Physikpraktikum für Vorgerückte: Digital-Elektronik

Der programmierbare Logikprozessor

Christian Walther ‹cwalther@gmx.ch›

November 2002 – Januar 2003

Inhaltsverzeichnis

1 Einleitung

Beim Versuch «Digital-Elektronik» im Physikpraktikum für Vorgerückte ist man in der privilegierten Lage, sich die Aufgaben, die es zu lösen gilt, selber stellen zu können. Herausgefordert durch die Möglichkeit, hier etwas zu machen, was nicht in der Anleitung steht (und natürlich auch aus Freude, zum eigentlich ersten Mal an der ETH so richtig basteln zu können), habe ich mich an die Konstruktion eines programmierbaren Logikprozessors (siehe «Aufgabenstellung» für die Definition desselben) gemacht, und mich auch nicht gescheut, viel mehr Zeit zu investieren, als eigentlich für das Praktikum vorgesehen wäre (auch angesichts der Tatsache, dass mein Werk als doppelter Versuch zählte). Dieser Artikel soll sowohl die Dokumentation meiner Arbeit sein als auch eine Anleitung für Leute, die etwas ähnliches bauen wollen. Wer dies plant, sei sich aber bewusst, dass die Arbeit umso langweiliger wird, je mehr vorgekaute Anleitung man liest – es sei jedem selber überlassen, an welchem Punkt er selber zu denken beginnen will.

Dieses Dokument und weiteres Material dazu (im Text erwähnt) ist bis auf weiteres erhältlich unter ‹http://www.n.ethz.ch/student/walthec/vp/digital/›, später auf Anfrage bei ‹cwalther@gmx.ch›. Unter dieser Adresse sind auch Rückmeldungen immer willkommen!

2 Aufgabenstellung

Baue einen programmierbaren Logikprozessor und schreibe dafür ein Programm, das drei Bits addieren kann!

Der programmierbare Logikprozessor, als einfaches Modell eines Computers, soll ein Gerät sein, das über einige 1-Bit-Speicherplätze verfügt (in Analogie zum richtigen Computerprozessor als Register bezeichnet) und ein Programm ausführen kann, das aus Schritten der Form «nimm den Inhalt von Register Nr. x und Register Nr. y, NANDe sie und speichere das Resultat in Register Nr. z» besteht. Die Operation NAND wurde hier gewählt, weil sich jede beliebige Logikfunktion aus NANDs zusammensetzen lässt. Man könnte stattdessen auch eine andere elementare Operation verwenden, z.B. das NOR.

Das Addieren von 3 Bits wird auch als Halbaddierer bezeichnet, weil es als Untereinheit eines Volladdierers verwendet wird, der beliebige Binärzahlen addieren kann: Binärzahlen werden wie beim schriftlichen Addieren von Dezimalzahlen stellenweise addiert, beginnend bei der niedrigsten Stelle, wobei man für jede Stelle die Stelle des ersten Summanden, die Stelle des zweiten Summanden und den Übertrag von der vorhergehenden Stelle, also 3 Bits, addieren muss.

3 Das Programm

Um festzulegen, wieviele Register unser Logikprozessor haben muss und wieviele Programmschritte er speichern können muss, beginnen wir damit, zu untersuchen, wieviele denn nötig sind, um die Programmaufgabe zu erfüllen.

In diesem Teil wird viel Boolesche Algebra vorkommen, dabei werde ich die Notation «AB» für «A und B», «$A + B$» für «A oder B» und «$\overline{A}$» für «nicht A» verwen-

den. Sie hat den Vorteil, dass das Distributivgesetz ($A(B+C) = AB + AC$) und die Operatorpräzedenz («und» vor «oder») dieselbe Form haben wie in den geläufigen Zahlenmengen. Ausserdem gelten folgende Regeln, wie man sich leicht anhand der Wahrheitstabellen klar macht: $\overline{A} + \overline{B} = \overline{AB}$ und $\overline{A}\,\overline{B} = \overline{A+B}$ (formal: «Verbinden oder Aufspalten von Querstrichen ändert das Operationszeichen darunter»).

3.1 Wahrheitstabelle und Graphen

Ein Halbaddierer, der die Summandenbits A, B und D zu einem Summenbit S und einem Übertragsbit (Carry) C addiert, soll die Wahrheitstabelle 1 erfüllen, oder, darge-

A	B	D	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabelle 1: Wahrheitstabelle eines Halbaddierers

stellt als Graphen der Funktionen $S, C : \{0,1\}^3 \to \{0,1\}$, Abbildung 1.

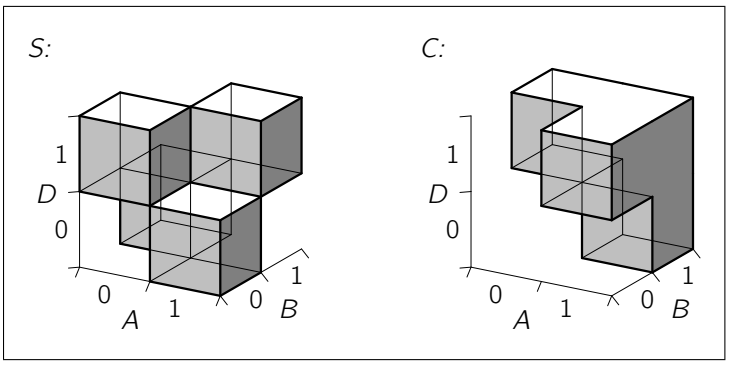

Abbildung 1: Funktionsgraphen des Halbaddierers (ausgefüllte Würfel bedeuten 1, leere 0)

3.2 NAND-Logik

Aus den Graphen können wir jetzt die Booleschen Ausdrücke für S und C lesen und sie durch Faktorisieren, Einfügen von 0-Summanden, Einfügen von doppelten Querstrichen (zweimal invertieren ändert gar nichts) und Anwenden der Querstrich-Verbinde-Regeln in NAND-Logik umwandeln, die dadurch charakterisiert ist, dass sie nur Multiplikationen enthält und über jeder Multiplikation ein Querstrich steht (X NAND $Y = \overline{XY}$). Ausserdem sind einzelne Querstriche als Abkürzung für NAND-mit-sich-selbst zugelassen: $\overline{X} = \overline{XX}$. Beim Umformen soll darauf geachtet werden, dass möglichst

viele Zwischenresultate wiederverwendet werden können, um am Ende mit möglichst wenigen NANDs auszukommen.

Summe:

$$\begin{aligned} S &= ABD + A\overline{B}\,\overline{D} + \overline{A}\,\overline{B}D + \overline{A}B\overline{D} \\ &= A(BD + \overline{B}\,\overline{D}) + \overline{A}(\overline{B}D + B\overline{D}) \\ &= AF + \overline{A}\,\overline{F} \\ &= \overline{\overline{AF}} + \overline{\overline{A}\,\overline{F}} \\ &= \overline{\overline{AF}\,\overline{\overline{A}\,\overline{F}}} \end{aligned}$$

Nebenrechnung für die Abkürzung F:

$$BD + \overline{B}\,\overline{D} = \overline{\overline{BD}} + \overline{\overline{B}\,\overline{D}} = \overline{\overline{BD}\,\overline{\overline{B}\,\overline{D}}} =: F$$

$$\begin{aligned} \overline{B}D + B\overline{D} &= \overline{B}D + B\overline{D} + \underbrace{\overline{B}B}_{0} + \underbrace{D\overline{D}}_{0} = (\overline{B} + \overline{D})(B + D) \\ &= \overline{\overline{(\overline{B} + \overline{D})}\,\overline{(B + D)}} = \overline{\overline{B}\,\overline{D}\,\overline{B}\,\overline{D}} = \overline{\overline{BD}\,\overline{B}\,\overline{D}} = \overline{F} \end{aligned}$$

Übertrag:

$$\begin{aligned} C &= AB + AD + BD \\ &= A(B + D) + BD \\ &= A\overline{\overline{B}\,\overline{D}} + BD \\ &= \overline{A\overline{\overline{B}\,\overline{D}}\,\overline{BD}} \end{aligned}$$

Diese Ausdrücke lassen sich nun in Form eines Schaltdiagramms darstellen, aus dem durch Identifikation der vertikalen Leitungen mit den Registern direkt ein Programm für den Logikprozessor hervorgeht (Abbildung 2).

3.3 Optimierung

Offensichtlich geht dieses Programm nicht sehr effizient mit den Registern um: jedes wird genau für einen Wert verwendet und auch nachdem dieser Wert nicht mehr gebraucht wird nicht für andere Werte wiederverwendet. Um dies zu verbessern, mache man sich klar, dass am Programm folgende Änderungen vorgenommen werden können, ohne seine Funktion zu verändern:

- Zwei vertikal benachbarte Gatter können vertauscht werden, wenn keines von beiden seinen Output auf einem Register hat, das vom anderen als Input verwendet wird.

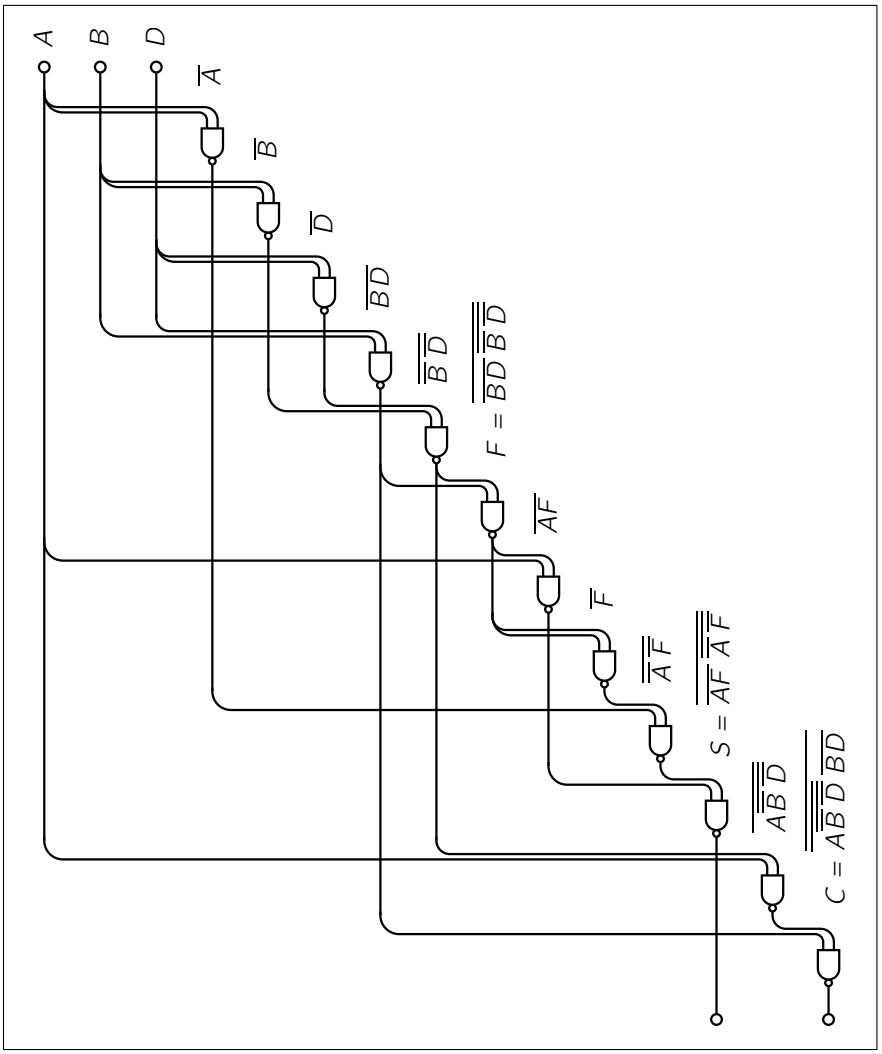

Abbildung 2: Rohfassung des Halbaddiererprogramms

- Ein Gatter kann mitsamt der ganzen von ihm ausgehenden vertikalen Leitung horizontal an einen anderen Ort verschoben werden, sofern dort Platz dafür ist (d.h. keine anderen Gatter und Leitungen dort sind).

Da sich eine grössere Anzahl solcher Änderungen ohne instantanes visuelles Feedback kaum durchführen lässt, habe ich eigens dafür ein Java-Applet geschrieben, wo sich die Gatter mit der Maus herumschieben lassen: ‹http://www.n.ethz.ch/student/walthec/vp/digital/editor/›. Es ist seither noch um einige zusätzliche Funktionen erweitert worden; wer es zur vollausgestatteten Logikprozessor-Programmierumgebung ausbauen möchte, der implementiere die Fähigkeiten, solche Optimierungen automatisch vorzunehmen sowie für eine gegebene Wahrheitstabelle ein optimales Programm zu finden.

Abbildung 3 zeigt eine durch Gatter-Herumschieben optimierte Version des Halbaddiererprogramms. Angenommen, dass sich der Halbaddierer nicht durch einen anderen Ansatz bei der Programmaufstellung noch einfacher realisieren lässt, braucht unser Logikprozessor also mindestens 4 Register und Programmspeicherplatz für 12 Programmschritte. In der Abbildung ist ausserdem die binäre Kodierung des Programms angegeben, die später in den Programmspeicher des Logikprozessors eingegeben wird: Jeder Schritt ist bestimmt durch ein Tripel von Binärzahlen (*Inputregister 1*, *Inputregister 2*, *Outputregister*).

Da ich zu diesem Zeitpunkt noch mit einer weniger gut optimierten Fassung des Programms arbeitete, bin ich bei der Dimensionierung der Schaltung ausgegangen von 7 Registern und 14 Programmschritten. Es wird sich aber herausstellen, dass eine Beschränkung auf 4 Register und 12 Programmschritte abgesehen von der Einsparung der Register keine wesentliche Vereinfachung bringt.

4 Das Design der Schaltung

4.1 Grobkonzept

Grob betrachtet, muss der Logikprozessor aus folgenden Teilen bestehen:

- mehreren Flip-Flops als Register
- einem Programmspeicher mit Programmzähler, der einen Programmschritt nach dem anderen von sich geben kann
- einem oder zwei Multiplexern, die den Wert eines (vom Programm bestimmten) Registers auswählen
- einem NAND-Gatter als Rechenwerk
- einem Demultiplexer, der das Resultat zurück in ein (wiederum vom Programm bestimmtes) Register führt
- einem Taktgeber, der abwechslungsweise einen Programmschritt (mit eventuellen Unterschritten) ausführt und den Programmzähler inkrementiert.

Hier ergibt sich eine erste Design-Entscheidung: Entweder liest man die beiden Operanden gleichzeitig aus ihren Registern, braucht dafür also zwei Multiplexer, oder man

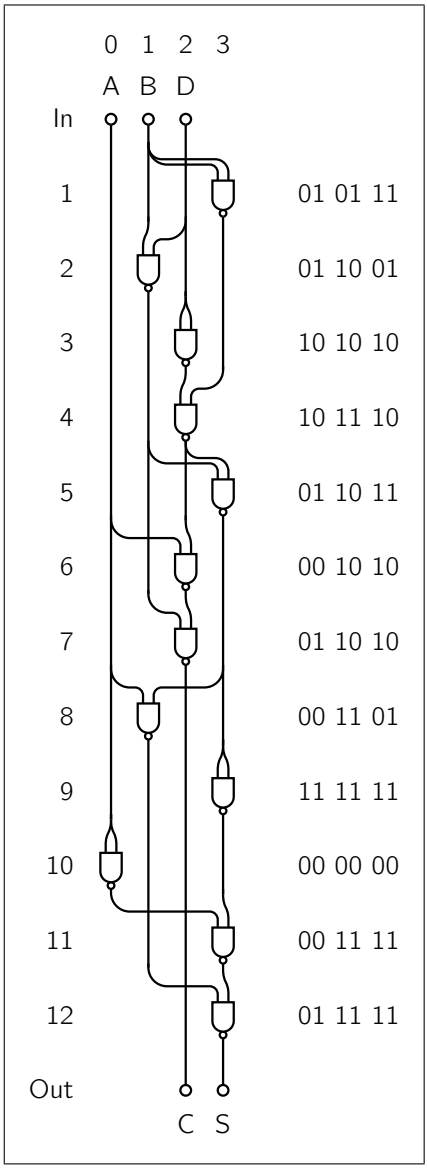

Abbildung 3: Optimiertes Halbaddiererprogramm und dessen binäre Kodierung

liest sie nacheinander, was nur einen Multiplexer braucht, aber dafür noch zwei Zwischenspeicherplätze (Operandenregister). Ich habe mich für die letztere Variante entschieden (hauptsächlich, weil sie gut zu meiner Implementation des Programmspeichers passen wird).

Dieses Grobkonzept ist als Blockschaltbild in Abbildung 4 dargestellt.

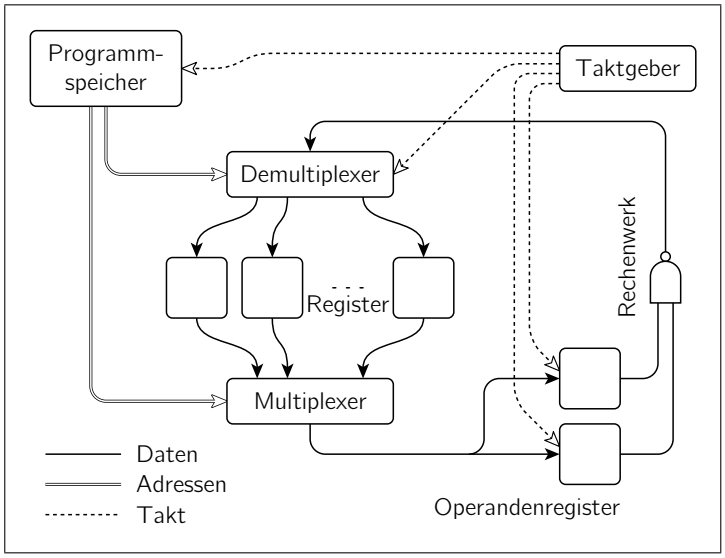

Abbildung 4: Grobkonzept des Logikprozessors

Mehr oder weniger willkürlich habe ich beschlossen, meine Schaltung in CMOS-Logik zu bauen (hauptsächlich, weil bei den im Praktikum ausgehängten Lagerlisten die CMOS-Liste länger war als die TTL-Liste). Durch gründliches Studium des Datenbuchs «Motorola CMOS Logic Data» (Rev. 2, 1990) versuchte ich nun herauszufinden, aus welchen Bausteinen man eine solche Schaltung aufbauen könnte.

4.2 Programmspeicher

Wie macht man einen Programmspeicher, der die oben definierte Funktion erfüllt und in den ausserdem das Programm leicht hineinzukriegen ist? Diese Frage stellte sich als Knacknuss heraus. Meine erste Idee war, eine steckbare Diodenmatrix zu bauen (Abbildung 5). Dies scheidet jedoch aufgrund der schieren Anzahl nötiger Diodensteckplätze als unpraktikabel aus: $3 \cdot \lceil \log_2(7) \rceil \cdot 14 = 126$ für 7 Register und 14 Schritte, für 4 Register und 12 Schritte sind es immerhin noch $3 \cdot \lceil \log_2(4) \rceil \cdot 12 = 72$.

Verschiedene Ansätze mit RAMs scheiterten an der mangelnden Verfügbarkeit von Bausteinen mit passenden Spezifikationen.

Die zum Erfolg führende Idee war schliesslich, das gesamte Programm seriell in einem langen Schieberegister zu speichern. Dadurch kann erst noch der Programmzähler eingespart werden, da verschiedene Programmschritte nicht mehr an verschiedenen Speicherplätzen gelesen werden müssen, sondern einfach der entsprechende Programmschritt an den Leseort geschoben wird. Als Schieberegister mit genügender Kapazität bietet sich der Baustein 4562 «128 bit static shift register» an. Allerdings

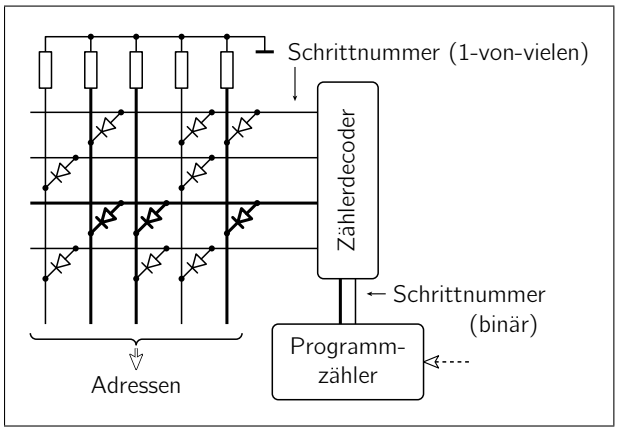

Abbildung 5: Diodenmatrix. Der Programmzähler setzt über den Decoder eine Zeile nach der anderen auf 1, worauf die Spalten auf 1 gehen, wo eine Diode vorhanden ist, oder durch den Pull-Down-Widerstand auf 0 bleiben, wo keine ist. Dioden (anstelle von einfachen Verbindungen) sind nötig, damit der Strom nicht von einer vertikalen Leitung zurück in eine horizontale und von dort in unerwünschte Ausgänge kann.

scheint dieser nirgends mehr lieferbar zu sein, weshalb ich als Ersatz den 4517 «Dual 64 bit static shift register» verwendete. Da wir die Schritte «Operand 1 lesen», «Operand 2 lesen», «Resultat speichern» nacheinander ausführen wollen, müssen wir (für 7 Register) 3 Bits parallel aus dem Schieberegister auslesen können. 4562 oder 4517 haben allerdings nirgends drei aufeinanderfolgende Ausgänge. Deshalb schalten wir einfach noch ein zusätzliches Schieberegister dahinter, das diese hat. Ein solches ist der 4014 «8 bit static shift register».

Die von 4517 und 4014 angebotene Parallel-Einlese-Funktion benötigen wir nicht, weshalb wir beim 4517 die «write enable»-Eingänge und beim 4014 den Parallel-Seriell-Umschalt-Eingang und die Parallel-Eingänge auf 0 legen.

Damit beim Betrieb das aus dem Schieberegister herausgeschobene Programm nicht verloren geht, soll der Ausgang des Schieberegisters wieder mit seinem Eingang verbunden werden, also ein Ringspeicher gebildet werden. Zur Eingabe des Programmes soll ausserdem vom Benutzer wählbar «1» oder «0» in den Speicher geschoben werden können. Dies lässt sich erreichen, indem man den Eingang des Schieberegisters über einen Dreifach-Umschalter, der auf «Ring», «1» oder «0» gestellt werden kann, mit seinem Ausgang (Ring), der (positiven) Betriebsspannung (1) oder der Masse (negativer Pol der Betriebsspannung) (0) verbindet.

$128 + 8 = 136$ Bit haben in diesem Programmspeicher Platz. Das reicht für 15 Programmschritte ($15 \cdot 9 = 135$), aber dann bleibt ein Bit übrig. Um den Programmspeicher nach der vollständigen Ausführung eines Programmes wieder im Ausgangszustand zu haben, wäre es bequemer, genau 135 Bit zu haben. Der Ringspeicher lässt sich auf 135 Bit verkürzen, indem man Ausgang 7 des 4014 statt Ausgang 8 an den Eingang zurückkoppelt, aber dann können nicht mehr die letzten 3 Bits des Ringspeichers ausgelesen werden, weil beim 4014 Ausgang 5 nicht nach aussen geführt ist. Dies ist aber kein Hindernis für die Funktion des Speichers – man muss sich einfach bewusst sein, dass gewissermassen das erste und das letzte Bit des 136-Bit-Speichers identifiziert

wurden (Abbildung 6).

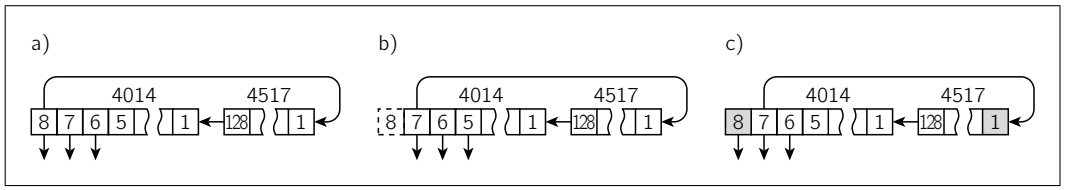

Abbildung 6: a) Naheliegende, aber 1 Bit zu grosse Lösung b) Ideale, aber nicht realisierbare Lösung c) Realisierbare Lösung mit identifiziertem erstem und letztem Bit

4.3 Register, Multiplexer und Demultiplexer

Ein auf den ersten Blick vielversprechendes Bauteil schien der 4599 «8 bit addressable latch», mit dem 8 Register, Multiplexer und Demultiplexer auf einem IC realisiert wären. Allerdings stellt sich damit die Schwierigkeit, wie die Eingangsdaten für das Programm in die Register zu kriegen sind. Der 4599 hat nämlich nur einen einzigen Dateneingang, der jeweils zu dem am Adresseingang adressierten Register führt. Dies würde die Dateneingabe eher benutzerunfreundlich gestalten, da für jedes Datenbit zusätzlich seine Adresse eingegeben werden müsste. Ausserdem erforderte es einen grösseren externen Schaltungsaufwand, die Adressen einerseits vom Programmspeicher und andererseits von der Dateneingabe lesen zu können. Deshalb verwarf ich diese Idee schliesslich.

Anforderung an die Register ist, dass sie auf einen Taktpuls ein an ihrem Eingang anliegendes Bit in ihr Inneres befördern und von da an ständig an ihrem Ausgang anbieten. Dies wird erfüllt durch das «D-Flip-Flop», von welchem man zwei Stück auf einem IC erhält mit dem 4013 «Dual D flip-flop». Ausserdem besitzen diese einen Set- und einen Reset-Eingang, die den Zustand beim Anlegen einer 1 sofort auf 1 resp. 0 setzen, was eine relativ einfache Dateneingabe erlaubt.

Als Multiplexer bietet sich der 4512 «8 channel data selector» an, der genau das tut, was man von ihm erwartet: Er nimmt, je nach an seinen 3 Adresseingängen anliegender Adresse, einen von 8 Eingängen und legt dessen Wert an seinen Ausgang. Seine «inhibit»- und «disable»-Funktionen werden nicht benötigt, weshalb wir die entsprechenden Eingänge konstant auf 0 legen.

Nach einem (digitalen) Demultiplexer durchforstete ich das Datenbuch zunächst vergeblich, bis ich auf die Idee kam, dass man dafür den 4028 «BCD-to-decimal/binary-to-octal decoder» verwenden kann. Dieser hat vier Eingänge (A, B, C, D) und 10 Ausgänge (Q0–Q9) und setzt jeweils genau den Ausgang auf 1, dessen Nummer als Binärzahl an A–D anliegt (mit niedrigstem Bit A und höchstem Bit D), und alle anderen auf 0; oder alle auf 0, wenn diese Zahl grösser als 9 ist. Bei Betrachtung seiner Wahrheitstabelle stellt man fest, dass diese fast dieselbe ist wie die eines 1-zu-8-Demultiplexers: Wenn die Ausgänge 8 und 9 ignoriert werden, liegt an dem durch die Binärzahl ABC adressierten Ausgang immer genau das Inverse des Eingangs D, an allen anderen 0. Durch Anhängen eines Inverters vor den Eingang D erhält man also genau die gewünschte Demultiplexerfunktion.

Zur Aufgabe des Demultiplexers überlege man sich, dass es nichts bringt, wenn der Demultiplexer die Daten demultiplext (auch wenn es im Blockschaltbild des Grobkon-

zepts, Abbildung 4, so aussieht): Wenn dann an alle Register ein Taktpuls geht, wird zwar das adressierte Register überschrieben (mit den gewünschten Daten), aber alle anderen auch: mit den Nullen, die aus den unadressierten Ausgängen des Demultiplexers kommen. Der Demultiplexer muss also den Takt demultiplexen, so dass nur an das adressierte Register ein Taktpuls geht. Dann können dafür die Daten an alle Register gleichzeitig angelegt werden – mit den unadressierten Registern geschieht mangels Taktpuls nichts. Damit ist auch festgelegt, dass der Grundzustand der Taktleitungen 0 ist, und ein Taktpuls aus einer kurzfristigen 1 besteht.

Da Multiplexer und Demultiplexer beide 8 Register unterstützen, bedeutet es keinen grossen Aufwand, statt der für den Halbaddierer unbedingt nötigen 4 (oder nach altem Programm 7) Register gleich deren 8 einzubauen, um die Schaltung auch für kompliziertere Aufgaben auszurüsten.

4.4 Operandenregister und Rechenwerk

Operandenregister und Rechenwerk realisieren sich auf naheliegende Weise: Als Operandenregister werden zwei weitere D-Flip-Flops aus einem 4013 verwendet, an deren Eingänge das vom Multiplexer kommende Datenbit gelegt wird. Als Rechenwerk dient ein NAND-Gatter mit zwei Eingängen, wie man es zu viert auf dem 4011 «Quad 2-input NAND» findet, das die Ausgänge der Operandenregister verknüpft und das Resultat zu den Eingängen der Register liefert.

4.5 Taktverteiler

Schliesslich bleibt noch die Steuerzentrale zu planen, die den Ablauf innerhalb eines einzelnen Programmschrittes kontrolliert:

Zuerst muss im Programmspeicher die Adresse des ersten Operanden an den Leseort geschoben werden: 3 Taktpulse zum Programmspeicher. Jetzt liegt die Adresse am Multiplexer und damit der Inhalt des adressierten Registers an den Eingängen der Operandenregister. Dann wird der erste Operand in sein Operandenregister gelesen: ein Taktpuls zum ersten Operandenregister. Adresse des zweiten Operanden an die Lesestelle schieben: 3 Taktpulse zum Programmspeicher. Den zweiten Operanden in sein Operandenregister lesen: ein Taktpuls zum zweiten Operandenregister. Jetzt sind beide Operanden in ihren Operandenregistern, werden vom Rechenwerk NAND-verknüpft, und das Resultat liegt an den Eingängen der Register. Die Destinationsadresse für das Resultat wird an die Lesestelle geschoben: 3 Taktpulse zum Programmspeicher. Die Adresse liegt am Demultiplexer. Schliesslich ein Taktpuls zum Demultiplexer, der von diesem an das adressierte Register weitergeleitet wird und dieses veranlasst, das am Eingang anliegende Resultat zu speichern.

Selbstverständlich ist die Festlegung der Grenze zwischen zwei Perioden willkürlich. Man könnte zum Beispiel ebensogut mit dem Lesen des 1. Operanden beginnen und das Weiterschieben des Programmes an den Schluss verschieben. Allerdings hat sich gezeigt, dass sich die beschriebene Variante am effizientesten realisieren lässt.

Gefragt ist also eine Schaltung mit vier Ausgängen, die von 12 Taktpulsen den 4. zum zweiten Ausgang (Operand 1), den 8. zum dritten Ausgang (Operand 2), den 12. zum vierten Ausgang (Demultiplexer) und alle anderen zum ersten Ausgang (Programmspeicher) verteilt und danach wieder in ihrem Ausgangszustand ist.

Um über seine 12 Zustände Buch zu führen braucht der Taktverteiler einen 4-Bit-Zähler. Diesen gibts zum Beispiel in Form des 4040 «12 bit binary counter», wenn man einfach die Ausgänge 5 bis 12 ignoriert. Sobald der Zähler auf 12 springen will (was der 13. Zustand ist, da beim Zählen bei 0 begonnen wird), muss er auf 0 zurückgesetzt werden. 12 ist genau dann, wenn die Ausgänge 3 und 4 erstmals beide 1 sind, also kann dies gelöst werden, indem die Ausgänge 3 und 4 über ein UND-Gatter an den Reset-Eingang geführt werden. Dass bei dieser Lösung der Zähler wegen der endlichen Schaltgeschwindigkeit für kurze Zeit tatsächlich auf 12 steht, stört bei unserer Anwendung nicht.

Vor die vier Ausgänge des Taktverteilers kommen als «Tore» UND-Gatter, die den zu verteilenden Takt verknüpfen mit den Bedingungen, bei welchem Zählerstand ein Taktpuls an diesen Ausgang soll: der Taktpuls kommt nur dann durch, wenn alle Bedingungen erfüllt sind. Wie diese Bedingungen lauten, liest man am besten aus einem Diagramm, das den zeitlichen Verlauf aller Signale über eine Periode (Programmschritt) angibt: Abbildung 7. So ist zum Beispiel das Zeitfenster für den Programm-

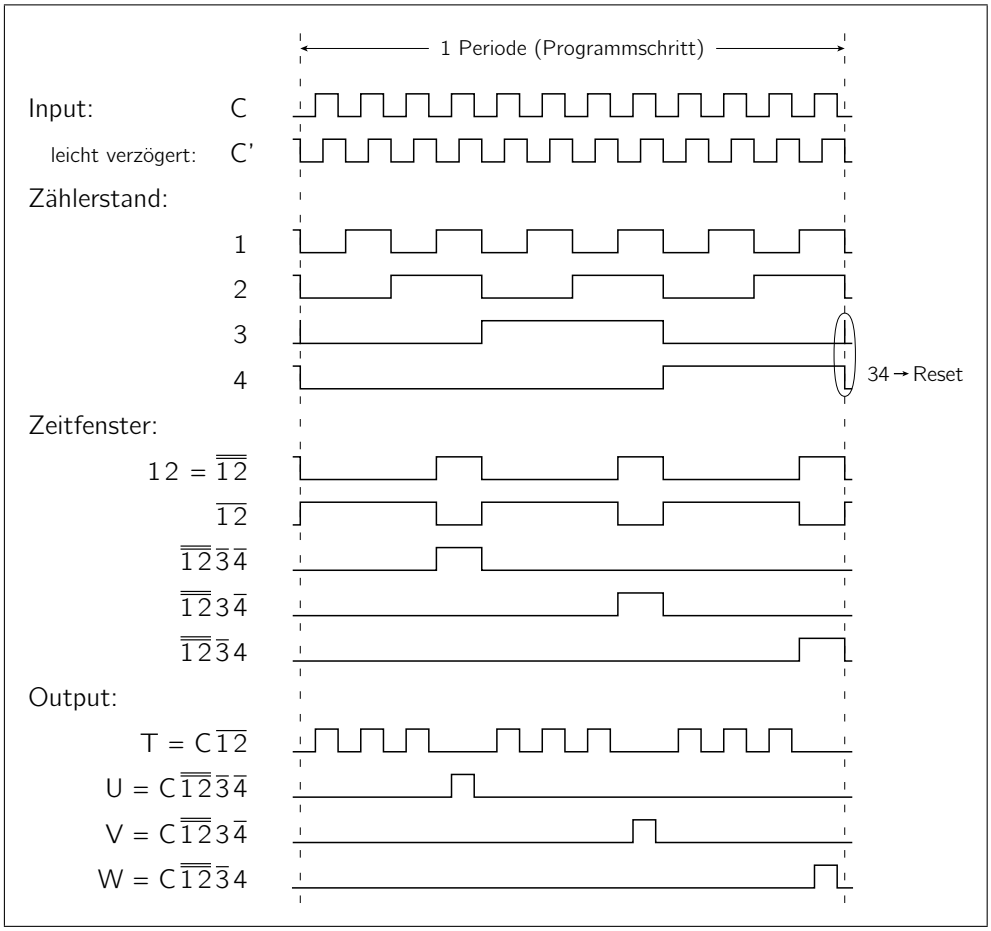

Abbildung 7: Zeitlicher Verlauf der Signale des Taktverteilers

schiebetakt, mit T bezeichnet, genau dann zu öffnen, wenn nicht beide der Zählerausgänge 1 und 2 auf 1 stehen, also in Boolescher Schreibweise bei $\overline{1\,2}$.

Wenn der Zähler springt, wird es aufgrund der endlichen Schaltgeschwindigkeit so sein, dass nicht alle seine Ausgänge gleichzeitig springen. Ausserdem wird am Ende der Periode der oben beschriebene Fall eintreten, dass der Zähler kurzzeitig auf 12 steht. Um zu verhindern, dass dies zu unvorhersehbaren Pulsen an den Taktausgängen führt (da die den Takt empfangenden Bauteile auf die Flanken des Taktes reagieren, können auch sehr kurze Pulse eine Funktion auslösen), soll zu diesen Zeiten der zu verteilende Takt C auf 0 stehen. Da der Zähler (anders als die Schieberegister des Programmspeichers und die Flip-Flops der Register und Operandenregister) bei der fallenden Flanke seines Takteingangs springt, erreicht man dies, indem man den Zähler-steuernden Takt C' eine leicht verzögerte Variante des zu verteilenden Taktes C macht (siehe auch Abbildung 7). Diese Verzögerung lässt sich erreichen, indem man einige Gatter so hintereinanderschaltet, dass das Ausgangssignal gleich dem Eingangssignal ist, zum Beispiel eine gerade Anzahl von Invertern. Eine andere Möglichkeit wird sich weiter unten zeigen.

Als Quelle des Taktsignals soll zunächst ein Taster dienen, der von Hand betätigt wird, damit man dem Logikprozessor beim Rechnen zusehen kann. Das Problem dabei ist, dass Taster im Allgemeinen eine Eigenart haben, die man als «Prellen» bezeichnet: Wegen Unreinheiten der Kontaktflächen und wegen des mechanischen Stossvorgangs wird der Kontakt beim Drücken oder Loslassen nicht sofort endgültig hergestellt oder unterbrochen, sondern ändert sich während einer kurzen Zeit mehrmals unregelmässig. Dies ist üblicherweise kein Problem, wenn es nur um den Endzustand geht, aber in unserer Anwendung ist es wichtig, dass beim Einschalten und Ausschalten genau eine steigende bzw. fallende Flanke des Signals auftritt. Natürlich lässt sich dieses Problem elektronisch lösen, dafür gibt es gebrauchsfertige Entprellschaltungen wie im CMOS-Bereich den 4490 «Hex contact bounce eliminator». Seine Funktion ist im Datenbuch im Detail beschrieben, grob gesehen tut er folgendes: Er hat einen eingebauten Oszillator, der an externer Beschaltung nur noch einen Kondensator braucht. Seine Frequenz ergibt sich aus der Kapazität des Kondensators (ausserdem ist sie von der Betriebsspannung abhängig). Mit Hilfe eines Schieberegisters in geeigneter Beschaltung bewirkt ein einzelner Entpreller, von dem 6 Stück in einem IC vorhanden sind, dass sein Ausgangssignal erst dann dem Eingangssignal folgt, wenn dieses über 4 bis 5 Perioden des Oszillators (je nach momentaner Phase des Oszillators) konstant bleibt. Ausserdem ist jeder Eingang mit Hilfe eines internen Pull-Up-Widerstandes mit der Betriebsspannung verbunden, man braucht also den Eingang nur noch über den Taster mit der Masse zu verbinden. Allerdings wird einem dadurch auch vorgeschrieben, dass bei gedrücktem Taster der Ausgang 0 ist und bei offenem 1. Da die Schieberegister des Programmspeichers und die Flip-Flops der Register und Operandenregister bei der steigenden Flanke des Taktes arbeiten, und wir die Aktionen beim Drücken und nicht beim Loslassen des Tasters haben möchten, schalten wir noch einen Inverter hinter den Entpreller.

Als Nebeneffekt bewirkt der Entpreller eine Verzögerung des Eingangssignals um die 4 bis 5 Oszillatorperioden. Dies lässt sich in unserer Schaltung gerade ausnützen, um die geforderte Verzögerung des Zählertaktes gegenüber dem Eingangstakt zu erreichen.

Alles zusammengesetzt, sieht ein provisorisches Schaltbild des Taktverteilers aus wie in Abbildung 8.

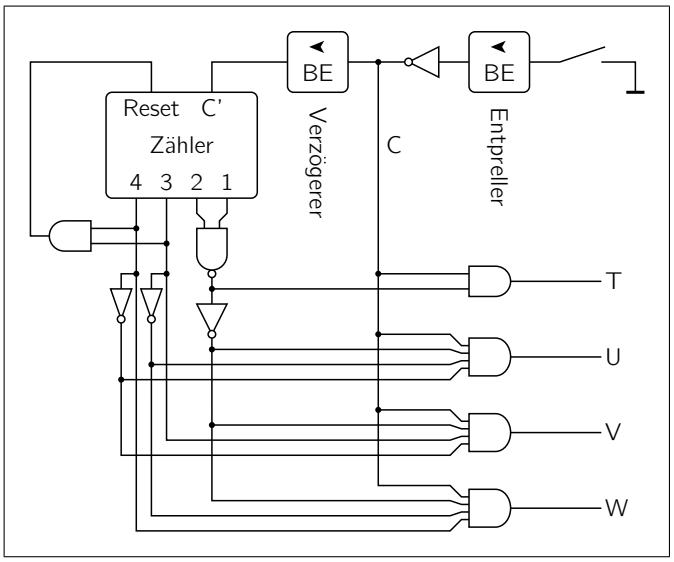

Abbildung 8: Provisorisches Schaltbild des Taktverteilers («BE»: «Bounce Eliminator»)

4.6 Schnittstellen zur Aussenwelt

Bis jetzt kann der Logikprozessor zwar rechnen, ist aber trotzdem nicht zu viel nütze, weil ihm niemand sagen kann, was er rechnen soll, und niemand sieht, was er rechnet. Deshalb sind Schnittstellen zur Aussenwelt, Eingabe- und Ausgabemöglichkeiten, nötig.

Programmeingabe
Die Eingabe des Programmes wurde schon zur Hälfte geplant: Mit dem Dreifach-Umschalter kann ausgewählt werden, ob eine 1 oder eine 0 in den Programmspeicher geschoben werden soll. Was noch fehlt, ist die Vorrichtung, um das Schieben auszuführen. Dies könnte natürlich mit dem Taster für den Rechentakt geschehen, da die meisten Taktpulse von dort an den Programmspeicher gehen, es wäre einfach jeder vierte Tastendruck wirkungslos. Benutzerfreundlicher ist es, einen eigenen Taster dafür vorzusehen. Ausserdem muss damit bei der Programmeingabe weniger peinlich darauf geachtet werden, dass die Lage der Bits im Programmspeicher mit dem Zustand des Taktverteilers synchron ist – eine Abweichung kann auch nachträglich durch Drücken der Programmschiebetaste bei eingeschaltetem Ringbetrieb korrigiert werden.

Der Einbau eines solchen Tasters stellt keine Probleme: Man benötigt dafür einen zusätzlichen Entpreller mit nachgeschaltetem Inverter und kombiniert dessen Signal vor dem Takteingang des Programmspeichers mittels eines ODER-Gatters mit dem Taktsignal vom Taktverteiler.

Dateneingabe
Die Eingabe der zu bearbeitenden Daten in die Register soll über deren Set- und Reset-Eingänge geschehen. Die naheliegende Variante wäre hier, an jedes Register zwei Taster anzuschliessen. 16 Taster brauchen allerdings viel Platz und bringen einen entsprechenden Aufwand beim Einbau. Eleganter wäre es, einen Taster pro Register zu haben,

der dessen Zustand wechselt, aber dies ist bei den D-Flip-Flops nicht ohne zusätzlichen äusseren Aufwand zu realisieren. Was ich schliesslich baute, ist folgendes: Über einen Umschalter wird entweder der Set- oder der Reset-Eingang jedes Registers mit einer gemeinsamen Leitung verbunden, die mit einem Taster an die Betriebsspannung gelegt werden kann. Man stellt also die gewünschten Zustände an den Umschaltern ein und drückt dann auf die Taste, um sie in die Register einzulesen. Um der bei CMOS-Bausteinen gültigen Regel nachzukommen, dass keine Eingänge unbeschaltet gelassen werden dürfen, werden ausserdem alle Set- und Reset-Eingänge über Pull-Down-Widerstände mit der Masse verbunden. Diese Lösung braucht nur einen Taster, und die Umschalter sind recht klein und brauchen wenig Platz. Ein Entpreller ist hier nicht nötig, da es nichts schadet, wenn ein Register mehrmals gesetzt oder zurückgesetzt wird.

Ausgabe
Für die Ausgabe der Resultate und zur generellen Überwachung der Funktion des Logikprozessors sollen jedes Register, die vier Taktleitungen und die drei Adressleitungen (welche gerade den letzten 3 Bits des Programmspeichers entsprechen) ihren Zustand anzeigen. Für die Anzeige eines Bits eignet sich eine Leuchtdiode (LED), die bei 1 leuchtet und bei 0 dunkel bleibt. Gewöhnliche LEDs brauchen aber um mit voller Helligkeit zu leuchten einen Strom von ca. 20 mA, dies ist mehr als ein CMOS-Baustein liefern kann. Man könnte also jede Leuchtdiode über einen Transistor ansteuern, was aber wieder ein zusätzlicher Schaltungsaufwand wäre. Eine einfachere Lösung sind sogenannte Low-Current-LEDs, die bereits bei ungefähr 2.5 mA ihre volle Helligkeit erreichen. Ein kurzer Test hat ergeben, dass bei 10 V Betriebsspannung ein CMOS-Baustein auch bei 5 parallelen Low-Current-LEDs an seinem Ausgang noch einen anderen CMOS-Baustein anzusteuern vermag, also sind diese LEDs für unseren Zweck geeignet.

4.7 Nachträge

Zwei Dinge sind mir erst aufgefallen, als ich die Schaltung schon komplett aufgebaut hatte. Um sie trotzdem in die folgenden Schaltpläne aufnehmen zu können, erwähne ich sie an dieser Stelle:

- Der beim Eingang des Programmspeichers benützte Dreifach-Umschalter hatte die unangenehme Eigenschaft, dass beim Umschalten zwischen zwei Stellungen kurzzeitig die beiden Anschlüsse dieser Stellungen miteinander verbunden wurden. An den Anschlüssen für «1» und «0» liegen aber Betriebsspannung und Masse, das heisst, beim Umschalten zwischen diesen beiden Stellungen wird für kurze Zeit ein Kurzschluss gemacht. Dieser nimmt der ganzen restlichen Schaltung den Strom weg, schaltet sie also effektiv aus, und nach dem Einschalten sind alle Speicherzellen, also die Register und der gesamte Programmspeicher, in einem nicht vorhersehbaren Zustand. Es ist also nicht möglich, ein Programm einzugeben, das nicht aus lauter Einsen oder Nullen besteht.

  Dieses Problem lässt sich beheben, indem der «0»-Anschluss des Umschalters nicht direkt, sondern über einen Widerstand mit der Masse verbunden wird. Dies hat auf die Funktion keinen Einfluss, da am Eingang des CMOS-Schieberegisters

nur ein winziger Strom fliessen muss, der keinen nennenswerten Spannungsabfall am Widerstand verursacht, andererseits wird der Strom beim Kurzschluss stark begrenzt, so dass dem Rest der Schaltung noch genügend Spannung bleibt.

- Beim Eingang des Taktverteilers gibt es einen Verzögerer mit einem Inverter dahinter. Was da am Anfang reingeht, kommt nach einer gewissen Zeit am Ende umgekehrt raus. Was passiert, wenn man in einer solchen Anordnung den Ausgang an den Eingang rückkoppelt? Dann hat man einen Oszillator gebaut. Es muss also nur eine einzige Verbindung in die bestehende Schaltung eingebaut werden, und der Logikprozessor läuft von selber, ohne dass jemand hunderte von Malen auf den Taktknopf drückt.

  Da der Logikprozessor natürlich weiterhin von Hand bedienbar sein soll, machen wir diese Verbindung temporär: mit einem weiteren Taster.

4.8 Definitive Schaltpläne

Setzen wir nun die Ergebnisse der vorangehenden Abschnitte zusammen. Um für die in der ganzen Schaltung verstreuten Inverter, UND-, ODER- und NAND-Gatter mit möglichst wenigen ICs auszukommen, können (wie schon beim Programm) gewisse Änderungen vorgenommen werden, die keinen Einfluss auf die Funktion haben: zum Beispiel Ersetzen eines ODER-Gatters mit invertierten Eingängen durch ein NAND oder Eliminieren von aufeinanderfolgenden Invertern. So entstehen die definitiven Schaltpläne: Übersicht als Blockschaltbild (Abbildung 9), Details über Taktverteiler (Abbildung 10), Programmspeicher (Abbildung 11) und Rechner (Abbildung 12). Alles zusammen auf einer Seite gibts unter ‹http://www.n.ethz.ch/student/walthec/vp/digital/schaltplan.pdf›.

4.9 Dimensionierungen

In der Schaltung kommen einige Widerstände und ein Kondensator vor. Was sollen diese für Werte haben?

Pull-Down-Widerstände
Die Pull-Down-Widerstände bilden zusammen mit ihrem Schalter einen Spannungsteiler. Um bei offenem Schalter Potential 0 und bei geschlossenem Schalter die volle Betriebsspannung am Ausgang des Spannungsteilers zu haben, muss der Wert des Widerstands viel grösser sein als der Widerstand des geschlossenen Schalters (≈ 0) und viel kleiner als der Widerstand des offenen Schalters ($\approx \infty$). Ausserdem soll bei geschlossenem Schalter ein nicht allzu grosser Strom fliessen. Die erste Anforderung wird durch einen breiten Bereich von Widerstandswerten erfüllt, die zweite hängt davon ab, was man als «nicht allzu grossen Strom» betrachtet. Ich habe mir «etwas weniger als 1 mA» vorgegeben und darum für 8–10 V Betriebsspannung 12 kΩ gewählt.

LED-Vorwiderstände
Rote LEDs arbeiten typischerweise bei einer Spannung von ca. 2 V. Für 8–10 V Betriebsspannung bleiben also 6–8 V am Widerstand zu vernichten. Wenn für die Low-Current-LED 2–2.5 mA durchfliessen sollen, ergibt das nach Ohmschem Gesetz einen Widerstandswert von 2.4–4.0 kΩ. Ich habe 3.3 kΩ gewählt.

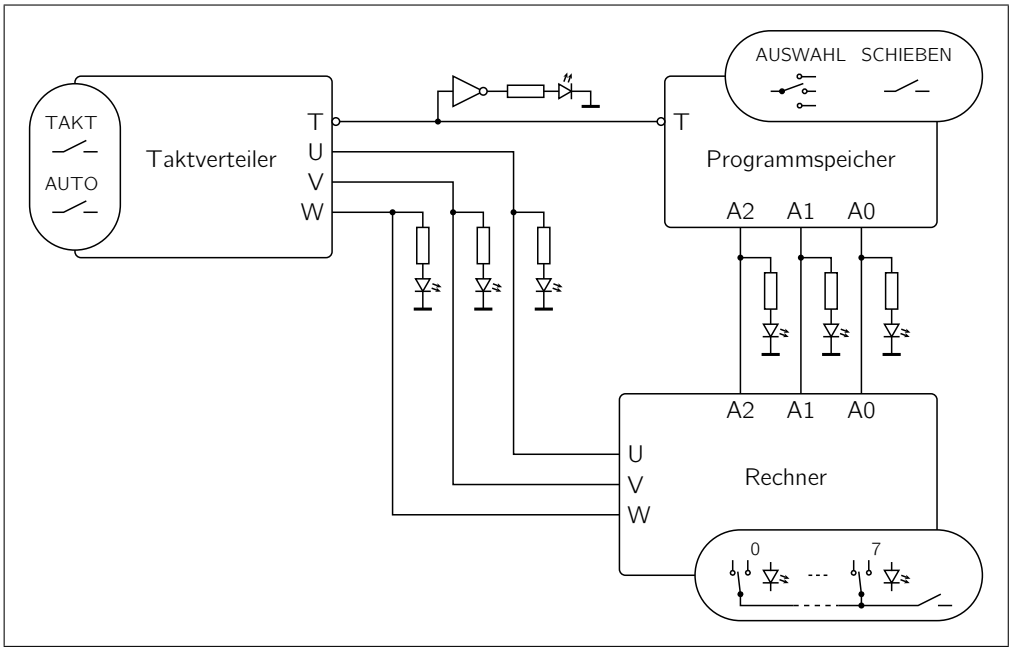

Abbildung 9: Blockschaltbild

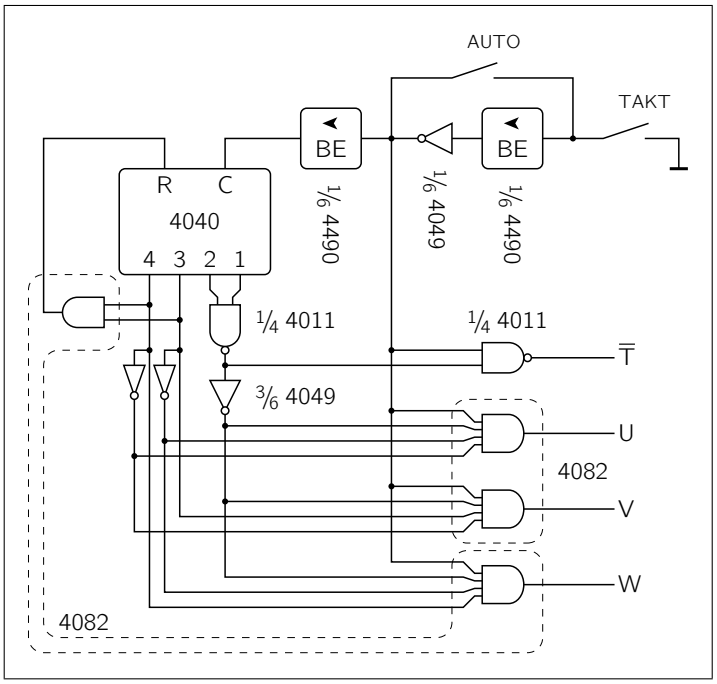

Abbildung 10: Taktverteiler

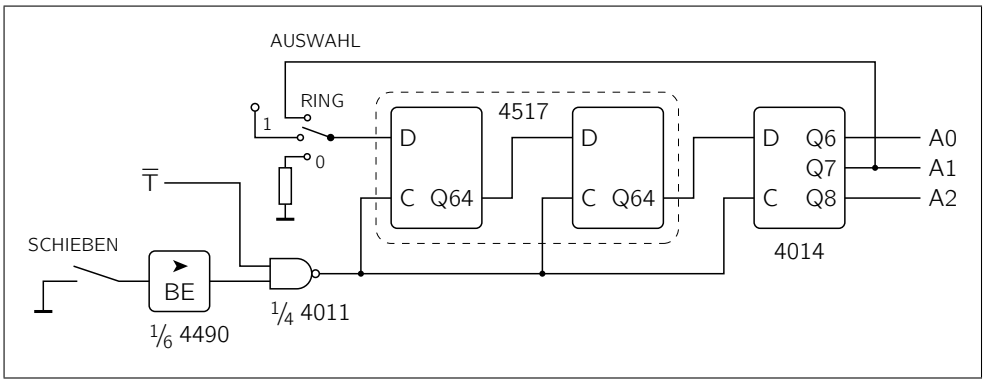

Abbildung 11: Programmspeicher

Kondensator
Der Kondensator am Entpreller-IC bestimmt die Oszillatorfrequenz des Entprellers. Im Datenbuch steht als Richtwert für 10 V Betriebsspannung die Formel $f = \frac{4.5}{C/\mu\mathrm{F}}\mathrm{Hz}$, sie stimmt ungefähr überein mit meinen eigenen Messungen der Periodendauer: 1.08 ms für 5.6 nF, 0.34 ms für 1.8 nF. Für die Takt-Automatik möchten wir die Frequenz so hoch wie möglich haben, andererseits muss sie genügend klein sein, dass der Entpreller seine Funktion noch erfüllt. Improvisierte Messungen mit dem Oszillographen an meinen Tastern (für zuverlässigere Messungen wäre ein Speicheroszillograph nötig, der am Praktikums-Arbeitsplatz nicht gerade zur Hand ist) haben ergeben, dass deren Prellzeit unter einer Millisekunde liegt. Um die Prellzeit innerhalb der 4–5 Oszillatorperioden zu haben, müsste also 1.8 nF genügen. Tests zeigten ausserdem, dass es auch bei geringeren Kapazitäten (durch Serienschaltung von Kondensatoren) kaum vorkommt, dass ein Tastendruck mehrere Impulse generiert. Ich benutzte also 1.8 nF.

5 Der Bau der Schaltung

Nach dem ganzen Aufwand der Planung hätte ich es schade gefunden, meine Schaltung nur so provisorisch auf den an den Praktikumsarbeitsplätzen vorhandenen Steckbrettern aufzubauen. Ich erkundigte mich deshalb nach einer Möglichkeit, eine dauerhafte Schaltung zu bauen, die auch mit nach Hause genommen werden kann. Diese Möglichkeit existiert tatsächlich, das Physikdepartement verfügt nämlich über für alle Institutsangehörigen (und damit über den Praktikumsassistenten auch für Praktikumsstudenten) zugängliche Einrichtungen zur Herstellung von Printplatten.

5.1 Beschaffung der Bauteile

Anhand der Schaltpläne stellen wir folgende Materialliste zusammen:

CMOS-ICs:

1	4040	12 bit binary counter
2	4082	Dual 4-input AND
1	4049	Hex inverter

1	4011	Quad 4-input NAND
1	4512	Dual 64 bit static shift register
1	4014	8 bit static shift register
1	4028	BCD-to-decimal/binary-to-octal decoder
1	4512	8 channel data selector
5	4013	Dual D flip-flop
1	4490	Hex contact bounce eliminator

Restliche Bauteile:

7	DIL-16 IC-Fassungen
8	DIL-14 IC-Fassungen
15	Low-Current-LEDs
4	Taster
8	Zweifach-Umschalter
1	Dreifach-Umschalter
15	Widerstände 3.3 kΩ
17	Widerstände 12 kΩ
1	Kondensator 1.8 nF

Die ICs 4040, 4082, 4049, 4011, 4014, 4028, 4013; IC-Fassungen, Taster, Widerstände und Kondensatoren sind im Praktikum an Lager. 15 rote Low-Current-LEDs mit 3 mm Durchmesser holte ich bei Pusterla in Zürich (‹http://www.pusterla.ch/›). Den Rest bestellte ich bei Distrelec ‹http://www.distrelec.ch/›:

	Artikel-Nr.	Bezeichnung	
1	20 24 06	205057-P-03V-7XX	Schiebeschalter
8	21 00 07	09 03201 02	Flexibler Brückenschalter
1	64 60 60	MC14490P/DIL-16	Hex Contact Bounce Eliminator
1	64 60 78	MC14512BCP/DIL-16	8-Channel Data Selector
1	64 94 02	HCF4517BEY/PDIP16	Dual 64-Bit Static Shift Reg.

Nachdem ich die Bauteile erhalten hatte, testete ich sie erst einmal aus, ob sie sie auch so verhielten, wie ich mir das aufgrund des Datenbuches vorgestellt hatte. Dies war tatsächlich bei allen der Fall.

5.2 Testlauf

Um meine theoretische Planung vor der Herstellung der Platine noch in der Praxis auszuprobieren, begann ich, den Taktverteiler (vorerst ohne Verzögerung) auf dem Steckbrett aufzubauen. Kurz von Hand getestet, schien er zu funktionieren, und mit Rechteckgenerator und Oszillograph vermochte ich ihm auch genau die in Abbildung 7 geforderten Kurven zu entlocken. Also machte ich schrittweise weiter: nachdem auch Entpreller und Programmspeicher auf Anhieb funktionierten, fügte ich noch den Rechner hinzu, mangels Platz auf dem Steckbrett und angesichts des mittlerweile monströse Ausmasse annehmenden Drahtgewirrs mit 4 statt 8 Registern. Da hatte ich nun also praktisch meine ganze Schaltung vor mir, und soweit ich sie testen konnte, funktionierte sie prima. Meine Planung schien also solide gewesen zu sein. Dass ich den Fehler mit dem Kurzschluss bei der Programmeingabe nicht bemerkte, lag daran, dass die An-

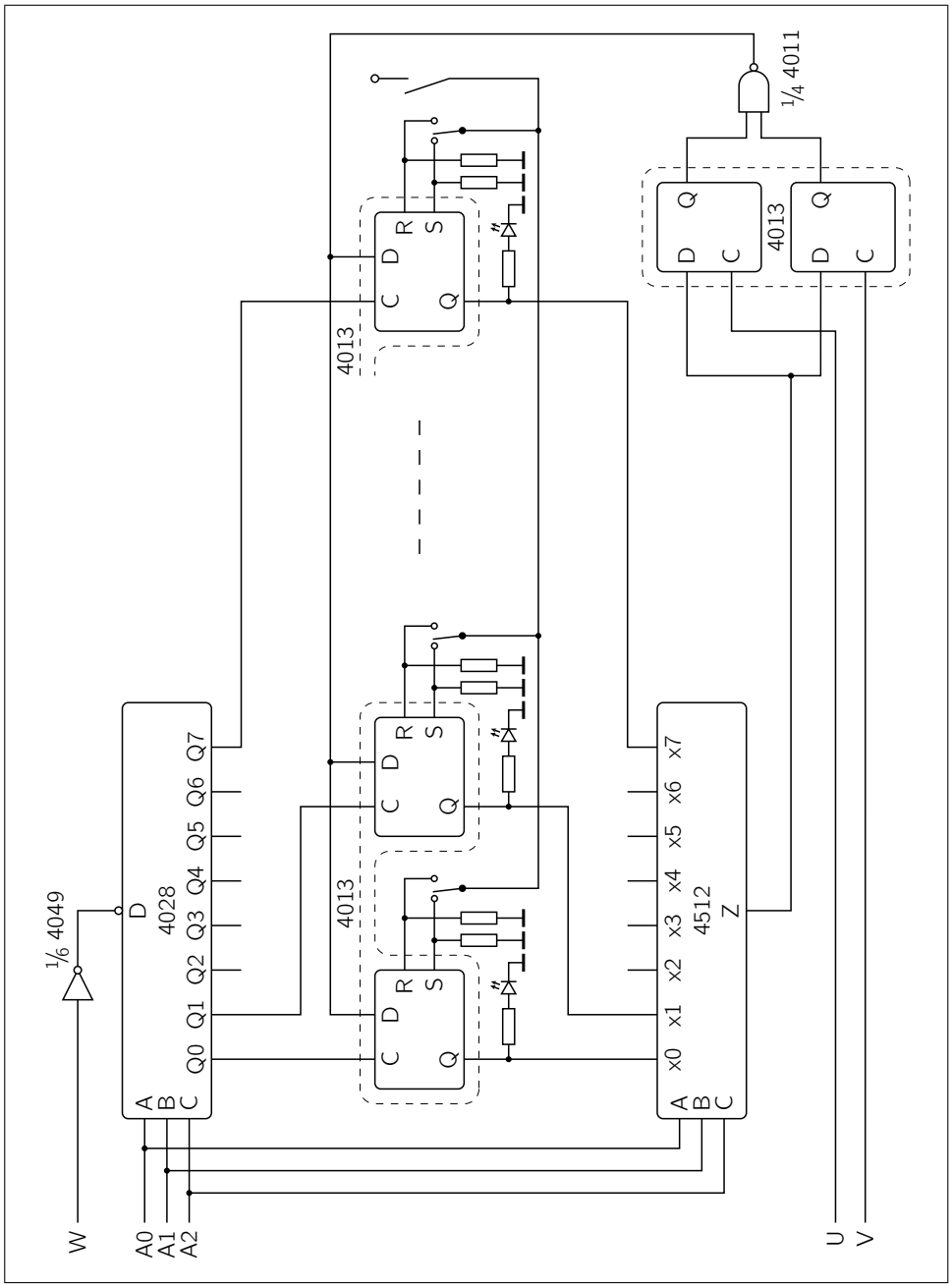

Abbildung 12: Rechner

schlüsse des dafür verantwortlichen Schiebeschalters nicht dem Standard-Rastermass von 1/10" (2.54 mm) entsprechen und somit nicht ins Steckbrett passen. Ich ersetzte den Schiebeschalter deshalb durch einen Draht, der einfach in verschiedene Löcher gesteckt wurde.

5.3 Platinendesign

Für die Planung des Platinenlayouts sind folgende Punkte nützlich zu wissen:

- Das photographische Übertragen der Vorlage und der Ätzprozess sind sehr genau, bei meiner Platine wurden Details bis zu einer Grösse von ca. 0.15 mm aufgelöst.
- Weil die CMOS-ICs nur sehr wenig Strom brauchen, sind keine sehr breiten Leiterbahnen nötig. Ich habe 0.5 mm gewählt, und meine Schaltung funktioniert problemlos damit. Damit ist es auch ohne weiteres möglich, eine Leiterbahn zwischen zwei IC-Beinen hindurchzuführen, was das Platinenlayout wesentlich vereinfacht.
- Die Lötpads sollen nur gerade so gross sein, dass man darauf bequem löten kann, und dass sie durch das Anschlussloch nicht gerade komplett weggebohrt werden. Für die ICs reichen 0.6 mm Lochdurchmesser, die grössten Löcher brauchen die Taster mit ihren ca. 0.8 mm breiten Anschlüssen. Ich habe fast überall rechteckige Pads mit 2.54 mm Breite und 1.52 mm Höhe und halbkreisförmig abgerundeten Enden verwendet. In der Mitte des Lötpads wird ein kleines Loch (ca. 0.3 mm) freigeätzt, damit man sieht, wo das Anschlussloch zu bohren ist.

Für das Platinendesign gibt es spezielle Programme. Ich habe für das grundlegende Layout EAGLE (‹http://www.cadsoft.de/›) verwendet. Leider wird dieses nur für x86 (Linux und Windows) angeboten und hat eine etwas gewöhnungsbedürftige Benutzeroberfläche, dafür bringt es eine ausführliche Bauteile-Datenbank und weitere Fähigkeiten mit, die einem allgemeinen Graphikprogramm fehlen: Man zeichnet zuerst mit Hilfe der Bauteile aus der Datenbank einen Schaltplan und wechselt dann zur Platinenansicht, wo man die in der richtigen Grösse angezeigten Bauteile anordnet und die zunächst als «Luftlinien» angezeigten Leitungen dazwischen durch führt. Abzuraten ist für das Design einer einseitigen Platine nach meiner Erfahrung vom Einsatz des «Autorouters», der das Verlegen der Leitungen automatisch ausführen soll: Er probiert sehr lange herum und kommt dabei nur zu einer unvollständigen Lösung, die auch ein Mensch in derselben Zeit zustande gebracht hätte, weil er keine Bauteile herumschieben kann und auch keine «Gateswaps» und «Pinswaps», also Vertauschung zweier äquivalenter Gatter oder Vertauschung äquivalenter Anschlüsse eines Gatters, durchführen kann.

Das Platinenlayout stellte sich als sehr grosser Aufwand heraus. Nach vielen Stunden war das Layout fertig verdrahtet und nahm nun einen Platz ein, dessen grösste Ausdehnung in beiden Richtungen nur ca. 1 cm grösser war als eine Standard-Platine mit den Massen 16 $\times$ 10 cm. Deshalb versuchte ich nun, das Layout so zu komprimieren, dass es auf eine solche Platine passt. Nach weiteren Stunden gelang mir dies schliesslich, und das Resultat ist zu sehen in Abbildung 13 (Platinenplan) und Abbildung 14 (Bestückungsplan). Diese beiden Abbildungen sind auch einzeln, auf A4 ver-

grössert, erhältlich: ‹http://www.n.ethz.ch/student/walthec/vp/digital/platinenplan.pdf› und ‹http://www.n.ethz.ch/student/walthec/vp/digital/bestueckungsplan.pdf›.

5.4 Platinenherstellung

Unter Anleitung des Assistenten machte ich mich nun an die Herstellung der Platine in den dafür eingerichteten Räumen im Keller des HPF-Gebäudes.

Dabei beginnt man mit einem vergrösserten Ausdruck des Platinenplans, zum Beispiel mit der oben erwähnten auf 170% vergrösserten Version, die mitsamt Schneidemarken auf ein A4-Blatt passt. Dieser wird dann mit einer grossen Reprokamera auf die richtige Grösse verkleinert auf einen Film abgebildet und von dort beim Entwickeln auf eine durchsichtige Kunststofffolie übertragen, so dass man nachher ein positives Bild auf der Folie hat.

Diese Folie wird jetzt auf den mit Kupfer und einem lichtempfindlichen Lack beschichteten Platinenrohling gelegt und das Ganze mit ultraviolettem Licht belichtet. Dadurch wird der Photolack an den belichteten Stellen löslich und kann anschliessend im Entwicklungsbad weggespült werden.

Danach kommt die Platine ins Ätzbad, wo an den vom Lack befreiten Stellen das Kupfer weggelöst wird. Eine Schwierigkeit dabei war, dass meine Platine bis fast an den Rand gefüllt ist und darum kein Platz bleibt für die Einspannvorrichtung. Es zeigte sich aber, dass dies keine Probleme verursacht, sofern man nur die eine Schraube in der rechten unteren Ecke anzieht, wo keine Leiterbahnen sind, da ausser unter der angezogenen Schraube auch unter der Einspannvorrichtung geätzt wird. Es empfiehlt sich, gegen Ende der Ätzzeit gelegentlich den Fortschritt zu überprüfen, da mit der Zeit die Ätzflüssigkeit unter die Ränder des Lacks fliesst und die Kupferbahnen immer schmaler werden.

Nach dem Abwaschen des Photolacks mit Aceton können die Löcher in die Platine gebohrt werden. Dafür benützt man eine extrem schnell drehende Bohrmaschine, die die Platine in Sekundenbruchteilen durchbohrt. Um die richtige Stelle zu treffen, gibt es an der Bohrmaschine eine Lampe, die ein Fadenkreuz auf die Platine projiziert. Nach einigen Übungslöchern an problemlosen Stellen kriegt man damit auch die regelmässig angeordneten Löcher für die ICs schnell hin. Es stehen Bohrer verschiedener Durchmesser zur Verfügung. Um möglichst wenig von den Lötpads wegzubohren, sollte man den jeweils kleinstmöglichen nehmen. Nach meinen Messungen sind die Anschlüsse der IC-Fassungen, LEDs und des Dreifach-Schiebeschalters ca. 0.5 mm dick, die der Zweifach-Schiebeschalter 0.65 mm, die von Widerständen 0.6–0.7 mm, die der Taster 0.8 mm und die Verbindungsdrähte 0.6 mm.

Schliesslich wird die Kupferseite der Platine mit einem Kolophonium-Lack besprüht, der das Kupfer am oxidieren hindert und das Lot besser fliessen lässt.

Zurück im Praktikumsgebäude werden die Bauteile eingelötet. Am besten beginnt man dabei mit den unempfindlichen Brückendrähten und endet mit den LEDs, damit die empfindlichen Halbleiterbauteile möglichst wenig Hitze abbekommen. Dies ist auch mit ein Grund, weshalb die ICs in Fassungen gesteckt anstatt direkt eingelötet werden (neben der Tatsache, dass sich so defekte ICs leichter austauschen lassen).

Sind alle Bauteile eingelötet und die ICs in ihre Fassungen gesteckt, steht eine erste Funktionskontrolle an. Netzgerät anschliessen – leuchten beim Drücken des Taktknopfs die Takt-LEDs in der richtigen Reihenfolge auf? Wenn man Nullen und Einsen in den Programmspeicher schiebt, kommen die am anderen Ende (angezeigt durch

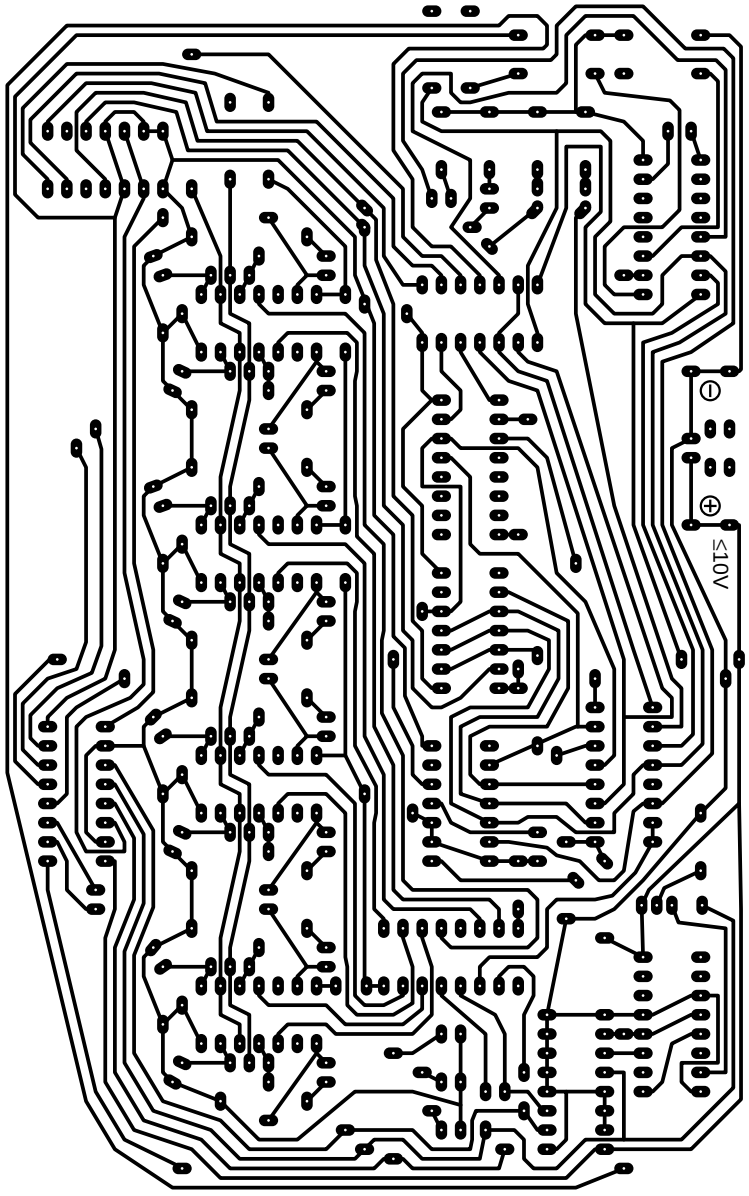

Abbildung 13: Platinenplan, von der Kupferseite her gesehen (Originalgrösse)

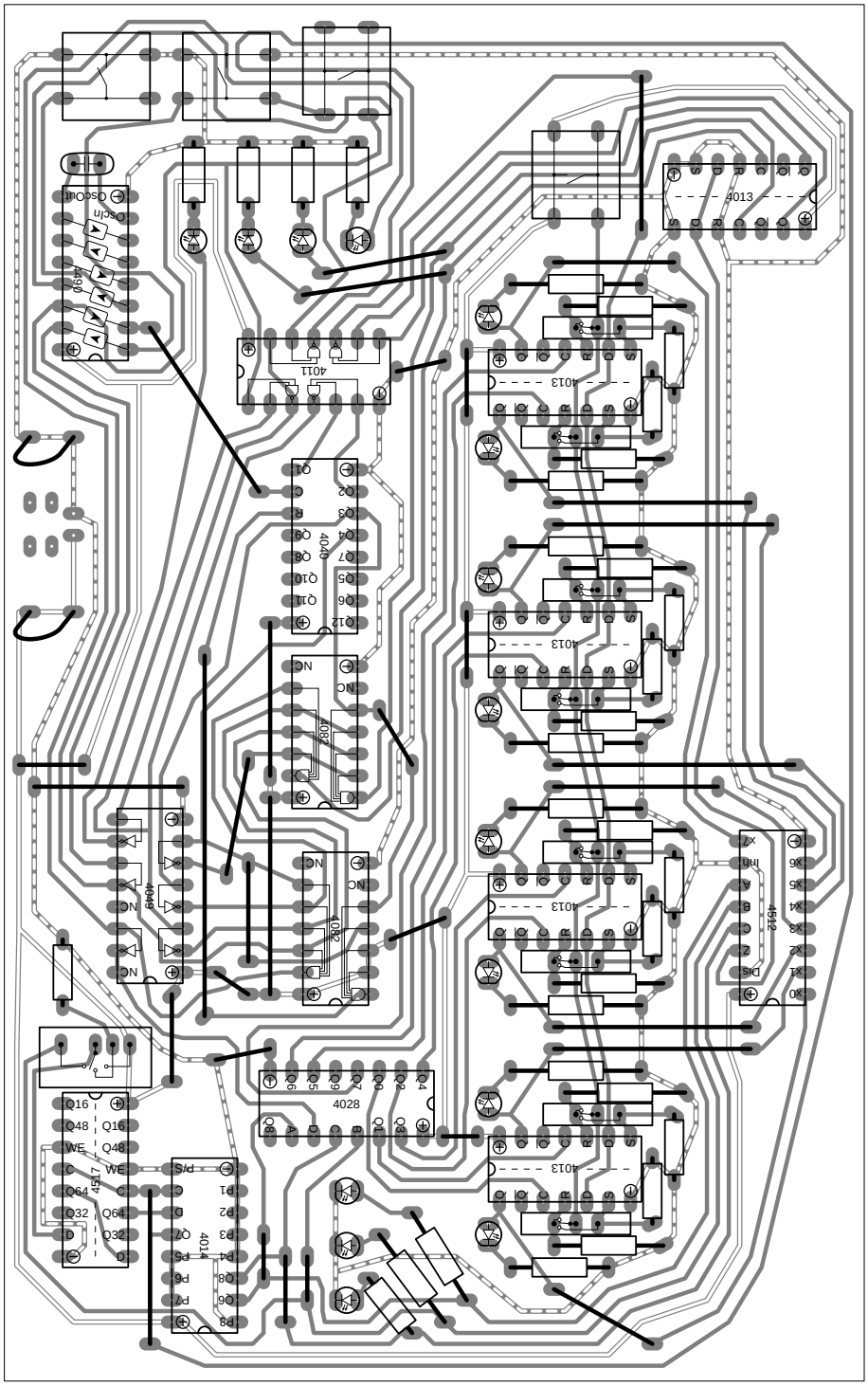

Abbildung 14: Bestückungsplan, von der Bauteilseite her gesehen. Zur besseren Übersicht sind die Spannungsversorgungsleitungen hervorgehoben: ausgezogen +, gestrichelt − (120% der Originalgrösse)

die Adress-LEDs) auch wieder raus? Lassen sich die Register mit ihren Schiebeschaltern setzen und zurücksetzen? Wenn man einen Programmschritt ausführt, der aus drei gleichen Adressen besteht, wird dann das entsprechende Register invertiert? Falls etwas nicht funktioniert, kann der als Logiktester verwendete Oszillograph behilflich sein. Bei mir war es zum Beispiel so, dass eine zu einer LED führende Leiterbahn durch einen Kratzer unterbrochen war, was ich mit einem kleinen Tropfen Lötzinn reparierte.

Wenn alles funktioniert und sicher ist, dass keine Änderungen auf der Lötseite mehr nötig sind, kann die Platine im Printraum in einem Ultraschallbad mit Isopropanol von Kolophoniumrückständen gereinigt werden. Das Bad macht den Bauteilen nichts aus, sie sollten einfach nachher gründlich mit Pressluft getrocknet werden. Anschliessend wird die Lötseite als Isolation und zum Schutz vor mechanischen Beschädigungen mit einem Kunststofflack besprüht.

6 Der Betrieb

Im Prinzip ist unser Logikprozessor nun bereit für die Ausführung seines ersten Programms, des im Abschnitt «Das Programm» entwickelten Halbaddierers. Da er aber inzwischen eine zusätzliche Fähigkeit erlangt hat, den automatischen Takt, und mehr Register erhalten hat, als unbedingt nötig sind, lässt sich dieses Programm noch etwas verbessern.

6.1 Programmanpassung

Beim automatischen Takt wird es nicht möglich sein, das Programm genau einmal auszuführen. Es wird einfach, solange die entsprechende Taste gedrückt ist, wiederholt (je nach Betriebsspannung ca. 2 mal pro Sekunde). Deshalb soll es seine Eingangsdaten nicht zerstören, damit sie am Ende einer Ausführung immer noch als Input für die nächste Ausführung bereitstehen. Wenn man die Taste loslässt, um neue Eingangsdaten einzugeben, wird die Ausführung irgendwo mitten in einem Programm stehenbleiben. Damit von diesem Zeitpunkt an bis zum Beginn des nächsten Programmes die eingegebenen Daten nicht zerstört werden, soll das Programm möglichst gar keine Änderungen an den Registern machen, die für die Eingabe verwendet werden (anstatt einfach ihren ursprünglichen Zustand am Ende des Programmes wieder herzustellen). Damit man die Ausgangsdaten gut ablesen kann, soll dasselbe auch für die Ausgaberegister gelten: Sie sollen ihren Zustand, ausser beim Hineinschreiben des Resultates, während eines Programmes nicht ändern, da man normalerweise nicht weiss, an welchem Ort im Programm man sich gerade befindet. In Abbildung 15 ist ein nach diesen Anforderungen umgestelltes Halbaddiererprogramm gezeigt. Da das Programm ursprünglich nur 12 Schritte lang ist, im Programmspeicher aber 15 Schritte Platz haben, wurden am Ende noch drei Dummy-NANDs auf nicht mehr benötigten Registern hinzugefügt.

6.2 Bedienung

Der Betrieb des Logikprozessors als Halbaddierer geht nun also folgendermassen vor sich (siehe Abbildung 16 für die Bedienungselemente):

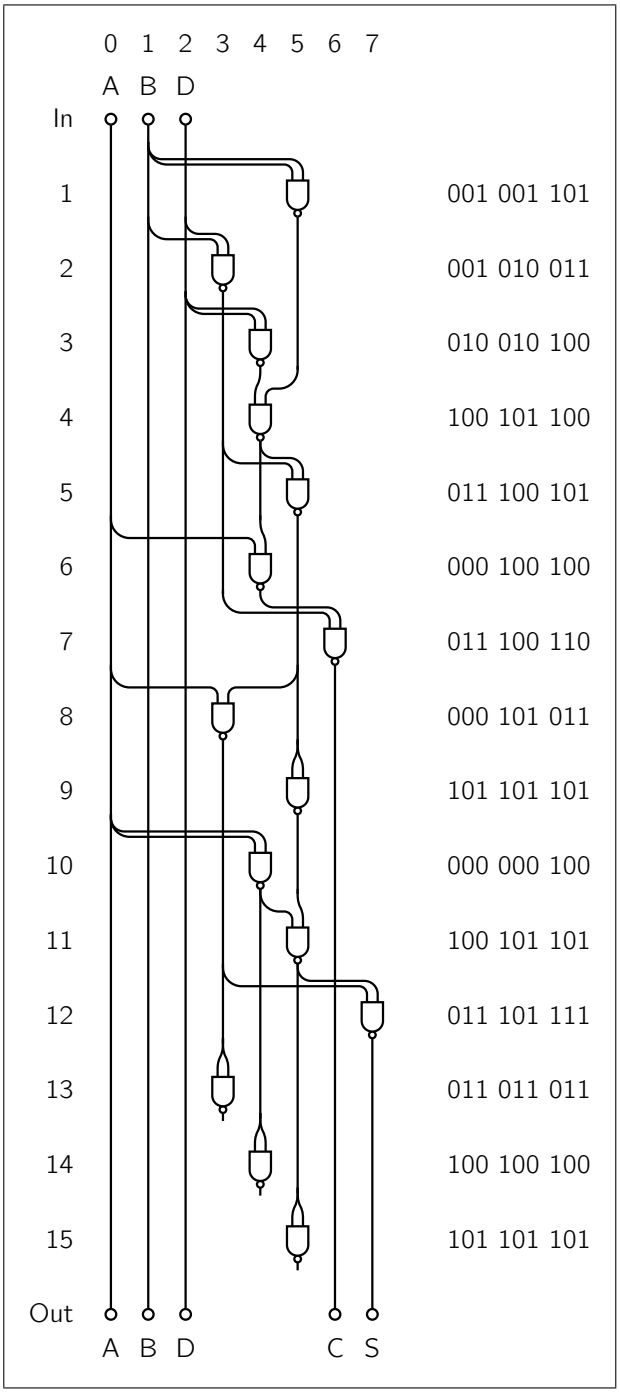

Abbildung 15: Input- und Output-erhaltendes Halbaddiererprogramm

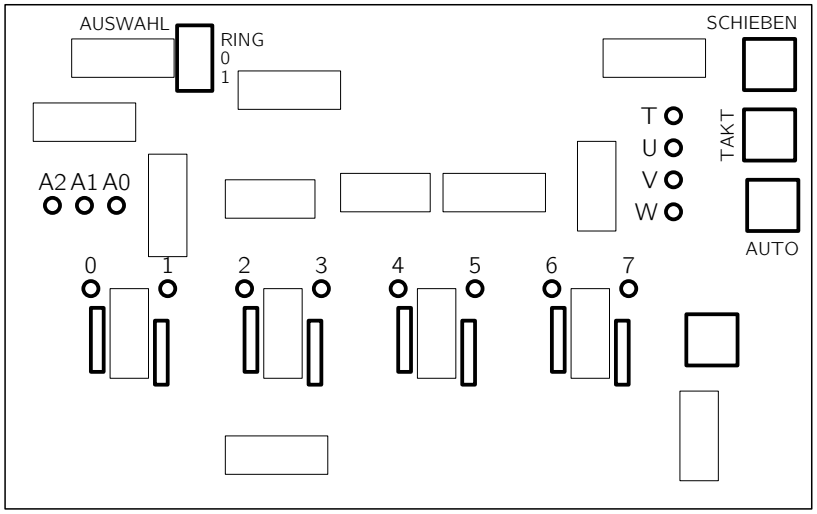

Abbildung 16: Bedienungselemente

- Unser Programm beginnt mit einer Null. Damit wir sehen, wenn es am Ende des Programmspeichers auftaucht, schieben wir zuerst einmal einige Einsen in den Programmspeicher: Stelle den Auswahl-Schalter auf 1 und drücke kurz auf den Auto-Knopf (nach ca. einer halben Sekunde ist damit der ganze Programmspeicher mit Einsen gefüllt).

- Drücke so viele Male auf den Taktknopf, bis ein W-Takt und die darauf folgenden drei T-Takte durch sind, das heisst, bis der Taktverteiler in dem Zustand ist, wo als nächstes ein U-Takt (1. Operand lesen) folgt.

- Stelle den Auswahl-Schalter auf 0, drücke zweimal auf den Schieben-Knopf, den Schalter auf 1 stellen, einmal schieben, Schalter auf 0, zweimal schieben, etc., bis das ganze Programm eingegeben ist. Nach dem Eingeben des letzten Bits müssten die ersten beiden Bits, hier Nullen, an den LEDs A0 und A1 erschienen sein, während auf A2 noch eine der zu Beginn eingegebenen Einsen steht. Falls dies nicht der Fall ist, wurde beim Eingeben ein Fehler gemacht, und es muss von vorne begonnen werden.

- Stelle den Auswahl-Schalter auf RING und drücke einmal auf den Schieben-Knopf, um die erste Adresse des ersten Programmschritts auf das Lesefenster A2–A0 auszurichten.

- Stelle die gewünschten Eingangsdaten an den Schaltern der Register 0 bis 2 ein und drücke auf den Register-Einlese-Knopf.

- Um das Programm manuell abzuarbeiten, drücke wiederholt auf den Taktknopf und beobachte, wie sich bei T-Takten das Programm von rechts nach links durchschiebt und bei W-Takten der Inhalt des adressierten Registers (möglicherweise) ändert.

- Um das Programm automatisch abzuarbeiten, halte den Auto-Knopf gedrückt. Nach einem Programmdurchlauf (also ca. einer halben Sekunde) steht das Resultat in den Registern 6 und 7 und bleibt von da an konstant.
- Um mit neuen Daten zu rechnen, lass den Auto-Knopf los und gib die neuen Daten wie oben ein. Bei erneutem Drücken des Auto-Knopfs wird zuerst, solange der Rest des gerade aktuellen Programmdurchlaufes abläuft, nichts sinnvolles gerechnet werden, aber beim Beginn des nächsten Durchlaufes sind die Eingangsdaten (nach Konstruktion des Programmes) immer noch unverändert, und ab Ende dieses Durchlaufes liegt wieder konstant das Resultat in den Ausgaberegistern.