

27.10.2021

Bau eines 8-Bit Prozessors

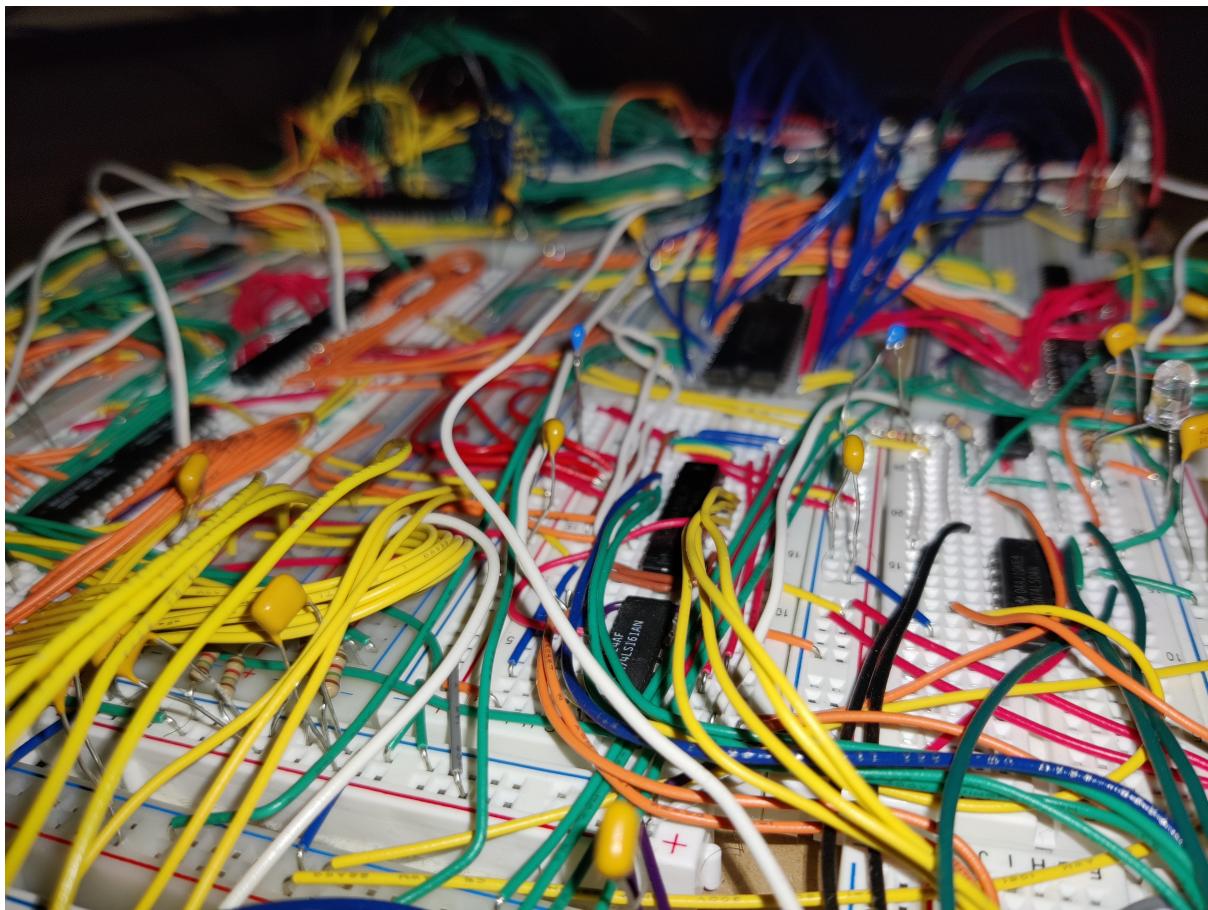


Abbildung 1: Bild des fertigen Prozessors

Verfasser: Dominik Schwaiger

Klasse: 2018NPb

Fachbereich: Informatik

Betreuer: Emil Müller

Ort: Hofackerstrasse 3, 8722 Kaltbrunn

1 Inhaltsverzeichnis

1	Inhaltsverzeichnis.....	2
2	Vorwort.....	3
2.1	Begründung der Themenwahl.....	3
2.2	Dank.....	3
3	Einleitung.....	4
3.1	Fragestellung.....	4
3.2	Ziele.....	4
3.3	Warum Steckbretter?.....	4
3.4	Vorgehensweise.....	4
4	Module.....	5
4.1	Transistor.....	5
4.2	Logikgatter.....	8
4.3	Taktgeber.....	11
4.4	Register.....	15
4.5	ALU.....	16
4.6	RAM.....	21
4.7	Programmzähler.....	23
4.8	Zentraleinheit.....	24
4.9	Ausgang, BUS und Stromzufuhr.....	28
5	Fazit.....	30
6	Quellenverzeichnis.....	32
7	Abbildungsverzeichnis.....	40
8	Tabellenverzeichnis.....	42
9	Weiterführende Literatur.....	44
10	Selbständigkeitserklärung.....	45
11	Anhang.....	46
11.1	Zeitplan.....	46
11.2	Protokoll.....	47
11.3	Liste der Bauteile.....	51
11.4	Wahrheitstabellen.....	53
11.5	Instruktionen Prozessor.....	56
11.6	Programme für den Computer.....	63
11.7	Code.....	65
11.8	Bilder des Prozessors.....	80

2 Vorwort

2.1 Begründung der Themenwahl

Ich hatte schon lange ein grosses Interesse für Computer, wusste aber nie, wie genau sie funktionieren. Mich nahm es sehr wunder, wie ein Computer bis auf der atomaren Ebene aufgebaut und wie er eigentlich entstanden ist. Dank eines Anstosses von meinem Betreuer Herrn Müller kam ich dann so zu einem Video von Ben Eater [1], welcher selbst ein ähnliches Projekt anging. Dies gab mir die Inspiration, die mir noch fehlte, um mein Eigenes zu starten und es als meine Maturaarbeit zu wählen.

2.2 Dank

Ich möchte meinen Eltern danken, welche mich bei meiner Arbeit moralisch unterstützten und mir ab und zu den nötigen Druck gaben. Auch möchte ich Herr Müller danken, welcher mir bei Fragen stets geholfen hat. Zuletzt danke ich auch Ben Eater [1], denn durch seine Videoreihe [2] kam ich erst zu dieser Arbeit.

3 Einleitung

3.1 Fragestellung

Wie funktioniert ein Prozessor und seine einzelnen Komponenten und ist es möglich, diesen selbst herzustellen?

3.2 Ziele

Das Ziel dieser Arbeit ist es, die grundlegenden Komponenten eines Prozessors zu verstehen und so nachzuvollziehen, wie dieser funktioniert und wie die einzelnen Funktionen entstehen. Auch ist es das Ziel, dadurch die heutige Technik besser zu verstehen und ihre Limitationen nachzuvollziehen.

3.3 Warum Steckbretter?

Warum genau sollte man nun aber ein Prozessor mithilfe von Steckbretter bauen? Steckbretter haben viele Vorteile. Einerseits kommen sie in einem Standardformat, in welchem sehr viele Bauteile vorkommen. So muss man sich keine Gedanken machen, ob nun ein Bauteil die richtigen Dimensionen hat oder nicht. Auch ermöglicht ein Steckbrett, dass man Bauteile sehr einfach neu platzieren kann. So kann man verschiedene Schaltkreise austesten, bis man den Besten gefunden hat. Natürlich haben Steckbretter auch ihre Nachteile. Vor allem sind sie relativ teuer im Vergleich zu den restlichen Bauteilen. Auch kann man schwer auf günstigere Versionen ausweichen, da bei diesen die Kontakte qualitativ minderwertiger verbaut sind und es so schwer erkennbare Probleme beim Bau geben könnte. Zudem sind die Dimensionen sehr gross, sodass der Prozessor viel Platz einnehmen wird. Trotzdem waren Steckbretter für dieses Projekt die beste Lösung, da sie perfekt geeignet sind, um Schaltkreise auszutesten, was hier am meisten gemacht werden wird.

3.4 Vorgehensweise

Zuerst wurde ein grober Zeitplan erstellt, der über die Arbeit hinweg verfeinert wurde, um ihn bei entstandenen Problemen und den daraus resultierenden Verzögerungen anzupassen. Während der Arbeit wurde auch ein Protokoll geführt, indem stichwortartig entstandene Probleme, Ideen und der Fortschritt notiert wurden. Zuerst wurde die Theorie erschlossen. Diese wurde grösstenteils mithilfe des Internets aufgebaut. Teilweise wurden Bücher oder Datenblätter der einzelnen Bauteile gebraucht. Diese Theorie wurde dann nochmals verständlicher in einem Notizbuch zusammengefasst. Die Grundidee der praktischen Arbeit war es dann, ein Prototyp eines Bauteils herzustellen, sodass die Theorie bestätigt werden konnte. Es sollte eine Art von «Proof of Concept» werden. Sobald dies gelang, konnten dann massenproduzierte Einzelteile benutzt werden, da der Prozessor ansonsten zu aufwendig zum Bauen wurde (Ein heutiger Prozessor besteht aus 12 Milliarden Transistoren, einer der ersten zwar nur aus 2300, trotzdem aber immer noch eine unglaublich grosse Menge [3]). Das Ziel war es, alles bis und mit dem Transistor zu bauen und natürlich auch zu verstehen.

4 Module

4.1 Transistor

Der Transistor ist das Kernstück jedes heutigen Mikroprozessors. Er ersetzt die vorherig gebrauchten Vakuumröhren [4], welche viel mehr Platz brauchten und viel fragiler waren. Vom Toaster bis zu den schnellsten Supercomputern auf der Erde, überall wird er gebraucht und überall basiert er auf derselben Idee. Er hat zwei grundlegende Funktionen. Einerseits kann er als Verstärker verwendet werden, andererseits kann er wie ein Schalter funktionieren und regeln, ob Strom in einem separaten Stromkreis fliesst oder nicht [5].



Abbildung 2: Nachbau des ersten Transistors

4.1.1 Aufbau

Es gibt zwei wichtige Arten des Transistors. Einerseits den Bipolartransistor (BJT) [6] und andererseits den Feldeffekttransistor (FET) [7]. Beide haben die gleichen grundlegenden Funktionen, wobei FET's spannungsgesteuert¹ und BJT's stromgesteuert² sind. Da bei diesem Projekt nur BJT's gebraucht wurden, wird von ihnen gesprochen, wenn das Wort Transistor gebraucht wird.

Der erste Transistor, welcher je gebaut wurde (siehe Abbildung 2: Nachbau des ersten Transistors [8]) war ein Spitzentransistor [9]–[12] (siehe Abbildung 3: Modell Spitzentransistor [13]). Dieser besteht aus einer Metallbasis auf einem nicht-leitenden Element (Plastik), welche an einen Draht angeschlossen ist, auf diesem Element liegt ein Halbleiter (Germanium) [14], auf welchem sich wiederum ein nicht-leitender Stoff (Plastik) befindet, welcher in Dreiecksform an einer Feder aufgehängt ist. Diese Feder ist wiederum an einem weiteren nicht-leitenden Element (Plastik) befestigt ist. Das nicht-leitende Dreieckselement ist mit einer dünnen Gold-Folie umwickelt, welche an der Spitze des Dreiecks durchtrennt wird, sodass ein sehr kleiner Schnitt entsteht [13]. Durch den kleinen Kontaktbereich kann eine Raumladungszone entstehen, sobald das erste Mal Strom zugeführt wird.

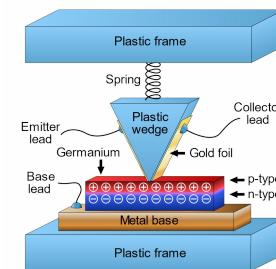


Abbildung 3: Modell Spitzentransistor

Heutige Transistoren werden vor allem nur noch MOSFETs gebraucht, da diese Platzsparender und einfacher herzustellen sind. Sie bestehen aus Silizium, was je nach Verwendung mit anderen Metallen und Stoffen verunreinigt wird. Dieser Prozess der Verunreinigung wird Dotierung genannt [15], [16]. Für den N-Typ des Transistors wird das Silizium mit Arsen verunreinigt, bei dem P-Typ wird es mit Indium oder Gallium verunreinigt. Das Silizium hat dabei die Form von Wafern [17], welche sehr dünne, runde Scheiben sind des puren Elementes. Der Prozess der kompletten

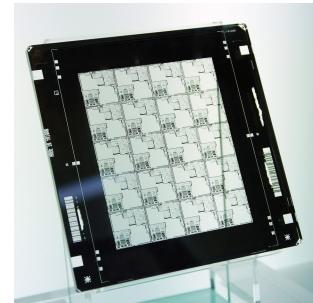


Abbildung 4: Beispiel Fotomaske, Hersteller und Art unbekannt

¹ Bei spannungsgesteuerten Transistoren wird die Ausgangsstromstärke mithilfe der Spannung am Eingang bestimmt. Dies heisst, dass der Transistor nur leitet, wenn eine grosse Spannung am Eingang besteht.

² Bei stromgesteuerten Transistoren wird die Ausgangsstromstärke mithilfe der Stromstärke am Eingang bestimmt. Dies heisst, dass der Transistor nur leitet, wenn eine grosse Stromstärke am Eingang besteht.

Herstellung ist sehr komplex und wird daher auf verschiedene Firmen aufgeteilt. So produzieren nur sehr wenige Firmen ihre Transistoren selber und planen sie auch. Die Transistormanufaktur ist so sehr stark globalisiert. Der eigentliche Schaltkreis des Transistors wird auf einer vergrösserten Platten aufgedruckt, welche Fotomaske [18], [19] genannt wird (siehe Abbildung 4: Beispiel Fotomaske, Hersteller und Art unbekannt [20]), und dann wie durch eine Schablone auf den Transistor gebrannt.

Die heutigen Prozessoren bestehen dabei aus bis zu 40 Milliarden Transistoren [21]. Dabei ist ein einzelner MOSFET 7nm ($2.7 \cdot 10^{-3} \mu\text{m}^2$) gross [22].

4.1.2 Funktionsweise

BJT Transistor

Der BJT Transistor (siehe Abbildung 5: Funktionsweise NPN-Transistor [23]) erweitert das Prinzip der Diode. Er besteht aus den gleichen Elementen, die jedoch anders verwendet werden. (BJT) Transistoren bestehen immer aus einem Halbleiter. Wie schon erwähnt wird heute dabei sehr reines Silizium gebraucht, manchmal auch Germanium [14]. Damit der Halbleiter aber auch als Transistor arbeiten kann, muss man ihn auf molekularer Ebene verändern, in einem Prozess welcher Dotierung genannt wird [15], [16]. Dabei entstehen zwei unterschiedliche Arten von Materialien welche P- und N-Typen genannt werden.

Der N-Typ wird, wie im Kapitel Aufbau erwähnt, mit Arsen verunreinigt. Dabei wird ein Arsen Atom mit einem Silizium Atom in der Kristallstruktur ausgetauscht. Da Arsen ein Elektron mehr hat als Silizium, wird dies nicht in der kovalenten Verbindung gebraucht und an den Kristall abgegeben. So hat der Kristall freie Elektronen und wird zu einem ausgezeichneten elektrischen Leiter. Der gesamte Kristall ist aber immer noch neutral geladen. Beim P-Typ entsteht das genaue Gegenteil. Das Silizium wird mit Indium und Gallium verunreinigt. Da diese Elemente je ein Elektron weniger haben als Silizium entstehen «Löcher». Dieser Stoff ist auch sehr stark elektrisch leitfähig, da Elektronen sich ohne viel Aufwand in die entstandenen Löcher begeben. Verbindet man nun den P- und N-Typ, so entsteht eine Diode [24]. Am Verbindungspunkt des P- und N-Typen entsteht eine Raumladungszone [25] (siehe Abbildung 6: Raumladungszone zwischen dem P-Typ und N-Typ [26]). Diese entsteht, da die «freien» Elektronen vom N-Typ in die «Löcher» des P-Typs herübergehen. So entsteht ein Elektronenfluss. Da nun der P-Typ negativ und der N-Typ positiv geladen ist, können Elektronen, wenn man die Diode anschliesst, nur in eine Richtung fliessen. (Der gesamte Stoff ist aber immer noch neutral geladen, da die Elektronen sich einfach um platziert haben, nicht aber gegangen sind.) Ein Transistor besteht nur aus entweder zwei P- und ein N-Typ oder zwei N- und ein P-Typ. Demnach werden sie PNP- oder NPN-Transistor genannt. Durch diese Verbindung entstehen zwei PN-Übergänge (die Diode hat im Vergleich nur ein Übergang). Dabei nennt man die beiden Verbindungen mit den äusseren Typen Kollektor und Emitter und die Verbindung mit dem mittleren Typ Basis. Nun kann man den Fluss der Elektronen mithilfe der Basis einstellen. Dabei entstehen wieder zwei Raumladungszenen (siehe Abbildung 7:

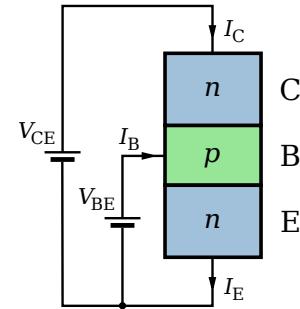


Abbildung 5: Funktionsweise
NPN-Transistor

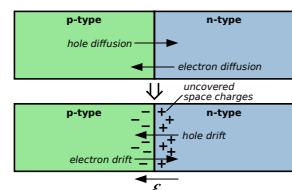


Abbildung 6: Raumladungszone
zwischen dem P-Typ und N-Typ

Raumladungszonen Transistor [27]). Wenn nun an der Basis eine positive Spannung liegt, dann können Elektronen frei zwischen Emitter und Kollektor fliessen, da die Löcher im P-Typ direkt immer gefüllt werden und die Elektronen im N-Typ sich frei bewegen können. Wenn nun auf der Basis keine Spannung liegt, dann hat es im P-Typ freie Löcher. Es entstehen zwei Ladungszonen. So kann kein Strom fliessen. So fliesst nur dann Strom, wenn auf der Basis auch Strom liegt. Damit kann man mit einer kleinen Spannung auf der Basis eine grosse Spannung kontrollieren. Auch kann ein Transistor als Verstärker funktionieren, indem Schwankungen an der Basis mit kleiner Spannung auf eine grosse Spannung übertragen und so verstärkt werden [28].

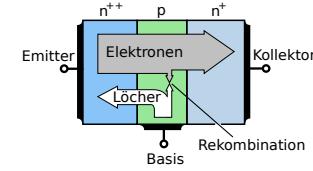


Abbildung 7: Raumladungszonen Transistor

Praktische Implementierung und Probleme

Eine Möglichkeit, einen Transistor zu bauen, ist es, den ersten jemals gebauten Transistor nachzubauen. Dieser ist der bereits beschriebene Spitzentransistor (siehe Aufbau). Hier gibt es aber das Problem, dass der Nachbau einerseits sehr viel Arbeit braucht und andererseits die Materialien auch relativ schwer zu bekommen sind. Da ein Transistor prinzipiell aus zwei aneinandergehängten Dioden besteht, könnte man auch einfach eine Diode umbauen und diese als Transistor gebrauchen. Heutige Dioden sind aber für das zu klein und zu kompliziert, sodass eine Punktkontaktdiode [29] (siehe Abbildung 8: Punktkontaktdiode [30]) besser dafür geeignet ist. Dabei kann man die beiden N-Teile verbinden und so ein PNP Transistor bauen oder zwei P-Teile auf einem N-Substrat erstellen.

Beim Versuch, dies nachzubauen, funktionierte es aber nicht. Dies liegt wahrscheinlich daran, dass die Kabel sehr dünn sind und bei der Handhabung zerstört wurden. Die Dioden wären Germaniumdioden gewesen, welche eigentlich für TV/Radio gedacht wären, aber auch gut für diesen Zweck funktioniert hätten (siehe Liste der Bauteile).³

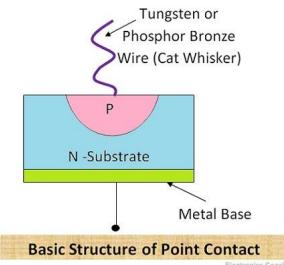


Abbildung 8: Punktkontaktdiode

³ Ganzer Abschnitt sinngemäss nach [31].

4.2 Logikgatter⁴

Nach dem Transistor sind Logikgatter die kleinsten Komponenten in einem Prozessor. Diese Komponenten führen verschiedene Logikoperationen durch. Allein bewirken sie nicht viel, aber wenn man sie komplex verknüpft, können sie unzählige Funktionen haben. Grundsätzlich gibt es sieben verschiedene Typen von Gatter; NOT, NAND, NOR, XNOR, XOR, OR und AND. Die Gatter NAND und NOR werden auch universelle Logikgatter [38] genannt, da man aus ihnen alle restlichen Gatter erstellen kann. Logikgatter-Schaltkreise bestehen in der Theorie nur aus Transistoren, werden aber in der Praxis noch mit Widerständen erweitert, einerseits zum Schutz, damit keine Kurzschlüsse entstehen, aber auch andererseits, damit sie eine höhere Effizienz haben. So wird ein Transistor nach dem Ausgang, aber vor GND, platziert, sodass der Strom immer über den Ausgang fliesst, da dieser einen kleineren Widerstand besitzt.

Alle Logikgatter bestehen aus Widerständen aus dem Widerstandskitt, BJT-Transistoren und Steckplatten (siehe Liste der Bauteile). Sie wurden gemäss den Schaltkreisen, welche in folgender Theorie erklärt werden, nachgebaut. Zusätzlich wurden Drucktaster mit den verschiedenen Eingängen und LEDs mit den Ausgängen verbunden, um die Bedienung einfacher zu gestalten. Es wurden die Widerstände gebraucht, welche sonst am wenigsten gebraucht werden, da es hier nicht gross auf die Stärke der Widerstände ankommt, um das Konzept zu erklären. So bewegen sich die Widerstände im Bereich von 220-16000 Ohm.

4.2.1 NOT

Das NOT-Gatter (auch Invertierer) ist sehr simpel in seiner Funktionsweise. Es invertiert das einkommende Signal. So wird ein logisches 0 zu einem 1 und umgekehrt (siehe Tabelle 4: Wahrheitstabelle NOT-Gatter).⁵ In der Mathematik entspricht dies dem $\neg A$ oder \bar{A} . Es existieren verschiedene Standards für die elektronischen Symbole. Die häufigsten drei sind ANSI [40], IEC [41] und DIN [42]. In einer Schaltung wird es meistens mit dem ANSI Symbol $\text{^}\rightarrow\circ$ [43] repräsentiert.

Das NOT-Gatter (siehe Abbildung 9: NOT-Gatter Aufbau [44], Abbildung 45: Programmzähler) besteht aus einem Transistor. Wenn U_e HIGH ist, ist der Transistor eingeschaltet, sodass Strom direkt (durch den Widerstand R_c) von V_{cc} zu GND⁶ laufen kann, sodass keine Spannung auf U_a liegt, da dort der Widerstand grösser ist (R_v). Wenn U_e Tief ist, dann kann Strom nur durch U_a fliessen, sodass U_a dann HIGH ist.

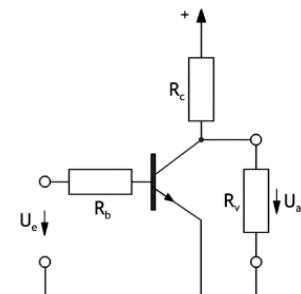


Abbildung 9: NOT-Gatter Aufbau

4.2.2 AND

Auch das AND-Gatter funktioniert sehr simpel. Es gibt nur HIGH aus, wenn beide Eingänge HIGH sind (siehe Tabelle 5: Wahrheitstabelle AND-Gatter). In der Mathematik entspricht dies dem $A \cdot B$ oder $A \wedge B$. In einer Schaltung wird es meistens mit dem Symbol ΞD [45] repräsentiert.

⁴ Ganzes Kapitel sinngemäss nach [32]-[37].

⁵ Ein logisches 0 (auch LOW) hat fast immer die Spannung 0V, ein logisches 1 (auch HIGH) hat meistens die Spannung 5V, kann aber auch andere haben. Die Spannungen sind aber auch nicht als einzelne Zahlen, sondern eher als Bereiche definiert [39].

⁶ V_{cc} = Betriebsspannung (logisches 1), GND = Ground (logisches 0).

Das AND-Gatter (siehe Abbildung 10: AND-Gatter Aufbau aus Transistoren [46], Abbildung 50: AND-Gatter) besteht aus zwei Transistoren und drei Widerständen. Da die Transistoren seriell verbunden sind, kann nur Strom von V zu GND fliessen, wenn A und B an sind (die Basen sind auf HIGH). Falls nur einer (oder beide) Transistoren nicht angeschaltet sind, so blockieren sie den Weg und es kann kein Strom fliessen.

4.2.3 OR

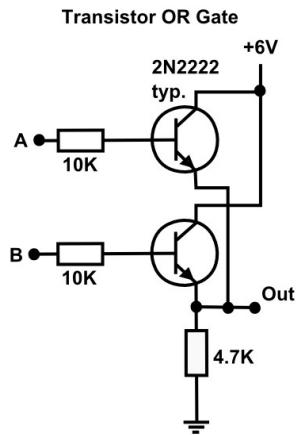


Abbildung 11: OR-Gatter Aufbau aus Transistoren

Das OR-Gatter gibt nur dann LOW aus, wenn beide Eingänge LOW sind (siehe Tabelle 6: Wahrheitstabelle OR-Gatter). In der Mathematik würde dies demnach $A+B$ oder $A \vee B$ entsprechen. In der Elektronik wird es meistens als  [47] gekennzeichnet.

Ein OR-Gatter (siehe Abbildung 11: OR-Gatter Aufbau aus Transistoren [48] und Abbildung 49: OR-Gatter) besteht aus zwei Transistoren und drei Widerständen. Beide Transistoren sind separat mit dem Ausgang Verbunden, so kann schon Strom von V zu GND fliessen, wenn schon nur ein Transistor angesteuert wird.

Transistor AND Gate

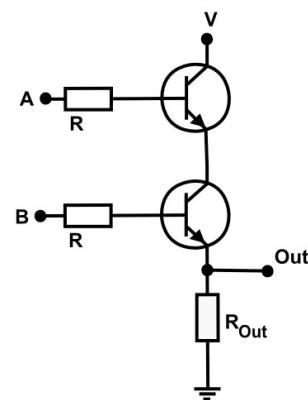


Abbildung 10: AND-Gatter Aufbau aus Transistoren

4.2.4 XOR

Das XOR-Gatter erweitert die Funktionalität des OR-Gatters. So gibt das XOR-Gatter nur HIGH aus, wenn beide Eingänge unterschiedlich sind (siehe Tabelle 7: Wahrheitstabelle XOR-Gatter). In der Mathematik wird es mit $A \cdot \bar{B} + \bar{A} \cdot B$ oder $(A+B) \cdot (\bar{A}+\bar{B})$ oder $A \oplus B$ beschrieben, in der Elektronik verwendet man meistens das Symbol  [49].

Physisch ist es das komplexeste Gatter. Man kann es auf verschiedene Arten erstellen. Einerseits kann man es aus vier NAND-Gatter oder fünf NOR-Gatter, also den universellen Gattern, erstellen. Der effizienteste Weg ist es aber, wenn man es aus einem OR-, einem NAND- und einem AND-Gatter baut (siehe Abbildung 12: XOR-Gatter aus Transistoren [50] und Abbildung 35: XOR-Gatter). Dies erfordert sechs Transistoren und 10 Widerstände.

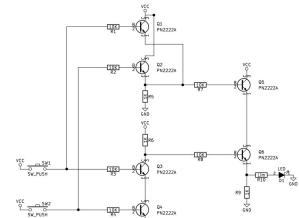


Abbildung 12: XOR-Gatter aus Transistoren

4.2.5 NAND

Das NAND-Gatter ist eines von zwei universellen Gatter. Aus ihm kann man jedes andere Gatter herstellen. Logisch gesehen gibt das Gatter immer HIGH aus, ausser wenn beide Eingänge HIGH sind (siehe Tabelle 8: Wahrheitstabelle NAND-Gatter). Somit funktioniert es gleich wie ein AND- und ein darauf folgendes NOT-Gatter. In der Mathematik wird es somit mit $\overline{A \cdot B}$ oder $\neg(A \wedge B)$ oder $A \uparrow B$ dargestellt. In der Elektronik wird meistens das Symbol  [51] verwendet.

Ein NAND-Gatter kann relativ einfach aus zwei Transistoren und drei Widerständen gebaut werden (siehe Abbildung 13: NAND-Gatter aus Transistoren [52] und Abbildung 32: NAND-Gatter). V ist hier nur mit GND verbunden, falls beide Eingänge HIGH sind, ansonsten fliesst der Strom über den Ausgang. Wenn beide Eingänge HIGH sind, hat es keinen Widerstand zwischen V und GND, sodass der Strom lieber direkt über GND als über den Ausgang fliesst.

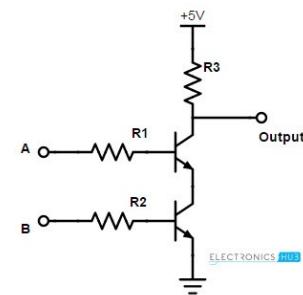


Abbildung 13: NAND-Gatter aus Transistoren

4.2.6 NOR

Das NOR-Gatter ist das zweite universelle Gatter. Auch aus ihm, wie aus dem NAND-Gatter, kann man jedes andere Gatter herstellen. Das NOR-Gatter ist nur dann HIGH, falls beide Eingänge LOW sind (siehe Tabelle 9: Wahrheitstabelle NOR-Gatter). In der Mathematik kann es mit $A \downarrow B$ oder $\neg(A \vee B)$ dargestellt werden. Das Symbol $\begin{array}{c} A \\ B \\ \text{---} \\ \text{O} \end{array}$ [53] wird meistens in der Elektronik gebraucht.

Ein NOR-Gatter kann aus zwei Transistoren und drei Widerständen gebaut werden (siehe Abbildung 14: NOR-Gatter aus Transistoren [54] und Abbildung 51: NOR-Gatter). Sind beide Eingänge auf LOW, so sind beide Transistoren nicht angesteuert, sodass Strom nur über U_a fliessen kann. Falls aber nur schon ein Transistor angesteuert wird, ist die Stromquelle direkt mit GND verbunden, sodass kein Strom mehr über U_a fliessen wird.

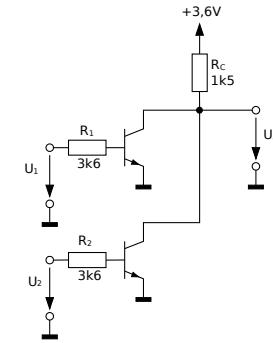


Abbildung 14: NOR-Gatter aus Transistoren

4.3 Taktgeber

Der Taktgeber fungiert als Herz eines Prozessors. Er gibt den Takt, wie einen Puls, an, mit welchem der Prozessor arbeitet. Mithilfe dieses Taktes können alle verschiedenen Komponenten miteinander synchronisiert werden. Das Signal, welches er ausgibt, kommt idealerweise in rechteckiger Form (entweder logisches 1 oder 0). Der Taktgeber kann auf sehr vielen verschiedenen Arten gebaut werden, je nachdem, welches Ergebnis man erzielt. Dabei wird von Oszillatoren [55] gesprochen. Diese nehmen als Eingang eine Gleichspannung und geben eine Wechselspannung (hier als Rechteckform) ab. Heutige Taktgeber bestehen praktisch nur noch aus Quarzoszillatoren [56]. Da es bei meinem Projekt aber nicht auf eine hohe Taktgeschwindigkeit oder Genauigkeit ankommt, wurde in diesem 8-Bit Prozessor ein simplerer Schaltkreis eingesetzt.

4.3.1 Funktionsweise

Quarzoszillator

Diese Art von Oszillatoren ist im Vergleich zu anderen sehr genau (Abweichung der Frequenz⁷ unter 0,0001% [56]). Er basiert auf den piezoelektrischen Eigenschaften [58] des Quarzkristalls. Piezoelektrizität [59] ist dabei die Eigenschaft eines Stoffes, eine Spannung aufzubauen, wenn dieser unter Druck steht. Dies funktioniert, da die Atome in einem piezoelektrischen Stoff Ionen sind. Dies heisst, dass einzelne Atome im gesamten Stoff eine Ladung haben. Auf den Stoff gesehen hat dies keinen Effekt, da es gleich viele negative wie positive Ladungen hat, und so der gesamte Stoff neutral erscheint. Auf atomarer Ebene hat dies aber einen grossen Effekt. Wenn der Stoff in seinem natürlichen Zustand ist, und keine äussere Kraft erfährt, so sind die einzelnen Ionen gleichmässig verteilt, und der Ladungsschwerpunkt⁸ der negativen und positiven Ionen liegt am gleichen Ort. Wird nun eine Kraft auf bestimmte Stoffe ausgeübt, so verschieben sich die Ionen in einer solchen Weise, dass der Ladungsschwerpunkt der positiv und negativ geladenen Ionen sich nicht mehr am gleichen Ort befindet. Dadurch entsteht ein Ladungsunterschied, was zu einer Spannung führt. Dabei ist anzumerken, dass dieser Ladungsunterschied nur besteht, solange der Stoff eine verändernde Kraft erfährt. Falls die Kraft gleich bleibt, gleichen sich die Ionen wieder aus und die Ladungsschwerpunkte [60] liegen wieder am gleichen Ort. Dieser Effekt lässt sich nun auch invers nutzen. So verschieben sich die verschiedenen Ionen, wenn sie einem elektrischen Feld ausgesetzt werden. Die positiven Ionen gehen dabei in Richtung der Anode und die negativen Ionen in Richtung der Kathode [61]. Dieser inverse Piezoeffekt wird nun auch beim Quarzkristall ausgenutzt. Das heisst, dass der Quarz unter einer elektrischen Spannung gesetzt wird. Dabei verformt er sich. Sobald diese Spannung unterbrochen wird, geht der Kristall in seine ursprüngliche Form zurück. Wenn man nun den Ausgang mit dem Eingang des Kristalls verbindet, entsteht eine positive Rückkoppelung [62]. Dabei passt sich die Schwingung des Ausgangs der natürlichen Schwingung des Kristalls an. So entsteht eine sehr genaue Ausgangsschwingung, welche als Taktfrequenz gebraucht werden kann. Dabei muss der Kristall aber in eine festgelegte Form gebracht werden. Früher wurden noch Bergkristalle gebraucht, doch heute stellt man sie künstlich her. Diese Quarzoszillatoren werden in vielen Produkten gebraucht, die auf einen exakten Takt

⁷ Frequenz = Anzahl der Schwingungen pro Zeitspanne [57].

⁸ Der Ladungsschwerpunkt lässt sich herausfinden, indem man die Ladungen als Vektoren darstellt, und dann eine Vektoraddition durchführt.

angewiesen sind. So zum Beispiel Uhren, Computer, aber auch kleinere Elektronikkomponenten.⁹

Mithilfe des NE555's

Da in diesem Projekt eine hohe Taktfrequenz und Genauigkeit von kleiner Bedeutung sind, weil die Programme meistens sehr kurz und nicht auf eine genau Zeitabsprache angewiesen sind, kann man auch eine einfachere Art des Taktgebers verwenden. Dabei wird Gebrauch von der astabilen Kippstufe des NE555es (siehe Astabiler Modus) gemacht. Hierbei funktioniert er wie ein Oszillator. Die Taktfrequenz ist frei bestimbar durch die Stärke der gewählten Komponenten. Die Frequenz f lässt sich hier durch die Widerstände R_1 und R_2 und dem Ladekondensator C berechnen (siehe Formel 1 [64]):¹⁰

$$f = \frac{1}{(R_1 + 2R_2) \cdot C \cdot \ln 2}$$

Formel 1: Frequenz NE555 in astabiler Konfiguration

4.3.2 NE555

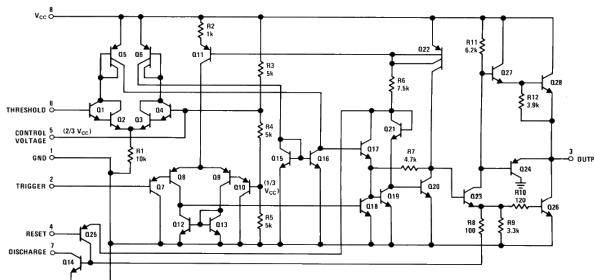


Abbildung 16: Schaltkreis des 555-Timers

Die 555-Timer sind eine der beliebtesten Schaltkreise. Wie man in Abbildung 16: Schaltkreis des 555-Timers [66] und vor allem Abbildung 17: Vereinfachter Schaltkreis des 555-Timers [67] sieht, besteht der 555-Timer aus zwei wichtigen Komponenten: Ein Komparator und ein Flipflop. Dadurch kann er Spannungen vergleichen und das Resultat dieses Vergleiches abspeichern. So ist es möglich, ein Taktgeberschaltkreis zu bauen. Der 555-Timer kann in 4 verschiedenen Modi benutzt werden (Astabil, Monostabil, Bistabil und als Schmitt-Trigger [68], [69]).

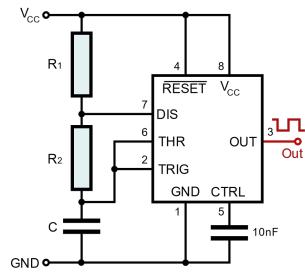


Abbildung 15: Astabile Konfiguration des NE555 Timers

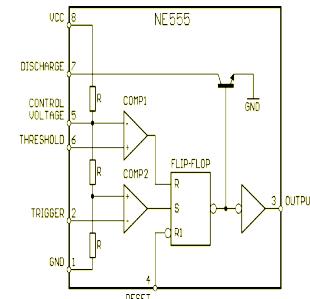


Abbildung 17: Vereinfachter Schaltkreis des 555-Timers

Astabiler Modus

Im astabilen Modus produziert er ein rechteckiges Signal, welches Perfekt als Takt gebraucht werden kann. Dabei wird der 555-Timer wie in Abbildung 15: Astabile Konfiguration des NE555 Timers mit zwei Widerständen und zwei Kondensatoren verbunden. Der RESET Eingang ist dabei immer mit V_{cc} verbunden, da die Schaltung nie zurückgesetzt wird. Ein logisches 0 bei RESET würde dazu führen, dass der Flipflop zurückgesetzt wird. Der

⁹ Ganzer Abschnitt sinngemäss nach [55], [56], [63].

¹⁰ Ganzer Abschnitt und Bild sinngemäss nach [64], [65].

CONTROL Eingang ist über einen Kondensator direkt mit GND verbunden. Der Kondensator hat die Funktion, Spannungsungleichheiten auszugleichen, indem er als Puffer fungiert. Die Vergleichsspannung ist damit immer 0V. THRESHOLD und TRIGGER sind miteinander verbunden. Sie sind über zwei Widerstände mit V_{cc} verbunden. Somit wird die Spannung aufgeteilt im Verhältnis der beiden Widerstände nach dem Gesetz der Reihenschaltung von Widerständen [70]. Ein Zyklus [71] funktioniert so, dass sich der Kondensator C auflädt. Dabei ist der Flipflop gesetzt, da der erste Komparator HIGH und der zweite LOW ausgibt. Dies bleibt so, bis eine Spannung von $\frac{2}{3}V_{cc}$ erreicht wird. Dann geht der erste Komparator auf LOW und der Zweite auf HIGH. Der Flipflop gibt nun LOW aus. Dies führt dazu, dass der Transistor bei DISCHARGE angesteuert wird, und so der Kondensator direkt mit GND verbunden wird. Der Kondensator entlädt sich dann wieder, bis er eine Spannung von $\frac{1}{3}V_{cc}$ erreicht. Dann stellt er sich wieder wie am Anfang ein, sodass der erste Komparator HIGH und der zweite LOW ist. Dieser Zyklus wiederholt sich nun, wobei ein rechteckiges Taktsignal entsteht.

Komparator

Ein Komparator ist ein Schaltkreis, welcher zwei Spannungen A und B vergleicht und entweder ein logisches 1 oder 0 ausgibt, je nachdem, ob die Spannung A oder die Spannung B grösser ist.

Aufbau

Ein Komparator besteht grundsätzlich aus einem (oder mehreren) Differenzverstärker (siehe Abbildung 18: Differenzverstärker [72] und Abbildung 47: Differenzverstärker). Wie der Name schon sagt, bewirken diese Schaltkreise, dass der Spannungsunterschied zwischen zwei Eingangsspannungen vergrössert wird. Dabei geht $+U$ zuerst durch eine Pufferstufe (current mirror). In dieser wird dafür gesorgt, dass auf beiden Transistoren der gleiche Strom fliesst. Man nennt diese Schaltung auch einen Stromspiegel [73]. Ist die Eingangsspannung von Q1 grösser als die von Q2, wird der Transistor bei Q1 stärker aufgesteuert und Q2 weniger, sodass bei Q2 ein grösseres Potenzial entsteht. Hängt man nun genügend solche Schaltkreise aneinander, wird der Unterschied so stark verstärkt, dass er entweder 0 oder 1 ist.¹¹

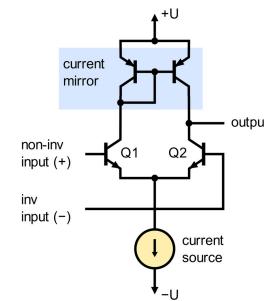


Abbildung 18: Differenzverstärker

Der Differenzverstärker wurde mithilfe von BJT-Transistoren, Widerstände aus dem Widerstandskit und Steckplatten nachgebaut. Beim Nachbau wurde die Pufferstufe aber durch zwei Widerstände auf den entsprechenden Leitungen ersetzt. Auch wurden vor den Basen der Transistor Widerstände platziert. Die Widerstände haben alle den Wert von 220Ω . Der Differenzverstärker funktioniert, aber der Faktor der Verstärkung der Differenz ist relativ klein und variiert sehr stark.

¹¹ Ganzer Abschnitt sinngemäss nach [74]–[77].

Flipflop

RS-Flipflop

Der RS(/SR)-Flipflop [78] (siehe Abbildung 19: RS-Flipflop [79]) ist die simpelste Variante der Flipflops. Er besteht nur aus NAND-Gatter, hat zwei Eingänge, Setzen (\bar{S}) und Rücksetzen (\bar{R}) und zwei Ausgänge (Q und \bar{Q} , wobei \bar{Q} (normalerweise) $\neg Q$). Das Spezielle ist, dass die Ausgänge der NAND-Gatter mit den Eingängen verbunden sind. Aus der Wahrheitstabelle (siehe Tabelle 10: Wahrheitstabelle RS-Flipflop) sieht man, dass der Flipflop dann Daten abspeichert, wenn die Eingänge entgegengesetzt sind. Dabei speichert er HIGH, wenn \bar{S} LOW und \bar{R} HIGH und LOW, wenn \bar{S} HIGH und \bar{R} LOW ist. Wenn beide Eingänge HIGH sind, dann gibt er den vorherigen gespeicherten Wert aus. Dies lässt sich so erklären, dass es durch die Rückkoppelung nur eine stabile Lage gibt. So ist es egal, ob die Verbindung zwischen den NAND-Gatter zuerst HIGH oder LOW ist, denn es gibt nur eine Weise, bei welcher der Schaltkreis stabil ist.

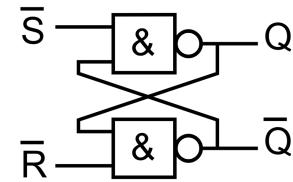


Abbildung 19: RS-Flipflop

Zum Beispiel wenn \bar{S} LOW und \bar{R} HIGH ist, dann kann Q nur HIGH sein und \bar{Q} LOW. Falls es anders wäre, würde sich das System wieder auf diesen Zustand einpendeln. Durch Einsetzen anderer Werte und Nachverfolgung lässt sich dies gut verstehen.

Wenn nun \bar{S} zu HIGH übergeht, dann wird dieses System erhalten, da das NAND Ergebnis von LOW und LOW gleich ist wie HIGH und HIGH. Q ist immer noch HIGH und \bar{Q} LOW. Somit wurde der vorherige Zustand gespeichert. Dies kann man überschreiben, indem man \bar{R} auf LOW setzt. Dadurch wird auch Q auf LOW gesetzt. Die Speicherung erfolgt dann wieder gleich. Beide Eingänge sollten nie LOW sein, da sonst beide Ausgänge LOW wären.

JK-Flipflop

Ein (JK-)Flipflop [78], [80]–[83] (siehe Abbildung 20: JK-Flipflop [84], Abbildung 44: D-Type Flipflop), bestehend aus vier NAND-Gatter, wobei beim Nachbau die CD4011 NAND-Gatter gebraucht wurden, hat drei Eingänge und zwei Ausgänge. Zwei dieser NAND-Gatter sind als RS-Flipflop angeordnet. Dieser JK-Flipflop kann mithilfe eines Taktgebers kontrolliert werden. Dabei liest der Flipflop die Werte an den Eingängen J und K nur dann ein, wenn das Taktsignal HIGH ist. Ansonsten funktioniert er gleich wie der RS-Flipflop, wobei $J = S$ und $K = R$.

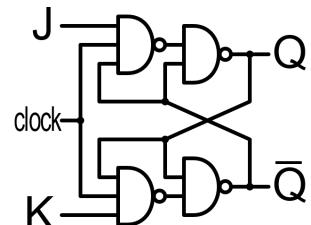


Abbildung 20: JK-Flipflop

Anzumerken ist, dass dieser Flipflop bei hoher Taktfrequenz fehlerhaft funktioniert, sodass er meistens mit komplexeren Schaltkreisen erweitert wird. Bei diesem Prozessor werden solch hohe Taktfrequenzen aber nicht erreicht, sodass dies hier keine Probleme bereiten sollte. Grundsätzlich funktionieren aber alle Flipflops auf dem gleichen Prinzip.

4.4 Register

Das Register besteht aus Flipflops. So hat zum Beispiel ein 8-Bit Register genau acht Flipflops, welche je genau ein Bit speichern können. Die Hauptfunktion des Registers besteht darin, kleine Daten (hier 8-Bit) kurzzeitig, meistens nur für einen Takt oder einen Befehl) für interne Prozesse zu speichern. Dabei ist eine hohe Lese- und Schreibgeschwindigkeit essenziell. Für jede Aufgabe gibt es hier verschiedene Register, so ist der Programmzähler (PC) zuständig für die nächste Speicheradresse der nächsten Instruktion, welche ausgeführt werden muss. Das Speicheradressregister (MAR) speichert die aktuelle Instruktion, welche vom Speicher geladen werden muss, oder es speichert die Speicheradresse, zu welcher Daten gesendet werden müssen. Das Speicherdatenregister (MBR) speichert den Inhalt, welcher sich im Speicher bei der Adresse gespeichert im MAR befindet oder er speichert Daten, welche an den Hauptspeicher weitergegeben werden müssen. Das Instruktionsregister (CIR) speichert die aktuelle Instruktion, welche ausgeführt werden muss. Der Akkumulator (ACC) speichert Daten, welche verrechnet worden sind, oder die Resultate von Rechnungen.¹²

4.4.1 Befehlszyklus

Durch die verschiedenen Rollen der Register entsteht ein systematischer und geregelter Ablauf in der CPU, der sogenannte «fetch-execute-cycle»[88]. Dabei wird zuerst (1) die Adresse, gespeichert im Programmzähler, in das Speicheradressregister kopiert. Danach (2) wird die Adresse im Programmzähler um eins erhöht. Nun hält der Programmzähler die Adresse der nächsten Instruktion. Der Prozessor sendet nun ein Signal dem BUS entlang zu der Speicheradresse gespeichert im Speicheradressregister (3). Nun (4) wird die Instruktion oder die Daten, welche in dieser Adresse gespeichert wurden, über den BUS an das Speicherdatenregister gesendet. Die Instruktion oder Daten im Speicherdatenregister werden nun (5) in das Instruktionsregister kopiert. Zum Schluss (6) wird die Instruktion oder die Daten decodiert und dann ausgeführt. Die Resultate der Berechnungen werden im Akkumulator gespeichert. Dieser Kreis wiederholt sich nun (1).

¹² Abschnitt sinngemäss nach [80], [85]–[87].

4.5 ALU

Die arithmetisch-logische Einheit (ALU, siehe Abbildung 48: ALU) [89], [90] ist im Prozessor zuständig für die Berechnungen. Es gibt auch noch die FPU [91], welche die gleichen Grundfunktionen wie die ALU hat. Sie ist nicht limitiert auf Operationen mit nur Ganzzahlen \mathbb{Z} und kann auch mit Gleitkommazahlen¹³ [92] rechnen. Da dieser Prozessor aber mit Ganzzahlen rechnet, wird dies hier nicht genauer behandelt.

4.5.1 Funktionsweise

Eine ALU sollte mindestens zahlen addieren, so wie subtrahieren können. Sehr viele können auch multiplizieren. Dividieren können sie nur dann, wenn es nicht anders geht, da dieser Schaltkreis um weiten komplizierter ist als die vorherig Genannten. Bei diesem Projekt wäre die Implementierung der Multiplikation auch gedacht gewesen, doch aus Zeitmangel konnte ein Modul dafür nicht vollständig gebaut und integriert werden. So wurde es zwar als alleinstehender Schaltkreis gebaut (siehe Abbildung 33: 4-Bit Multiplizierer), aber nicht mehr in den Prozessor integriert. Die ALU hat bei diesem Prozessor zwei 74LS173N Register (siehe Abbildung 54: Register A und Abbildung 55: Register B), welche separat angesteuert werden können. Beide speichern die Werte, mit welchen Rechenoperation ausgeführt werden können. Das Resultat ist direkt (über einen 74LS245N Sendeempfänger) mit dem BUS verbunden. So wird kein Akkumulator gebraucht. Das Resultat kann, falls nötig, im RAM gespeichert oder im Ausgang dargestellt werden.

Addition/Subtraktion

Addition und Subtraktion funktionieren relativ einfach im Vergleich zur Multiplikation. Da Subtraktion einfach Addition ist, bei der ein Summand mit -1 multipliziert wird, ist das Additionsmodul dasselbe wie das Subtraktionsmodul. Bei der Addition unterscheidet man zwischen zwei Untermodulen, dem Halbaddierer und dem Volladdierer.

Halbaddierer¹⁴

Ein Halbaddierer (siehe Abbildung 21: Halbaddierer Schaltkreis [96]) addiert zwei Zahlen. Er besteht aus einem 74LS08N AND- und 74LS86N XOR-Gatter. Wie man in der Wahrheitstabelle (siehe Tabelle 11: Wahrheitstabelle Halbaddierer) sieht, ist das «carry-bit» nur HIGH, wenn beide Eingänge HIGH sind, da so die Summe grösser als eins ist und es ein Überlauf gibt. Man kann aber mit einem Halbaddierer nur jeweils zwei Bits addieren, da es keinen dritten Eingang hat für das Übertragungsresultat eines externen Addierers. Dieses Problem löst der Volladdierer.

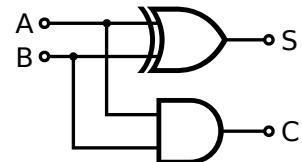


Abbildung 21: Halbaddierer
Schaltkreis

Volladdierer¹⁵

Ein Volladdierer (siehe Abbildung 43: Volladdierer und Abbildung 22: Volladdierer Schaltkreis [99]) besteht aus zwei Halbaddierern und einem NOT-Gatter. Durch diese Konfiguration können drei Eingänge erzielt werden, zwei für das jeweilige Eingangsbit und ein Eingang für den Übertragungswert einer vorherigen Rechnung. Er besitzt die gleichen Ausgänge wie ein normaler Halbaddierer. Aus dieser Konfiguration ergibt sich eine etwas kompliziertere Wahrheitstabelle (siehe Tabelle 12: Wahrheitstabelle Volladdierer). Ein einziger Volladdierer kann zwei Bits miteinander verrechnen.

¹³ Unter Gleitkommazahlen kann man einfacheitshalber die Menge der Rationalen Zahlen \mathbb{R} verstehen.

¹⁴ Ganzer Abschnitt sinngemäss nach [93]–[95].

¹⁵ Ganzer Abschnitt sinngemäss nach [94], [95], [97], [98].

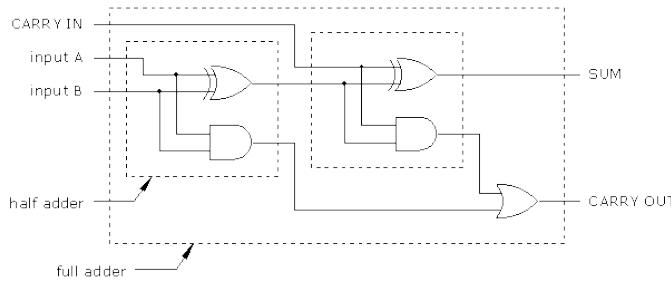


Abbildung 22: Volladdierer Schaltkreis

Da dieser Addierer aber einen zusätzlichen Eingang hat, kann man mehrere Einheiten miteinander zu einer Kette verbinden, sodass es für eine n-Bit Rechnung n Volladdierer braucht. Der erste Addierer kann hierbei durch einen Halbaddierer ersetzt werden, da es beim ersten Addierer keinen dritten Eingang braucht und so Platz gespart wird.

Subtraktion

Wie bereits erwähnt ist die Subtraktion eine Addition, aber der Subtrahend wird mit -1 multipliziert. So findet die Subtraktion und Addition im gleichen Abschnitt des ALUs statt (siehe Abbildung 23: 4bit Volladdierer mit Subtraktionsmethode). Wenn SW3 HIGH ist, dann wird von B, A subtrahiert, ansonsten wird addiert. Eine Multiplikation mit -1 ist aber nicht so einfach möglich, da ein Computer nur 1 oder 0 speichern kann. Hierbei benutzt man das Zweierkomplement [100], [101].

Zweierkomplement

Damit ein Computer auch negative Zahlen speichern und verarbeiten kann, muss die binäre Schreibweise angepasst werden. So wird das «erste Bit von links» zum Vorzeichenbit. Hat dieses den Wert null, so ist die Zahl positiv und kann direkt ausgelesen werden. Hat es jedoch den Wert eins, so steht die Zahl als Zweierkomplement.

Das Zweierkomplement wird gebildet, indem man zum Einerkomplement eins addiert. Das Einerkomplement wiederum wird gebildet, indem man die ursprüngliche Zahl invertiert. Diese Rechnung wird in dem Schaltkreis Abbildung 23: 4bit Volladdierer mit Subtraktionsmethode mit den XOR-Gatter und dem NOT-Gatter realisiert.

Mit dieser Methode lassen sich nun negative Zahlen speichern. Ein solches verändertes Byte nennt man «signed byte» (ein Byte ohne Vorzeichenbit heisst demnach «unsigned byte»). Dabei ist aber zu beachten, dass dieses Byte nun weniger speichern kann, da ein Bit immer besetzt ist. Genauer kann es nun nur noch $2^{(n-1)} - 1$ grosse Zahlen speichern.

4.5.2 Multiplikation

Die Multiplikation ist schon etwas komplizierter und erfordert eine grössere Schaltung, ist aber immer noch möglich. Die einfachste Art der Multiplikation ist die mit zwei. Dabei verschiebt man jedes Bit in der zu multiplizierenden Zahl um eine Position nach links. Genau diese Funktion hat ein Schieberegister.

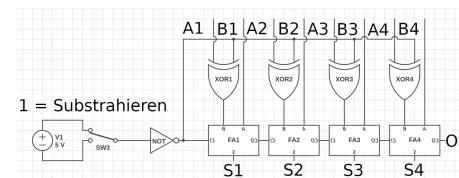


Abbildung 23: 4bit Volladdierer mit Subtraktionsmethode

Schieberegister

Ein Schieberegister [102]–[104] (siehe Abbildung 24: SIPO Schieberegister [105]) besteht aus Flipflops, welche miteinander verbunden sind. Dabei sind die Ausgänge mit den Eingängen verbunden. Schieberegister können in verschiedenen Modi benutzt werden. Dabei unterscheidet man zwischen dem seriellen und parallelen Modus. Beim seriellen Modus werden die Bits nacheinander ein-/ausgelesen, beim parallelen Modus findet dies gleichzeitig statt. Die Daten werden immer bei einem neuen Takt eingelesen und weitergeleitet. So dauert es n -Takte, bis eine n -Bit-Zahl eingelesen ist. Um die Zahl nun mal zwei zu nehmen, muss man sie nur um eine Position im Schieberegister nach links verschieben. Fast immer muss man aber nicht nur Multiplikationen mit zwei ausführen. Dann wird ein komplizierteres Verfahren gebraucht.

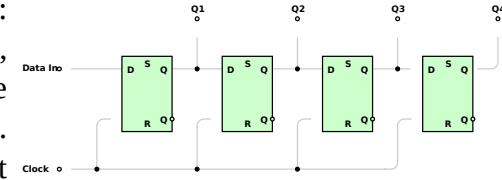


Abbildung 24: SIPO Schieberegister

Binärmultiplikation von Hand¹⁶

Multiplikation mit Binärzahlen basiert auf dem gleichen Prinzip wie die Multiplikation mit Dezimalzahlen. So wird in der Multiplikation mit Binärzahlen der Multiplikator und Multiplikand übereinander aufgelistet. Die einzelnen Ziffern werden dann miteinander multipliziert, und der Überschuss wird in die nächste Rechnung mitgenommen. In der Binärmultiplikation funktioniert es etwa gleich. Auch hier werden beide Zahlen übereinander geschrieben. Danach wird der Multiplikand mit jeder Ziffer des Multiplikators einzeln multipliziert, was je ein Zwischenresultat ergibt. Zusätzlich wird jedes Zwischenresultat um die Nummer der Position der multiplizierten Ziffer nach links verschoben. Am Schluss werden alle Zwischensummen miteinander addiert. Wie man in diesem Beispiel sieht:

	0110 (Entspricht der Zahl 6)
*	1011 (Entspricht der Zahl 11)

+	0110 (0110 * 1)
+	0110 (0110 * 1)
+	0000 (0110 * 0)
+	0110 (0110 * 1)

1000010 (Entspricht der Zahl 66)	

Abbildung 25: Beispiel Binärmultiplikation

ist dies im Vergleich zur Dezimalmultiplikation relativ einfach zu verwirklichen. Um es technisch umzusetzen braucht es aber einen aufwendigeren Schaltkreis.

Multiplikation mit Volladdierer¹⁷

Multiplikation kann mithilfe von Volladdierer erzielen. Dabei braucht es für eine n -Bit Multiplikation ($n-1$) n -Bit-Volladdierer. Zudem braucht es auch noch mehrere AND-Gatter (siehe Abbildung 26: 4-Bit Multiplizierer aus Volladdierer und AND-Gatter, selbst designet, [108]). Hierbei wird das genau gleiche Prinzip wie in Binärmultiplikation von Hand angewendet, aber durch elektronische Bauteile verwirklicht. Die Volladdierer addieren immer die Zwischenresultate, die AND-Gatter sind zuständig für die eigentliche

¹⁶ Abschnitt sinngemäß nach [106].

¹⁷ Abschnitt sinngemäß nach [107].

Multiplikation der Ziffern. Da das Resultat der Multiplikation von zwei Ziffern nur eins ist, wenn beide Ziffern auch eins sind, entspricht dies genau der Wertetabelle der AND-Gatter (siehe AND).

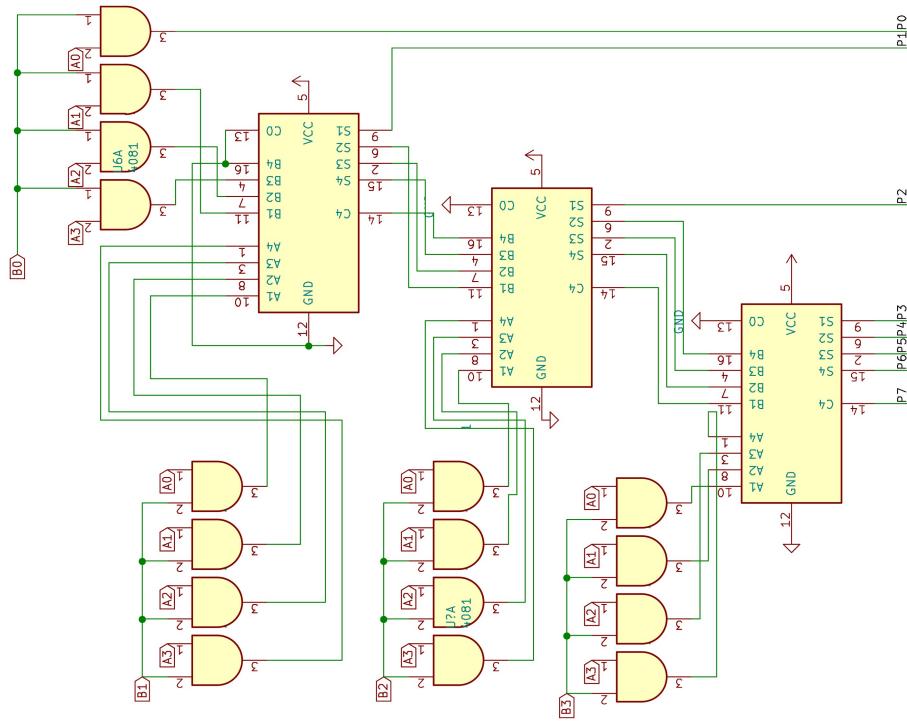


Abbildung 26: 4-Bit Multiplizierer aus Volladdierer und AND-Gatter, selbst designet

Das Multiplikationsmodul (siehe Abbildung 33: 4-Bit Multiplizierer) wurde zwar, wie am Anfang dieses Kapitels bereit erwähnt, gemäss dem Plan in Abbildung 26 (4-Bit Multiplizierer aus Volladdierer und AND-Gatter, selbst designet) nachgebaut, aber konnte nicht mehr in den Prozessor integriert werden, da die nötige Zeit fehlte (siehe Fazit). Die gebrauchten Komponenten waren dabei SN74283N Volladdierer und 74F08PC AND-Gatter.

4.5.3 Division¹⁸

Die Division ist die komplizierteste Grundoperation einer ALU. Division durch zwei könnte man wie bei der Multiplikation mit Schieberegister erzielen, doch für alles andere braucht es einen sehr komplizierten Schaltkreis. Um eine Division umzusetzen, braucht es Subtraktion und Multiplikation. Aufgrund der hohen Komplexität des Schaltkreises wird in dieser Arbeit nur die Division von Hand erklärt. So funktioniert die schriftliche Binärdivision wie auch die Dezimaldivision (siehe Abbildung 27: Beispiel Binärdivision).

Zuerst wird die erste Ziffer von links des Dividenden runter genommen. Danach wird geschaut, ob diese Zahl grösser oder kleiner als der Divisor ist. Falls die Zahl kleiner ist, wird eine Null an den Wert des Quotienten angehängt und eine Null unter die Zahl geschrieben. Falls die Zahl grösser oder gleich dem Divisor ist, wird eine Eins an den Wert des Quotienten angehängt und der Divisor wird unter die Zahl geschrieben. Danach wird die untere Zahl von der oberen subtrahiert. Das Ergebnis wird darunter geschrieben. An das Zwischenresultat wird die nächste Ziffer des Dividenden angehängt. Dies wird nun wiederholt, bis alle Ziffern des Dividenden heruntergenommen wurden.

¹⁸ Abschnitt sinngemäss nach [109].

$10010 : 1001 = 00010$	(18:9)
- - - - -	
1	
0	(0, da $0 < 1001$)
-	
10	
0	(0, da $10 < 1001$)
- -	
100	
0	(0, da $100 < 1001$)
- - -	
1001	(1001, da $1001 = 1 * 1001$)
1001	
- - -	
0	(0, da $1001 - 1001 = 0$)
0	(0, da $0 < 1001$)
- - -	
0	

Abbildung 27: Beispiel Binärdivision

Mit dem Rest könnte genau gleich fortgefahrene werden, bis der Rest 0 ergibt oder sich eine Periode entwickelt. Dies würde aber eine Gleitkommazahl ergeben. Daher sollte eine Division immer nur eine Null als Rest haben. Falls dies nicht so ist, wird der Wert nach dem Komma abgeschnitten, sodass wieder eine Ganzzahl entsteht.

4.6 RAM

(S)RAM [110] wird gebraucht, um Daten zu speichern. Daher funktioniert er praktisch gleich wie ein Register. Auch RAM wird aus Flipflops (oder Kondensatoren) gebaut. Der Unterschied liegt darin, dass im Register Daten meist nur kurzzeitig gespeichert werden. RAM hingegen wird gebraucht, um Daten über die ganze Laufzeit des Prozessors zu speichern. Die Daten im RAM sind aber trotzdem abhängig von einer aktiven Stromzufuhr, da er nur aus Flipflops (oder Kondensatoren) besteht. Daher sind alle Daten verloren, sobald es einen Stromunterbruch gibt.

Generell gibt es zwei verschiedene Arten von RAM, zum einen DRAM [111], zum anderen SRAM [112], [113]. Der grosse Unterschied der beiden Typen besteht darin, dass SRAM die Daten mithilfe von Flipflops, DRAM aber die Daten mithilfe eines Transistors und einem Kondensator speichert. Dabei muss der Datenstand regelmässig aufgefrischt werden, da die Kondensatoren ihre Ladung mit der Zeit verlieren. Der Vorteil liegt darin, dass ein DRAM billiger ist als ein SRAM, da dieser ja weniger Bauteile enthält. Dafür ist ein SRAM schneller und braucht eine weniger dynamische Energieversorgung. In diesem Projekt wurden nur SRAMs benutzt, da diese einfacher zu verstehen und besser im 8-Bit-Format erhältlich sind.

4.6.1 SRAM

Wie bereits beschrieben besteht SRAM aus Flipflops, welche in einem Raster angeordnet werden. Dadurch kann immer eine Spalte mithilfe der Adresse ausgewählt werden, in der die Daten gespeichert werden. Diese Spalte nennt man ein Wort. Ein SRAM kann daher $2^m \cdot n$ Bits abspeichern, wobei m die Anzahl der Bits in der Adresse und n die Anzahl Bits in einem Wort sind.

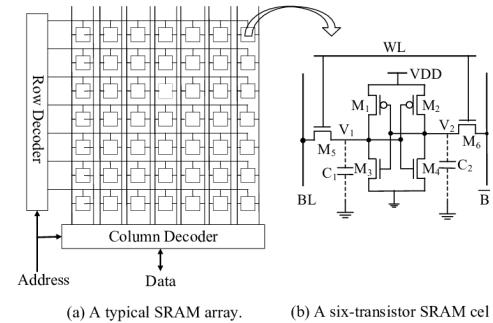


Abbildung 28: Struktur SRAM

4.6.2 Funktion im Prozessor

Der RAM wird in diesem Prozessor gebraucht, um einerseits das Programm selbst zu speichern und andererseits Daten mehrere Takte lang zu speichern. Da die Daten nach jedem Neustart verloren gehen, muss das Programm jedes Mal neu eingelesen werden. Programm und Daten werden in zwei unterschiedlichen RAM-Modulen gespeichert. Dies ermöglicht, anstelle von 4-Bit Instruktionen und 4-Bit Speicheradressen beides in 8-Bit zu haben. Andererseits heisst dies natürlich auch, dass Instruktionen und Speicheradressen nicht gleichzeitig auf dem Bus sein können. Um dieses Problem zu beheben, wurde eine extra Adressbuslinie erstellt, auf welcher sich nur Speicheradressen befinden. Auch haben beide RAM Register, um die Adressen über mehrere Takte zu speichern, sodass auch nach dem Einlesen der Adresse die Daten später von der gleichen Stelle ausgelesen werden können. Der 62256LP-15 RAM, welcher die Daten speichert (siehe Abbildung 53: RAM um Daten zu speichern) ist mit zwei 74LS173N Register verbunden, um die Daten über mehrere Takte zu speichern. So kann eine Adresse, welche über den BUS verteilt wird, in einem Takt eingelesen werden, und in einem anderen Takt können dann die Daten an dieser Stelle ausgelesen oder geschrieben werden. Das Gleiche gilt für die Daten, welche zuerst in einem Register kurz gespeichert werden. So können Daten ausgelesen, aber noch nicht direkt benutzt werden. Die vier 4-Bit-Register sind wiederum über 74LS245N Sendeempfänger mit dem BUS verbunden, sodass sie nicht mit anderen Datenübertragungen interferieren. Der

HM628128ALP-10 RAM mit den Instruktionen funktioniert relativ gleich, nur dass Daten nicht geschrieben werden können, da hier nur das Programm gespeichert wird, welches nur einmal am Anfang programmiert werden muss. Auch ist die Adresse nicht mit dem BUS verbunden, sondern direkt mit dem Programmzähler, da die Instruktionen der Reihenfolge nach gespeichert sind (Erste Instruktion bei 0000 0001, Zweite bei 0000 0010, und so weiter). Auch die Ausgänge sind nicht mit dem BUS verbunden, sondern direkt mit der Zentraleinheit, da dort dann die Instruktionen decodiert werden.

Die Programmierung der jeweiligen RAM-Modulen beginnt am Anfang. Dafür wird der Programmzähler manuell gestoppt und zurückgesetzt. Die Daten werden dann mithilfe eines kleinen Moduls (siehe Abbildung 36: Einheit zur Programmierung des RAMs) von Hand einprogrammiert. Dies ist sehr zeitaufwendig, da man die Daten nach der Programmierung auf Vollständigkeit überprüfen muss. Auch muss man das Modul immer einzeln in den RAM-Modulen verbinden, da die Adressen- und Daten-Pins der RAM-Modulen aus offensichtlichen Gründen nicht miteinander verbunden werden dürfen.

4.7 Programmzähler

Der Programmzähler [114] (siehe Abbildung 37: Ausgang) zählt, in welcher Stelle des Programms wir uns befinden und gibt somit an, welche Instruktion ausgeführt werden muss. Er ist sehr einfach aufgebaut und besteht aus zwei 74LS161N Zählern. Diese bestehen aus mehreren Flipflops, welche miteinander synchronisiert sind. Der Ausgang des einen Flipflops ist mit dem Taktgeber Eingang des nächsten Flipflops verbunden. So wird das Eingangssignal pro Flipflop halbiert, wodurch ein Zähler entsteht. In dieser Version (siehe Abbildung 29: Programmzähler Schaltkreis [115]) gibt es auch noch die Option, ihn zu einem bestimmten Wert zu laden, was uns später die JUMP-Funktionalität gibt. Grundsätzlich ist der Programmzähler mit einem Taktgeber verbunden, sodass nach jedem Takt der Wert um eins erhöht wird. In diesem Prozessor ist er aber nicht mit dem prozessorweiten Taktgeber verbunden, sondern zuerst noch mit einem weiteren Zähler. So können Instruktionen verschiedene Makroinstruktionen haben. Genauer wird dies in dem Kapitel Zentraleinheit beschrieben. Der Wert wird dann an die Zentraleinheit weitergeleitet.

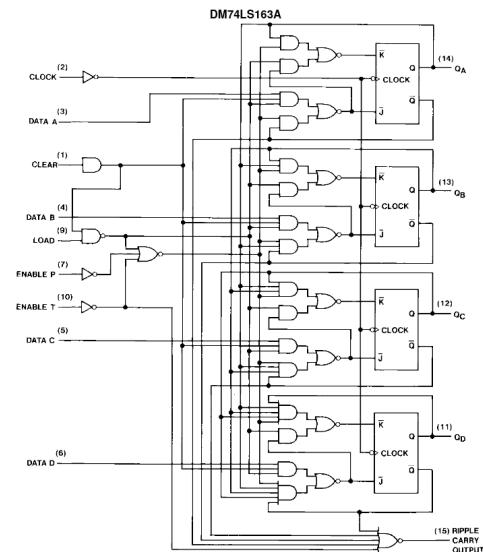


Abbildung 29: Programmzähler Schaltkreis

4.8 Zentraleinheit

Wenn man beim Taktgeber vom Herz des Prozessors spricht, dann ist die Zentraleinheit [116] das Gehirn. Sie bestimmt, was die Befehle bedeuten und führt diese aus. In diesem Projekt wurde sie mithilfe von 28C16A-25 EEPROMs bewerkstelligt. Dies wurde gemacht, da EEPROMs neu programmiert werden können und es daher einfacher ist, Veränderungen vorzunehmen.

EEPROM [117], [118] steht für **Electrically Erasable Programmable Read-only Memory**. Er ist also ein Speicher, welcher elektronisch neu beschrieben werden kann. Daher ist er in einer Unterkategorie der ROMs (Read-only Memory), welche selbst auf verschiedene Weisen und auf Arten implementiert werden können. Es ist ein Überbegriff, welche viele verschiedene Speicher beschreibt. Die genauere Definition dieser Speicher sagt dann aus, ob und wie diese neu beschrieben werden können. Gewählt wurde diese Art des Speichers daher, da er elektronisch am einfachsten zu bearbeiten ist. Wäre er nicht wiederbeschreibbar, so wäre das Risiko hoch, dass ein Fehler bei der einmaligen Beschreibung entstanden wäre. Es gibt auch Versionen, welche durch Beleuchtung mit UV-Licht beschrieben werden können (EPROM). Da dies aber auch mehr Arbeit braucht, wurde die einfachste Methode gewählt.

Da EEPROMs einfach Speicher sind, welche Daten auch ohne Strom speichern können, haben sie eine Adresse und an dieser Adresse Daten. So kann man die Instruktionswerte (siehe Tabelle 13: Befehlslinien Prozessor) als Daten und den «Instruktionsnamen» (die Zahl der Instruktion, siehe 11.5.2 Instruktionen) als Adresse speichern. Sobald nun die richtige Adresse angewählt wird, werden mithilfe der Datenausgänge die richtigen Module angesteuert.

Dies könnte man durch eine komplizierte Logik ersetzen, welche in einem massenproduzierten Prozessor sinnvoll wäre, da die Effizienz so deutlich gesteigert werden kann. Würde der Prozessor daher in der finalen Form nochmals gebaut, dann sollte man die EEPROMs daher durch eine Reihung von Logikgatter-Verknüpfungen ersetzen.

Die Zentraleinheit ist demzufolge die Verknüpfung zwischen manuell programmierten Instruktionen im RAM und den einzelnen Untermodulen des Prozessors. Die Instruktionen werden ausgelesen, ausgewertet und ausgeführt. Die Zentraleinheit ist mit jedem Organ des Prozessors verbunden und kann so kontrollieren, was diese ausführen sollen. Instruktionen werden als Befehle fest vorgeben. Ein Befehl kann dabei mehrere Unterbefehle haben. Um die Unterbefehle zu bewerkstelligen, braucht die Zentraleinheit einen unabhängigen Zähler vom Programmzähler. Dieser Zähler ist direkt mit dem Taktgeber verbunden. Sobald dieser Zähler sein Maximum erreicht hat, wird ein Puls an den Programmzähler gesendet, welches diesen um eins erhöht, wodurch die nächste Instruktion eingelesen wird. Der eigenständige Zähler besteht dabei in diesem Prozessor aus einem 4-Bit-Zähler, wobei er nur bis 111 Zählen kann, da dieser limitiert wird durch die Länge der Adresse der EEPROMs. Diese hat eine Länge von elf Bits. Acht davon werden für die Instruktion gebraucht. Die drei restlichen Bits sind daher für den Zähler. Dies ermöglicht sieben Unterbefehle.

Im EEPROM selber werden die Befehle als Daten an Adressen abgespeichert. Die ersten drei Bits bestehen aus dem Wert des Zählers, welcher beim dritten Bit zu zählen beginnt. Die nächsten acht Bits bestehen aus der Nummer für die Instruktion, welche beim vierten Bit zu zählen beginnt. Jede Instruktionsnummer muss einzigartig sein.

Dieser Prozessor hat 13 verschiedene vorprogrammierte Instruktionen, welche aber auch noch nach Belieben erweitert werden können. Die Instruktionen (siehe Instruktionen Prozessor) lauten: IA (Daten von dem Daten RAM mit der Adresse des aktuellen Standes des Programmzählers in den Register A einlesen), IB (Daten von dem Daten RAM mit der Adresse des aktuellen Standes des Programmzählers in den Register B einlesen), ADD (Daten im Register A und B miteinander addieren und ausgeben), SUB (Daten im Register A von B subtrahieren und ausgeben), OA (Daten im Register A ausgeben), OB (Daten im Register B ausgeben), SA (Daten im Register A im RAM speichern), SB (Daten im Register B im RAM speichern), SADD (Speichert das Resultat von ADD im RAM), SSUB (Speichert das Resultat von SUB im RAM), O (Gibt die Daten vom RAM aus), JUMP (Springt zu einer Stelle im Programm) und HALT (Hält das Programm an).

Gewisse Instruktionen brauchen aber auch einen zusätzlichen Wert, so zum Beispiel IA. Hier muss der Wert angegeben werden, welcher im Register A abgespeichert werden sollte. Nun wird der Wert in dem Programmzähler genommen und als Adresse an den RAM weitergeleitet. Die Daten, welche an dieser Adresse stehen, werden dann je nach gewählter Instruktion weiterverarbeitet. Dadurch müssen immer zwei RAMs programmiert werden.

Der Wert des Programmzählers wird nicht standardmäßig im RAM für Instruktionen eingelesen. Daher muss immer die gleiche Instruktion ausgeführt werden, sobald ein neues Programm beginnt, was dann geschieht, wann immer der Zähler denn Wert eins hat. Diese Instruktion ladet den neuen Programmzähler-Wert in den RAM für Instruktionen. Das soll dazu führen, dass unabhängig von dem Wert im RAM für Instruktionen zuerst der neue Programmzähler Wert eingelesen wird und die programmierte Instruktion für diesen Wert ausgegeben wird. Dies wurde im Code mit einer einfachen Überprüfung verwirklicht.

4.8.1 Programmierung des EEPROMs

Die Programmierung des EEPROMs bereitete erstaunlich viele Probleme, wenn man bedenkt, was für eine kleine Rolle sie im Gesamtprojekt spielt. Zuerst war die Idee, ihn mit einer NodeMCU [119], [120] Einheit zu programmieren (siehe Abbildung 52: Erste Version EEPROM Programmierer). Dies hätte den Vorteil, dass dieser Chip genügend Ausgänge aufweist, um ihn direkt mit allen Eingängen des EEPROMs zu verbinden. Der Code dazu wurde mithilfe Visual Studio Code [121] und der PlatformIO [122] Erweiterung geschrieben.¹⁹

Da dies der letzte Schritt beim Bau der ersten Version des Prozessors war, ging es nachher daran, den Prozessor zu testen. Doch dieser gab nicht den erwarteten Wert aus. Da der Prozessor vorher noch funktionierte, musste der Fehler beim EEPROM liegen. Um zu kontrollieren, ob dieser richtig funktionierte, musste ein weiteres Modul (siehe Abbildung 39: Steckbrett zur manuellen Überprüfung des EEPROMs) gebaut werden. Beim Überprüfen stellte sich jedoch heraus, dass sich an jeder Adresse den Wert 255 befindet. In binärer Schreibweise entspricht dies dem Wert 1111 1111. Laut Datenblatt [123] sind dies die Standardwerte. Dies heisst, dass der EEPROM nicht beschrieben wurde. Manuell lässt er sich aber beschreiben und gibt den zu erwartenden Wert zurück. Daraus lässt sich folgern, dass beim Programmierprozess ein Fehler vorliegen musste.

¹⁹ Diese Version des Programms lässt sich immer noch auf GitHub unter https://github.com/quiode/EEPROM_CAT28C16A_Programmer/releases/tag/v1.0 ansehen.

Zuerst wurde ein simpler Fehler im Programmcode erwartet. Syntax-weise stimmte der Code, da PlatformIO diesen vor Übertragung auf den NodeMCU überprüft. Beim wiederholten Durchlesen fiel auf, dass der Code zur Auslesung der Bit-Stelle falsch war. So wurde zuerst angenommen, dass bei einer binären Zahl von links gezählt wird. Das erste Bit wäre demnach das Bit, welches von links zuerst kommt. Dies ist natürlich falsch. Eine binäre Zahl wird, wie jede andere auch, von rechts gelesen. So ist das erste Bit dasjenige, welches von rechts aus an erster Stelle liegt.

Danach wurde mithilfe von einfachen Print-Ausgaben geprüft, ob das Programm wie erwartet funktioniert. Das Programm lief dabei makellos. Es musste also bei der physischen Implementation ein Fehler vorliegen. Hier könnte es mehrere Fehler geben. Zum einen könnten die Kabel falsch verbunden - oder die Pins im Programmcode falsch definiert sein. Andererseits könnte aber auch ein Fehler beim EEPROM vorliegen. So wäre es einerseits möglich, dass die Spannung zu tief ist. Der NodeMCU gibt nur 3.3V, anstelle von 5V aus. Laut Datenblatt sollte dies aber funktionieren, da der EEPROM bis zu einer Spannung von 3V spezifiziert ist. Ein weiteres Problem könnte auch bei der Zeitabfolge entstehen. So waren die Spezifikationen laut Datenblatt [123] schwer zu verstehen, und es hätte auch sein können, dass der EEPROM zu viel oder zu wenig Zeit bekommt, um Befehle auszuführen.

Um zu erkennen, wo die Ursache des Problems der falsch abgespeicherten Daten lag, wurde überprüft, ob die erwartete Spannung auf den Pins des EEPROM liegt. Dabei stellte sich heraus, dass die Pins falsch definiert wurden. Dies wurde korrigiert. Obwohl nun die Pins stimmten, wurde der EEPROM immer noch nicht richtig beschrieben. Das Problem kann also nur noch bei der Spannung oder bei der Zeitabfolge liegen. Da Letzteres wahrscheinlicher ist, da es laut Spezifikation ja funktionieren sollte, wurden verschiedene Längen für verschiedene Befehle getestet. Doch auch diese Optimierung brachte nichts. So konnte das Problem nur noch daran liegen, dass die Spannung nicht genügend gross ist. Zwar ist sie während des Stand-by Modus des EEPROMs über der Mindestspannung, aber es kann sein, dass sie während des Betriebs unter dieses Niveau fällt. Wäre dies so, würde der EEPROM sich automatisch für eine gewisse Zeit deaktivieren. Dies würde das Nicht-Beschreiben erklären.

Um dieses Problem zu umgehen, wurde versucht, Transistoren einzusetzen. Wie bereits am Anfang der Arbeit erklärt, kann mithilfe von Transistoren ein Schaltkreis durch einen anderen gesteuert werden. Die Idee war, mithilfe der 3.3V des NodeMCUs einen 5V Schaltkreis zu steuern. Um dies zu bewerkstelligen, wurde ein neues Steckbrett (siehe Abbildung 40: Zweite Version EEPROM Programmierer) gebaut, wobei jeder PIN des EEPROMs zuerst mit einem Transistor und erst dann mit dem NodeMCU verbunden wurde.

Doch auch hier funktionierte das Beschreiben immer noch nicht. Die Spannung lag zwar bei einem logischen 0 wie erwartet um die 4.2V, bei einem logischen 0 lag sie nun jedoch nicht bei 0V sondern eher bei 2V. Dies war viel zu hoch für den EEPROM. Daher wurde angenommen, dass die Spannung immer noch das Problem sei, da die restlichen Faktoren (Programmzeit, richtige Pins und richtige Daten) nochmals getestet wurden und keine Fehler festgestellt werden konnten.

Die Idee, den EEPROM mit dem NodeMCU zu programmieren, wurde also verworfen, da es nicht möglich war, mit dem NodeMCU eine ausreichende Spannung zu erzeugen. Es wurde ein neuer Weg gesucht, um den EEPROM zu programmieren. Da aber alle anderen vorhandenen Arduino Modelle nicht genügend Pins hatten, musste eine kompliziertere

Methode gesucht werden, um den EEPROM zu programmieren. Dabei wurde eine Herangehensweise mithilfe von Schieberegister gewählt. Dies funktioniert so, dass alle Bits seriell eingelesen (also ein Bit nach dem anderen) und dann parallel ausgegeben werden.²⁰ Dadurch können durch zwei Pins (einen für die serielle Datenausgabe und einen für das Taktsignal) beliebig viele Pins angesteuert werden. Mit Aneinanderreihung zweier 74HC595 Schieberegister können 16 Pins mithilfe von zwei Pins gesteuert werden. Dies gibt insgesamt zusätzlich 14 freie Pins. Programmiert wird nun neu mit einem Arduino Nano [126]. Dieser hat 13 digitale²¹ Pins. Zwei Pins werden dabei für die Kommunikation mit dem Computer gebraucht. Mit den zusätzlichen 14 Pins durch die Schieberegister bekommt man 27 ansteuerbare Pins. Der EEPROM braucht 22. Somit hat es nun mehr als genügend freie Pins, um den EEPROM zu programmieren.

Nach der Fertigstellung des Programmcodes und des Baus eines neuen Steckbretts (siehe Abbildung 38: Dritte Version EEPROM Programmierer), wurde diese Herangehensweise getestet. Auch diesmal gab der EEPROM nicht die erwarteten Werte aus. Nach genauerer Analyse wurde beobachtet, dass die Schieberegister immer um ein Bit verschoben sind. Um dies zu korrigieren, wurde im Programm und nicht in der Hardware die Funktion immer um ein zusätzliches null Bit ergänzt. Als nun das Programm getestet wurde, sahen die Daten im EEPROM schon etwas anders aus. Das Problem lag nun daran, dass die Daten zwar verändert waren, aber immer noch nicht so, wie sie sein sollten. Zum einfacheren Verständnis wurde eine Funktion erstellt, welche die eingetragenen Daten überprüft und ausgibt, ob sie wie zu erwarten programmiert sind. In dieser Funktion hat sich ein einfacher Fehler einer Falschbenennung einer Funktion versteckt, welcher jedoch noch sehr lange braucht gebraucht hat bis er entdeckt werden konnte. Schlussendlich funktionierte aber das Programm²², und auf den EEPROMs befinden sich nun die richtigen Daten.

Die Daten waren nun so wie geplant einprogrammiert, der Prozessor funktionierte leider immer noch nicht. Dies lag daran, dass die Daten verkehrt eingegeben wurden. Auch hier gab es wieder ein Problem mit der Position des ersten Bits. Dies konnte aber schnell gelöst werden, indem eine neue Funktion hinzugefügt wurde, welche jedes Byte zuerst umkehrte, bevor es einprogrammiert wurde.

²⁰ Die Idee dazu kam vor allem von [103], [124], [125].

²¹ Digitale Pins schreiben oder lesen entweder 1 oder 0. Analoge Pins im Vergleich dazu können unterschiedliche Spannungen schreiben oder lesen.

²² Finale Version ist verfügbar unter Code oder auch https://github.com/quiode/EEPROM_CAT28C16A_Programmer/releases/tag/v3.0.

4.9 Ausgang, BUS und Stromzufuhr

4.9.1 Ausgang

Das Modul, welches für den grafischen Ausgang erstellt wurde (siehe Abbildung 37: Ausgang), wurde recht simpel gehalten. Es besteht aus acht LEDs, welche über ein 74LS273N Register mit dem BUS verbunden sind. Die Speicherung des BUS-Zustandes kann über die Zentraleinheit gesteuert werden.

Ursprünglich wäre gedacht gewesen, ein Display zu gebrauchen, welches die Daten in dezimaler Schreibweise angibt. Dies wurde aber nicht realisiert, da der EEPROM, welcher für diese Funktion nötig gewesen wäre, schon für die Instruktionen gebraucht wurde. Auch reichte es aus Zeitmangelgründen nicht mehr, dies fertigzustellen. Falls noch weiter am Prozessor gearbeitet wird, wäre es aber möglich, ihn mit dieser Option zu erweitern.²³

4.9.2 BUS

Der Bus ist einer der einfachsten Komponenten des Prozessors. Trotzdem ist er essenziell für seine Funktion. Der Bus stellt einen prozessorweiten Weg zur Datenübertragung dar. Dabei gibt es verschiedene Arten des Buses, so gibt es einen Datenbus, welcher nur Daten übermittelt und einen Adressbus, welcher allein für Speicheradressen zuständig ist [128]. Damit nicht jedes Modul direkt mit dem BUS verbunden ist und Elektrizität oder Fehlinformationen austeilten kann, werden sie über einen 74LS245N Sendeempfänger mit dem BUS verbunden.

In diesem Prozessor wurde der Bus sehr simpel gestaltet. Er besteht aus acht Leitungen, welche aus Stromversorgungsstreifen der Steckplatten [129] hergestellt wurden. Der Bus selbst ist auch noch über grössere Widerstände mit der 0V Leitung verbunden, sodass dieser, falls nicht anders bestimmt, den Wert 0 hat (siehe Abbildung 30: BUS mit angeschlossenen Modulen und Abbildung 31: BUS Grossaufnahme).

Sendeempfänger

Ein Sendeempfänger [130] funktioniert relativ einfach. Er verbindet zwei Datenleitungen miteinander, kann aber kontrollieren, ob Daten fliessen und in welche Richtung. Zwei Puffer [131] sind je mit dem Ausgang und Eingang verbunden. Diese können kontrolliert werden, wobei das Kontrollsiegel des einen Puffers das invertierte Kontrollsiegel des anderen ist. So können Daten immer nur in eine Richtung fliessen. Zudem ist der ganze Schaltkreis auch noch mit einem Kontrollsiegel verbunden, welches beide Puffer ausschalten kann und so den Datenfluss komplett unterdrücken kann. Dies ist vor allem wichtig, da Eingänge (und auch Ausgänge) Strom ausgeben, obwohl sie es eigentlich nicht sollten und daher mit dem gewollten Datenfluss interferieren.

4.9.3 Stromzufuhr

Die Stromzufuhr wurde am Anfang des Projektes durch einen Arduino UNO [132] geregelt. Dies hatte den Vorteil, dass dieser direkt 5V und GND Pins hatte. Anfangs bereitete dies auch keine Probleme. Die Stromversorgung lief gut und die Spannung blieb meistens auf 5V. Als die Arbeit aber komplexer wurde und damit die Anforderung an die Stromversorgung stieg, gab es Probleme, eine konstante Spannung zu liefern. Hier reichte es vorerst aus, alle 5V Pins des Arduinos einzeln an verschiedenen Stellen des Prozessors zu verbinden. Auch

²³ Idee von [127].

war die Stromversorgung über die einzelnen Module miteinander verbunden. Dies erwies sich später als ein Fehler.

Mit dem Anschluss weiterer Module funktionierte die Stromversorgung auch mit allen Arduino Pins nicht mehr. Die Spannung fiel oft unter die Mindestanforderungen der einzelnen Komponenten. Dies bereitete viele Probleme und wurde auch relativ spät bemerkt. Danach wurde eine Energieversorgung direkt per Kabel realisiert. Dabei wurde ein herkömmliches USB-Typ A zu Micro-USB [133] Kabel an dem Micro-USB Ende aufgeschnitten. Die 5V und GND Versorgung wurde isoliert und mit Jumperkabeln verlötet. Dieses Kabel (siehe Abbildung 46: Stromversorgung) wurde dann mit einem 5V, 6A Netzteil verbunden. Dies ist ein Schnellladernetzteil. Es wurde ausgewählt, da es am meisten Strom liefern kann. Auch wurde der Prozessor neu verdrahtet. Die Verbindungen zwischen den einzelnen Modulen wurden aufgelöst und es wurde eine Stromlieferungsleiste angebracht. Jedes Modul ist separat mit dieser Leiste verbunden. So haben alle Module gleich viel Strom, und am wichtigsten ist, dass sie direkt mit der Stromlieferungsquelle verbunden sind. Seit dieser Verbesserung gab es keine Probleme mehr mit der Stromversorgung. Die Spannungswerte bei jedem Modul halten sich meist im Bereich zwischen vier und fünf Volt.

5 Fazit

Während des Bauprozesses fielen einige optimierbare Einzelheiten auf, welche den Bau eindeutig erleichtert hätten. So wäre es einfacher gewesen, hätte man von Anfang an eine klare Definition der Kabelfarben gehabt. Schon eine Farbe für 5V, eine für GND und eine für den Datenfluss könnte viel Zeit und einige der angetroffenen Probleme ersparen. Im nächsten Projekt wäre dies sicher zu beachten. Auch wäre es hilfreich gewesen, wenn häufiger LEDs eingebaut worden wären. Dies hätte es deutlich vereinfacht, den Status der einzelnen Module abzulesen.

Es gab Schwierigkeiten, welche am Anfang des Projektes nicht vorausgesehen werden konnten. So war die Namensgebung der Bauteile schwieriger als erwartet. Da die Quellen grösstenteils englisch waren, sind die deutschen Namen der Bauteile meist unbekannt. Für einige war es sogar sehr schwer, ein eingedeutschtes Wort zu finden, da es auch im deutschen Sprachraum üblich ist, den englischen Begriff zu verwenden. Auch war das Debuggen schwerer als zuerst angenommen, da jede Stelle einzeln mit einem Spannungsmessergerät getestet werden musste. Daher nahm dies sehr viel Zeit in Anspruch und verzögerte das Entdecken der Fehler. Zudem gab es am Anfang des Projektes häufig Verwechslungen der Dimensionen. So werden in der Industrie hauptsächlich zwei Formate gebraucht, DIP und SMD. DIP ist der Standard, welcher für Steckbretter gebraucht wird. SMD ist ein viel kleinerer Formfaktor, welcher bei maschinellen Produktionen verwendet wird. So kam es dazu, dass anfangs einige Bauteile im falschen Format bestellt wurden und diese nach Feststellung der unterschiedlichen Formate nochmals neu bestellt werden mussten. Dies verzögerte den Beginn des Projektes stark. Dazu kam auch noch das Problem mit dem Auffinden der Bauteile. Für Steckbretter und Drähte war dies zwar nicht ein grosses Problem, aber die richtigen integrierten Schaltungen in der Schweiz zu finden war sehr schwierig. So wurden die meisten Bauteile aus dem Ausland bestellt, was eine gute Planung erforderte, da eine Lieferung eine Mindestmenge erforderte und die Lieferzeit vor allem unter dem Aspekt der COVID-19-Pandemie auch nicht zu unterschätzen war.

Eine grosse persönliche Enttäuschung war es, dass ich den Prozessor trotz intensiver Auseinandersetzung nicht zu einer funktionierenden Version fertig bauen konnte. Zwar funktionieren die einzelnen Untermodule und von Hand hat er sich auch noch bedienen lassen und gab ein richtiges Resultat aus, doch nach dem Bau der Zentraleinheit funktionierte er nicht mehr. Die manuelle Steuerung wurde an diesem Punkt entfernt, aber es kann davon ausgegangen werden, dass diese immer noch funktionieren würden, wäre sie noch eingebaut. Obwohl in unterschiedliche Richtungen und oft auch tiefgründig nach einer Lösung gesucht wurde (siehe Protokoll), ist der Prozessor noch nicht in der Lage, ein Programm vollständig und korrekt auszuführen.

Folgende Erweiterungen haben es nicht in die finale Version geschafft, obwohl sie zu Beginn vorgesehen waren. So wäre einerseits ein Display, wie im Abschnitt Ausgang beschrieben, gedacht gewesen. Dieses wurde durch einfache LEDs ersetzt. Auch waren erweiterte Funktionen für die ALU geplant. Wie die Implementierung der Multiplikation. Diese wurde aber auch ausgelassen. Zusätzlich wäre auch noch ein direkt verfügbares Schieberegister geplant gewesen, welches dann in einem Programm angesteuert werden könnte. Mithilfe von dieser Option wäre die Multiplikation und Division mit zwei sowie herkömmliche

Schiebeoperationen möglich geworden. Auch wäre ein Statusregister [134], [135] geplant gewesen. Dies hätte auch wieder weitere Möglichkeiten für Instruktionen präsentiert.

Im Nachhinein kann gesagt werden, dass für ein Projekt mit diesen Zielen die zur Verfügung stehende Zeit innerhalb einer Maturaarbeit um ein Vielfaches übersteigen kann, insbesondere dann, wenn unerwartete Probleme zu lösen sind.

Schlussendlich hat mir dieses Projekt aber stark geholfen zu verstehen, wie ein Prozessor funktioniert und was die Mechanismen dahinter sind. Ich verstehe nun, wie man von einem relativ einfachen Transistor darauf kam, komplexe Programme auszuführen. Auch gab es mir einen guten Einblick in der Entstehung der Programmiersprachen. So kann ich nun besser einzelne Datentypen verstehen oder auch generell, wie eine Programmiersprache physisch umgesetzt wird. Diese Herausforderung hat mir, trotz mehreren Problemen und stundenlangen debuggen, viel Spass gemacht und ich habe viel neues dazugelernt. Ich würde dieses oder ein ähnliches Projekt jedem empfehlen, selbst zu bestreiten, falls auch sie sich für die komplexen Mechanismen eines Prozessors interessieren. Empfehlen kann ich hierbei die Videoreihe und den Bausatz von Ben Eater [2]. Da diese Maturaarbeit als Vorlage dienen kann, sind alle Bauteile dieses Prozessors in Tabelle 3: Liste der Bauteile aufgelistet. Preislich gesehen kann man sehr günstig einen Prozessor bauen, da (bei richtiger Bestellung) sämtliche Bauteile auch in der Industrie gebraucht werden und daher in grossen Mengen zur Verfügung stehen.

6 Quellenverzeichnis

- [1] B. Eater, «Ben Eater». <https://eater.net> (zugegriffen Jan. 13, 2021).
- [2] «Build an 8-bit computer | Ben Eater». <https://eater.net/8bit/> (zugegriffen Dez. 22, 2020).
- [3] «12 Milliarden Transistoren in einer CPU», *pressetext*. <https://www.pressetext.com/news/20180112018> (zugegriffen Sep. 16, 2021).
- [4] «Vacuum tube», *Simple English Wikipedia, the free encyclopedia*. Apr. 02, 2021. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: https://simple.wikipedia.org/w/index.php?title=Vacuum_tube&oldid=7456580
- [5] «Transistors: Your Complete Guide On How To Use Them In Electronics», *Electronics Design HQ*, Dez. 12, 2016. <https://www.electronicsdesignhq.com/transistors/> (zugegriffen Dez. 25, 2020).
- [6] «Bipolartransistor», *Wikipedia*. Juni 02, 2021. Zugegriffen: Juni 28, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Bipolartransistor&oldid=212598738>
- [7] «FET Transistor Homemade From Cadmium Sulfide Photocell.» <http://sparkbangbuzz.com/cds-fet/cds-fet.htm> (zugegriffen Dez. 25, 2020).
- [8] Stahlkocher, «Nachbau_des_ersten_Transistors.jpg». Wikipedia, Okt. 03, 2004. Zugegriffen: Juni 28, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:Nachbau_des_ersten_Transistors.jpg
- [9] «Point-Contact Transistor - an overview | ScienceDirect Topics». <https://www.sciencedirect.com/topics/engineering/point-contact-transistor> (zugegriffen Dez. 25, 2020).
- [10] «Point-contact transistor», *Wikipedia*. Dez. 22, 2020. Zugegriffen: Dez. 25, 2020. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Point-contact_transistor&oldid=995771298
- [11] «Point Contact Transistor PBS». <https://www.pbs.org/transistor/science/events/pointctrans.html> (zugegriffen Dez. 26, 2020).
- [12] «The Point-contact Transistor». <http://www.wylie.org.uk/technology/semics/pointcon.htm> (zugegriffen Dez. 26, 2020).
- [13] 忍者猫, «Point-contact_transistor.svg». Wikipedia, März 09, 2021. [Online]. Verfügbar unter: https://upload.wikimedia.org/wikipedia/commons/2/28/Point-contact_transistor.svg
- [14] «List of semiconductor materials», *Wikipedia*. Dez. 06, 2020. Zugegriffen: Dez. 25, 2020. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=List_of_semiconductor_materials&oldid=992737223
- [15] «Doping in Semiconductors - Tutorialspoint». https://www.tutorialspoint.com/semiconductor_devices/doping_in_semiconductor_devices.htm (zugegriffen Juni 28, 2021).
- [16] «Doping (semiconductor)», *Wikipedia*. Juli 26, 2021. Zugegriffen: Aug. 22, 2021. [Online]. Verfügbar unter: [https://en.wikipedia.org/w/index.php?title=Doping_\(semiconductor\)&oldid=1035621525](https://en.wikipedia.org/w/index.php?title=Doping_(semiconductor)&oldid=1035621525)
- [17] «Wafer», *Wikipedia*. März 10, 2021. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Wafer&oldid=209652615>
- [18] «Photomask», *Wikipedia*. Aug. 16, 2021. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: <https://en.wikipedia.org/w/index.php?title=Photomask&oldid=1039092640>

- [19] «Photolithography», *Wikipedia*. Aug. 30, 2021. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: <https://en.wikipedia.org/w/index.php?title=Photolithography&oldid=1041506530>
- [20] Peellden, *English: A photomask*. 2011. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:Semiconductor_photomask.jpg
- [21] «Transistor count», *Wikipedia*. Juni 17, 2021. Zugegriffen: Juni 28, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Transistor_count&oldid=1029095489
- [22] «7 nm process», *Wikipedia*. Juni 22, 2021. Zugegriffen: Juni 28, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=7_nm_process&oldid=1029939930
- [23] Inductiveload, *English: A diagram of the structure of an NPN BJT, showing the collector-emitter voltage (VCE), base-emitter voltage (VBE) and the collector, base and emitter currents and directions (IC, IB and IE)*. 2010. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:NPN_BJT_-_Structure_&_circuit.svg
- [24] «Diode», *Wikipedia*. Aug. 03, 2021. Zugegriffen: Aug. 23, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Diode&oldid=214462231>
- [25] «Raumladungszone», *Wikipedia*. Dez. 15, 2020. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Raumladungszone&oldid=206564648>
- [26] Inductiveload, *English: Diagram of the diffusion across a pn junction, with the resultant uncovered space charges, the electric field and the drift currents*. 2007. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:Pn_Junction_Diffusion_and_Drift.svg
- [27] K. Cepheiden, *English: Basic operation of a NPN type bipolar junction transistor*. 2009. Zugegriffen: Sep. 16, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:NPN_transistor_basic_operation.svg
- [28] «Transistor als Verstärker in Physik | Schülerlexikon | Lernhelfer». <https://www.lernhelfer.de/schuelerlexikon/physik-abitur/artikel/transistor-als-verstaerker> (zugegriffen Aug. 23, 2021).
- [29] «What is Point Contact Diode ? - Construction, Working & Applications», *Electronics Coach*, Aug. 01, 2017. <https://electronicscoach.com/point-contact-diode.html> (zugegriffen Aug. 27, 2021).
- [30] «Basic-Structure-of-point-contact-diode.jpg». electronicscoach. Zugegriffen: Aug. 27, 2021. [Online]. Verfügbar unter: <https://electronicscoach.com/point-contact-diode.html>
- [31] Jeri Ellsworth, *Make a Point Contact Transistor at home*, (Nov. 04, 2010). Zugegriffen: Dez. 26, 2020. [Online Video]. Verfügbar unter: <https://www.youtube.com/watch?v=vmotkjMSKnI&feature=youtu.be>
- [32] K. Franz, «Building Logic Gates with Transistors – Digilent Blog». <https://blog.digilentinc.com/building-logic-gates-with-transistors/> (zugegriffen Jan. 09, 2021).
- [33] «Small Logic Gates — The building blocks of digital circuits - Part 2», *Nuts and Volts Magazine*. <https://www.nutsvolts.com/magazine/article/small-logic-gates-spawn-big-dreams-part-2> (zugegriffen Jan. 09, 2021).
- [34] «How do logic gates work? - Explain that Stuff». <https://www.explainthatstuff.com/logicgates.html> (zugegriffen Dez. 22, 2020).
- [35] «Logic Gates». https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm (zugegriffen Aug. 30, 2021).

- [36] «Logic gate», Wikipedia. Aug. 28, 2021. Zugegriffen: Aug. 30, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Logic_gate&oldid=1041025379
- [37] «Logikgatter», Wikipedia. Feb. 14, 2021. Zugegriffen: Aug. 30, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Logikgatter&oldid=208823171>
- [38] «Universal Logic Gates - Technical Articles». <https://www.allaboutcircuits.com/technical-articles/universal-logic-gates/> (zugegriffen Aug. 30, 2021).
- [39] «IC power-supply pin», Wikipedia. März 22, 2021. Zugegriffen: Aug. 30, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=IC_power-supply_pin&oldid=1013695064
- [40] «American National Standards Institute», Wikipedia. Juli 14, 2021. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=American_National_Standards_Institute&oldid=1033594918
- [41] «International Electrotechnical Commission», Wikipedia. Jan. 07, 2020. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=International_Electrotechnical_Commission&oldid=195601071
- [42] «DIN-Norm», Wikipedia. Aug. 05, 2021. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=DIN-Norm&oldid=214519564>
- [43] Inductiveload, «NOT_ANSI_Labelled.svg». Wikipedia, Jan. 16, 2009. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/File:NOT_ANSI_Labelled.svg
- [44] SteveK, «Inverter_(Transistor).png». Wikipedia, März 13, 2006. Zugegriffen: Aug. 30, 2021. [Online]. Verfügbar unter: [https://de.wikipedia.org/wiki/Datei:Inverter_\(Transistor\).png](https://de.wikipedia.org/wiki/Datei:Inverter_(Transistor).png)
- [45] jjbeard, «AND_ANSI.svg». Wikipedia, Juni 02, 2006. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/File:AND_ANSI.svg
- [46] EBattleP, «TransistorANDgate.png». Wikipedia, Mai 03, 2014. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/File:TransistorANDgate.png>
- [47] Inductiveload, «OR_ANSI_Labelled.svg». Wikipedia, Jan. 16, 2009. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: Inductiveload
- [48] EBattleP, «Transistor_OR_Gate.png». Wikipedia, Apr. 11, 2014. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/File:Transistor_OR_Gate.png
- [49] jjbeard, «XOR_ANSI.svg». Wikipedia, Juni 02, 2006. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:XOR_ANSI.svg
- [50] sullystationtechnologies, «NPN Transistor XOR Gate Circuit | Sully Station Technologies», Sully Station Technologies. <http://sullystationtechnologies.com/npxorgate.html> (zugegriffen Jan. 31, 2021).
- [51] Inductiveload, «NAND_ANSI_Labelled.svg». Wikipedia, Jan. 16, 2009. Zugegriffen: Sep. 07, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:NAND_ANSI_Labelled.svg
- [52] «Universal Gates-NAND Gate», Electronics Hub, Juli 08, 2015. <https://www.electronicshub.org/universal-gates-nand-gate/> (zugegriffen Sep. 07, 2021).
- [53] Chris-martin, «File:Nor-gate-en.svg - Wikimedia Commons», Wikimedia, Mai 17, 2003. <https://commons.wikimedia.org/wiki/File:Nor-gate-en.svg> (zugegriffen Sep. 10, 2021).

- [54] 30px MovGP0, *Aufbau eines MC717 Schaltkreises*. 2006. Zugegriffen: Sep. 10, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:MC717_Circuit.svg
- [55] «Oszillator», *Wikipedia*. Juli 14, 2021. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Oszillator&oldid=213864483>
- [56] «Quarzoszillator», *Wikipedia*. Mai 14, 2021. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Quarzoszillator&oldid=211942308>
- [57] «Frequenz», *Wikipedia*. Apr. 01, 2021. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Frequenz&oldid=210437740>
- [58] «Piezoelektrizität», *Wikipedia*. Juli 27, 2021. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Piezoelektrizit%C3%A4t&oldid=214267956>
- [59] «What is Piezoelectricity?», *OnScale*. <http://https%253A%252F%252Fonscale.com%252Fpiezoelectricity%252Fwhat-is-piezoelectricity%252F> (zugegriffen Sep. 25, 2021).
- [60] «Dipol-Dipol-Kräfte». <https://www.chemie.de/lexikon/Dipol-Dipol-Kr%C3%A4fte.html> (zugegriffen Sep. 25, 2021).
- [61] «Are anodes positive or negative? – www.mbdanceapparel.com». <https://www.mbdanceapparel.com/2021/05/16/are-anodes-positive-or-negative/> (zugegriffen Sep. 25, 2021).
- [62] «Positive Rückkopplung», *Wikipedia*. Aug. 06, 2021. Zugegriffen: Sep. 25, 2021. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Positive_R%C3%BCckkopplung&oldid=214552243
- [63] «Schwingquarz», *Wikipedia*. Aug. 18, 2021. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Schwingquarz&oldid=214869898>
- [64] «NE555», *Wikipedia*. Okt. 28, 2020. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=NE555&oldid=204955251>
- [65] «555 timer IC», *Wikipedia*. Juli 01, 2021. Zugegriffen: Juli 05, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=555_timer_IC&oldid=1031381674
- [66] «555timer Schematic Diagram.png». <https://en.wikipedia.org/wiki/de:Benutzer:Stefan506>, Apr. 02, 2010. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:555_esquema.png
- [67] «555 timer block schematic». <https://en.wikipedia.org/wiki/de:Benutzer:Stefan506>, Apr. 02, 2010. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:555_esquema.png
- [68] «Schmitt Trigger Oscillator - Online Digital Electronics Course». <http://electronics-course.com/schmitt-trigger-oscillator> (zugegriffen Feb. 06, 2021).
- [69] «Exactly How Schmitt Trigger Oscillators Work - Technical Articles». <https://www.allaboutcircuits.com/technical-articles/exactly-how-schmitt-trigger-oscillators-work/> (zugegriffen Feb. 06, 2021).
- [70] «Reihenschaltung von Widerständen». <https://www.elektronik-kompendium.de/sites/slt/0110191.htm> (zugegriffen Sep. 25, 2021).

- [71] «555 Timer Animation». https://www.petervis.com/GCSE_Design_and_Technology_Electronic_Products/555-timer/555-timer-animation.html (zugegriffen Sep. 25, 2021).
- [72] «Long_tailed_pair.svg». Wikipedia, März 26, 2016. Zugegriffen: Aug. 29, 2021. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Datei:Long_tailed_pair.svg
- [73] «Stromspiegel», Wikipedia. Mai 06, 2020. Zugegriffen: Sep. 25, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Stromspiegel&oldid=199666474>
- [74] «Differenzverstärker», Wikipedia. Mai 27, 2020. Zugegriffen: Feb. 08, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Differenzverst%C3%A4rker&oldid=200358584>
- [75] Prof. Dr. W. Matthes, «Komparatoren». <http://controllersandpcs.de/>, März 03, 2011. Zugegriffen: März 03, 2021. [Online]. Verfügbar unter: https://www.google.ch/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiDg_Hg hpTvAhWR2KQKHUiuDAwQFjAKegQIARAD&url=http%3A%2F%2Fwww.controllersandpcs.de%2Flehrarchiv%2Fpdfs%2Felektronik%2Fkomparatoren_01.pdf&usg=AOvVaw01vsS8OhFl8rjagAWnGpCS
- [76] «Komparator (Analogtechnik)», Wikipedia. Jan. 10, 2020. Zugegriffen: Aug. 21, 2021. [Online]. Verfügbar unter: [https://de.wikipedia.org/w/index.php?title=Komparator_\(Analogtechnik\)&oldid=195676239](https://de.wikipedia.org/w/index.php?title=Komparator_(Analogtechnik)&oldid=195676239)
- [77] «Comparators». <https://www.sound-au.com/articles/comparators.htm> (zugegriffen Feb. 08, 2021).
- [78] «Flipflop», Wikipedia. Juni 29, 2021. Zugegriffen: Aug. 30, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Flipflop&oldid=213405125>
- [79] M. Frey, «Flipflop_SR2.svg». Wikipedia, Aug. 29, 2006. Zugegriffen: Aug. 30, 2021. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Datei:Flipflop_SR2.svg
- [80] «Flip Flops and Registers». <http://ianfinlayson.net/class/cpsc305/labs/09-flipflops> (zugegriffen Jan. 14, 2021).
- [81] «JK Flip-Flop Circuit Diagram, Truth Table and Working Explained». <https://circuitdigest.com/electronic-circuits/jk-flip-flop-truth-table-working/> (zugegriffen Mai 16, 2021).
- [82] «J-K Flip-Flop». <http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/jkflipflop.html> (zugegriffen Mai 16, 2021).
- [83] «Dual J-K Flip-Flops With Clear». Texas Instruments Incorporated, Dez. 1983. Zugegriffen: Aug. 06, 2021. [Online]. Verfügbar unter: <https://www.jameco.com/Jameco/Products/ProdDS/46412.pdf>
- [84] Cmglee, «JK flip-flop NAND.svg». Wikipedia, Apr. 20, 2014. Zugegriffen: Aug. 30, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:JK_flip-flop_NAND.svg
- [85] «Digital Registers - TutorialsPoint». https://www.tutorialspoint.com/computer_logical_organization/digital_registers.htm (zugegriffen Jan. 14, 2021).
- [86] «Registers | Computer Architecture Tutorial | Studytonight». <https://www.studytonight.com/computer-architecture/registers#> (zugegriffen Jan. 14, 2021).
- [87] «Register - What is Registers? Types of Registers», Computer Notes, Jan. 22, 2013. <https://ecomputernotes.com/fundamental/input-output-and-memory/what-is-registers-function-performed-by-registers-types-of-registers> (zugegriffen Sep. 05, 2021).
- [88] «The fetch-execute cycle - Computer systems - AQA - GCSE Computer Science Revision - AQA - BBC Bitesize». [The fetch-execute cycle - Computer systems - AQA - GCSE Computer Science Revision - AQA - BBC Bitesize](#).

- <https://www.bbc.co.uk/bitesize/guides/z7qqmsg/revision/7> (zugegriffen Dez. 22, 2020).
- [89] «Arithmetic logic unit», *Wikipedia*. Juli 18, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Arithmetic_logic_unit&oldid=1034139057
- [90] «Arithmetisch-logische Einheit», *Wikipedia*. Aug. 14, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Arithmetisch-logische_Einheit&oldid=214752002
- [91] «Floating-point unit», *Wikipedia*. Sep. 01, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Floating-point_unit&oldid=1041707835
- [92] «Gleitkommazahl», *Wikipedia*. Juli 15, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Gleitkommazahl&oldid=213902670>
- [93] «Half Adder in Digital Logic», *GeeksforGeeks*, Aug. 03, 2015. <https://www.geeksforgeeks.org/half-adder-in-digital-logic/> (zugegriffen Jan. 09, 2021).
- [94] «Adder (electronics)», *Wikipedia*. Aug. 16, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: [https://en.wikipedia.org/w/index.php?title=Adder_\(electronics\)&oldid=1039048908](https://en.wikipedia.org/w/index.php?title=Adder_(electronics)&oldid=1039048908)
- [95] «Binary Additions using Logic Gates», *101 Computing*, Jan. 04, 2018. <https://www.101computing.net/binary-additions-using-logic-gates/> (zugegriffen Jan. 09, 2021).
- [96] inductiveload, «Half_Adder.svg». Wikipedia, Aug. 05, 2006. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/File:Half_Adder.svg
- [97] «Volladdierer», *Wikipedia*. Feb. 27, 2020. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Volladdierer&oldid=197231706>
- [98] Sasmita, «Full Adder», *Electronics Post*, Mai 09, 2015. <https://electronicspost.com/full-adder/> (zugegriffen Jan. 09, 2021).
- [99] Iancovici, «Volladdierer mithilfe zwei Halbaddierer». Electrical Engineering Stack Exchange, Okt. 04, 2013. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: <https://electronics.stackexchange.com/questions/84321/counting-the-number-of-0s>
- [100] «Zweierkomplement», *Wikipedia*. Mai 21, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Zweierkomplement&oldid=212186832>
- [101] «Two's Complement». <https://www.tutorialspoint.com/two-s-complement> (zugegriffen Jan. 12, 2021).
- [102] «MC74HC595A - 8-Bit Serial-Input/Serial or Parallel-Output Shift Register With Latched 3-State Outputs». Semiconductor Components Industries, Dez. 2016. Zugegriffen: Aug. 07, 2021. [Online]. Verfügbar unter: <https://www.jameco.com/Jameco/Products/ProdDS/46105Datasheetb.PDF>
- [103] «Shift register», *Wikipedia*. Mai 06, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Shift_register&oldid=1021685343
- [104] «Schieberegister», *Wikipedia*. Juni 12, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Schieberegister&oldid=212893223>
- [105] TheGoodAndHolyLord, «4-Bit_SIPO_Shift_Register.svg». Wikipedia, Mai 05, 2021. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:4-Bit_SIPO_Shift_Register.svg

- [106] «Multiplizierer (Digitaltechnik)», *Wikipedia*. Juli 11, 2021. Zugegriffen: Sep. 06, 2021. [Online]. Verfügbar unter: [https://de.wikipedia.org/w/index.php?title=Multiplizierer_\(Digitaltechnik\)&oldid=213774730](https://de.wikipedia.org/w/index.php?title=Multiplizierer_(Digitaltechnik)&oldid=213774730)
- [107] «Binary Multiplier - Types & Binary Multiplication Calculator», *ELECTRICAL TECHNOLOGY*, Mai 25, 2018. <https://www.electricaltechnology.org/2018/05/binary-multiplier-types-binary-multiplication-calculator.html> (zugegriffen Aug. 07, 2021).
- [108] Covrl, English: *This 4-Bit mutiplier was built using three Full-Adders and 16 AND-Gates. Is was designed with KiCad and inspired by https://www.electricaltechnology.org/2018/05/binary-multiplier-types-binary-multiplication-calculator.html.* 2021. Zugegriffen: Sep. 09, 2021. [Online]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:4-Bit_Multiplier_Circuit.svg
- [109] «Division von Binärzahlen», *mathtreff-online*. <https://www.mathtreff-online.de/wissen/mathelexikon/division-von-binaerzahlen> (zugegriffen Sep. 10, 2021).
- [110] «Random-access memory», *Wikipedia*. Aug. 24, 2021. Zugegriffen: Sep. 10, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Random-access_memory&oldid=1040353353
- [111] «Dynamic random-access memory», *Wikipedia*. Aug. 16, 2021. Zugegriffen: Sep. 10, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Dynamic_random-access_memory&oldid=1039092059
- [112] «Static random-access memory», *Wikipedia*. Aug. 01, 2021. Zugegriffen: Aug. 03, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Static_random-access_memory&oldid=1036516643
- [113] «What is SRAM (static random access memory)? - Definition from WhatIs.com», *WhatIs.com*. <https://whatis.techtarget.com/definition/SRAM-static-random-access-memory> (zugegriffen Sep. 10, 2021).
- [114] «Program counter», *Wikipedia*. Okt. 28, 2020. Zugegriffen: März 05, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Program_counter&oldid=985902974
- [115] «DM74LS161A * DM74LS163A Synchronous 4-Bit Binary Counters». Fairchild Semiconductors, Aug. 1986. Zugegriffen: Juli 06, 2021. [Online]. Verfügbar unter: <https://www.jameco.com/Jameco/Products/ProdDS/46818.pdf>
- [116] «Zentraleinheit», *Wikipedia*. Feb. 21, 2018. Zugegriffen: Sep. 11, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Zentraleinheit&oldid=174243743>
- [117] «EEPROM», *Wikipedia*. Aug. 13, 2021. Zugegriffen: Sep. 13, 2021. [Online]. Verfügbar unter: <https://en.wikipedia.org/w/index.php?title=EEPROM&oldid=1038665786>
- [118] «What is EEPROM (Electrically Erasable Programmable Read-Only Memory)?» <https://www.computerhope.com/jargon/e/eeprom.htm> (zugegriffen Sep. 13, 2021).
- [119] «NodeMCU ESP32 - Joy-IT». <https://joy-it.net/en/products/SBC-NodeMCU-ESP32> (zugegriffen Sep. 11, 2021).
- [120] «NODEMCU ESP32». joy-it, Juni 29, 2021. Zugegriffen: Aug. 07, 2021. [Online]. Verfügbar unter: <http://www.joy-it.net/>
- [121] «Visual Studio Code - Code Editing. Redefined». <https://code.visualstudio.com/> (zugegriffen Okt. 11, 2021).
- [122] PlatformIO, «PlatformIO is a professional collaborative platform for embedded development», *PlatformIO*. <https://platformio.org> (zugegriffen Okt. 11, 2021).
- [123] «CAT28C16A 16K-Bit CMOS PARALLEL E2PROM». Catalyst Semiconductor, Inc., 1998. Zugegriffen: Aug. 06, 2021. [Online]. Verfügbar unter: <https://www.jameco.com/Jameco/Products/ProdDS/74691.pdf>

- [124] D. Workshop, «Shift Registers - 74HC595 & 74HC165 with Arduino», *DroneBot Workshop*, März 07, 2020. <https://dronebotworkshop.com/shift-registers/> (zugegriffen Okt. 12, 2021).
- [125] Electrical4U, «Serial in Parallel Out (SIPO) Shift Register | Electrical4U», <https://www.electrical4u.com/>. <https://www.electrical4u.com/serial-in-parallel-out-sipo-shift-register/> (zugegriffen Okt. 12, 2021).
- [126] «Arduino - ArduinoBoardNano». <https://www.arduino.cc/en/pmwiki.php?n>Main/ArduinoBoardNano> (zugegriffen Okt. 12, 2021).
- [127] Ben Eater, *Build an 8-bit decimal display for our 8-bit computer*, (März 18, 2017). Zugegriffen: Okt. 16, 2021. [Online Video]. Verfügbar unter: <https://www.youtube.com/watch?v=dLh1n2dErzE>
- [128] «Bus (Datenverarbeitung)», Wikipedia. Juni 17, 2021. Zugegriffen: Aug. 02, 2021. [Online]. Verfügbar unter: [https://de.wikipedia.org/w/index.php?title=Bus_\(Datenverarbeitung\)&oldid=213035054](https://de.wikipedia.org/w/index.php?title=Bus_(Datenverarbeitung)&oldid=213035054)
- [129] «Steckplatine», Wikipedia. Apr. 27, 2021. Zugegriffen: Aug. 02, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Steckplatine&oldid=211346845>
- [130] «Bus Transceiver uses Bidirectional Buffers to Send and Receive», Basic Electronics Tutorials, Juni 15, 2021. <https://www.electronics-tutorials.ws/combination/bus-transceiver.html> (zugegriffen Sep. 11, 2021).
- [131] «Digital buffer», Wikipedia. Aug. 11, 2021. Zugegriffen: Sep. 11, 2021. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Digital_buffer&oldid=1038323445
- [132] «Arduino - ArduinoBoardUno». <https://www.arduino.cc/en/Main/arduinoBoardUno>> (zugegriffen Okt. 01, 2021).
- [133] J. Schulze, «USB-Typen: Die Unterschiede zwischen USB-Typ-A, -B und -C», DEINHANDY - Magazin, Aug. 02, 2019. <https://blog.deinhandy.de/usb-typen-die-unterschiede-zwischen-usb-typ-a-b-und-c> (zugegriffen Okt. 01, 2021).
- [134] «Flags Register». https://www.csee.umbc.edu/courses/undergraduate/CMSC211/fall01/burt/tech_help/flags.html (zugegriffen Okt. 16, 2021).
- [135] «Flag register in 8085 microprocessor», GeeksforGeeks, Apr. 16, 2018. <https://www.geeksforgeeks.org/flag-register-8085-microprocessor/> (zugegriffen Okt. 16, 2021).
- [136] «Halbaddierer», Wikipedia. Okt. 21, 2020. Zugegriffen: Sep. 05, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Halbaddierer&oldid=204743895>
- [137] «Fibonacci-Folge», Wikipedia. Sep. 11, 2021. Zugegriffen: Okt. 17, 2021. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=Fibonacci-Folge&oldid=215494145>

7 Abbildungsverzeichnis

Abbildung 1: Bild des fertigen Prozessors.....	1
Abbildung 2: Nachbau des ersten Transistors.....	5
Abbildung 3: Modell Spitzentransistor.....	5
Abbildung 4: <i>Beispiel Fotomaske, Hersteller und Art unbekannt</i>	5
Abbildung 5: Funktionsweise NPN-Transistor.....	6
Abbildung 6: Raumladungszone zwischen dem P-Typ und N-Typ.....	6
Abbildung 7: Raumladungszonen Transistor.....	7
Abbildung 8: Punktkontaktdiode.....	7
Abbildung 9: NOT-Gatter Aufbau.....	8
Abbildung 10: AND-Gatter Aufbau aus Transistoren.....	9
Abbildung 11: OR-Gatter Aufbau aus Transistoren.....	9
Abbildung 12: XOR-Gatter aus Transistoren.....	9
Abbildung 13: NAND-Gatter aus Transistoren.....	10
Abbildung 14: NOR-Gatter aus Transistoren.....	10
Abbildung 15: Astabile Konfiguration des NE555 Timers.....	12
Abbildung 16: Schaltkreis des 555-Timers.....	12
Abbildung 17: Vereinfachter Schaltkreis des 555-Timers.....	12
Abbildung 18: Differenzverstärker.....	13
Abbildung 19: RS-Flipflop.....	14
Abbildung 20: JK-Flipflop.....	14
Abbildung 21: Halbaddierer Schaltkreis.....	16
Abbildung 22: Volladdierer Schaltkreis.....	17
Abbildung 23: 4bit Volladdierer mit Subtraktionsmethode.....	17
Abbildung 24: SIPO Schieberegister.....	18
Abbildung 25: Beispiel Binärmultiplikation.....	18
Abbildung 26: 4-Bit Multiplizierer aus Volladdierer und AND-Gatter, selbst designet.....	19
Abbildung 27: Beispiel Binärdivision.....	20
Abbildung 28: Struktur SRAM.....	21
Abbildung 29: Programmzähler Schaltkreis.....	23

Abbildung 30: BUS mit angeschlossenen Modulen.....	80
Abbildung 31: BUS Grossaufnahme.....	80
Abbildung 32: NAND-Gatter.....	81
Abbildung 33: 4-Bit Multiplizierer.....	81
Abbildung 34: Grossaufnahme des Prozessors.....	81
Abbildung 35: XOR-Gatter.....	82
Abbildung 36: Einheit zur Programmierung des RAMs.....	82
Abbildung 37: Ausgang.....	82
Abbildung 38: Dritte Version EEPROM Programmierer.....	83
Abbildung 39: Steckbrett zur manuellen Überprüfung des EEPROMs.....	83
Abbildung 40: Zweite Version EEPROM Programmierer.....	83
Abbildung 41: Taktgeber (Clock).....	84
Abbildung 42: Nachbau NOT-Gatter.....	84
Abbildung 43: Volladdierer.....	84
Abbildung 44: D-Type Flipflop.....	85
Abbildung 45: Programmzähler.....	85
Abbildung 46: Stromversorgung.....	85
Abbildung 47: Differenzverstärker.....	85
Abbildung 48: ALU.....	86
Abbildung 49: OR-Gatter.....	86
Abbildung 50: AND-Gatter.....	86
Abbildung 51: NOR-Gatter.....	87
Abbildung 52: Erste Version EEPROM Programmierer.....	87
Abbildung 53: RAM um Daten zu speichern.....	87
Abbildung 54: Register A.....	88
Abbildung 55: Register B.....	88

8 Tabellenverzeichnis

Tabelle 1: Zeitplan Praktische Arbeit.....	46
Tabelle 2: Protokoll Maturaarbeit.....	50
Tabelle 3: Liste der Bauteile.....	52
Tabelle 4: Wahrheitstabelle NOT-Gatter.....	53
Tabelle 5: Wahrheitstabelle AND-Gatter.....	53
Tabelle 6: Wahrheitstabelle OR-Gatter.....	53
Tabelle 7: Wahrheitstabelle XOR-Gatter.....	54
Tabelle 8: Wahrheitstabelle NAND-Gatter.....	54
Tabelle 9: Wahrheitstabelle NOR-Gatter.....	54
Tabelle 10: Wahrheitstabelle RS-Flipflop.....	54
Tabelle 11: Wahrheitstabelle Halbaddierer.....	55
Tabelle 12: Wahrheitstabelle Volladdierer.....	55
Tabelle 13: Befehlslinien Prozessor.....	56
Tabelle 14: Instruktion IA.....	57
Tabelle 15: Instruktion IB.....	57
Tabelle 16: Instruktion ADD.....	58
Tabelle 17: Instruktion SUB.....	58
Tabelle 18: Instruktion OA.....	58
Tabelle 19: Instruktion OB.....	58
Tabelle 20: Instruktion SA.....	59
Tabelle 21: Instruktion SB.....	59
Tabelle 22: Instruktion SADD.....	60
Tabelle 23: Instruktion SSUB.....	60
Tabelle 24: Instruktion O.....	61
Tabelle 25: Instruktion JUMP.....	61
Tabelle 26: Instruktion HALT.....	61
Tabelle 27: Instruktion LOADA.....	62
Tabelle 28: Instruktion LOADB.....	62
Tabelle 29: Programm für eine einfache Addition.....	63

Tabelle 30: Programm für Fibonacci-Folge.....64

9 Weiterführende Literatur

- [1] W. Schlagenhauf, *Allgemeinbildung Technik für Dummies*. Wiley-VCH, 2020.
Zugegriffen: Aug. 02, 2021. [Online]. Verfügbar unter:
<https://www.buchhaus.ch/de/buecher/fachbuecher/technik/tabellen/detail/ISBN-9783527714971/Schlagenhauf-Wilfried/Allgemeinbildung-Technik-f%C3%BCr-Dummies>

10 Selbständigkeitserklärung

Ich habe diese Maturaarbeit unter Benutzung der angeführten Quellen selbstständig entworfen, abgefasst, gestaltet und geschrieben.

Hofackerstrasse 3, 8722 Kaltbrunn, 27.10.2021

X

Dominik Schwaiger
Autor

11 Anhang

11.1 Zeitplan²⁴

<u>Woche</u>	<u>Plan</u>
1-4	eine erste Übersicht erstellen
4-8	beginnen, die Theorie zusammenzufassen
8-12	die Zusammenfassung der Theorie finalisieren nötigen Bauteile einkaufen
12-14	beginnen, die einzelnen Module zu bauen
14	Konzept zur Herstellung eines Transistors erreicht haben Planung des Aufbaus des 8-Bit-Prozessors abgeschlossen haben
14-18	Theorie weiter erschliessen, Bauteile einkaufen
18-23	weiter an Modulen bauen
23-25	Transistor Bauplan vervollständigen
25-30	Transistor bauen
30	Transistor fertig gebaut haben teilweiser Aufbau des Prozessors erreicht haben
30-35	Beginn an dem schriftlichen Teil der Matura Beginn des Baus des Hauptprozessors und die Verbindung mit den Modulen
35	Abgabe des Probekapitels Inhaltsverzeichnis mit Korrektur
35-40	Fertigung letzter Module Vervollständigung des Transistors weiter an schriftlicher Matura arbeiten
41	Fertigstellung der schriftlichen Maturaarbeit
42	Korrektion und Druck der Maturaarbeit
43	Abgabe Maturaarbeit

Tabelle 1: Zeitplan Praktische Arbeit

²⁴ Gelb markierte Bereiche signalisieren festgelegte Termine in der Projektvereinbarung.

11.2 Protokoll

<u>Datum</u>	<u>Eintrag</u>
22.12.2020	<p>Ich versuche, den grundlegenden Aufbau eines Prozessors zu verstehen Wichtige Daten in Kalender eingetragen Transistoren genauer angeschaut Ideen</p> <ul style="list-style-type: none"> • Bau Transistoren/weitere Module zuerst in einem Programm planen <ul style="list-style-type: none"> ◦ https://library.io/ ◦ https://workspace.circuitmaker.com/ • Transistor bauen (Silikon vielleicht schon fertig/vielleicht selber machen) • Alle weiteren Bauteile immer aus den vorherigen Bauen, aber die Vorherigen dürfen von einer Fabrik sein • Sobald ich ein Bauteil selber gebaut habe, darf ich es von einer Fabrik gebrauchen • Arbeit gut mit Bilder Dokumentieren • In einer Applikation für Notizen die Theorie gut Zusammenfassen
27.11.2020	Notizen etwas erweitert
09.01.2021	An Transistoren weitergearbeitet Mit Logikgatter angefangen Binäre Mathematik angeschaut
12.01.2021	Einkaufsliste erstellt Additionstheorie aufgeschrieben
15.01.2021	Produkte eingekauft
18.01.2021	Bauteile eingekauft Widerstandstheorie Theorie aufgeschrieben
24.01.2021	Erkannt, dass die Elektronischen Bauteile zu klein sind, ich muss sie nochmals kaufen, aber in einem grösseren Format
31.01.2021	UND- und NICHT-Gatter nachgebaut UND-Gatter brauchte viel Arbeit, da der Widerstand zu GND am letzten Transistor zu gross war und daher das Indikator-licht ständig brannte. Problem gelöst indem mit dem Transistor experimentiert wurde
01.02.2021	Jedes Logikgatter hergestellt XOR-Gatter hat Probleme und funktioniert nicht richtig Problem gelöst
08.02.2021	Gute Theorie zu Komparator gefunden, ist aber sehr kompliziert, verstehe sie noch nicht.
14.02.2021	Testen der Komponenten
01.03.2021	Versuche immer noch, den Komparator zu verstehen, frage im Code Camp nach Hilfe
14.03.2021	Differenzverstärker gebaut <ul style="list-style-type: none"> • Funktioniert mehr oder weniger wie erwartet, da er nicht sehr stark verstärkt
05.04.2021	Debuggen des Flipflops Flipflop funktioniert nicht richtig Aufbau gleich wie in Schaltkreis, aber die Daten, welche ich bekomme,

	scheinen zufällig zu sein Versuche, an einer anderen Komponente zu arbeiten und komme später auf den Flipflop zurück Beginn der Arbeit am Volladdierer
12.04.2021	Volladdierer fertig gebaut Schalter beim Flipflop funktioniert nicht wie angenommen, sollte zwischen GND und 5V wechseln, aber wechselt nur zwischen 5V und 0V ALU fertig gebaut
20.04.2021	Flipflop funktioniert nach einigen Verbesserungen Angefangen, an der schriftlichen Arbeit zu arbeiten
26.04.2021	Nachforschung zu Kristall Oszillator Spannungsverstärker für Oszillator gekauft Planung des genauen Aufbaus des Prozessors
17.05.2021	An der Theorie für den Zähler und am generellen Aufbau gearbeitet
05.07.2021	Falsche Bauteile für Schmidt-Trigger bestellt
20.07.2021	Bau des Registers Programmzähler funktioniert nur manchmal, scheint auf hoher Geschwindigkeit aber richtig zu funktionieren Ein Register wurde fertig gebaut
26.07.2021	Probleme mit der Spannung etwas gelöst: https://electronics.stackexchange.com/questions/85599/voltage-is-reduced-across-chained-breadboards Bau des zweiten Registers Die Register haben ein Problem mit der Speicherung der Daten Auf dem Bus befindet sich andauern eine zu grosse Spannung Bus Problem gelöst, indem er über kleine Widerstände mit GND verbunden wurde ALU gibt unerwartete Ergebnisse aus
27.07.2021	ALU wurde fertig gebaut Bau eines einfachen manuellen Kontrollmoduls Gebrauch eines Schalters für den Subtraktionsbefehl, da ansonsten eine zu grosse Spannung auf dem Bus liegt
03.08.2021	Das Register für die RAM Adresse wurde falsch verbunden Bus wurde falsch an Register angeschlossen, Fehler wird aber behalten, da Register einfach umgekehrt mit dem RAM verbunden werden können
04.08.2021	RAM gibt zu viel Spannung an den BUS ab, muss Gebrauch von einem Sendeempfänger machen RAM Register scheint nicht richtig zu funktionieren RAM Register ist mit Sendeempfänger verbunden und schaltet daher gleichzeitig aus, ich muss eine Verzögerung oder etwas ähnliches einbauen Verzögerung funktioniert nicht, kann beide Module nicht verbinden Problem gelöst, Verzögerung war nicht das Problem, die Kabel waren einfach falsch verbunden RAM funktioniert wie erwartet, aber der Sinn des CS Signals verstehe ich nicht Ich verbinde einfach CS mit GND und hoffe, das dies nichts verändert RAM funktioniert wie angenommen
06.08.2021	Zähler funktioniert

	EEPROMs sind zu 50% verbunden Problem mit dem RAM für Instruktionen Wie verbindet man zwei 4Bit RAMs so dass sie als ein 8Bit RAM funktionieren? Es geht nicht! Muss 8Bit RAM kaufen und später ersetzen
07.08.2021	Erste Version des Prozessors NodeMCU funktioniert nicht um EEPROMs zu programmieren, muss es mit Arduino Nano probieren
08.08.2021	Prozessor v1.0 fertig EEPROM Programmier-Programm v1.0 fertig EEPROMs programmiert Test Programm programmiert EEPROMs scheinen nicht zu funktionieren
04.10.2021	Computer funktioniert nicht Überall auf den EEPROMs befinden sich 1 EEPROM ist Programmierbar EEPROM gibt nicht die erwarteten Werte aus Besser Potentiometer in Prozessor verbaut
05.10.2021	Überprüfen des Programms Vielleicht ein Fehler im Programm gefunden Daten werden richtig in den Array geschrieben Testen ob die EEPROMs jetzt funktionieren (tun sie nicht) Immer noch alles 1 Fehler bei Pins gefunden Falsches Datenblatt gebraucht für die Pins Pins verbessert EEPROM immer noch falsch programmiert
06.10.2021	vielleicht wurde die Zeitabfolge falsch geschrieben? Datenblatt durchgelesen und Verständnis erweitert, die Zeitabfolge sollte kein Problem spielen Problem könnte von «Data Polling» kommen HALT Befehl hinzugefügt EEPROM funktioniert immer noch nicht Implementierung eines WE gesteuerten Schreibvorganges funktioniert immer noch nicht WE Signal zu einem sehr kurzen Puls ändern funktioniert immer noch nicht anderer EEPROM testen, funktioniert auch nicht Spannung testen, könnte zu klein sein
07.10.2021	Transistoridee implementieren Neues Brett ausprobieren Code verbessern, so das RX/TX Pins kein Problem darstellen funktioniert immer noch nicht grösserer Zeitabstand einprogrammieren funktioniert auch nicht Spannung ist zu hoch bei 0V einige Möglichkeit bleibt mit Schieberegister

	Pin an EEPROM abgebrochen, muss neu angelötet werden
08.10.2021	Programm für Arduino Nano mit Schieberegister programmieren Programm ist zu gross, muss effizienter geschrieben werden funktioniert immer noch nicht scheint so als ob Schieberegister nicht funktionieren Verzögerung zu Schieberegister hinzugefügt funktioniert immer noch nicht extra Bit zu Schieberegister hinzugefügt Daten wieder manuell überprüft, scheint so als ob sie dieses mal stimmen Daten mit Programm überprüft, stimmen nicht
09.10.2021	Testprogramm scheint fehlerhaft zu sein einzelne Bits falsch, vielleicht sind diese Kabel nicht richtig verbunden? Gebrauch der Funktion pinMode anstelle von digitalWrite in der Überprüffunktion Problem gefunden EEPROMs werden richtig programmiert Ausgaberegister gebaut RAM für Instruktionen von vier auf acht Bit verbessert
10.10.2021	Zähler scheint falsch zu sein erstes Bit direkt mit dem Taktzähler verbunden, sollte aber erst das Zweite sein EEPROM verarbeitet Zählerdaten verkehrt, verbessere dies
13.10.2021	Zähler aus Testzwecken mit Jumperkabel mit dem Taktgeber verbunden EEPROM verkehrt verbunden verbessere dies mithilfe einer Funktion im Programm Prozessor funktioniert nicht, überprüfe einzelne Module
14.10.2021	Programmzähler Verbindung funktioniert nicht richtig verbessert und manuelles clear-Kabel hinzugefügt Programmzähler funktioniert immer noch nicht letztes Bit geht nicht war beim Sendeempfänger mit GND verbunden nur HALT Befehl testen funktioniert nicht
15.10.2021	Ich höre auf, an dem Prozessor zu arbeiten und konzentriere mich auf das Schreiben der Matura, versuche, das Problem vielleicht in der letzten Woche noch zu lösen
16.10.2021	Daten im RAM falsch abgespeichert, muss Programm nochmals verbessern
17.10.2021	Maturaarbeit vollständig überprüft und verbessert
19.10.2021	Start des Baus des Multiplikationsmodul
20.10.2021	Probleme beim Bau mehrmals kontrolliert, mehrere Kabel wurden falsch verbunden Multiplikation funktioniert

Tabelle 2: Protokoll Maturaarbeit

11.3 Liste der Bauteile

<u>Art des Bauteils</u>	<u>Name des Ladens</u>	<u>Artikelnummer</u>
Diode	Amazon	A712000110UK
Kabel	Amazon	B07G72DRKC
Kabel	Conrad	98001c421
Drucktaster	Conrad	2050000224997
Transistor	Conrad	2050000033735
Transistor	Conrad	2050001284112
Multimeter	Conrad	4260625440308
Steckplatine	Conrad	4250236815305
Komparator	Conrad	2050005490359
Potentiometer	Conrad	2050000031533
Kabel	Conrad	2050004592399
Kabel	Conrad	4016139254937
Potentiometer	Grieder Bauteile	2322-390-66535 232239066535
Potentiometer	Grieder Bauteile	2322.380.66514 232238066514
RAM	Grieder Bauteile	HM628128ALP-10
Volladdierer	Grieder Bauteile	SN74283N
AND-Gatter	Grieder Bauteile	74F08PC
Shift-Register	Jameco	74HC595
Shift-Register	Jameco	74HC595N
RAM	Jameco	62256LP-15
NAND-Gatter	Jameco	CD4011B
Komparator	Jameco	74LS682
Schalter	Jameco	38820
Schalter	Jameco	38842
EEPROM	Jameco	28C16A-25
Volladdierer	Jameco	74LS283
Flipflop	Jameco	74LS273
Transceiver	Jameco	74LS245
RAM	Jameco	74189
Register	Jameco	74LS173

Zähler	Jameco	74LS161
Daten-Auswähler	Jameco	74LS157
Dekodierer	Jameco	74LS139
Dekodierer	Jameco	74LS138
XOR-Gatter	Jameco	74LS86
Flipflop	Jameco	74LS107
OR-Gatter	Jameco	74LS32
AND-Gatter	Jameco	74LS08
Hexadezimal-Invertierer	Jameco	74LS04
NOR-Gatter	Jameco	74LS02
NAND-Gatter	Jameco	74LS00
555-Timer	Jameco	LM555CN-R
Potentiometer	Mouser	858-P162G100QRA20A13
Steckplatine	Mouser	510-GS-100
Steckplatine	Mouser	424-240-131
LED-Anzeigen	Mouser	424-240-131
Kabel	ELV	4023392588310
Widerstände	ELV	5410329238391
Schiebeschalter	ELV	4023392313950

Tabelle 3: Liste der Bauteile

11.4 Wahrheitstabellen

11.4.1 Logikgatter²⁵

NOT

<u>Eingang</u>	<u>Ausgang</u>
LOW	HIGH
HIGH	LOW

Tabelle 4: Wahrheitstabelle NOT-Gatter

AND

<u>Eingang</u>		<u>Ausgang</u>
<u>A</u>	<u>B</u>	<u>Q</u>
LOW	LOW	LOW
LOW	HIGH	LOW
HIGH	LOW	LOW
HIGH	HIGH	HIGH

Tabelle 5: Wahrheitstabelle AND-Gatter

OR

<u>Eingang</u>		<u>Ausgang</u>
<u>A</u>	<u>B</u>	<u>Q</u>
LOW	LOW	LOW
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	HIGH

Tabelle 6: Wahrheitstabelle OR-Gatter

XOR

<u>Eingang</u>		<u>Ausgang</u>
<u>A</u>	<u>B</u>	<u>Q</u>
LOW	LOW	LOW
LOW	HIGH	HIGH
HIGH	LOW	HIGH

²⁵ Wahrheitstabellen nach [37].

<u>Eingang</u>		<u>Ausgang</u>
HIGH	HIGH	LOW

Tabelle 7: Wahrheitstabelle XOR-Gatter

NAND

<u>Eingang</u>		<u>Ausgang</u>
<u>A</u>	<u>B</u>	<u>Q</u>
LOW	LOW	HIGH
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	LOW

Tabelle 8: Wahrheitstabelle NAND-Gatter

NOR

<u>Eingang</u>		<u>Ausgang</u>
<u>A</u>	<u>B</u>	<u>Q</u>
LOW	LOW	HIGH
LOW	HIGH	LOW
HIGH	LOW	LOW
HIGH	HIGH	LOW

Tabelle 9: Wahrheitstabelle NOR-Gatter

11.4.2 RS-Flipflop²⁶

<u>S</u>	<u>R</u>	<u>Q</u>
HIGH	HIGH	HIGH oder LOW (vorherig gespeicherter Wert)
HIGH	LOW	1
LOW	HIGH	0
LOW	LOW	$Q = \bar{Q} = 1$ (Fehler)

Tabelle 10: Wahrheitstabelle RS-Flipflop

²⁶ Wahrheitstabelle nach [78].

11.4.3 Halbaddierer²⁷

<u>Eingang</u>		<u>Ausgang</u>	
<u>A</u>	<u>B</u>	<u>C</u>	<u>S</u>
LOW	LOW	LOW	LOW
LOW	HIGH	LOW	HIGH
HIGH	LOW	LOW	HIGH
HIGH	HIGH	HIGH	LOW

Tabelle 11: Wahrheitstabelle Halbaddierer

11.4.4 Volladdierer²⁸

<u>Eingang</u>			<u>Ausgang</u>	
<u>A</u>	<u>B</u>	<u>C_{in}</u>	<u>C_{out}</u>	<u>S</u>
LOW	LOW	LOW	LOW	LOW
LOW	LOW	HIGH	LOW	HIGH
LOW	HIGH	LOW	LOW	HIGH
LOW	HIGH	HIGH	HIGH	LOW
LOW	HIGH	HIGH	HIGH	LOW
HIGH	LOW	HIGH	HIGH	LOW
HIGH	HIGH	LOW	HIGH	LOW
HIGH	HIGH	HIGH	HIGH	HIGH

Tabelle 12: Wahrheitstabelle Volladdierer

²⁷ Wahrheitstabelle nach [136].

²⁸ Wahrheitstabelle nach [97].

11.5 Instruktionen Prozessor

11.5.1 Befehlslinien

<u>Linie</u>	<u>Abkürzung</u>	<u>Beschreibung</u>
1	AO	Ausgang Register A Ein
2	AI	Eingang Register A Ein
3	BO	Ausgang Register B Ein
4	BI	Eingang Register B Ein
5	ALUO	Ausgang ALU Ein
6	PCL	Programmzähler Laden Ein
7	PCC	Neustart Programmzähler
8	PCO	Ausgang Programmzähler Ein
9	ALUS	ALU Subtraktion Ein (B-A)
10	PCD	Programmzähler Aus
11	DRAMO	RAM mit Daten Ausgang Ein
12	LDRAM	RAM mit Daten Adresse Laden Ein
13	LDRAMDR	RAM mit Daten Register für Daten Laden Ein
14	ODRAMRD	RAM mit Daten Register für Daten Ausgang Ein
15	CDRAMDB	RAM mit Daten Register für Daten mit BUS verbinden Ein
16	DRAMW	RAM mit Daten Schreiben Ein
17	IRAMWLA	RAM mit Instruktionen Adresse Laden Ein
18	SBUS	BUS Status Speichern (Output Register)

Tabelle 13: Befehlslinien Prozessor

11.5.2 Instruktionen

IA

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>															<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9	10	11	12	13	14	15	16	17	18					Erklärung
0 0 0 0 0 0 0 1	1 0 0	0 0 0 0 0 0 0 1	0	0	0	0	0	0	0	0	1	0	0	Lade Instruktion in Instruktionsregister			
0 0 0 0 0 0 0 1	0 1 0	0 0 0 0 0 0 0 1	0	0	0	0	0	0	0	1	1	1	0	Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus			
0 0 0 0 0 0 0 1	1 1 0	0 1 0 0 0 0 0 0	0	0	0	0	0	0	0	0	1	1	0	Speichere Daten in Register A			

Tabelle 14: Instruktion IA

IB

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>															<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9	10	11	12	13	14	15	16	17	18					Erklärung
0 0 0 0 0 0 1 0	1 0 0	0 0 0 0 0 0 0 1	0	0	0	0	0	0	0	0	0	0	1	Lade Instruktion in Instruktionsregister			
0 0 0 0 0 0 1 0	0 1 0	0 0 0 0 0 0 0 1	0	0	0	0	0	0	0	1	1	1	0	Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus			
0 0 0 0 0 0 1 0	1 1 0	0 0 0 1 0 0 0 0	0	0	0	0	0	0	0	0	1	1	0	Speichere Daten in Register B			

Tabelle 15: Instruktion IB

ADD

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>															<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9	10	11	12	13	14	15	16	17	18					Erklärung
0 0 0 0 0 0 1 1	1 0 0	0 0 0 0 0 0 0 1	0	0	0	0	0	0	0	0	0	0	1	Lade Instruktion in Instruktionsregister			
0 0 0 0 0 0 1 1	0 1 0	0 0 0 0 0 0 0 1	0	0	0	0	0	0	0	1	1	1	0	Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus			
0 0 0 0 0 0 1 1	1 1 0	0 0 0 0 1 0 0 0	0	0	0	0	1	0	0	0	0	0	0	Gib ALU Daten aus			

Tabelle 16: Instruktion ADD

SUB

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>														<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18	<u>Erklärung</u>											
0 0 0 0 0 1 0 0	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	1 0	Lade Instruktion in Instruktionsregister										
0 0 0 0 0 1 0 0	0 1 0	0 0 0 0 0 0 0 1	0 0 1 1 1 1 0 0	0 0	0 0 1 1 1 1 0 0	Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus										
0 0 0 0 0 1 0 0	1 1 0	0 0 0 0 1 0 0 0	1 0 0 0 0 0 0 0	0 1	0 1	Gib ALU Daten aus										

Tabelle 17: Instruktion SUB

OA

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>														<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18	<u>Erklärung</u>											
0 0 0 0 0 1 0 1	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	1 0	Lade Instruktion in Instruktionsregister										
0 0 0 0 0 1 0 1	0 1 0	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 1	0 1	Gib Daten in Register A aus										

Tabelle 18: Instruktion OA

OB

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>														<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18	<u>Erklärung</u>											
0 0 0 0 0 1 1 0	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	1 0	Lade Instruktion in Instruktionsregister										
0 0 0 0 0 1 1 0	0 1 0	0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0	0 1	0 1	Gib Daten in Register B aus										

Tabelle 19: Instruktion OB

SA

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>														<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18	<u>Erklärung</u>											
0 0 0 0 0 1 1 1	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	1 0	Lade Instruktion in Instruktionsregister										

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>
0 0 0 0 0 1 1 1	0 1 0	0	0	0	0	0	0	0	1	0	0	1	1	1 0 0
0 0 0 0 0 0 1 1 1	1 1 0	0	0	0	0	0	0	0	0	0	0	0	1	0 1 1 0
0 0 0 0 0 0 1 1 1	0 0 1	1	0	0	0	0	0	0	0	0	0	0	0	1 1 0 0

Tabelle 20: Instruktion SA

SB

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>				
1 2 3 4 5 6 7 8	1 2 3	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17 18
0 0 0 0 1 0 0 0	1 0 0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1 0
0 0 0 0 1 0 0 0	0 1 0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0	0	0 0
0 0 0 0 1 0 0 0	1 1 0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0 0
0 0 0 0 1 0 0 0	0 0 1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0 0

Tabelle 21: Instruktion SB

SADD

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>				
1 2 3 4 5 6 7 8	1 2 3	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17 18
0 0 0 0 1 0 0 1	1 0 0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1 0
0 0 0 0 1 0 0 1	0 1 0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0	0	0 0
0 0 0 0 1 0 0 1	1 1 0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0 0

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>			
0 0 0 0 1 0 0 1	0 0 1	0 0 1 0 0 0 0 0	0 0 0 0 0 0 1 1	0 0													Lade Daten vom ALU in den RAM für Daten

Tabelle 22: Instruktion SADD

SSUB

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>			
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18													Erklärung
0 0 0 0 1 0 1 0	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1													Lade Instruktion in Instruktionsregister
0 0 0 0 1 0 1 0	0 1 0	0 0 0 0 0 0 0 1	0 0 1 1 1 1 0 0	0 0													Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus
0 0 0 0 1 0 1 0	1 1 0	0 0 0 0 0 0 0 0	0 0 0 1 0 1 1 0	0 0													Lade die Daten im RAM als Adresse in den Daten-RAM
0 0 0 0 1 0 1 0	0 0 1	0 0 1 0 0 0 0 0	1 0 0 0 0 0 1 1	0 0													Lade Daten vom ALU in den RAM für Daten

Tabelle 23: Instruktion SSUB

O

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>			
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18													Erklärung
0 0 0 0 1 0 1 1	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1													Lade Instruktion in Instruktionsregister
0 0 0 0 1 0 1 1	0 1 0	0 0 0 0 0 0 0 1	0 0 1 1 1 1 0 0	0 0													Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus
0 0 0 0 1 0 1 1	1 1 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1	0 0													Gib Daten vom RAM aus

Tabelle 24: Instruktion O

JUMP

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>			
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18													Erklärung

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>										<u>Erklärung</u>
0 0 0 0 1 1 0 0	1 0 0	0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0	1	Lade Instruktion in Instruktionsregister							
0 0 0 0 1 1 0 0	0 1 0	0 0 0 0 0 0 0 0 1	0 0 1 1 1 1 0 0	0 0	Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus							
0 0 0 0 1 1 0 0	1 1 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0	Mache nichts ²⁹							
0 0 0 0 1 1 0 0	0 0 1	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0	Mache nichts							
0 0 0 0 1 1 0 0	1 0 1	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0	Mache nichts							
0 0 0 0 1 1 0 0	0 1 1	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0	Mache nichts							
0 0 0 0 1 1 0 0	1 1 1	0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 1 1 0	0 0	Lade Daten vom RAM in den Programmzähler							

Tabelle 25: Instruktion JUMP

HALT

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>										<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18	Erklärung							
0 0 0 0 1 1 0 1	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1 0	Lade Instruktion in Instruktionsregister							
0 0 0 0 1 1 0 1	0 1 0	0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0	0 0	Schalte Programmzähler aus							

Tabelle 26: Instruktion HALT

LOADA

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>										<u>Erklärung</u>
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18	Erklärung							
0 0 0 0 1 1 1 1	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1 0	Lade Instruktion in Instruktionsregister							
0 0 0 0 1 1 1 1	0 1 0	0 0 0 0 0 0 0 1	0 0 1 1 1 1 0 0	0 0	Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus							
0 0 0 0 1 1 1 1	1 1 0	0 0 0 0 0 0 0 0	0 0 0 1 0 1 1 0	0 0	Lade die Daten im RAM als Adresse in den Daten-RAM							

²⁹ Dies hat den Grund, dass gewartet werden muss, bis der Zähler sein Maximum erreicht, da es ansonsten Probleme geben könnte, da der Programmzähler Wert geändert wird.

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>		
0 0 0 0 1 1 1 1	0 0 1	0 1 0 0 0 0 0 0	0 0 1 1 0 1 1 1	0 0	Speichere Daten an dieser Adresse im Register A											

Tabelle 27: Instruktion LOADA

LOADB

<u>Instruktion Nr.</u>	<u>Zähler</u>	<u>Befehlslinie</u>												<u>Erklärung</u>		
1 2 3 4 5 6 7 8	1 2 3	1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18	Erklärung											
0 0 0 1 0 0 0 0	1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1 0	Lade Instruktion in Instruktionsregister											
0 0 0 1 0 0 0 0	0 1 0	0 0 0 0 0 0 0 1	0 0 1 1 1 1 0 0	0 0	Lade Programmzähler Wert als Adresse in RAM für Daten und gibt Daten aus											
0 0 0 1 0 0 0 0	1 1 0	0 0 0 0 0 0 0 0	0 0 0 1 0 1 1 0	0 0	Lade die Daten im RAM als Adresse in den Daten-RAM											
0 0 0 1 0 0 0 0	0 0 1	0 0 0 1 0 0 0 0	0 0 1 1 0 1 1 1	0 0	Speichere Daten an dieser Adresse im Register A											

Tabelle 28: Instruktion LOADB

11.6 Programme für den Computer

11.6.1 Einfache Addition

<u>Programmzähler Wert</u>	<u>Instruktion</u>	<u>ursprünglicher Wert im RAM</u>	<u>Kurze Erklärung</u>
0000 0001	0000 0001	0000 0110	Speichere sechs im Register A ab
0000 0010	0000 0010	0000 1001	Speichere neun im Register B ab
0000 0011	0000 0011	XXXX XXXX ³⁰	Addiere 6 zu 9 und gib es aus
0000 0100	0000 1101	XXXX XXXX	Halte den Prozessor an

Tabelle 29: Programm für eine einfache Addition

11.6.2 Fibonacci-Folge [137]

<u>Programmzähler Wert</u>	<u>Instruktion</u>	<u>ursprünglicher Wert im RAM</u>	<u>Kurze Erklärung</u>
0000 0001	0000 0001	0000 0000	Speichere null im Register A ab
0000 0010	0000 0010	0000 0001	Speichere eins im Register B ab
0000 0011	0000 1001	0000 0001	Addiere die Werte in den Register und speichere das Resultat bei der Adresse 0000 0001 ab
0000 0100	0000 1111	0000 0001	Lade den Wert an der Adresse 0000 0001 in das Register A
0000 0101	0000 0101	XXXX XXXX	Gib den Wert im Register A aus
0000 0110	0000 1001	0000 0010	Addiere die Werte in den Register und speichere das Resultat bei der Adresse 0000 0010 ab
0000 0111	0001 0000	0000 0010	Lade den Wert an der Adresse 0000 0010 in das Register B
0000 1001	0000 0110	XXXX XXXX	Gib den Wert im Register B aus

³⁰ Ein X bedeutet, dass der Wert nicht beachtet wird.

0000 1010	0000 1100	0000 0010	Beginne von vorne (Lade Wert 0000 0010 in Programmzähler)
-----------	-----------	-----------	--

Tabelle 30: Programm für Fibonacci-Folge

11.7 Code³¹

```
/*

```

How I'll do it:

shift-registers can save 2*8 bits of data.

Nano can output/input 9 bits of data. (TX/RX used, 13 Data pins, 2 pins for shift registers => 9 bits)

shift-registers save address (11 bits)

nano saves data (8 bits) so it can read/write

WE and OE is also controlled by the shift-registers (2 bits)

CE is controlled by the nano (1 bit)

so data in shift registers looks like this:

ARDUINO NANO | [0][0][0][WE][OE][A10][A9][A8][A7][A6][A5][A4][A3][A2][A1][A0] |

EEPROM

```
*/

```

```
#include <Arduino.h>
```

```
#define SHIFT_CLOCK 3 // T
#define SHIFT_DATA 13 // T
```

```
#define IO0 12 // T
#define IO1 8 // T
#define IO2 9 // T
#define IO3 11 // T
#define IO4 4 // T
#define IO5 10 // T
#define IO6 5 // T
#define IO7 2 // T
```

```
#define CE 7 // T
```

```
#define EEPROM_NUMBER 3
#define EEPROM_ADDRESS_SIZE 2048U
```

```
#define EEPROM_DATA_TYPE unsigned char
```

```
#define EEPROM_ADDRESS_TYPE unsigned short int
```

```
#define DEBUG_MODE 0 // 0 = off, 1 = print to serial, 2 = single write/read, 3 = Test Voltage, 4 = shift-register, 5 = check function
```

```
const EEPROM_ADDRESS_TYPE eeprom_data[][2] = {
#if EEPROM_NUMBER == 1
{0b00000001100, 0b00000001},
{0b00000001010, 0b00000001},
{0b00000001110, 0b01000000},
// IB
```

³¹ Code Online verfügbar unter https://github.com/quiode/EEPROM_CAT28C16A_Programmer.

```

{0b00000010100, 0b00000001},
{0b00000010010, 0b00000001},
{0b00000010110, 0b00010000},
// ADD
{0b00000011100, 0b00000001},
{0b00000011010, 0b00000001},
{0b00000011110, 0b00001000},
// SUB
{0b00000100100, 0b00000001},
{0b00000100010, 0b00000001},
{0b00000100110, 0b00001000},
// OA
{0b00000101100, 0b00000001},
{0b00000101010, 0b10000000},
// OB
{0b00000110100, 0b00000001},
{0b00000110010, 0b00100000},
// SA
{0b00000111100, 0b00000001},
{0b00000111010, 0b00000001},
{0b00000111110, 0b00000000},
{0b00000111001, 0b10000000},
// SB
{0b00001000100, 0b00000001},
{0b00001000010, 0b00000001},
{0b00001000110, 0b00000000},
{0b00001000001, 0b00100000},
// SADD
{0b00001001100, 0b00000001},
{0b00001001010, 0b00000001},
{0b00001001110, 0b00000000},
{0b00001001001, 0b00100000},
// SSUB
{0b00001010100, 0b00000001},
{0b00001010010, 0b00000001},
{0b00001010110, 0b00000000},
{0b00001010001, 0b00100000},
// O
{0b00001011100, 0b00000001},
{0b00001011010, 0b00000001},
{0b00001011110, 0b00000000},
// JUMP
{0b00001100100, 0b00000001},
{0b00001100010, 0b00000001},
{0b00001100110, 0b00000100},
// HALT
{0b00001101100, 0b00000001},
{0b00001101010, 0b00000000},
#elif EEPROM_NUMBER == 2

```

```

// IA
{0b00000001100, 0b00000000},
{0b00000001010, 0b00111100},
{0b00000001110, 0b00000110},
// IB
{0b00000010100, 0b00000000},
{0b00000010010, 0b00111100},
{0b00000010110, 0b00000110},
// ADD
{0b00000011100, 0b00000000},
{0b00000011010, 0b00111100},
{0b00000011110, 0b00000000},
// SUB
{0b00000100100, 0b00000000},
{0b00000100010, 0b00111100},
{0b00000100110, 0b10000000},
// OA
{0b00000101100, 0b00000000},
{0b00000101010, 0b00000000},
// OB
{0b00000110100, 0b00000000},
{0b00000110010, 0b00000000},
// SA
{0b00000111100, 0b00000000},
{0b00000111010, 0b00111100},
{0b00000111110, 0b00010110},
{0b00000111001, 0b00000011},
// SB
{0b00001000100, 0b00000000},
{0b00001000010, 0b00111100},
{0b00001000110, 0b00010110},
{0b00001000001, 0b00000011},
// SADD
{0b00001001100, 0b00000000},
{0b00001001010, 0b00111100},
{0b00001001110, 0b00010110},
{0b00001001001, 0b00000011},
// SSUB
{0b00001010100, 0b00000000},
{0b00001010010, 0b00111100},
{0b00001010110, 0b00010110},
{0b00001010001, 0b10000011},
// O
{0b00001011100, 0b00000000},
{0b00001011010, 0b00111100},
{0b00001011110, 0b00000110},
// JUMP
{0b00001100100, 0b00000000},
{0b00001100010, 0b00111100},

```

```

{0b00001100110, 0b00001100},
// HALT
{0b00001101100, 0b00000000},
{0b00001101010, 0b01000000},
#elif EEPROM_NUMBER == 3
// IA
{0b00000001100, 0b10000000},
{0b00000001010, 0b00000000},
{0b00000001110, 0b00000000},
// IB
{0b00000010100, 0b10000000},
{0b00000010010, 0b00000000},
{0b00000010110, 0b00000000},
// ADD
{0b00000011100, 0b10000000},
{0b00000011010, 0b00000000},
{0b00000011110, 0b01000000},
// SUB
{0b00000100100, 0b10000000},
{0b00000100010, 0b00000000},
{0b00000100110, 0b01000000},
// OA
{0b00000101100, 0b10000000},
{0b00000101010, 0b01000000},
// OB
{0b00000110100, 0b10000000},
{0b00000110010, 0b01000000},
// SA
{0b00000111100, 0b10000000},
{0b00000111010, 0b00000000},
{0b00000111110, 0b00000000},
{0b00000111001, 0b00000000},
// SB
{0b00001000100, 0b10000000},
{0b00001000010, 0b00000000},
{0b00001000110, 0b00000000},
{0b00001000001, 0b00000000},
// SADD
{0b00001001100, 0b10000000},
{0b00001001010, 0b00000000},
{0b00001001110, 0b00000000},
{0b00001001001, 0b00000000},
// SSUB
{0b00001010100, 0b10000000},
{0b00001010010, 0b00000000},
{0b00001010110, 0b00000000},
{0b00001010001, 0b00000000},
// O
{0b00001011100, 0b10000000},

```

```
{0b00001011010, 0b00000000},  
{0b00001011110, 0b01000000},  
// JUMP  
{0b00001100100, 0b10000000},  
{0b00001100010, 0b00000000},  
{0b00001100110, 0b00000000},  
// HALT  
{0b00001101100, 0b01000000},  
{0b00001101010, 0b00000000},  
#endif  
};  
  
// Gets bit from an int  
//   https://www.studymite.com/cpp/examples/program-to-get-nth-bit-of-a-number-in-cpp/  
bool GetBit(unsigned int b, unsigned char bitNumber)  
{  
    return (1 & (b >> (bitNumber - 1)));  
}  
  
// reverses a byte  
// https://stackoverflow.com/a/2602885/13556449  
unsigned char reverse(unsigned char b)  
{  
    b = (b & 0xF0) >> 4 | (b & 0x0F) << 4;  
    b = (b & 0xCC) >> 2 | (b & 0x33) << 2;  
    b = (b & 0xAA) >> 1 | (b & 0x55) << 1;  
    return b;  
}  
  
// sets the data in the shift registers  
void setShiftRegisterData(unsigned int data, unsigned char data_length)  
{  
#if DEBUG_MODE == 4  
Serial.println(data, BIN);  
#endif  
// Set address  
for (unsigned char i = 1; i < data_length + 1; i++)  
{  
    digitalWrite(SHIFT_DATA, GetBit(data, i));  
    delay(1);  
    digitalWrite(SHIFT_CLOCK, LOW);  
    delay(1);  
    digitalWrite(SHIFT_CLOCK, HIGH);  
    delay(1);  
#if DEBUG_MODE == 4  
Serial.print(i);  
Serial.print(" : ");  
Serial.println(GetBit(data, i));  
}
```

```
while (Serial.available() == 0)
{
delay(1);
}
Serial.read();
#endif
}
#if DEBUG_MODE == 4
Serial.println();
// digitalWrite(SHIFT_DATA, 0);
// delay(1);
// digitalWrite(SHIFT_CLOCK, LOW);
// delay(1);
// digitalWrite(SHIFT_CLOCK, HIGH);
// delay(1);
#endif
}

// sets the shift-registers to 0
void resetShiftRegisters()
{
setShiftRegisterData(0, 16);
}

// sets the shift-registers to write data
void writeShiftRegister(EEPROM_ADDRESS_TYPE address)
{
resetShiftRegisters();

setShiftRegisterData(address, 11);

setShiftRegisterData(0b000001, 6);
}

// sets the shift-registers to read data
void readShiftRegister(EEPROM_ADDRESS_TYPE address)
{
resetShiftRegisters();

setShiftRegisterData(address, 11);

setShiftRegisterData(0b000010, 6);
}

// sets the data at the data pins
void setPinData(EEPROM_DATA_TYPE data)
{
digitalWrite(IO0, GetBit(data, 1));
digitalWrite(IO1, GetBit(data, 2));
```

```
digitalWrite(IO2, GetBit(data, 3));
digitalWrite(IO3, GetBit(data, 4));
digitalWrite(IO4, GetBit(data, 5));
digitalWrite(IO5, GetBit(data, 6));
digitalWrite(IO6, GetBit(data, 7));
digitalWrite(IO7, GetBit(data, 8));
}

// Sets the pins for writing Data
void setWritePinMode()
{
    // set all pins to low before using them to be sure nothing gets wrongly set
    digitalWrite(IO0, LOW);
    digitalWrite(IO1, LOW);
    digitalWrite(IO2, LOW);
    digitalWrite(IO3, LOW);
    digitalWrite(IO4, LOW);
    digitalWrite(IO5, LOW);
    digitalWrite(IO6, LOW);
    digitalWrite(IO7, LOW);

    digitalWrite(SHIFT_CLOCK, HIGH);
    digitalWrite(SHIFT_DATA, LOW);

    digitalWrite(CE, HIGH);

    // set pins to the correct mode
    pinMode(IO0, OUTPUT);
    pinMode(IO1, OUTPUT);
    pinMode(IO2, OUTPUT);
    pinMode(IO3, OUTPUT);
    pinMode(IO4, OUTPUT);
    pinMode(IO5, OUTPUT);
    pinMode(IO6, OUTPUT);
    pinMode(IO7, OUTPUT);

    pinMode(SHIFT_CLOCK, OUTPUT);
    pinMode(SHIFT_DATA, OUTPUT);

    pinMode(CE, OUTPUT);

    delay(1);
    digitalWrite(SHIFT_CLOCK, LOW);
    delay(1);
    digitalWrite(SHIFT_CLOCK, HIGH);
}

// Sets the pins for reading Data
void setReadPinMode()
```

```
{  
// set all pins to low before using them to be sure nothing gets wrongly set  
digitalWrite(SHIFT_CLOCK, HIGH);  
digitalWrite(SHIFT_DATA, LOW);  
  
digitalWrite(CE, HIGH);  
  
// set pins to the correct mode  
pinMode(IO0, INPUT);  
pinMode(IO1, INPUT);  
pinMode(IO2, INPUT);  
pinMode(IO3, INPUT);  
pinMode(IO4, INPUT);  
pinMode(IO5, INPUT);  
pinMode(IO6, INPUT);  
pinMode(IO7, INPUT);  
  
pinMode(SHIFT_CLOCK, OUTPUT);  
pinMode(SHIFT_DATA, OUTPUT);  
  
pinMode(CE, OUTPUT);  
  
delay(1);  
  
digitalWrite(SHIFT_CLOCK, LOW);  
delay(1);  
digitalWrite(SHIFT_CLOCK, HIGH);  
}  
  
void serialStart()  
{  
Serial.begin(9600);  
while (!Serial)  
{  
delay(1);  
}  
Serial.println("Serial started");  
delay(100);  
}  
  
void finishedMessage()  
{  
Serial.println("Finished");  
delay(1000 * 60);  
}  
  
// disable the EEPROM and set data and shift-registers to 0  
void disableEEPROM()  
{
```

```
digitalWrite(CE, HIGH);

delay(20);

resetShiftRegisters();
setPinData(0);
}

// checks if the address is in the eeprom address list
short checkAddress(EEPROM_ADDRESS_TYPE address)
{
for (EEPROM_ADDRESS_TYPE i = 0; i < (sizeof(eeprom_data) / sizeof(eeprom_data[0])); i++)
{
if (eeprom_data[i][0] == address)
{
return i;
}
}
return -1;
}

void fullEEPROMWrite()
{
// set mode for writing
setWritePinMode();

// write data
for (EEPROM_ADDRESS_TYPE i = 0; i < EEPROM_ADDRESS_SIZE; i++)
{
EEPROM_DATA_TYPE writeData;
short checkAddressResult = checkAddress(i); // is the position in the eeprom_data array where the correct address and data is
// checks if the address is defined
if (checkAddressResult != -1)
{
writeData = eeprom_data[checkAddressResult][1]; // at 0 the address is stored, at 1 the data
}
else if (GetBit(i, 3) && !GetBit(i, 2) && !GetBit(i, 1))
{
// Because the Instruction has to be loaded every time set the instruction for the first step for every instruction to the same value
#if EEPROM_NUMBER == 1
writeData = 0b00000001;
#elif EEPROM_NUMBER == 2
writeData = 0b00000000;
#elif EEPROM_NUMBER == 3
writeData = 0b10000000;
}
```

```
#endif
}
else // if nothing was predefined, set data to 0
{
writeData = 0;
}
writeData = reverse(writeData); // make it correct
#if DEBUG_MODE == 1
Serial.print("Writing: ");
Serial.print(writeData, BIN);
Serial.print(" at address: ");
Serial.println(i, BIN);
#endif
setPinData(writeData);
delay(1);
writeShiftRegister(i); // write the address
delay(1);
digitalWrite(CE, LOW);
delay(1);
digitalWrite(CE, HIGH);
delay(20);
}

// finishing touches
disableEEPROM();
}

// sets the eeprom to everywhere 0
void setEEPROMtoZero()
{
// set mode for writing
setWritePinMode();

// write data
for (EEPROM_ADDRESS_TYPE i = 0; i < EEPROM_ADDRESS_SIZE; i++)
{
setPinData(0);
delay(1);
writeShiftRegister(i); // write the address
delay(1);
digitalWrite(CE, LOW);
delay(1);
digitalWrite(CE, HIGH);
delay(20);
}

// finishing touches
disableEEPROM();
}
```

```
EEPROM_DATA_TYPE readDataPins()
{
delay(1);
EEPROM_DATA_TYPE data = 0;
data |= digitalRead(IO0) << 0;
data |= digitalRead(IO1) << 1;
data |= digitalRead(IO2) << 2;
data |= digitalRead(IO3) << 3;
data |= digitalRead(IO4) << 4;
data |= digitalRead(IO5) << 5;
data |= digitalRead(IO6) << 6;
data |= digitalRead(IO7) << 7;
return data;
}

void readFullEEPROM()
{
// set mode for reading
setReadPinMode();

Serial.println("Reading EEPROM");

// read data
for (EEPROM_ADDRESS_TYPE i = 0; i < EEPROM_ADDRESS_SIZE; i++)
{
readShiftRegister(i);
delay(1);
digitalWrite(CE, LOW);
delay(1);
Serial.print(readDataPins(), BIN);
Serial.print(" at ");
delay(1);
Serial.println(i, BIN);
digitalWrite(CE, HIGH);
delay(1);
}

disableEEPROM();
}

// write one byte of data to one address
void writeEEPROM(EEPROM_ADDRESS_TYPE address, EEPROM_DATA_TYPE data)
{
// set mode for writing
setWritePinMode();

// write data
setPinData(data);
```

```
delay(1);
writeShiftRegister(address); // write the address
delay(1);
digitalWrite(CE, LOW);
delay(1);
#if DEBUG_MODE == 3
while (Serial.available() == 0)
{
delay(1);
}
Serial.read();
#endif
digitalWrite(CE, HIGH);
delay(20);
}

// read one byte of data from one address
void readEEPROM(EEPROM_ADDRESS_TYPE address)
{
// set mode for reading
setReadPinMode();

// read data
readShiftRegister(address);
delay(1);
digitalWrite(CE, LOW);
delay(1);
Serial.print(readDataPins(), BIN);
Serial.print(" at ");
digitalWrite(CE, HIGH);
Serial.println(address, BIN);
delay(1);
}

// checks if every address is saved correctly, if notify is = false, checkEEPROM will
// not print to serial and just return false if one or more addresses are not saved
// correctly
bool checkEEPROM(bool notify = true)
{
bool return_val = true;
EEPROM_DATA_TYPE expectedData;
// set mode for reading
setReadPinMode();
Serial.println("Checking EEPROM");
delay(1);

// read data
for (EEPROM_ADDRESS_TYPE i = 0; i < EEPROM_ADDRESS_SIZE; i++)
{
```

```
expectedData = 0;
readShiftRegister(i);
delay(1);
digitalWrite(CE, LOW);
delay(1);
EEPROM_DATA_TYPE readData = readDataPins();
#if DEBUG_MODE == 5
Serial.println(readDataPins(), BIN);
#endif
delay(1);
digitalWrite(CE, HIGH);
delay(1);
short checkAddressResult = checkAddress(i); // is the position in the eeprom_data
array where the correct address and data is
// checks if the address is defined
if (checkAddressResult != -1)
{
#if DEBUG_MODE == 5
Serial.print("Checking: ");
Serial.print(readData, BIN);
Serial.print(" checkAddressResult: ");
Serial.print(checkAddressResult);
#endif
expectedData = eeprom_data[checkAddressResult][1]; // at 0 the address is stored,
at 1 the data
#if DEBUG_MODE == 5
Serial.print(" Data: ");
Serial.println(expectedData, BIN);
#endif
}
else if (GetBit(i, 3) && !GetBit(i, 2) && !GetBit(i, 1))
{
#if DEBUG_MODE == 5
Serial.print("Address: ");
Serial.print(i, BIN);
Serial.print(" Check: ");
Serial.println((GetBit(i, 3) && !GetBit(i, 2) && !GetBit(i, 1)));
#endif
#if EEPROM_NUMBER == 1
expectedData = 0b00000001;
#elif EEPROM_NUMBER == 2
expectedData = 0b00000000;
#elif EEPROM_NUMBER == 3
expectedData = 0b10000000;
#endif
}
expectedData = reverse(expectedData); // make it correct
if (readData != expectedData)
{
```

```
if (notify)
{
Serial.print("Data at address: ");
Serial.print(i, BIN);
Serial.print(" is not correct: ");
Serial.print(readData, BIN);
Serial.print(" should be: ");
Serial.println(expectedData, BIN);
return_val = false;
}
else
{
return false;
}
}
else
{
if (notify)
{
Serial.print("Data at address: ");
Serial.print(i, BIN);
Serial.print(" is correct: ");
Serial.println(readData, BIN);
}
}
#endif DEBUG_MODE == 5
#endif
}

disableEEPROM();
return return_val;
}

// gets called once on startup
void setup()
{
// startup functions
serialStart();

#if DEBUG_MODE == 0
// write EEPROM
fullEEPROMWrite();

// // read EEPROM for verification
// readFullEEPROM();

// check EEPROM for verification
checkEEPROM();
#endif DEBUG_MODE == 1
```

```
fullEEPROMWrite();
readFullEEPROM();
#elif DEBUG_MODE == 2 || DEBUG_MODE == 3
for (int i; i < 100; i++)
{
// writeEEPROM(0b10101010101, 0b00000001);
readEEPROM(0b10101010101);

disableEEPROM();
}
#elif DEBUG_MODE == 4
readEEPROM(0b10101101101);
#elif DEBUG_MODE == 5
checkEEPROM();
#endif
}

// gets called repeatedly
void loop()
{
finishedMessage();
}
```

11.8 Bilder des Prozessors³²

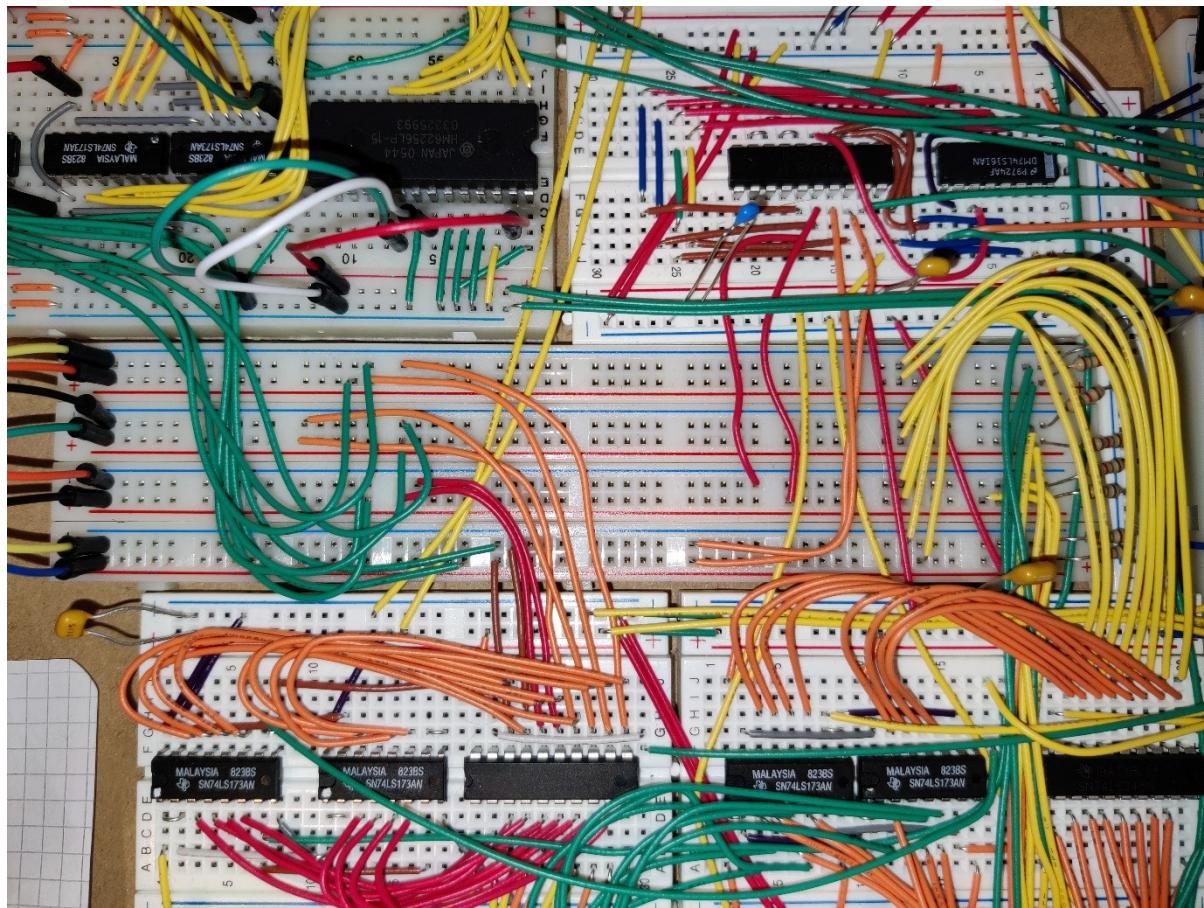


Abbildung 30: BUS mit angeschlossenen Modulen

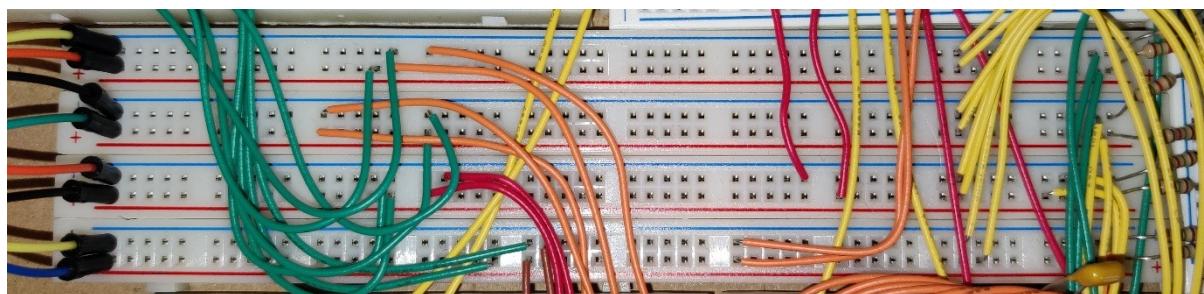


Abbildung 31: BUS Grossaufnahme

³² Bilder auch erhältlich unter https://github.com/quiode/EEPROM_CAT28C16A_Programmer/tree/master/images

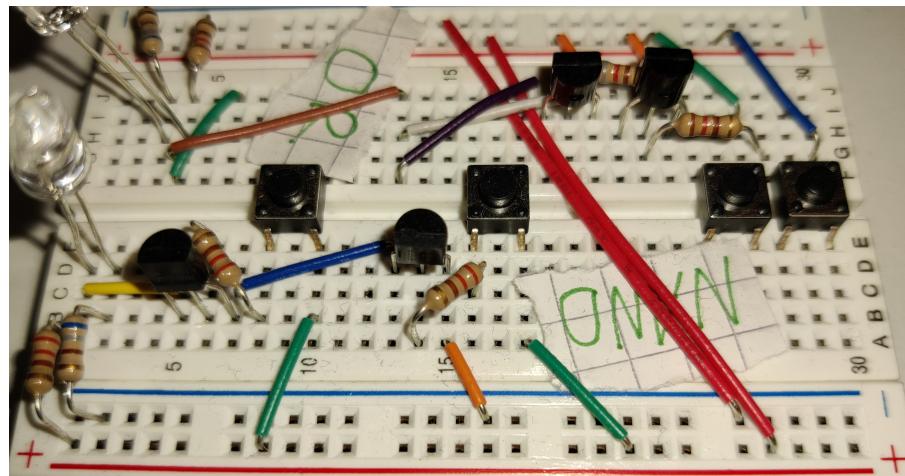


Abbildung 32: NAND-Gatter

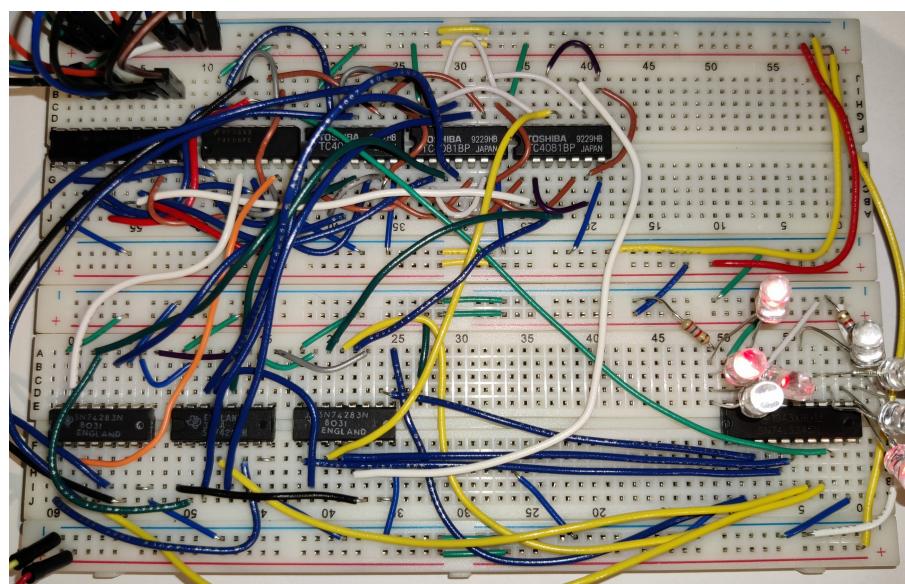


Abbildung 33: 4-Bit Multiplizierer

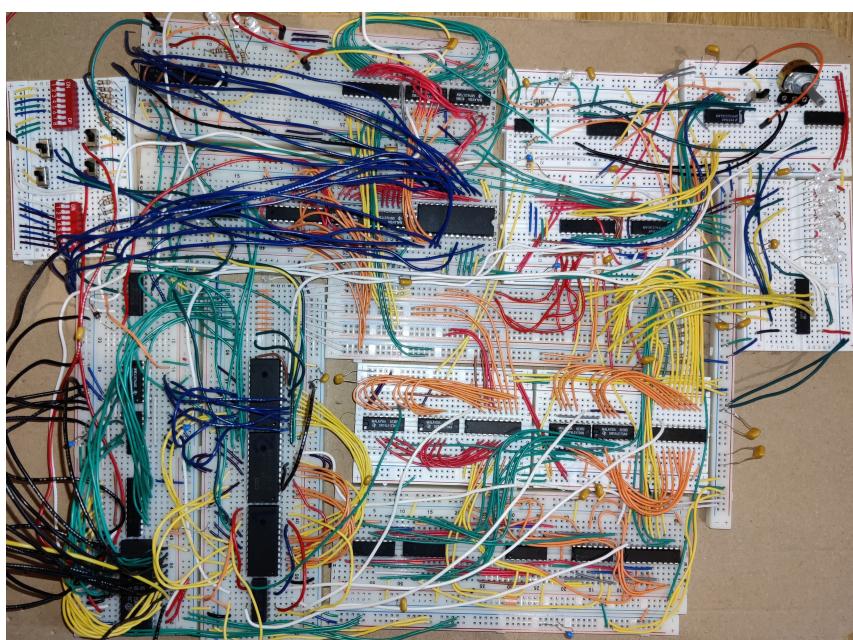


Abbildung 34: Grossaufnahme des Prozessors

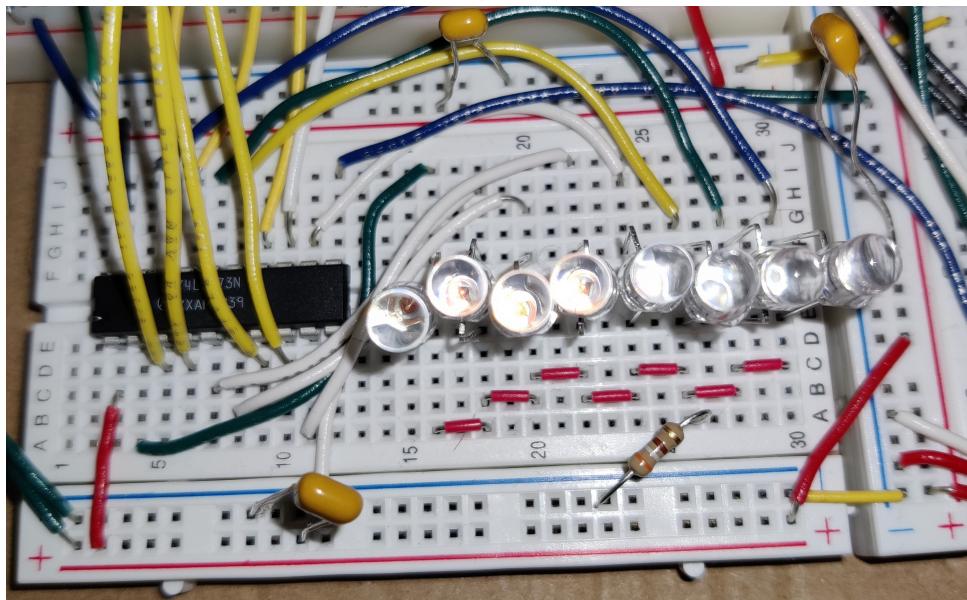


Abbildung 37: Ausgang

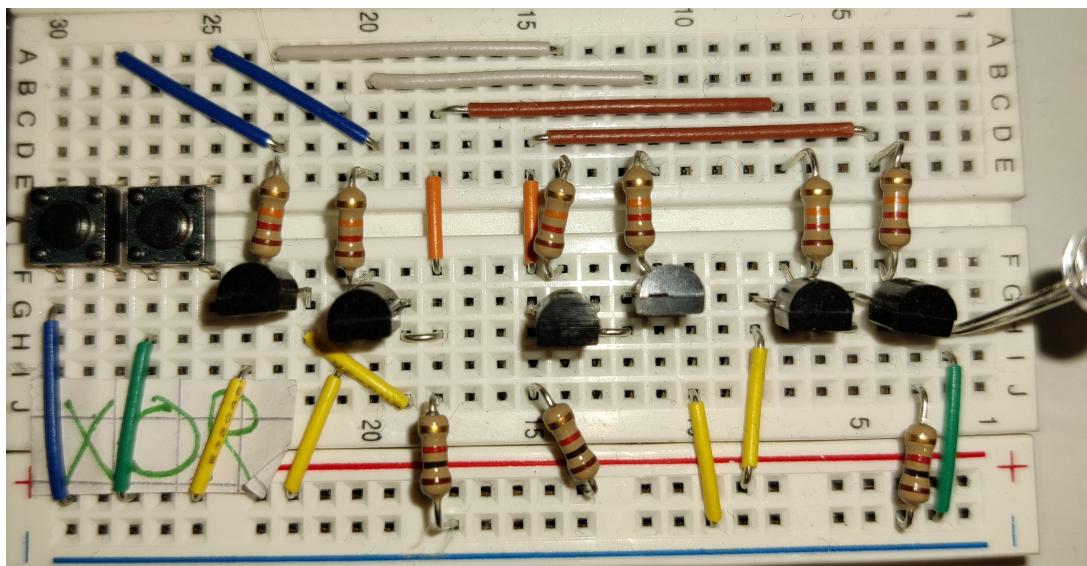


Abbildung 35: XOR-Gatter

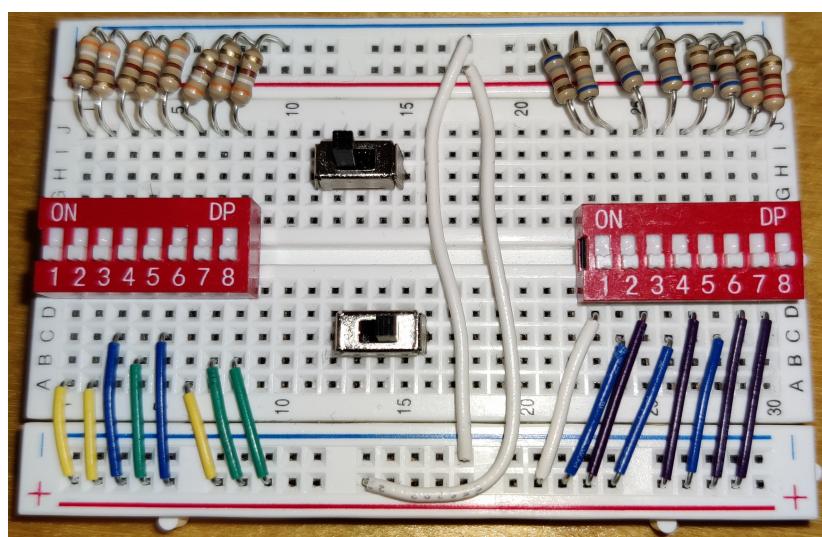


Abbildung 36: Einheit zur Programmierung des RAMs

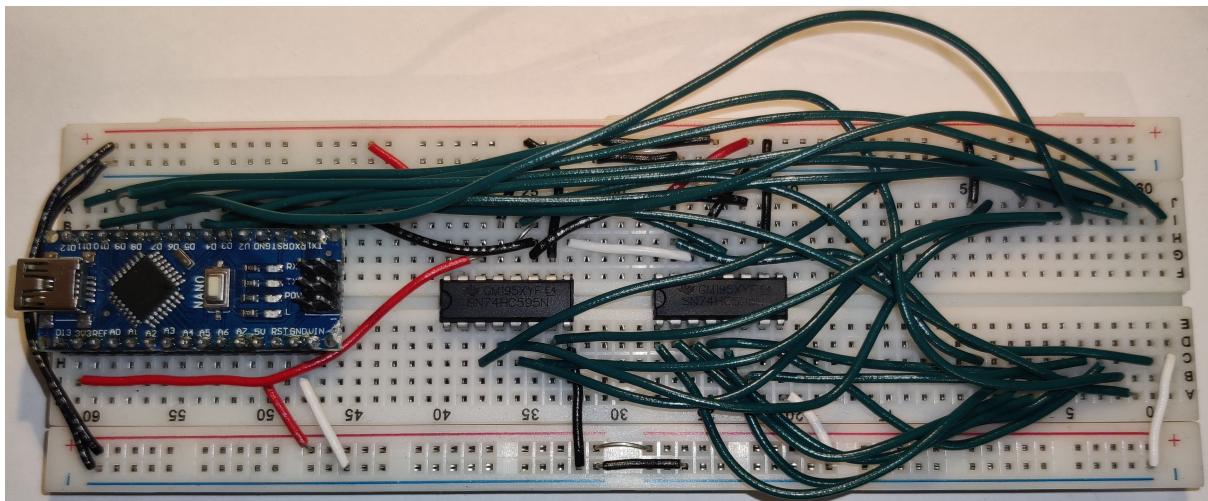


Abbildung 38: Dritte Version EEPROM Programmierer

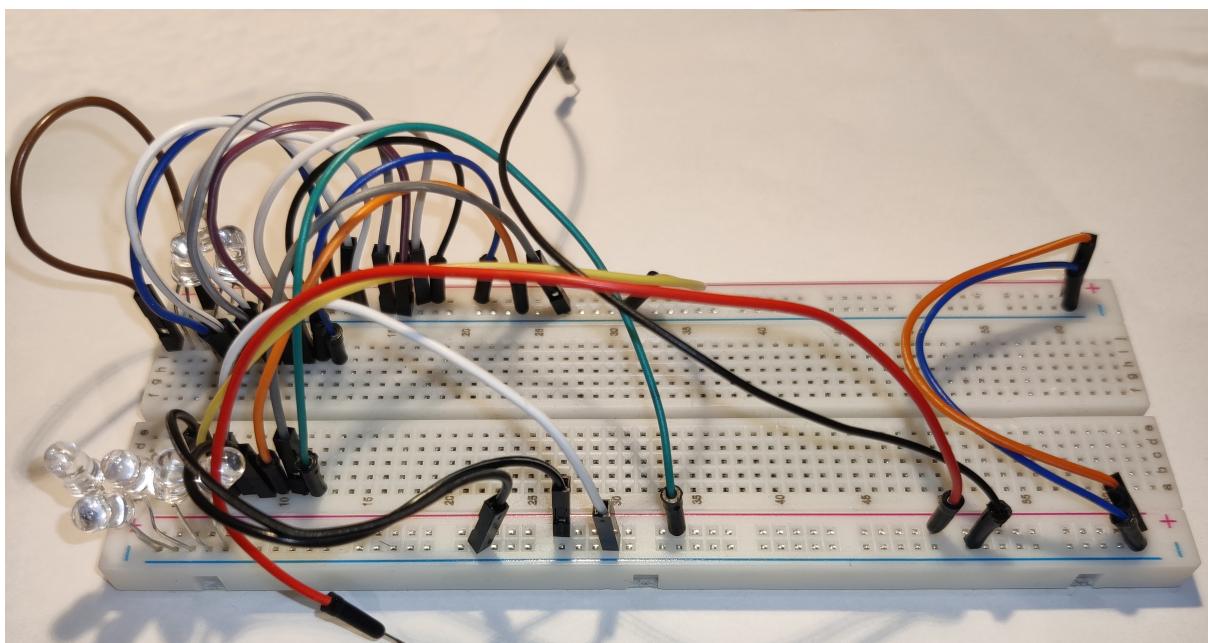


Abbildung 39: Steckbrett zur manuellen Überprüfung des EEPROMs

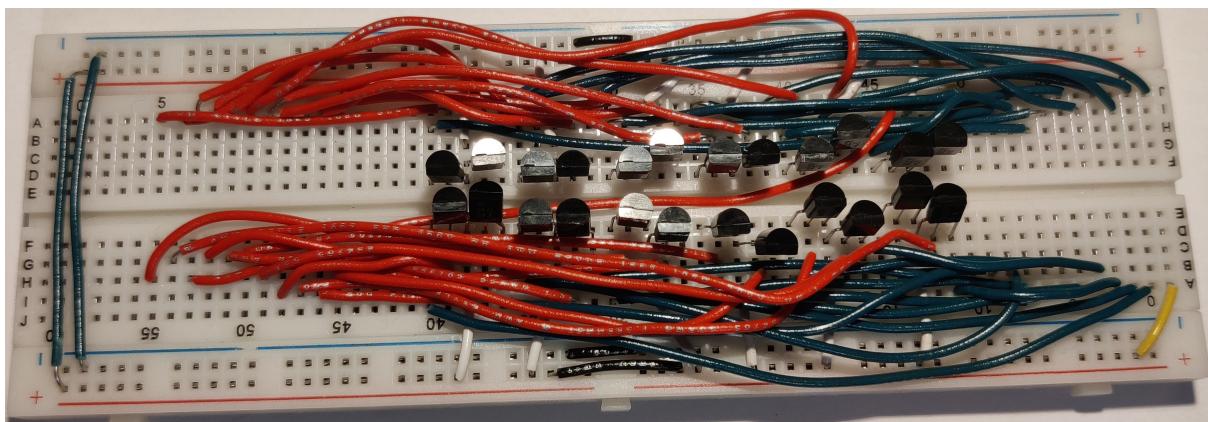


Abbildung 40: Zweite Version EEPROM Programmierer

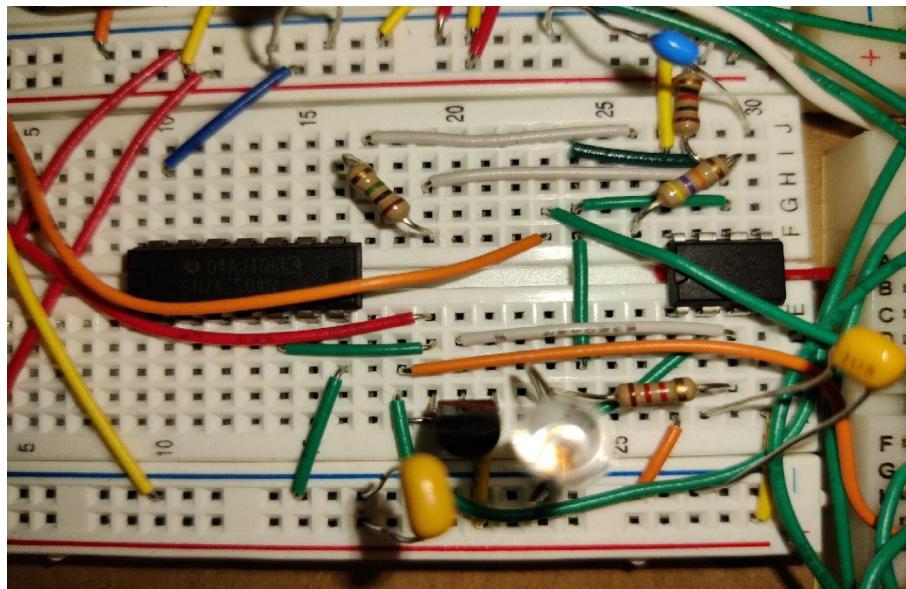


Abbildung 41: Taktgeber (Clock)

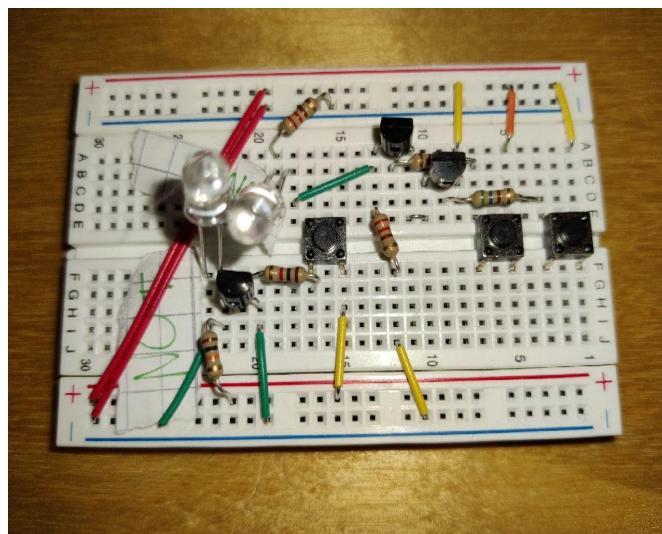


Abbildung 42: Nachbau NOT-Gatter

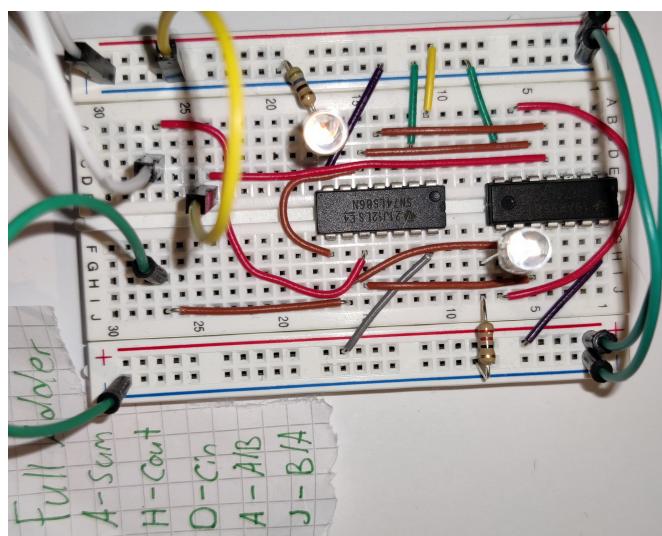


Abbildung 43: Volladdierer

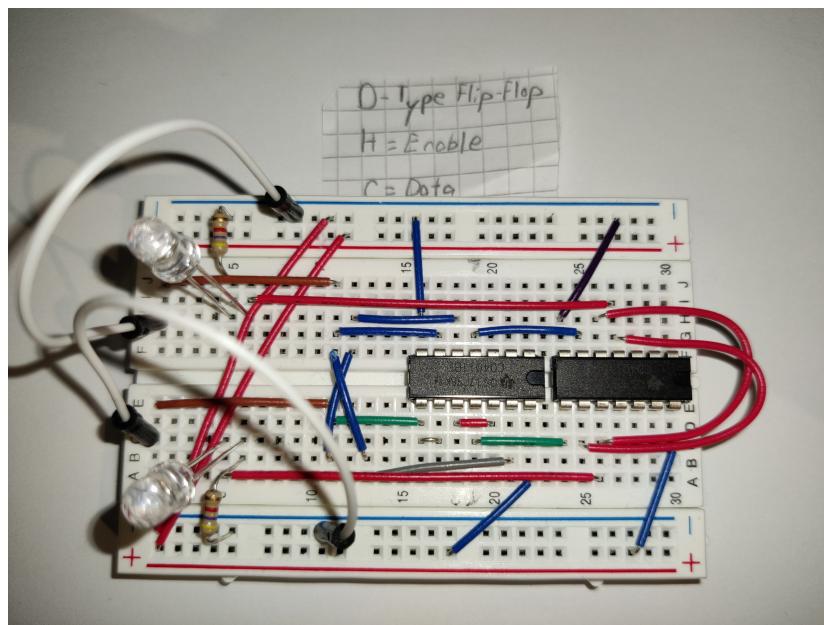


Abbildung 44: D-Type Flipflop

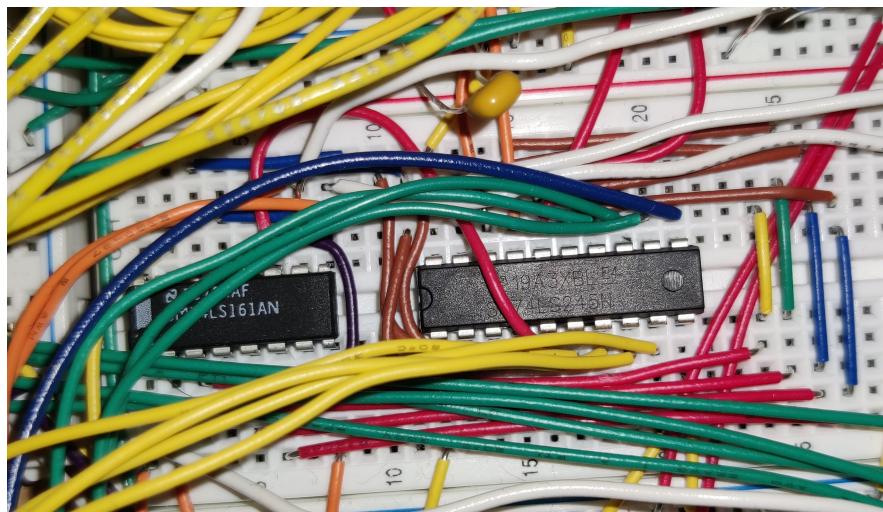


Abbildung 45: Programmzähler

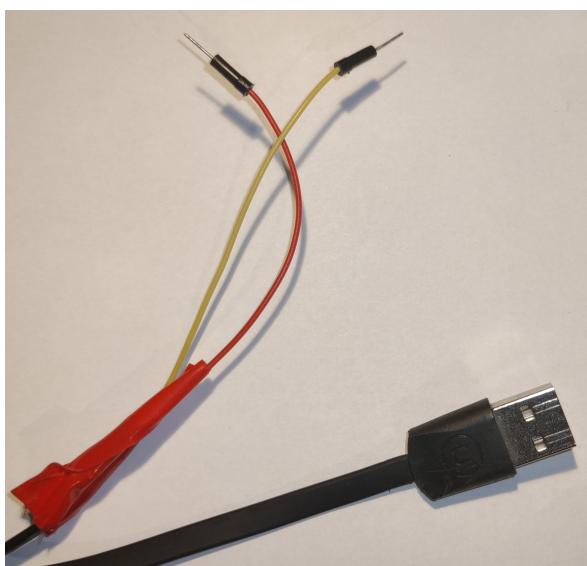


Abbildung 46: Stromversorgung

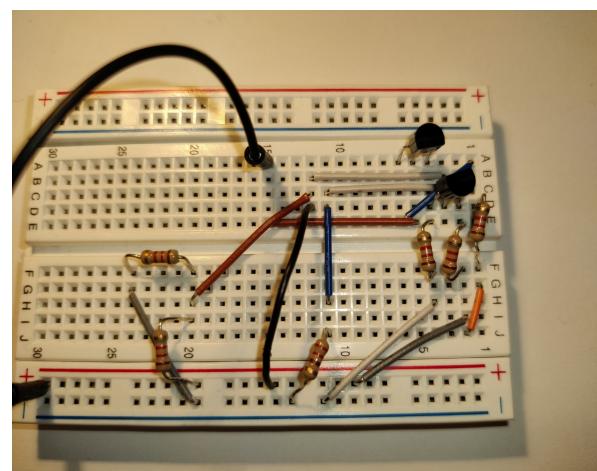


Abbildung 47: Differenzverstärker

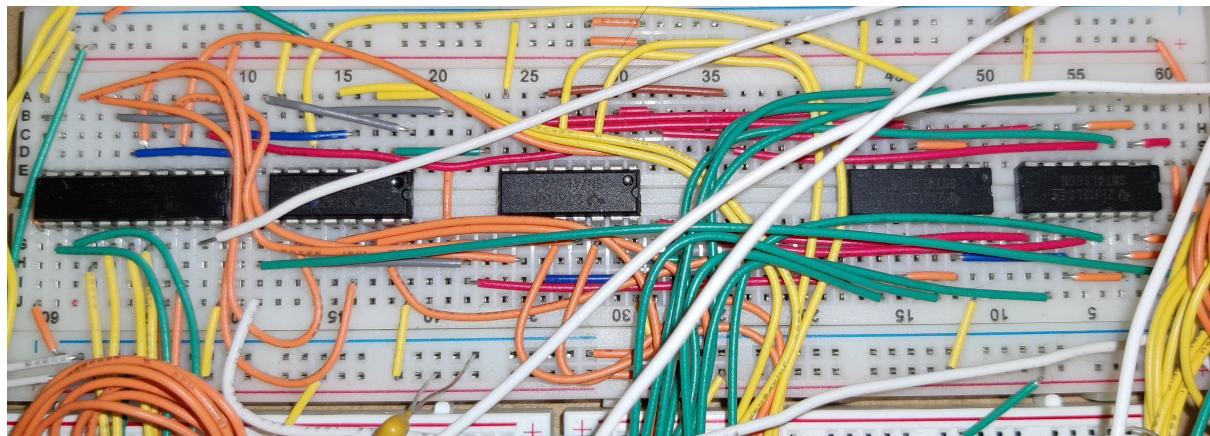


Abbildung 48: ALU

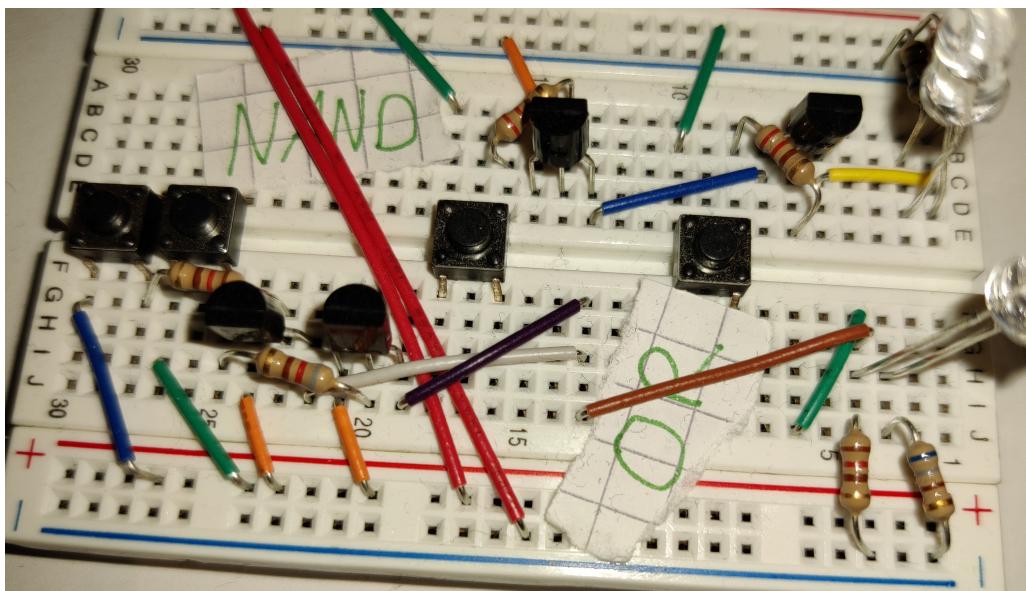


Abbildung 49: OR-Gatter

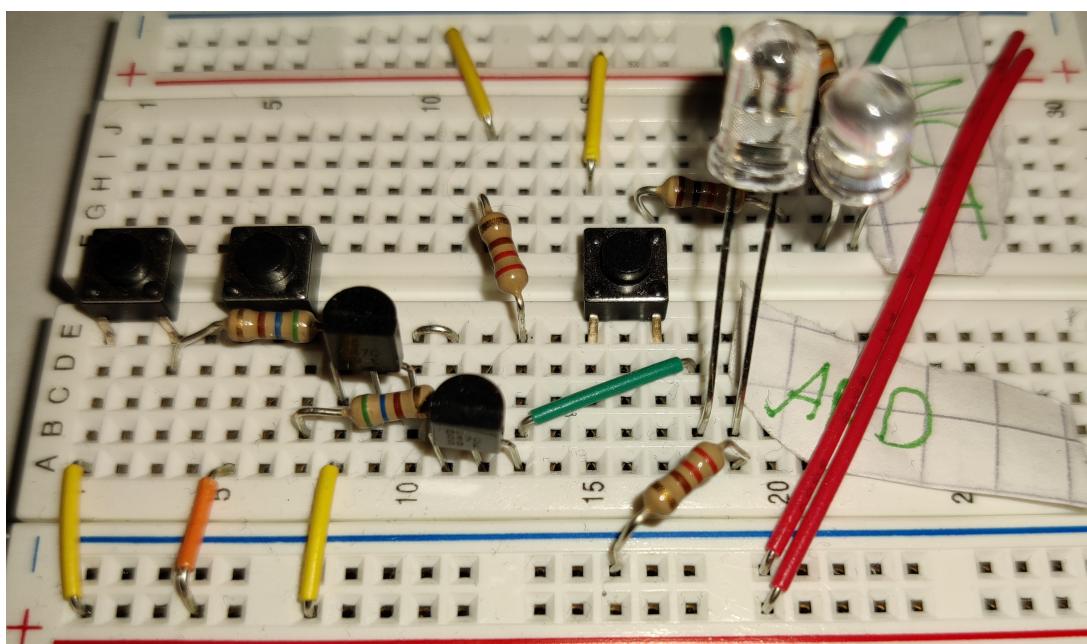


Abbildung 50: AND-Gatter

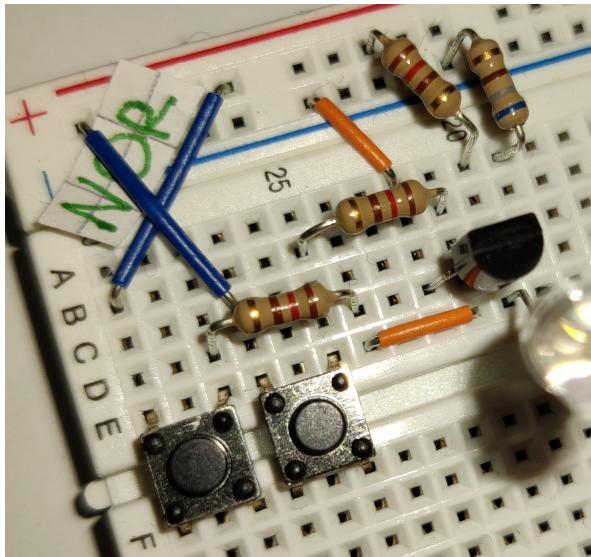


Abbildung 51: NOR-Gatter

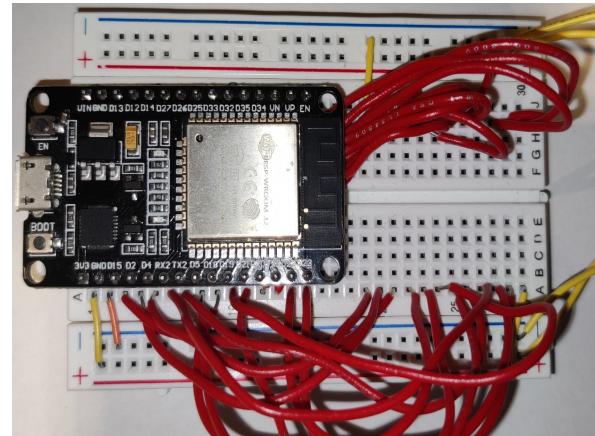


Abbildung 52: Erste Version EEPROM Programmierer

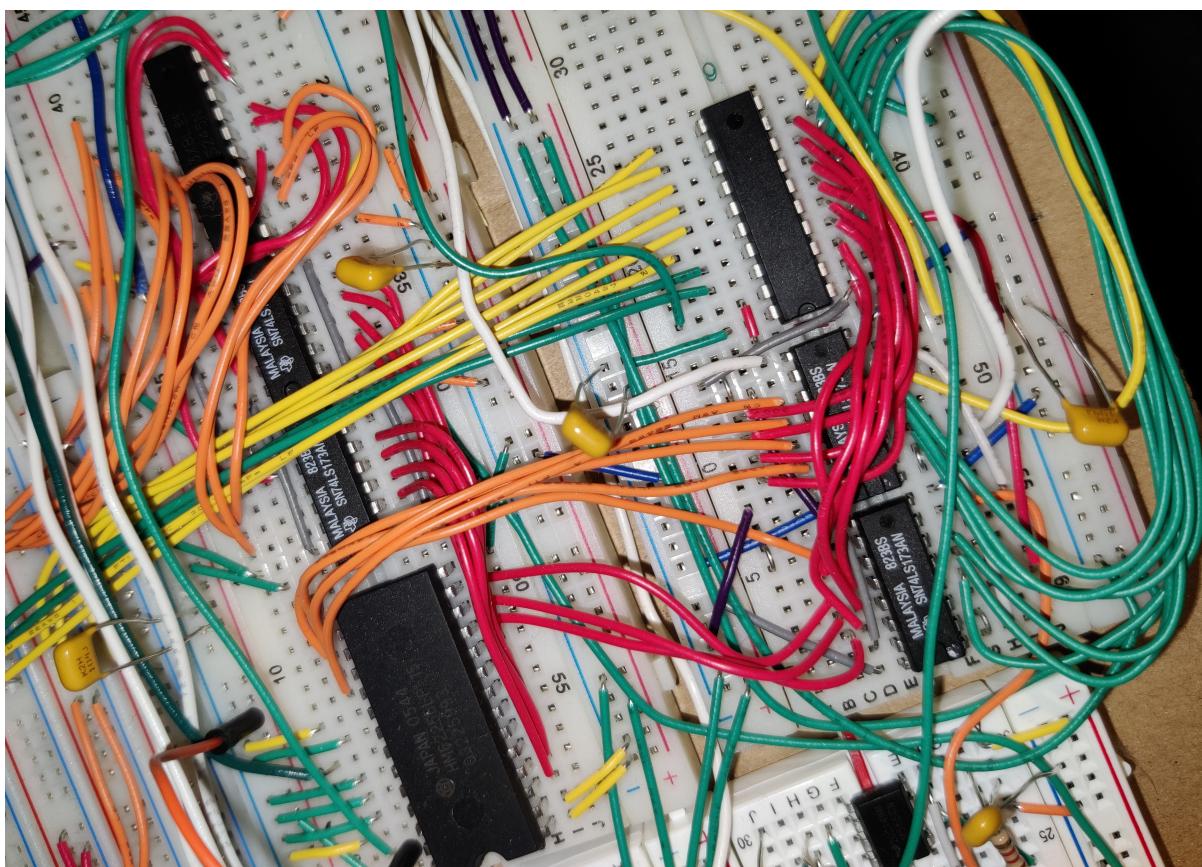


Abbildung 53: RAM um Daten zu speichern

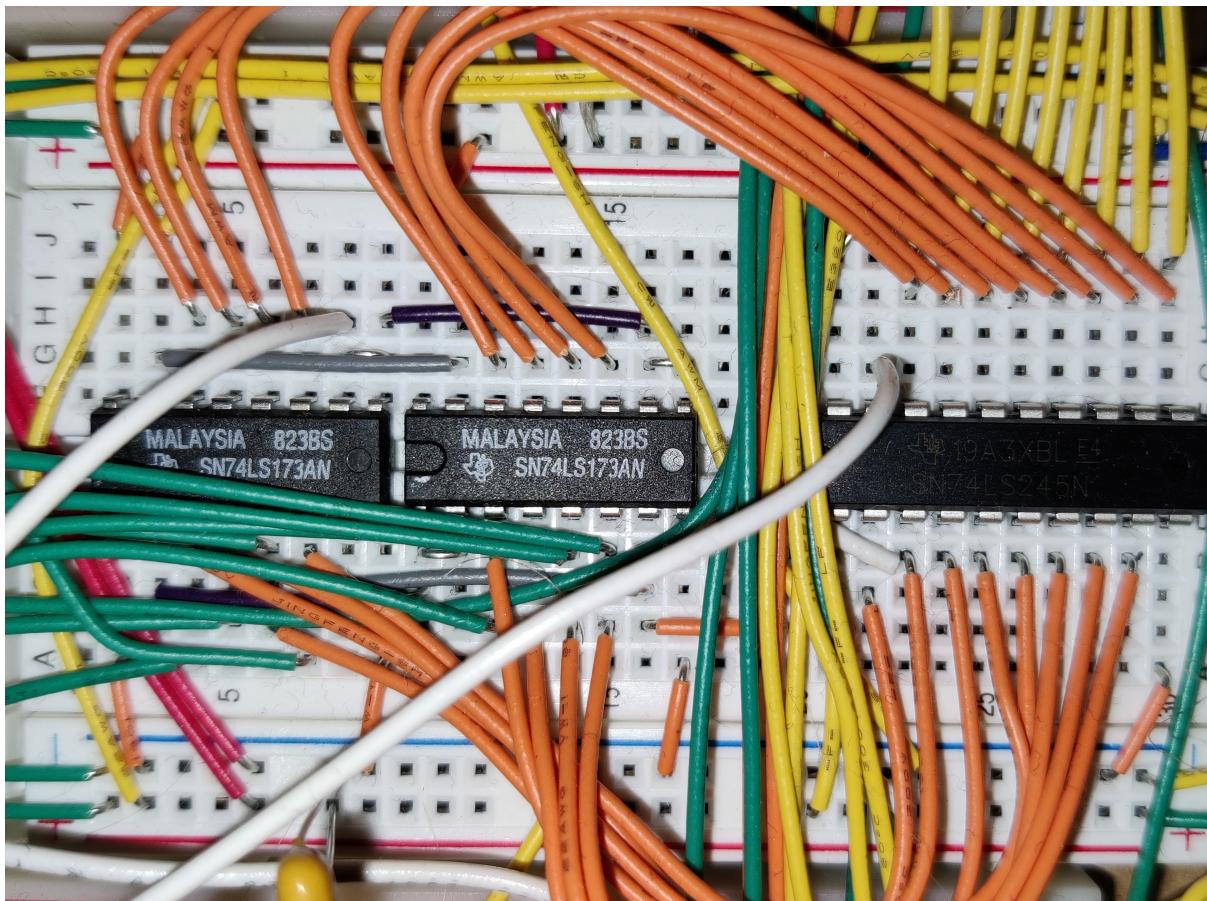


Abbildung 54: Register A

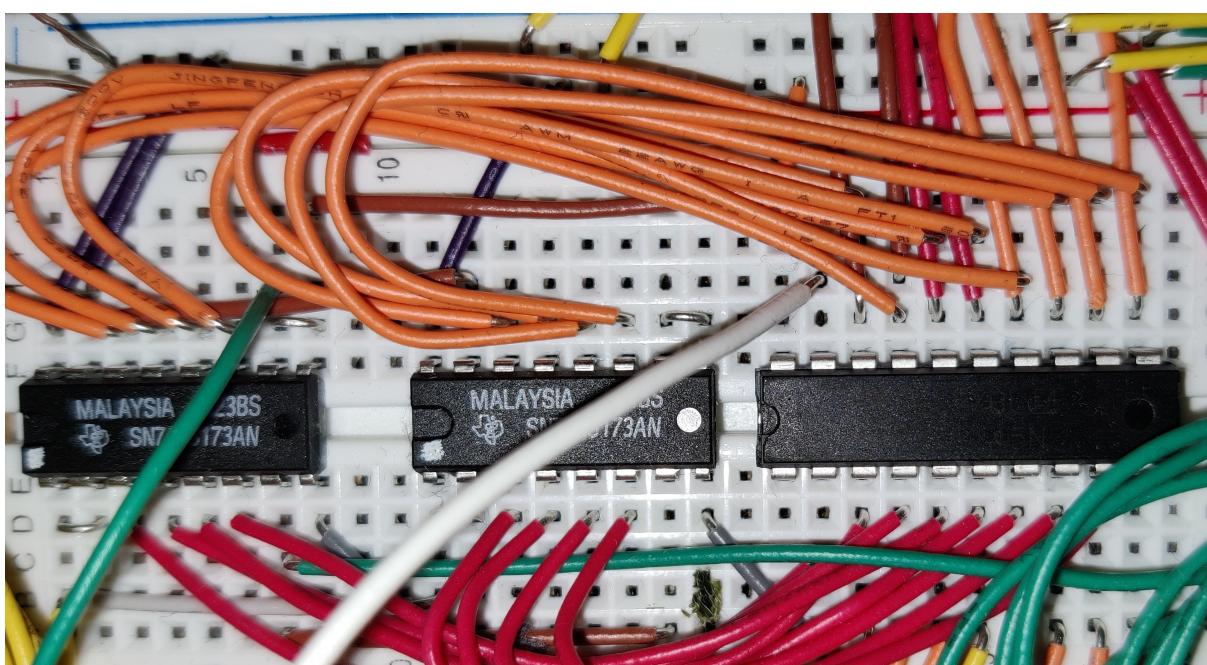


Abbildung 55: Register B