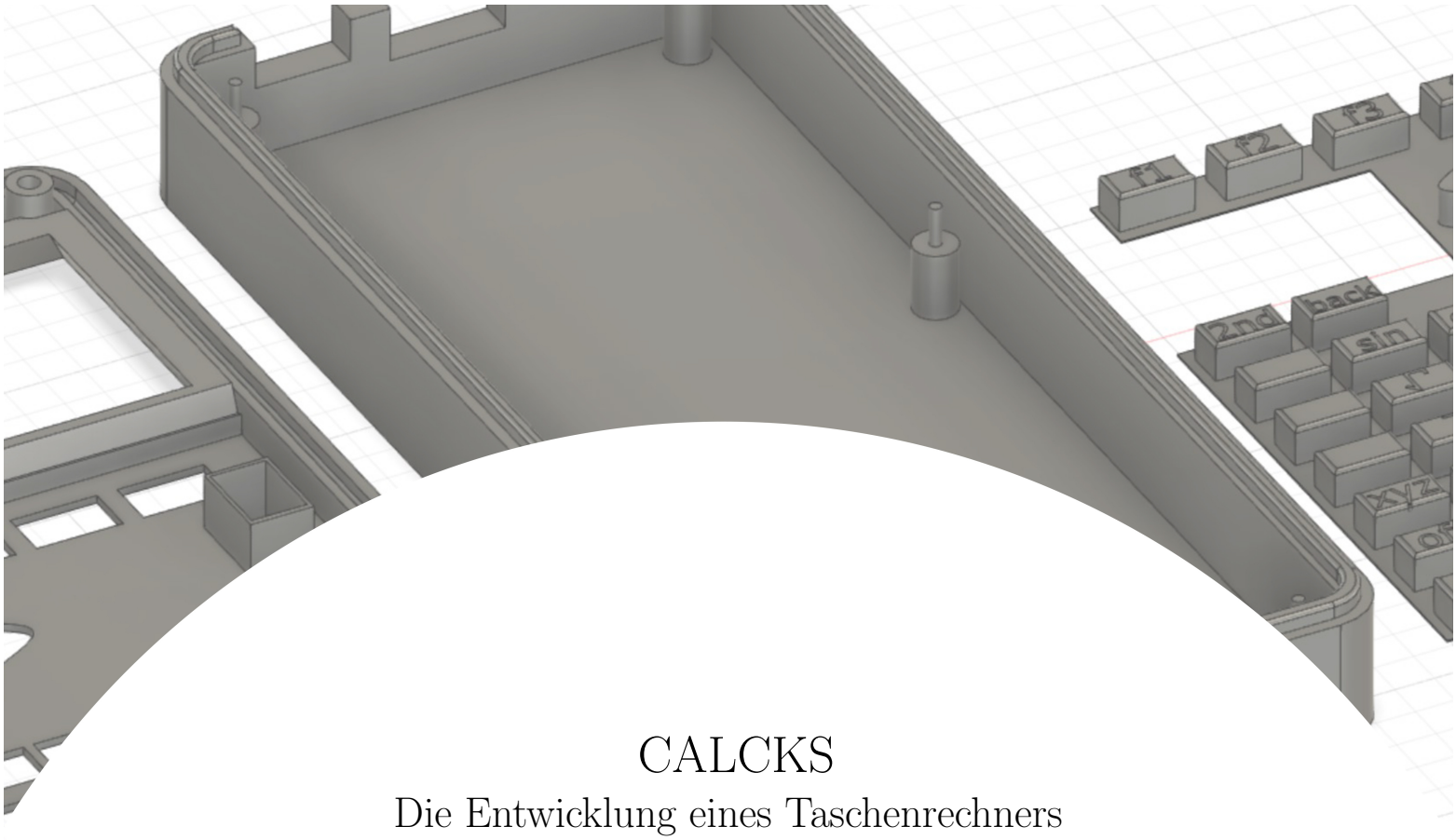


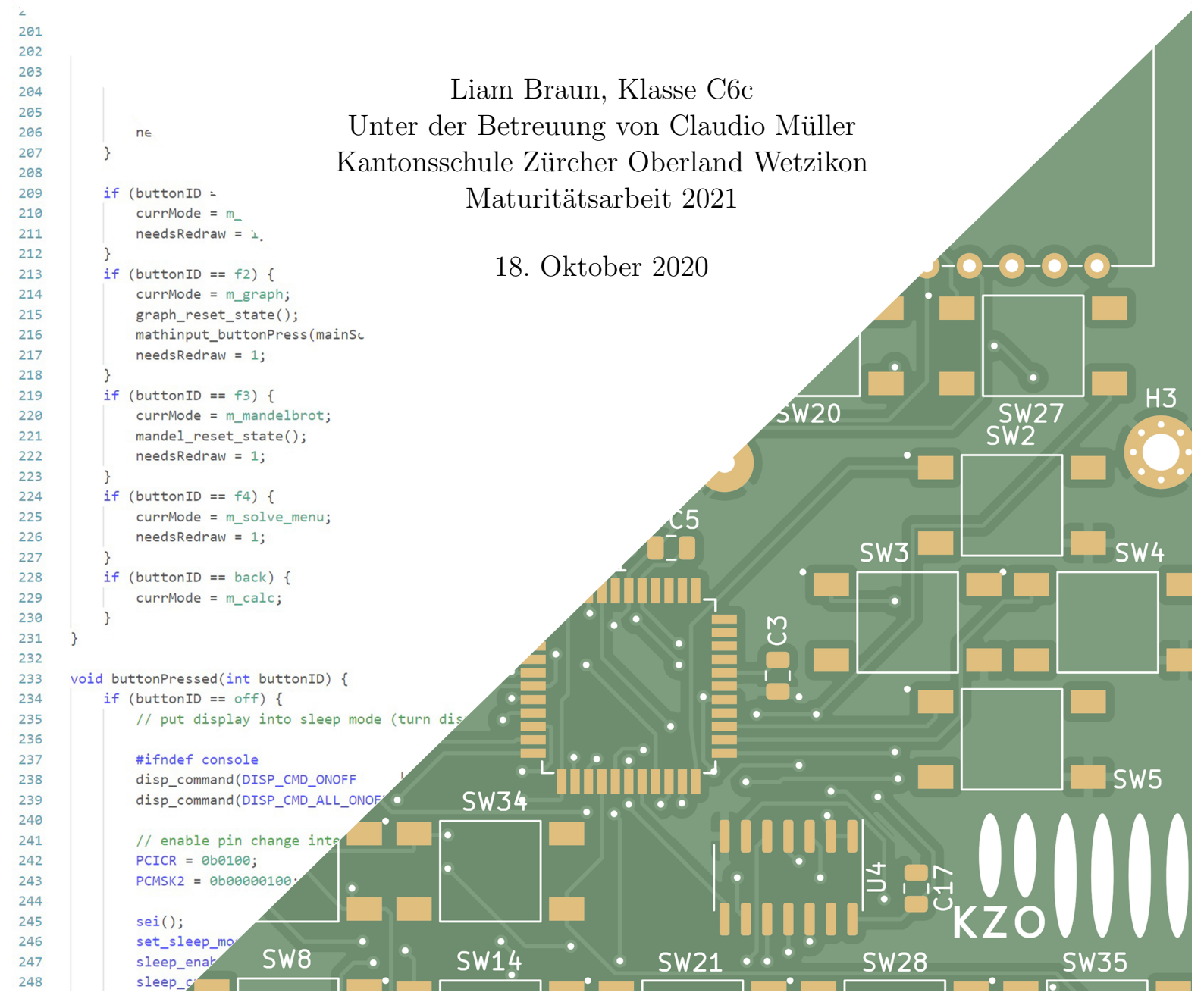
CALCKS

Die Entwicklung eines Taschenrechners

Liam Braun, Klasse C6c
Unter der Betreuung von Claudio Müller
Kantonsschule Zürcher Oberland Wetzikon

Maturitätsarbeit 2021

18. Oktober 2020



Abstract

Diese Arbeit beschreibt den Weg vom Entwurf bis zur Programmierung eines selbstentworfenen Taschenrechners. Thematisiert werden sowohl Fragestellungen bei der Entwicklung als auch technische Details bei der Umsetzung.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Persönliches

Schon sehr früh in meinem Leben war ich fasziniert von Technologie. Eine meiner Lieblingsbeschäftigungen war damals, mit dem wissenschaftlichen Taschenrechner, den mein Vater zu seiner Schulzeit an der KZO besass, zu spielen. Meine erste Programmiererfahrung fand auch auf diesem Gerät, dem SHARP EL-9300 statt.

Abbildung 1.1: Ich mit 3 Jahren, am Spielen mit dem Taschenrechner meines Vaters.

1.2 Wahl des Projekts

Bei der Wahl des Themas kamen verschiedene Bereiche auf, die ich in die Arbeit einbeziehen wollte. Zum einen war das Programmieren, egal ob als Teil eines Spiels, einer Website oder der Software eines Geräts, von Anfang der Fixpunkt, von dem aus ich das genauere Thema bestimmte. Ich war mir sicher, dass dies ein Bereich sei, auf den ich mich ohne Probleme über die acht Monate der Maturarbeit konzentrieren könnte, weil ich mich in meiner Freizeit seit etwa zehn Jahren mit dem Programmieren beschäftige. Ich wollte aber gleichzeitig meinen Horizont in der Informatikwelt erweitern und entschied mich deshalb, das Programmieren mit einem Hardwareprojekt zu kombinieren. Als ich mich an meine Taschenrechnerfaszination zurückerinnerte, war mir klar, was mein Maturprojekt sein soll.

1.3 Ziele der Arbeit

Zu Beginn der Arbeit hatte ich mir einige Erwartungen an das Endprodukt gesetzt:

- Der Taschenrecher sollte ein Matrix-LCD besitzen, damit er Funktionen plotten kann.
- Der Taschenrecher sollte mindestens die Operationen Plus, Minus, Mal, Durch, Klammern, Wurzeln und Potenzen beherrschen, damit rechnen und Graphen zeichnen können.
- Die Stromquelle des Taschenrechners sollte im Gerät integriert sein, entweder in Form einer gewöhnlichen Knopfzelle oder der eines Akkus.

Mir war am Endprodukt vor allem auch wichtig, dass ich einen Nutzen vom Taschenrechner hatte, also, dass der Taschenrechner Funktionen hat, die ein gewöhnlicher Taschenrechner nicht hat.

1.4 Gliederung des schriftlichen Teils

Der schriftliche Teil fokussiert sich mehrheitlich auf die Hardware. Grund dafür ist, dass auch wenn ein Grossteil der Arbeit am Taschenrecher Programmieren war, sind viele Teile des Programms nicht gross kommentierbar. Das Durchlesen des Quellcodes, der aus etwa 3'000 Zeilen Code besteht, ist dem Leser als freiwillige Aufgabe überlassen.

Zuerst werden im Kapitel **Grundlagen** Begriffe erklärt, die vermehrt in der Arbeit vorkommen. In **Material und Methodik** wird der Arbeitsprozess aus meiner Sicht beschrieben. Das Kapitel **Technische Umsetzung** schaut das Endprodukt an und beschreibt, wie einzelne Teile davon funktionieren. Screenshots und Fotos vom Taschenrechner sind im Kapitel **Resultate** zu finden.

Kapitel 2

Grundlagen

2.1 Komponenten in elektronischen Geräten

2.1.1 PCB

Ein Printed Circuit Board, kurz PCB, ist eine meist grüne Platte, die sich in praktisch jedem elektronischen Gerät befindet. Auf sie werden die elektronischen Komponenten wie Prozessor, Knöpfe, Batterien usw. gelötet. Das Selbstherstellen einer solchen Leiterplatte ist bei simplen Schaltkreisen zwar möglich, aber ziemlich aufwendig. Wenn man auf dem Computer ein PCB designt, erstellt man als Erstes ein Schema, wo die elektronischen Komponenten möglichst klar und verständlich ausgelegt werden. Beim PCB-Layout hingegen ist der Fokus weniger bei der Verständlichkeit gegenüber dem Entwickler, sondern darauf, dass die Verbindungen zwischen Komponenten die richtige Breite haben und dass sie möglichst kurz und ohne viele Umwege sind.

Das Schema ist für Menschen gedacht und soll die Idee des Zusammenspiels der Komponenten übersichtlich machen. Das PCB-Layout ordnet die Komponenten so an, dass sie optimal und zweckorientiert platziert sind. Ein USB-Port zum Beispiel könnte auf dem Schema beliebig platziert sein, aber auf dem PCB müsste er am Rand liegen, sodass er für den Nutzer gut erreichbar ist.

2.1.2 Steckplatine

Eine Steckplatine, auch Breadboard genannt, wird für den Prototypenbau verwendet und ist eine Plastikform, die in Reihen aufgeteilt ist. In jeder dieser Reihen hat ein Breadboard Klemmen, die sich über mehrere Löcher spannen und somit ein elektrisches Signal an andere Komponenten in derselben Reihe übertragen können. Ganz links und rechts verlaufen meist je zwei Spalten, die dazu gedacht sind, an eine positive und negative Spannung anzuschliessen, aber grundsätzlich kann man beliebige Signale auf diese Reihen leiten.

2.1.3 IC

Ein IC, kurz für Integrated Circuit (Deutsch: Integrierter Schaltkreis), ist ein Schaltkreis, der Funktionen wie Timer, Speicher, Rechnen usw. aufweist und auf einem PCB verlötet wird oder auf ein Breadboard gesteckt wird. ICs kommen in vielen verschieden Formen und Grössen vor. Spricht man von den Namen dieser Formen,

nennt man dies den Packaging-Typ des ICs.

Die meisten ICs lassen sich in zwei Hauptgruppen unterteilen: SMT- und THM-ICs. SMT steht für "Surface Mount Technology" und bedeutet, dass eine SMT-Komponente auf Pads, die sich auf einem PCB befinden, platziert und angelötet werden. Pads sind kleine rechteckige Flächen auf einem PCB, worauf die Pins einer Komponente gelegt werden. THM steht für "Through-Hole mounting", was bedeutet, dass Komponenten dieser Art in durch das PCB durchgehende Löcher platziert und angelötet werden. Alternativ können diese Komponenten in die Löcher von Steckplatinen gesteckt werden, weshalb man für den Prototypbau auf Steckplatinen ausschliesslich THM-Komponenten verwendet.

2.1.4 Speicher

Es gibt verschiedene Arten, wie Daten in einem IC gespeichert werden können. Hier werden die in der Arbeit erwähnten kurz erklärt.

Flash und EEPROM

Flash- und EEPROM-Speicher sind sehr ähnlich, weil beide Technologien Daten auch ohne laufenden Stromzufuhr beibehalten können. Um einen bereits benutzten Bereich des Speichers zu überschreiben, muss zuerst dieser Bereich gelöscht werden. Bei der EEPROM (Electrically Erasable Programmable Read-Only Memory)-Technologie kann man meistens wenige Bytes aufs Mal löschen, aber bei Flash-Speicher sind diese Löschblöcke in der Regel über 512 Bytes.

RAM

RAM steht für Random Access Memory und ist eine Speicherart, die Strom benötigt, um Daten nicht zu verlieren.

2.1.5 Mikrocontroller

Ein Mikrocontroller ist ein IC, der eine CPU und Speicher enthält. Das heisst, dass ein Mikrocontroller eine Instruktionsmenge beherrscht, mit der er an Ein- und Ausgaben an den Pins operieren kann. Er enthält einen Flash-Speicher, um ein Programm zu speichern, RAM, um mit temporären Daten während des Ausführens des Programms festzuhalten und manchmal auch ein permanenter Speicher wie ein EEPROM, worin während der Exekution eines Programms geschrieben werden kann. Grundsätzlich kann man sich einen Mikrocontroller als einen Computer in Form eines ICs vorstellen, der nur ein einziges Programm ausführen kann.

2.1.6 Shiftregister

Ein Shiftregister ist ein IC, der Input- und Output-Pins hat, die je nach Shiftregister seriell oder parallel Daten einlesen und ausgeben. Ein Shiftregister hat einen Takt-Pin (CLK), der den aktuell im IC vorhandenen Wert bitweise nach rechts verschiebt. Ein Beispiel: Ein Serial-In-Parallel-Out (SIPO)-Shiftregister mit Grösse 4 Bit hat einen Input-Pin und 4 Ausgangs-Pins. Intern hat das Shiftregister einen 4-Bit grossen

Speicher, von dem jedes Bit mit einem der Output-Pins korrespondiert. Jedes Mal, wenn der CLK-Pin geschaltet wird, verschiebt sich jedes Bit im Speicher nach rechts und der Wert im Input-Pin wird in das Bit ganz links geschrieben. Meist haben Shiftregister zusätzlich noch einen Clear-Pin (CLR), der den aktuell gespeicherten Wert auf 0 zurücksetzt.[1]

[1] vgl. AspenCore, Inc.: The Shift Register.

Kapitel 3

Material und Methodik

3.1 Dokumentation und Notizen

Ich erstellte zu Beginn der Arbeit ein OneNote-Notizbuch, das ich mit Herrn Müller teilte, sodass er den Fortschritt mitverfolgen konnte. Darin gab es 4 Abschnitte:

3.1.1 Allgemeines

In diesen Abschnitt schrieb ich vor dem Unterschreiben der Vereinbarung mit meinem Betreuer auf, wie ich mir die Arbeit vorstelle: welche Schritte involviert sein werden und wie lange diese grob geschätzt dauern werden. Der zweite Eintrag in diesen Abschnitt war eine Liste von möglichen Komponenten wie Display, Prozessoren usw., die ich für das Projekt gefunden habe.

3.1.2 Arbeitsprozess

Als im März das Thema feststand, erstellte ich eine Roadmap, also Daten, an denen ich fertig mit bestimmten Meilensteinen sein wollte. Bei mir waren diese sehr offen formuliert.

Woche 12	Display ansteuern
Woche 15	Auf Prototyping-Board alles verbinden
Woche 16	Beginn Design des PCBs
Woche 21	PCB bestellen
Woche 29	Fertige Hardware
Woche 43	Abgabe des Maturprojekts

In diesen Abschnitt stellte ich laufend alle Rechnungen der Bestellungen, die ich für das Projekt aufgab, hinein, damit ich einen Überblick über die Ausgaben haben konnte.

3.1.3 Protokoll

Hier führte ich ein Arbeitsprotokoll, worin ich meinen Fortschritt notierte. Dies habe ich geführt, um während des Schreibens der schriftlichen Arbeit einen besseren

Überblick zu haben, welche Schritte mir leicht fielen und welche mir Mühe oder Sorgen bereitet hatten.

3.1.4 Notizen

In diesen Abschnitt schrieb ich meine Notizen während der Arbeit, wie etwa der Aufbau einer bestimmten Datenstruktur oder Ideen bei der Gestaltung des Gehäuses.

3.2 Arbeitsprozess

3.2.1 Komponentenwahl

Mikrocontroller

Zu Beginn kamen zwei Prozessorarten infrage: Die ATmega-Reihe von Microchip und die MSP430-Serie von Texas Instruments. Hier sind einige dieser Mikrocontroller aufgelistet und verglichen.

Prozessor	**Flash (KB)**	**EEPROM (B)**	**RAM (B)**	**Preis (CHF)** [1]
MSP430G2553	16	0	512	2.47
ATmega1284	128	4'096	16'384	4.38
ATmega168A	32	1'024	2'048	1.96

Die MSP430-Serie besteht aus Mikrocontrollern, die einen sehr niedrigen Stromverbrauch haben. Auch die aufgelisteten ATmega-Chips werden in den Datasheets als Mikrocontroller mit geringem Stromverbrauch angegeben. Vergleicht man die zwei Serien, ist der Stromverbrauch sehr ähnlich. Im aktiven Modus haben die ATmega-Chips und der MSP430G2553-Chip einen Verbrauch von 0.2mA respektive 0.23mA bei einer Taktfrequenz von 1 MHz.

Ich kaufte mir ein Entwicklungskit für die MSP430-Serie, um mit diesen Chips zu spielen. Ich realisierte aber ziemlich schnell, dass 16KB Programmspeicher nicht genug für meine Anwendung war. 16KB war aber auch die grösstmögliche Speichergrösse der Serie, also wechselte ich auf die ATmega-Chips von Atmel.

Zwar gab es kein Entwicklungskit spezifisch für den ATmega1284, aber ich hatte einen Arduino UNO, der einen ATmega328P enthielt und somit praktisch denselben Zweck erfüllte. Mit diesem konnte ich deshalb schon recht weit kommen, bevor ich mir darüber Gedanken machen musste, welchen Chip ich für den Taschenrechner verwenden würde. Mit diesem Arduino UNO konnte ich dann auch den Chip auf dem PCB programmieren.

Display

Als ich mich auf die Suche nach einem geeigneten Display machte, erstellte ich aus der riesigen Auswahlmöglichkeit eine Liste mit ein paar wenigen dieser Displays.

[1] Stückpreis von `https://www.digikey.ch` beim Kauf von einem Stück, Stand 10.10.2020

Display	Pixel	Diagonale Sichtbereich (Zoll)	Preis (CHF) [2]
NHD-C12832A1Z	128x32	1.50	10.61
DOGM128W-6	128x64	2.35	20.97
Adafruit 3787	240x240	1.54	18.35

Das Display DOGM128W-6 von Electronic Assembly gefiel mir besonders, weil der Steuerungschip für das Display sich in einem kleinem Bereich über dem eigentlichen Displayteil befand (siehe Abbildung 3.1). Somit befand sich die Anzeige nicht auf einem separaten PCB und konnte direkt auf das PCB des Taschenrechners gelegt werden. Auch wenn es das teuerste dieser Liste war, entschied ich mich für dieses Display.

Abbildung 3.1: Das Display DOGM128W-6 auf einer Steckplatine.

Knöpfe

Als ich mir zum ersten Mal Gedanken darüber machte, welche Art Knöpfe ich benutzen soll, nahm ich meinen Schultaschenrechner auseinander und sah, dass die Plastikknöpfe auf eine Plastikmembran drücken, die eine Strombrücke auf dem PCB bildeten, wenn sie heruntergedrückt waren. Diese Membran herstellen lassen ist für ein Produkt, von dem es nur sehr wenige Endprodukte geben wird, nicht realisierbar. Also musste ich auf ganz normale Tastschalter zurückgreifen.

Auch wenn ich eine grosse Auswahl auf `https://digikey.ch` vorfand, war der Auswahlprozess ziemlich leicht. Ich wollte nämlich Knöpfe, auf die man eine Plastikkappe aufsetzen und somit die Knöpfe beschriften kann. Nachdem ich nach diesen Merkmalen filterte, sortierte ich nach aufsteigendem Preis und nahm die günstigsten Knöpfe. Diese sind in Abbildung 3.2 zu sehen. Schlussendlich brauchte ich diese Plastikkappen eigentlich gar nicht, weil ich mit dem 3D-Drucker ein Knopffeld drucken konnte, dass ich einfach auf das Knopfgitter legen konnte.

Abbildung 3.2: Der Tastschalter TL3301SPF260QG.

[2] Stückpreis von `https://www.digikey.ch` beim Kauf von einem Stück, Stand 15.10.2020

3.2.2 Prototyping

Auf mehreren Steckplatinen platzierte ich Komponenten wie Display und Knöpfe und verband sie miteinander. Abbildung 3.3 zeigt den letzten Stand des Prototyps, bevor ich das PCB bestellte. Man sieht auf der rechten Hälfte zuoberst den Arduino UNO, der alles steuert. Die neun blauen Komponenten sind 1μF-Kondensatoren, die zum Betrieb des Displays notwendig sind. Unter dem Display ist links der EEPROM zu sehen, der die Schriftart und Grafiken abspeichert. Auf der linken Steckplatine stecken zehn Tastschalter, die von einem Streifen Papier abgedeckt sind, der die Funktion der Knöpfe anzeigt. Die Sekundärfunktionen der Tasten, die blau auf dem Papierstreifen abgebildet sind, konnte man mit Druck auf den Knopf rechts unter dem Display aktivieren.

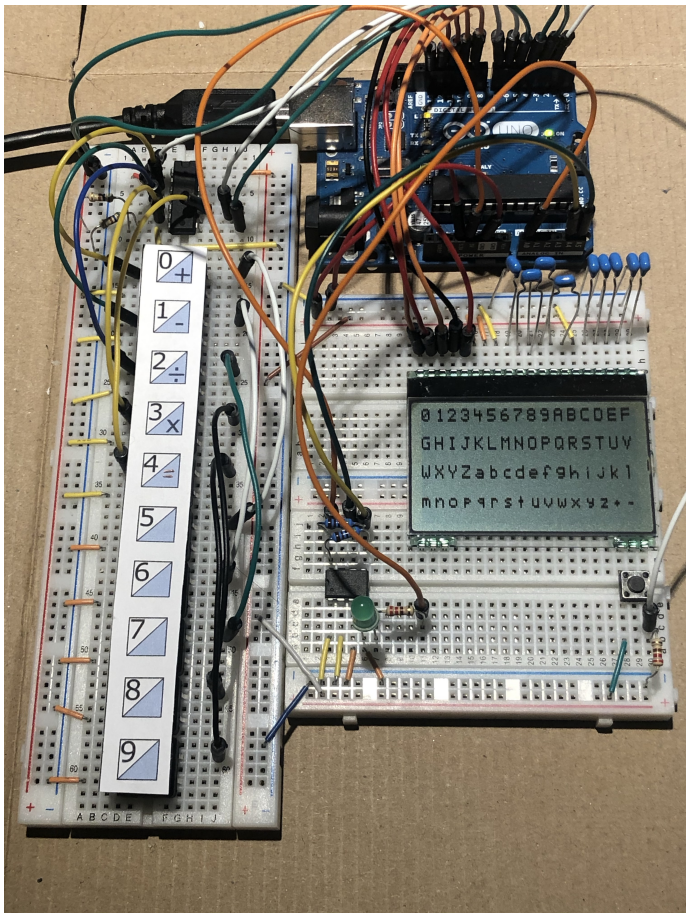


Abbildung 3.3: Der letzte Prototyp auf Breadboards.

3.2.3 Design des PCBs

Im Mai begann ich mit der Suche nach Programmen, mit denen PCBs erstellt werden können. Als Erstes stiess ich auf EasyEDA, eine Web-Applikation, die eine grosse Auswahl an Komponenten schon integriert hatte, weil man als Benutzer Komponenten, die noch nicht auf der Seite verfügbar sind, erstellen und hochladen kann. Persönlich gefiel mir das Interface nicht wirklich, daher suchte ich weiter, nachdem ich mich eine Weile mit EasyEDA auseinander gesetzt hatte.

Ich probierte als Zweites Autodesk EAGLE aus. Die reguläre Version der Software ist

in dem Fusion 360-Abonnement inbegriffen[3] und somit kostenpflichtig, aber es wird eine Version für den kostenlosen Gebrauch angeboten, die bestimmte Limitationen hat. Unter anderem ist die PCB-Grösse auf 80cm^2 beschränkt, was ich aber vorerst nicht bemerkte, weil zu Beginn noch eine 30-tägige Testversion aktiv war. Das Display musste mindestens eine Breite von 60mm einnehmen, also hätte die Länge unter 135mm sein müssen, worauf ich kaum alle geplanten Komponenten hätte plazieren können. Somit sah ich mich gezwungen, erneut das Programm zu wechseln. Ich entschied mich für KiCad, eine Open-Source-Software.

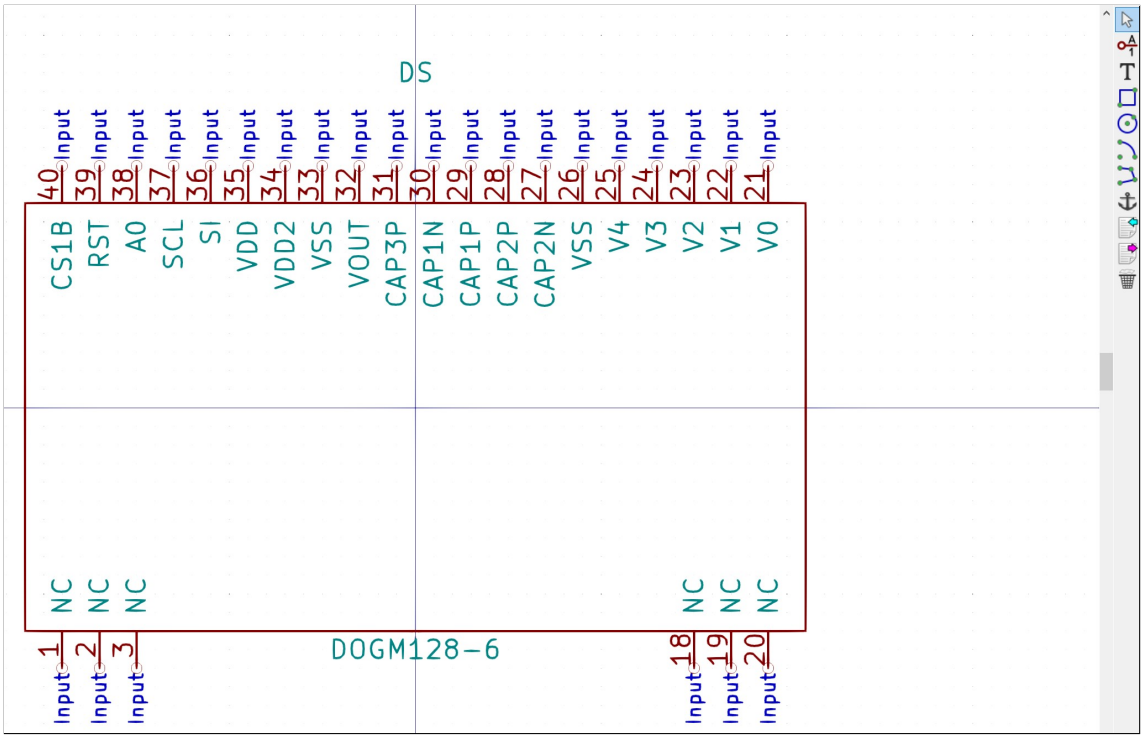


Abbildung 3.4: Der Symboleditor in KiCad.

In KiCad war das Display, dass ich benutzen wollte, das DOGM128W-6 von Electronic Assembly, nicht in der Footprint-Bibliothek vorhanden. Ich musste daher das Symbol für das Schema und den Footprint selbst erstellen. Ich begann mit dem Symbol und erstellte ein Rechteck, wo ich die Pins einsetzte, beschriftete und durchnummerierte (siehe Abbildung 3.4). Für den Footprint gab es wesentlich mehr Arbeit, weil die Maschinen, die das PCB herstellen, wissen müssen, wo die Löcher für die Pins genau sind. Ich musste also die genauen Dimensionen und Abstände mit dem Datasheet des Displays herausfinden. Danach konnte ich anhand dieser Messungen im Programm den Footprint entwerfen. Ich druckte das PCB-Layout auf Papier aus und legte das Display darauf und konnte so verifizieren, dass die Dimensionen korrekt waren.

[3]siehe `https://www.autodesk.com/products/eagle/free-download` (Abgerufen 13.10.2020)

3.2.4 Erster Prototyp auf PCB

Das PCB bestellte ich bei einer chinesischer PCB-Fabrik namens JLCPCB. Es gibt zwar Möglichkeiten, PCBs in Europa herstellen lassen, aber die Preise sind viel günstiger bei den chinesischen Angeboten. Zum Vergleich: Der Preis, fünf meiner Taschenrechner-PCBs bei JLCPCB zu bestellen, ist 23.30 CHF mit Versandkosten[4]. Bei Eurocircuits kosten fünf PCBs hingegen 112 CHF[5], was keineswegs ein Ausnahmefall ist.

Anfangs Juli kam das bestellte PCB an. Weil die meisten Komponenten SMD-ICs waren, wäre es sehr mühsam gewesen, diese mit einem gewöhnlichen Löteisen anzulöten. Stattdessen kaufte ich eine SMD Reworkstation, eine Art Heissluftföhn, dessen Temperatur präzis eingestellt werden kann. Um mit der Reworkstation die Komponenten anlöten, muss zuerst auf die Kontakte Lötpaste gestrichen werden. Diese besteht aus kleinen Bällchen Lötzinn gemischt mit Flussmittel, die mit Hitze zusammenschmelzen und zwischen den Komponenten und den Pads eine elektrische Verbindung bilden[6]. Aber auch mit diesen Hilfsmitteln hatte ich grosse Schwierigkeiten, die kleinen SMD-Komponenten wie den Mikrocontroller anzulöten. Entweder gab ich zu viel Lötpaste auf die Pads und mehrere Pins verbanden sich miteinander; es bildeten sich sogenannte Brücken, oder ich benutzte zu wenig Lötpaste und es bildete sich keine Verbindung zwischen den Pins und Pads.

Als dann die ICs, das Display und etwa die Hälfte der Knöpfe angelötet waren, versuchte ich, mit dem Mikrocontroller über den Arduino UNO zu kommunizieren, ihn zu programmieren und das Display und die Knöpfe anzusteuern. Leider musste ich unzählige Male den Chip ab- und wieder anlöten und auf Lötbrücken testen, bis ich nach ein paar Tagen einen ersten Erfolg hatte und der Arduino den ATmega1284 erkannte.

Das Mühsamste an der Fehlersuche während dieser Zeitspanne, wo ich versuchte, den PCB-Prototyp zum Laufen zu bringen, war die Unsicherheit, ob es sich um ein Software- oder Hardwareproblem handelte. Ich rechnete zwar meistens damit, dass es ein Hardwareproblem war, aber einige tückische Programmierfehler schlichen sich von Zeit zu Zeit ein, die ich lange übersah, weil ich an der Hardware nach Fehlern suchte. Wie sich schlussendlich herausstellte, gab es zwei grosse Fehler am PCB-Prototypen:

1. Ich hatte bei den Knöpfen falsch verstanden, welche Pins immer verbunden sind und welche nur bei Knopfdruck Strom durchfliessen lassen. Dies führte dazu, dass der Mikrocontroller immer alle Knöpfe als gedrückt sah.

2. Das Grundpotenzial des Akkus floss nicht zu allen Komponenten, sondern nur zum Grundpotenzial des Auflade-ICs. Somit konnte der Akku dem Taschenrechner keinen Strom liefern.

Den ersten Fehler konnte ich temporär beheben, indem ich die Knöpfe um 30 Grad drehte, sodass die Pads, zwischen denen Strom bei Knopfdruck fliessen soll, mit den Pins der Knöpfe, die tatsächlich nur bei Knopfdruck Strom leiten, verbunden sind.

[4] siehe `https://jlcpcb.com/` (Aufgerufen 17.10.2020)
[5] siehe `https://www.eurocircuits.com/` (Aufgerufen 17.10.2020)
[6] vgl. RS Components GmbH: Lötpasten.

Das zweite Problem liess sich nicht so einfach umgehen und ich musste auf eine zweite Version des PCBs warten. Weil aber der Arduino UNO über die Programmierpins auch Strom liefert, konnte ich trotzdem am Taschenrechner arbeiten.

3.2.5 Zweiter Prototyp auf PCB

In KiCad behob ich die beiden erwähnten Probleme. Zusätzlich fügte ich neben dem USB-Port noch vier Pins hinzu, die von links nach rechts an die GND-, TXD-, RXD- und 5V-Pins der Mikrocontollers angeschlossen sind. Mit diesen Pins war es möglich, Sensoren oder andere Erweiterungen anschliessen. Der Plan war, mit dem Taschenrechner eine erste Anwendung zu finden auf einer Gletscherexkursion, wo mit einem angeschlossenen Barometer-Sensor die aktuelle Höhe berechnet werden konnte (Mehr dazu im Kapitel 6). Ich bestellte das neue PCB, sowie die Komponenten ein zweites Mal. Dieses Mal schien alles zu klappen und auch der Akku konnte dem Taschenrechner Strom liefern. Als ich dann aber versuchte, den Taschenrechner über den USB-Port aufzuladen, stieg die Spannung des Akkus, die ich mit einem Multimeter mass, nicht. Grund war, dass der 0V-Pin des Auflade-ICs zum 0V-Pin des USB-Ports ging und nicht zum Minuspol des Akkus. Dieses Problem konnte ich zum Glück mit einem Kabel, dass von einem 0V-Pin auf dem Taschenrechner zu dem 0V-Pin des USB-Ports ging (in Abbildung 5.1 zu sehen), lösen. Ausser diesem Fehler gab es keine anderen Hardwareprobleme.

3.2.6 Programmentwicklung

Das Programm "Visual Studio Code" von Microsoft ist eine Programmierumgebung, die aber nicht speziell für eine spezifische Programmiersprache entwickelt wurde, sondern eher ein Code-Editor ist, der mit Erweiterungen für viele Programmiersprachen Unterstützung bietet. Es enthält ein Dateibrowser, Git-Integrierung und vieles mehr.

3.2.7 Benennung des Taschenrechners

Als es an der Zeit war, ein Projekt auf GitHub, einer Version Control-Plattform, zu erstellen, musste ich mir einen Namen einfallen lassen. Nach nicht besonders langem Überlegen entschied ich mich, einen temporären Namen zu geben, den ich vorhatte, später abzuändern: CALCKS. Der Name stand für "**Calc**ulator from a **K**ZO **s**tudent". Schlussendlich nahm ich aber doch nie eine Namensänderung vor und der Name des Taschenrechners blieb bei CALCKS.

3.2.8 Gehäuse

Ich hatte mir schon seit einiger Zeit einen 3D-Drucker aus diversen Gründen kaufen wollen, und das Maturprojekt sah ich als einen guten Zeitpunkt, diesen Kauf zu tätigen, weil das Gehäuse mit einem 3D-Drucker gut herstellbar ist. Die Schwierigkeit bei der Wahl war, dass der Drucker eine genug grosse Druckfläche haben musste, aber trotzdem noch im meinem Budget von circa 300-400 Franken liegen sollte. Ich entschied mich schlussendlich für den Ender 3 Pro von Creality und kaufte dazu noch schwarzes PLA-Filament.

Dies bedeutete, dass ich ein weiteres neues Programm für das Design von druckbaren

3D-Objekten erlernen musste. Das Programm Fusion 360 ist auch ein Autodesk-Produkt, aber zu diesem war eine Lizenz verfügbar, das für Schüler und Studenten die Basisversion des Programms gratis zur Verfügung stellt[7]. Wie ich später feststellte, hätte ich diese Lizenz auch für das Erstellen des PCBs brauchen können, aber weil das PCB zu dem Zeitpunkt schon fertig war, machte es keinen Sinn mehr, zurückzuwechseln.

Auf alle Fälle begann ich mit dieser Software, das Gehäuse zu entwerfen. Ich versuchte als erstes, Knopfkappen mit Zahlen zu modellieren und zu drucken. Nach langem Experimentieren hatte ich zwar mehr oder weniger funktionsfähige Knopfkappen (siehe Abbildung 3.5), aber ich realisierte, dass es mehr Sinn machen würde, einen sehr dünnen und somit flexiblem Boden mit herausragenden Knöpfen auszudrucken, weil dann weniger zusammengesteckt werden musste und ich mir keine Sorgen darüber machen musste, dass die Knopfkappen abfallen könnten.

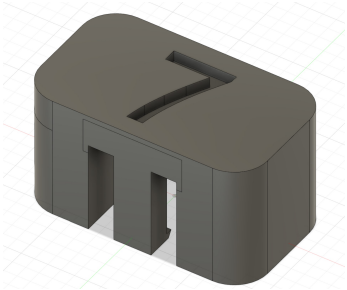


Abbildung 3.5: Eine Knopfkappe, entworfen in Fusion 360.

Auch das Design des Gehäuses erforderte viel Probierarbeit und Geduld, weil das Ausdrucken der Gehäuseteile zwei bis sieben Stunden dauerte. Aus diesem Grund konnte ich höchstens alle zwei Tage einen neuen Druck haben, weil ich dazwischen die Fehler beheben und dann auf den Druck warten musste.

[7] siehe `https://www.autodesk.com/campaigns/education/fusion-360` (Abgerufen 16.10.2020)

Kapitel 4

Technische Umsetzung

4.1 Hardware

4.1.1 Knopfmatrix

Der ATmega1284 (im Packaging-Typ 44-TQFP) hat 44 Pins, von denen 32 Pins als Input/Output verwendet werden können[1]. Der Taschenrechner hat aber 40 Knöpfe, also konnte ich nicht jeden einzelnen Knopf an einen Pin anschliessen. Stattdessen verwendete ich die folgende Methode:

Die Knöpfe sind grundsätzlich, wie in Abbildung 4.1, in einem 7x5-Gitter angeordnet, aber solange die Verbindungen gleich bleiben, können die tatsächlichen Positionen der Knöpfe willkürlich gesetzt werden. Bei diesem Gitter sind alle rechten Pins der Knöpfe jeder Spalte miteinander verbunden und führen zu den Pins des Mikrocontrollers (in der Abbildung: BTN_ROW1–BTN_ROW5). Alle linken Pins der Knöpfe in einer Reihe sind miteinander verbunden und führen zu je einem Output-Pin des 8-Bit Serial-In-Parallel-Out Shiftregister-ICs (siehe Kapitel 2.1.6) SN74HC164. Die vier Pins CLR, $\overline{\text{CLK}}$, A und B führen zu Pins des Mikrocontrollers. A und B werden intern im Shift-Register mit einem UND-Gatter kombiniert und haben die Funktion des Input-Pins, weshalb A und B zusammenführen und zum selben Pin am ATmega1284 führen.

Will der Taschenrechner auf gedrückte Knöpfe überprüfen, sendet er einen Startpuls zum Shiftregister, was dazu führt, dass Strom durch die erste Reihe fliesst. Ist nun ein Knopf in dieser Reihe gedrückt, leitet der Knopf den Strom weiter, durch die Spalte, in welcher der Knopf ist. Der Mikrocontroller kann somit, falls ein Signal bei einem der fünf Reihen-Pins landet, genau wissen, welcher Knopf gedrückt ist. Danach betätigt der Mikrocontroller den CLK-Pin und der Strom fliesst durch die nächste Reihe. Der $\overline{\text{CLK}}$-Pin wird weitere fünf Male betätigt, bis er bei der letzten Reihe ankommt. Danach wird das Shiftregister über den CLR-Pin zurückgesetzt. Ein Problem entsteht hierbei: Der Mikrocontroller muss einen weiteren IC ansteuern, um die Zustände der Knöpfe auszulesen. Wenn aber der ATmega in den Schlafmodus geht, führt er auch keinen Code mehr aus. Das Aufwachen aus dem Power-down-Zustand kann er nur durch eine direkte Pinänderung oder eine Adressierung des Chips über

[1] vgl. Microchip Technology: megaAVR® Data Sheet. 2020, S. 2.

das TWI-Modul[2].

Aus diesem Grund enthält der Taschenrechner fünf zusätzliche Knöpfe, die direkt an den Mikrocontroller angeschlossen sind. Dies ist der Anschalteknopf und die vier Richtungsknöpfe.

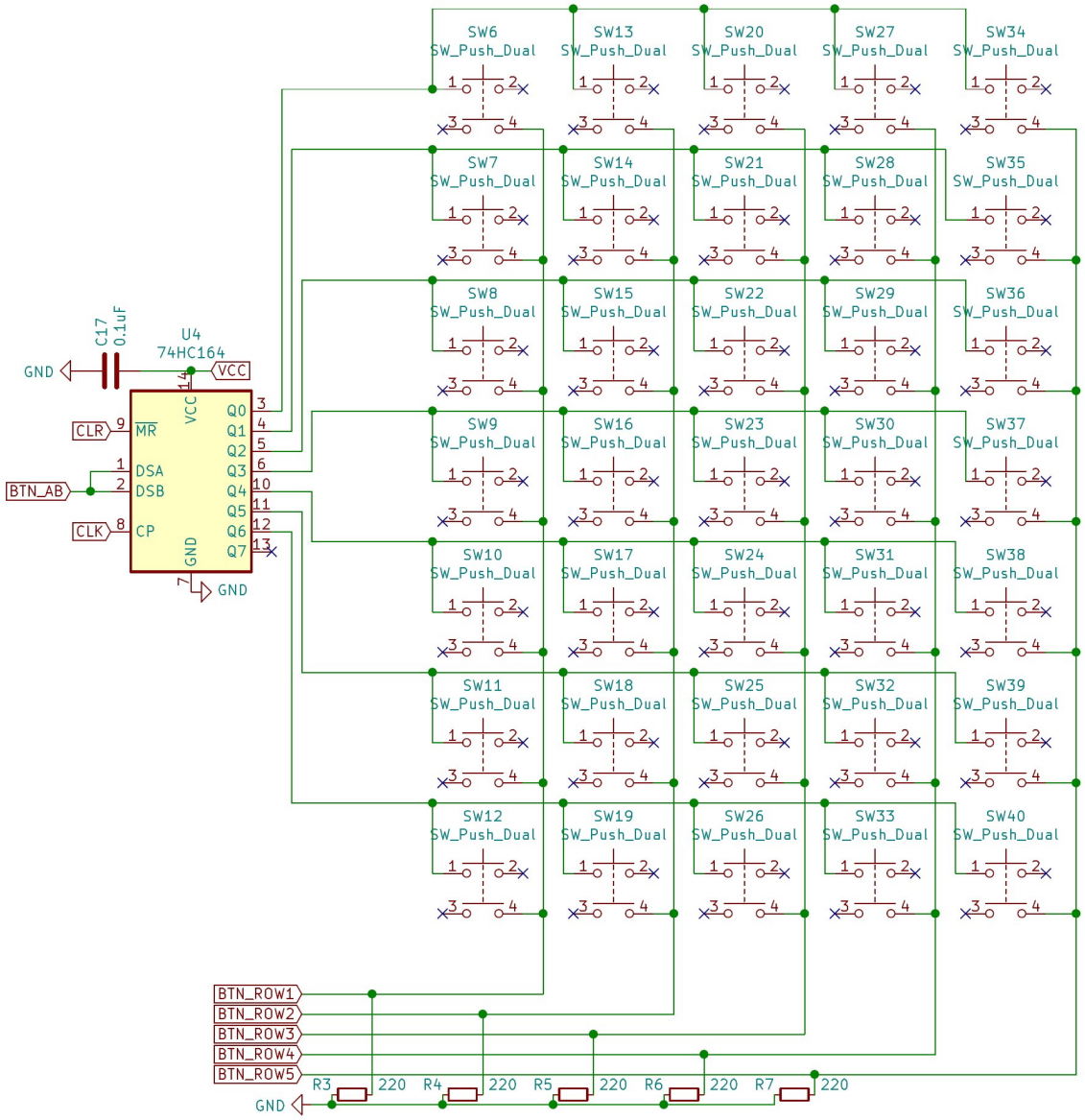


Abbildung 4.1: Das Knopfgitter im Schema.

4.1.2 PCB

Das Schema und eine gerenderte Version des PCB-Layouts ist im Anhang vorzufinden.

4.1.3 Stromversorgung

Ich entschied mich schlussendlich, einen wiederaufladbaren Lithium-Polymer-Akku zu verwenden. Die benötigten Komponenten, damit der Akku dem Taschenrechner

[2]vgl. Microchip Technology: megaAVR® Data Sheet. 2020, S. 50.

Strom liefern und wieder aufgeladen werden kann, sind die folgenden:

- Ein IC, der die richtige Ladespannung generieren kann und automatisch beim Erreichen des aufgeladenen Zustands aufhört, Strom an den Akku zu übertragen.
- Ein Mikro-USB-Anschluss, damit der Taschenrechner über eine universelle Schnittstelle aufgeladen werden kann.
- Ein IC, der den Akku vor Tiefentladung schützt. Diesen könnte man theoretisch weglassen, aber wenn ein Akku unter der minimalen Spannung sinkt, reduziert dies die Lebensdauer des Akkus stark.

Der Akku ist ein Lithium-Polymer-Akku von Renata SA, der eine Nominalspannung von 3.7V und eine Nominalkapazität von 450mAh hat. Dieser Akku hat einen Schutzkreislauf bereits integriert, der ihn automatisch bei 3.0V ausschaltet.[3]

Für den Auslade-IC sah ich zwei Möglichkeiten. Ich hätte ein vorgefertigtes Akku-Auflade-PCB kaufen können, welches schon einen Mikro-USB Anschluss und 4 Lötpads hat, die zum Akku und zum Haupt-PCB führen. Die andere Möglichkeit war, einen Auflade-IC direkt auf dem PCB, wo alle anderen Komponenten darauf lagen, zu integrieren. Der Vorteil hier war, dass ich nicht ein zweites PCB in das Gehäuse montieren müsste. Der Nachteil war, dass ich so ein grösseres Risiko einginge, etwas im Schema falsch zu verbinden. Nach einer Weile Recherche entschied ich mich, den IC auf direkt auf dem PCB zu platzieren und wählte den Auflade-IC MCP73831 von Microchip. Dieser hat ausser den Spannungspins einen PROG-Pin, der die Ladestromstärke bestimmt und einen STAT-Pin, der angibt, ob der Akku gerade aufgeladen wird, voll geladen ist oder nicht aufgeladen wird.[4]

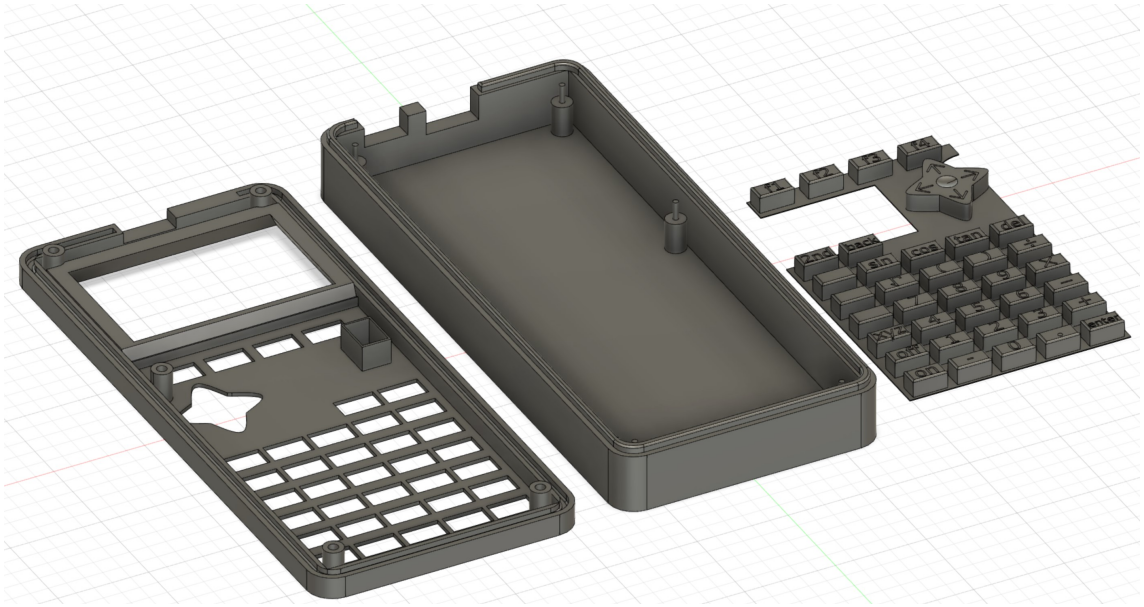
4.1.4 Externer EEPROM

Auf dem PCB befindet sich ein externer EEPROM, der urspünglich geplant war, die Schriftart und das Aufstartlogo abzuspeichern. Dieser musste diesen Zweck aber schlussendlich nicht erfüllen, weil der 4KB grosse EEPROM auf dem ATmega1284 ohnehin genug Platz hatte. Ich entschied mich, den externen EEPROM aber im PCB beizubehalten, falls ich in der Zukunft mit dem Taschenrecher zum Beispiel Messdaten aufnehmen und speichern will. Dafür hat es mit diesem 64KB grossen EEPROM mehr als genug Platz.

4.1.5 Gehäuse

Das Gehäuse ist aus drei Teilen aufgebaut: der Basis, worin das PCB liegt, dem Deckel, der Ausschnitte für die Knöpfe hat, und dem Knopffeld.

[3] vgl. Renata SA: Datasheet ICP582930PR-01. 2019, S. 1.
[4] vgl. Microchip Technology: MCP73831/2. 2020, S. 11.



In der Basis hat es fünf herausragende Zylinder, die durch die Löcher im PCB gehen sollten und unten für den Akku Platz freimachen. Obwohl ich unzählige Male nachgemessen und berechnet habe, wo diese genau herausragen müssten, waren nie alle am ganz korrekten Ort platziert. Stattdessen wurden beim Zusammenbau vier der fünf kleinen Zylinder entfernt, damit das PCB in das Gehäuse gelegt werden kann, aber weil der Umriss aussen gut gepasst hat, merkt man kaum einen Unterschied. Es war ursprünglich geplant, mit Schrauben das PCB und das Gehäuse festzuhalten, aber aus dieser Idee wurde nichts. Stattdessen hat der Deckel eine Einbuchtung im Rand, die auf das herausragende Stück in der Basis passt und das Knopffeld wird zwischen das PCB und den Deckel gelegt.

4.2 Software

Das Projekt ist auf GitHub unter `https://github.com/LetsPlentendo-CH/CALCKS` gehostet, einerseits, damit ich von verschieden Geräten aus weiterarbeiten konnte, andererseits damit der Quellcode öffentlich sichtbar für Leser dieser Arbeit ist. Die Projektstruktur ist wie folgt gegliedert (nur Ordner abgebildet):

```
CALCKS
├── Designs
├── Firmware
│   ├── avr
│   ├── binsend
│   ├── common
│   ├── console
│   ├── eepromgen
│   └── server
└── Hardware
    ├── 3D Models
    └── CALCKS
```

4.2.1 Kompilation

Im `Firmware`-Verzeichnis befinden sich eigentlich zwei Versionen der Taschenrechner-Software. Die erste wird für die tatsächliche Hardware, also den ATmega1284, kompiliert. Die zweite wird für die Windows, Mac oder Linux-Konsole kompiliert. Bei dieser Version wird der Bildschirm und die Knöpfe durch Konsolen-Ein- und Ausgabe ersetzt, sodass der simulierte Bildschirm mit einem weiteren Programm angezeigt werden kann. Es wäre aber ziemlich unschön und unübersichtlich, die Codeausschnitte, welche in den zwei Versionen identisch sind, zu duplizieren. Deshalb ist dieser identische Code im Verzeichnis `common`.

Da für den Grossteil des Codes die Hardware mittels selbst geschriebenen Grafik- und Inputlibraries abstrahiert wird, ist es nicht so schwer, die Firmware als Konsolenapplikation zu kompilieren. Dazu müssen nur die Display- und Knopffunktionen neu geschrieben werden, sodass über die Konsole Knopfdrücke eingegeben und der aktuelle Displaystand ausgegeben werden können. Die hardwarespezifischen Dateien sind in `Firmware/avr` für den Mikroprozessor und `Firmware/console` für den Emulator.

4.2.2 Programmtransfer auf das Gerät

Von dem ATmega1284 gehen die Pins VCC, GND, MOSI, MISO, SCK und $\overline{\text{RESET}}$ zu einem 2x3-Pin-Header. An diesen werden Steckbrückenkabel an einen AVR-Programmer, in meinem Fall an einen Arduino mit einem laufenden ISP-Programm[5], angeschlossen. Das Programm `avrdude` transferiert das kompilierte Programm über den Arduino zum ATmega1284.

4.2.3 Implementation: Text und Grafiken

Für die Text- und Grafikdarstellung sind die Dateien `display.c`, `gui.c` und die dazugehörigen Header-Dateien verantwortlich. `display.c` sendet die individuellen Befehle an das 128x64 Pixel grosse Display. Das Datasheet des Displaydriver[6] (Abbildung 4.2) beschreibt die Werte für die vorhandenen Befehle in einer übersichtlichen Tabelle.

In der Headerdatei `display.h` befinden sich für jeden Befehl #define-Statements, um das Senden dieser Befehle für den Programmierer leserlicher zu machen. Ein kleiner Ausschnitt aus dem Code:

```
...
#define DISP_CMD_START_LINE 0x40
#define DISP_CMD_PAGE 0xB0
#define DISP_CMD_COL_MSB 0x10
#define DISP_CMD_COL_LSB 0x00
#define DISP_CMD_ADC 0xA0
...
```

[5]Dieses Programm ist in der regulären Arduino-IDE in den Beispielen verfügbar.
[6]Sitronix: Datasheet ST7565R. 2006, S. 50.

Table 16: Table of ST7565R Commands (Note) *: ignored data

Command	Command Code											Function
	A0	/RD	/WR	D7	D6	D5	D4	D3	D2	D1	D0	
(1) Display ON/OFF	0	1	0	1	0	1	0	1	1	1	0 1	LCD display ON/OFF 0: OFF, 1: ON
(2) Display start line set	0	1	0	0	1	Display start address						Sets the display RAM display start line address
(3) Page address set	0	1	0	1	0	1	1	Page address				Sets the display RAM page address
(4) Column address set upper bit Column address set lower bit	0	1	0	0	0	0	1	Most significant column address				Sets the most significant 4 bits of the display RAM column address. Sets the least significant 4 bits of the display RAM column address.
				0	0	0	0	Least significant column address				
(5) Status read	0	0	1	Status				0	0	0	0	Reads the status data
(6) Display data write	1	1	0	Write data								Writes to the display RAM
(7) Display data read	1	0	1	Read data								Reads from the display RAM
(8) ADC select	0	1	0	1	0	1	0	0	0	0	0 1	Sets the display RAM address SEG output correspondence 0: normal, 1: reverse
(9) Display normal/ reverse	0	1	0	1	0	1	0	0	1	1	0 1	Sets the LCD display normal/ reverse 0: normal, 1: reverse
(10) Display all points ON/OFF	0	1	0	1	0	1	0	0	1	0	0 1	Display all points 0: normal display 1: all points ON
(11) LCD bias set	0	1	0	1	0	1	0	0	0	1	0 1	Sets the LCD drive voltage bias ratio 0: 1/9 bias, 1: 1/7 bias (ST7565R)
(12) Read-modify-write	0	1	0	1	1	1	0	0	0	0	0	Column address increment At write: +1 At read: 0
(13) End	0	1	0	1	1	1	0	1	1	1	0	Clear read/modify/write
(14) Reset	0	1	0	1	1	1	0	0	0	1	0	Internal reset
(15) Common output mode select	0	1	0	1	1	0	0	0 1	*	*	*	Select COM output scan direction 0: normal direction 1: reverse direction
(16) Power control set	0	1	0	0	0	1	0	1	Operating mode			Select internal power supply operating mode
(17) V_0 voltage regulator internal resistor ratio set	0	1	0	0	0	1	0	0	Resistor ratio			Select internal resistor ratio(Rb/Ra) mode
(18) Electronic volume mode set Electronic volume register set	0	1	0	1	0	0	0	0	0	0	1	Set the V_0 output voltage electronic volume register
				0	0	Electronic volume value						
(19) Static indicator ON/OFF Static indicator register set	0	1	0	1	0	1	0	1	1	0	0 1	0: OFF, 1: ON Set the flashing mode
				0	0	0	0	0	0	0	Mode	
(20) Booster ratio set	0	1	0	1	1	1	1	1	0	0	0	select booster ratio 00: 2x,3x,4x 01: 5x 11: 6x
				0	0	0	0	0	0	step-up value		
(21) Power save	0	1	0									Display OFF and display all points ON compound command
(22) NOP	0	1	0	1	1	1	0	0	0	1	1	Command for non-operation
(23) Test	0	1	0	1	1	1	1	*	*	*	*	Command for IC test. Do not use this command

Abbildung 4.2: Die Operationen des Displaytreibers.

Das Datenblatt des LCD-Displays enthält auch eine empfohlene Startsequenz[7], die das Display zurücksetzt, den Kontrast setzt und einiges weiteres. Diese kann nun mithilfe diesen #define-Statements ziemlich leserlich dargestellt werden.

[7]Electronic Assembly: Datasheet DOGM128-6. 2009, S. 6.

```
static const char disp_initSequence[DISP_INIT_LEN] = {
    DISP_CMD_START_LINE | 0b000000, // Start at line 0
    DISP_CMD_ADC | 0,               // ADC normal -> left to right
    DISP_CMD_COM_OUT_DIR | 0b1000,  // Select reverse common output
    DISP_CMD_DISP_DIR | 0,          // Set Display direction normal
    DISP_CMD_LCD_BIAS | 0,          // Set bias 1/9 - duty 1/65
    DISP_CMD_POWER_CTRL | 0b111,    // Power control: Booster on,
   Regulator on, Follower on
    DISP_CMD_BOOSTER_RATIO_MODE,    // Booster ratio:
    0x00,                           // 4x
    DISP_CMD_V0_RATIO | 7,          // V0 Voltage set: 7
    DISP_CMD_ELVOL_MODE,            // Electronic volume mode:
    0x16,                           // 22
    DISP_CMD_INDICATOR_ONOFF | 0,   // Static indicator off
    0x00,                           // Flashing mode: 0
    DISP_CMD_ALL_ONOFF | 0,         // Display all points: no
    DISP_CMD_ONOFF | 1              // Display on
};
```

Beim Aufstarten des Taschenrechners wird dieser Array durchiteriert und jeder Befehl an das Display gesendet.

Im EEPROM des ATmega1284 sind zwei Schriftarten gespeichert: eine mit Auflösung 6x7 Pixel pro Zeichen und eine mit Auflösung 8x16 Pixel (siehe Abbildung 4.3 und 4.4). Die Schiftarten habe ich selbst erstellt in Aseprite, einem Programm, dass für das Zeichnen von Bildern niedriger Auflösung gedacht ist.

Abbildung 4.3: Der 6x7-Font.

Abbildung 4.4: Der 8x16-Font.

Sowohl diese Schriftarten als auch das Aufstartbild werden mit einem *node.js*-Programm in eine Binärdatei umgewandelt und mit `avrdude` auf den EEPROM des Mikrocontrollers kopiert.

Im Verzeichnis `Firmware/binsend` befindet sich ein mittlerweise obsoletes Programm, dass von der Zeit des Prototyps auf der Steckplatine stammt. Es wurde auf den Arduino geladen und schrieb die Schriftart und das Aufstartlogo, die der Arduino über USB empfing, in den externen EEPROM.

4.2.4 Implementation: Termanalyse

Das zentralste Element der Software für den Taschenrechner ist das Analysieren von mathematischen Termen. Dies ist nicht nur essentiell für das Ausrechnen von Termen im Hauptbildschirm, sondern auch für das Zeichnen von Graphen. Den Algorithmus, den ich für das Parsen verwendete, ist eine modifizierte Variante des Shunting-yard Algorithmus. Das Grundprinzip ist, dass man von links nach rechts den Term durchläuft und mithilfe eines Stacks die Operandenpriorität erkannt wird. In der Originalvariante des Shunting-yard-Algorithmus entsteht aus einem Term in

der Infixnotation, also der, die wir aus dem alltäglichen Leben kennen, in einen Term in der Postfixnotation [8]. Diese Postfixnotation hat den Vorteil, dass keine Kenntnisse von Operandenprioritäten wie Punkt vor Strich notwendig sind. Stattdessen steht der Operand immer nach den zwei Operatoren. Zwei Beispiele:

`4 + 3` wäre in der Postfixnotation `4 3 +` und `5 * 9 + 2` wäre `5 9 * 2 +`.

Ich habe mich aber entschieden, den Term nicht in eine weitere lineare Struktur umzuformen, sondern in einen Ausdrucksbaum. Dies verkompliziert zwar den Umwandlungsalgorithmus, vereinfacht aber das Ausrechnen eines Terms.

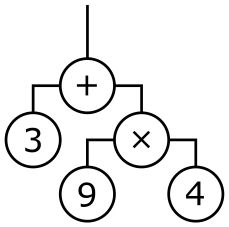


Abbildung 4.5: Die häufigste Darstellung eines Ausdrucksbaums.

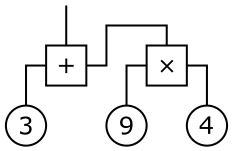


Abbildung 4.6: Eine kompaktere Darstellung eines Ausdrucksbaums.

Abbildung 4.5 zeigt die Darstellung eines Ausdrucksbaums, wie man ihn im Mathematikunterricht kennenlernt. Beim Erklären meines Algorithmus macht es aber meiner Meinung nach mehr Sinn, sie etwas kompakter und horizontal darzustellen, wie in Abbildung 4.6. Bei dieser Darstellung sind die Operanden in Quadraten in der oberen Reihe dargestellt, die Zahlen in Kreisen in der unteren Reihe. Die wichtigste Datenstruktur in meiner Implementierung ist der `opNode`-struct. Dies ist ein Knoten im Ausdrucksbaum, der die dazugehörige mathematische Operation, die beiden Operanden und der Typ der Operanden speichert. Der Algorithmus folgt nun in Pseudocode.

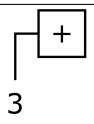
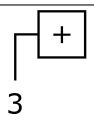
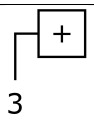
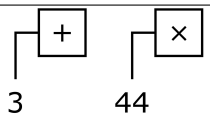
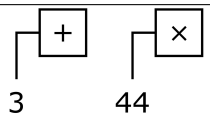
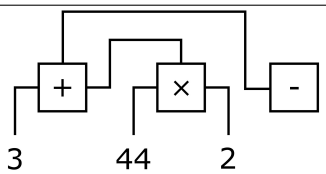
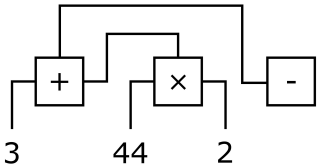
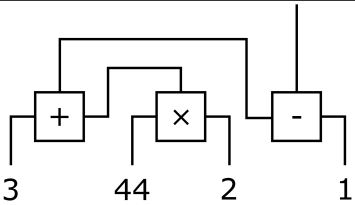
[8] vgl. Hafizji, Ali: Shunting Yard Algorithm

Algorithm 1: Umwandlung Infixnotation in Operationsbaum

```
currValue ← 0;
operatorStack ← [];
define latestOp;
while current character ≠ end of input do
    curr ← current character;
    if type of curr = number then
        currValue ← currValue ∗ 10 + curr;
    else if type of curr = operator then
        latestOp ← create opNode(curr);
        latestOp.leftOperand ← currValue;
        latestOp.operation ← operation of curr;
        if operatorStack.size ≠ 0 then
            if precedence(latestOp.operation) ≤ precedence(operatorStack.top.operation)
             then
                currStackOp ← operatorStack.top;
                while currStackOp ≠ undefined and precedence(latestOp.operation) ≤
                 precedence(currStackOp.operation) do
                    currStackOp ← currStackOp.prev;
                latestOp.leftOperand ← currStackOp;
                while currStackOp.next ≠ undefined do
                    nextOpInStack ← currStackOp.next;
                    currStackOp.rightOperand ← nextOpInStack;
                    remove currStackOp from operatorStack;
                    currStackOp ← nextOpInStack;
                currStackOp.rightOperand ← currValue;
                remove currStackOp from stack;
        operatorStack.push(latestOp);
        currValue ← 0;
    Move to next character in input;
latestOp.rightOperand ← currValue;
remove latestOp from operatorStack;
while operatorStack.size ≠ 0 do
    operatorStack.top.rightOperand ← latestOp;
    latestOp ← operatorStack.top;
    remove operatorStack.top from operatorStack;
/* Die oberste Operation im Ausdrucksbaum is nun latestOp.          */
```

In der Form von Programmcode gibt es noch einige Spezialfälle, die eingebaut werden mussten. Zum Beispiel: Wenn eine Eingabe nur aus einer einzigen Zahl oder Variable besteht, hat der Algorithmus keinen Operator-Knoten, wo diese einzelne Zahl oder Variable eingesetzt werden könnte. Stattdessen erstellt das Programm in diesem Fall einen Plus-Operator-Knoten mit 0 als erster Operand und der Zahl oder Variable als zweiter Operand. Im Pseudocode sind ausserdem keine Klammern erwähnt, weil diese den Pseudocode sehr unleserlich gemacht hätten. Interessierte Leser können die volle Implementation in der Datei `Firmware/common/term.c` finden.

Es folgt eine Anwendung dieses Algorithmus am Beispielsterm $3 + 44 * 2 - 1$. Der erste Schritt stellt den Ausgangszustand dar. Jeder folgende Schritt ist der Zustand nach jedem Durchlauf der While-Schlaufe, die über alle Zeichen iteriert. Der letzte Schritt ist der Zustand am Ende des Algorithmus.

Schritt	**currValue**	**Operatoren**	**operatorStack**
1	0	(leer)	(leer)
2	3	(leer)	(leer)
3	0		+
4	4		+
5	44		+
6	0		+, ∗
7	2		+, ∗
8	0		−
9	1		−
10	0		(leer)

Kapitel 5

Resultate

5.1 Aussehen

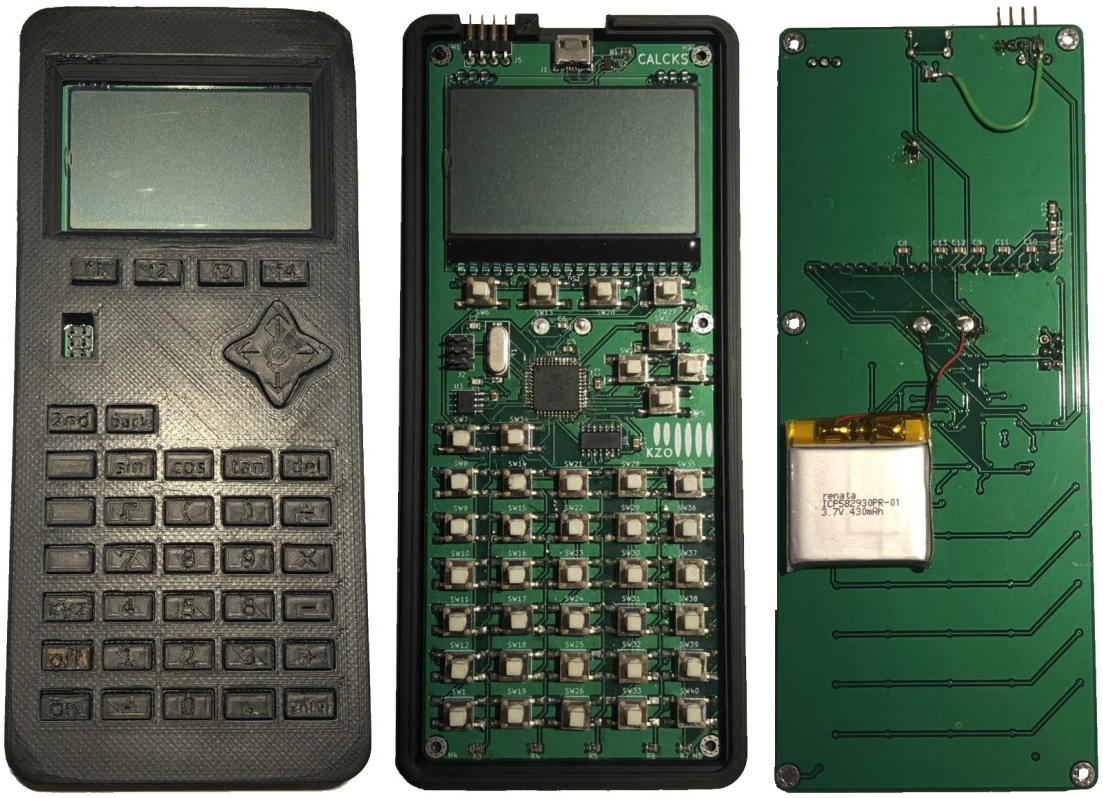


Abbildung 5.1: Drei Ansichten des Taschenrechners.

Abbildung 5.1 zeigt den Taschenrechner in dem Gehäuse und die Ober- und Unterseite des PCBs. In der dritten Abbildung sieht man oben rechts das in Kapitel 3.2.5 erwähnte Kabel, das angelötet wurde, damit der Taschenrechner aufladbar ist.

5.2 Funktionalität

Beim Aufstarten des Taschenrechners wird man von diesem Logo gegrüsst.

Abbildung 5.2: Das Aufstart-Logo des Taschenrechners.

Danach sieht man den Hauptbildschirm, wo Rechnungen eingegeben werden können. Die vier Tabs unten geben Auskunft, welches Menu die vier Funktionstasten f1 bis f4 öffnen.

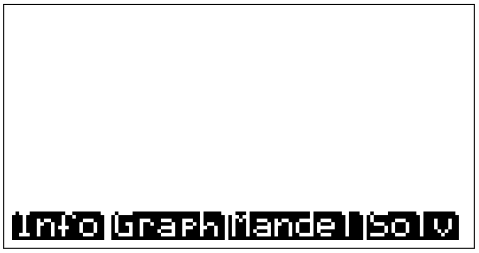


Abbildung 5.3: Der Hauptbildschirm des Taschenrechners.

Mit den Zahlen- und Operationstasten kann man eine Rechnung eingeben und mit der Enter-Taste berechnen lassen. Zum Zeitpunkt der Abgabe der Arbeit unterstützt der Taschenrechner die folgenden Operationen:

- Grundoperationen Plus, Minus, Mal, Durch
- Klammern
- Potenzen

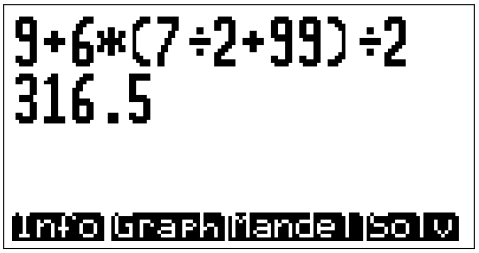


Abbildung 5.4: Eine Beispielsrechnung.

Drückt man die f1-Taste, führt dies zum Info-Bildschirm, der Versionsnummer und Akkustand anzeigt.

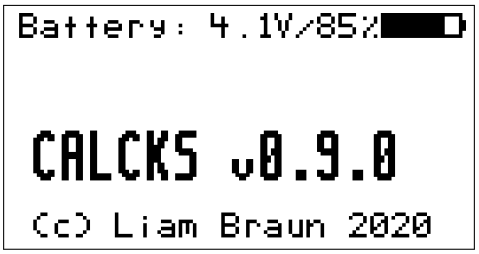


Abbildung 5.5: Der Informationsbildschirm des Taschenrechners.

Um Graphen zu zeichnen, gibt man im Hauptbildschirm einen Term ein, der x enthält und drückt die f2-Taste.

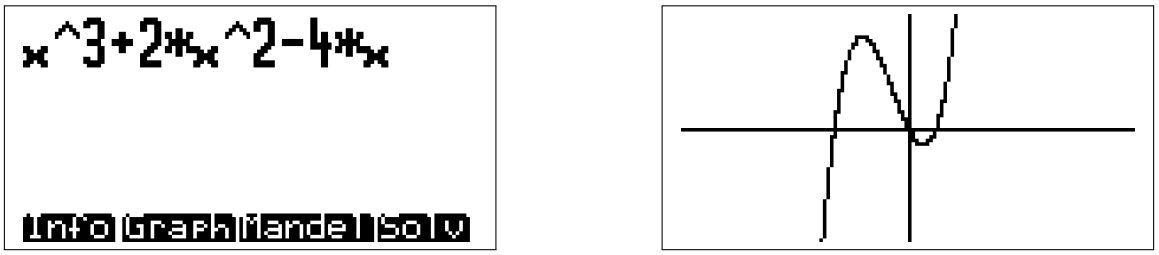


Abbildung 5.6: Graphendarstellung der Gleichung $f(x) = x^3 + 2x^2 - 4x$.

Die f3-Taste zeichnet auf dem Bildschirm das Mandelbrot-Fraktal. Diese Funktion habe ich ursprünglich geschrieben, um das Zeichnen von einzelnen Pixeln zu testen und auch die Geschwindigkeit des Rechners. Auf der Hardware, auf dem ATmega1284 mit einer Taktfrequenz von 8MHz dauert das Zeichnen des Fraktals etwa 17 Sekunden. Ich habe auch versucht, zuerst alle Pixel zu berechnen und alles gleichzeitig anzuzeigen, aber dies beschleunigte die Funktion nur um eine Sekunde, weshalb ich schlussendlich wieder auf die alte Version zurückgriff und die Pixel laufend anzeigte.

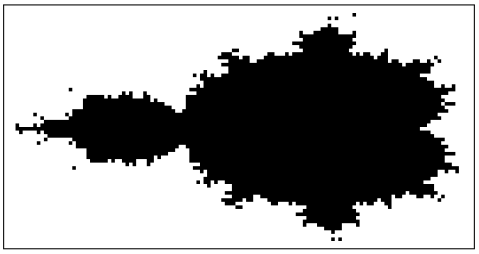


Abbildung 5.7: Das Mandelbrot-Fraktal, berechnet durch den Taschenrechner.

Der Taschenrechner enthält auch einen Gleichungslöser, den man mit f4 aktiviert. Momentan kann man nur lineare Gleichungen lösen, für weitere Gleichungsarten reichte die Zeit nicht. Man kann die selben Operationen wie im Hauptbildschirm verwenden, um die Werte für m und q einzugeben, mit der enter-Taste geht man zum nächsten Eingabefeld.

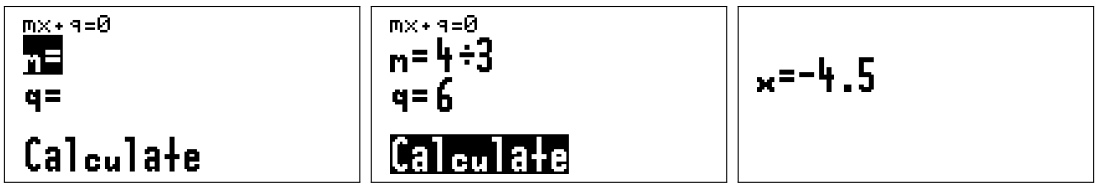


Abbildung 5.8: Der Taschenrechner löst die Gleichung $\frac{4}{3}x + 6 = 0$.

Kapitel 6

Schlusswort

Vergleicht man das Endprodukt mit den ursprünglichen Zielen der Arbeit, findet man eine grosse Übereinstimmung. Eine erste Anwendung fand ich bei einer Gletscherexkursion, wo meine Gruppe eine Höhenlinie von zwei Querschnitten des Rhonegletschers machen mussten. Ich konnte mir einen Drucksensor ausleihen, den ich an die Erweiterungs- und Programmierpins anschloss. Mit der Gleichung $h = 44331 * (1 - \frac{p}{p_0})^{0.1903}$, wobei h die gesuchte Höhe, p der gemessene Luftdruck und p_0 der Luftdruck bei Meereshöhe ist, konnte laufend die Höhe über Meer bestimmt werden[1].

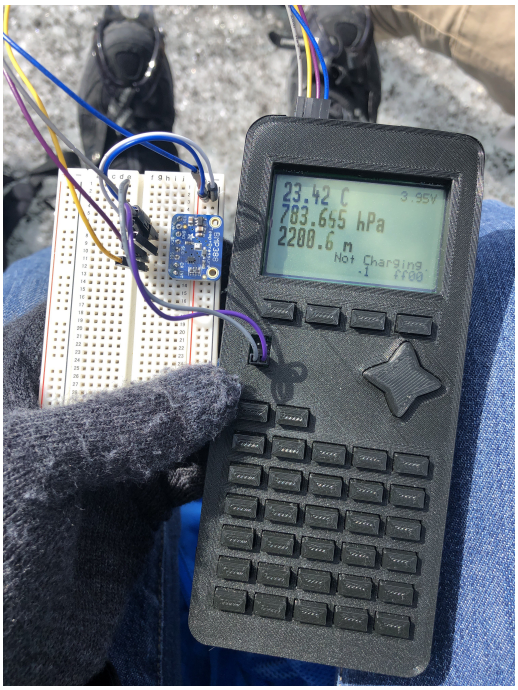


Abbildung 6.1: Der Taschenrechner dabei auf der Gletscherexkursion.

[1]Formel von `https://community.bosch-sensortec.com/t5/Question-and-answers/How-to-calculate-the-altitude-from-the-pressure-sensor-data/qaq-p/5702` (Abgerufen 16.10.2020)

Auch wenn ich gerne bei der Abgabe der Arbeit mehr Funktionalität implementiert gehabt hätte, bin ich sehr zufrieden, wie die Arbeit herausgekommen ist. Der Grund für die fehlenden Features waren nicht Hardware- oder Softwarelimitationen, sondern der Zeitdruck, der mich zwang, mich auf andere Aspekte der Arbeit zu konzentrieren. Persönlich überrascht hat mich, für welche Teile der Arbeit ich wie viel Zeit investieren musste. Ich hätte zum Beispiel nicht gedacht, dass das Anlöten der Komponenten so eine schwierige Arbeit sein werde. Hingegen war das Erfolgserlebnis, als das Anlöten ohne Brückenprobleme geklappt hat, umso grösser, weshalb ich vermutlich auch in Zukunft solche Projekt angehen werde. Ich konnte sehr viel Wertvolles über das Hardwaredesign, die Elektronik und der Informatik mitnehmen und glaube, dass ich auch nach der Abgabe noch weiterarbeiten werde, um ein nützliches Tool bei anderen Projekten zu haben. Die Maturarbeit hat mir auch insofern bei der Studienwahl geholfen, weil ich mich lange nicht zwischen den Studiengängen Informatik und Elektrotechnik & Informationstechnologie an der ETH Zürich entscheiden konnte. Ich habe während der Arbeit bemerkt, dass mir das Programmieren und die Algorithmensuche extrem viel Spass machte, weshalb ich vorhabe, den theoretischeren der beiden Studiengänge, Informatik, zu wählen.

Kapitel 7

Anhang

7.1 Danksagung

Vielen Dank an meinen Betreuer Claudio Müller für die diversen Ideen für Funktionen des Taschenrechners und wertvollen Hinweise beim Schreiben des schriftlichen Teils. Vielen Dank auch an Matthias Braun für das Korrekturlesen und die konstruktive Kritik an der Arbeit.

7.2 Benützte Software

Programmierumgebung: Visual Studio Code, Version 1.49.3
PCB Design: KiCad, Version 5.1.5
CAD: Autodesk Fusion 360 mit Studentenlizenz, Version 2.0.9009
C-Compiler für ATmega: WinAVR, Version 20090313
AVR Uploader: avrdude, Version 5.10
Design der Schriftart und Logo: Aseprite, Version 1.2.25-x64
Illustrationen für den schriftlichen Teil: Inkscape, Version 1.0
Darstellung der Bill of Materials: Microsoft Excel für Microsoft 365
Schriftlicher Teil: Geschrieben und kompiliert mit Texmaker, Version 5.0.4

7.3 Anmerkung zu Abbildungen

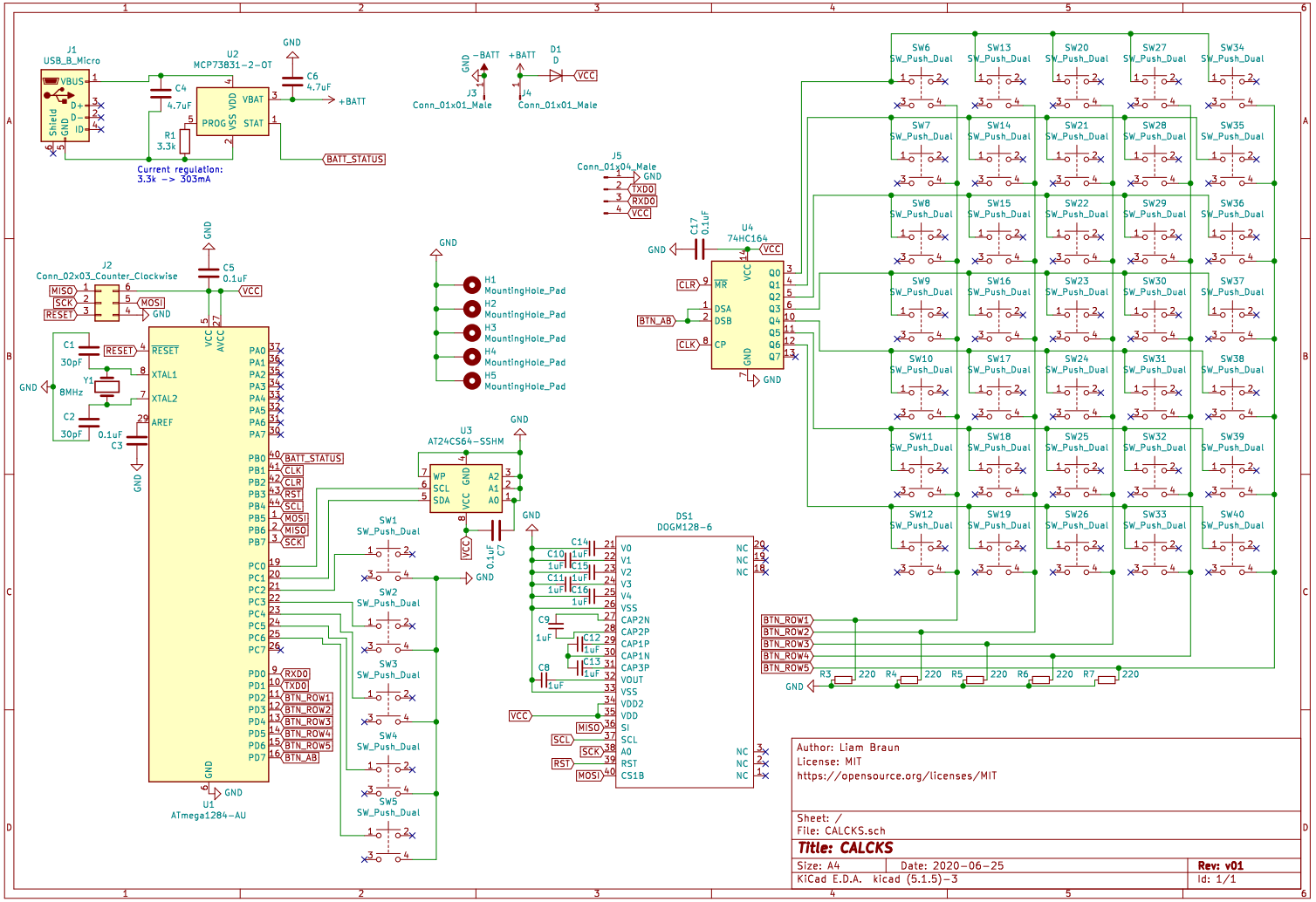
Abbildung 1.1 ist von Matthias Braun fotografiert.
Abbildung 3.2 stammt von DigiKey Electronics und ist unter `https://www.digikey.ch/product-detail/de/e-switch/TL3301SPF260QG/EG1870CT-ND/271567` (Abgerufen 16.10.2020) zu finden.
Die gerenderten PCB-Layouts im Anhang wurden mit `https://www.gerblook.org/` (Aufgerufen 14.10.2020) von den selbsterstellten Projektdateien generiert.
Alle anderen in dieser Arbeit verwendeten Abbildungen sind von dem Autor mit den erwähnten Programmen selbst erstellt worden.

7.4 Stückliste

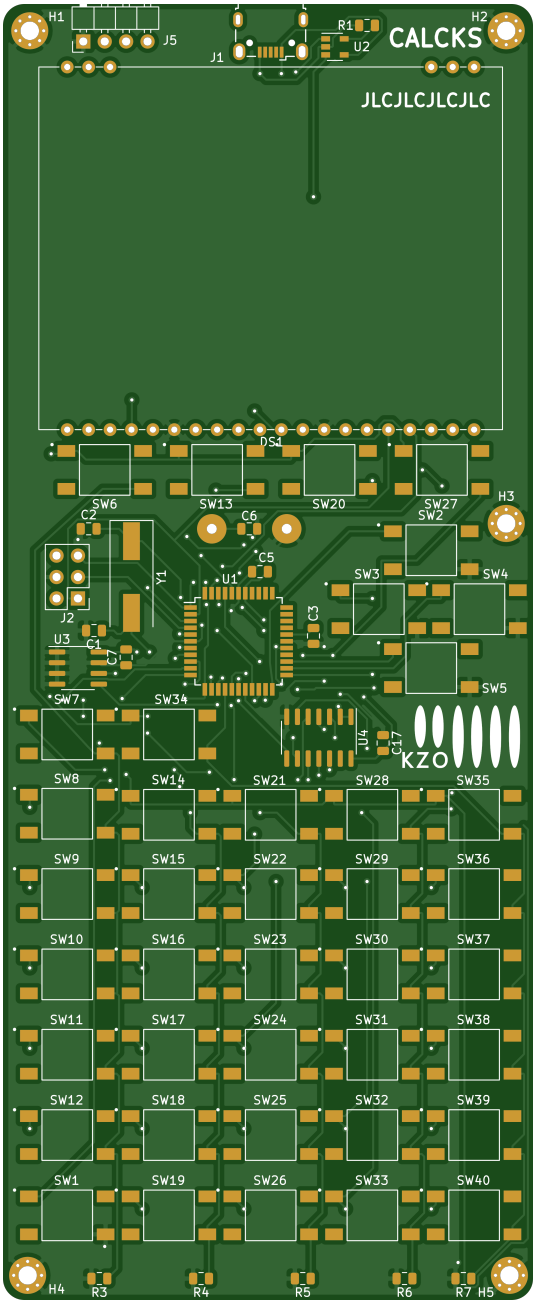
Diese Liste wurde automatisch von KiCad generiert und besteht aus allen Komponenten, die im Schema aufgelistet sind. Dies beinhaltet den Akku nicht, weil im Schema für diesen nur die dazugehörigen Lötpads eingetragen sind.

Reference	Quantity	Value	Footprint
C14 C10 C15 C11 C16 C9 C12 C13 C8	9	1uF	Capacitor_SMD:C_0805_2012Metric
C2 C1	2	30pF	Capacitor_SMD:C_0805_2012Metric
C5 C7 C17 C3	4	0.1uF	Capacitor_SMD:C_0805_2012Metric
C6 C4	2	4.7uF	Capacitor_SMD:C_0805_2012Metric
D1	1	D	Diode_SMD:D_SOD-323_HandSoldering
DS1	1	DOGM128-6	CALCKS:LCD_DOGM128-6
H1 H2 H3 H4 H5	5	MountingHole_Pad	MountingHole:MountingHole_2.2mm_M2_Pad_Via
J1	1	USB_B_Micro	Connector_USB:USB_Micro-B_Wuerth_629105150521
J2	1	Conn_02x03_Counter_Clockwise	Connector_PinHeader_2.54mm:PinHeader_2x03_P2.54mm_Vertical
J3 J4	2	Conn_01x01_Male	Connector_Wire:SolderWirePad_1x01_Drill1.2mm
J5	1	Conn_01x04_Male	Connector_PinHeader_2.54mm:PinHeader_1x04_P2.54mm_Horizontal
R1	1	3.3k	Resistor_SMD:R_0805_2012Metric
R6 R5 R4 R7 R3	5	220	Resistor_SMD:R_0805_2012Metric
SW6 SW13 SW20 SW27 SW34 SW7 SW14 SW21 SW28 SW35 SW8 SW15 SW22 SW29 SW36 SW9 SW16 SW23 SW30 SW37 SW10 SW17 SW24 SW31 SW38 SW11 SW18 SW25 SW32 SW39 SW12 SW19 SW26 SW40 SW1 SW2 SW3 SW4 SW5 SW33	40	SW_Push_Dual	Downloaded Footprints:SW_TL3301SPF260QG
U1	1	ATmega1284-AU	Package_QFP:TQFP-44_10x10mm_P0.8mm
U2	1	MCP73831-2-OT	Package_TO_SOT_SMD:SOT-23-5
U3	1	AT24CS64-SSHM	Package_SO:SOIC-8_3.9x4.9mm_P1.27mm
U4	1	74HC164	Package_SO:SOIC-14_3.9x8.7mm_P1.27mm
Y1	1	8MHz	Crystal:Crystal_SMD_HC49-SD

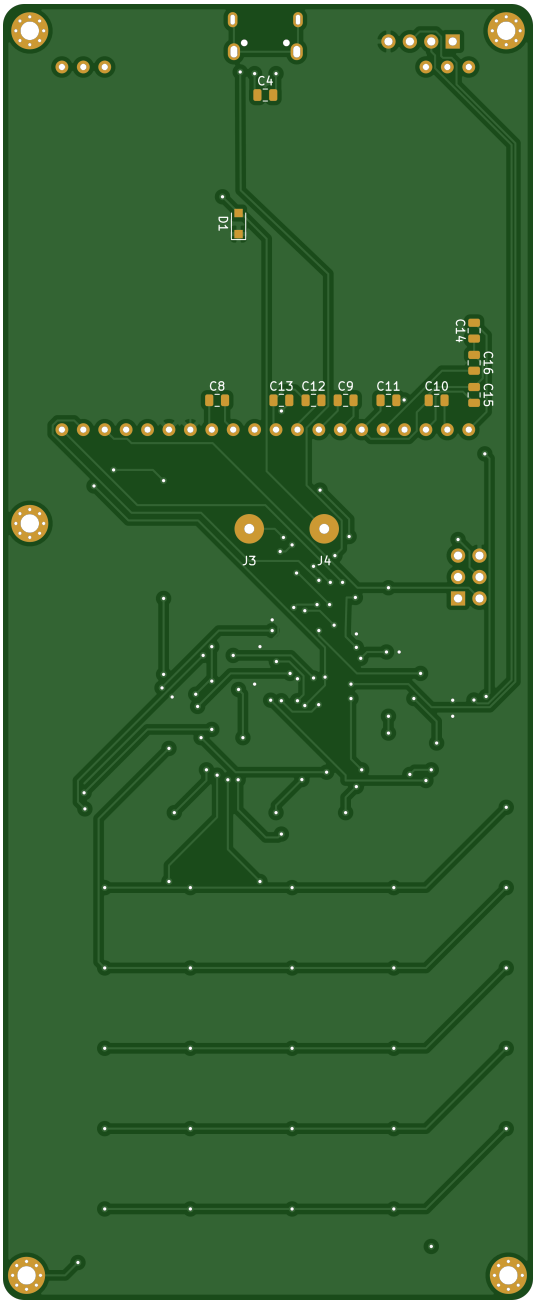
7.5 Schema


J1
USB_B_Micro
U2
MCP73831-2-OT
C4
4.7uF
C6
4.7uF
R1
3.3k
Current regulation:
3.3k -> 303mA
BATT_STATUS
+BATT
J3
Conn_01x01_Male
J4
Conn_01x01_Male
D1
D
J5
Conn_01x04_Male
TXD0
RXD0
J2
Conn_02x03_Counter_Clockwise
C5
0.1uF
C1
30pF
C2
30pF
Y1
8MHz
C3
0.1uF
U1
ATmega1284-AU
H1
MountingHole_Pad
H2
MountingHole_Pad
H3
MountingHole_Pad
H4
MountingHole_Pad
H5
MountingHole_Pad
U3
AT24CS64-SSHM
C7
0.1uF
U4
74HC164
C17
0.1uF
DS1
DOGM128-6
SW1
SW_Push_Dual
SW2
SW_Push_Dual
SW3
SW_Push_Dual
SW4
SW_Push_Dual
SW5
SW_Push_Dual
SW6
SW_Push_Dual
SW7
SW_Push_Dual
SW8
SW_Push_Dual
SW9
SW_Push_Dual
SW10
SW_Push_Dual
SW11
SW_Push_Dual
SW12
SW_Push_Dual
SW13
SW_Push_Dual
SW14
SW_Push_Dual
SW15
SW_Push_Dual
SW16
SW_Push_Dual
SW17
SW_Push_Dual
SW18
SW_Push_Dual
SW19
SW_Push_Dual
SW20
SW_Push_Dual
SW21
SW_Push_Dual
SW22
SW_Push_Dual
SW23
SW_Push_Dual
SW24
SW_Push_Dual
SW25
SW_Push_Dual
SW26
SW_Push_Dual
SW27
SW_Push_Dual
SW28
SW_Push_Dual
SW29
SW_Push_Dual
SW30
SW_Push_Dual
SW31
SW_Push_Dual
SW32
SW_Push_Dual
SW33
SW_Push_Dual
SW34
SW_Push_Dual
SW35
SW_Push_Dual
SW36
SW_Push_Dual
SW37
SW_Push_Dual
SW38
SW_Push_Dual
SW39
SW_Push_Dual
SW40
SW_Push_Dual
BTN_ROW1
BTN_ROW2
BTN_ROW3
BTN_ROW4
BTN_ROW5
R3 220
R4 220
R5 220
R6 220
R7 220
Author: Liam Braun
License: MIT
https://opensource.org/licenses/MIT
Sheet: /
File: CALCKS.sch
Title: CALCKS
Size: A4
Date: 2020-06-25
Rev: v01
KiCad E.D.A. kicad (5.1.5)-3
Id: 1/1

7.6 PCB Render



(a) Vorderseite



(b) Rückseite

Literaturverzeichnis

[1] Electronic Assembly. *Datasheet DOGM128-6*, 2009. Verfügbar unter: https://www.lcd-module.de/deu/pdf/grafik/dogm128.pdf (Abgerufen 09.09.2020).

[2] RS Components GmbH. Lötpasten. Verfügbar unter: https://de.rs-online.com/web/c/elektrowerkzeuge-loten-schweissen/loten/lotpasten/ (Abgerufen 14.10.2020).

[3] Ali Hafizji. Shunting yard algorithm. Verfügbar unter: https://aquarchitect.github.io/swift-algorithm-club/Shunting%20Yard/ (Abgerufen 13.10.2020).

[4] AspenCore Inc. The shift register. Verfügbar unter: https://www.electronics-tutorials.ws/sequential/seq_5.html (Abgerufen 15.10.2020).

[5] Renata SA. *Datasheet ICP582930PR-01*, August 2019. Verfügbar unter: https://www.renata.com/fileadmin/downloads/productsheets/lithium_polymer/ICP582930PR-01.pdf (Abgerufen 14.10.2020).

[6] Sitronix. *65 x 132 Dot Matrix LCD Controller/Driver*, 2006. Verfügbar unter: https://www.lcd-module.de/eng/pdf/zubehoer/st7565r.pdf (Abgerufen 09.09.2020).

[7] Microchip Technology. *MCP73831/2*, Juni 2020. Verfügbar unter: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP73831-Family-Data-Sheet-DS20001984H.pdf (Abgerufen 16.10.2020).

[8] Microchip Technology. *megaAVR® Data Sheet*, 2020. Verfügbar unter: http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega164A_PA-324A_PA-644A_PA-1284_P_Data-Sheet-40002070B.pdf (Abgerufen 29.09.2020).