Technische Informatik 2 - Rechnerarchitektur

Eigenes Skript

Studiengang Angewandte Informatik

Duale Hochschule Baden-Württemberg Karlsruhe

von

Andre Meyering

Kurs:	TINF16B2
Dozent:	Prof. Dr. Jürgen Röthig
Semester:	3. Semester (09.10.2017)
letzte Änderung:	*6. November 2017*

Dies ist das eigene Skript für „Rechnerarchitektur“ bei Herrn Prof. Dr. Jürgen Röthig für das 3. Semester im Jahr 2017. Es enthält fast alles, was im Unterricht an die Tafel geschrieben oder besprochen wurde. Die LaTeX-Dateien sollten sich im gleichen Share befinden, in dem du diese PDF-Datei gefunden hast.

Bei Fragen, Fehlern oder Ergänzungen – oder sollten die LaTeX-Dateien fehlen – wende dich bitte an dhbw@andremeyering.de. Ich hoffe, diese PDF hilft dir beim Lernen.

©2017- keine Garantie auf Korrektheit

Inhaltsverzeichnis

1 Vorwort

Herr Röthig schreibt alles, was für seine Klausuren von Bedeutung ist, an die Tafel. Es ist daher nur zu empfehlen, alles mitzuschreiben, da er kein Skript besitzt und auch keinen Foliensatz. Der Unterricht ist im Vergleich zu anderen Dozenten unterscheidet sich darin, dass während der Klausur keine Hilfsmittel verwendet werden dürfen. Dafür besteht die Klausur zu 90% nur aus Abfrageaufgaben.

Dieses Skript enthält *alles*, was Herr Röthig 2017 an Wissen voraussetzt. Auf den letzten Seiten dieses Skripts findet sich zusätzlich noch eine Übungsklausur. Die Klausuren unterscheiden sich jedes Jahr nur um einige wenige Aufgaben. Ist man zwei, drei Übungsklausuren durchgegangen, so ist die Klausur einfach zu bestehen.

Zusammen mit meinem Kurs TINF16B2 haben wir dieses Skript ausgedruckt und korrigiert. Inhaltliche Fehler sollten daher (fast) keine mehr enthalten sein.

Ich wünsche dir viel Erfolg bei Herrn Röthig im Fach Rechnerarchitektur (Technische Informatik 2). Solltest du diese Skript erweitern wollen, so kannst du dich an dhbw@andremeyering.de wenden.

2 Einstieg in die Vorlesung

Dozent:	Prof. Dr. Jürgen Röthig
Modul:	Technische Informatik II
Fach:	Rechnerarchitektur

2.1 Digitaltechnik (Vorlesung)

- 36h/33h / 4h pro Woche
- Klausur: 21.12.2017 | 60min (ohne Hilfsmittel, Verrechnung mit Betriebssysteme)
- kein Skript, kein Foliensatz

2.2 Übersicht

1. Einführung, Begriffsbildung, Historie, Fundamentalarchitektur
2. Rechenwerke
3. Speicherwerk: Hauptspeicherorganisation
   Speicher: Cache, nicht-flüchtige Speichertechnologie
4. ausgewählte Kapitel aus Steuer-/Ein- und Ausgabewerke sind in 2 + 3 enthalten.

3 Rechner

Rechner

Hilfsmittel zum Durchführen von „Rechnungen“.

- schneller
- fehlerfreier
- besseres Speichervermögen

Rechenmaschine

- Abakus (mechanisch, digital)
- Rechenschieber (mechanisch, analog)

Arbeitsweise

Man unterscheidet zwischen „mechanisch vs elektrisch“ und „digital vs analog“. Moderne „Rechner“ (PC & Co.) arbeiten elektrisch und digital. Dem gegenüber stehen elektrische Analogrechner, die um die 1920er genutzt wurden.

3.1 Geschichte

3.1.1 Elektrischer Digitalrechner

ZUSE Z1, Z2 (ab ~1940)

Relais als zentrale Bauteile (elektromagnetischer Schalter mit Elektromagnet)

- ⊕ Automatismus möglich
- ⊖ langsame Geschwindigkeit
- ⊖ großer Platzverbrauch
- ⊖ Geräusche beim Schalten
- ⊖ hoher Energieverbrauch beim Schalten
- ⊖ großer Verschleiß

ENIAC (~1945)

Die ENIAC besitzt als zentrales Bauteil eine Elektronenröhre. Eine Elektronenröhre ist ein eigentlich analog arbeitender Verstärker, wird hier aber als digitaler Schalter genutzt. Die Funktionsweise wird in Abbildung 3.1 dargestellt, wobei die Kathode negativ und die Anode positiv geladen sind.

- ⊕ sehr hohe Geschwindigkeit
- ⊖ großer Platzverbrauch
- ⊖ ständiges Summen bei $50Hz$ oft möglich
- ⊖ hoher, ständiger Energieverbrauch
- ⊖ großer Verschleiß

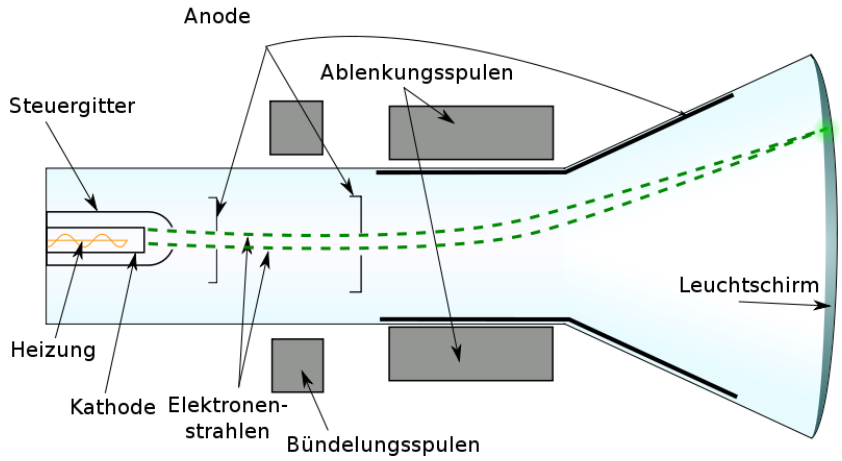

Abbildung 3.1: Funktionsweise Kathodenstrahlröhre [Quelle: Wikipedia]

Moderne Rechner Moderne transistorisierte Digitalrechner (z. B. UNIVAC ab Ende der 1950er).

- Transistor als zentrales Bauteil. Ein Transistor ist ein analog arbeitender Verstärker, wird hier aber als digital arbeitender Schalter genutzt.

⊕ sehr hohe Geschwindigkeit
⊕ sehr geringer Platzverbrauch
⊕ keine Geräuschentwicklung (außer Lüfter)
⊕ sehr niedriger Energieverbrauch
⊕ geringer Verschleiß

3.2 Fundamentalarchitektur

3.2.1 von-Neumann-Architektur

In Abbildung 3.2 wird die von-Neumann-Architektur vereinfacht dargestellt. Diese besteht aus:

Zentraleinheit (CPU) Die CPU besteht aus:

Rechenwerk Rechnen mit Zahlen und logischen Werten

Steuerwerk Zuständig für das Steuern und Koordinieren aller anderen Komponenten ⇒ Interpretation und Ausführung des (Maschinensprachen-)Programms

Speicherwerk (Hauptspeicher, Primärspeicher)
Speichern von Informationen (sowohl Programmcode als auch Nutzdaten gleichermaßen)

Bus verbindet alle Komponenten und ermöglicht den Informationsaustausch/Datenfluss zwischen ihnen.

Eingabewerk „Schnittstelle“ für Eingabegeräte (z. B. USB-Controller, S-ATA-Controller) nicht jedoch das Peripheriegerät selbst (also nicht die Tastatur)

Ausgabewerk „Schnittstelle“ für Ausgabegeräte (z. B. Grafikkarte)

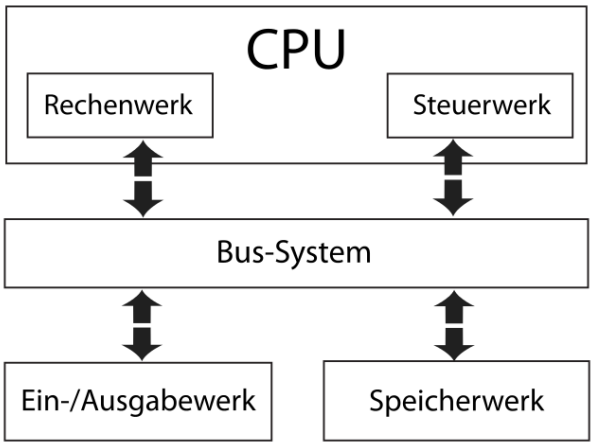

Abbildung 3.2: Vereinfachte Darstellung der von-Neumann-Architektur

3.2.2 Harvard-Architektur

Die Harvard-Architektur ist ähnlich der von-Neumann-Architektur, besitzt aber anstatt eines gemeinsamen, zwei getrennte Speicherwerke für Nutzdaten und Programmcode. Zusätzlich kann noch ein optionales zweites Eingabewerk existieren, welches nur für den Programmcode vorhanden ist. Das Speicher- und Eingabewerk für den Programmcode wird über einen zweiten Bus angebunden.
Dadurch ist eine klare physikalische Trennung von Programmcode und Nutzdaten möglich. Abbildung 3.3 auf der nächsten Seite zeigt die Harvard-Architektur und wie sich diese von der von-Neumann-Architektur unterscheidet.

3.2.3 Vergleich

Tabelle 3.1 auf Seite 7 vergleicht die von-Neumann-Architektur mit der Harvard-Architektur.

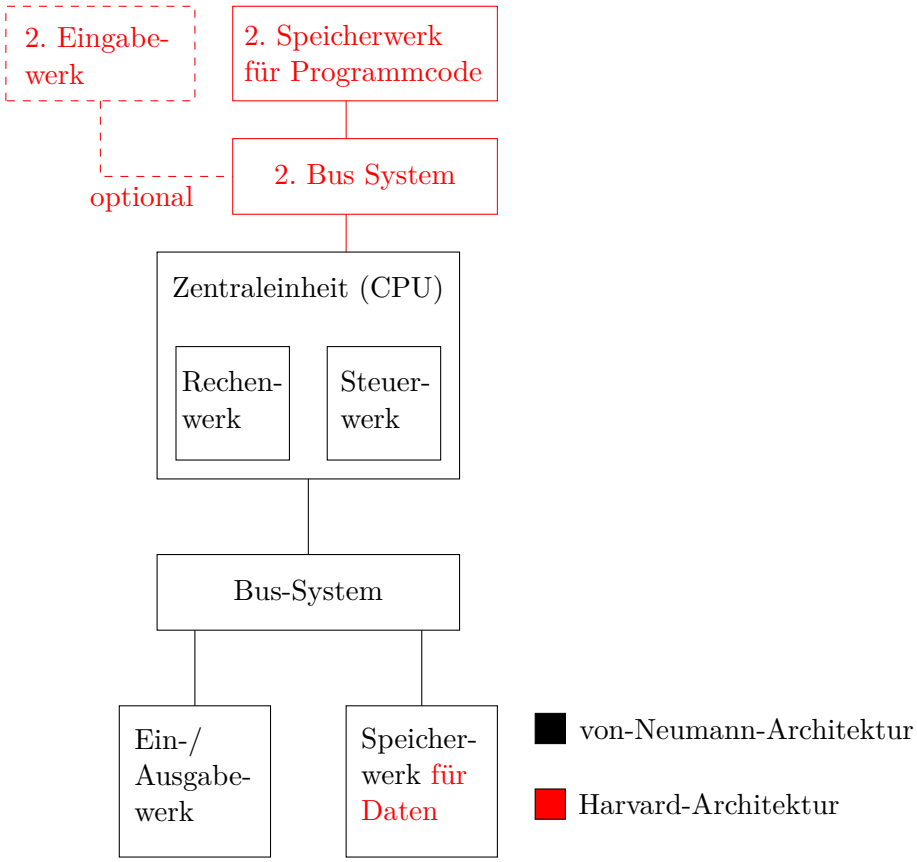

Abbildung 3.3: Vereinfachte Darstellung der Harvard-Architektur

	von-Neumann-Architektur		**Harvard-Architektur**
⊖	Virenanfälligkeit: Nutzdaten können als Programm ausgeführt werden	⊕	nahezu immun gegen unabsichtlichen Virenbefall
⊕	universelle Programmierbarkeit (z. B. Compiler-Ausgabe wird als Programme ausgeführt)	⊕	keine (unbeabsichtigten oder ungewollten) Änderungen an der Betriebssoftware möglich
⊕	flexible Speicheraufteilung zwischen Programmcode und Daten	⊖	komplexer und teurer (durch 2 Bus und ggf. 2 Eingabewerke)
⊖	möglicher Flaschenhals Bus & Speicherwerk	⊖	schwer update-fähig
⊕	kostengünstig	⊕	bessere Performance möglich durch gleichzeitigen Zugriff auf beide Speicherwerke
		⊖	ohne zweites Eingabewerk keine Möglichkeit anderen Programmcode auszuführen.
		⊖	unflexible Aufteilung des Speichers, wenn das eine Speicherwerk voll ist, kann das andere Speicherwerk nicht genutzt werden.
	Einsatz		
•	übliche PC-Architektur – „Universal-PC“	•	„embedded systems“ (z. B. in Waschmaschinen, KFZ-Elektronik, etc.)
		•	Smartphones & Co.
		•	Bestandteile von PCs: BIOS, CPU-Cache in modernen CPUs (Trennung in Cache für Programmcode und Nutzdaten), NX-Flag (Non-Executable) im Hauptspeicher

Tabelle 3.1: Vergleich der von-Neumann- und Harvard-Architektur

4 Rechenwerk / Rechnen in Hardware

Grundlagen: Addition

	4711
	+0815
Übertrag	0100
	5526

Die Addition von mehrstelligen Zahlen wird reduziert auf die Addition von zwei (oder drei) einstelligen Zahlen (bzw. Ziffern) zu einer einstelligen Zahl sowie einem einstelligen Übertrag, also einer zweistelligen Zahl als Ergebnis ⇒ genauso funktioniert dies im Binärsystem.

Hinweis

Herr Röthig meint mit „einstellig“ und „mehrstellig“ die Anzahl der Zahlen und nicht die Stellenanzahl einer einzelnen Zahl. Im Beispiel oben werden zwei Zahlen addiert und daraus ergeben sich zwei weitere Zahlen als Ergebnis.

4.1 Schaltnetzsysnthese – Wiederholung / Grundlegendes

Schaltnetz Kann nur die derzeitigen Eingangsdaten verarbeiten, da keine Rückkopplung vorliegt

Schaltwerk „Hat ein Gedächtnis“, da eine Rückkopplung vorliegt

Vollkonjunktion/Minterm `UND`-Verknüpfung aller vorkommenden Variablen entweder in negierter oder nicht-negierter Form

Disjunktive Normalform (DNF) Eine Disjunktion (`ODER`-Verknüpfung) von Konjunktionstermen (`UND`-Verknüpfungen).

Disjunktive Minimalform (DMF) Ist die minimale Darstellung einer Ausgabefunktion und damit eine Vereinfachung einer DNF

4.1.1 Schaltungsanalyse

Eine Schaltungsanalyse ist die Bestimmung des „Aufwands“. Dabei kann der Aufwand sein:

- „Hardware-Aufwand“ (in Anzahl an Transistoren)
- Zeitaufwand (in Gatterlaufzeit)

4.1.2 Warum soll der Zeitaufwand analysiert werden?

Es wird der Zeitaufwand betrachtet, da Gatter Schaltzeiten haben, welche typischerweise ~10 Pikosekunden betragen. Insgesamt werden bei einem Signaldurchgang auf dem IC sehr viele Gatter durchlaufen. Damit sind die Schaltzeiten um Größenordnungen größer als die reine Laufzeit der Signale auf dem Leiter angegeben (letztere wird vernachlässigt, Zeitverzögerung wird in „Anzahl Gatterlaufzeiten (GLZs)“ angegeben).

4.2 Halbaddierer

Vollzieht die Addition von zwei einstelligen Binärzahlen a und b zu einer zweistelligen Binärzahl $c_{out}s$ (Übertrag und Summe). Schaltsymbol und Schaltnetz des Halbaddierers werden in Abbildung 4.1 dargestellt.

Die folgende Tabelle zeigt den Gedankenweg, wie ein Halbaddierer funktioniert.

Nr.	b	a	c_{out}	s	Minterm	DNF
0	0	0	0	0		
1	0	1	0	1	$\overline{b}a$	$c_{out} = ba$
2	1	0	0	1	$b\overline{a}$	$s = \overline{b}a \vee b\overline{a}$
3	1	1	1	0	ba	$\Rightarrow$ beides gleichzeitig auch DMF

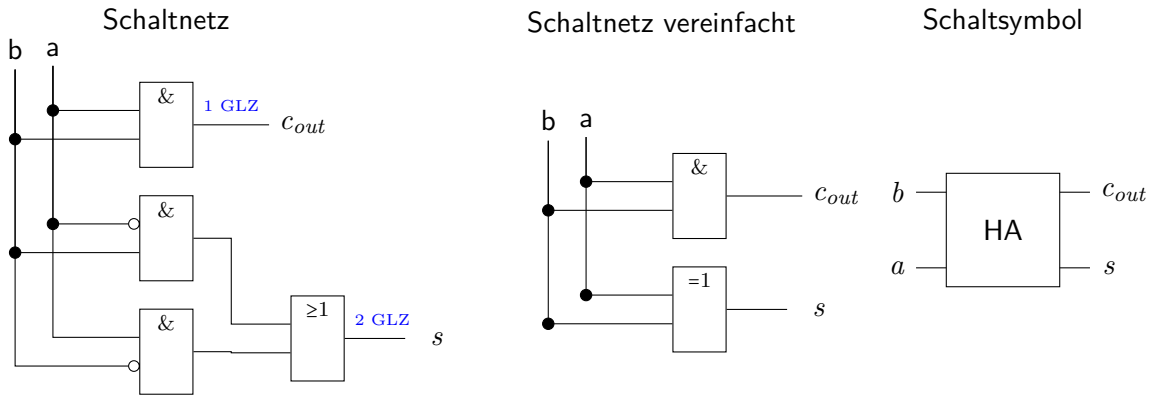

Abbildung 4.1: Halbaddierer – Schaltnetz und Schaltsymbol

4.3 Volladdierer

Vollzieht die Addition von drei einstelligen Binärzahlen a, b und c_{in} zu einer zweistelligen Binärzahl $c_{out}s$ (Übertrag und Summe). Schaltsymbol und Schaltnetz des Volladdierer werden in Abbildung 4.2 dargestellt.

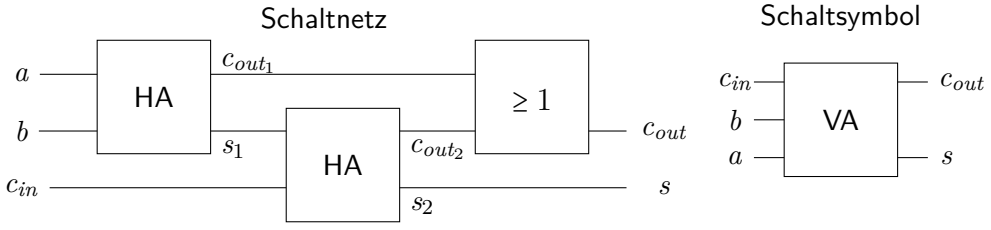

Abbildung 4.2: Volladdierer – Schaltnetz und Schaltsymbol

Hinweis: Für die Verknüpfung von c_{out_1} und c_{out_2} zu c_{out} wäre eigentlich ein `XOR` notwendig. Da aber der Fall $c_{out_1} = c_{out_2} = 1$ (also beide Eingänge des `XOR` „1“) nie auftritt, reicht ein `OR`.

4.4 Paralleladdierer (4-Bit-Ripple-Carry-Paralleladdierer RC-PA)

Der RC-PA ist ein mehrstelliger Addierer für Binärzahlen. In den folgenden Beispielen ist er ein Addierer vierstelliger Binärzahlen $a_3a_2a_1a_0$ und $b_3b_2b_1b_0$. Das Ergebnis ist $s_4s_3s_2s_1s_0$ und somit eine 5-stellige Zahl. s_4 ist der Überlauf.

Abbildung 4.3 zeigt das Schaltnetz und Schaltsymbol eines Paralleladdierers.

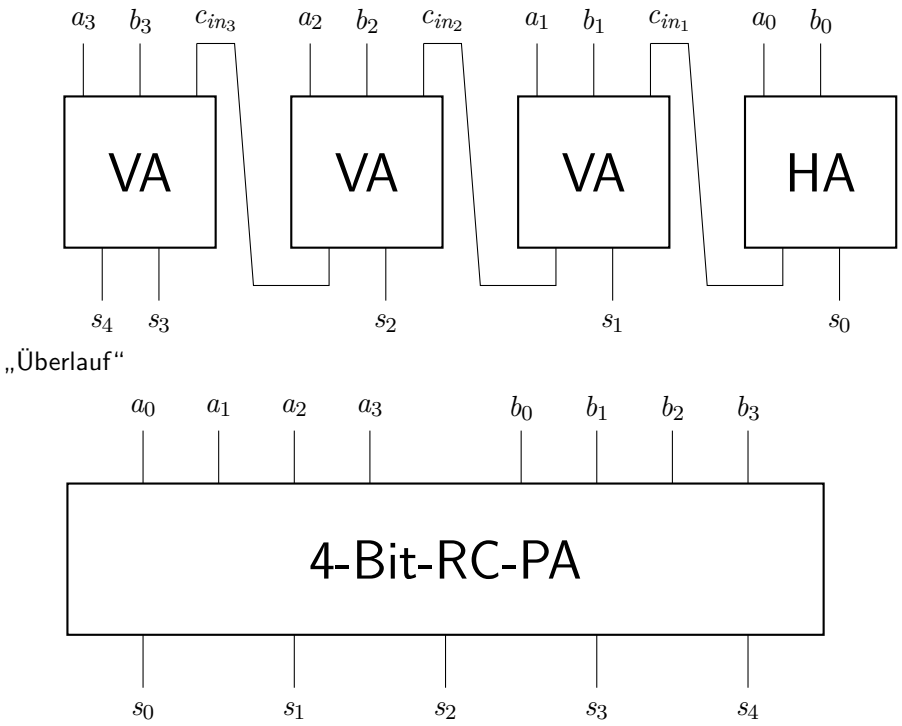

Abbildung 4.3: RC-Paralleladdierer – Schaltnetz und Schaltsymbol

Hinweis

Ein n-Bit RC-PA ist ein Schaltnetz, kein Schaltwerk! Eine zeichnerische Anordnung mit Verbindungen nur nach unten ist nämlich möglich.

4.4.1 Hardwareaufwand des 4 Bit RC-PA

für HA:2 Transistoren für c_{out} und 6 Transistoren für s $\Rightarrow$ 8 Transistoren
für VA: 2 HA und 2 Transistoren für c_{out} $\Rightarrow 2 \cdot 8 + 2 = 18$ Transistoren

Für den 4 Bit RC-PA werden benötigt:

$$\begin{aligned} 1 \text{ HA} + (n-1) \text{ VA} &\Rightarrow 8 \text{ Transistoren} + (n-1) \cdot 18 \text{ Transistoren} \\ &\Rightarrow 8 + (18n - 18) \text{ Transistoren} = 18n - 10 \text{ Transistoren} \\ &= O(n) \end{aligned}$$

Dies heißt, dass der HW-Aufwand linear mit der Breite der Summanden steigt. Dies ist gut, denn besseres (also weniger Aufwand) ist kaum zu erwarten.

4.4.2 Zeitaufwand des 4 Bit RC-PA

für HA: max. 2 Gatterlaufzeiten (GLZs) („Tiefe 2“, siehe Abbildung 4.1 auf Seite 9)
für VA: max. 4 Gatterlaufzeiten (GLZs)

Beim RC-PA liegen die einzelnen s_i nach unterschiedlicher Zeit an. s_i wird nach $(i+1)\cdot 2$ GLZ erreicht. Das längste s_i ist bei n-Bit-RC-PA $i = n-1$ und damit ergibt sich ein Zeitaufwand bei n-Bit-RC-PA von $2n$ GLZ!

Dies ist ein schlechter Zeitaufwand bei einem Paralleladdierer, denn zu erwarten wäre $O(1)$!
Auswirkung: Beim Wechsel von 32- auf 64-Bit-CPU hätte sich die Taktfrequenz halbiert. Daraus lässt sich folgern, dass kein 64-Bit-RC-PA in der CPU verbaut ist.

Hinweis

Anmerkung zum RC-PA: Die $2n$ GLZ Zeitaufwand werden nur im „schlimmsten Fall“ bei einer ununterbrochene Kette von c_{out}-Ausgängen, welche sich **alle** im Laufe der Berechnung von `0` auf `1` ändern, erreicht.

Diese Zeit muss aber trotzdem abgewartet werden.

4.5 Paralleladdierer (4-Bit-Carry-Look-Ahead-Paralleladdierer CLA-PA)

Idee: Der c_{in}-Eingang wird nicht von vorausgehenden VA (oder HA) übernommen, sondern durch ein „magisches CLA-Schaltnetz“ nachberechnet. *Genauer*: Für die Berechnung von c_{in_i} müssen alle vorherigen Eingänge $a_j, b_j, j < i$ berücksichtigt werden. Abbildung 4.4 zeigt dieses „magische CLA-Schaltnetz“.

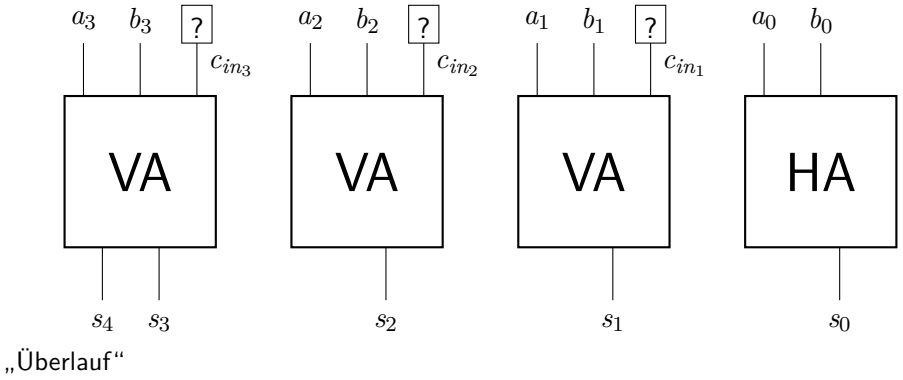

Abbildung 4.4: Carry-Look-Ahead-Paralleladdierer – Schaltnetz

Aber wie sieht das Schaltnetz (und die boolesche Formel) für die Berechnung von c_{in_i} aus?

$$\begin{aligned}
\text{HA}: \quad c_{out} &= a \wedge b \\
s_{out} &= a\overline{b} \vee \overline{a}b \\
\text{VA}: \quad c_{out} &= (a \wedge b) \vee (c_{in} \wedge (a\overline{b} \vee \overline{a}b))
\end{aligned}$$

$$\begin{aligned}
c_{in_1} = c_{out_0} &= a_0 \wedge b_0 && \textit{HA} \\
c_{in_2} = c_{out_1} &= (a_1 \wedge b_1) \vee (c_{in_1} \wedge (a_1\overline{b}_1 \vee \overline{a}_1 b_1)) && \textit{VA} \\
&= (a_0 \wedge b_1) \vee (a_0 \wedge b_0 \wedge (a_1\overline{b}_1 \vee \overline{a}_1 b_1)) && \textit{Einsetzen} \\
c_{in_3} = c_{out_2} &= (a_2 \wedge b_2) \vee (c_{in_2} \wedge (a_2\overline{b}_2 \vee \overline{a}_2 b_2)) && \textit{VA} \\
&= (a_2 \wedge b_2) \vee ((a_0 \wedge b_1) \vee (a_0 \wedge b_0 \wedge (a_1\overline{b}_1 \vee \overline{a}_1 b_1)) \wedge && \textit{Einsetzen} \\
&\quad (a_2\overline{b}_2 \vee \overline{a}_2 b_2)) \\
&\ldots
\end{aligned}$$

4.5.1 Zeitaufwand

Für die Berechnung der c_{in_i} kann jeweils eine DNF, DMF oder jede andere DxF (oder KxF) verwendet werden. Diese haben jeweils nur genau (bzw. maximal) 2 GLZ Zeitaufwand.

Insgesamt hat jedes s_i beim CLA-PA genau 6 GLZ Zeitaufwand (außer s_0: 2 GLZ, s_1: 5 GLZ). Somit haben wir einen konstanten Zeitaufwand von $O(1)$.

4.5.2 Hardwareaufwand des 4 Bit CLA-PA

Für den Aufwand für die c_{in_i}-Berechnung gilt die Annahme, das Schaltnetz wäre eine Realisierung der DNF.

- ⇒ Für jedes c_{in_i} gibt es insgesamt $2i$ Eingänge.
- ⇒ Insgesamt gibt es max. 2^{2i} verschiedene Vollkonjunktionen, welche in der DNF auftreten können.
  Jede dieser Vollkonjunktionen wird mit $2i$ Transistoren realisiert.
- ⇒ Falls alle Vollkonjunktionen verwendet werden müssten, wäre der Hardwareaufwand $2^{2i} \cdot 2i$ Transistoren.

Evtl. nicht korrekt. Die 15er haben da was anderes...

In der Realität werden natürlich nicht alle Vollkonjunktionen benötigt, sondern ein (vermutlich halbwegs konstanter) Anteil $0 < k < i$. Damit ist der Aufwand für $c_{in_i} = O(i \cdot 4^i)$ und somit der Aufwand für n-Bit-CLA-PA: $O(n^2 \cdot 2^n)$

Achtung

Im folgenden wird fälschlicherweise von einem Aufwand $O(n^2 \cdot 2^n)$ ausgegangen. Richtig wäre $O = (n^2 \cdot 4^n)$. Herr Röthig hat die O-Notation falsch vereinfacht. Der 2015er Jahrgang hat dies „noch falscher“ gemacht und zu $O(4^n)$ vereinfacht.

Der Hardwareaufwand steigt beim n-Bit-CLA-PA überexponentiell mit n. Beim Wechsel von 32-Bit auf 64-Bit-CLA-PA wäre der 16 Trillionen-fache Aufwand an Transistoren nötig gewesen.

Bei $n = 4$: $4^2 \cdot 2^4 = 16 \cdot 16 = 256$ Transistoren
Bei $n = 8$: $8^2 \cdot 2^8 = 64 \cdot 256 =$ ~16384 Transistoren (64-fache von $n = 4$) ⇒ *zu viel für die CPU*

Vergleichen: Siehe Notizen, etc.

4.5.3 Kombination mehrerer kleinen CLA-PAs

Der 32-Bit Addierer wird in acht 4-Bit-CLA-PA gesplittet (siehe Abbildung 4.5).

⇒ hintereinander geschaltet nach RC-Prinzip
⇒ damit ist das n der nicht-CLA-PA noch klein ⇒ erträglicher Hardwareaufwand

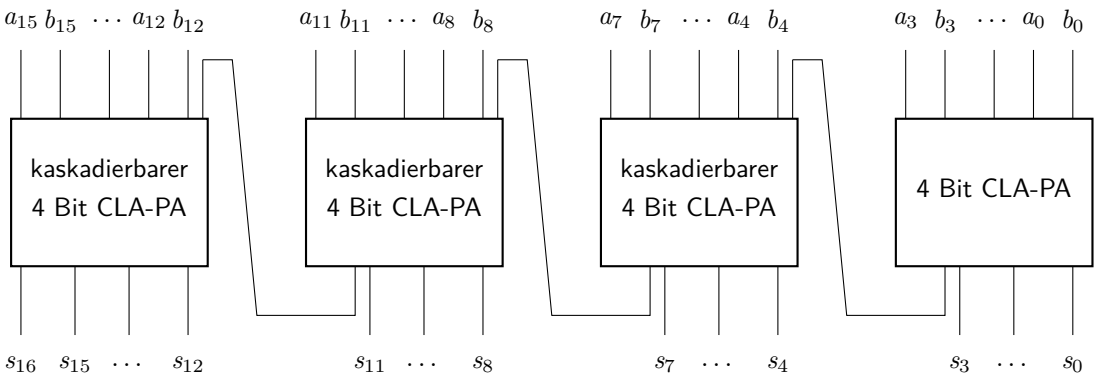

Abbildung 4.5: Kaskadierbarer 4-Bit-CLA-PA

4.6 Serielladdierer

Idee: Angelehnt an die Verfahrensweise des Menschen sollen die Stellen der beiden Summanden nacheinander (und nicht gleichzeitig) addiert werden. Dadurch wird nur **ein** VA und mehrere Schieberegister (SR) benötigt. Daher ist der Serielladdierer (SA) ein Schaltwerk, kein Schaltnetz! Abbildung 4.6 zeigt das Schaltwerk eines Serielladdierer.

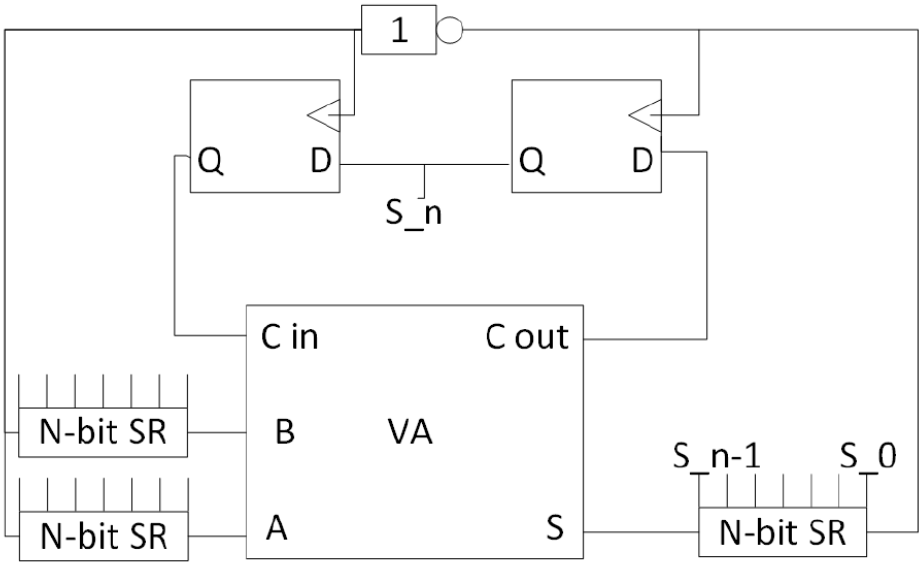

Abbildung 4.6: Serielladdierer

Als Tikz übernehmen

4.6.1 Zeitaufwand (n-Bit-SA)

Der Zeitaufwand für einen n-Bit-SA beträgt n Taktzyklen, also $O(n)$

Ist dies wie beim RC-PA?
Jein, denn 1 Taktzyklus dauert deutlich mehr als doppelt solang wie die Berechnung des VA (Sicherheitsmargen!). Beispiel: 1 Taktzyklus > ~10 GLZ ⇒ fünffache Berechnungszeit des RC-PA

4.6.2 Hardwareaufwand (N-Bit-SA)

1 VA	= 18 Transistoren
2 D-FF = $2 \cdot 6$	= 12 Transistoren. *(siehe Abbildung 4.7 rechts)*
3 n-Bit-SR	= $4 \cdot 6n = 18n$ Transistoren *(siehe Abbildung 4.7 links)*
Takterzeugung	*(im folgenden nicht näher betrachtet)*
gesamt	$18n + 30$ Transistoren

Zum Vergleich: RC-PA: $18n - 10$, d. h. der SA braucht 40 Transistoren mehr (bei längerer Bearbeitungszeit)!

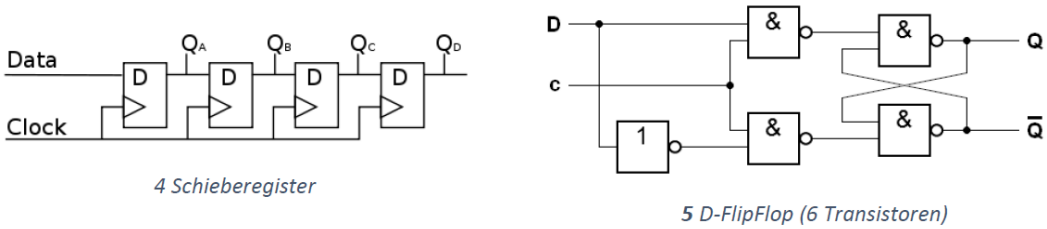

Abbildung 4.7: Schieberegister und D-Flip-Flop

Achtung

Die Takterzeugung muss in der Klausur für den Serielladdierer auf jeden Fall genannt werden, auch wenn sie hier nicht weiter betrachtet wird!

4.6.3 Hardwareoptimierung des Serielladdierers

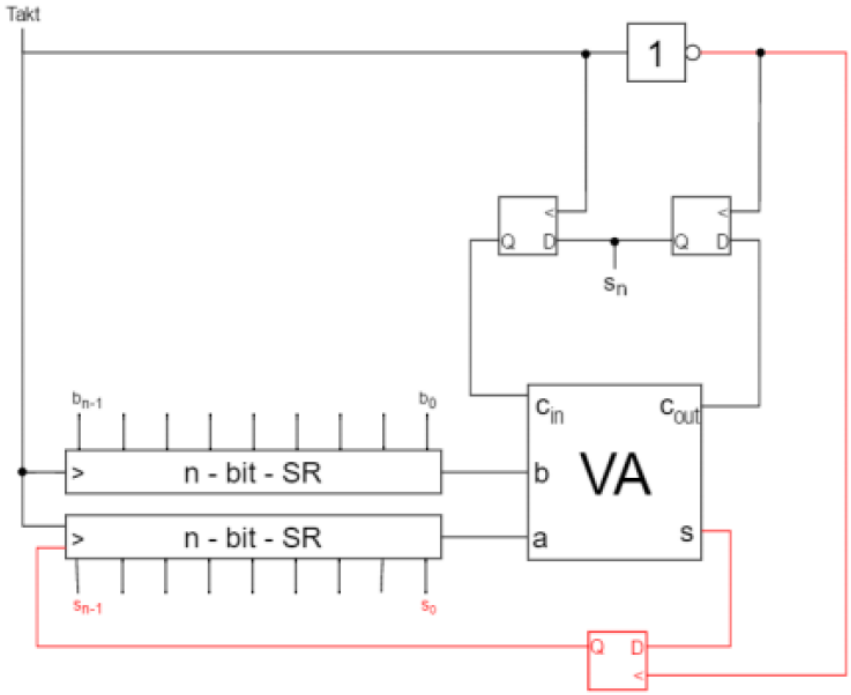

Abbildung 4.8: Serielladdierer mit Hardwareoptimierung

Wie in Abbildung 4.8 zu sehen ist, wird ein Schieberegister weniger benötigt $\Rightarrow 12n + 36$ Transistoren

Vergleich eines RC-PA mit dem verbesserten SA

Tabelle Tabelle 4.1 vergleicht einen RC-PA mit dem verbesserten SA.

n	RC-PA (18n – 10)	SA (12n +36)
1	8	48
2	26	60
3	44	72
4	62	84

Tabelle 4.1: Vergleich von Hardwareaufwand eines RC-PA mit dem verbesserten SA

Break-Even („*Gewinnschwelle*“): $18n - 10 = 12n + 36 \Rightarrow 46 = 6n \Rightarrow n = 7\frac{2}{3}$

$\Rightarrow$ Ab 8-Bit lohnt sich der SA

Mögliche Anwendung eines Serielladdierer:

- nicht im Rechenwerk der CPU (aufgrund der Geschwindigkeit)!
- aber möglicherweise in „Embedded System“, falls z. B. Sensordaten sowieso seriell angeliefert werden

$\Rightarrow$ eventuell sind sogar weitere Hardware-Einsparungen möglich!

4.7 Subtraktion

Es ist kein spezieller Hardware-Subtrahierer notwendig. Eine Subtraktion wird über die Addition des 2er-Komplements realisiert:

1. Bits invertieren (Inverter)
2. +1 addieren (Addierer)

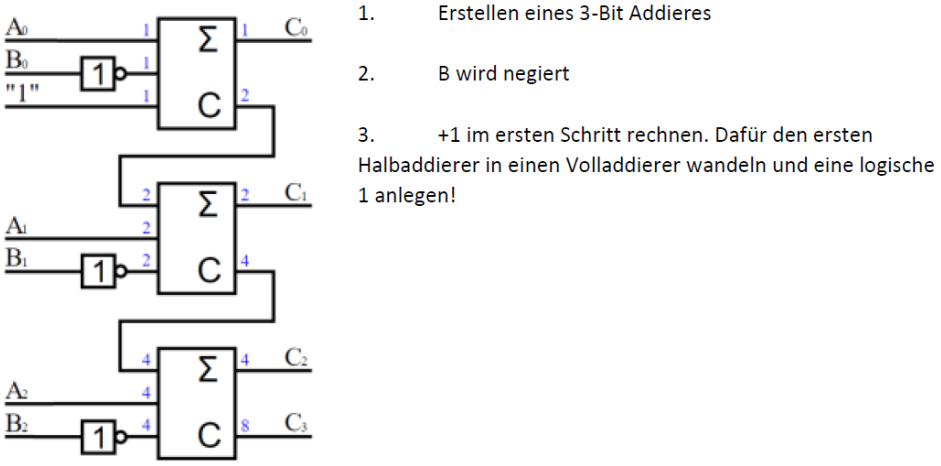

Abbildung 4.9: Schaltnetz Subtrahierer

4.8 Multiplikation

4711×815
35
67688
4711
23
23555
3839465

Mögliche Probleme / Schwierigkeiten

1. kleines 1×1 ist ein wenig Lernaufwand
2. Überträge beim kleinen 1×1 sind möglich
3. Addition von mehr als zwei Ziffern gleichzeitig
4. mehrstellige Überträge bei der Summenbildung möglich
5. für jede Zwischensumme muss der Addierer eine Stelle mehr verarbeiten

Punkt 5 wird deutlich durch eine Multiplikation einer 6- und 4-stelligen Zahl im Binärsystem „von links nach rechts“, wie sie in Tabelle 4.2 dargestellt wird.

101010×1010	(42×10)
101010	
000000	*7-stellige Addition*
1010100	*Zwischensumme*
101010	*8-stellige Addition*
11010010	*Zwischensumme*
000000	*9-stellige Addition*
110100100	*10-stelliges Ergebnis*

Tabelle 4.2: Schriftliche Multiplikation „von links nach rechts“

Abhilfe

b	a	ab
0	0	0
0	1	0
1	0	0
1	1	1

1. Kleines 1×1 im Binärsystem ist ein einfaches `UND`
2. Es gibt keine Überträge beim kleinen 1×1 im Binärsystem
3. Bilden von Zwischensummen
4. keine mehrstelligen Überträge bei Addition von zwei Summanden
5. rechten Faktor beginnend mit niederwertigster Stelle abarbeiten, siehe Tabelle 4.3

1 0 1 0 1 0 × 1 0 1 0	(42×10)
0 0 0 0 0 **0**	
1 0 1 0 1 0	p_0
1 0 1 0 1 **0**	
0 0 0 0 0 0	p_1
0 1 0 1 0 **1**	
1 0 1 0 1 0	p_2
0 1 1 0 1 0 0	
$p_9\ p_8\ p_7\ p_6\ p_5\ p_4\ p_3$	

Tabelle 4.3: Schriftliche Multiplikation „von rechts nach links“

4.8.1 Multiplikation mit Paralleladdierer

Ein $n \times m$-Bit-Parallelmultiplizierer (PM), bspw. ein 5×4-Bit-Parallelmultiplizierer ist in Abbildung 4.10 dargestellt.

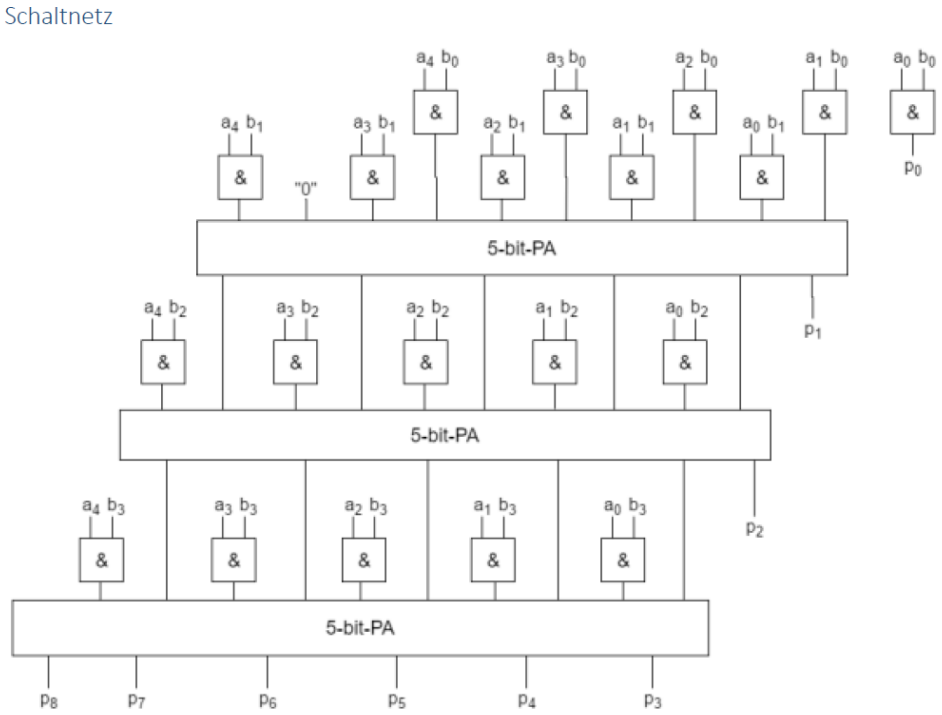

Abbildung 4.10: 5-Bit Parallelmultiplizierer

Hinweis zur Abbildung

Unbedingt auf die „0“ als Eingang achten! Ansonsten gibt es in der Klausur punktabzug!

Hinweis

In der Klausur muss evtl. ein 4-Bit Parallelmultiplizierer gezeichnet werden, also nicht mit 5 oder 6 Bit.

Analyse: Hardwareaufwand für die Multiplikation mit Paralleladdierer

Benötigt werden:

- $m-1$ n-Bit-PA
- $n \cdot m$ `UND` mit jeweils 2 Eingängen $\Rightarrow 2 \cdot n \cdot m$ Transistoren

Somit ergibt sich bei...

... Verwendung eines **RC-PA**:

$\Rightarrow$ n-Bit-RC-PA: $18n - 10$ Transistoren

$\Rightarrow$ davon $m-1$: $(m-1)(18n-10)$ Transistoren $= 18nm - 18n - 10m + 10$ Transistoren
$\Rightarrow$ Insgesamt: $20nm - 18n - 10m + 10$ Transistoren $\Rightarrow O(n)$

... Verwendung eines **CLA-PA**

- n-Bit-CLA-PA: $\approx O(n \cdot 2^n)$
- $\Rightarrow$ davon $m-1$: $(m-1) \cdot O(n \cdot 2^n) = O(n \cdot m \cdot 2^n)$

Hier bei verschieden großen Faktoren also besser $m > n$ bei Carry-Look-Ahead-Paralleladdierer (CLA-PA)) für geringeren HW-Aufwand.

Demgegenüber bei Verwendung von RC-PA: Besser $n > m$ für geringeren HW-Aufwand.

Besser umformulieren

Analyse: Zeitaufwand für die Multiplikation mit Paralleladdierer

1 GLZ für einstellige Multiplikation (UND-Gatter) sowie $(m-1)$ × Berechnungszeit(n-Bit-PA)

1. Annahme: PA sind n-Bit-RC-PA.
   Berechnungszeit eines n-Bit-RC-PA: $2n$ GLZ
   Insgesamt: $1 + (m-1) \cdot 2n = 2nm - 2n + 1$ GLZ
   Damit besser $n > m$ bei Verwendung von RC-PA um geringeren Zeitaufwand zu bekommen.

2. Annahme: PA sind n-Bit-CLA-PA
   Berechnungszeit eines n-Bit-CLA-PA: 6 GLZ
   Insgesamt: $1 + (m-1) \cdot 6$ GLZ $= 6m - 5$ GLZ $= O(m)$
   damit: besser $n > m$ bei Verwendung von CLA-PA, um geringeren Zeitaufwand zu bekommen
   $\lightning$ zu großer HW-Aufwand (wächst exponentiell mit n)

Kontrollieren...

Hinweis für die Klausur

Logarithmischen Aufwand für CLA-PA auf Wikipedia nachschauen. Dies wird wahrscheinlich in der Klausur abgefragt! Unserer Variante hat exponentiellen Hardwareaufwand und konstanten Zeitaufwand. Die Variante auf Wikipedia nicht.

4.8.2 Seriellmultiplizierer

Motivation: Noch engere Anlehung an das schriftliche Multiplikationsverfahren, um den Aufwand für die Addierglieder gering zu halten.

Siehe Abbildung 4.11.

Hinweis

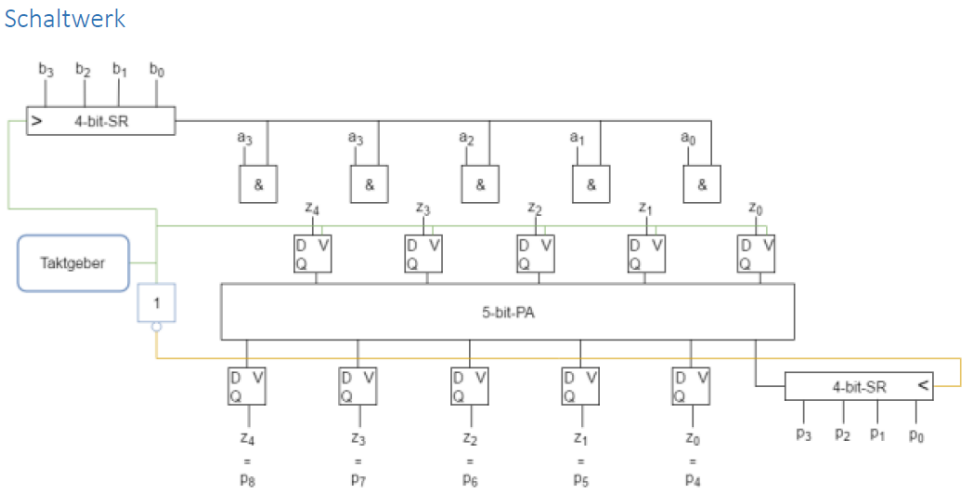

Abbildung 4.11: 5-Bit Seriellmultiplizierer

> Vor dem ersten Takt müssen alle D-FF auf `0` gesetzt werden („Reset-Eingang“ oder ähnliches)

5 Abkürzungsverzeichnis

CLA-PA Carry-Look-Ahead-Paralleladdierer 19
CPU central processing unit
D-FF D-Flip-Flop
DMF Disjunktive Minimalform 8
DNF Disjunktive Normalform 8
ENIAC Electronic Numerical Integrator and Computer
IC Integrated Circuit
GLZ Gatterlaufzeit 8
HA Halbaddierer
HW Hardware
PC Personal Computer
PA Paralleladdierer
PM Parallelmultiplizierer 18
RC-PA Ripple-Carry-Paralleladdierer
SA Serielladdierer 14
SR Schieberegister 14
UNIVAC Universal Automatic Computer
VA Volladdierer

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis

Stichwortverzeichnis