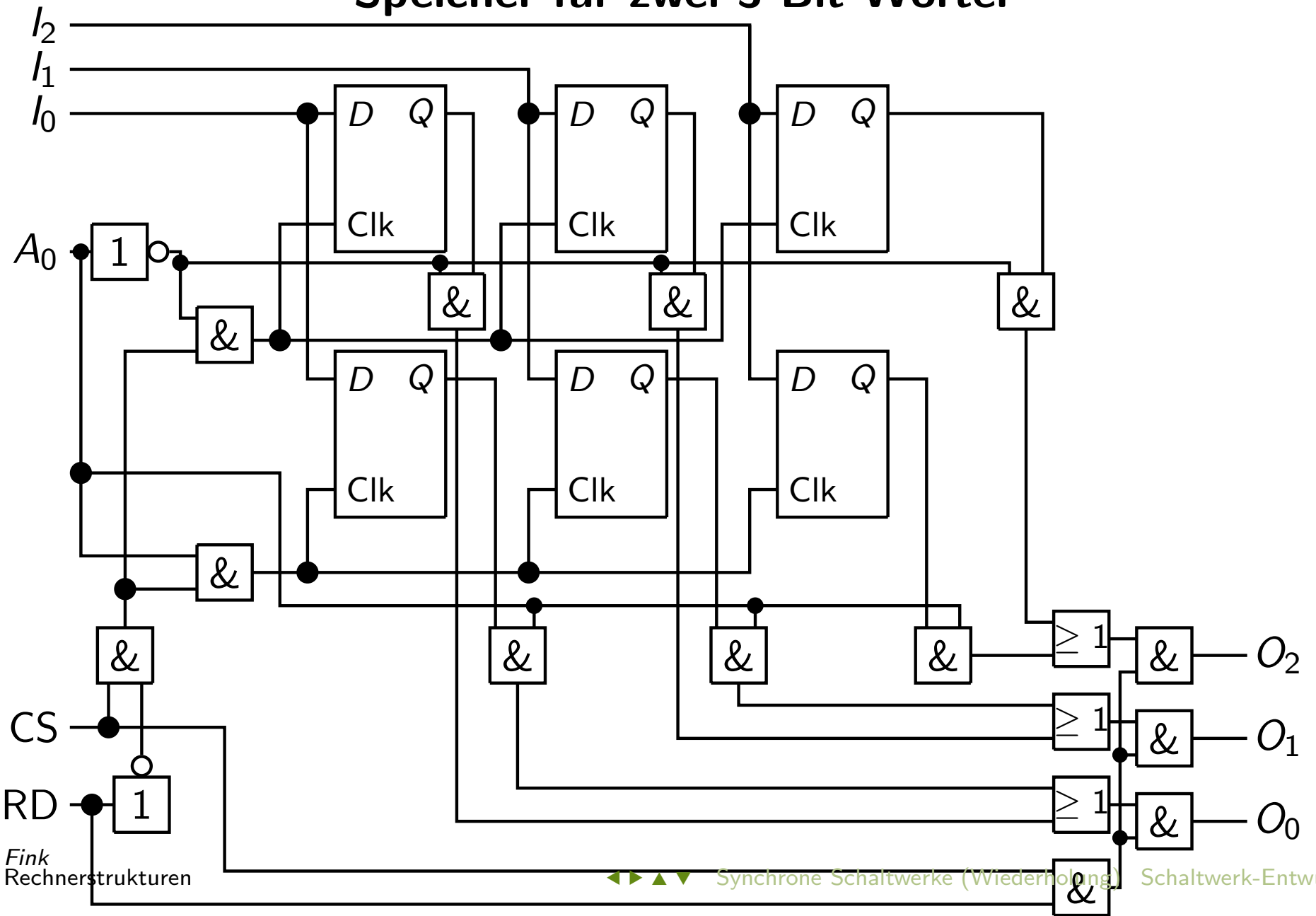
Wunsch Speichererweiterung

Beachte Speichererweiterung bedeutet
„weiteren Speicherbaustein (gleicher Art) hinzufügen“,
nicht „vorhandenen Speicherbaustein durch größeren Speicherbaustein ersetzen“

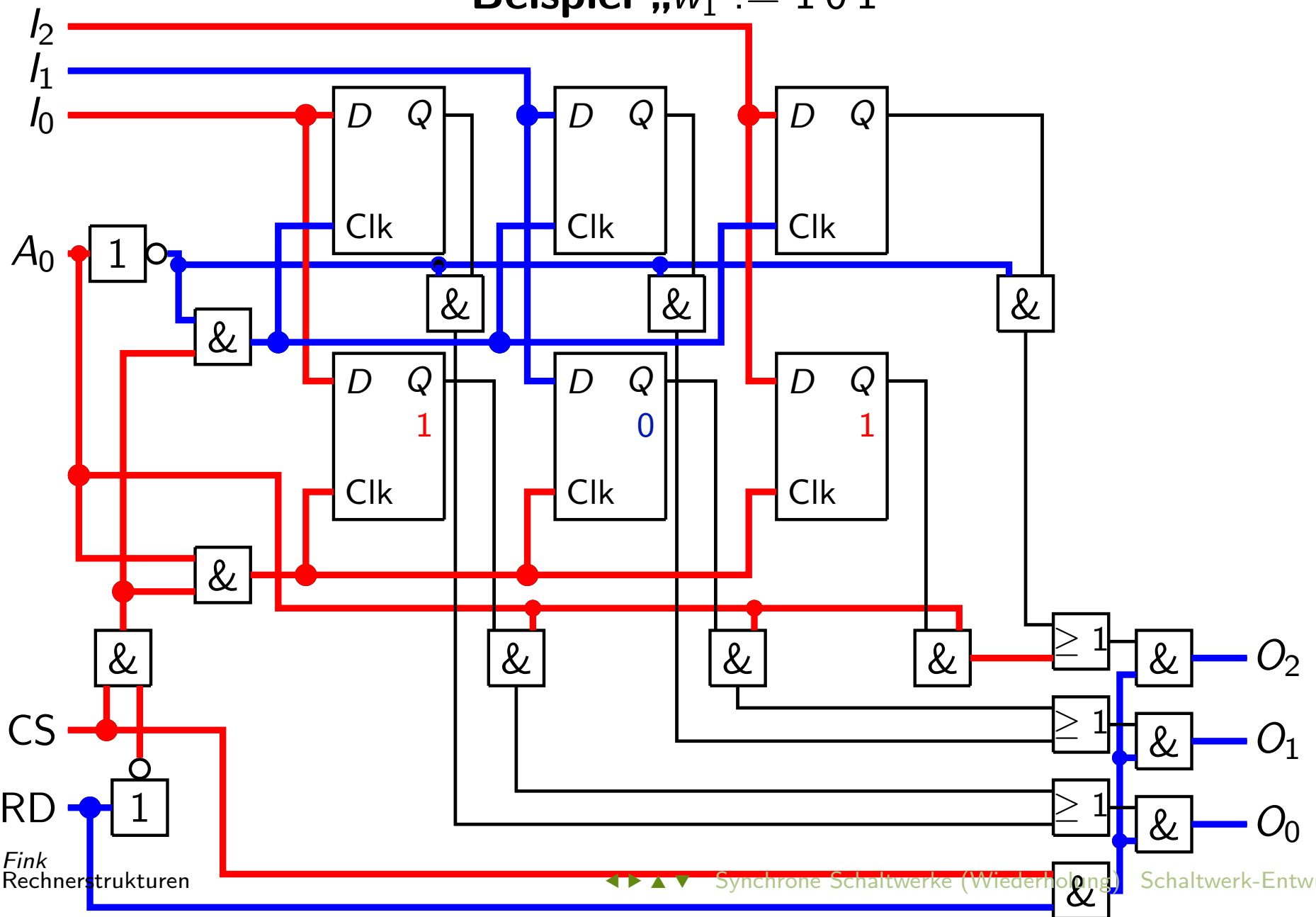
Wie können wir das unterstützen?

Idee zusätzliche Eingabe „Chip Selected“ (CS)

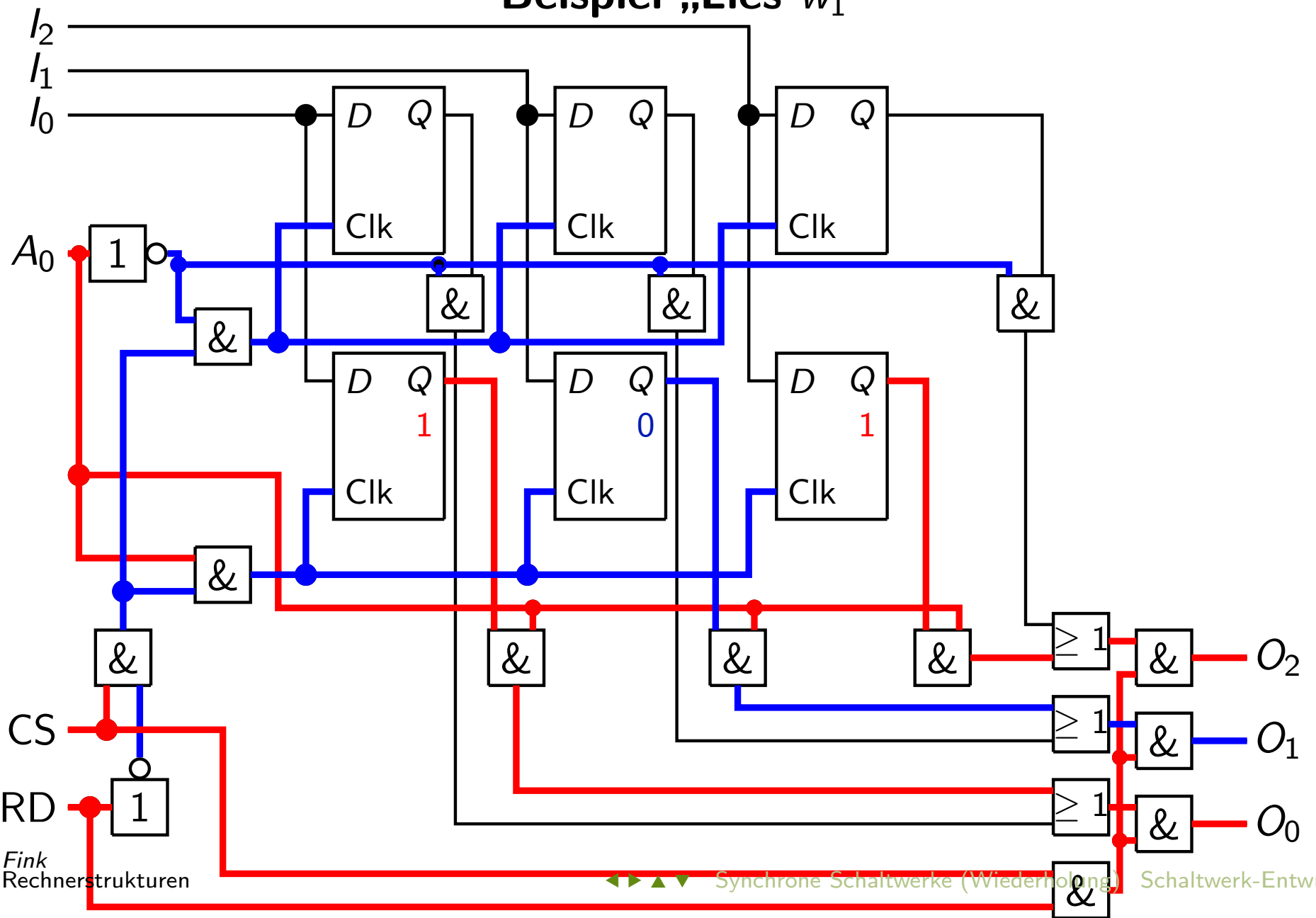
Speicher für zwei 3-Bit-Wörter



Beispiel „$w_1 := 1\,0\,1$“



Beispiel „Lies w_1“



Realistischere Speichergrößen

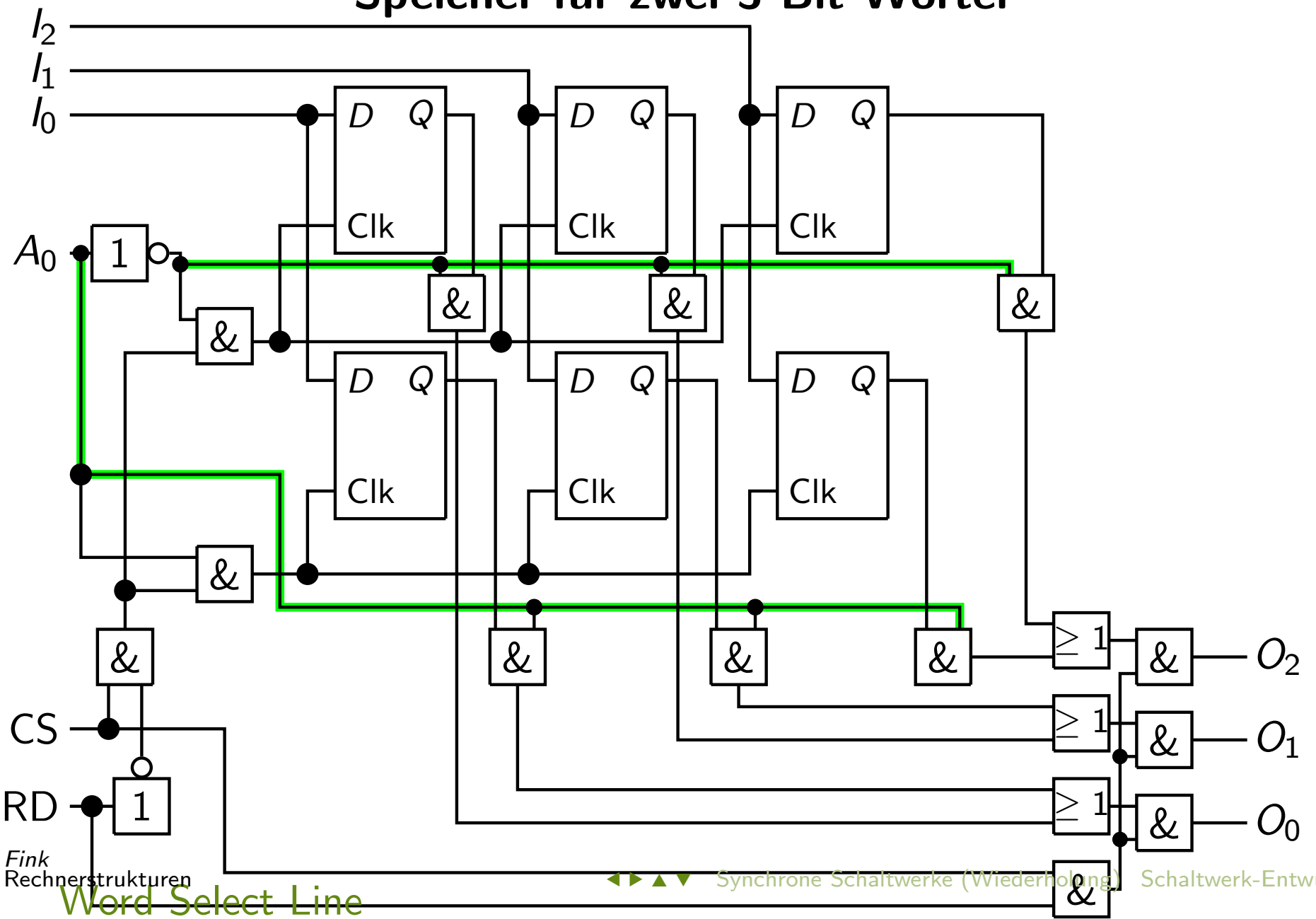
Beobachtung Speichergröße „2 Wörter“ ist nicht realistisch
auch nicht bei Benutzung einiger Bausteine

Wie kommen wir zu realistischen Speichergrößen?

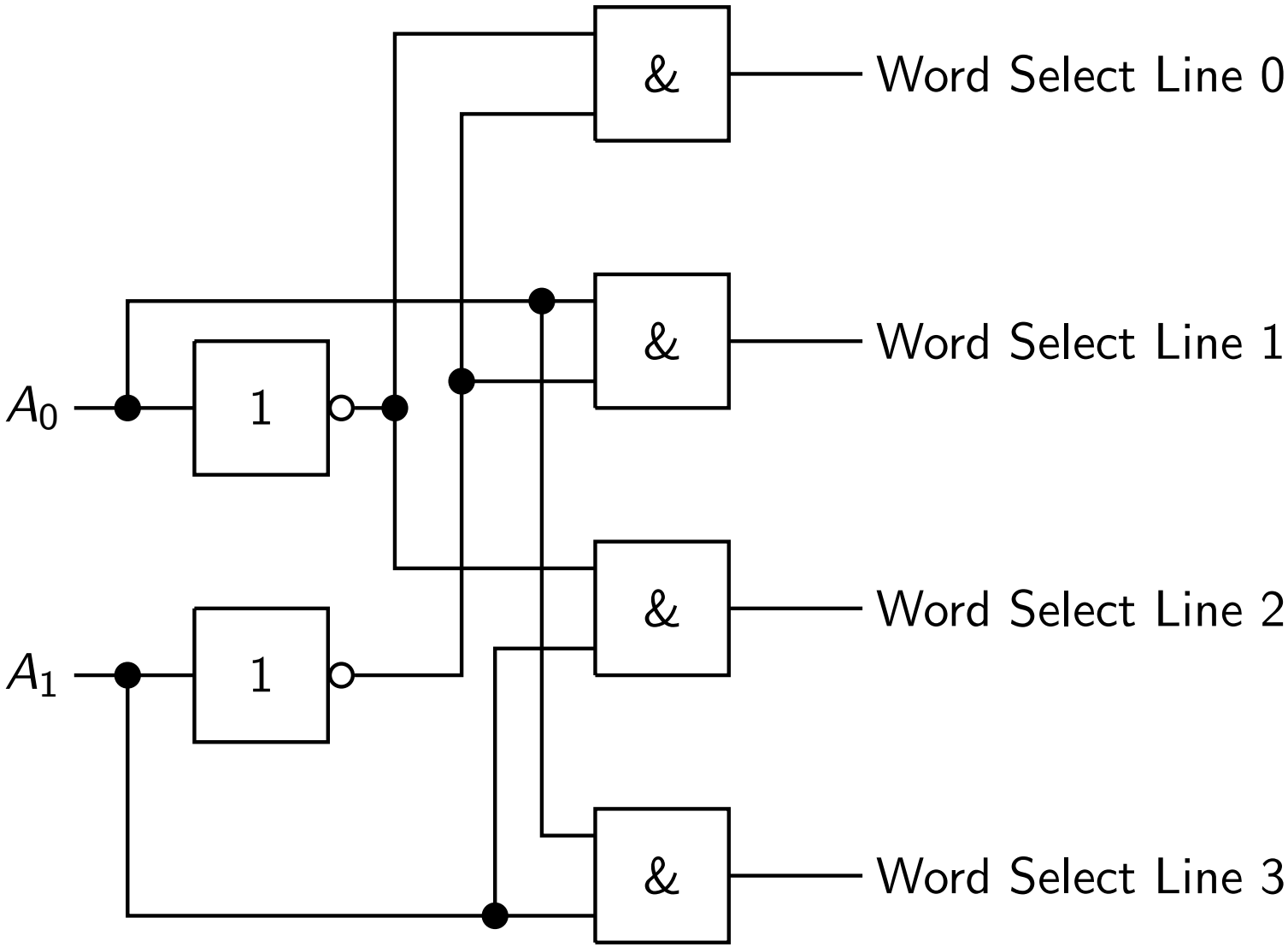
klar größere Wortlänge funktioniert im Prinzip gleich

offen Wie verallgemeinern wir auf > 2 Wörter?

Speicher für zwei 3-Bit-Wörter



Word Select Lines für vier Wörter



Speicher für viele Wörter

allgemein 2^k Wörter speichern

klar k Adressleitungen benötigt

zunächst formale Beschreibung als boolesche Funktion

$f : \{0,1\}^k \to \{0,1\}^{2^k}$
mit $f(A_0, A_1, \ldots, A_{k-1}) = (s_0, s_1, \ldots, s_{2^k-1})$

$$\text{mit } s_i = \begin{cases} 1 & \text{falls } (A_{k-1}\, A_{k-2} \cdots A_0)_2 = i \\ 0 & \text{sonst} \end{cases}$$

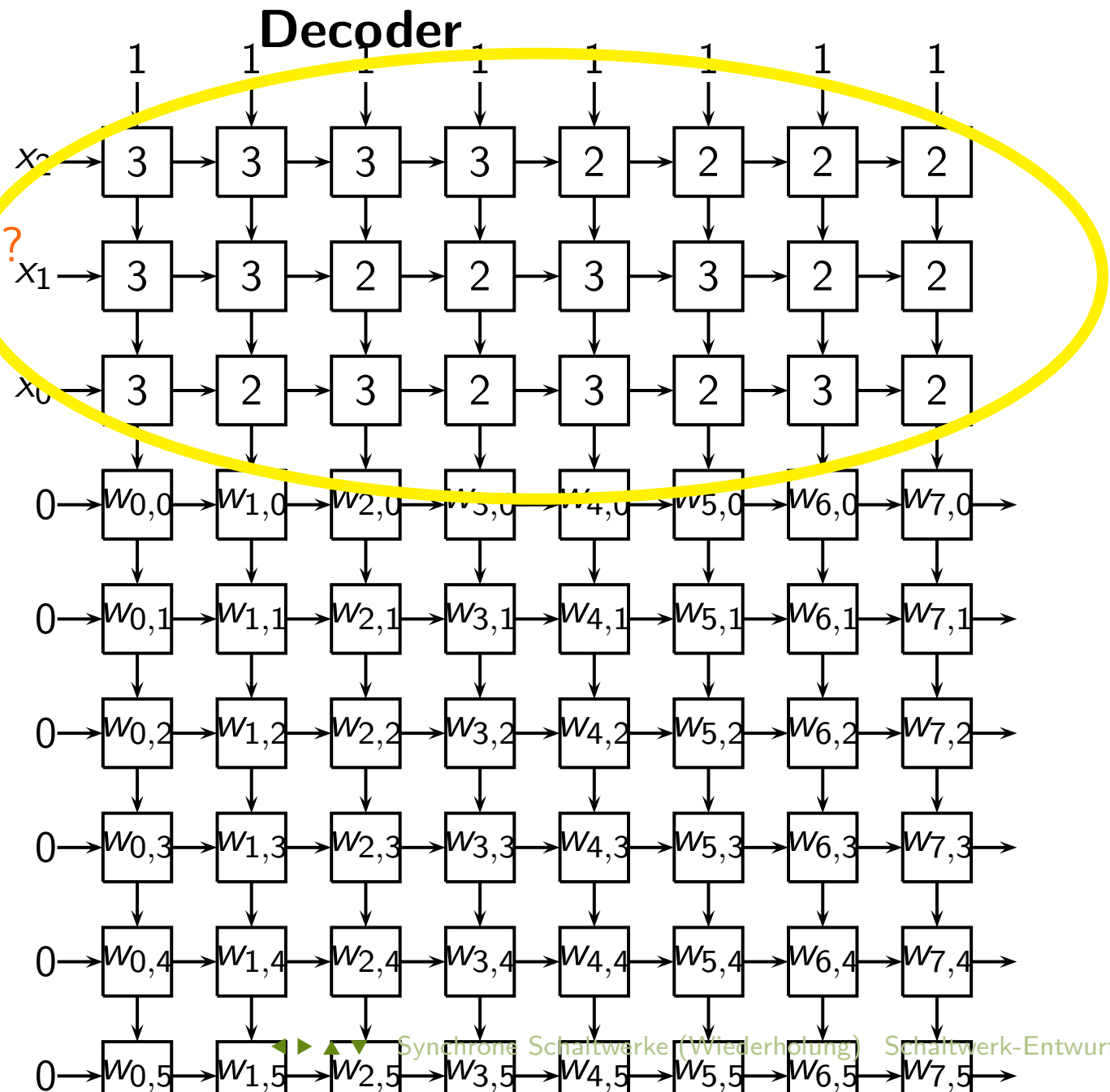
Name der Funktion Decoder
genauer $k \times 2^k$-Decoder

Nachdenken

Kommt uns das nicht bekannt vor?

Beobachtung

Und-Teil realisiert Decoder



Demultiplexer

Erinnerung $\mathrm{MUX}_d(y_1, \ldots, y_d, x_0, x_1, \ldots, x_{2^d-1}) = x_{(y_1\, y_2 \cdots y_d)_2}$

Erinnerung Decoder als boolesche Funktion

$\mathrm{DECODE}_d \colon \{0,1\}^d \to \{0,1\}^{2^d}$ mit

$\mathrm{DECODE}_d(A_0, A_1, \ldots, A_{d-1}) = (s_0, s_1, \ldots, s_{2^d-1})$ mit

$$s_i = \begin{cases} 1 & \text{falls } (A_{d-1}\, A_{d-2}\, \ldots A_0)_2 = i \\ 0 & \text{sonst} \end{cases}$$

Demultiplexer $\mathrm{DEMUX}_D \colon \{0,1\}^{d+1} \to \{0,1\}^{2^d}$ mit

$$\begin{aligned} &\mathrm{DEMUX}_D(x, A_0, A_1, \ldots, A_{d-1}) \\ &= \big(x \wedge \{\mathrm{DECODE}_d(A_0, A_1, \ldots, A_{d-1})\}_0, \\ &\quad x \wedge \{\mathrm{DECODE}_d(A_0, A_1, \ldots, A_{d-1})\}_1, \ldots, \\ &\quad x \wedge \{\mathrm{DECODE}_d(A_0, A_1, \ldots, A_{d-1})\}_{2^d-1}\big) \end{aligned}$$

Speicher

Realisiert man Speicher wirklich so?

... nicht für Hauptspeicher von Rechnern (kompakter als DRAM [dynamic random access memory] möglich)

tatsächlich typische SRAM-Realisierung (SRAM = static RAM)

Eigenschaften

- dauerhaft
- schnell
- Zugriffszeit von Daten unabhängig
- teuer
- hoher Stromverbrauch

Bemerkung typisch für Cache- oder Register-Speicher

Cache

$128\ \mathrm{K} \times 8$
$= 128 \cdot 1024 \times 8$
$= 2^{17} \times 8$
darum 17 Adress-leitungen A_0–A_{16}

Chip Enable
bei uns CS
Write Enable
bei uns RD $= 0$
Output Enable
bei uns RD $= 1$

Beobachtung
$128 \cdot 1024 \cdot 8$
$= 1\,048\,576$
Flip-Flops

SRAM

128K X 8 HIGH SPEED CMOS STATIC RAM

FEATURES

- Fast Address Access Times : 10/12/15ns
- Single 3.3V ±0.3V power supply
- Center power/ground pin configuration
- Low Power Consumption : 110/105/100mA
- TTL I/O compatible
- 2.0V data retention mode
- Automatic power-down when deselected
- Available packages :
  - 32-pin 300 mil and 400 mil SOJ
  - 32-pin TSOP 8x13.4mm and 8x20mm
  - 36-Ball CSP (8x10mm)

GENERAL DESCRIPTION

The T14L1024N is a one-megabit density, fast static random access memory organized as 131,072 words by 8 bits. It is designed for use in high performance memory applications such as main memory storage and high speed communication buffers. Fabricated using high performance CMOS technology, access times down to 10ns are achieved.

PART NUMBER EXAMPLES

	PACKAGE	SPEED
T14L1024N-10J	SOJ 300mil	10ns
T14L1024N-10W	SOJ 400 mil	10ns
T14L1024N-10P	TSOP 8x13.4mm	10ns
T14L1024N-10H	TSOP 8x20mm	10ns
T14L1024N-10C	36-Ball CSP	10ns

BLOCK DIAGRAM