Betreuer: Albers

Fortgeschrittenen–Praktikum am
Institut für Kernphysik

Digitale Elektronik[1]

Oliver Flimm
Oberstraße 74
51149 Köln

Uwe Münch
Schmittgasse 92
51143 Köln

Wintersemester 93/94[2]

e-mail: flimm@ph-cip.uni-koeln.de
muench@ph-cip.uni-koeln.de

[1] Versuch 8c
[2] Durchführung des Versuchs: 7.2.94

Vorwort

Wir wollen hier zunächst ein paar Worte zum Aufbau unseres Versuchsprotokolls verlieren. Denn auf den nächsten Seiten ziert ein „© by Oliver Flimm und Uwe Münch“ die letzte Zeile. Warum dies? Sind diese beiden Praktikanten so eingebildet oder übergeschnappt? Nein, das sind wir nicht, und wir wollen jetzt also begründen, warum wir diese Zeile am Ende jeder Seite für nötig halten.

Viele Praktikanten bereiten ihre Versuche im CIPLAB vor und werten sie auch dort aus. Nur einige davon schützen ihre Arbeit vor Zugriff durch andere. Das bedeutet, daß man sich vielfach die `.dvi`-Dateien anschauen kann, aber auch daß man direkt auf `.tex`-Sourcen zugreifen könnte. Dieser Zustand ist ja allgemein bekannt. Wie ist nun unsere Einstellung dazu?

Wir denken, daß es nicht Sinn der Sache (d. h. des FP's) sein kann, sich einfach die Sourcen zu kopieren. Zumeist sind diese Vorlagen auch lückenhaft oder enthalten Fehler. Kurz gesagt, aus diesen Gründen benutzten wir keine solchen Textvorlagen. (Über unser Verfahren, was Bilder angeht, werden wir uns gleich äußern.) Wie sieht das nun mit `.dvi`-Files aus? Wir denken, daß ein Aspekt des Fortgeschrittenenpraktikums ist, daß wir lernen, wie man durchgeführte Versuche protokolliert, beschreibt und (später mal) veröffentlicht. Zu diesem Lernprozeß gehört es sicherlich auch, anhand von Beispielen zu sehen, wie so etwas geschieht. (Allerdings bezweifeln wir, daß dazu die Vorlagen immer so geeignet sind.) Zumindest halten wir es für keinen Frevel, wenn man sich anhand anderer Ausarbeitungen eine Idee holt, wie Versuchsprotokolle anzufertigen sind. Daher beabsichtigen auch wir, unsere `.dvi`-Dateien im CIPLAB allgemein lesbar zur Verfügung zu stellen. Dies wird *nicht* mit unseren `.tex`-Sourcen geschehen! Diese halten wir natürlich geschützt. Wir wollen nämlich nicht, daß unsere Mühen *einfach so* von anderen, uns bekannten oder unbekannten Studenten weiter genutzt werden. Wie oben dargelegt, haben wir aber nichts dagegen, wenn unsere Ausarbeitung als Beispiel und Denkanstoß dient. Für Nachfragen, etc. haben wir unsere vollständigen Normal- und unsere e-mail-Adressen angegeben. Damit niemand auf die Idee kommt, sich das Leben doch zu einfach zu machen und die `.dvi`-Datei einfach ausdruckt und in der Hoffnung, daß sie keiner wiedererkennt, abgibt, haben wir die Copyright-Zeile eingefügt. Diese dürfte so etwas effektiv verhindern. Außerdem (wovon wir weniger überzeugt sind) dient sie vielleicht als psychologische Sperre, zu wörtlich von uns abzuschreiben.

Nachdem wir uns so ausführlich über diese Vorgehensweise ausgelassen haben, wollen wir dann auch noch ein paar Worte über Bildvorlagen verlieren. Bei manchen Versuchen benutzten wir aus Büchern eingescannte Graphiken. Wir haben dann jeweils die Herkunft vollständig (also am Bild und im Literaturverzeichnis) dokumentiert. Alle sonstigen Bilder sind von uns mittels `xfig`, `gnuplot` oder ähnlichen Programmen selbst erstellt worden.

Ein paar abschließende Worte zur benutzten TEX-Umgebung: Wir erstellten unsere Ausarbeitungen bereits mit der Testversion des neuen Formats LATEX 2_ε. Die benutzten Classes und Packages sind allerdings noch von Hand von uns angepaßt worden. Aber nun in medias res....

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Vorbereitung

1.1 Einleitung

In Versuch [FP-8a] haben wir uns mit den Grundelementen der *analogen* Elektronik beschäftigt. Insbesondere deren Zeit- und Frequenzverhalten. In diesem Versuch betrachten wir grundlegende Probleme und Bauteile der *digitalen* Elektronik. Hierbei liegt das Hauptaugenmerk auf der Anordnung und Verknüpfung von logischen Grundelementen, nicht jedoch auf dem tatsächlichen elektronischen Aufbau dieser Grundelemente. Diese grundlegenden Bauteile werden in der Sektion *Boole'sche Algebra* vorgestellt. Danach beschäftigen wir uns mit dem Aufbau wichtiger Grundschaltungen, mit denen digitale Signale gespeichert und verknüpft werden können.

1.2 Boole'sche Algebra

Im vorigen Jahrhundert hat G. Boole eine symbolische, auf zweiwertige Variable aufgebaute Logik formuliert. Mit Hilfe der von ihm geschaffenen sog. Boole'schen Algebra können logische Zusammenhänge rechnerisch erfaßt werden. Als zweiwertige Variable geht man häufig auf die Paare *wahr/falsch*, *0/1*, *high/low*, oder, was dem Aufbau elektrischer Schaltungen eher entspricht, auf die Paare *an/aus*, *Spannung/Keine Spannung* über. Hierbei identifiziert man meist das Paar *Spannung/Keine Spannung* mit niedrigen und hohen Spannungbereichen. In der Praxis ist noch zu beachten, daß zwischen diesen zwei Spannungsbereichen ein dritter Bereich existiert, bei dem die Behandlung der Zustände undefiniert ist. Dies ist natürlich unerwünscht und daher ist dieser Bereich bei elektronischen Schaltungen zu vermeiden. Wir ignorieren natürlich solche ungünstigen Fälle im folgenden.

Operationen

Man kann nun logische/digitale Zustände miteinander verknüpfen. Hierbei stehen einige Grundoperationen zur Verfügung. Wir geben sie im folgenden an, wobei jeweils Synonyme für die entsprechenden Operationen in einer Zeile zusammengefaßt sind:

1. a und b; a and b; $a \wedge b$; $a \cdot b$; $\min(a, b)$
2. a oder b; a or b; $a \vee b$; $a + b$; $\max(a, b)$; nichtausschließenden Oder
3. nicht a; not a; $\neg a$; $\overline{a}$
4. wenn a, so b; if a then b; $a \rightarrow b$
5. a äquivalent b; a iff b; $a \iff b$

6. a aut b; a xor b; $a \nleftrightarrow b$; ausschließendes Oder
7. Peircescher Pfeil; a nor b; $a \downarrow b$
8. Shefferscher Strich; a nand b; $a \uparrow b$.

Die elektronischen Pendants dieser Operationen nennt man *Gatter*.

Wie der Name schon sagt, beschreibt die Boolesche Algebra eine Algebra. Es gelten folgende Gesetze:

Kommutativgesetz	$x \wedge y = y \wedge x$
	$x \vee y = y \vee x$
Assoziativgesetz	$(x \wedge y) \wedge z = x \wedge (y \wedge z)$
	$(x \vee y) \vee z = x \vee (y \vee z)$
Verschmelzungsgesetze	$(x \wedge y) \vee x = x$
	$(x \vee y) \wedge x = x$
Distributivgesetze	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Komplementgesetze	$x \wedge (y \vee \overline{y}) = x$
	$x \vee (y \wedge \overline{y}) = x$
de Morgansche Regeln	$\overline{x \wedge y} = \overline{x} \vee \overline{y}$
	$\overline{x \vee y} = \overline{x} \wedge \overline{y}$

Im allgemeinen braucht man auf einer Boolschen Algebra nicht alle Operationen. Viele können ineinander umgewandelt werden. So gilt z.B.

$$a \wedge b = (a \wedge b) \vee (a \wedge b) = \neg((\neg(a \wedge b)) \wedge (\neg(a \wedge b))) = (a \uparrow b) \uparrow (a \uparrow b) \; .$$

Man faßt vielfach bestimmte Operationen zusammen und sagt, sie sind *funktional vollständig*. Das heißt, aus ihnen können alle anderen Operationen hergeleitet werden.

Funktional vollständig sind:

- $\{\wedge, \vee, \neg\}$. So fängt jeder an. Aber man kann verkürzen zu:
- $\{\wedge, \neg\}$ oder
- $\{\vee, \neg\}$. Noch weiter verkürzt kann man sogar nur von
- $\{\uparrow\}$ oder
- $\{\downarrow\}$ ausgehen.

Charakterisiert werden diese Grundoperationen durch *Wahrheitstabellen*. Beispiele für Wahrheitstabellen sind in Abbildung 1.1 aufgeführt.

Weiterhin besitzt jede Operation ein *Schaltsymbol*. Hierbei existieren jedoch verschiedene Normen, und demnach auch verschiedene Schaltsymbole (IEEE, DIN) für ein und dieselbe Operation. Typische Schaltsymbole (nach neuer DIN-Norm) findet man in Abbildung 1.2.

AND

A \ B	0	1
0	0	0
1	0	1

OR

A \ B	0	1
0	0	1
1	1	1

NOT

A	$\overline{A}$
0	1
1	0

NAND

A \ B	0	1
0	1	1
1	1	0

NOR

A \ B	0	1
0	1	0
1	0	0

XOR

A \ B	0	1
0	0	1
1	1	0

Abbildung 1.1: Wahrheitstabelle der Grundoperationen

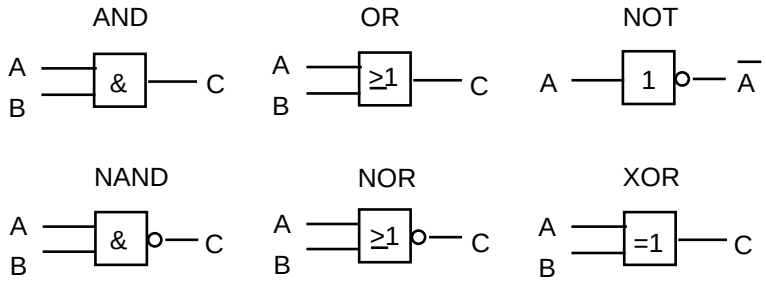

Abbildung 1.2: Schaltsymbole

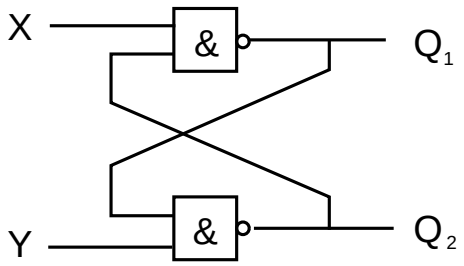

Abbildung 1.3: Latch

1.3 Flip-Flops

Die Speicherung binärer Informationen erfolgt in den sog. bistabilen Elementen, das sind Schaltungen, mit zwei stabilen Zuständen, kurz *Flip-Flops* genannt. Der Zustand eines Flip-Flops kann erst dann geändert werden, wenn es von außen angeregt und in den anderen Zustand überführt wird. Dies ist dann auch der große Unterschied zu einem Gatter, das nur in der Lage ist, momentan eingehende Signale zu verarbeiten.

Flip-Flops können durch an den Eingang rückkoppelnde Ausgangssignale mit Hilfe von Gattern realisiert werden. Es gibt viele verschiedene Arten von Flip-Flops, von denen einige hier besprochen werden. Jedes wird anhand der angegebenen Wahrheitstafel vollständig charakterisiert.

Latch

Wahrheitstabelle

X	Y	Q_1	Q_2
0	0	?	?
1	0	0	1
0	1	1	0
1	1	$Q_{1,v}$	$Q_{2,v}$

Das *Latch* stellt die einfachste Art eines Flip-Flops dar. Zur Beschreibung des logischen Verhaltens dieser Schaltung dient die nebenstehend gezeigte Wahrheitstafel. Die Ausgangsvariablen Q_1 und Q_2 geben an, welche Stellung das Flip-Flop innehat, wenn die einstellenden Informationen nicht mehr an den Eingängen liegen. Wenn an mindestens einem NAND-Gatter (vgl. Abb. 1.3) ein 0-Signal anliegt, dann nimmt per definitionem der zugehörige Ausgang 1-Potential an. Exemplarisch wird die generelle Vorgehensweise zur Bestimmung der Wahrheitstabelle kurz erläutert. Liegt am Eingang $X = 1$ und $Y = 0$ an, so gilt einerseits $Q_1 = \overline{1 \wedge Q_2} = 0 \vee \overline{Q_2} = \overline{Q_2}$, andererseits $Q_2 = \overline{0 \wedge Q_1} = \overline{0} \vee \overline{Q_1} = 1$. Damit ergibt sich $Q_1 = 0$ und $Q_2 = 1$. Wird nun auch noch $Y = 1$ geschaltet, so ändert sich der zugehörige NAND-Gatter-Ausgang nicht, denn Q_1 ist ja Null. Die Rückkopplung an das andere NAND-Gatter ist somit auch unverändert, insbesondere auch dessen Ausgang. Wir sehen, der vorige Zustand der Ausgänge bleibt erhalten, wir haben also tatsächlich den Zustand gespeichert.

Das Anlegen von $X = 1$ und $Y = 1$ stellt eine Art Grundzustand dar, in dem die Speicherung der Ausgänge sichergestellt ist. Man schaltet deshalb immer wieder in diesen Zustand zurück.

Mit Ausnahme des Falles $X = 0$ und $Y = 0$ erkennen wir, daß Q_1 und Q_2 jeweils komplementär sind, also $Q_1 = \overline{Q_2}$ gilt. Vielfach werden die Ausgänge daher nur mit Q und $\overline{Q}$ bezeichnet.

Während des Anlegens von $X = 0$ und $Y = 0$ liegen sowohl Q_1 wie auch Q_2 auf 1. Das ist ein unerwünschter Effekt, denn wenn man nun beide Eingänge auf 1 legt, bekommt man einen undefinierten Zustand, da nicht beide Eingänge wirklich gleichzeitig auf 1 gelegt werden können. Ein Eingang wird

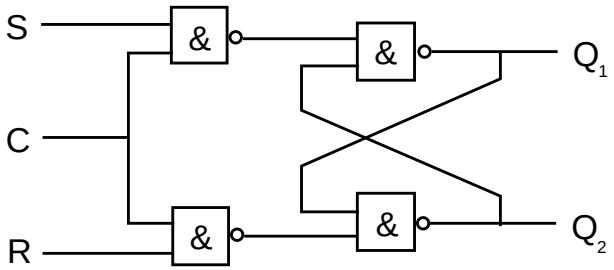

Abbildung 1.4: Schaltplan eines getakteten RS-Flip-Flops

immer vorher auf 1 liegen, während der andere noch auf 0 liegt. Die aus dieser Konfiguration resultierenden Ausgangswerte werden dann im FlipFlop gespeichert, wenn schließlich auch der zweite Eingang auf 1 geht.

Man bewältigt dieses Problem beim Latch, indem man es nicht weiter betrachtet. Weiterhin versucht man dieses Problem zu umgehen, indem man es zunächst verlagert. Wir kommen dann zum *getakteten RS-Flip-Flop*.

Getaktetes RS-Flip-Flop

Wahrheitstabelle beim 1-Takt

S	R	Q_1	Q_2
0	0	$Q_{1,v}$	$Q_{2,v}$
1	0	1	0
0	1	0	1
1	1	?	?

Anhand der Wahrheitstabelle erkennt man, daß wir das Problem des Latches tatsächlich nur verlagert haben. Bestand dort das Problem beim Anlegen von $X = 0$ und $Y = 0$, so bekommen wir nun Schwierigkeiten beim Anlegen von $S = 1$ und $R = 1$.

Im Vergleich zum Latch haben wir diesem zwei NAND's vorgeschaltet, die quasi als Inverter der Eingangssignale wirken (vgl. Abb. 1.4). Wir sehen nun, daß erst dann die *Set*- bzw. *Reset*-1-Signale als 0-Signale an das Latch weitergegeben werden, wenn der vorgeschaltete Inverter am Taktsignal C ein 1-Signal anliegen hat. Liegt am Takt eine 0 an, so liegen an den Eingängen des Latches 1-Signale an, der vorige Zustand bleibt also gespeichert. Wir sehen hier noch deutlicher, daß der 1-1-Zustand am Latch der speichernde Grundzustand ist. Bis auf die vorgeschalteten NAND's kann zur Findung der Wahrheitstabelle wie beim Latch argumentiert werden.

Die Abkürzungen S und R stehen im übrigen für *Set* und *Reset*, wobei diese Benennung der Eingänge sich darauf bezieht, ob der Ausgang Q_1 auf 1 gesetzt oder auf 0 zurückgesetzt wird. Das C steht für *Clock*, also für den englischen Begriff von Takt.

Wir gewinnen durch diese Schaltung also schonmal die Möglichkeit, zu festen Zeitpunkten neue Informationen zu speichern. Diese müssen kurz vor dem 1-Signal des Taktes anliegen und, während der Takt auf 1 steht, konstant anliegen. Die Signale sind dann während der ganzen Zeit, in der der Takt auf 0 liegt, gespeichert. Die Signale an S und R bei $C = 0$ sind irrelevant.

Allerdings: Noch haben wir das Problem mit den beiden Eingangs-1-Signalen nicht bewältigt. Dies können wir aber, und zwar mit dem *JK-MS-Flip-Flop*.

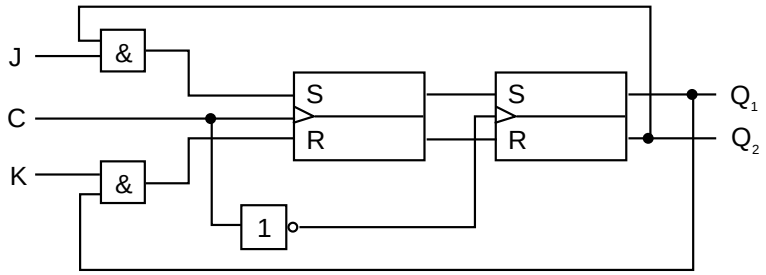

Abbildung 1.5: Zweiflanken-getaktetes JK-MS-Flip-Flop

Zweiflanken-getaktetes JK-MS-Flip-Flop

Wahrheitstabelle beim 1-Takt

J	K	Q_1	Q_2
0	0	$Q_{1,v}$	$Q_{2,v}$
1	0	1	0
0	1	0	1
1	1	$\overline{Q}_{1,v}$	$\overline{Q}_{2,v}$

Es sind hierbei zwei RS-FF's (*Master* und *Slave*, siehe Abbildung 1.5) derart hintereinandergeschaltet, daß bei Clocksignal 1 das erste Master-FF die anliegenden Signale verarbeitet, während das Slave-FF sperrt und die Ausgänge somit noch auf dem alten Zustand liegen. Fällt das Clocksignal wieder auf 0, so übernimmt der Slave-Teil die Informationen vom jetzt sperrenden Master-Teil und die Ausgänge Q_1 und Q_2 springen entsprechend um.

Die Vorteile des JK-MS-FF liegen darin, daß die Form des Taktimpulses weitgehend gleichgültig ist und daß während des Taktimpulses von den Ausgängen noch die Informationen der vorherigen Taktsequenz abgegriffen werden kann. Zudem besitzt diese Schaltung keine undefinierten Zustände mehr. Denn dadurch, daß das Master-FF durch den unterschiedlichen Takt vom Slave-FF abgekoppelt ist, kann man die Ausgänge des Slave-FF zum Master-FF rückkoppeln, ohne daß sich dies direkt auf die Ausgänge des JK-MS-Flip-Flops auswirkt (was bisher immer geschehen wäre). Da die Rückkopplung über AND-Gatter läuft (vgl. erneut mit Abb. 1.5) und ein Ausgang immer auf 0 liegt, können also nicht mehr beide Eingänge auf 1 liegen. Daher sind die Zustände der RS-Flip-Flops eindeutig definiert. Unser Dilemma von oben ist also bewältigt.

1.4 Zähler

Ein *Zähler* ist eine Schaltung, die, wie der Name es schon sagt, hoch- oder runterzählt. Eine prinzipielle Realisation eines extrem einfachen Zählers ist die in Abbildung 1.6 dargestellte Schaltung eines *Dualzählers*: Dort ist ein RS-Flip-Flop geeignet rückgekoppelt. Ist $Q_1 = 1$, so setzt dieses Flip-Flop beim nächsten Takt Q_1 zurück; und wenn $Q_1 = 0$, also $Q_2 = 1$, dann wird Q_1 beim nächsten Takt wieder gesetzt. Diese Schaltung diente allerdings nur zum prinzipiellen Verständnis. Denn beim RS-Flip-Flop würden die Ausgänge noch während des Taktes immer wieder rückkoppeln, was ein undefiniertes Verhalten hervorrufen würde. In praxi müßte man also ein entsprechend verschaltetes JK-MS-Flip-Flop verwenden.

Man unterscheidet nun zwischen *synchronen* und *asynchronen* Zählern. Hierbei sind die synchronen diejenigen, bei denen alle FF's durch ein und dasselbe Taktsignal getaktet werden. Bei den asynchronen Zählern liegt das Taktsignal nur am ersten FF an. Dabei erhalten nachfolgende FF's ihre Taktsignale

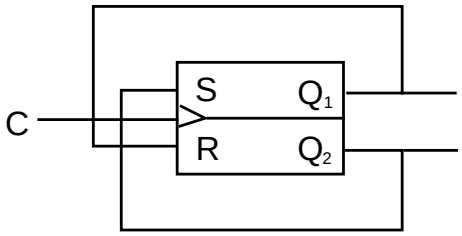

Abbildung 1.6: Dualzähler

immer von dem Ausgang Q_1 des jeweils vorangehenden FF. In Bild 1.7 und Bild 1.8 sind mögliche Schaltungen für synchrone und asynchrone Zähler angegeben.

Diese Zähler sind jeweils aus zweiflanken-getakteten JK-MS-FF's aufgebaut. Wir erklären hier hauptsächlich den synchronen Zähler:

Synchroner Zähler

Im Anfangszustand sollten über einen hier nicht mitskizzierten Eingang alle Ausgänge Q_1 bis Q_5 auf 0 gelegt werden. Der erste FF ändert nun bei jedem Clocksignal seinen Zustand Q_1, da an seinen beiden Eingängen ständig eine 1 anliegt. Der zweite FF benötigt zur Invertierung seines Ausganges ebenso 1/1 an seinen Eingängen. Zu beachten ist jedoch für ein folgendes FF, daß während des Clocksignals die Ausgänge der vorgeschalteten FF's noch auf dem alten Zustand stehen. So invertiert z.B. das zweite FF seinen Ausgang im $(i+1)$-ten Takt, wenn der Ausgang des ersten FF im i-ten Takt von 0 auf 1 gesprungen ist. Die AND-Gatter sorgen dafür, daß auf den Eingängen eines FF nur dann ein zur Invertierung notwendiges Paar 1/1 liegt, wenn alle vorherigen Ausgänge auf 1 liegen. Dies besagt insbesondere, daß in dem Takt, in dem das i-te FF (Bit) erstmalig auf 0 gesetzt wird, alle vorherigen FF's wieder auf 0 gesetzt werden.

Diese Schaltungsmechanismen ergeben zusammengefaßt folgende Wahrheitstabelle:

Wahrheitstabelle des Synchronen Zählers

Takt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	31
Q_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	...	1
Q_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	...	1
Q_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	...	1
Q_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	...	1
Q_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	...	1

Wir können das Verhalten von Zählern aber auch als mehrstufige *Phasenverdopplung* deuten. Hierbei wechselt das erste FF bei jedem Takt seinen Zustand, das zweite FF bei jedem zweiten Takt, das dritte bei jedem vierten Takt, allgemein das i-te bei jedem 2^{i-1}-ten Takt. Auch das können wir der Tabelle entnehmen.

Man erkennt, daß die Ziffernkombinationen Q_1 bis Q_5 gerade die Zahlen 0 bis 31 im Dualsystem darstellen. Greift man bei $\overline{Q_1}$ bis $\overline{Q_5}$ ab, so erkennt man ein Zurückzählen von 31 nach 0.

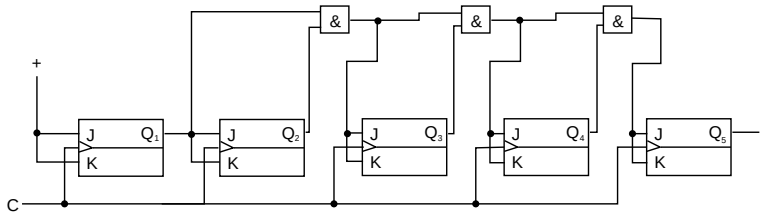

Abbildung 1.7: Synchroner Zähler

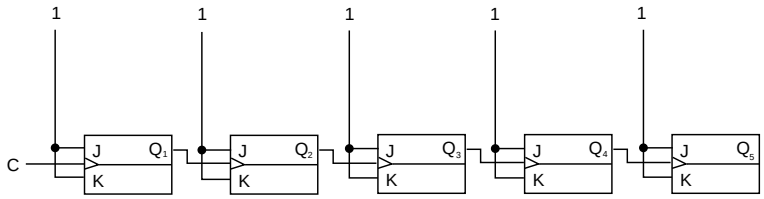

Abbildung 1.8: Asynchroner Zähler

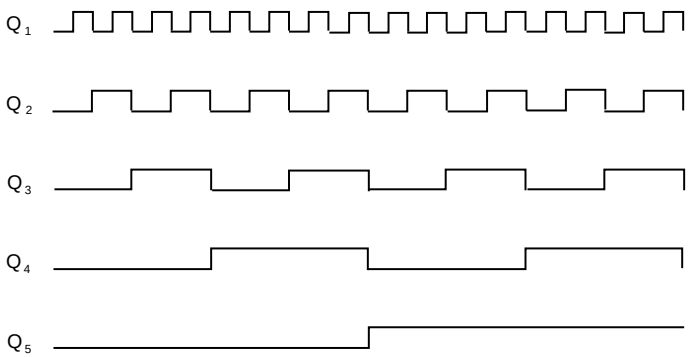

Abbildung 1.9: Phasenverdopplung

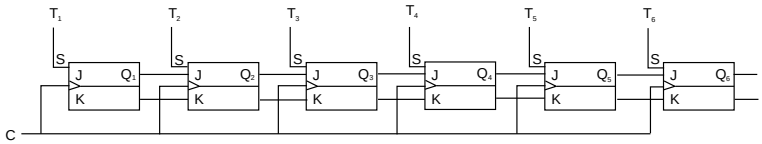

Abbildung 1.10: 6-Bit Schieberegister

Asynchrone Zähler

Bei asynchronen Zählern werden, wie oben angedeutet, nicht mehr alle FF's von den gleichen Taktimpulsen gesteuert, sondern nur noch der erste bekommt direkt einen solchen. Alle nachfolgenden FF's bekommen von ihren Vorgängern das Taktsignal. Anhand dieser asynchronen Schaltung erkennt man noch viel deutlicher die Phasenverdopplung, denn hier entspricht jeweils der Takt des $(i+1)$-ten Flip-Flops dem zählenden Wert Q_i.

Stehen alle Flip-Flops vor dem letzten Flip-Flop auf 1, so müssen alle FF's vor ihm umgeschaltet haben, bevor es seine Information erhält. Bei einem Taktsignal an das erste Flip-Flop dauert es also $T = n \cdot T_0$ (mit Umschaltzeit T_0 für ein FF), bis das Signal ankommt und alle FF's die richtige Zahl anzeigen. Beim synchronen Zähler schalten alle FF's gleichzeitig, die Signalübertragung braucht hier also nur die Zeit T_0. Diesen Nachteil gleicht der asynchrone Zähler durch seinen einfachen Aufbau aus.

1.5 Schieberegister

Ein *Schieberegister* dient dazu, mehrstellige Dualzahlen zu speichern. Darüberhinaus ist es möglich, verschiedene Operationen auf dem Schieberegister auszuführen.

In unserem Versuch wird ein 6-Bit-Schieberegister mit MS-FF's realisiert. Hierbei können mit den Tastern 1-6 die MS-FF's beliebig initialisiert werden. Bei einem kompletten Taktzyklus werden diese Zustände an das jeweils rechte FF weitergereicht. Bei Q_6 können die Informationen dann seriell ausgelesen werden, wir haben also einen Parallel(Eingabe)-Seriell(Ausgabe)-Wandler. Darüberhinaus erkennen wir in der Verschiebung nach rechts eine Division durch Zwei, vorausgesetzt, daß von links immer Nullen nachgeschoben werden.

Analog ist natürlich auch ein Seriell(Eingabe)-Parallel(Ausgabe)-Wandler realisiert. Dazu legt man vor jedem Takt (und natürlich während des Taktes) das einzulesende Signal an den Eingang J und invertiert an den Eingang K des ersten Flip-Flops. Nach 6 Takten ist die Zahl eingelesen und kann an den Ausgängen Q_1 bis Q_6 parallel abgelesen werden.

Eine weitere Möglichkeit, Operationen mit Schieberegistern auszuführen, ist die Rückkopplung des letzten mit dem ersten FF. Hierbei gibt es nun zwei Möglichkeiten. Wenn Ein- und Ausgänge überkreuz rückgekoppelt werden, so wird das im letzten FF vorliegende Signal invertiert an das erste FF weitergereicht. Nach 6 Takten liegen die Informationen invertiert wieder im Schieberegister vor. Werden die Ein- und Ausgänge aber parallel geschaltet, so wird das im letzen FF vorliegende Signal unverändert an das erste FF weitergereicht. Nach 6 Takten liegen die Information wieder unverändert an ihrer ursprünglichen Stelle. Es wurde also rotiert.

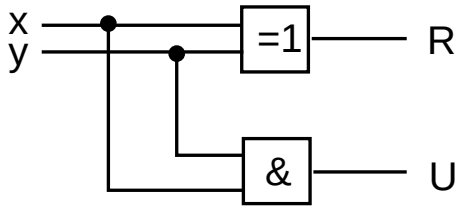

Abbildung 1.11: Halbaddierer

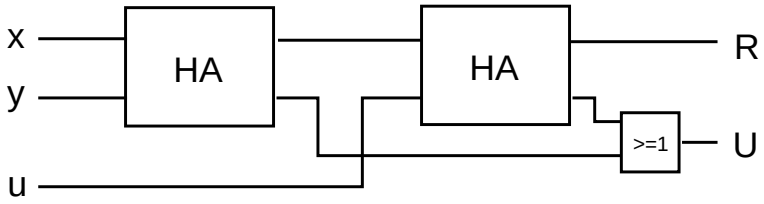

Abbildung 1.12: Volladdierer

1.6 Addierwerke

Mit Hilfe von Schieberegistern lassen sich *Addierwerke* bauen. Sie sind in der Lage Dualzahlen fester Länge zu addieren. Die einfachste Addierschaltung zur Addition von zwei 1-Bit-Zahlen ist der *Halbaddierer*. Er ist in Zeichnung 1.11 dargestellt. Die ihn vollständig charakterisierende Wahrheitstabelle ist in nachfolgender Tabelle angegeben. Der Halbaddierer addiert also zwei 1-Bit-Zahlen und gibt das Ergebnis und gegebenenfalls einen Übertrag aus.

Wahrheitstabelle Halbaddierer

x	y	R	U
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Wahrheitstabelle Volladdierer

x	y	u	R	U
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Eine weitere Addierschaltung ist der *Volladdierer*. Er läßt sich aus Halbaddierern zusammenbauen, wie Zeichnung 1.12 zeigt. Die zugehörige Wahrheitstabelle ist wieder angegeben. Der Volladdierer addiert also drei 1-Bit-Zahlen und gibt wieder Ergebnis und gegebenenfalls einen Übertrag aus. Als eine dieser 1-Bit-Zahlen wird oft zweckmäßigerweise der Übertrag aus der vorangegangenen Rechnung gewählt.

Mit Hilfe von Halbaddierern, Volladdierern und Schieberegistern lassen sich nun komplexere Addierwerke zur Addition von n-Bit-Zahlen aufbauen. Einige grundlegende Addierwerke sind der Serien- und der Parallel-Addierer.

Serien-Addierer

Einen typischen Serienaddierer zeigt Zeichnung 1.13. Er besteht aus zwei 4-Bit-Schieberegistern, in denen zu Anfang die zu addierenden Zahlen stehen, einem 1-Bit-Schieberegister, in dem jeweils der Übertrag der vorhergehenden Addition gespeichert wird, und einem Volladdierer. Das obere Schieberegister ist mit dem Volladdierer rückgekoppelt, so daß dort nach 4 Takten das Ergebnis der Rechnung steht. Im 1-Bit-Schieberegister steht dabei der Übertrag der Addition. Der Serienaddierer geht nun

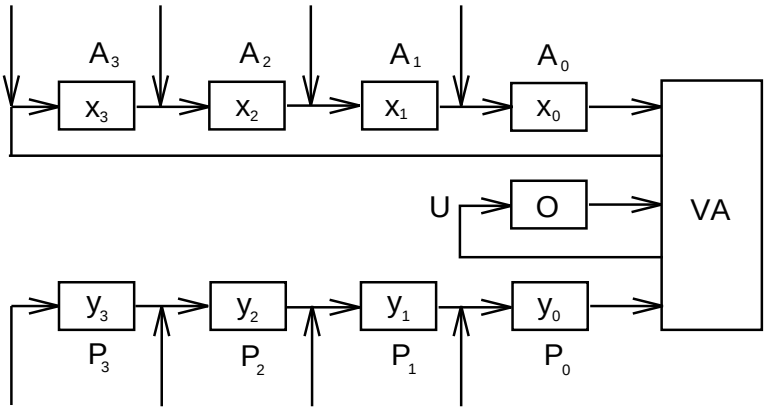

Abbildung 1.13: 4-Bit Serienaddierer

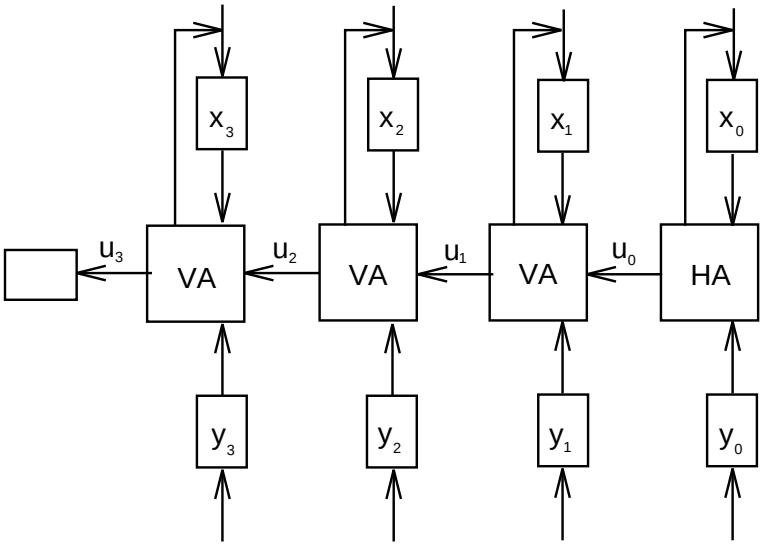

Abbildung 1.14: 4-Bit Paralleladdierer

folgendermaßen vor. Der VA addiert x_0 und y_0. Der Übertrag liegt nun am 1-Bit-Register an, das zu Anfang mit Null initialisiert ist. Das Ergebnis der Rechnung liegt nun an x_3 an. Beim nächsten Takt wandert der Übertrag nach U, das Ergebnis nach x_3 und es können nun x_1 und y_1 auf dieselbe Weise addiert werden, usw. Wir sehen, das der Serien-Addierer in 4 Schritten das Ergebnis liefert.

Parallel-Addierer

Das Parallel-Addierwerk liefert hingegen in einem Schritt das Ergebnis. Ein Paralleladdierer ist in Zeichnung 1.14 realisiert. Die Schnelligkeit der Rechnung ist natürlich durch die Vielzahl der Volladdierer erkauft. Die Funktionsweise ist aus der Schaltung direkt erkennbar: die Dualzahlen $x_3x_2x_1x_0$ und $y_3y_2y_1y_0$ werden addiert und das Ergebnis wieder als Zahl $x_3x_2x_1x_0$ in die oberen FF's geschrieben. Der Übertrag der Rechnung liegt an u_3 an.

Subtraktion von Dualzahlen

Die Addition dualer Zahlen kann in Kombination mit der Komplementbildung auch zur Subtraktion eingesetzt werden. Durch Invertieren jedes einzelnen Bits einer Dualzahl erhält man das sogenannte *Einerkomplement*. Praktisch hatten wir das Invertieren einer Dualzahl beim Schieberegister realisiert. Durch Addition von 1 auf das Einerkomplement erhält man das *Zweierkomplement*.

Die Subtraktion wird nun folgendermaßen bewerkstelligt: Man schreibt die Zahlen als gleich lange duale Zahlen, sagen wir z. B. als 7-Bit-Zahlen. Zahlen, die positiv sein sollen, bekommen noch eine Null als Vorzeichenbit vorgesetzt, also in unserem Beispiel ein 8. Bit des Wertes 0. Von allen negativen Zahlen (also auch den positiven Zahlen, die subtrahiert werden sollen) bildet man das Zweierkomplement und setzt eine Eins als Vorzeichenbit vor dieses Komplement. In unserem Beispiel erhalten wir also eine 8-Bit-Zahl, die mit 1 beginnt. Diese Zahlen addiert man nun wie gewohnt. Ein eventuelles 9. Übertrags-Bit wird ignoriert. Beginnt die Zahl mit 0 als achtem Bit, so ist die Zahl positiv und wir können sie direkt ablesen. Ist aber das Anfangsbit gleich Eins, so ist das Ergebnis negativ und wir bilden erneut das Zweierkomplement, um das Ergebnis direkt ablesen zu können.

Wir demonstrieren diese Beschreibung an ein paar Beispielen:

dezimal		dual		Einer-komplement		Zweier-komplement		Rechnung
+25	↷	011001	↷					0 011001
−11	↷	− 001011	↷	110100	↷	110101	↷	1 110101
+14	↶	001110	↶					1 0 001110

dezimal		dual		Einer-komplement		Zweier-komplement		Rechnung
+7	↷	00111	↷					0 00111
−22	↷	− 10110	↷	01001	↷	01010	↷	1 01010
				01110	↘		↶	1 10001
−15	↶	− 01111	↶			01111		

dezimal		dual		Einer-komplement		Zweier-komplement		Rechnung
−37	↷	− 100101	↷	011010	↷	011011	↷	1 011011
+43	↷	101011	↷					0 101011
+6	↶	000110	↶					1 0 000110

dezimal		dual		Einer-komplement		Zweier-komplement		Rechnung
−5	↷	− 0101	↷	1010	↷	1011	↷	1 1011
−8	↷	− 1000	↷	0111	↷	1000	↷	1 1000
				1100	↘		↶	1 1 0011
−13	↶	− 1101	↶			1101		

2. Erfahrungsbericht

2.1 Was ist alles schiefgelaufen...

Zähler

Beim Aufbau des Zählers war eines der sechs JK-MS-FlipFlops, wobei jeweils zwei in einem IC eingebaut waren, nicht korrekt ansprechbar. Anscheinend war die Leiste drumherum nicht in Ordnung. So bauten wir halt — mehr sollten wir nach Anleitung ja auch nicht tun — nur einen 5-Bit-Zähler.

Schieberegister

Wir bauten ein erstes 3-Bit-Schieberegister, das einwandfrei funktionierte. Danach fingen wir an, ein zweites 3-Bit-Schieberegister für den Addierer zusammenzustecken und testeten nochmals das erste, das auf einmal nicht mehr funktionierte. Wir bauten das zweite zu Ende, das einwandfrei arbeitete. Das erste wollte aber noch immer nicht. Dann schließlich wackelten wir an den Kabeln des ersten, woraufhin plötzlich auch dieses einwandfrei funktionierte. Jeder Baustein will halt nett aufgefordert werden, mitzuarbeiten :-)

Addierer

Der Addierer addierte 101 und 010 nicht richtig, 010 und 101 aber schon. Daraufhin vertauschten wir an dem Additionsbaustein die Kabel für Wert 1 und Wert 2. Plötzlich klappte alles. Anscheinend glaubte der Baustein nicht an das Kommutativgesetz der Addition :-)

...und sonst

Ansonsten haben sich mehrere Kabel einfach gelöst. So haben sich z. B. das positive Spannungsversorgungs-Kabel und später das Clock-Kabel beim Addierer gelöst. Wir haben dies aber jeweils recht bald bemerkt, auch wenn es jeweils ein paar Augenblicke dauerte...

3. Auswertung

Diese Auswertung besteht im wesentlichen aus den Wertetabellen der Tests, die wir mit den Schaltungen durchgeführt haben.

3.1 Latch

In der folgenden Tabelle sind die Messungen für das Latch wiedergegeben. Bei der Messung $X = 0$ und $Y = 0$ ist die Anzeige angegeben, die während des Anlegens dieser Werte vom Latch gezeigt wurde. Beim Umschalten auf $X = 1$ und $Y = 1$ wurde unkontrollierbar entweder 0/1 oder 1/0 angezeigt.

Bei der Messung für $X = 1$ und $Y = 1$ muß man unterscheiden, was vorher am Latcheingang angelegen hat; wir erwarten ja eine Invertierung der Eingangssignale, was einer Speicherung der Ausgangswerte entspricht. Diese Möglichkeiten sind durch das Wort „vorher“ gekennzeichnet.

Wahrheitstabelle

X	Y	Q_1	Q_2
0	0	1	1
1	0	0	1
0	1	1	0
1	1		
vorher			
0	1	1	0
1	0	0	1

3.2 Getaktetes RS-FlipFlop

Natürlich gilt hier Entsprechendes. Es ist wohl unnötig, nochmal alles im Detail zu erklären...

Wahrheitstabelle beim 1-Takt

S	R	Q_1	Q_2
0	0		
vorher			
1	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1
1	1	1	1

3.3 Dualzähler

Wir haben einen Dualzähler an einem JK-MS-Flip-Flop realisiert. Wenn das Taktsignal C gegeben wurde, sprangen die Wertepaare (Q_1, Q_2) jeweils von $(0,1)$ auf $(1,0)$, bzw. umgekehrt.

3.4 5-Bit-Zähler

Mit den Zählern konnte man hoch- und runterzählen, je nachdem ob man Q_i oder $\overline{Q_i}$ bei den Flip-Flops abgriff. Sowohl der synchrone als auch der asynchrone Zähler zählten von 0 bei Takt 0 bis 31 bei Takt 31 hoch. Bei Takt 32 sprang das Ergebnis wieder auf 0. Ebenso zählten beide Zählerarten einwandfrei herunter.

3.5 Schieberegister

Einfaches Durchschieben nach rechts

Hierbei lag der J-Eingang des ersten FlipFlops auf Null, der K-Eingang auf Eins. Es wurde bei diesem Versuch das Schieberegister von links mit Nullen aufgefüllt.

Invertieren

Hierbei wurden Ein- und Ausgänge des Schieberegisters über kreuz verbunden. Nach 6maligem Druck auf den Takter, lagen die Informationen des Schieberegisters in invertierter Form vor.

Rotieren

Hierbei wurden Ein- und Ausgänge des Schieberegisters parallel verbunden. Jedes Bit, was rechts herausgeschoben wurde, kam von links wieder in das Schieberegister.

3.6 Addierer

Folgende Additionen wurden von uns probeweise durchgeführt. Vom Assistenten wurden weitere Kombinationen erfolgreich getestet, die wir nicht mitprotokolliert haben.

Zahl 1	Zahl 2	Ergebnis
000	000	0000
000	001	0001
000	010	0010
000	011	0011
000	100	0100
000	101	0101
000	110	0110
000	111	0111

Zahl 1	Zahl 2	Ergebnis
001	011	0100
010	011	0101
011	101	1000
100	011	0111
100	101	1001
101	010	0111
110	011	1001
111	111	1110

3.7 Ergebnis

Die experimentellen Daten entsprechen voll unseren, in der Vorbereitung dargerstellten, Erwartungen. Bei einem digitalen Versuch ist da ja nicht mehr zu diskutieren. Über die verhältnismäßig geringen Probleme, die auftraten, haben wir ja bereits berichtet.

Literaturverzeichnis

Im folgenden wollen wir auf die Versuchsprotokolle der anderen Experimente im Fortgeschrittenenpraktikum, die von uns durchgeführt wurden und das vorliegende Protokoll hinsichtlich der theoretischen Grundlagen ergänzen, verweisen. Außerdem stellen wir eine Liste der Literatur auf, die wir am stärksten (nicht als einzige) zur Vorbereitung und Auswertung nutzten.

[FP-8a] O. Flimm, U. Münch: *Analoge Elektronik*, Versuch 8a im Fortgeschrittenenpraktikum am Institut für Kernphysik. Versuchsprotokoll. Köln, 1994

[Inf-1] Prof. T. Meis: *Informatik I.* Vorlesungsmitschrift von O. Flimm. Köln, SS 1993

[Inf-2] Prof. T. Meis: *Informatik II.* Vorlesungsmitschrift von O. Flimm. Köln, WS 1993/94

[Lo] R. Isernhagen: *Logischer Entwurf von Digitalschaltungen.* Valvo GmbH, Oktober 1970

[Re] W. Oberschelp und G. Vossen: *Rechneraufbau und Rechnerstrukturen.* Fünfte Auflage. München: R. Oldenbourg Verlag GmbH, 1990

Index