Vorbereitung: Schaltlogik

Marcel Köpke (1588978)
Gruppe 7

06.01.2012

Inhaltsverzeichnis

1 Bauelemente

Logische Gatter können durch Dioden, Transistoren und CMOS-Bauteile aufgebaut werden. Zumeist werden Dioden verwendet, da hier weniger Bauteile benötigt werden. Das Problem dabei ist jedoch, dass an den Dioden eine Spannung abfällt, sodass der Potentialunterschied zwischen dem LOW- und dem HIGH-Signal geringer wird.

1.1 AND-Gatter

A	B	C
1	1	1
1	0	0
0	1	0
0	0	0

Tabelle 1.1: Wahrheitstabelle

Am Ausgang «C» des AND-Gatters liegt nur 1 (HIGH) an, wenn auf beiden Eingängen «A» und «B» ebenfalls 1 anliegt. Ansonsten wird 0 (LOW) durchgeschaltet. Eine mögliche Realisierung des AND-Gatters besteht aus 2 Dioden und einem Widerstand:

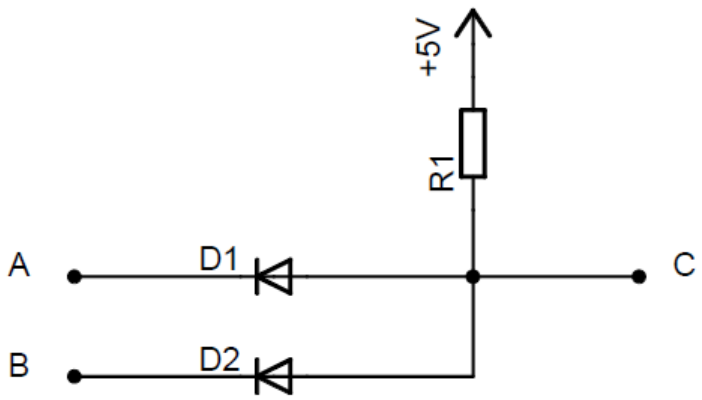

Abbildung 1.1: AND-Gatter

Die Dioden lassen den Strom nur in Pfeilrichtung durch. Liegt also auf einem der Eingänge 0 an, so schalten die Dioden durch und idealierweise fällt die komplette Spannung

an diesen ab ⇒ am Ausgang liegt ebenfalls 0 an. Liegt jedoch an beiden Dioden 1 an, so findet kein Potentialabfall statt und der Ausgang befindet sich ebenfalls auf 1.

Das AND-Gatter kann auch mit einer TTL-Schaltung («Transistor-Transistor-Logik») realisiert werden:

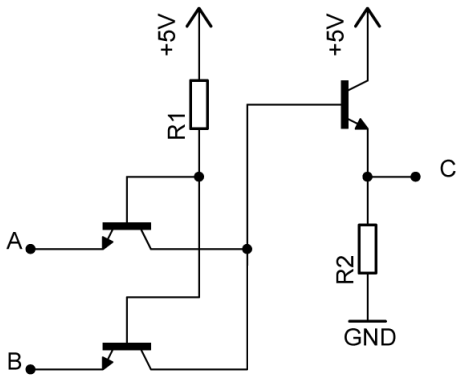

Abbildung 1.2: AND-Gatter (TTL)

Das Schaltsymbol des AND-Gatters ist:

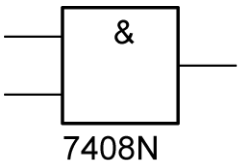

Abbildung 1.3: Schaltsymbol

1.2 NOT-Gatter

A	B
1	0
0	1

Tabelle 1.2: Wahrheitstabelle

Umgangsprachlich kehrt das NOT-Gatter den angelegten Eingang um, es negiert ihn also logisch. Das folgende Schaltbild zeigt eine TTL-Realisierung:

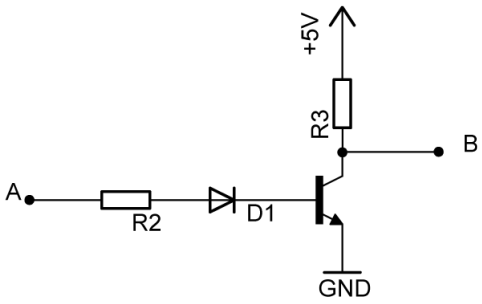

Abbildung 1.4: NOT-Gatter

Legt man am Eingang «A» 1 an, so fließt ein Strom durch die Basis des Transistors und dieser schaltet durch. Damit fließt ein Strom über den Widerstand «R3» und den Transistor in die Erdung. Damit liegt (idealerweise) keine Spannung am Ausgang «B» an: 0

Legt man jedoch 0 auf den Eingang, so fließt kein Basisstrom, der Transistor sperrt und der Ausgang «B» befindet sich auf hohem Potential: 1

Das Schaltsymbol des NOT-Gatters ist:

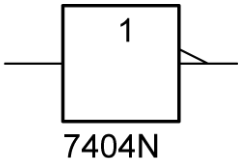

Abbildung 1.5: Schaltsymbol

Das NOT-Verhalten wird durch den schrägen Balken am Quadrat charakterisiert.

1.3 NAND-Gatter

A	B	C
1	1	0
1	0	1
0	1	1
0	0	1

Tabelle 1.3: Wahrheitstabelle

Das NAND-Gatter ist einfach die «Umkehrung» des AND-Gatters, also die Negierung. Es liegt nahe dies durch ein Hintereinanderschalten eines AND- und eines NOT-Gatters zu realisieren:

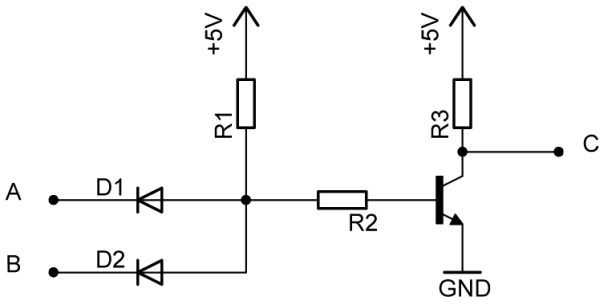

Abbildung 1.6: NAND-Gatter

Das entsprechende Schaltsymbol ist:

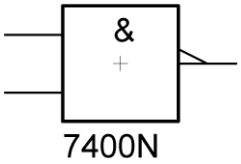

Abbildung 1.7: Schaltsymbol

Der Schräge Balken nach dem Quadrat steht auch hier wieder für die Negierung.

1.4 OR-Gatter

A	B	C
1	1	1
1	0	1
0	1	1
0	0	0

Tabelle 1.4: Wahrheitstabelle

Der Ausgang «C» des OR-Gatters ist immer genau dann 1, wenn mindestens einer der beiden Eingänge «A» oder «B» auf 1 liegt. Ansonsten ist der Ausgang auf 0.

Schalttechnisch ist das OR-Gatter dem AND-Gatter ähnlich. Es besteht wiederum aus 2 Dioden und einem Widerstand. Nur sind die Dioden hier umgedreht und der Widerstand wird geerdet:

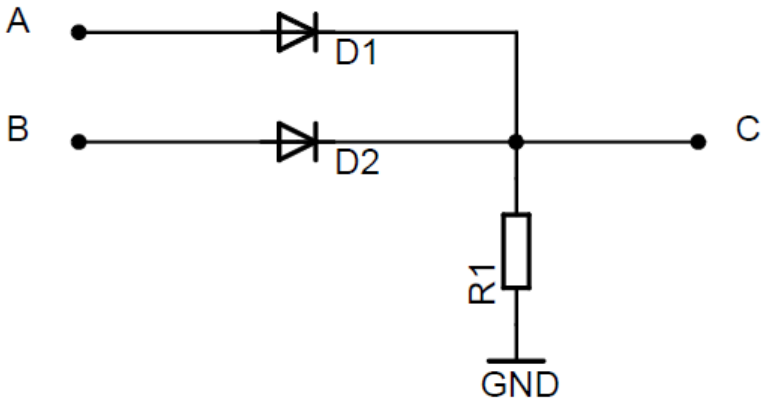

Abbildung 1.8: OR-Gatter

Das Schaltsymbol ist:

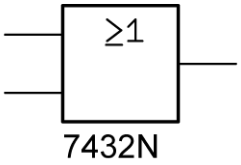

Abbildung 1.9: Schaltsymbol

1.5 Fragen

1.5.1 Welche Probleme entstehen beim Hintereinanderschalten vieler solcher Dioden-Gatter?

An den Dioden fällt eine Spannung ab, sodass der Unterschied zwischen dem HIGH- und dem LOW-Signal mit zunehmender Bauteilanzahl immer geringer wird. Im Extremfall kann dann nicht mehr zwischen 0 und 1 unterschieden werden!

1.5.2 Wozu werden die Dioden im AND-/OR-Gatter benötigt?

Die Dioden sorgen dafür, dass sich die Eingänge des Gatters nicht gegenseitig beeinflussen, da sie in einer Richtung immer sperren! Dadurch ist eine Beeinflussung ausgeschlossen.

1.5.3 Wie ließe sich ein AND-/OR-Gatter mit mehr als zwei (beliebig vielen) Eingängen realisieren?

Man fügt einfach mehr Anschlussarme mit einer Diode an. Dann zeigt das resultierende Gatter ebenfalls die Charakteristik des «kleinen» AND-/OR-Gatters.

2 Weitere logische Gatter

Die im folgenden besprochenen Gatter können aus den oben bereits beschriebenen Gattern zusammengeschaltet werden. Integriert man so mehrere Gatter in ein «neues» Bauteil, so wird dies IC (Integrated Circuit) genannt.

Im weiteren wird noch folgendes Gatter verwendet/eingeführt:

- NOR-Gatter

A	B	C
1	1	0
1	0	0
0	1	0
0	0	1

Abbildung 2.1: Wahrheitstabelle NOR

Kann analog zum NAND-Gattter aufgebaut werden. Das Schaltsymbol ist:

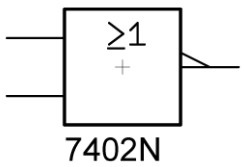

Abbildung 2.2: Schaltsymbol NOR

2.1 Inverter aus NAND bzw. NOR

Hier soll aus einem NAND- bzw. einem NOR-Gatter ein NOT-Gatter realisiert werden.

Die Wahrheitstabelle des NAND-Gatters war:

A	B	C
1	1	0
1	0	1
0	1	1
0	0	1

Tabelle 2.1: Wahrheitstabelle NAND

Man sieht, dass für die Zeilen 1, 3 und 4 der Ausgang «C» bereits die Negierung des Eingangssignals «A» ist.

Unter anderem sind Zeile 1 und 4 interessant. Man sieht dass, wenn «A» und «B» auf demselben logischen Zustand sind das Ausgangssignal gerade für «A» (bzw. dann auch für «B») negiert wird. Praktisch schließt man also einfach die beiden Eingänge kurz um ein NOT-Gatter zu realisieren.

Man sieht auch leicht ein, dass wenn «B» auf 1 liegt, «A» immer negiert wird. Man kann also auch einfach konstant 1 an einen Eingang des NAND-Gatters anlegen und den anderen als «tatsächlichen» Eingang für das «neue» NOT-Gatter verwenden.

Entsprechendes Funktioniert auch mit einem NOR-Gatter. Die Wahrheitstabelle war:

A	B	C
1	1	0
1	0	0
0	1	0
0	0	1

Abbildung 2.3: Wahrheitstabelle NOR

Kurzschließen ist auch hier wieder eine Möglichkeit. Außerdem kann als zweite Möglichkeit ein Eingang konstant auf 0 gelegt werden.

Damit kann das NOT-Gatter auch durch folgende Schaltsymbole dargestellt werden:

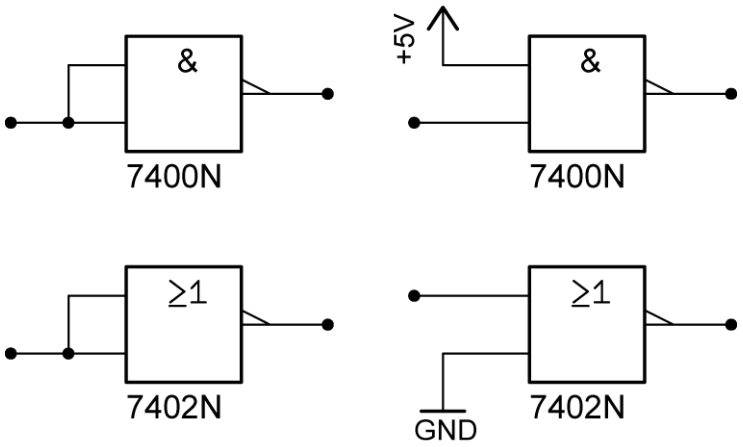

Abbildung 2.4: NOT aus NAND bzw. NOR

2.2 XOR-Gatter

A	B	C
1	1	0
1	0	1
0	1	1
0	0	0

Tabelle 2.2: Wahrheitstabelle XOR

Verhält sich ähnlich wie das OR-Gatter mit dem Unterschied das 1 XOR 1 = 0 ist. Das Schaltsymbol ist:

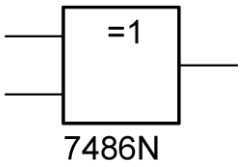

Abbildung 2.5: Schaltsymbol XOR

Die Wahrheitstabelle kann auch kurz in folgender Schreibweise dargestellt werden:

$$C = (\overline{A} \wedge B) \vee (A \wedge \overline{B}) \tag{2.1}$$

Man sieht also, dass für das XOR-Gatter 2 NOT-Gatter, 2 AND-Gatter und 1 OR-Gatter verwendet werden können um es zu realisieren. Nutzt man nun noch die Möglichkeit, dass

das NOT-Gatter durch eine NAND-Gatter dargestellt werden kann so folgt folgender Schaltplan:

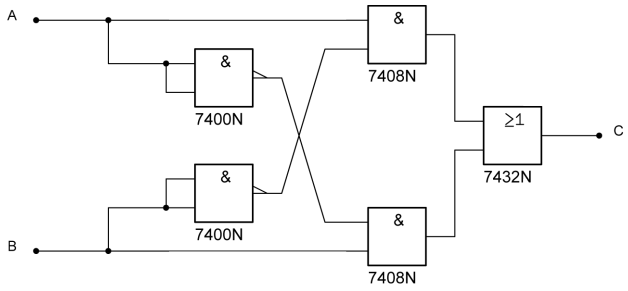

Abbildung 2.6: XOR-Gatter

2.3 XOR aus NAND

Formt man Gleichung 2.1 entsprechnend um so sieht man, dass das XOR-Gatter auch ausschließlich aus NAND-Gattern augebaut werden kann:

$$
\begin{aligned}
C &= (A \wedge \overline{B}) \vee (\overline{A} \wedge B) \\
&= (A \wedge \overline{B}) \vee (A \wedge \overline{A}) \vee (\overline{A} \wedge B) \vee (B \wedge \overline{B}) \\
&= A \wedge (\overline{A} \vee \overline{B}) \vee B \wedge (\overline{A} \vee \overline{B}) \\
&= A \wedge \left(\overline{\overline{\overline{A} \vee \overline{B}}}\right) \vee B \wedge \left(\overline{\overline{\overline{A} \vee \overline{B}}}\right) \\
&= A \wedge \overline{(\overline{A} \wedge \overline{B})} \vee B \wedge \overline{(\overline{A} \wedge \overline{B})} \\
&= \overline{\overline{\underbrace{A \wedge \overline{(\overline{A} \wedge \overline{B})}}_{X} \vee \underbrace{B \wedge \overline{(\overline{A} \wedge \overline{B})}}_{Y}}} \\
&= \overline{\overline{A \wedge \overline{(\overline{A} \wedge \overline{B})}} \wedge \overline{B \wedge \overline{(\overline{A} \wedge \overline{B})}}} \\
&= \overline{\overline{A\overline{AB}B\overline{AB}}}
\end{aligned}
$$

Abbildung 2.7: Quelle: Vorbereitungshilfe

Aus der letzten Zeile ergibt sich also folgender Schaltplan:

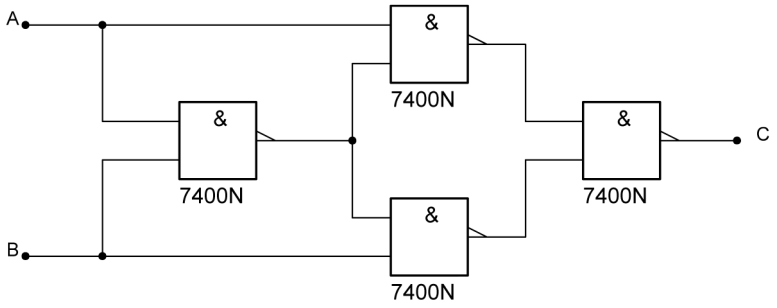

Abbildung 2.8: XOR-Gatter aus NAND

2.4 Fragen

2.4.1 Was bewirkt das Offenlassen eines Eingangs eines ICs?

Der «offene» Eingang verhält sich, so als ob 1 angelegt worden wäre.

2.4.2 Wie erhält man aus einen NAND- bzw. NOR-Gatter ein NOT-Gatter

siehe oben

2.4.3 Wie kommt man, ausgehend von der Wahrheitstabelle einer Funktion, auf deren disjunkte Normalform?

Man geht Schrittweise vor:

Zuerst nimmt man an, dass A und B auf 1 liegen. Dann betrachtet man die Zeilen in denen der Ausgang auf 1 liegt. Jede dieser Zeilen stellt eine «Bedingung» die durch eine UND-Verknüpfung realisiert wird. Beispielsweise ist A = 1 und B = 0. Dann ist die entsprechende Bedinung dazu: $A \wedge \overline{B}$. Alle Bedingungen werden daraufhin mit OR verknüpft.

Die Fälle, in denen A und B nicht beide auf 1 liegen bzw. C auf 0 sind dann automatisch auch erfüllt! Durch weiteres Umformen kann nur noch eine verkürzte Form gefunden werden.

2.4.4 Welche Vorteile ergeben sich bei einer Umforumg der disjunkten Normalform in die reine NAND- oder NOR-Schreibweise?

Will man einen IC tatsächlich industriell fertigen, so ist es einfacher und kostengünstiger mehrere gleiche Bauteile zu benutzen als viele verschiedene.

3 Addierer

3.1 Halbaddierer

A	B	S	Ü
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

Tabelle 3.1: Funktionstabelle

Der Halbaddierer addiert zwei 1-Bit Werte «A» und «B», und gibt dabei eine zwei Bit-Zahl aus die aus der Summe «S» und dem Überlauf/Übertrag «Ü» besteht. Man sieht leicht ein, dass «S» das Ergebnis einer XOR- und «Ü» das Ergebnis einer AND-Verknüpfung von «A» und «B» ist. Daraus ergibt sich folgendes Schaltbild:

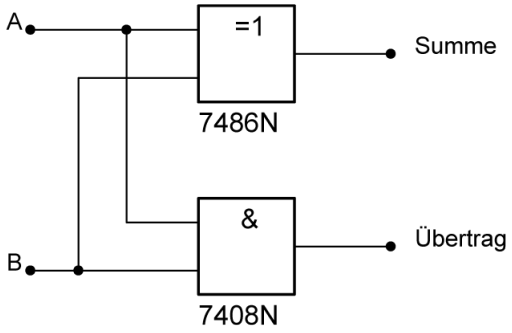

Abbildung 3.1: Halbaddierer

3.2 Volladdierer

Um mehr als zwei 1-Bit Werte zu addieren benötigt man einen weiteren Halbaddierer der den Übertrag «U_{-1}» einer vorran gegangenen Addition berücksichtigt. Der resultierende Übertrag «Ü» wird dann mit einem OR-Gatter und den Teilüberträgen der beiden Halbaddierer erzeugt, da aus jedem Halbadditionsschritt der Übertrag «mitgenommen» werden muss.

A	B	U_{-1}	S	U
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

Tabelle 3.2: Funktionstabelle

Man erhält also folgenden Schaltplan für 1-Bit Werte:

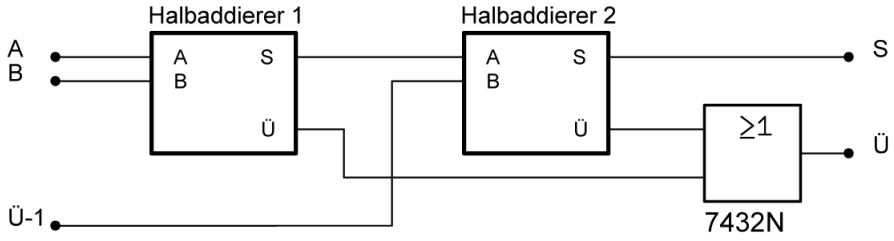

Abbildung 3.2: 1-Bit-Volladdierer

Entsprechend mit der Gatterdarstellung des Halbaddierers:

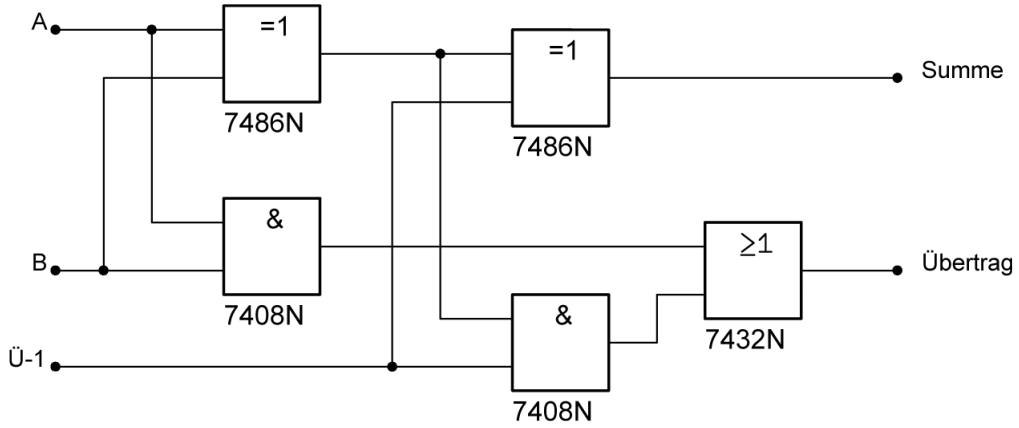

Abbildung 3.3: 1-Bit-Volladdierer (mit Gatter)

Will man nun mehr-Bit-Zahlen addieren, so schaltet man einfach mehrere 1-Bit-Volladdierer hinteinander, indem man den Übertrag des vorangehenden Addierers an den U_{-1}-Anschluss

des nächsten Addierers anschließst. Das Ergebnis wird dann an den «S» Anschlüssen für die entsprechende Binärstelle ausgegeben. Hier Beispielhaft für einen 4-Bit-Volladdierer:

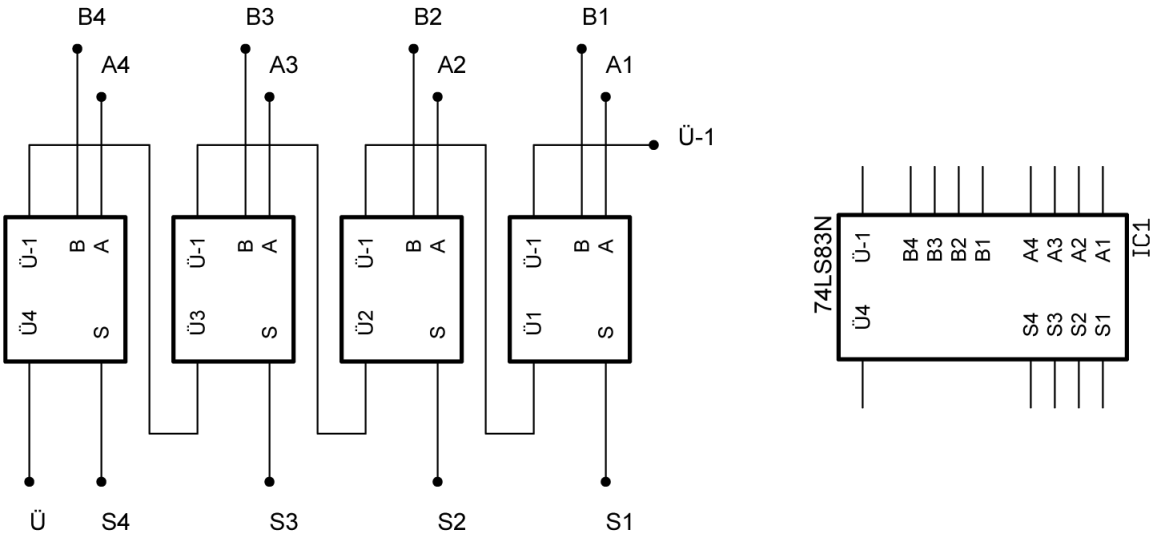

Abbildung 3.4: 4-Bit-Volladdierer

3.3 Subtrahierer

Ein Subtrahierer ist im Prinzip eine normaler Volladdierer, der jedoch das «negative» Zweier-Komplement einer Zahl zu einer anderen Zahl addiert. Aus $A - B$ wird also $A + (-B)$ bzw. $A + \overline{B}$. Ein 4-Bit-Subtrahierer z.B. besteht dann aus einem 4-Bit-Volladdierer, 5-NOT-Gattern und 4 XOR-Gattern:

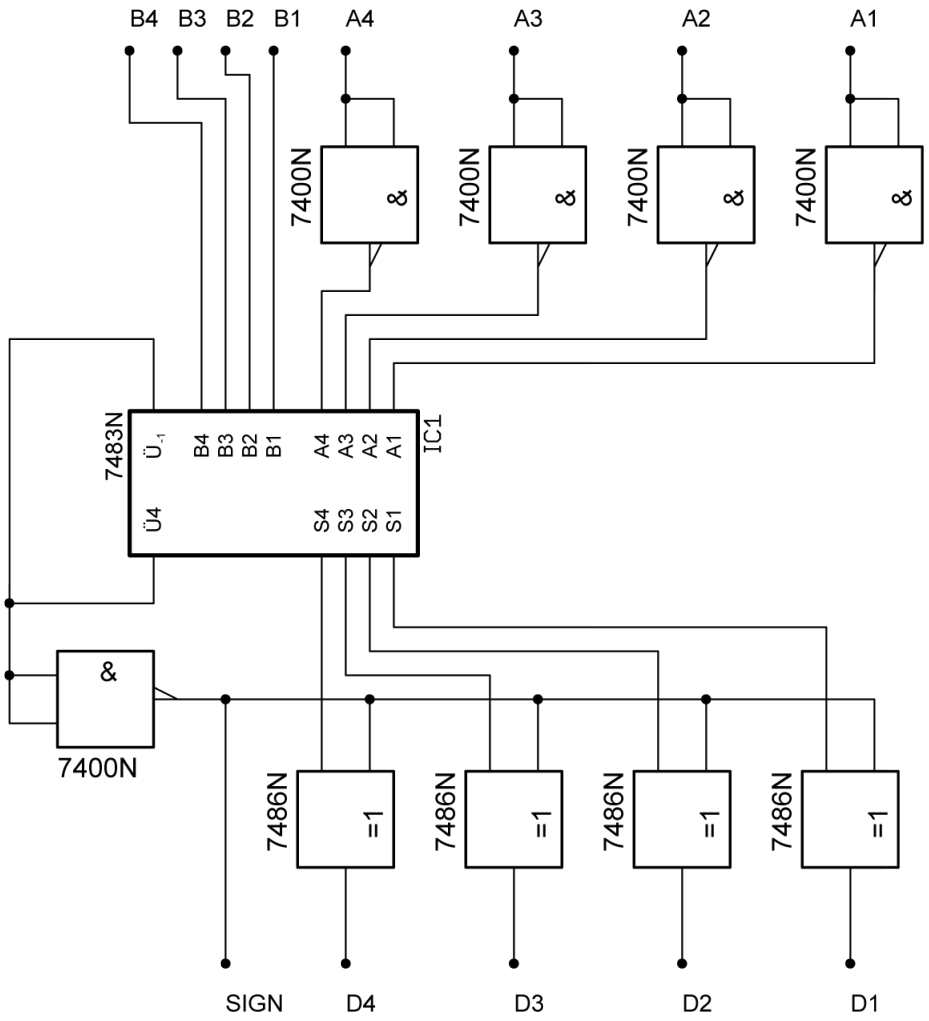

Abbildung 3.5: 4-Bit-Subtrahierer

Wie gesagt werden die Eingänge A_1 bis A_4 negiert und dann auf B_1 bis B_4 addiert. Jedoch müssen noch drei fälle Unterschieden werden:

1. $B - A > 0$:
   Das Ergebnis ist positiv, damit ist $\ddot{U}_4 = 1$ und ein Übertrag wird zusätzlich dazuaddiert da U_{-1} mit U_4 kurzgeschlossen ist. Durch das NOT-Gatter liegt SIGN dann auf 0 (stellvertretend für Plus) und die XOR-Gatter beinflussen das Ergebnis nicht weiter.

2. $B - A < 0$:
   Das Ergebnis ist negativ, damit ist $U_4 = 0$ und es wird kein zusätzlicher Übertrag dazuaddiert. SIGN liegt dann auf 1 (stellvertretend Minus). Dadurch negieren die XOR-Gatter das Ergebnis, sodass es in die Zweier-Komplementdarstellung gebracht

wird.

3. $B - A = 0$:
   Unabhängig davon ob U_{-1} vor der Addition auf 0 oder 1 liegt wird das richtige Ergebnis an D_1 bis D_4 zurückgegeben (nämlich 0). Jedoch kann das Vorzeichen (SIGN) fehlerhaft sein (siehe Fragen).

3.4 Fragen

3.4.1 Welches Problem ergibt sich beim Halbaddierer?

Der Halbaddierer ist nicht in der Lage einen bereits zuvor berechneten Übertrag zu berücksichtigen. Man kann also nur 1-Bit Werte miteinander addieren, jedoch keine größeren Zahlen.

3.4.2 Wie kommt man darauf aus welchen Gattern der Halbaddierer aufgebaut ist?

Aus dem Vergleich der Funktionstabelle mit den Wahrheitstabellen der Gatter.

3.4.3 Wozu wird das OR-Gatter beim Volladdierer benötigt?

Bei den beiden Halbadditionen ergeben sich zwei Teilüberträge. Das OR-Gatter verknüpft diese sinnvoll, sodass kein Übertrag «verloren geht».

3.4.4 Wozu werden beim Subtrahierer XOR-Gatter benötigt und in welchem Fall haben sie kein Wirkung?

Die XOR-Gatter negieren, dass Ergebnis im Fall $B - A < 0$ und bringen es so in die richtige Zweier-Komplementdarstellung. Für $B - A = 0$ negieren sie ebenfalls das Ergebnis S_1 bis S_i falls U_{-1} vor der Addition auf 0 war und bringen so die Ausgänge D_1 bis D_i alle auf 0.

Im Fall $B - A > 0$ (bzw. $B - A = 0$ und $U_{-1} = 1$) haben sie keine Wirkung.

3.4.5 Welches Problem besteht beim Subtrahierer im Bezug auf die Ausgabe der Null?

Je nachdem ob vor der Addition U_{-1} auf 0 oder 1 lag wird an den Ausgängen S_1 bis S_i das Ergebnis NOT 0 oder 0 ausgegeben. Da aber in jedem Fall $U_4 = U_{-1}$ vor und nach der Addition gilt kann durch die Konstruktion mit dem NOT-Gatter und den XOR-Gattern das Ergebnis an D_1 bis D_i immer auf 0 gesetzt werden. Allerdings liegt dann SIGN nicht immer auf 0!

Für den Fall des 4-Bit-Subtrahierers, $B - A = 0$ und SIGN = 1, wird also eigentlich das Ergebnis -16 zurückgegeben.

4 Speicherelemente

Eines der einfachsten Speicherelemente ist ein FlipFlop. Er speicher einen Zustand solange bis ein anderer gesetzt wird.

4.1 RS-FlipFlop (RS-FF)

Der Reset-Set-FlipFlop besitzt zwei Eingänge «R» und «S». Das nachfolgende Schaubild die Funktionstabelle stellt den RS-FF ohne vorgeschaltete NOT-Gatter jedoch mit den Eingängen «$\overline{S}$» und «$\overline{R}$». Dann kann Q mit S=1 auf 1 bzw. mit R=1 auf 0 gesetzt werden. S=R=1 ist jedoch verboten!

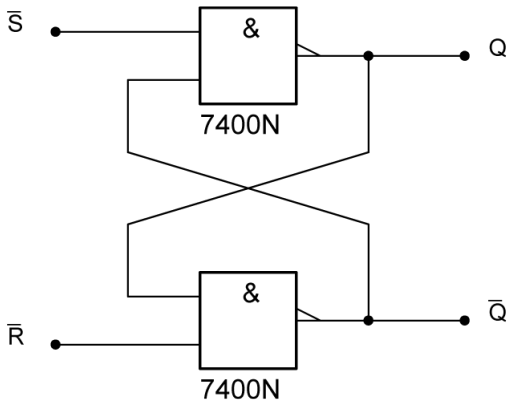

Abbildung 4.1: RS-FlipFlop

Wie man sieht ist ein Ausgang und ein Eingang des jeweils anderen NAND-Gatters kurzgeschlossen. Damit wird jedes Gatter von dem Ergebnis, des jeweils anderen abhängig!

$\overline{S}$	$\overline{R}$	S	R	Q_n	$\overline{Q_n}$	
1	1	0	0	Q_{n-1}	$\overline{Q_{n-1}}$	keine Änderung
1	0	0	1	0	1	setze 0
0	1	1	0	1	0	setze 1
0	0	1	1	1	1	verboten

Tabelle 4.1: Funktionstabelle

4.2 Getaktetes RS-FlipFlop (RST-FF)

Im Prinzip ein RS-FF, nur dass ein weiterer Takt-Eingang «T» existiert, der dafür sorgt, dass nur dann Signale auf das RS-FF durchgelassen werden wenn T auf 1 liegt (sprich wenn T, R und S an einem Taktgeber angeschlossen sind, wird nur im Arbeitszyklus ein Signal an das RS-FF weiter gegeben). Das entsprechende Schaltbild dazu:

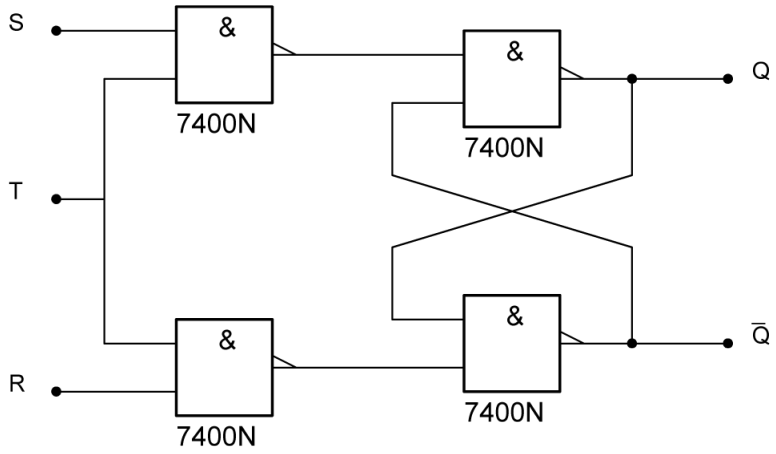

Abbildung 4.2: RST-FF

Und die Funktionstabelle:

T	S	R	Q_n	$\overline{Q_n}$	
0	0	0	Q_{n-1}	$\overline{Q_{n-1}}$	kein Takt, keine Änderung
0	0	1	Q_{n-1}	$\overline{Q_{n-1}}$	kein Takt, keine Änderung
0	1	0	Q_{n-1}	$\overline{Q_{n-1}}$	kein Takt, keine Änderung
0	1	1	Q_{n-1}	$\overline{Q_{n-1}}$	kein Takt, keine Änderung
1	0	0	Q_{n-1}	$\overline{Q_{n-1}}$	Takt, aber keine Änderung
1	0	1	0	1	setze 0
1	1	0	1	0	setze 1
1	1	1	1	1	verboten

Tabelle 4.2: Funktionstabelle

Man kann diesen Aufbau noch dahingehend verändern, dass man den Verbotenen Zustand R=S=1 umgeht. Dazu ist es sinnvoll R als das negierte S Signal zu setzen. Um Bauteile zu sparen erreicht man dies indem man für R den Ausgang des ersten NAND-Gatters an S benutzt:

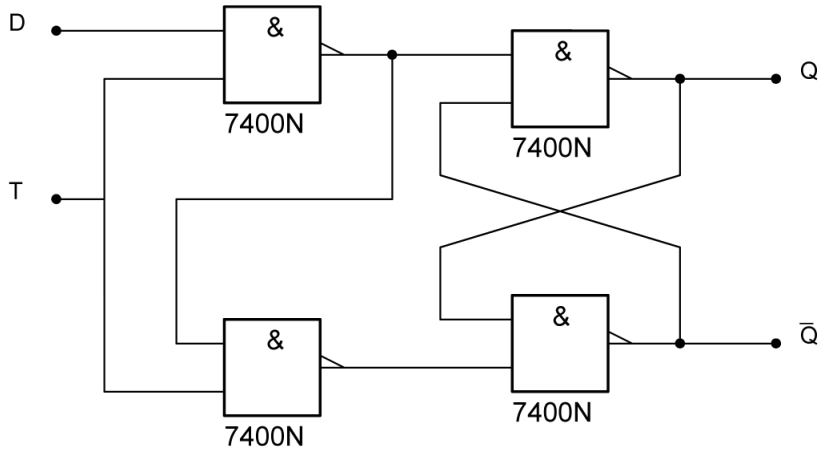

Abbildung 4.3: Data-FlipFlop

Diesen Aufbau nennt man Data-FlipFlop. Im Schaltbild ist der Eingang mit «D» bezeichnet. Der Nachteil dieser Schaltung ist, dass im Prinzip der FlipFlop bei jedem Takt T=1 gesetzt wird. Die Information wird also nur einen vollständigen Takt lang auf den Ausgängen gehalten.

4.3 Taktzustands- und Taktflankensteuerung

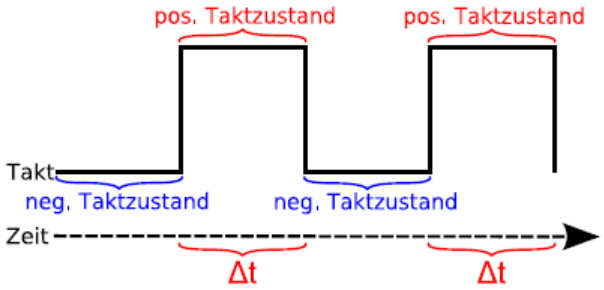

Abbildung 4.4: Taktzustandssteuerung

Die Abbildung beschreibt ein Taktzustandsgesteuertes FlipFlop. Man sieht, dass der positive Taktzustand 1 über einen Zeitraum Δt vorhanden ist. Während dieser Zeit kann der FlipFlop beliebig geändert werden, die Ausgänge übernehmen den angelegten Zustand. Geht der Takt in den negativen Zustand 0 über, so bleibt der letzte angelegte Zustand erhalten.

Der Unterschied zur Taktflankensteuerung besteht nun darin, dass bei dieser versucht wird den Zeitraum Δt, in welchem eine Änderung vollzogen werden kann, möglichst

klein zu halten, indem man die Potentialänderung als logische Zustandswerte auffasst. Folgende Abbildung zeigt eine solche Steuerung:

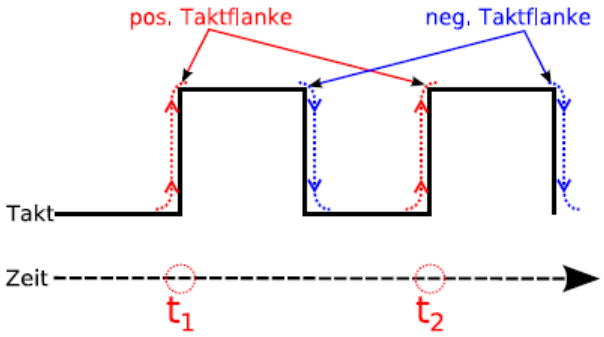

Abbildung 4.5: Taktflankensteuerung

Hier wird der Übergang vom positiven zum negativen (bzw. umgekehrt) Taktzustand als Takt verstanden. Die konstanten Taktzustände werden als 0 angesehen, sodass eine Änderung der Eingänge eines FlipFlops während diesen nichts bewirken. Erst beim Übergang zwischen zwei Taktzuständen kann der FlipFlop mit dem zuvor angelegten Signal beschrieben werden. Man unterscheidet allgemein noch positive und negative Flanken!

4.4 Entprellen eines Schalters

Bei mechanischen Schaltern kann es vorkommen, dass beim umlegen des Schalters der Kontakt z.B. kurz nachdem er hergestellt wurde wieder gelöst und dann erneut geschlossen wird. Dies nennt man «Prellen». Will man beispielsweise mit einem Zähler feststellen wie oft ein Schalter geschlossen wurde, so kann dies zu Fehlern führen.

Um Schalter zu entprellen benutzt man RS-FlipFlops:

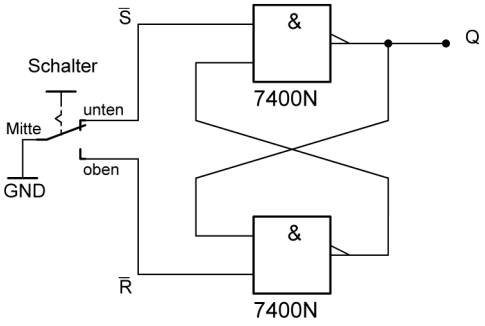

Abbildung 4.6: Entprellschaltung

Wie bereits zuvor erwähnt gelten nicht angeschlossene Eingänge eines Gatters, so als ob dort 1 anliege. Prellt nun ein Schalter beim umlegen, so verliert er kurzzeitig den Kontakt. Dies macht jedoch keinen Unterschied, denn um den Zustand des Flipflops zu ändern müsste der jeweils andere Kontakt zuerst geschlossen werden. Dies geschieht aber auf keinen Fall.

4.5 Jump-Kill-Master-Slave-FlipFlop (JK-MS-FF)

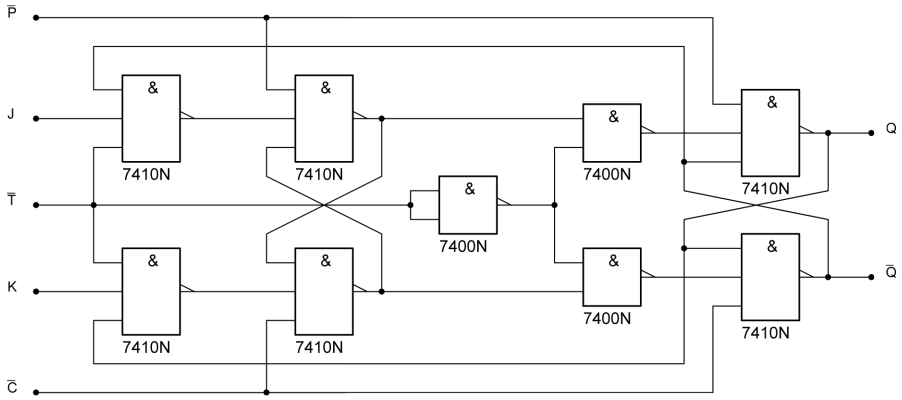

Abbildung 4.7: JK-MS-FF

Die Abbildung zeigt ein JK-MS-FF. Es besteht prinzipiell aus 2 RST-FF. Jedoch ist der Takteingang des zweiten FF (Slave) über ein NOT-Gatter (Inverter) an den äußeren Takteingang angeschlossen. Damit ist immer eins der beiden RST-FF «aktiv» und kann beschrieben werden.

Gilt T=0 so kann das erste RST-FF (Master) über den Eingang J (Jump) beschrieben oder durch K (Kill) resettet werden. Wird nun T auf 1 geschaltet so kann das Master-FF nicht mehr geändert werden. Jedoch ist nun das Slave-FF aktiv und verarbeitet die Ausgänge des Master-FF weiter. Dieses gibt auch den Zustand an den Ausgänge Q und $\overline{Q}$ weiter. Man benötigt also immer zuerst eine negative und dann eine positive Taktflanke um das JK-MS-FF zu beschreiben und den Zustand an die Ausgänge weiter zu geben! Man spricht auch von einem positiv taktflankengesteuerten FlipFlop.

Hier die Funktionstabelle:

J	K	Taktflanke	Q_{Slave}	$\overline{Q_{Slave}}$	
0	0	+	Q_{Master}	$\overline{Q_{Master}}$	keine Änderung
0	0	-	Q_{Master}	$\overline{Q_{Master}}$	falsche Taktflanke ⇒ keine Änderung
0	1	+	0	1	setze 0
0	1	-	Q_{Master}	$\overline{Q_{Master}}$	falsche Taktflanke ⇒ keine Änderung
1	0	+	1	0	setze 1
1	0	-	Q_{Master}	$\overline{Q_{Master}}$	falsche Taktflanke ⇒ keine Änderung
1	1	+	$\overline{Q_{Master}}$	Q_{Master}	toggeln
1	1	-	Q_{Master}	$\overline{Q_{Master}}$	falsche Taktflanke ⇒ keine Änderung

Tabelle 4.3: Funktionstabelle

Mit den Eingängen P und C können die Ausgänge des JK-MS-FF direkt geschaltet werden. Soll das FlipFlop normal betrieben werden so sind beide auf 1 (offen).

4.6 Fragen

4.6.1 Wie entsteht der verbotene Zustand des RS-FF? Warum ist er verboten?

Der verboten Zustand ist: R=S=1. Dabei sollen also gleichzeitig 1 und 0 gesetzt werden. Dies ist nicht möglich und daher verboten. (Tatsächlich wird $Q = \overline{Q} = 1$ gesetzt und somit nicht sinnvolles Verhalten hervorgerufen).

4.6.2 Wie kann der verboten Zustand beim RST-FF umgangen werden?

Z.B. indem man R immer auf NOT S setzt. Siehe oben Data-FlipFlop.

4.6.3 Was versteht man unter dem «Prellen» eines Schalters und wie kann es umgangen werden?

Siehe oben

4.6.4 Was sind die besonderen Eigenschaften des JK-MS-FF?

taktflankengesteuert, kein verbotener Zustand, 2 Taktflanken um Information an die Ausgänge weiter zu geben.

4.6.5 Wie wird beim JK-MS-FF der verbotene Zustand verhindert?

Die Ausgänge werden gekreutz in die Eingänge des Master-FF zurückgegeben.

4.6.6 Wozu wird die Taktleitung zwischen Master und Slave invertiert und was bewirkt man damit?

Für jeden Zustand des Eingangs T wird immer nur entweder Master oder Slave aktiviert! Damit werden die Ausgänge und Eingänge trotz Rückkoppelung dennoch unabhängig voneinander.

4.6.7 Was ist der Unterschied zwischen Taktzustands- und Taktflankensteuerung?

Siehe oben

4.6.8 Um welche Art von Taktsteuerung würde es sich handeln, falls die Negierung in der Taktleitung zwischen Master- und Slave-FF nicht vorhanden wäre?

Taktzustandssteuerung

5 Schieben, Multiplizieren, Rotieren

Man kann Daten parallel oder seriell (nacheinander) übertragen. Dazu werden sie in Datenblöcke unterteilt. Um zwischen den Übertragungsarten wechseln zu können gibt es Prallel-Seriell- bzw. Seriell-Parallel-Wandler.

5.1 4-Bit-Schieberegister

Einfacher Seriell-Parallel-Wandler. Besteht aus 4 JK-MS-FF. Jump und Kill Eingang werden ähnlich wie beim Data-FF zusammengelegt (K = NOT J). Die Ausgänge der einzelnen FF werden an den nächsten FF weitergegeben und zudem parallel dazu abgezweigt. Nach jeder positven Flanke wird so der Zustand des einen FF an das nächste weiter gegeben. Wenn das Schieberegister so gefüllt wurde können immer 4 Zustände gleichzeitig gelesen werden.

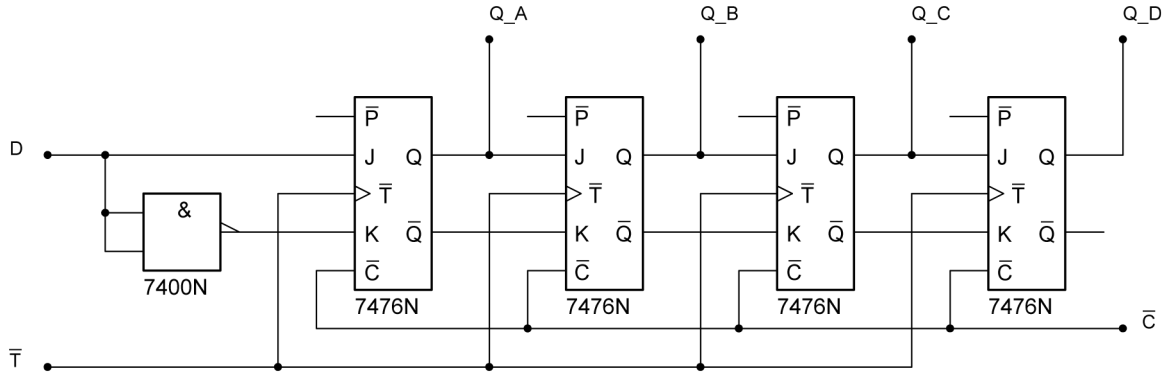

Abbildung 5.1: 4-Bit-Schieberegister

Außerdem kann man durch schieben eines Zustands diesen entsprechend der Leserichtung mit 2 mulitplizieren bzw. dividieren.

5.2 4-Bit-Rotationsregister

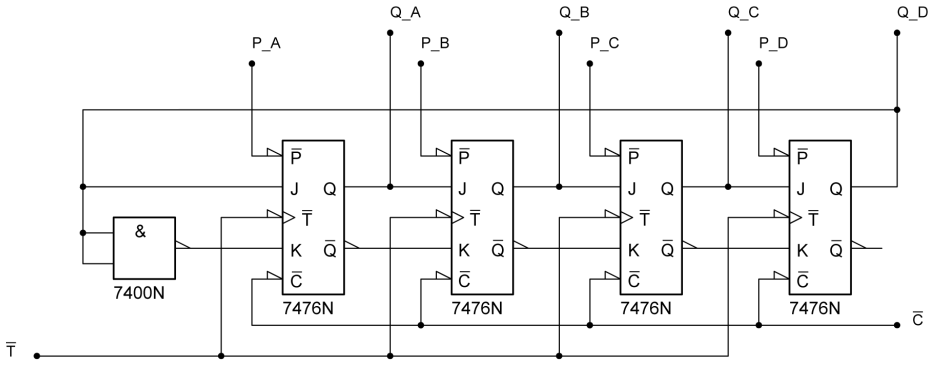

Abbildung 5.2: 4-Bit-Rotationsregister

Ähnlich dem Schieberegister, nur dass der letzte Ausgang mit dem ersten Eingang rückgekoppelt ist. Die eigentlichen Eingänge werden über parallele Preset Eingänge P realisiert. Durch die Rückkoppelung wird also der letzte Zustand ständig wieder der Anordnung zugeführt.

5.3 Fragen

5.3.1 Wozu lässt sich das Schieberegister verwenden?

Zum multiplizieren bzw. dividieren mit k-Potenzen von 2^k.

5.3.2 Welches Problem kann auftreten, wenn das Taktsignal über einen gewöhnlichen Schalter eingegeben wird.

Prellen, siehe oben.

5.3.3 Lässt sich das Rotationsregister auch ohne die NOT-Verknüpfung am ersten JK-MS-FF verwirklichen?

Ja, indem man anstatt Q den $\overline{Q}$ Ausgang benutzt.

6 Zähler

6.1 4-Bit-Asynchronzähler

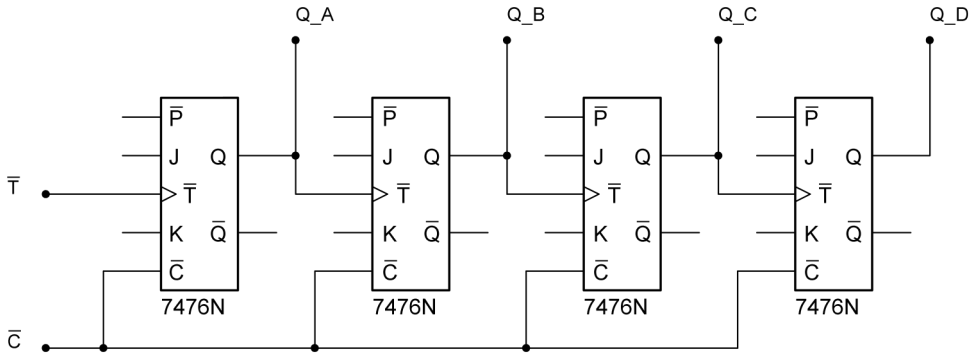

Abbildung 6.1: 4-Bit-Asynchronzähler

Besteht aus mehrern JK-MS-FF. Jedoch werden die Takteingänge mit dem Ausgang des vorherigen kurzgeschlossen. Jump und Kill liegen bei allen offen (1). Preset ebenfalls. Damit arbeiten alle FlipFlop im Toggle-Betrieb. Mit dem C kann man den Zähler zurücksetzen.

Dies ergibt folgende Funktionstabelle:

Clk	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Tabelle 6.1: Funktionstabelle

Ein JK-MS-FF benötigt 2 Taktflanken um den Zustand umzuschalten. Damit benötigt z.B. das 2. JK-MS-FF bereits 4 äußere Taktflanken um seinen Zustand weiter zu schalten, etc. Sprich der Takt wird an jedem weitern FlipFlop halbiert. Deshalb nennt man den Zähler asynchron.

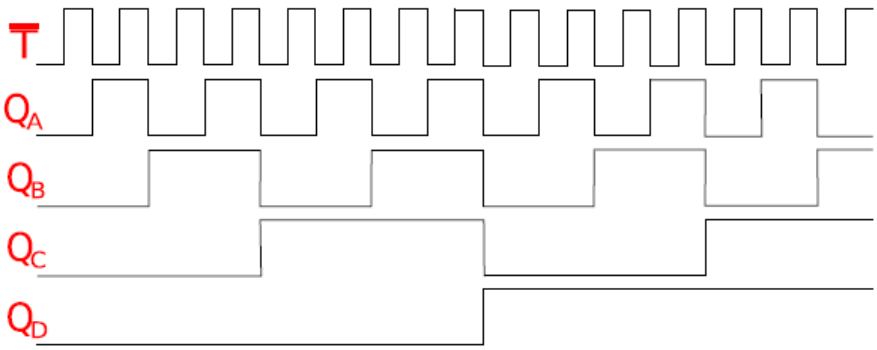

Abbildung 6.2: Taktflankenübersicht

Dabei ergibt sich ein weiteres Problem. Die JK-MS-FF benötigen eine gewisse Zeit um umzuschalten. Mit jeder weiteren Stufe summiert sich die Umschaltzeit der ganzen Anordnung. Größere asynchrone Zähler können also fehlerbehaftet sein.

6.2 Asynchroner Dezimalzähler

Um einen Dezimalzähler aufzubauen nimmt man einen Binärzähler und setzt ihn zurück wenn der Binärzähler bis 10 gezählt hat. Dies ist dann der Fall wenn an Q_B und Q_D gleichzeitig 1 anliegt. Man verbindet also den $\overline{C}$-Eingang über ein NAND-Gatter mit Q_B und Q_D.

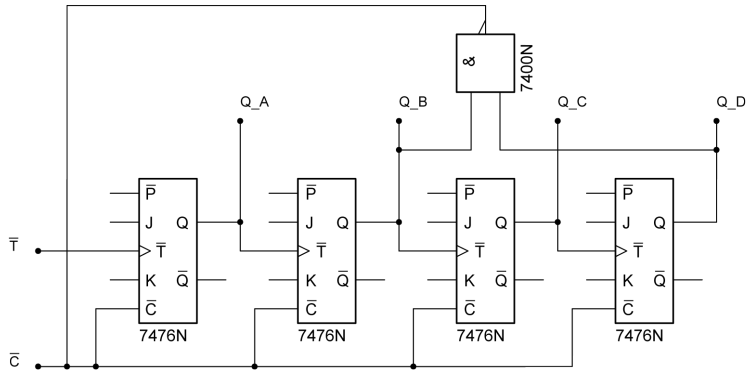

Abbildung 6.3: Asynchroner Dezimalzähler

6.3 4-Bit-Synchronzähler

Analog zum Asynchronzähler besteh dieser Zäher ebenfalls aus 4 JK-MS-FF. Der Unterschied besteht darin, dass alle FF mit dem gleichen Takt betrieben werden (äußerer Taktanschluss wird an alle Takteingänge der FF angeschlossen). D.h. der Zähler arbeitet synchron. Um dennoch die binäre Zählung zu gewährleisten wird nun ein weiterer Mechanismus eingebaut. AND-Gatter sorgen dafür, dass die jeweilige Stufe nur umschaltet, wenn die jeweils vorgeschalteten Stufen bereits auf 1 stehen.

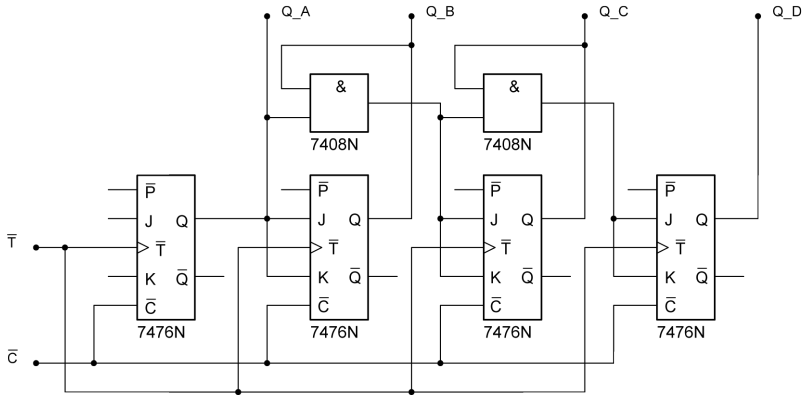

Abbildung 6.4: 4-Bit-Synchronzähler

6.4 Synchroner Dezimalzähler

Entweder wir bauen den Dezimalzähler anlog wie oben auf oder wir benutzen folgenden angeblich eleganteren Schaltplan ...

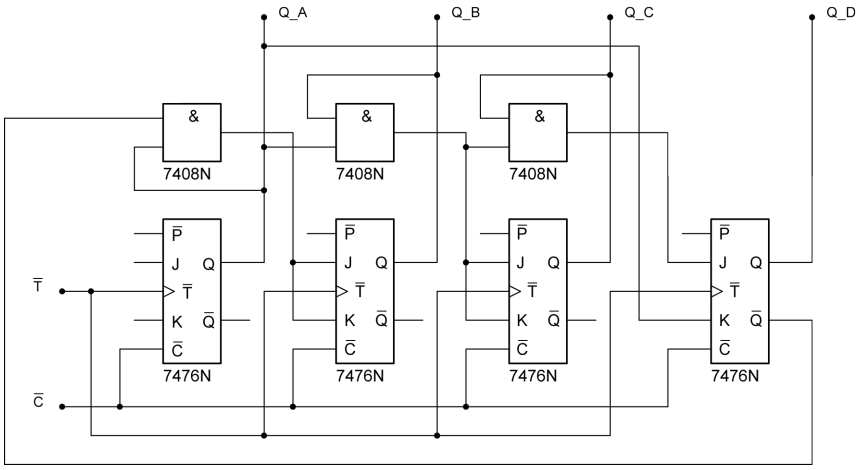

Abbildung 6.5: Synchroner Dezimalzähler

6.5 Fragen

6.5.1 Was ist beim Asynchronzähler asynchron?

Die Takte der einzelnen JK-MS-FF.

6.5.2 Wie könnte man den Asynchronzähler rückwärts zählen lassen (angenommen vor Eingabe des Taktsignals seien alle FFs über P_A ... P_D auf 1 gesetzt worden)?

Wenn der Zustand 1111 erreicht ist zählt der Zähler automatisch rückwärts.

6.5.3 Wozu wird das NAND beim asynchronen Dezimalzähler benötigt?

Falls der Zustand 1010 binär bzw 10 dezimal ist wird so mit Hilfe der Clear-Eingänge der Zähler zurückgesetzt und er beginnt von vorn zu zählen.

6.5.4 Wie könnte man aus dem asynchronen Zähler einen Oktalzähler machen?

Anstatt bei 1010 zurückzusetzen wird bei 0100 zurückgesetzt. Man benötigt also nur ein NOT-Gatter am Q_C Ausgang, und leitet das negierte Signal an alle Clear Eingänge zurück.

6.5.5 Wozu benötigt man die ANDs beim synchronen 4-Bit-Zähler?

Mit ihnen wird gewährleistet, dass die jeweiligen FlipFlop nur im richtigen Moment umschalten. Sprich damit höhre Bits, davon wissen ob vor ihnen bereits 1 1 steht, sodass sie im nächsten Takt von 0 auf 1 schalten können.

6.5.6 Wie viele ANDs bräuchte man für einen synchronen 5-Bit-Zähler und wo würde man die/das zusätzliche einbauen?

Man braucht ein weiteres AND-Gatter und baut dies am 5ten Bit ein, sodass das 5te Bit nur hochzählt wenn vor ihm bereits 1111 gilt. Sprich die Eingänge des AND-Gatter sind das 3te und 4te Bit. Der Ausgang geht auf den Jump/Kill-Eingang des 5ten Bits.

7 Digital-Analog-Wandlung

7.1 Wandler

Man benutz einen Zähler, schließt an die Ausgänge Wiederstände so an, dass von Ausgang zu Ausgang der fließende Strom verdoppelt wird:

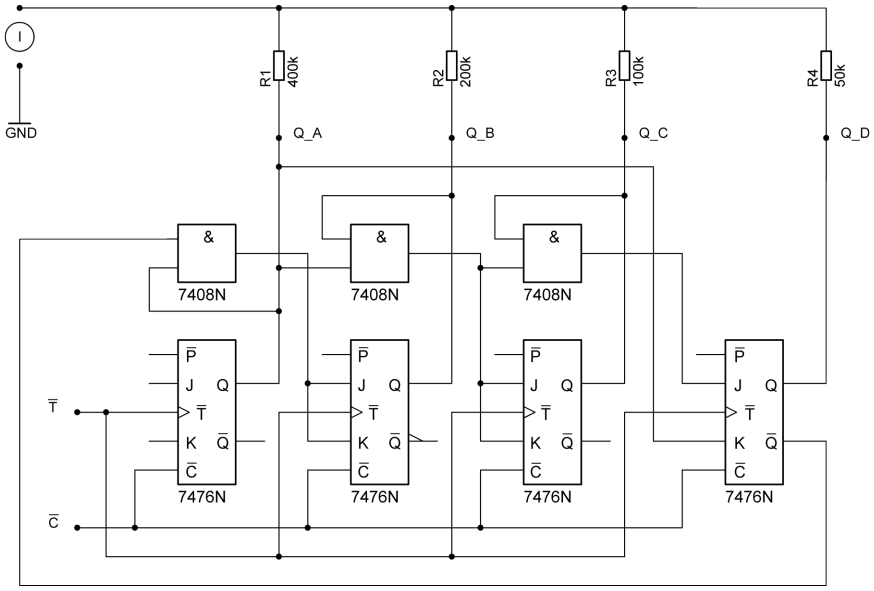

Abbildung 7.1: Digital-Analog-Wandler

7.2 Fragen

7.2.1 Warum können wir nicht einfach alle Ausgänge Q_A ... Q_D verbinden und dann die Ausgangsspannung messen.

Die Ausgänge sind paralell geschaltet und würden somit alle auf dem gleichen Potential liegen. Es würde sich nie etwas verändern.

7.2.2 Addieren sich die Ströme der einzelnen Stufen oder ihre Spannungen?

Die Ströme natürlich.

7.2.3 An welche LED muss der größte Widerstand angeschlossen werden?

die mit dem kleinsten Bit

7.2.4 Wieso müssen die Widerstände von einer zur nächsten Stufe immer halbiert werden?

Der Zähler ist im Prinzip immernoch binär. Damit diese Information auf die Ströme übergeht müssen die Widerstände so angelegt werden, dass die Wertigkeiten der Bits durch die Ströme repräsentiert werden können. Daher werden die Widerstände halbiert um die Ströme von Stufe zu Stufe zu verdoppeln.