Rechneraufbau und Rechnerstrukturen

Von
Walter Oberschelp,
Gottfried Vossen

10., überarbeitete und erweiterte Auflage

Oldenbourg Verlag München Wien

Prof. Dr. Gottfried Vossen lehrt seit 1993 Informatik am Institut für Wirtschaftsinformatik der Universität Münster. Er studierte, promovierte und habilitierte sich an der RWTH Aachen und war bzw. ist Gastprofessor u.a. an der University of California in San Diego, USA, an der Karlstad Universitet in Schweden, an der University of Waikato in Hamilton, Neuseeland sowie am Hasso-Plattner-Institut für Softwaresystemtechnik in Potsdam. Er ist europäischer Herausgeber der bei Elsevier erscheinenden Fachzeitschrift Information Systems.

Prof. Dr. Walter Oberschelp studierte Mathematik, Physik, Astronomie, Philosophie und Mathematische Logik. Nach seiner Habilitation in Hannover lehrte er als Visiting Associate Professor an der University of Illinois (USA). Nach seiner Rückkehr aus den USA übernahm er den Lehrstuhl für Angewandte Mathematik an der RWTH Aachen, den er bis zu seiner Emeritierung im Jahr 1998 inne hatte.

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
www.oldenbourg-wissenschaftsverlag.de

Lektorat: Margit Roth
Herstellung: Anna Grosser
Umschlagkonzeption: Kraxenberger Kommunikationshaus, München
Gedruckt auf säure- und chlorfreiem Papier
Druck: Oldenbourg Druckerei Vertriebs GmbH & Co. KG
Bindung: R. Oldenbourg Graphische Betriebe Binderei GmbH

ISBN 3-486-57849-9
ISBN 978-3-486-57849-2

Kapitel 5

Schaltungen mit Delays (Schaltwerke)

In den vorangegangenen Kapiteln haben wir Möglichkeiten kennengelernt, das Verhalten einer „Black Box" logisch durch Schaltfunktionen (Definition 1.1) zu beschreiben. Ferner haben wir erläutert, wie sich vorgegebene Schaltfunktionen, insbesondere Boolesche Funktionen (Definition 1.2) durch Schaltnetze (Definition 1.7) realisieren lassen. Dabei sind wir jeweils davon ausgegangen, dass sich bei einer solchen Schaltung nach Anlegen von Input-Signalen nach einer gewissen Zeit ein stabiler Zustand an den Ausgängen einstellt. Die Zeit, welche vom Anlegen der Inputs bis zum Ablesen des Outputs verging, wurde dabei (bis auf die Ausnahme der Schaltungshasards) vernachlässigt. Außerdem war für die Berechnung eines Outputs nur der aktuelle Input maßgebend, nicht jedoch z. B. irgendeine frühere Eingabe. Solche Schaltungen bezeichnet man allgemein als *asynchrone* Schaltnetze. In diesem Kapitel wollen wir diese erweitern zu *Schaltwerken* durch Hinzunahme von Speicherbausteinen.

5.1 Einführung

Zur Motivation betrachten wir folgendes Beispiel:

Beispiel 5.1 Gesucht ist ein Ringzähler für vierstellige Dualzahlen, genauer eine Schaltung für folgende Funktion (vgl. Aufgabe 1.13):

$$\Re : B^4 \to B^4, \text{ definiert durch}$$

$$\Re(d(i)) := d(i + 1 \bmod 16).$$

Dabei sei $d(i)$ die (vierstellige) Dualdarstellung von $i \in \{0, \ldots, 15\}$. Wie der Leser leicht verifiziert, leistet die in Abbildung 5.1 gezeigte Schaltung offenbar das Gewünschte. □

Gibt man z. B. $x_3 = 0, x_2 = 1, x_1 = 1, x_0 = 0$ in die in Abbildung 5.1 gezeigte Box $\Re$ ein, wird man nach einer gewissen Zeit an den Outputs $y_3 = 0, y_2 = 1, y_1 = 1, y_0 = 1$ ablesen können. Allerdings kann man diese Schaltung in dieser Form noch

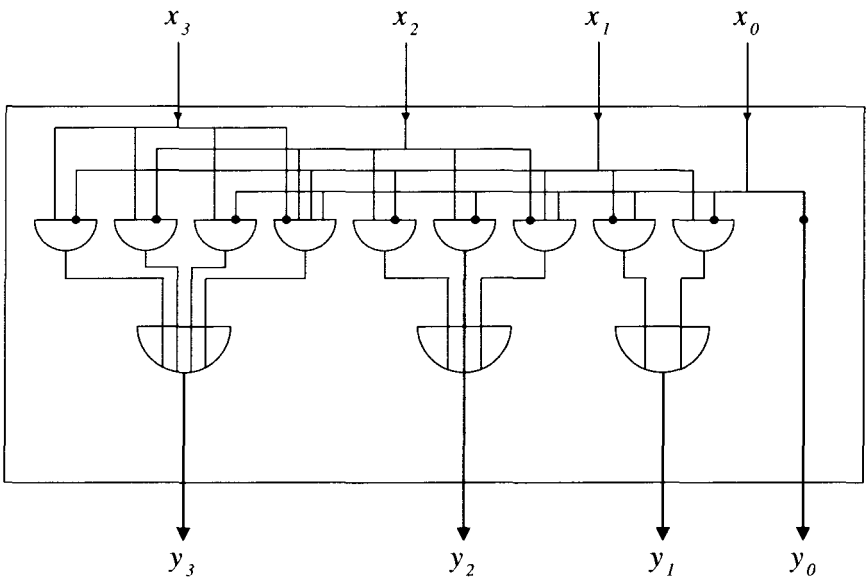

Abbildung 5.1: 4-Bit-Ringzähler $\mathcal{R}$.

nicht benutzen, denn die Aufgabe eines „Ringzählers" besteht per definitionem darin, (zyklisch) bis 15 zu zählen, sodann alle Outputs auf 0 zu setzen, bis 15 zu zählen usw. Dazu ist es offensichtlich erforderlich, die Ausgaben als nächste Eingaben aufzufassen, d. h. die Outputs mit den Inputs per Rückkopplung zu verbinden. Nach Definition 1.7 lassen Schaltnetze (als zykelfreie Graphen) derartige Konstruktionen jedoch nicht zu (vgl. die Flimmerschaltung aus Beispiel 1.4).

Abhilfe verschaffen wir uns in dieser Situation dadurch, dass wir in obiger Schaltung eine Kontrollinstanz einführen, welche Rückkopplungen durch von einer zentralen *Uhr* (Clock) ausgehende *Taktimpulse synchronisiert*. Als neues Bauteil verwenden wir dazu ein so genanntes *Delay*, für welches wir das in Abbildung 5.2 gezeigte Symbol verwenden. Aus logischer Sicht besteht ein Delay aus einem Vorspeicher V und einem Speicher S. Ein Delay ist in der Lage, ein Bit zu speichern. Einer „Schleuse" ähnlich arbeitet es in zwei Phasen, welche durch den Takt unterschieden werden:

(1) *Arbeitsphase*: Der Inhalt von S wird „nach rechts" (im Allgemeinen an ein Schaltnetz) abgegeben; er steht also als Signal y_i für eine längere Zeit zur Verfügung. Ein (eventuell anderes) Signal x_i wird in V „abgelegt". V und S sind durch eine Sperre getrennt.

(2) *Setzphase*: Eine zentrale Synchronisation (die Clock, welche Taktimpulse erzeugt) hebt die Sperre kurzzeitig auf und bewirkt dadurch die als „Setzen" bezeichnete Abgabe des Inhalts von V an S.

In der Setzphase, welche im Allgemeinen wesentlich kürzer ist als die Arbeitsphase, werden also keine Signale von außen aufgenommen oder nach außen abgegeben.

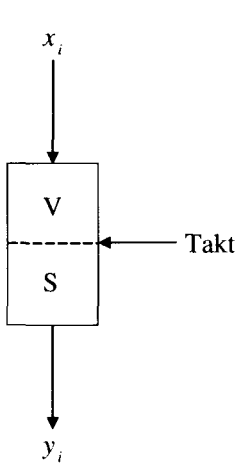

Abbildung 5.2: Delay.

Befindet sich zum Zeitpunkt i der Wert x_i im Vorspeicher, y_i im Speicher, so wird beim nächsten Takt, der den Übergang vom Zeitpunkt i zum Zeitpunkt $i+1$ markiert, der Wert x_i in den Speicher geschrieben, d. h. formal $y_{i+1} \Leftarrow x_i$.

Damit lässt sich nun bereits das oben beim Entwurf eines „funktionstüchtigen" Ringzählers aufgetretene Problem lösen:

Beispiel 5.1 (Fortsetzung): Wir vervollständigen die in Abbildung 5.1 angegebene Schaltung durch vier Delays, in denen ein gerade erzeugter Output gespeichert wird, um nach dem nächsten Takt als neuer Input zu dienen; das Ergebnis dieser Erweiterung ist in Abbildung 5.3 gezeigt. □

In der Praxis wird dabei die Clock so beschaffen sein müssen, dass ein neuer Taktimpuls erst dann erzeugt wird, wenn man sicher sein kann, dass der von $\mathfrak{R}$ erzeugte Output die gesamte Schaltung durchlaufen hat. Um Probleme dieser Art wollen wir uns hier nicht kümmern; wir verzichten von nun an sogar darauf, die Clock bzw. den Takteingang eines Delays bei der Angabe von Schaltungen einzuzeichnen. Wir nehmen jedoch bei der Verwendung von Delays ein *getaktetes*, synchronisiertes Arbeiten an; dementsprechend bezeichnen wir Schaltungen mit Delays als *(synchrone) Schaltwerke*.

Es sei bemerkt, dass ein Delay einen Fan-Out am *Ausgang* besitzen darf (vgl. Abbildung 5.4). Damit kann es seine gespeicherte Information in einem (Takt-) Schritt an mehrere Stellen abgeben. Die mit dieser Möglichkeit auftretenden physikalischen Verstärkungsprobleme werden bei der von uns gewählten logischen Sicht bewusst außer acht gelassen.

Auf das entsprechende Problem eines Fan-In am Eingang werden wir am Ende von Abschnitt 5.3 eingehen.

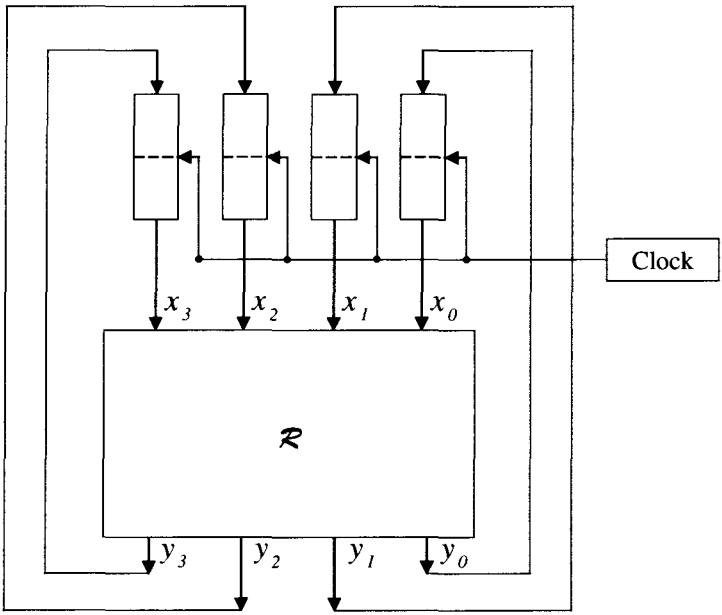

Abbildung 5.3: 4-Bit-Ringzähler mit Delays.

Kommen wir kurz zurück auf die eingangs erwähnte Flimmerschaltung durch Rückkopplung: Durch die Einführung von Delays können wir eine Schnittstelle schaffen, welche Rückkopplungen erlaubt, da ein Delay eine saubere Trennung kurz geschlossener Signale bewirkt. Die in Abbildung 1.8 gezeigte (und verworfene) Schaltung ist mit der in Abbildung 5.5 gezeigten Änderung nun zulässig.

Wie das Beispiel Ringzähler bereits andeutete, wird man zu Speicherungszwecken im Allgemeinen nicht mit nur einem Delay auskommen; meistens benötigt man Folgen von Delays, welche man auch als *Register* bezeichnet. In Beispiel 5.1 haben wir ein vierstelliges Register verwendet, dessen einzelne Komponenten paarweise voneinander unabhängig waren in dem Sinne, dass zwischen ihnen keine Verbindung bestand. Für derartige Register werden wir im Folgenden als Abkürzung für die in Abbildung 5.6 (a) gezeigte Situation auch das in Abbildung 5.6 (b) dargestellte Symbol verwenden, wobei wir uns im letzten Bild nur den Speicher-Teil der Delays gezeichnet denken.

Register sind also in der Lage, Worte der Länge n über B zu speichern, welche dann z. B. n-stellige Dualzahlen darstellen (D_i enthält dann die Ziffer der i-ten Stelle dieser Zahl). Daher bezeichnet man n auch als *Wortlänge* eines Rechners, welcher mit n-stelligen Registern arbeitet. Gängige Wortlängen sind z. B. $n = 16, 32, 64$. Zur Vereinfachung z. B. der Angabe von Registerinhalten fasst man im Allgemeinen 8 Bits zu einem so genannten *Byte* zusammen, oder – wie in Kapitel 1 im Anschluss an Beispiel 1.4 erwähnt – man fasst je drei bzw. vier Bits zu einer Oktal- bzw. Hexadezimalziffer zusammen. Prinzipiell werden alle Delays eines solchen Registers bzw. eines Schaltwerkes (bzw. eines Rechners insgesamt) gleichzeitig getaktet, wobei zwischen zwei aufeinanderfolgenden Taktimpulsen in jedem Delay Arbeits- *und* Setz-

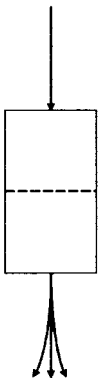

Abbildung 5.4: Fan-Out am Ausgang eines Delays.

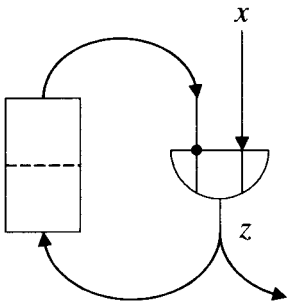

Abbildung 5.5: (Zulässige) Flimmerschaltung (mit Delay).

phase ablaufen. Diese Zeit (die zwischen zwei Taktimpulsen vergeht) nennt man die *Taktzeit* eines Rechners; in genau diesem Grundrhythmus, den wie gesagt die Rechner-Clock erzeugt, vollziehen sich synchron alle rechnerinternen Abläufe. Dabei passiert es häufig aufgrund langer Signalwege bzw. durch die Verzögerungen, welche Signale beim Durchlaufen von Schaltnetzen erfahren, dass nicht jeder Takt genutzt werden kann, so dass nicht bei jedem Takt ein Fortschritt z. B. einer aktuellen Rechnung erzielt werden kann (z. B. kann die Addition zweier Dualzahlen 12 Takte dauern). Hier finden dann Zähler (z. B. Ringzähler mod 12) Anwendung, mit deren Hilfe sich steuern lässt, welcher nächste Takt eine „Zustandsänderung" bei den entscheidenden Delays bewirkt.

Taktungen liegen heute in der Größenordnung von Bruchteilen von Nanosekunden (10^{-9} sec., entsprechende Frequenz Gigahertz, GHz). Innerhalb einer Taktzeit müssen alle Signalwege zwischen Schaltelementen durchlaufen worden sein — anders kann die Sicherheit des Betriebes nicht gewährleistet werden. Da das Licht als schnellstes aller Kommunikations-Medien in einer Nanosekunde nur 30 cm, in einer Picosekunde (10^{-12} sec) also nur 0.3 mm zurücklegt, erfordern noch wesentlich schnellere Taktungen als die heute üblichen schon aus rein physikalischen Gründen eine sehr hohe

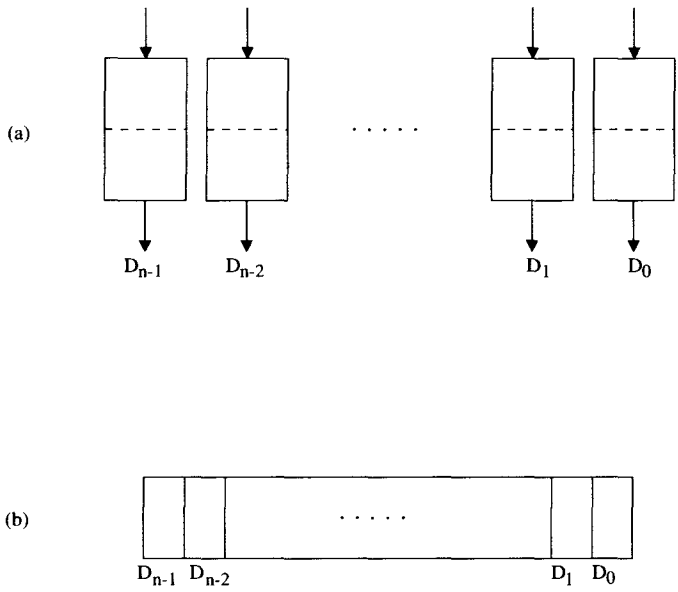

Abbildung 5.6: n-stelliges Register: (a) Prinzip; (b) Symbol.

Miniaturisierung aller Schaltungen. Aus diesen Notwendigkeiten lassen sich absolute Grenzen für künftig noch mögliche Taktzeiten herleiten, denn Schaltelemente, die kleiner als ein Molekül sind, können nach heutiger Erkenntnis sicherlich niemals realisiert werden. Zudem treten schon im Nanobereich unvermeidbare Störungen auf (Quantenrauschen), welche zusätzliche Sicherheitsrisiken bewirken (vgl. Abschnitt 7.1). Die Rechenleistung kann also nicht ins Unermessliche steigen. Schließlich setzt die Endlichkeit des menschlichen Lebens auch eine anthropologische Grenze für künftig machbare Rechenprojekte. Diese muss nach solchen vorsichtigen Abschätzungen deutlich unterhalb von 10^{30} Takten liegen, es sei denn, man spekuliert auf astronomisch viele parallel geschaltete subatomar kleine Prozessoren und auf Nutzer, deren methusalemisches Alter mit Mega-Geduld verknüpft ist.

Rechenprobleme, die mehr als 10^{30} Lösungs-Schritte erfordern, sind aber in großer Zahl vorhanden — man denke nur an die Aufgaben einer exakten Wetter-Vorhersage. Angesichts dieser Grenzen stellt sich die Frage, ob völlig andersartige Rechner-Technologien weiter gehende Perspektiven bieten. Der in der Vergangenheit häufig zitierte Molekular-Rechner wird diese Stelle nicht einnehmen können, da die Anzahl der hier zur Verfügung stehenden Rechner-Bestandteile (dies sind chemisch reagierende Moleküle) in ähnlicher Weise letztlich doch zu gering sein dürfte.

Der heute von der Spekulation favorisierte Quantenrechner, dessen physikalische Realisierung noch in weiter Ferne liegt, könnte zwar z. B. Hilfe bieten für das heute als zu schwer geltende Problem der Primfaktor-Zerlegung von Zahlen mit mehreren hundert Dezimalstellen. Damit könnte dann die auf dieser Unmöglichkeit beruhende kryptologische RSA-Datensicherungstechnik obsolet werden. Es ist aber heute noch nicht erkennbar, ob ein Quantenrechner als *universeller* Rechner eingesetzt werden kann.

Wir geben nun eine formale Beschreibungsmöglichkeit für *Schaltwerke*, d. h. für Schaltnetze mit Delays, an, welche von den „Zuständen" eines Delays wesentlich Gebrauch macht: Betrachtet man den Inhalt eines Speichers, so kann ein Delay die beiden Zustände 0 und 1 annehmen. Daher ist ein Schaltwerk S durch einen (deterministischen) *endlichen Automaten A_S mit Ausgabe* („Mealy-Automat") beschreibbar, welcher folgende Komponenten hat: $A_S = (Q, \Sigma, \Delta, q_0, F, \delta)$ mit

Q : endliche Zustandsmenge
Σ : endliches Eingabealphabet
Δ : endliches Ausgabealphabet
$q_0 \in Q$ Startzustand
$F \subseteq Q$ Menge der ausgezeichneten Zustände
$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to Q \times (\Delta \cup \{\epsilon\})$

Der Zusatz $\{\epsilon\}$ zu Σ bzw. Δ bedeutet, dass wahlweise bei einem Übergang kein Input verbraucht wird bzw. kein Output erzeugt wird. (ϵ steht dabei für das leere Wort.)

Hat S etwa n Delays, so ist das n-Tupel der Zustände dieser Delays der aktuelle Zustand von S. Eingaben bzw. Ausgaben in bzw. von S sind Worte über Σ bzw. Δ. Befindet sich A_S im Zustand q, und ist für $x \in \Sigma \cup \{\epsilon\}$

$$\delta(q, x) = (q', y),$$

so geht der Automat als nächstes in den Zustand q' über und gibt y aus. Wesentliches Merkmal eines endlichen Automaten ist seine *endliche* Gedächtnisleistung, die er durch seine nur endlich vielen Zustände besitzt: Durch den Übergang in einen bestimmten Zustand „merkt" sich der Automat eine bestimmte Situation; dieses Merken aber bedeutet im entsprechenden Schaltwerk gerade „speichern in den Delays".

Beispiel 5.1 (Fortsetzung): Sei $\Re$ der oben angegebene Ringzähler mit den vier Delays $D_3, \ldots, D_0$. Eine Beschreibung von $\Re$ als endlicher Automat lautet:

$$A_\Re = (Q_\Re, \Sigma_\Re, \Delta_\Re, q_\Re, F_\Re, \delta_\Re) \text{ mit}$$

1. $Q_\Re = B^4$; jedes $q \in Q_\Re$ hat die Form $q = (d_3, d_2, d_1, d_0)$; dabei ist d_i der Zustand von D_i.

2. $\Sigma_\Re = \Delta_\Re = \emptyset$; der Ringzähler arbeitet „autonom", d. h. er hat weder Ein- noch Ausgabe.

3. $q_\Re$ ist daher beliebig wählbar, etwa $q_\Re = (0, 0, 0, 0)$.

4. $F_\Re = \emptyset$

5. $\delta_\Re : Q_\Re \times \{\epsilon\} \to Q_\Re \times \{\epsilon\}$ ist definiert durch $\delta_\Re(q, \epsilon) := (\Re(q), \epsilon)$:
   $\Re$ wurde zu Beginn dieses Kapitels angegeben. □

Weitere endliche Automaten als Entsprechungen zu Schaltwerken werden in den Übungen betrachtet.

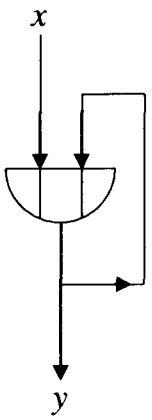

Abbildung 5.7: Rückgekoppeltes Signal.

5.2 Zur technischen Realisierung von Delays

In der Praxis wird ein Delay durch ein so genanntes *Flip-Flop* oder durch ein *Latch* realisiert; je nach Anwendung kommen dabei verschiedene Typen (z. B. SR-, D-, JK-, Master-Slave-Flip-Flop) zum Einsatz. Wir werden uns hier nur exemplarisch mit dem SR-Flip-Flop beschäftigen, da wir wie bereits bei Gattern nur die *logische*, *nicht* aber die *technologische* Sicht darstellen, und diese für alle Flip-Flop-Typen gleich ist.

Um die grundsätzliche Arbeitsweise eines Speicherelements zu verstehen, betrachten wir noch einmal die nun schon hinreichend bekannte Flimmerschaltung in einer etwas vereinfachten Form: Abbildung 5.7 zeigt eine abgewandelte Signalrückkopplung (ohne Inverter). Hierbei gilt offensichtlich Folgendes: Falls irgendwann $x = 1$ gilt, also der x-Eingang des Oder-Gatters auf 1 gesetzt wird, so wird y ebenfalls 1, und aufgrund der Rückkopplung bleibt das Ausgangssignal sodann auf 1 stehen. Mit anderen Worten: Die Schaltung *speichert* in dieser Situation eine Eins. (Dies ist offensichtlich eine idealisierte Überlegung, die vom Energieverbrauch abstrahiert.)

Damit man nun diese Eins auch wieder zurücksetzen kann, nehmen wir eine zweite Rückkopplung hinzu wie in Abbildung 5.8 gezeigt. Hier verwenden wir für die wesentlichen Signale bereits die in der Literatur üblichen Bezeichnungen S für „Set" und R für „Reset": Ist $S = 1$ und $R = 0$, so ist die Schaltung im „Zustand" 1, denn das linke Oder-Gatter hat in diesem Fall am Ausgang eine 1, welche am oberen Eingang des rechten Oder-Gatters in 0 invertiert wird. Wegen $R = 0$ produziert der Inverter am Ausgang des rechten Oder-Gatters eine 1, die nach außen abgegeben wird. Ist dagegen $S = 0$ und $R = 1$, so geht die Schaltung in den „Zustand" 0 über (man setze zunächst $R = 1$, was am Ausgang und in der Rückkopplung eine 0 verursacht; zusammen mit dem Rücksetzen auf S auf 0 ergibt sich der gewünschte Effekt).

Wir stellen die Schaltung auf Abbildung 5.8 ab sofort etwas anders dar (vgl. Abbildung 5.9) und bezeichnen die Schaltung als *Phasen gesteuertes SR-Latch*. Man erkennt jetzt deutlicher die Verwendung von Nor-Gattern; entsprechend sind Latches auch aus Nand-Gattern aufbaubar. Um die mögliche, aber nicht sinnvolle Eingabe

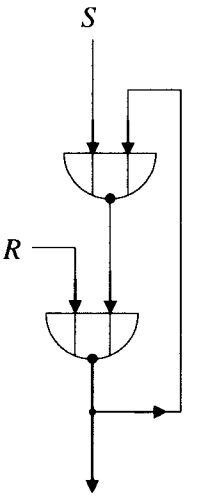

Abbildung 5.8: Zweifache Rückkopplung.

$S = R$ auszuschließen, nimmt man im Allgemeinen einen Takt hinzu, wodurch aus dem Phasen gesteuerten SR-Latch ein *Takt gesteuertes SR-Flip-Flop* wird (vgl. Abbildung 5.10; mit S wird wie vorher der von außen zugreifbare Eingang bezeichnet, der Set-Eingang des Latch heißt jetzt S'; entsprechend für die Reset-Leitung). Ein Zustandswechsel ist höchstens bei $Clk = 1$ möglich; bei $Clk = 0$ dagegen wird der Zustand gehalten, da die Und-Gatter, in die S bzw. R hineinführt, dann „blockiert" sind. Die Datenleitung D gibt das gespeicherte Bit nach außen ab.

5.3 Addierwerke

Wir wollen nun Schaltwerke entwerfen, die in dieser oder ähnlicher Form in fast jedem Rechner anzutreffen sind, nämlich *Addierwerke*, welche in der Lage sind, für festes n Dualzahlen dieser Stellenzahl zu addieren.

5.3.1 Parallel- und Serienaddierer

Das Addierproblem an sich ist bereits aus Beispiel 1.5 bekannt; in Kapitel 2 haben wir außerdem bereits ein aus Halb- bzw. Volladdierern bestehendes (asynchrones) Addiernetz für $n = 4$ kennen gelernt, welches prinzipiell auf beliebige n erweiterbar ist. Offen blieb jedoch z. B. die Frage, woher die Summanden kommen und wohin das Ergebnis geht. Dies klären wir jetzt dadurch, dass wir ein Addiernetz mit zwei Registern, einem *Akkumulator* (kurz: Akku) und einem *Puffer*, versehen, so dass wir den in Abbildung 5.11 gezeigten Grundaufbau erhalten.

Akku und Puffer enthalten zu Beginn einer Rechnung die beiden Summanden; am Ende steht das Ergebnis wieder im Akku. Da sich bei der Addition von zwei n-stelligen Dualzahlen eine $(n+1)$-stellige Zahl ergeben kann, wird man das Addiernetz darüber

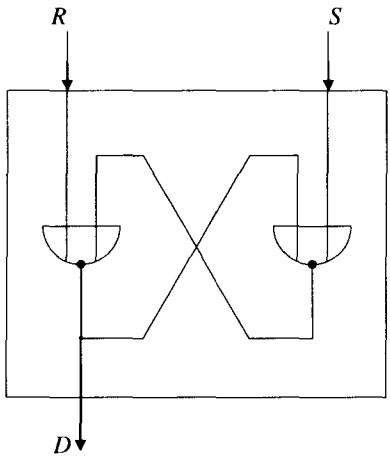

Abbildung 5.9: (Phasen gesteuertes) SR-Latch.

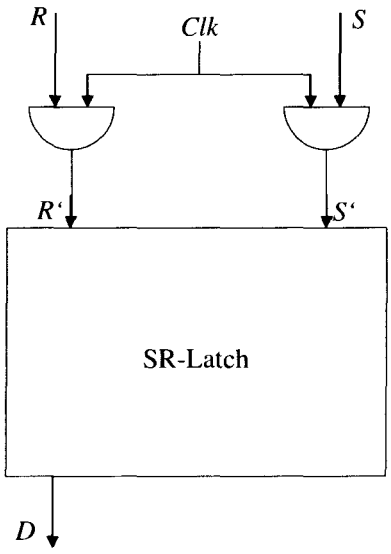

Abbildung 5.10: (Takt gesteuertes) SR-Flip-Flop.

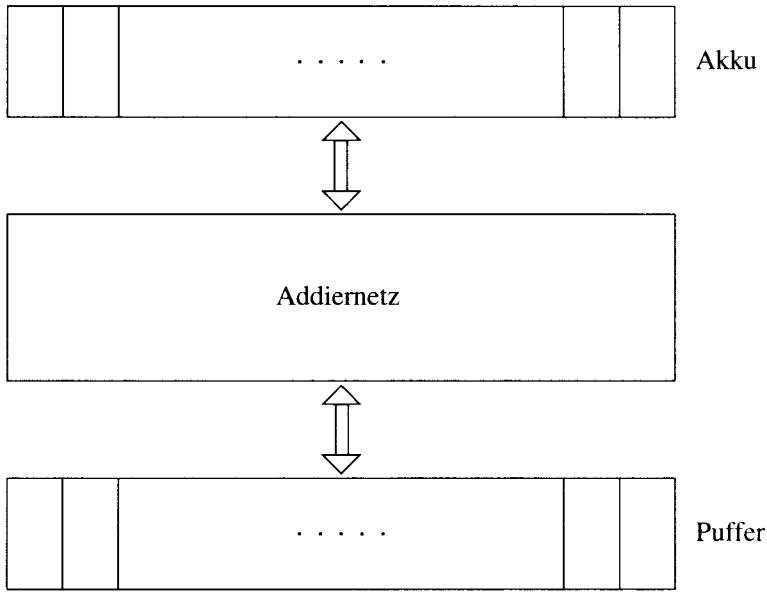

Abbildung 5.11: Organisationsplan eines Addierwerks.

hinaus noch mit einem weiteren Delay ausstatten, welches den Übertrag aufnimmt („Link").

Das bereits bekannte (vgl. Kapitel 2) asynchrone Parallel-Addiernetz, d. h. der Ripple-Carry-Adder, lässt sich damit sofort zu einem (synchronen) *Parallel-Addierwerk* machen, wie in Abbildung 5.12 dargestellt (für $n = 4$). Dabei tritt das im letzten Abschnitt bereits erwähnte Problem auf, dass am Eingang der Akku-Delays ein Fan-In vorliegt: Diese Speicherelemente können ihre Signale aus verschiedenen Quellen erhalten. Wir stellen die Lösung dieses Problems, die darin besteht, den Eingängen noch eine „Logik" vor zu schalten, zurück.

Das Parallel-Addierwerk liefert in *einem* Schritt das Ergebnis; andererseits ist aber die Laufzeit der Signale in diesem Volladdierer-Halbaddierer-Schaltnetz sehr groß, da es sich beim Halbaddierer nach den Ausführungen in Kapitel 2 um eine zweistufige, beim Volladdierer um eine vierstufige Schaltung handelt. Für feste Stellenzahl n der Inputs muss also jedes Signal $2+4(n-1) = 4n-2$ Stufen durchlaufen. In Abschnitt 2.4 errechneten wir daraus eine (hypothetische) Schaltzeit von 240 psec für das vierstellige Addiernetz. Das bedeutet, dass erst nach dieser Zeit der Taktimpuls erfolgen darf, welcher das Addierwerk für die nächste Additionsaufgabe initialisiert. Des Weiteren sei bemerkt, dass dieses Addierwerk die durch Speicherbausteine gegebene Möglichkeit des *schrittweisen Arbeitens* nicht ausnutzt.

Dies ist anders beim *seriellen Addierwerk*, welches wir als nächstes beschreiben: Akku und Puffer sind dabei als *Schieberegister* organisiert; bei jedem Taktimpuls wird die gespeicherte Information um eine Stelle nach rechts (oder links) geschoben. Im Falle des Rechts-Verschiebens, wie wir es hier verwenden, wird dabei jeweils das linkeste Delay des Registers frei, der Inhalt des rechtesten „fällt heraus". Wir geben das

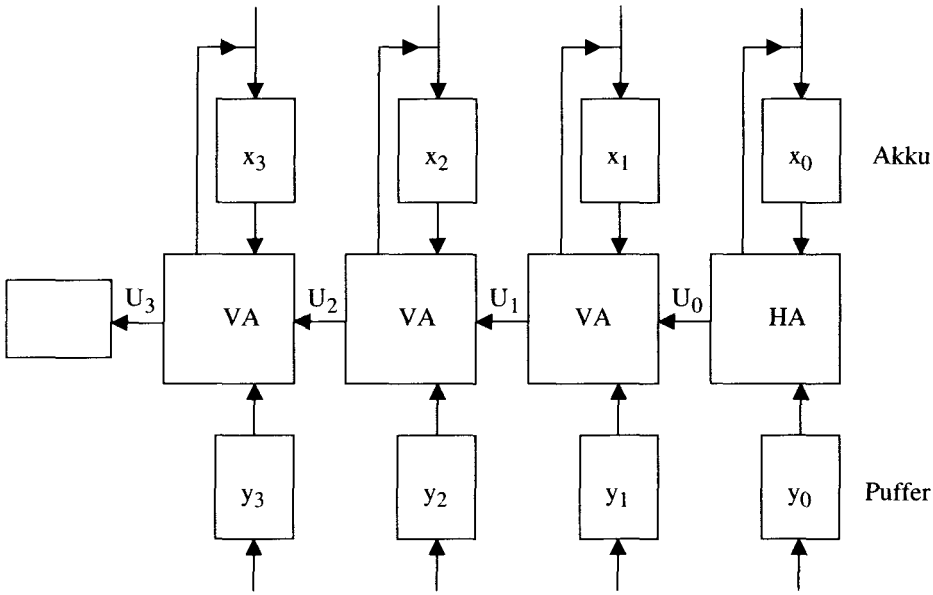

Abbildung 5.12: 4-Bit-Parallel-Addierwerk (Ripple-Carry-Adder mit Delays).

Schaltwerk in Abbildung 5.13 wieder für $n = 4$ an. Dieses Bild gibt den „Startzustand" des Serien-Addierers an, der ebenfalls durch einen endlichen Automaten beschrieben werden kann (vgl. Aufgabe 5.10).

Offensichtlich ist das Serien-Addierwerk leicht auf höhere Stellenzahlen erweiterbar: Man benötigt lediglich größere Register, d. h. mehr Delays, jedoch keine zusätzliche Logik wie etwa beim Parallel-Addierwerk. Im Vergleich zu diesem liefert das serielle Addierwerk das Ergebnis erst nach n Schritten, wodurch es im Wesentlichen obsolet geworden zu sein scheint; dafür ist jedoch jetzt die Schaltzeit kurz, da man lediglich eine vierstufige Schaltung benötigt.

5.3.2 Das von Neumann-Addierwerk

Wir kombinieren nun das Charakteristikum des Paralleladdierers, für jede Stelle einen separaten Addierer zu verwenden, mit dem des Serienaddierers, schrittweise zu arbeiten und Überträge zwischenzuspeichern, zu einem Addierwerk, dessen Schrittzahl von den Summanden abhängt. Dieses sogenannte *Von-Neumann-Addierwerk* hat den in Abbildung 5.14 gezeigten logischen Aufbau (wieder dargestellt für $n = 4$). Wie die bisher beschriebenen Addierwerke besteht auch dieses aus den Registern Akku und Puffer sowie einem Übertrags-Delay; man kommt jedoch mit Halbaddierern aus. Es handelt sich hierbei offensichtlich um einen getakteten Carry-Save-Adder, der aus Halbaddierern aufgebaut ist. Daneben gibt es noch ein Status-Delay S, welches angibt, ob eine aktuelle Rechnung beendet ist oder nicht. Im Einzelnen arbeitet der Von-Neumann-Addierer wie folgt:

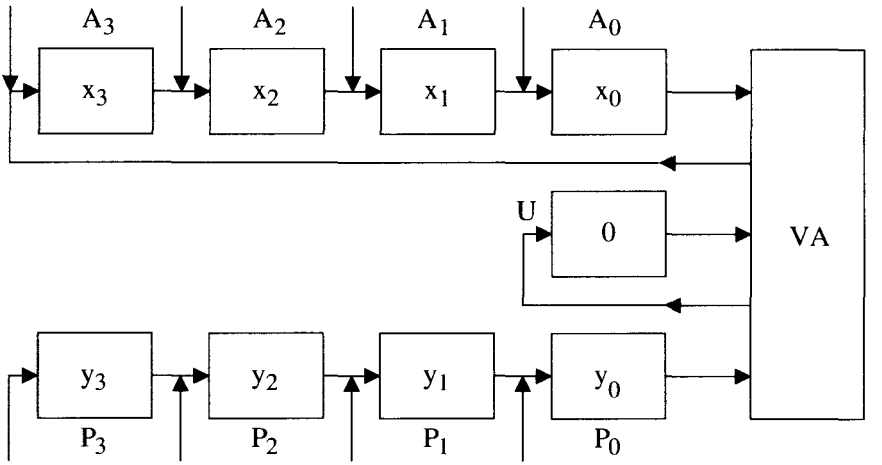

Abbildung 5.13: 4-Bit-Serien-Addierer.

Im ersten Schritt werden alle 4 Stellen parallel addiert, für jede Stelle i wird das Resultat $A_i \leftrightarrow P_i$ nach A_i zurückgeschrieben und der Übertrag $A_i \cdot P_i$ an P_{i+1} weitergegeben bzw. für $i = 3$ ins Übertrags-Delay gebracht. Für P_0 ist damit kein neuer Wert definiert; dieser Teil des Puffers erhält den Wert 0, welchen wir durch $P_0 \cdot \overline{P}_0$ erzeugen. Nach diesem Schritt enthält der Akku ein „vorläufiges" Additions-Ergebnis und der Puffer die noch zu addierenden Überträge. Sind solche noch vorhanden, liefert also das Oder-Gatter vor dem Status-Delay den Wert 1, so wird im nächsten Schritt das gesamte Vorgehen iteriert, wobei ein „endgültiger" Übertrag, welcher in U bereits erschienen ist, durch die dort konstruierte Rückkopplung jeweils erhalten bleibt. Werden in einem Schritt keine Überträge mehr erzeugt und erhält S somit den Wert 0, so ist eine aktuelle Rechnung beendet, und das Ergebnis steht im Akku (unter Hinzunahme des Übertrags-Delays).

Im Gegensatz zu den beiden vorher angegebenen Addierwerken haben wir vorläufig auf das Einzeichnen von Leitungen, über welche die Eingabe der Summanden bzw. die Ausgabe des Ergebnisses erfolgt, verzichtet. Außerdem dürfte dem Leser unmittelbar einleuchten, wie die in Abbildung 4.10 angegebene Schaltung auf höhere Stellenzahlen als $n = 4$ zu erweitern ist.

Wir geben nun die Arbeitsweise dieses Addierers beispielhaft in Tabellenform an (vgl. Tabelle 5.1): nacheinander sollen die Aufgaben 13+11, 10+12, 15+15, 9+10, 0+0 gelöst werden. Die erste Zeile dieser Tabelle gibt den „Startzustand" des Addierwerks wieder: das Status-Delay S hat den Wert 0 und zeigt damit an, dass eine neue Rechnung starten kann. Nun werden die ersten beiden Summanden in Akku bzw. Puffer geladen und das Status-Bit auf 1 gesetzt (Zeile 2). Sodann läuft die erste Rechnung wie oben erläutert ab: wird das Status-Delay erneut 0, so ist diese Rechnung beendet. Das Ergebnis steht dann im Akku inklusive Übertrags-Delay U und kann von dort ausgelesen werden (Zeile 6). Sodann wird ein neuer Input geladen (Zeile 7), und es beginnt eine neue Rechnung. Die nachfolgenden Abläufe vollziehen sich dann völlig analog.

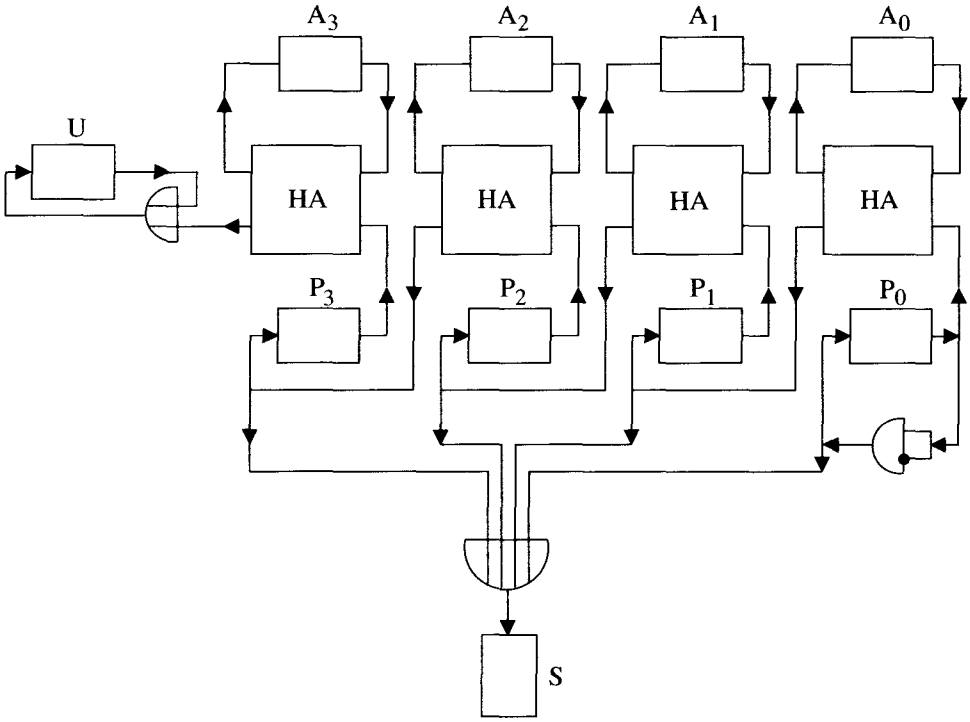

Abbildung 5.14: 4-Bit-Von-Neumann-Addierwerk.

An diesen Beispielen ist ein interessanter Sachverhalt abzulesen: Im $(i+1)$-ten Schritt jeder Rechnung sind (mindestens) die rechten i Stellen des Puffers gleich 0. Dies bedeutet, dass der (4-stellige) Von-Neumann-Addierer nach spätestens 5 Schritten jede Rechnung beendet. Für beliebiges festes n gilt das Gleiche: Jede Rechnung ist nach spätestens $n + 1$ Schritten beendet. Wie man aber an der Tabelle auch bereits erkennen kann, kann bei „günstigen" Summanden schon viel früher mit einem Ergebnis gerechnet werden (die Additionen 10+12 und 9+10 z. B. waren bereits nach 2 Schritten beendet). Tatsächlich kann man zeigen:

Satz 5.1 Das n-Bit-Von-Neumann-Addierwerk addiert zwei Summanden in durchschnittlich ld $n + 1$ Schritten. (ld $n = \log_2 n$ ist dabei der Duallogarithmus, d. h. der Logarithmus zur Basis 2.)

Wir verzichten auf einen exakten Beweis dieses Satzes, da dazu eine Reihe von wahrscheinlichkeitstheoretischen Begriffen benötigt wird; stattdessen skizzieren wir nur die Beweisidee:

Wir betrachten zwei „zufällig" ausgewählte Summanden. Man darf dann erwarten, dass jeder dieser Summanden $\frac{n}{2}$ Einsen und $\frac{n}{2}$ Nullen hat, und dass die Nullen und Einsen unabhängig voneinander verteilt sind, so dass jede der Kombinationen 00, 01, 10, 11 von Puffer- und Akku-Bits gleich oft, nämlich $\frac{n}{4}$-mal auftritt. Man betrachte nun noch einmal die Funktionstafel des Halbaddierers (vgl. Tabelle 5.1): Man erkennt,

Tabelle 5.1: Beispiel zur Arbeitsweise des Von-Neumann-Addierwerks.

Zeile	S	U	Puffer-Inhalt dual $P_3P_2P_1P_0$	Puffer-Inhalt dezimal	Akku-Inhalt dual $A_3A_2A_1A_0$	Akku-Inhalt dezimal	Schritt
1	0	0	0000	0	0000	0	
2	1	0	1101	13	1011	11	1
3	1	1	0010	2	0110	22	2
4	1	1	0100	4	0100	20	3
5	1	1	1000	8	0000	16	4
6	0	1	0000	0	1000	24	5
7	1	0	1010	10	1100	12	1
8	0	1	0000	0	0110	22	2
9	1	0	1111	15	1111	15	1
10	1	1	1110	14	0000	16	2
11	0	1	0000	0	1110	30	3
12	1	0	1001	9	1010	10	1
13	0	1	0000	0	0011	19	2
14	1	0	0000	0	0000	0	1
15	0	0	0000	0	0000	0	2

Tabelle 5.2: Funktionstafel des Halbaddierers.

A_i	P_i	U_i	R_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

dass man nach dem ersten, also im zweiten Schritt im Akku wieder $\frac{n}{2}$ Einsen (und $\frac{n}{2}$ Nullen), im Puffer aber nur noch $\frac{n}{4}$ Einsen zu erwarten hat. Nach der obigen „Zufälligkeits- und Unabhängigkeitsannahme" ist weiter zu erwarten, dass an der Hälfte dieser Stellen, also an $\frac{n}{8}$ Stellen, auch im Akku eine 1 steht, so dass beim dritten Schritt im Puffer nur noch an $\frac{n}{8}$ Stellen eine Eins zu erwarten ist. Allgemein kann man also im i-ten Schritt im Puffer mit $\frac{n}{2^i}$ Einsen rechnen, während im Akku unverändert mit $\frac{n}{2}$ Einsen gerechnet werden kann. Stehen im Puffer keine Einsen mehr, ist eine Rechnung beendet, und das ist spätestens dann zu erwarten, wenn ein Schritt i_0 erreicht ist, für den $n < 2^{i_0}$, d. h. $i_0 > \operatorname{ld} n$ gilt. □

Das Von-Neumann-Addierwerk ist also *im Mittel* nach wenigen Schritten (z. B. 11 Schritte für $n = 2^{10} = 1024$) mit einer aktuellen Rechnung fertig; außerdem ist sein Schaltnetz (die parallelen Halbaddierer) nur zweistufig, was eine hohe Taktfrequenz ermöglicht. Damit ist es gegenüber den vorher vorgestellten Addierwerken bzgl. der Rechenzeit unter Umständen zu bevorzugen.

Tabelle 5.3: Zur Lösung des Fan-In-Problems bei Delay-Eingängen I.

S	I	R	f	Output
0	0	0	0	I
0	0	1	0	I
0	1	0	1	I
0	1	1	1	I
1	0	0	0	R
1	0	1	1	R
1	1	0	0	R
1	1	1	1	R

Wir kommen nun auf das Problem des Fan-In bei Delay-Eingängen zurück: Beim Von-Neumann-Addierer kann jedes Akku-Delay ein Input- Signal oder das Resultats-Signal eines Halbaddierers, jedes Puffer-Delay ebenfalls ein Input-Signal oder das Übertrags-Signal eines Halbaddierers (bzw. eine Null im Falle des rechtesten Delays) erhalten. Welches Signal zu speichern ist, hängt offensichtlich davon ab, ob sich der Addierer in einem Rechenschritt ($S = 1$) oder einem I/O-Schritt ($S = 0$) befindet. Daher lässt sich leicht eine Logik angeben, welche dafür sorgt, dass jedes A_i bzw. P_i nur mit einem „richtigen" Signal beschickt wird. Wir beschreiben diese durch eine Boolesche Funktion $f : B^3 \to B$, welche für den Akku definiert ist durch

$$f(S, I, R) := \begin{cases} I & \text{falls } S = 0 \\ R & \text{falls } S = 1 \end{cases}$$

I steht dabei für ein Input-Signal, R für ein Resultatssignal. Für den Puffer ersetze man R durch U. Für f erhält man die in Tabelle 5.3 angegebene Funktionstafel.

Das Fan-In-Problem für die Akku-Delays wird also gelöst, indem man im (in Abbildung 5.14 gezeigten) Schaltbild des Von-Neumann-Addierers den in Abbildung 5.15 (a) gezeigten Teil durch den in Abbildung 5.15 (b) gezeigten Teil ersetzt (analog für die Puffer-Delays).

Abschließend sei erwähnt, dass man beim Von-Neumann-Addierwerk wie bei asynchronen Addiernetzen eine Beschleunigung dadurch erzielen kann, dass man Bitgruppen zusammenfasst; durch erhöhten Hardwareaufwand ist ungefähr eine Halbierung der Schrittzahl erreichbar. Eine Carry-Bypass-Schaltung zur schnellen Bestimmung des Übertrags ist ebenfalls analog zum asynchronen Fall angebbar (vgl. Abschnitt 2.5).

Es muss zugegeben werden, dass das von Neumannsche Addierwerk heute eher akademischen Charakter hat, da in modernen Rechnern ausschließlich asynchrone Addiernetze verwendet werden. Die relativ langen Taktzeiten können offenbar gegen die Geschwindigkeit asynchroner Netzschaltungen nicht konkurrieren. Außerdem benötigt man im modernen Pipelining *sicher* eingehaltene Additionszeiten und kann spekulative Verkürzungen nicht ohne weiteres verwerten. Sollten allerdings künftig noch weit größere Wortlängen als die zurzeit üblichen (32 bzw. 64 Bit) Verwendung finden, so könnte die von Neumannsche Idee erneut relevant werden.

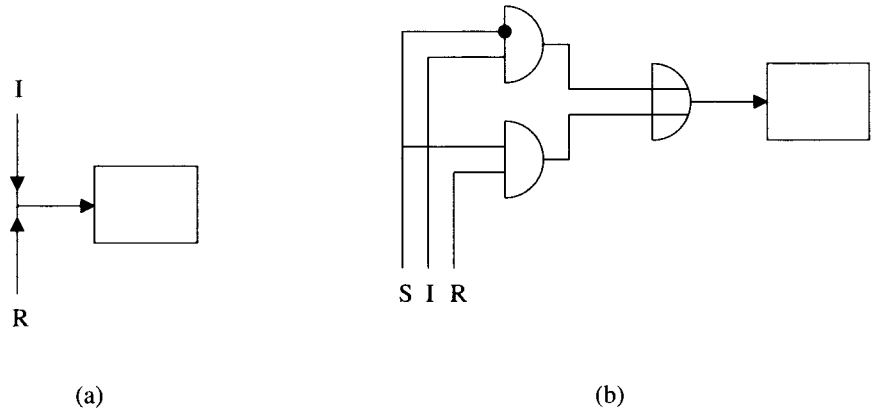

Abbildung 5.15: Zur Lösung des Fan-In-Problems bei Delay-Eingängen II.

Mit einem Addierwerk kann man auch die anderen Grundrechenarten durchführen: Die Subtraktion kann auf die Addition des Einer-Komplements zurückgeführt werden, welches man durch stellenweises Komplementieren erhält (vgl. Aufgabe 2.2). Das den Subtrahenden aufnehmende Register muss also in der Lage sein, diese einfache Operation durchzuführen. Durch ein zusätzliches Signal T lässt sich dabei steuern, ob das Werk addieren ($T = 0$) oder subtrahieren ($T = 1$) soll, so dass ein kombiniertes Addier/Subtrahierwerk etwa den in Abbildung 5.16 gezeigten Aufbau hat. Ebenso lassen sich Multiplikation und Division auf die Addition zurückführen, wenn man zusätzlich noch Links- bzw. Rechts-Verschiebungen von Register-Inhalten realisiert.

5.4 Lineare Schaltkreise und Anwendungen

In diesem Abschnitt wird eine wichtige Klasse von Schaltungen mit Delays behandelt, welche — ähnlich den Addierwerken — ein breites Anwendungsspektrum besitzen. Aufgrund ihrer speziellen Schaltungs-Struktur sind sie für einen getakteten Dauerbetrieb mit beständig neuem Input geeignet.

Wir setzen im Folgenden voraus, dass sich die zu verarbeitende Information allgemein aus Elementen eines endlichen Körpers zusammensetzt. Eine Sonderstellung unter den endlichen Körpern nehmen die Primkörper der Charakteristik p (p Primzahl) ein; hierbei handelt es sich um Restklassenkörper der Charakteristik p: Man kann in ihnen rechnen (addieren, subtrahieren, multiplizieren) wie im Bereich der ganzen Zahlen, muss dabei nach jeder Rechnung aber eine Reduktion modulo p vornehmen, um das Ergebnis in den Bereich der Zahlen x mit $0 \leq x < p$ zu verlegen. Kennzeichnet man die reduzierten Ergebnisse durch Unterstreichen, so gilt z. B. im Restklassenkörper modulo $p = 8191$

$$\underline{3517} + \underline{6000} = \underline{1326}$$
$$\underline{246} - \underline{4118} = \underline{4319}$$
$$\underline{7167} + \underline{1024} = \underline{0}$$
$$\underline{433} \cdot \underline{5219} = \underline{7302}$$