Mikrokontroller-Anwendungen

Projektarbeit

Prototyp einer Akku-Aufbewahrungsstation auf Basis der Arduino-Nano-Entwicklungsplatine

Martin Morcinietz und Andrej Lisnitzki

Hochschule für Technik Wirtschaft und Kultur Leipzig
Fakultät Informatik, Mathematik und Naturwissenschaften

Professor: Herr Prof. Dr. rer. nat. Klaus Bastian

Leipzig, 24. März 2016

Inhaltsverzeichnis

1 Einleitung....4
2 Vorüberlegungen für eine Lithium-Ionen-Akku Aufbewahrungsstation....5
2.1 Hardware....5
2.2 Software....6
3 Der Lithium-Ionen-Akku....7
3.1 Aufbau eines Lithium-Ionen-Akku....7
3.2 Kennwerte eines Lithium-Ionen-Akku....7
3.3 Laden eines Lithium-Ionen-Akku....8
4 Die verwendete Hardware....11
4.1 Die Entwicklungsplatine....11
4.2 LCD-Anzeige....12
4.3 Temperatursensor....12
4.4 RTC (Real Time Counter)....13
5 Die verwendeten Protokolle....14
5.1 I²C-Bus....14
5.1.1 Aufbau eines I²C-Bus-Systems....14
5.1.2 Die Kommunikation auf dem I²C-Bus....15
5.1.3 Adressierung....16
5.1.4 Nutzdatenübertragung....17
5.2 One-Wire-Bus....17
5.2.1 Aufbau des One-Wire-Bus-Systems....18
5.2.2 Kommunikation auf dem One-Wire-Bus....19
5.2.3 Die Kommunikation im Detail....19
6 Umsetzung des Projekts....21
6.1 Verwendete Entwicklungswerkzeuge....21
6.1.1 Einstellen der Arduino-IDE....22
6.2 Beschreibung des Programms....22
6.2.1 Verwendete Bibliotheken....22
6.2.2 Programmablauf....25
6.3 Beschreibung der Schaltung....27
6.3.1 Anschluss der Peripherie-Module an das Arduino Nano....27
6.3.2 Ladeschaltung....28
6.3.2.1 Spannungsregler....28
6.3.2.2 Stromregler....29
6.3.2.3 Zusammengesetzte Schaltung....30
7 Zusammenfassung....32
Literaturverzeichnis....33
Anhang....34

Abbildungsverzeichnis

Abbildung 1: Ladecharakteristik eines Kobalt LiIon-Akkus2.....9
Abbildung 2: Block-Diagramm ATmega 328P.....11
Abbildung 3: Schaltplan des PCF8574 mit LCD-Anzeige.....12
Abbildung 4: Aufbau eines I²C-Bus-Systems.....14
Abbildung 5: Start-/Stopp-Bedingungen beim I²C-Bus.....15
Abbildung 6: Datenübertragung auf dem I²C-Bus.....17
Abbildung 7: Aufbau des One-Wire-Bus-Systems.....18
Abbildung 8: Kommunikation über den One-Wire-Bus.....19
Abbildung 9: Anschluss der Peripherie-Module an die Arduino Nano-Entwicklungsplatine.....27
Abbildung 10: Spannungsregler.....28
Abbildung 11: Stromregler.....29
Abbildung 12: Komplette Ladeschaltung.....30

1 Einleitung

Im Rahmen der Lehrveranstaltung Mikrocontroller-Anwendungen ist eine Projektarbeit als Prüfungsleistung zu erstellen, bei welcher komplexe Aufgabenstellungen unter Zuhilfenahme von Mikrocontrollern gelöst werden sollen. Ziel dieser Projektarbeit ist die Entwicklung und die Umsetzung einer Aufbewahrungsstation für Lithium-Ionen-Akkus.
Diese Arbeit entstand in Zusammenarbeit von Andrej Lisnitzki und Martin Morcinietz und umfasst folgende Kapitel:

- Kapitel 2 Hier werden die Vorüberlegungen näher erläutert, welche vor der Umsetzung des Projekts zu dessen Umsetzung angestellt wurden.
- Kapitel 3 In diesem Kapitel wird der Lithium-Ionen-Akku näher beschrieben. Die Erläuterungen sind sehr ausführlich, da es sich bei diesem Projekt um eine Aufbewahrungsstation für Lithium-Ionen-Akkus handelt und hier deren Besonderheiten näher erläutert werden.
- Kapitel 4 Dieses Kapitel stellt die zur Umsetzung des Projekts verwendete Hardware vor, wobei diese kurz beschrieben werden.
- Kapitel 5 In diesem Kapitel werden die zur Umsetzung des Projekts, also für die Kommunikation der verschiedenen Hardware-Teile untereinander verwendeten Kommunikationsbusse beschrieben, wobei deren Besonderheiten näher beleuchtet werden.
- Kapitel 6 Dieses Kapitel bildet den Hauptteil dieser Arbeit. Hier wird zum einen die Umsetzung und die Verwendung der Hardware beschrieben, sowie zum Anderen die Entwicklung der zugehörigen Software beleuchtet.
- Kapitel 7 Hier geschieht eine abschließende Zusammenfassung der Arbeit, welche Voraussetzungen für die Entwicklung nötig sind, welche Funktionalitäten in diesem Projekt umgesetzt wurden und wie diese zukünftig noch erweitert werden können.

2 Vorüberlegungen für eine Lithium-Ionen-Akku Aufbewahrungsstation

Dieses Kapitel widmet sich den Vorüberlegungen für die Umsetzung einer Aufbewahrungsstation für Lithium-Ionen-Akkus.

Im Allgemeinen soll die hier entworfene Aufbewahrungsstation eine Kombination aus einem Ladegerät und einer zyklisch die Spannung des Akkus prüfenden Aufbewahrungsstation bilden. Diese Aufbewahrungsstation soll dazu dienen, eine Tiefentladung des gelagerten Akkus zu verhindern und ihn zu gegebener Zeit wider auf seine volle Kapazität aufladen.

2.1 Hardware

Für die Funktionalität des Ladens ist eine elektronische Schaltung zu entwickeln, welche für dass Laden eines Lithium-Ionen-Akkus geeignet ist und den Akku mit 1C lädt. Durch die besonderen Anforderungen die Lithium-Ionen-Akkus stellen (siehe Kapitel 3), sollen die Akkus während der Aufbewahrung nur zu 70% ihrer Kapazität geladen werden und sich durch den Effekt der Selbstentladung bis maximal 50% ihrer Kapazität entladen. Die Kapazität des Akkus ist daher zyklisch zu prüfen und ggf. ist der Akku bis zu der für die Aufbewahrung bestimmten Kapazität wider aufzuladen. Zu einem bestimmten Datum, welches vorher eingestellt werden können soll, soll der Akku dann auf eine ebenfalls vorher eingestellte Kapazität geladen werden, um ihn dann verwenden zu können. Diese Funktionalitäten setzen die Verwendung eines Batteriegepufferten Echtzeit-Zählers voraus. Weiterhin ist es durchaus sinnvoll die Temperatur des sich im Laden befindlichen Akkus zu überwachen, für welches ein Temperatursensor benötigt wird. Für die Visualisierung der verschiedenen Einstellungen, der Darstellungen von Ausgaben, sowie der Anzeige der aktuellen Kapazität des Akkus soll eine LCD-Anzeige verwendet werden. Ebenfalls ist für die Umsetzung dieser Funktionalitäten ein geeigneter Mikrocontroller zu wählen. Dieser sollte ausreichend für die beschriebenen Anforderungen dimensioniert sein, zum Beispiel genügend digitale Anschlüsse für die Benutzereingaben mit Tastern und für die Ansteuerungen besitzen. Er sollte aber nicht zu überdimensioniert sein, da dies die Kosten des Projekts und die Leistung des Systems unnötig erhöhen würde.

2.2 Software

Die Software dieser Aufbewahrungsstation sollte in der Lage sein die unter 2.1 beschriebenen Hardware korrekt zu verwalten und die bereits erwähnten Abläufe zu steuern. Dazu bietet sich eine Umsetzung der Software als endlicher Zustandsautomat an. Dieser Automat sollte genügend Zustände für das Aufladen des Akkus, die Aufbewahrung des Akkus, die Einstellungen des Benutzers und evtl. für den Umgang mit zum Beispiel einer zu hohen Temperatur des Akkus besitzen. Weiterhin sind für die Software geeignete Datenstrukturen zu entwickeln, welche es ermöglichen, eine Übersichtliche Software zu entwerfen. An die Verarbeitungsgeschwindigkeit der Software werden keine großen Anforderungen gestellt, da es sich hierbei nicht um eine Echtzeit-Anwendung handelt. Die Werte des Akkus sollen allerdings in einer akzeptablen Zeit geprüft werden können. Für die Verwendung von Tastern zur Benutzereingabe müssen diese auch auf geeignete Weise Softwaretechnisch entprellt werden, um Fehlfunktionen zu vermeiden. Für die Visualisierung auf einer LCD-Anzeige ist hierfür ein geeignetes Kommunikationsprotokoll zu entwickeln, durch welches eine vernünftige Darstellung auf der Anzeige möglich ist. Es ist allerdings darauf zu achten, dass die Software dennoch nicht zu umfangreich wird, da die Speicherkapazität des internen Programmspeichers des Mikrocontrollers meist begrenzt ist.

3 Der Lithium-Ionen-Akku[1]

Lithium-Ionen-Akkus (LiIon-Akkus, oder Li-Ionen-Akkus) zeichnen sich durch ihre hohe Leistungsdichte aus, die mit mehreren 1.000 W/kg höher ist als bei allen anderen Akkumulatoren. Darüber hinaus ist der LiIon-Akku thermisch stabil, hat eine konstante Ausgangsspannung über den gesamten Entladezeitraum, eine lange Lebensdauer und kennt keinen Memory-Effekt. Lithium hat in der elektrochemischen Spannungsreihe ein hohes negatives Potential von -3,05V, wodurch die Potentialdifferenz mit anderen Materialien sehr hoch ist. In Verbindung mit Kupfer (0,16V) ergibt sich eine Spannungsdifferenz von 3,21V. Die Nennspannung von Lithium-Ionen-Akkus liegt um die 3V, die elektromotorische Kraft beträgt 3,6V, die Ladeschlussspannung liegt bei 4,2V, die Entladeschlussspannung bei etwa 2,5V und der C-Koeffizient für den Ladestrom beträgt 1C.

3.1 Aufbau eines Lithium-Ionen-Akku

Beim Lithium-Ionen-Akku besteht die Anode aus einer Kupferfolie, die mit Kohle oder einer Graphitverbindung beschichtet ist. Die positive Kathode besteht aus einer Lithiumverbindung, bestehend aus Kobalt-, Mangan- oder Nickel-Oxyd. Der zwischen den Elektroden liegende Elektrolyt ist ein gelöstes Lithiumsalz. Je nachdem, ob der Elektrolyt flüssig oder fest ist, spricht man von Lithium-Ionen-Akkus oder Lithium-Polymer-Akkus. Die verschiedenen Lithium-Ionen-Akkus unterscheiden sich hauptsächlich im Kathodenwerkstoff, der Kobalt, Mangan, Nickel-Kobalt, Nickel-Kobalt-Mangan (NKM), Eisenphosphat oder Titanat sein kann. Die verschiedenen Kathodenwerkstoffe bewirken unterschiedliche Energiedichten, Leistungsdichten, Nennspannungen und Ladezyklen. Bei Kobalt reichen die Werte für die Energiedichte bis zu 190 Wh/kg, bei Mangan bis 120 Wh/kg und bei Nickel-Kobalt-Mangan (NKM) bis 130 Wh/kg. Die Energiedichte liegt bei Hochenergie-Versionen wie dem Lithium-Schwefel-Akku bei bis zu 400 Wh/kg oder dem Lithium-Luft-Akku bei über 1.000 Wh/kg und bei Hochleistungsversionen zwischen 2 kW/kg und 4 kW/kg. Die mittlere Anzahl der Entlade- und Ladezyklen liegt bei etwa 1.500.

3.2 Kennwerte eines Lithium-Ionen-Akku

Ein konventioneller LiIon-Akku liefert eine Nennspannung von 3,7 Volt, die damit rund dreimal so hoch wie die eines NiMH-Akkus ist. Die Energiedichte ist mit ca. 100 Wh/kg etwas geringer als die von Alkali-Mangan-Batterien, aber deutlich größer als die konventioneller Akkus. Achtet man

1 http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries

auf eine Entladespannung von minimal 3,5V, um die Lebensdauer zu erhöhen, reduziert sich die Energiedichte auf ca. 60–70 Wh/kg. Heutige LiIon Akkus für Kraftfahrzeuge erreichen allerdings schon eine Energiedichte von über 120 Wh/kg.

Die Kapazität eines Lithium-Ionen-Akkus verringert sich selbst ohne Benutzung mit der Zeit, hauptsächlich durch parasitäre Reaktion des Lithiums mit den Elektrolyten. Die Zersetzungsgeschwindigkeit steigt mit der Zellspannung und der Temperatur. Eine Tiefentladung unterhalb 2,4V kann den Akku dauerhaft schädigen. Hersteller empfehlen eine Lagerung bei 15℃ und einem Ladestand von 60%, ein Kompromiss zwischen beschleunigter Alterung und Selbstentladung. Ein Akku sollte etwa alle sechs Monate auf 40 bis 60% nachgeladen werden. Zur Zeit gilt die Faustregel, dass ein LiIon-Akku nach ca. drei Jahren mehr als 50% seiner Kapazität eingebüßt hat. Generell sollte das Entladen unter 40% vermieden werden, da es bei „tiefen Zyklen“ zu größeren Kapazitätsverlusten aufgrund irreversibler Reaktionen in den Elektroden kommen kann. Grundsätzlich ist es besser, LiIon-Akkus „flach“ zu zyklen, wodurch sich deren Lebensdauer verlängert, d.h. dass ein LiIon-Akku bis 40% seiner Kapazität entladen und bis maximal 60% seiner Kapazität geladen werden sollte. In diesem Projekt wurde sich für die Aufbewahrung auf einen Wert von 50% (3,7V) als untere Kapazität und 60% (3,8V) für die obere Kapazität festgelegt.

3.3 Laden eines Lithium-Ionen-Akku

Geladen werden LiIon-Akkus ab der Tiefentladeschwelle nach dem IU-Ladeverfahren mit konstantem Strom. Die Ladung mit einem konstanten Strom geschieht dann bis zum Erreichen der Nennspannung. Danach werden die LiIon-Akkus mit einer konstanten Spannung weiter geladen um die Zellen des Akkus zu sättigen, d.h. ihre maximale Kapazität wieder herzustellen. Liegt der Ladezustand unterhalb der Tiefentladeschwelle, dann wird der tiefentladene Akku vorbereitet, indem er mit geringer Stromstärke bis zum Erreichen der Mindestspannung geladen wird.

Bei einer Tiefentladung oder Überladung schaltet im günstigsten Fall eine interne Sicherung den Akku, meist nur temporär, ab. Im Falle einer Tiefentladung liegt dann an den externen Kontakten des Akkupacks überhaupt keine Spannung mehr an, d.h., er kann nicht noch weiter entladen werden. Leider weigern sich etliche Geräte einen derartig defekt anmutenden Akku wieder zu laden, da in diesem Fall an den externen Kontakten nur eine Spannung von 0V messbar ist, obwohl der Akku von seiner Schutz-Elektronik wieder an die Kontakte geschaltet werden würde, sobald ein Ladestrom anliegt. Ein anderes Ladegerät zu probieren kann in solchen Fällen weiterhelfen.

Im Falle einer Überladung wird der Akku von den externen Kontakten getrennt, bis keine zu hohe Lade-Spannung mehr anliegt. Danach kann er meist ohne Probleme wieder verwendet werden.

Die Ladecharakteristik eines LiIon-Akkus ist i.d.R nichtlinear. Wie bereits erwähnt werden LiIon-Akkus mit dem IU-Ladeverfahren geladen, folgende Abbildung 1 veranschaulicht den Ladevorgang eines Kobalt LiIon-Akkus, welcher sehr ähnliche Spannungscharakteristiken zu dem in diesem Projekt verwendeten Lithium-Polymer-Akku besitzt. Auf der rechten Seite der Abbildung 1 ist die Spannung in Volt und auf der linken Seite der Ladestrom in Ampere zu sehen. Am oberen und unteren Bildrand befinden sich die Ladephasen, hier als Stage 1 bis 4 bezeichnet, mit einer kurzen Beschreibung der jeweiligen Phase. Des Weiteren ist an der x-Achse die Ladezeit in Stunden aufgetragen. Das Laden eines LiIon-Akku erfolgt in der ersten Phase mit einem konstanten Ladestrom und steigender Spannung, in der zweiten Phase bleibt dann sie Spannung konstant und der Ladestrom wird langsam verringert, welches als Sättigung bezeichnet wird. In der dritten Phase (Stage 3) wird das Laden des Akku beendet, dies geschieht durch eine gering Reduktion der angelegten Spannung und dem anschießenden Abschalten des Ladegeräts. Eine optionale Ladephase ist die sog. Endladephase (Stage 4), diese muss nur gelegentlich durchgeführt werden. Der Übergang zwischen den verschiedenen Ladephasen ist durch die senkrechten, gestrichelten Linien gekennzeichnet.

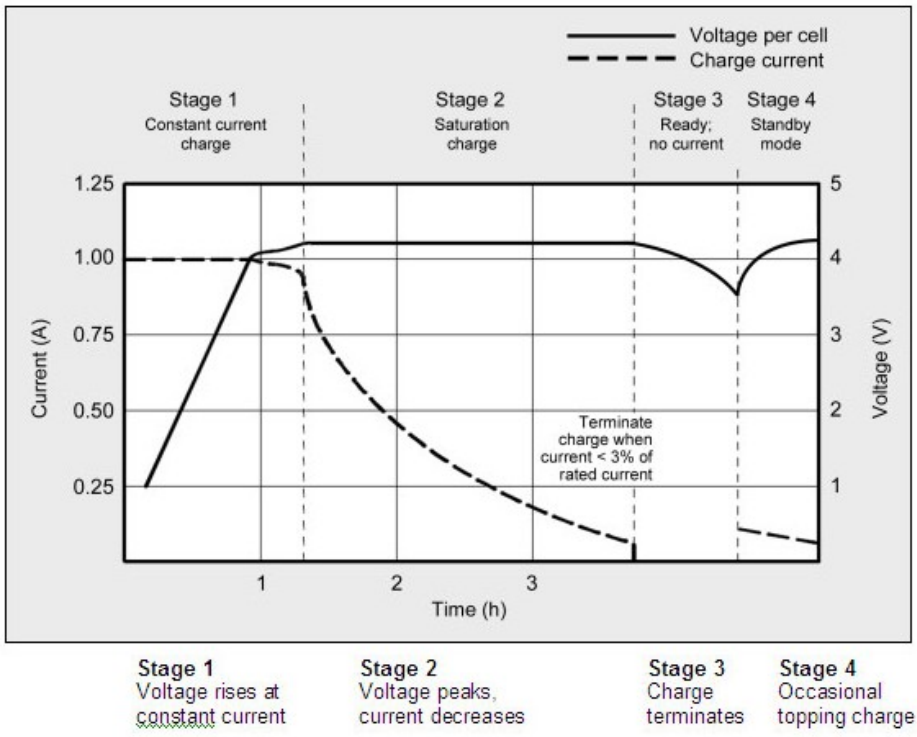

Abbildung 1: Ladecharakteristik eines Kobalt LiIon-Akkus[2]

Für dieses Projekt wurde sich auf einen Ladestrom von 100mA für die erste Phase und 20mA für die zweite Phase festgelegt.

Folgende Tabelle 1 zeigt die geschätzten Kapazitäten eines LiIon-Akkus in Prozent mit und ohne Sättigung bei unterschiedlichen Spannungsschwellen und die dazugehörige geschätzte Ladezeit.

Ladung in V/Zelle	**Kapazität ohne Sättigung**	**Ladezeit in Minuten**	**Kapazität mit voller Sättigung**
3,80	60%	120 min.	~65%
3,90	70%	135 min.	~75%
4,00	75%	150 min.	~80%
4,10	80%	165 min.	~90%
4,20	85%	180 min.	~100%

Tabelle 1: Ladecharakteristik eines LiIon-Akkus[2]

An den Kapazitätswerten in dieser Tabelle 1 kann der Einfluss des Sättigungshase auf die Kapazität des geladenen Akkus abgelesen werden, sie ist nach der Sättigung zwischen 5 bis 15% größer. Für die prozentualen Kapazitätsangaben bei diesem Projekt wurde sich an den Werten der letzten Spalte in dieser Tabelle 1 orientiert.

2 http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries

4 Die verwendete Hardware

In diesem Kapitel wird die für die Umsetzung dieses Projekts verwendete Hardware näher erläutert. Dies ist erstens die verwendete Entwicklungsplatine mit dem zugehörigen Mikrocontroller, welcher unter 4.1 näher erläutert wird. Zweitens wird unter 4.2 die zur Visualisierung verwendete LCD-Anzeige und der zugehörige I²C-parallel Umsetzer beschrieben. Weiterhin wird der hier verwendete digitale Temperatursensor unter Punkt 4.3 näher erläutert. Schließlich wird unter Punkt 4.4 der zur Umsetzung des Projekts benötigte RTC beschrieben.

4.1 Die Entwicklungsplatine

Für dieses Projekt wurde die Arduino-Nano-Entwicklungsplatine verwendet, auf welchem sich der 8-Bit Mikrocontroller Atmega 328P des Herstellers Atmel befindet. Das Arduino-Nano-Board besitzt 14 Digitale Ein-/Ausgangspins, welche mit bis zu 40 mA belastet werden können. Des Weiteren besitzt das Arduino-Nano-Board acht analoge Eingangspins. Zu den Spezifikationen des Atmega 328P Mikrocontrollers gehören zum einen ein 32 KB Flash-Speicher, zum anderen 1 KB EEPROM und 2 KB SRAM. Des Weiteren verfügt der Mikrocontroller über einen 16 MHz Takt und eine RISC-Architektur. Folgende Abbildung 2 veranschaulicht das Block-Diagramm des ATmega 328P.

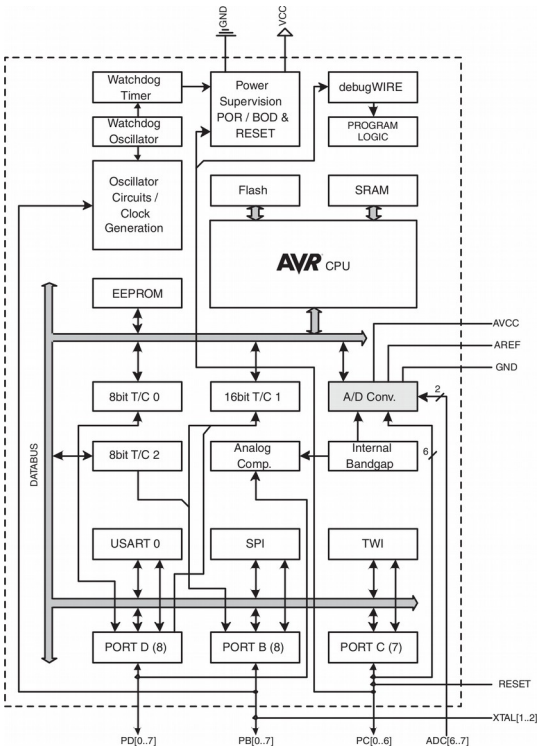

Abbildung 2: Block-Diagramm ATmega 328P

Das Arduino-Nano-Board wird mit 5V Spannung versorgt, welches entweder via USB oder durch eine externe Spannungsquelle erfolgen kann. Das in diesem Projekt verwendete Arduino-Nano-Board ist ein chinesischer Klon des originalen Arduino-Nano-Boards, der Unterschied zwischen diesen beiden Boards besteht in der Verwendung eines FTDI[3], welcher auf dem Original, aber nicht auf dem chinesischen Klon vorhanden ist. Als FTDI Ersatz wird für den Klon eine CH340G-Schaltung verwendet.

4.2 LCD-Anzeige

Zur Visualisierung des Auswahlmenüs und des aktuellen Ladezustandes wird eine Zwei-Zeilen LCD Anzeige verwendet. Diese Anzeige kann pro Zeile 16 Zeichen darstellen und besitzt eine LED Hintergrundbeleuchtung. Da die Anzeige parallel angesteuert wird, für eine parallele Ansteuerung allerdings nicht genügend freie Pins zur Verfügung stehen, wurde ein I²C-parallel Umsetzer PCF8574 verwendet. Folgende Abbildung 3 veranschaulicht den Anschluss der LCD-Anzeige an den PCF8574.

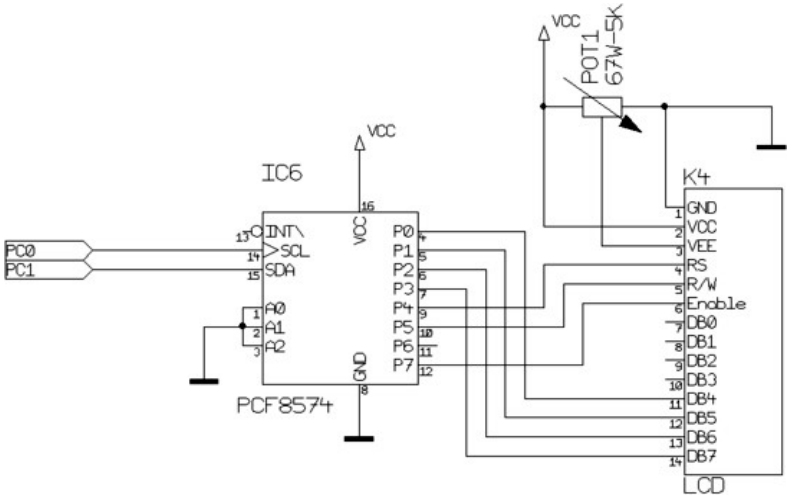

Abbildung 3: Schaltplan des PCF8574 mit LCD-Anzeige

4.3 Temperatursensor

Während des Ladevorgangs muss die Temperatur des Lithium-Ionen-Akku überwacht werden, um bei einer zu hohen Temperatur einen Schaden des Akku zu verhindern. Die Temperaturerfassung geschieht mittels eines digitalen Temperatursensors DS18B20. Dieser Temperatursensor ist über eine sog. One-Wire-Bus (siehe 5.2) mit dem Arduino-Nano-Board verbunden und ist in der Lage Temperaturen im Bereich von -55℃ bis +125℃ mit einer Genauigkeit von +/-0.5℃ zu erfassen.

3 USB-Seriell-Umsetzer

4.4 RTC (Real Time Counter)

Da es bei dieser Akku Aufbewahrungsstation möglich sein soll den Tag und die Uhrzeit anzugeben, zu welcher der Akku auf eine vorher eingestellte Kapazität aufgeladen werden soll und für die zyklische Prüfung der Akku Kapazität während der Lagerung, muss ein Batteriegepufferter Echtzeit-Zähler (engl. Real Time Counter, kurz RTC) verwendet werden. In diesem Projekt kommt für diese Aufgabe ein DS3231 zum Einsatz, welcher über den I²C-Bus mit dem Mikrocontroller kommuniziert.

5 Die verwendeten Protokolle

In diesem Kapitel werden die für dieses Projekt verwendeten Kommunikationsbusse näher erläutert, dies sind zum einen der unter 5.1 beschriebene I²C-Bus und zum anderen der unter 5.2 beschriebene One-Wire-Bus.

5.1 I²C-Bus[4]

Der I²C-Bus (Inter-Integrated Circuit) ist ein im Jahr 1982 von der Firma Philips entwickelter serieller Datenbus zur kostengünstigen Vernetzung von integrierten Schaltkreisen (ICs), Mikrocontrollern, sowie Ein- und Ausgabegeräten. Er wird hauptsächlich zur geräteinternen Kommunikation zwischen verschiedenen Schaltungsteilen benutzt, zum Beispiel zur Kommunikation zwischen Mikrocontrollern. Der I²C-Bus unterstützt verschiedene Übertragungsraten. Als Standard-Übertragungsrate wurden anfangs maximal 100 kBit/s definiert, diese Übertragungsrate wird heute als „Standard-Mode“ bezeichnet. Der I²C-Bus wurde im Laufe der Zeit mit steigenden Anforderungen um weitere Übertragungsraten-Modi erweitert. So gibt es heutzutage den „Fast-Mode“ mit bis zu 400 kBit/s, den „High-Speed-Mode“ mit bis zu 3,4 Mbit/s und seit der Version 4.0 den „Ultra-Fast-Mode“ mit einer maximalen Übertragungsrate von 5 Mbit/s. Die tatsächlich genutzte Übertragungsrate kann allerdings beliebige Werte zwischen 0 kBit/s und der jeweiligen maximalen Übertragungsrate annehmen.

5.1.1 Aufbau eines I²C-Bus-Systems

Dieser Punkt widmet sich dem Aufbau eines I²C-Busses, welcher in Abbildung 4 veranschaulicht wird.

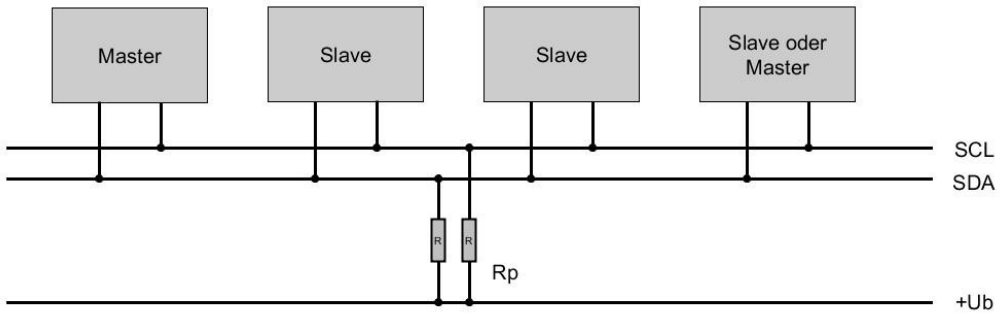

Abbildung 4: Aufbau eines I²C-Bus-Systems

Der I²C-Bus ist ein bidirektionaler Zwei-Draht-Bus mit Master-Slave-Konzept und Software-

4 **Bähring, Helmut:** *Anwendungsorientierte Mikroprozessoren: Mikrocontroller und Digitale Signalprozessoren*. Berlin Heidelberg: Springer-Verlag, 2010.–ISBN 978-3-642-12292-7 S. 287-290.

Adressierung. Die Daten werden bitseriell und synchron übertragen. Über die Taktleitung SCL (Serial Clock) wird ein Taktsignal übertragen, welches an allen Teilnehmern angeschlossen ist. Auf der Datenleitung SDA (Serial Data) werden die Nutzdaten zu den Teilnehmern transportiert. Die beiden Leitungen (SCL und SDA) werden jeweils über einen Pull-Up-Wiederstand (Rp) mit der Betriebsspannung (+Ub) verbunden. Es können beliebig viele Knoten an den I²C-Bus angeschlossen werden, bis entweder die Adressmenge erschöpft ist oder eine maximale kapazitive Belastung des Busses von 400 pF erreicht ist. Der Master initiiert jeden Datentransfer und erzeugt das Taktsignal für den Bus (SCL). Der I²C-Bus ist Multimaster-fähig, d.h. jeder Knoten kann als Master oder Slave agieren, des Weiteren können alle Knoten des I²C-Busses sowohl Sender, als auch Empfänger sein. Es können auch Geräte mit unterschiedlichen Übertragungsraten gemischt eingesetzt werden. Bei einem Slave mit Standard-Übertragungsrate von 100kBit/s (Standard-Mode) und einem Master mit schnellerer Übertragungsrate kann der Slave das SCL-Signal durch Verzögerung der Taktphase im Low-Pegel beliebig verlängern und der Master geht währenddessen in einen Wartezustand (Wait-State), somit findet die Übertragung mit der Übertragungsrate des Slave statt. Im umgekehrten Fall, d.h. ein Master mit Standard-Mode-Übertragungsrate von 100kBit/s und einem Slave mit schnellerer Übertragungsrate findet die Übertragung aufgrund des vom Master bereitgestellten SCL-Signals mit der Übertragungsrate des Master, also im Standard-Mode statt.

5.1.2 Die Kommunikation auf dem I²C-Bus

Beim I²C-Bus werden die Daten in festen Rahmen übertragen die jeweils ein Byte groß sind. Durch den Master werden besondere Start-/Stopp-Bedingungen auf den Bus-Leitungen erzeugt, welche den Beginn und das Ende eines Rahmens signalisieren. In der Abbildung 5 wird dies veranschaulicht.

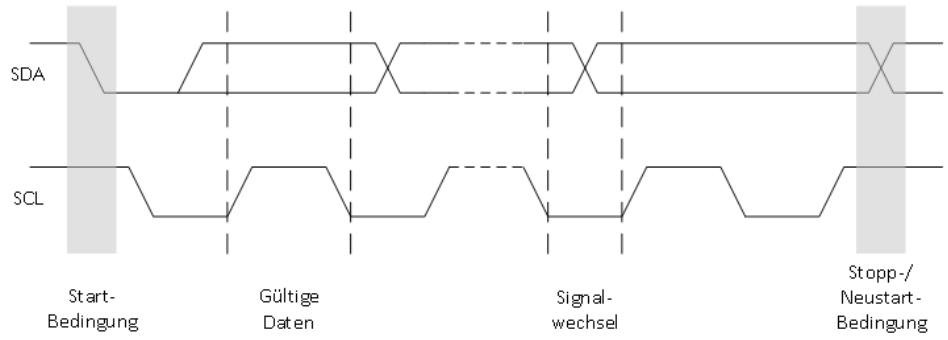

Abbildung 5: Start-/Stopp-Bedingungen beim I²C-Bus.

Erst bei einem High-Pegel des SCL-Signals sind die Daten gültig und dürfen übernommen werden. Das Datensignal SDA darf während der Datenübertragung seinen Zustand nur dann

ändern, wenn sich das SCL-Signal auf Low-Pegel befindet. Zum Kennzeichnen der Start-/Stopp-Bedingung dient eine Verletzung dieser Regel. Bei der Start-Bedingung weist das SDA-Signal eine negative Flanke auf, während sich das SCL-Signal auf High-Pegel befindet. Die Stopp-Bedingung wird dadurch angezeigt, dass das SDA-Signal eine positive Flanke aufweist, während sich das SCL-Signal auf High-Pegel befindet, dies gibt den Bus wieder frei. Durch eine Neustart-Bedingung, bei der das SDA-Signal am Ende einer Übertragung eine negative Flanke aufweist, während sich das SCL-Signal auf High-Pegel befindet, ist der aktuelle Master in der Lage unmittelbar eine weitere Übertragung anzuschließen. Daten werden auf dem I²C-Bus stets byteweise transportiert, wobei jedes Byte mittels eines Bestätigungs-Bit (Acknowledgement, ACK) gesondert quittiert und mit dem höherwertigen Bit (Most Significant Bit, MSB) zuerst ausgegeben wird. Das Bestätigungs-Bit wird auf der SDA-Leitung als Low-Pegel übertragen und liegt direkt als neuntes Bit nach acht übertragenen Datenbits. Falls das Bestätigungs-Bit einen High-Pegel besitzt, so teilt der Empfänger mit, dass dieser nicht mehr an der Kommunikation teilnimmt und kein weiteres Byte mehr empfangen kann. Die Kommunikation auf einem I²C-Bus durchläuft zwei Phasen, zum einen die Adressierung in der die Selektion des Empfängers stattfindet und zum anderen eine weitere Phase in der die Übertragung der Nutzdaten stattfindet.

5.1.3 Adressierung

Die Adressierung der Knoten kann entweder mit einer 7 oder 10-Bit-Adresse erfolgen. Die ursprünglich vorgesehene 7-Bit-Adressierung wurde aus Mangel an verfügbaren Adressen um die Möglichkeit einer 10-Bit-Adressierung erweitert, damit sind bis zu 1024 Teilnehmer an einem I²C-Bus möglich. Durch die Kombination beider Adressierungsarten können 1136 Teilnehmer an einem Bus erreicht werden. Mit der 7-Bit-Adressierung sind 112 Teilnehmer möglich, da 16 der 128 möglichen Adressen für Sonderzwecke, zum Beispiel für die 10-Bit-Adressierung oder für zukünftige Erweiterungen reserviert sind. Bei der 7-Bit-Adressierung werden zunächst die 7-Adressbits übertragen. Das letzte Bit dieses ersten Byte ist das Lese- Schreibrichtungs-Bit (R/W), dieses gibt dem Empfänger zu verstehen, ob der Master lesen (R/W=1) oder schreiben möchte (R/W=0). Im Falle einer 10-Bit-Adressierung werden zuerst die ersten 5 Bit der für die 10-Bit-Adressierung reservierten Adresse, zwei Bit der eigentlichen Adresse und das Lese-Schreibrichtungs-Bit gesendet. Dies geschieht zur Konfliktvermeidung mit Knoten, die für eine 7-Bit-Adressierung vorgesehen sind. Nach der Bestätigung des ersten Byte folgt jetzt der zweite Teil der eigentlichen Adresse, dies ist diesmal ein vollständiges Byte. Die Phase der Adressierung endet mit einer Bestätigung durch den angesprochenen Knoten und dem Übergang hin zur Nutzdatenübertragung.

5.1.4 Nutzdatenübertragung

Während der Phase der Nutzdatenübertragung werden die Nutzdaten byteweise solange übertragen bis die Übertragung des gesamten Pakets regulär durch den Sender oder vorzeitig durch den Empfänger beendet wird. Ein reguläres Ende der Übertragung, bei der der Master der Sender ist, zeigt der Master durch die Stopp- bzw. Neustart-Bedingung an. Falls der Master der Empfänger ist, so kann er die Übertragung ebenfalls nach jedem empfangenen Byte durch eine Stopp- bzw. Neustart-Bedingung vorzeitig abbrechen. Kann der Slave als Empfänger keine weiteren Daten mehr annehmen, so quittiert er das zuletzt empfangene Byte nicht mehr und der Sender, also der Master, bricht die Übertragung anschließend mit der Stopp-Bedingung vorzeitig ab. Folgenden Abbildung 6 veranschaulicht eine komplette Datenübertragung auf dem I²C-Bus, bei der der Master der Sender ist, von der Start-Bedingung zur Adressierung über das Senden eines Byte bis hin zur Stopp-Bedingung.

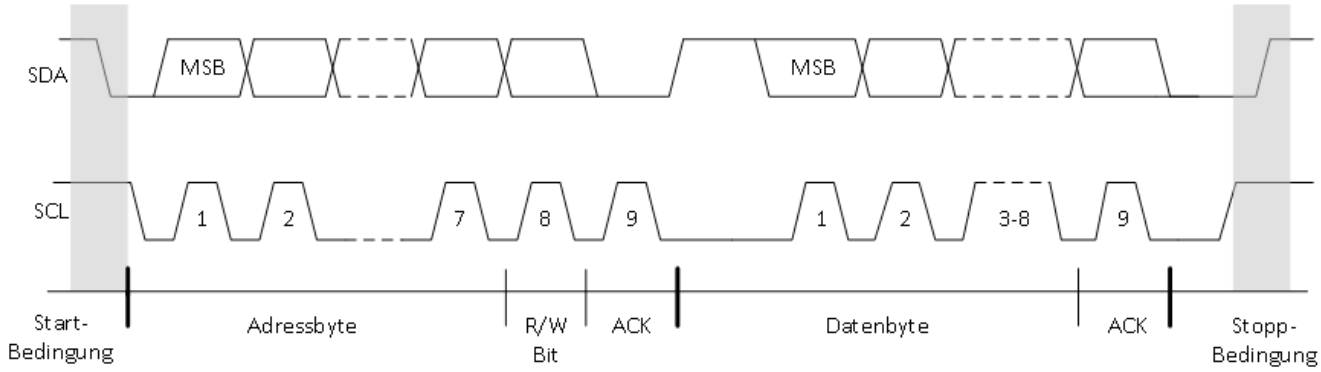

Abbildung 6: Datenübertragung auf dem I²C-Bus

5.2 One-Wire-Bus[5]

One-Wire oder 1-Wire ist ein digitaler, serieller Bus des Herstellers Maxim (ehem. Dallas), der in seiner kleinsten Ausbaustufe mit einer Datenleitung (DQ) und einer Masseleitung (GND) auskommt. Die Bezeichnung One-Wire leitet sich daher ab, dass zu der vorhandenen Masseleitung nur eine weitere Ader, die Datenleitung, für die gesamte Buskommunikation notwendig ist. In einigen Dokumenten wird der One-Wire-Bus auch als "MicroLAN" bzw. "Single-Wire Serial Interface" bezeichnet. Ursprünglich wurde der One-Wire-Bus für die Kommunikation zwischen den Komponenten eines Gerätes bzw. innerhalb eines Schaltschrankes entwickelt. Im Laufe der Zeit wurde er als ein einfach anwendbares Bussystem für längere Strecken, mit bis zu mehreren hundert Metern, erweitert ("1-Wire Extended Network Standard"). Dies wurde

5 https://prof.hti.bfh.ch/uploads/media/Studienarbeit_1-Wire.pdf

insbesondere durch verbesserte Busmaster / Buskoppler mit Kontrolle der Signalanstiegszeit ("Slew Rate Control") und aktiven Pull Up Widerstand, sowie durch Rauschfilter und ein optimiertes Timing erreicht.

5.2.1 Aufbau des One-Wire-Bus-Systems

Für den One-Wire-Bus als Datenbus wird nur ein einziges Kabel benötigt, außerdem können die meisten One-Wire-Komponenten parasitär mit Strom aus der Datenleitung versorgt werden, wodurch mit zwei Leitungen sowohl die Datenübertragung, als auch die Stromversorgung abgedeckt wird. Der Datenbus wird standardmäßig mit einem Pullup-Widerstand an 3-5,5 V gelegt. Die One-Wire-Komponenten bezieht ihren Strom von der Datenleitung (DQ) und können für kurze Zeiträume (z.B. das Senden von Bits) mit einem integrierten Kondensator die Low-Zeiten von DQ überbrücken. Folgende Abbildung 7 veranschaulicht den Aufbau eines One-Wire-Bus-Systems.

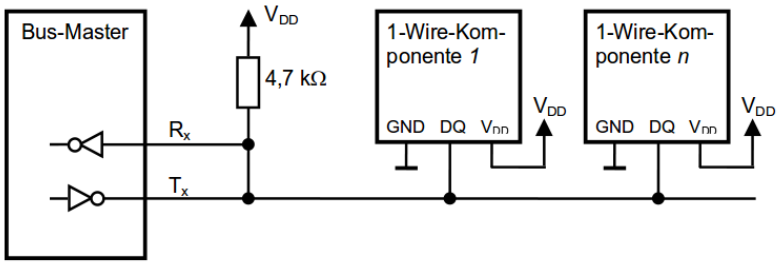

Abbildung 7: Aufbau des One-Wire-Bus-Systems

Da manche One-Wire-Komponenten aber zumindest temporär mehr Strom benötigen, als durch den Pullup-Widerstand ohne größeren Spannungsabfall fließen könnte, und der integrierte Kondensator diesen Strom ebenfalls nicht für die nötige Zeit aufrecht erhalten könnte, gibt es hierbei zwei Varianten, dieses Problem zu lösen:

- Die Komponente wird mit einer zusätzlichen Leitung mit Strom versorgt
- Die Komponente wird über DQ mit Strom versorgt

5.2.2 Kommunikation auf dem One-Wire-Bus

Da das Konzept des 1-Wire-Systems auf einer Datenleitung basiert, kann es kein zusätzliches Taktsignal geben, aus diesem Grund gibt es One-Wire-Protokoll sogenannte "Time-Slots". Ein Time-Slot ist ein festes Zeitfenster von 60µs, in dem genau ein Bit über den One-Wire-Bus übertragen wird; der Spannungsverlauf an DQ entscheidet darüber, ob eine "1" oder eine "0" übertragen wurde. Des Weiteren wird ein Time-Slot immer durch ein kurzes "Low" vom Master initiiert egal ob vom Master ein Bit gesendet oder empfangen werden soll. Die Kommunikation über den One-Wire-Bus läuft in den folgenden drei Schritten ab:

- **Initialization**

  Hier geben Master und Slave des 1-Wire-Netzwerks erstmals "Lebenszeichen" von sich: Der Master führt einen Reset durch, auf den der Slave mit einem "Presence-Pulse" antwortet. Dies zeigt dem Master, dass mindestens ein Slave am One-Wire-Bus angeschlossen ist.

- **Rom-Command**

  Dieser Befehl vom Master bezieht sich immer auf die einmalige ROM-ID jedes One-Wire-Devices. Er kann z.B. eine bestimmte Komponente adressieren oder alle Komponenten ansprechen. Damit wird also übermittelt, für "wen" der folgende Function-Command bestimmt ist.

- **Function-Command**

  Hier kommt erst der eigentliche Befehl des Masters an den Slave. Während die Schritte 1 und 2 bei allen 1-Wire-Komponenten gleich sind, beinhaltet der Function-Command bauteilspezifische Befehle (z.B. bei Temperatursensoren "wandele die Temperatur um" oder bei digitalen Switches "öffne folgenden Schalter").

5.2.3 Die Kommunikation im Detail

Folgende Abbildung 8 veranschaulicht die Kommunikation über den One-Wire-Bus anhand der Kommunikation mit einem digitalen Temperatursensor DS18S20.

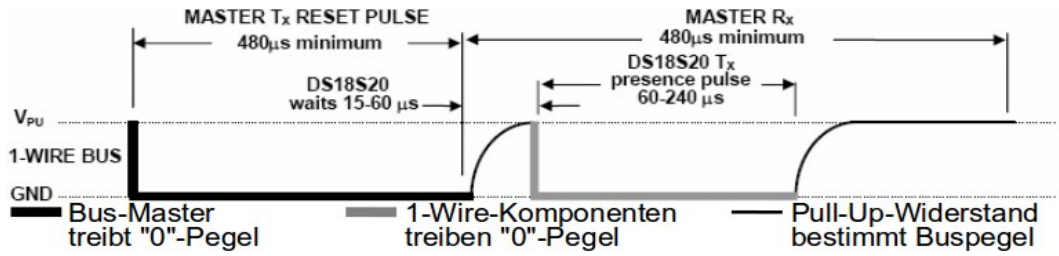

Abbildung 8: Kommunikation über den One-Wire-Bus

Reset und Presence-Pulse (Initialization)

Nach dem One-Wire-Standard steht am Anfang jeglicher Kommunikation der Reset. Dazu hält der Master DQ für mindestens 480 µs auf Low und überlässt die Datenleitung danach wieder dem Pullup-Widerstand, der DQ auf V_{PU} zieht. Nach weiteren 15–60 µs ziehen alle angeschlossenen Slaves DQ für ca. 60–240 µs auf Low. Dies ist für den Master das Zeichen, dass mindestens ein Slave mit dem Bus verbunden ist (der sogenannte Presence-Pulse). Danach muss gewartet werden, bis die 960 µs (480 µs Reset-Pulse und 480 µs Presence-Detection) der Initialisierung verstrichen sind und der eigentliche Datenaustausch stattfinden kann.

Senden von Bits an eine One-Wire-Komponente

Durch den Pullup-Widerstand wird die Datenleitung des One-Wire-Bus bei keiner Bus-Aktivität immer auf V_{PU} gezogen. Daher basiert die gesamte Kommunikation auf unterschiedlichen Low Zeiten. Um eine "0" zu übertragen, zieht der Master DQ für die Dauer des gesamten Time-Slots (min. 60 µs) auf Low und überlässt danach DQ wieder dem Pullup-Widerstand, der DQ auf V_{PU} bringt. Eine "1" wird ähnlich übertragen: Der Master zieht DQ auf Low, wartet hier jedoch maximal 15 µs, um die Datenleitung durch den Pullup wieder an V_{PU} zu legen. Nachdem DQ wieder High ist, muss noch die restliche Zeit eines Time-Slots gewartet werden, bis das nächste Signal übertragen werden kann.

Empfangen von Bits einer One-Wire-Komponente

Ein Grundsatz des 1-Wire-Systems ist, dass ein Slave nie "unaufgefordert" agiert. Das bedeutet, dass das Empfangen eines Bits ebenfalls durch den Master initiiert werden muss. Dies geschieht dadurch, dass der Master DQ für min. 1 µs auf Low zieht und dann DQ wieder freigibt. Nun kommt es auf die Reaktion des Slave an, ob er eine "0" oder eine "1" empfangen wird. Ist der Datenbus nach dem "loslassen" des Masters immer noch Low (der Slave zieht hier DQ also auf Low), so bedeutet dies, dass der Slave eine "0" gesendet hat. Wird DQ jedoch sofort nach dem kurzen Low des Masters wieder durch den Pullup an V_{PU} gelegt, so hat der Slave eine "1" gesendet. Auch hier muss bis zum Ende des Time-Slots gewartet werden, bevor weitere Kommunikation stattfinden kann.

6 Umsetzung des Projekts

In diesem Kapitel wird der gesamte Verlauf der Implementierung des Projektes beschrieben. Dazu werden unter Punkt 6.1 zum ersten die bei diesem Projekt verwendeten Entwicklungswerkzeuge näher beschrieben. Als zweites folgt eine detaillierte Beschreibung der entwickelten Software, Punkt 6.2. Schließlich wird unter Punkt 6.3 die für das Projekt entwickelte, elektronische Schaltung näher beschrieben.

gehören folgende Punkte:

- die eingesetzten Werkzeuge (siehe 6.1)
- die Beschreibung des Programms (6.2)
- die Beschreibung der Schaltung (6.3)

6.1 Verwendete Entwicklungswerkzeuge

Da für das Projekt eine Arduino-Entwicklungsplatine gewählt wurde, wurde für die Entwicklung der Software die Arduino-IDE verwendet. Dem Umstand, dass sich die Arduino-Entwickler aufgespalten haben[6], wird verdankt das derzeit zwei unterschiedliche Arduino-IDEs existieren. Diese unterscheiden sich zum Teil in der Bedingung (zum Beispiel: Verwaltung der benutzten Bibliotheken) und in der Unterstützung der Entwicklungsplatinen. Hier wurde sich für die Verwendung einer Arduino-IDE aus folgender Quelle entschieden: http://www.arduino.cc/. Des Weiteren wurde entschieden, die Version 1.6.8 der Arduino-IDE[7] zu verwenden.

Die Arduino-IDE beinhaltet einen Texteditor, welcher Hervorhebungen der Syntax und eine automatische Formatierung des Quelltextes beherrscht. Da diese Funktionen der Arduino-IDE jedoch als mangelhaft angesehen wurden, wurde für das Schreiben, bzw. das Editieren des Quelltextes die Qt Creator-IDE[8] verwendet. Diese IDE bietet hierfür wesentlich mehr Komfort und ermöglicht ein übersichtlicheres Entwickeln der Software. Aufgrund der Verträglichkeit mit den Arduino-Entwicklungsplatinen wurde das Projekt aber weiterhin mit der Arduino-IDE kompiliert und auf die Entwicklungsplatine geladen. Für das Debuggen wurde der serielle Monitor verwendet, welcher ebenfalls in der Arduino-IDE enthalten ist.

6 http://www.arduino.cc/ und http://www.arduino.org/

7 Die Aktuellste Version kann unter dem Link heruntergeladen werden: https://www.arduino.cc/en/Main/Software.

8 http://www.qt.io/download-open-source/

6.1.1 Einstellen der Arduino-IDE

Um das Programm auf die jeweilige Entwicklungsplatine übertragen zu können, sollte die Arduino-IDE korrekt eingestellt werden. Dazu muss im Menü „Werkzeuge“ die richtige Entwicklungsplatine und der richtige Mikrocontroller ausgewählt werden. In diesem Fall das Arduino-Nano-Board als Entwicklungsplatine und der ATmega 238 als Mikrocontroller.

Da es sich hier jedoch um einen Arduino-Nano Klon handelt, wird ein Treiber für den USB-Seriell-Umsetzer (CH340G) benötigt um mit der Entwicklungsplatine kommunizieren zu können. Der Treiber ist bei Ubuntu 14.04 allerdings schon vorhanden und muss für alle anderen Betriebssysteme (Windows) wahrscheinlich nachinstalliert werden.

Im Menü „Werkzeuge“ muss ebenfalls der Kommunikationsport ausgewählt werden, welcher beim Verbinden der Entwicklungsplatine mit dem PC automatisch erkannt wird.

6.2 Beschreibung des Programms

In der Arduino-IDE erfolgt die Softwareentwicklung in der Programmiersprache C++. Für ein funktionierendes Programm müssen in der Hauptquelldatei zwei Funktionen, *setup()* und *loop()* definiert werden. Die s*etup()*-Funktion wird einmalig beim Start oder beim Reset des Mikrocontrollers aufgerufen. In dieser Funktion findet die Initialisierung der Register (zum Beispiel: Ports werden als Eingänge oder Ausgänge definiert) oder von deklarierten Datenstrukturen statt. In der *loop()*-Funktion wird die Logik des Programms abgebildet. Diese Funktion kann auch als *while(1){...}*-Schleife betrachtet werden, welche immer wieder durchlaufen wird.

Der Softwaretechnische Teil des Projekts besteht aus drei Quelldateien:

- `main.ino` - die Hauptquelldatei mit der *setup()*- und der *loop()*-Funktionen
- `defines.h` - alle Definitionen und nötige Variablen bzw. Datenstrukturen
- `functions.h` - alle selbst-definierten Hilfsfunktionen

6.2.1 Verwendete Bibliotheken

Die Bibliotheken wurden in `defines.h` eingebunden. In diesem Projekt werden einige externe Bibliotheken verwendet, welche sich sehr gut mit der Arduino-IDE installieren lassen. Im Sketch-Menü geht man auf „Bibliothek einbinden“ und wählt dann „Bibliotheken verwalten“ aus, dort können die gewünschten Bibliotheken gefunden und installiert werden.

Die Implementierung des Projektes wurde durch die Einbindung von externen[9] Bibliotheken vereinfacht, da alle nötigen Protokolle und Initialisierungen (zum Beispiel die Initialisierung der Register) bereits in existierenden Bibliotheken implementiert sind. Allerdings sollte beachtet werden, dass die Bibliotheken sehr universell geschrieben sind, was zum Einem die Bibliothek für unterschiedliche Probleme einsetzbar macht, zum Anderem das Programm aber sehr schnell zu groß für den vorhandenen Speicherplatz werden lassen kann. Beim Kompilieren zeigt die Arduino-IDE den verwendeten Programmspeicher und den dynamischen Speicher des Mikrocontrollers an. Falls der benötigte Speicherplatz gegen 95%[10] tendiert sollte in Erwägung gezogen werden, das Programm dahingehend zu optimieren, dass die durch die Bibliotheken bereitgestellten Funktionalitäten ggf. anderweitig zu implementieren sind.

Für die Verwendung der Hardware-Module wurden folgende externe Bibliotheken verwendet:

`<DallasTemperature.h>`

Diese Bibliothek ist für den Einsatz des Temperatursensors benötigt. Intern verwendet diese noch eine weiters externe Bibliothek, `<OneWire.h>`[11]. Um den Temperatursensor ansprechen zu können, wird zuerst ein Objekt der Klasse OneWire angelegt. Dabei muss dem Konstruktor der Pin für die Kommunikation über den One-Wire-Bus übergeben werden. Anschließend wird ein DallasTemperature-Objekt angelegt, an welches die Referenz des Objekts der Klasse OneWire übergeben wird. Diese Schritte bewirken eine Grundinitialisierung der Mikrocontroller-Register. Ein Beispiel wird in folgendem Listing 1 gezeigt.

```
static OneWire oneWire(TEMP_SENSOR);
static DallasTemperature sensors(&oneWire);
```

Listing 1: Anlegen der OneWire- und DallasTemperature-Objekte

Als Nächstes muss der One-Wire-Bus initialisiert werden. Dies erfolgt mit dem Aufruf der Funktion *begin()* des DallasTemperature-Objektes (alle weitere Funktionsaufrufe beziehen sich auch auf dieses Objekt). Die Temperatur wird mit der Funktion *requestTemperatures()* abgefragt und in dem DallasTemperature-Objekt gespeichert. Nun kann die Temperatur mit der Funktion getTempCByIndex() zurückgegeben werden, wobei dieser Funktion eine 0 als Parameter übergeben wird, welches der Index des Temperatursensors ist.

9 Bibliotheken, welche nicht in der Arduino-IDE enthalten sind.

10 Es ist nicht bekannt wie der genutzte Speicher berechnet wird. Es kann sein, dass die lokale Variablen bzw. Objekte nicht berücksichtigt werden, deshalb wurden hier 95% als kritisch angesehen.

11 Diese Bibliothek ist für das One-Wire-Protokoll zuständig.

<LiquidCrystal_I2C.h>

Für die Benutzung der LCD-Anzeige muss ein Objekt der Klasse LiquidCrystal_I2C mit einigen Übergabeparametern angelegt werden. Ein Beispiel hierfür wird in dem folgendem Listing 2 gezeigt.

```
static LiquidCrystal_I2C lcd(0x27, 16, 2);
```

Listing 2: Anlegen des LiquidCrystal_I2C-Objektes

Der erste Parameter 0x27 bezeichnet die Adresse der LCD-Anzeige. Der zweite und dritte Parameter, also 16 und 2 geben die Anzahl der Zeichen pro Zeile und die Anzahl der Zeilen an. In diesem Fall sind es zwei Zeilen zu je 16 Zeichen.

Mit der Funktion *lcd.init()* werden das Objekt er Klasse LiquidCrystal_I2C und der I2C-Bus initialisiert. Falls die LCD-Anzeige eine Hintergrundbeleuchtung besitzt, kann diese mit der Funktion *lcd.setBacklight(uint8_t flag)* eingeschaltet oder ausgeschaltet werden. Nun kann mit der Funktion *lcd.setCursor(uint8_t pos, uint8_t line)*[12] die Position des Cursors gesetzt und mit der Funktion *lcd.print(type t)*[13] eine Information auf der LCD-Anzeige ausgeben werden.

Es kann vorkommen, dass die Bibliotheken angepasst oder auf Fehler überprüft werden müssen. So enthielt die oben genannte Bibliothek einen Fehler, durch welchen die Zeichen auf der LCD-Anzeige falsch dargestellt wurden. Der Fehler wurde gefunden und korrigiert. Später kam ein Update für diese Bibliothek und der Fehler wurde dort auch behoben.

<RtcDS3231.h>

Für die Verwendung des RTC wird zunächst ein Objekt von RtcDS3231 angelegt. Mit der unktion *rtcObject.Begin()* erfolgt im ersten Schritt die Initialisierung des I2C-Buses. Die Zeit wird in einer Datenstruktur „RtcDateTime“ verwaltet, welche auch separat angelegt werden sollte. Die aktuelle Zeit kann mit der Funktion rtcObject.GetDateTime() abgefragt und anschließend in der Datenstruktur RtcDateTime gespeichert werden.

Diese Bibliothek wurde für die Bedürfnisse dieses Projekts ein wenig erweitert. Es wurde eine Funktion für den Operator „-=“ hinzugefügt, welche es ermöglicht, die aktuelle Anzahl der Sekunden von der in der in der Klasse *RtcDateTime* gespeicherten Gesamtanzahl der Sekunden zu subtrahieren.

12 pos ⟶ Zeichenposition (mit 0 indexiert), line ⟶ Zeilennummer (mit 0 indexiert)

13 (type t) deutet auf eine polymorphe Funktion hin. Für eine genauere Funktionsbeschreibung sehen Sie bitte in den Bibliothek-Quellen nach

6.2.2 Programmablauf

Eine einleitende Beschreibung der Funktionsweise der für dieses Projekt entworfenen Akku Aufbewahrungsstation wurde im Kapitel 2 bereits erwähnt. Aus dieser Beschreibung kann die Information entnommen werden, dass das Programm mehrere voneinander getrennte Zustände hat. Diese Zustände werden im Folgendem detailliert beschrieben.

- **BAT_CAPACITY**

  Hier wird gewünschte Ladekapazität eingestellt, bis zu welcher der Akku geladen werden soll. Außerdem wird in diesem Zustand die Klemmspannung des Akkus gemessen. Falls diese einen Wert von unter 3V besitzt, so sollte der Akku ersetzt werden, da eine Klemmspannung von 3V auf einen tiefentladenen Akku hindeuten, welche nicht mit den hier umgesetzten Ladestrategien aufgeladen werden können. Ob ein Akku in die Aufbewahrungsstation eingesetzt ist, wird ebenfalls geprüft, indem die Ladestrategie1 kurzzeitig aktiviert wird. Falls kein Akku eingesetzt ist, so steigt die Klemmspannung auf mehr als 4,3V und dies ist das gewünschter Ausnahmekriterium.

- **BAT_LOW_VOLTAGE**

  Dieser Zustand kann nur aus dem Zustand **BAT_CAPACITY** erreicht werden und zeigt die Information an, dass der angeschlossene Akku tiefentladen ist. Im Anschluss daran soll die Taste „OK“ betätigt werden.

- **BAT_NO_BAT**

  Dieser Zustand wird von den Zuständen **BAT_CAPACITY, CHARGE1, CHARGE2, CHARGE3 und COOLING** erreicht. Es wird eine Information angezeigt, dass die Akku nicht in die Aufbewahrungsstation eingesetzt ist. Im Anschluss soll die Taste „OK“ betätigt werden.

- **RTC_DATE**

  In diesem Zustand wird die gewünschte Zeit eingestellt, zu welcher die Akku aufgeladen sein soll. Die Zeiteinstellung erfolgt in der Reihenfolge: Jahr → Monat → Tag → Stunde. Das aktuelle Datum und die aktuelle Zeit werden dabei angezeigt. Die Zeit kann dabei nicht in die Vergangenheit eingestellt werden. Am Ende wird die eingestellte Zeit komplett angezeigt. Dies kann dann mit der Taste „OK“ akzeptiert werden (der Zustand der Aufbewahrungsstation wechselt dann zum Zustand **CHARGE1**) oder mit der Taste „NO“ noch einmal korrigiert werden. In diesem Moment kann die Taste „SERVICE“ betätigt

werden, dann gelangt die Aufbewahrungsstation in den Zustand **RTC_SET**.

- **RTC_SET**

  In diesem Zustand wird der RTC eingestellt. Hier wird ähnlich vorgegangen wie im Zustand **RTC_DATE**.

- **BAT_CHECK**

  Dies ist ein Service-Zustand. Hier wird die offene Akkuklemmspannung angezeigt. Dieser Zustand wird von den Zuständen **CHARGE1, CHARGE2, CHARGE3** durch druck der Taste „SERVICE“ erreicht. Nach dem erneutem Betätigen der Taste „SERVICE“ gelangt die Aufbewahrungsstation in den vorhergehenden Zustand.

- **CHARGE1**

  In diesem Zustand wird die eingestellte Aufladezeit überwacht. Falls noch ein Tag bis zur eingestellten Zeit verbleibt, wird der Aufladevorgang gestartet, in dem die Aufbewahrungsstation in den Zustand **CHARGE2** gelangt. Anderenfalls ist die Akkuaufbewahrungsfunktion im Modus „aktiv“. Wenn die Klemmspannung unter einen Wert 3,5 Volt gelangt, wird die Ladestrategie1 aktiviert. Dabei wird der Akku bis 3,6V aufgeladen.

- **CHARGE2**

  In diesem Zustand wird der Akku mit der Ladestrategie2 bis zur vorher eingestellten Ladekapazität aufgeladen und anschießenden in den Zustand **CHARGE3** übergegangen. Da während der Ladestrategie2 größere Ströme (bis 100mA) fließen, wird dabei ständig die Temperatur überwacht. Falls die Temperatur 30°C überschreitet, gelangt die Aufbewahrungsstation in den Zustand **COOLING**.

- **CHARGE3**

  In diesem Zustand wird die Ladestrategie1 verwendet. Der Akku wird dabei langsam mit dem geringerem Ladestrom bis zur vorher eingestellten Ladekapazität gesättigt.

- **COOLING**

  In diesem Zustand findet eine Abkühlung des Akkus statt. Wenn die Temperatur des Akkus 30°C unterschreitet gelangt die Aufbewahrungsstation in den Zustand **CHARGE2**.

- **`WAITING`**

  Dieser Zustand ist der abschliessende Zustand, welcher den aufgeladenen Zustand „Akku aufgeladen“ symbolisiert.

Ein Programmablaufplan der Software befindet sich im Anhang A1. Die Übergänge der Zustände wurden mit einer Mehrfachverzweigung (Switch-Anweisung) realisiert.

6.3 Beschreibung der Schaltung

In diesem Kapitel wird beschrieben wie die einzelnen Peripherie-Module mit dem Arduino Nano verbunden sind. Außerdem wird auch näher auf die eigentliche Ladeschaltung eingegangen.

6.3.1 Anschluss der Peripherie-Module an das Arduino Nano

Alle für dieses Projekt eingesetzten Peripherie-Module wurden so ausgewählt, dass diese ein mit dem Arduino Nano kompatibles Bussystem aufweisen. Der Anschluss der einzelnen Peripherie-Module wird in der folgenden Abbildung 9 gezeigt, wobei die Verbindung des I2C-Parallel-Umsetzers mit der LCD-Anzeige nur schematisch dargestellt wird. Der Schaltplan dieser beiden Module wurde nicht untersucht, da diese eine kompatible Steckverbindung besitzen.

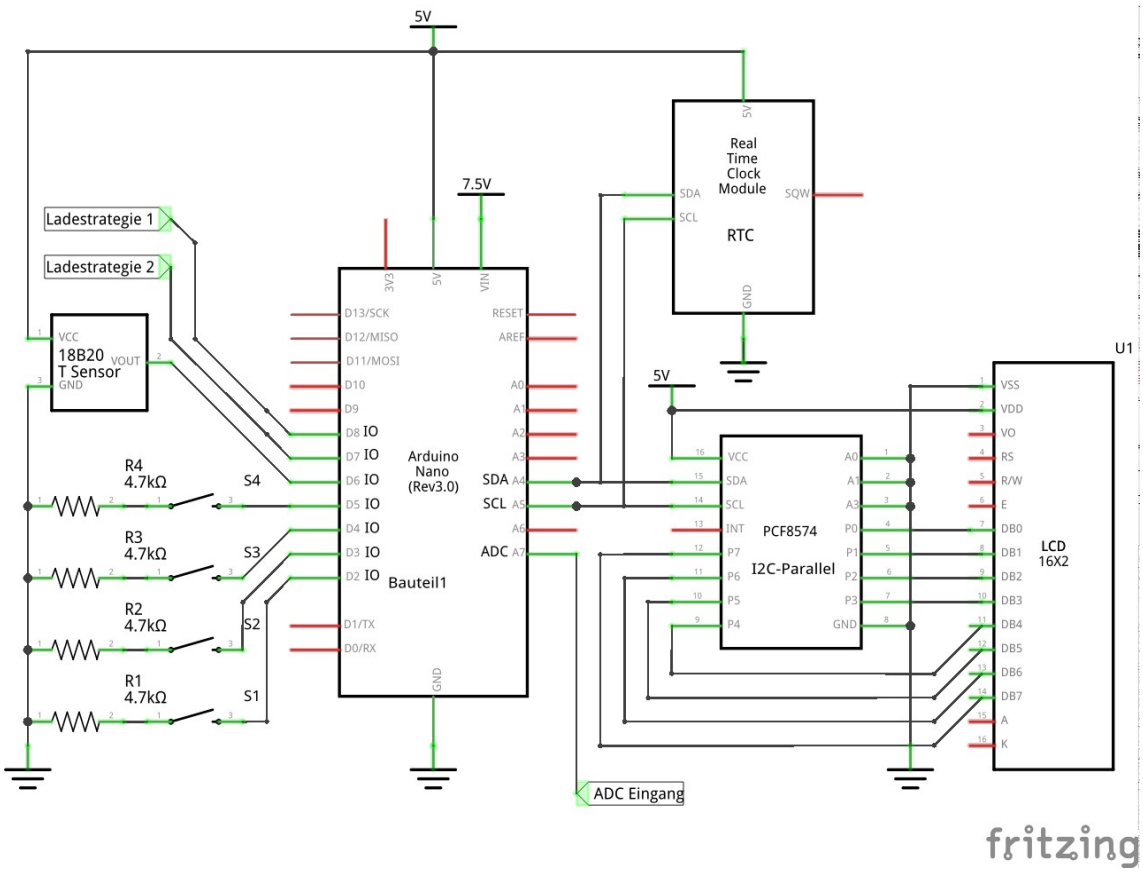

Abbildung 9: Anschluss der Peripherie-Module an die Arduino Nano-Entwicklungsplatine

Für die Bedienung der Akku-Aufbewahrungsstation werden 4 Tasten benötigt. Diese werden mit den IO-Pins D2-D5 des Mikrocontrollers verbunden. Die IO-Ports wurden als Eingänge mit den internen 10kΩ[14] Pullup-Widerständen[15] konfiguriert. Zur Sicherheit wurden die Taster jeweils noch über 4,7kΩ Widerstände mit der Masse verbunden. Für das Umschalten der Ladestrategien wurden die Pins D7 und D8 jeweils als Ausgänge konfiguriert.

Der Akku wird über den Pin A7 überwacht, welcher als ADC konfiguriert wurde. Als Referenzspannung wird die externe Versorgungsspannung benutzt. Diese wird mit einem linearen Festspannungsregler, welcher sich auf der Arduino Nano-Entwicklungsplatine befindet, auf 5,02V[16] geregelt.

6.3.2 Ladeschaltung

Die Akku wird mit zwei unterschiedlichen Ladestrategien geladen. Die erste Ladestrategie besteht aus einem Spannungsregler. In diesem Fall wird die Ladespannung konstant gehalten und der Ladestrom ändert sich umgekehrt proportional zur Akkuschlussspannung. Die zweite Ladestrategie besteht aus einem Stromregler, bei welchem der Ladestrom konstant gehalten wird.

Zum Einsatz kam hierfür ein einstellbarer linearer Spannungsregler LM317. Mit diesem Spannungsregler lassen sich die beiden Ladestrategien realisieren.

6.3.2.1 Spannungsregler

In der folgenden Abbildung wird eine Beschaltung des Spannungsreglers LM317 gezeigt, welche die erste Ladestrategie realisiert.

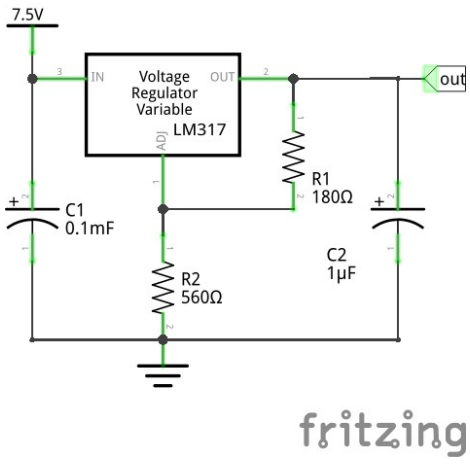

Abbildung 10: Spannungsregler

Um einen sanften Ladestrom zu erzeugen, soll die Differenz zwischen Akkuschlussspannung und der Ladespannung gering sein. Dafür wurde auf dem Labornetzteil 4,5V eingestellt und geschaut wie viel Strom zum Akku fließen, welcher auf 4,1V geladen war. Mit einem Strommessgerät wurden dabei ca. 20mA gemessen. 20mA wurden für gut befunden, weswegen die Ladespannung auf 4,5V festgelegt wurde.

14 10kΩ wird in der Spezifikation vom ATmega238-Mikrocotroller angegeben.

15 Der ATmega238-Mikrocontroller besitzt intern nur Pullup-Widerstände.

16 5,02V wurden mit einem Voltmeter gemessen.

Im Hardware-Manual zum LM317 wird eine Formel angegeben, mit welcher sich die Ausgangsspannung berechnen lässt: $V_{out}=1.25\,V\left(1+\frac{R_2}{R_1}\right)+I_{Adj}\cdot R_2$, wobei $I_{Adj}\cdot R_2$ sehr klein ist und damit vernachlässigt[17] werden kann. Die Formel wurde dann, wie folgt nach R_2 umgestellt: $R_2=\frac{(V_{out}-1{,}25\,V)\cdot R_1}{1{,}25\,V}$. Der Hersteller empfehlt für R1 einen Widerstand von 240Ω[18] zu verwenden, um eine ordnungsgemäße Spannungsregelung zu gewährleisten. Hier wurde ein 180Ω Widerstand in die Formel eingesetzt. Die Rechnung ergab damit 468Ω also ≈470Ω.

6.3.2.2 Stromregler

Dieser einstellbare Stromregler kann auch als einfacher Stromregler dienen. Folgende Abbildung 11 veranschaulicht den Schaltplan.

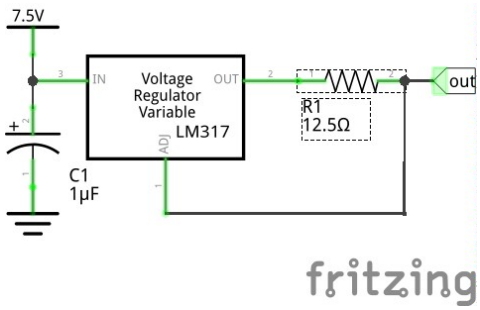

Abbildung 11: Stromregler

Der Spannungsregler stellt sich so ein, dass zwischen Adj und Vout 1,25V anliegen. Deswegen kann die Stromstärke mit folgender Formel $I=1{,}25\,V/R_1$ berechnet werden. Der für dieses Projekt verwendete Akku weist eine Ladekapazität von 820mA auf. Schließlich wurde sich für einen Ladestrom von ca. 100mA entschieden, damit der Akku in ca. 10-12 Stunden geladen ist. Obige Formel wurde wie folgt nach R_1 umgestellt, $R_1=1{,}25\,V/I$. Das Resultat der Berechnung ergibt einen Widerstand von 12,5Ω.

17 Aus dem Hardware-Manual zum LM317 übernommen.

18 Zwischen Adj und Out sollen 3,5-10mA fließen.

6.3.2.3 Zusammengesetzte Schaltung

Die Akku-Aufbewahrungsstation soll zwischen den beiden Ladestrategien umschalten können., deswegen wurden die Spannungsregler- und Stromreglerschaltung kombiniert. Folgende Abbildung 12 zeigt die resultierende Schaltung.

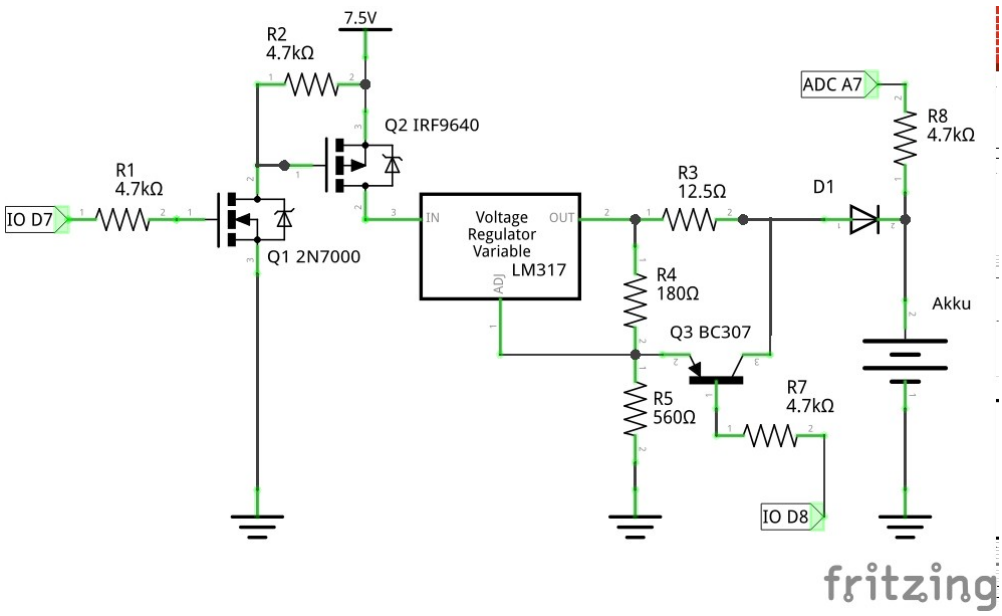

Abbildung 12: Komplette Ladeschaltung

Diese Schaltung wird mit den beiden Pins D7 und D8 gesteuert. Der Pin D7 Schaltet die Ladeschaltung ein in dem er eine Spannung an das Gate eines N-Channel 2N7000[19] MOSFETs anlegt. Dieser wird geöffnet und das Gate vom P-Channel IRF9640[20] MOSFET wird auf die Masse gezogen. Im Anschluss wird der linearer Spannungsregler LM317 mit Strom versorgt. Bei den leistungsfähigen MOSFETs soll beachtet werden, dass die Gate-Source- und die Drain-Source-Spannung ausreichend groß ist, damit der nötige Strom fließen kann. In diesem Fall liegen die beiden Spannungen bei ca. 7,5V. Aus dem Hardware-Manual des IRF9640 MOSFETs kann eine Drain-Stromstärke von über 10A agelesen werden, deshalb erwiesen sie sich als geeignet. Der wesentliche Vorteil bei der Verwendung dieser MOSFETs ist der, dass sie Leistungslos geschaltet werden könne und dadurch keine Verlustleistung an der Gate-Source-Strecke zu beachten ist..

Der Pin D8 ist bei dieser Schaltung für die Bestimmung der Ladestrategie zuständig. Wenn dieser auf HIGH gezogen wird, wird der BC307 PNP-Transistor geschlossen und der Spannungsregler wird aktiv. Anderenfalls[21] wird der Transistor geöffnet und der Stromregler wird aktiv.

19 leistungsarmer N-Channel MOSFET

20 leistungsstarker P-Channel MOSFET

21 D8 Pin auf LOW gezogen

Die Akkuanschlussspannung wird mit dem Pin A7 überwacht, welcher als ADC konfiguriert wurde. Damit keine große Rückströme zur ausgeschalteten Akku-Aufbewahrungsstation fließen, wurde in die Schaltung eine Diode D1 eingesetzt. Die Flussspannung dieser Diode liegt bei 0,6V[22], deshalb muss die Spannungsregler-Schaltung an die Diode angepasst werden. Die Berechnung $R_2 = \frac{(4{,}5\,V + 0{,}6\,V - 1{,}25\,V) \cdot R_1}{1{,}25\,V}$ ergibt einen Wert von 568,8Ω. Es wurde daher ein Widerstand R_2 von 560Ω verwendet.

22 0,6V wurden experimentell ermittelt

7 Zusammenfassung

Die Entwicklung einer Akku-Aufbewahrungsstation oder eines Ladegerätes ist mit vielen unterschiedlichen Aspekten verbunden. Dazu gehören die genaue Untersuchung der Problemstellung, welche gewisse Kenntnisse in der Elektrotechnik, Chemie und Informatik voraussetzt.

Bei dieser Akku-Aufbewahrungsstation wurden folgende Funktionalitäten realisiert:

- Informationsausgabe auf einer LCD-Anzeige
- Erkennung deines tiefentladenen Akkus
- Erkennung des Einsetzens eines Akkus ins Gerät
- Akku-Aufbewahrungsfunktion
- Aufladen des Akkus bis zur eingestellten Zeit
- zwei Ladestrategien, welche den Akku sättigen
- Temperaturschutz

Das für dieses Projekt gesteckte Ziele wurde erreicht. Die oben genannte Funktionalitäten können aber auch mit den Folgenden Funktionen erweitert werden:

- Verpolschutz
- einstellbare Stromregler-Schaltung (für unterschiedliche Akku-Kapazitäten) oder
- bessere Ladestromkontrolle, welche den Ladestrom variabel macht
- Feststellung der eigentlichen Akku-Kapazität (bei unterschiedlichen Akku-Typen)
- Akku-Wiederbelebungs-Ladestrategie für Tiefentladene Akkus

Literaturverzeichnis

[Bae]

Bähring, Helmut: *Anwendungsorientierte Mikroprozessoren: Mikrocontroller und Digitale Signalprozessoren*. Berlin, Heidelberg: Springer-Verlag, 2010.–ISBN 978-3-642-12292-7

Internetquellen:

http://home.arcor.de/RoBue/1-Wire/Download/1-Wire%20Bussystem_Grundlagen_Tipps.pdf (23.03.2016)

https://prof.hti.bfh.ch/uploads/media/Studienarbeit_1-Wire.pdf (23.03.2016)

http://batteryuniversity.com/learn/article/ (23.03.2016)

http://www.elektronik-kompendium.de/sites/bau/0201113.htm (23.03.2016)

http://www.elektronik-kompendium.de/public/schaerer/uregspec.htm (23.03.2016)

http://playground.arduino.cc/Learning/OneWire-DE (23.03.2016)

http://www.hobbytronics.co.uk/ds18b20-arduino (23.03.2016)

https://www.arduino.cc/en/Reference/Float (23.03.2016)

Anhang

A1 Programmablaufplan