Unterschrift des Betreuers

TECHNISCHE UNIVERSITÄT WIEN

Diplomarbeit

Konzept und Test einer Steuerung für einen magnetischen Wanderwellenresonator zur Wellenlängenselektion von langsamen Neutronen

ausgeführt am
Atominstitut
der Technischen Universität Wien

unter der Anleitung von

Em.Univ.Prof. Dipl.-Ing. Dr.techn. Gerald Badurek
und
Ass.Prof. Dipl.-Ing. Dr.techn. Erwin Jericha

durch
Arno Frank, BSc.
Schüttelstraße 81/13
1020 Wien

Datum

Unterschrift Student

Zusammenfassung

Im Rahmen dieser Arbeit soll ein Konzept für einen magnetischen Spin-Resonator zur Wellenlängenselektion von Neutronenstrahlen im Energiebereich kalter und thermischer Neutronen präsentiert werden. Das Grundkonzept, die räumliche magnetische Spin-Resonanz, wurde dabei schon von Drabkin et al. in den 1960er Jahren vorgestellt. Dabei bewegen sich antiparallel zu einem vertikalen statischen Magnetfeld polarisierte Neutronen horizontal durch ein transversales oszillierendes, aber zeitlich konstantes Magnetfeld. Die Wechselwirkung der Neutronen mit einer solchen NMR-artigen Kombination gekreuzter Magnetfelder führt zu einem resonanten Spinflip-Prozess für eine nur von der Geometrie des Resonators und der Stärke des vertikalen Magnetfeldes abhängigen Wellenlänge. Daher können nur Neutronen dieser einen Wellenlänge einen hinter dem Resonator angebrachten Analysator passieren, da deren Polarisation aufgrund ihres Spinflips zur gegebenen Orientierung des Analysators passt.

Um diesen kontinuierlichen monoenegetischen Strahl in für Flugzeitmessungen erforderliche Neutronenpakete zu zerhacken, genügt es im Prinzip, das transversal alternierende Magnetfeld des Resonators einfach periodisch ein- und auszuschalten. Um dabei jedoch die kleinstmögliche Pulsbreite erzielen zu können, ist es erforderlich, dass synchron mit der Durchflugszeit der Neutronen sukzessive jeweils nur ein einziges der aufeinanderfolgenden Resonatorelemente aktiviert wird, um die durchfliegenden Teilchen sozusagen durch den Resonator zu 'begleiten'. Im Gegensatz zum 'konventionell' gepulsten Betrieb ist dies der sogenannte *'Wanderwellenmodus'* eines solchen räumlichen Neutronen-Spinresonanz-Systems.

Erfolgreiche Tests mit dieser Art der Wellenlängenselektion wurden bereits mit sehr kalten Neutronen erzielt, durch ein neues Konzept der Stromversorgungen für die einzelnen Resonatorelemente soll nun auch die Selektion im Bereich thermischer Neutronen ermöglicht werden. Da diese eine höhere Geschwindigkeit besitzen, müssen die Magnetfelder, die durch die speziellen Einzelwindung-Resonatorspulen erzeugt werden, einerseits höher sein und sich andererseits auch schneller stabilisieren, da die Zeit, während der sich ein Neutron im Einflussbereich einer Spule befindet, deutlich kürzer ist.

Aus diesem Grund wurde eine neue Stromversorgung konzipiert, die in der Lage ist, den erhöhten Ansprüche zu genügen. Um den Wanderwellenmodus realisieren zu können, wird jede Spule von einer eigenen Stromversorgung separat angesteuert. Die vorliegende Arbeit beschäftigt sich im Wesentlichen mit dieser neuen, ziemlich komplexen Stromversorgung und liefert ein Konzept, sowohl in hardware- als auch softwareseitiger Hinsicht, das eine Wellenlängenselektion, ob gepulst oder permanent, im Bereich von kalten bis hin zu thermischen Neutronen ermöglicht.

Abstract

In this thesis, a concept for a magnetic spin-resonator enabling a wavelength selection of cold and thermal neutron beams is presented. The main concept, the spatial magnetic spin-resonance, has been presented by Drabkin et al. in the 1960's. There, neutrons, polarized antiparallel to a vertical homogeneous static magnetic field, propagate horizontally through a transversal spatially alternating, but constant in time, magnetic field. The interaction of the neutrons with such a NMR-like arrangement of crossed magnetic fields leads to a resonant spinflip process just for a certain wavelength, which depends only on the resonator geometry and the strength of the vertical magnetic field. Thus only neutrons of this specific wavelength can pass an analyzer mounted behind the resonator, since due to their spinflip their polarization is properly aligned along the analyzer orientation.

In order to chop this continuous monoenergetic beam into neutron packets which are required for time-of-flight measurements it is sufficient simply to turn the transversally alternating magnetic field of the resonator on and off periodically in time. However,to achieve the smallest possible pulse width it is necessary that synchronously with the neutron passage time only one of the successive resonator should be activated, thereby 'accompanying' the particles during their propagation through the resonator. In contrast to the 'conventionally' pulsed mode of operation this is the so-called 'travelling-wave' mode of such a spatial magnetic spin resonance system.

Successful tests with this kind of wavelength selection have already been performed on very cold neutrons, a new concept for the power supplies of the resonator coils should make it possible to select also wavelengths in the range of cold and thermal neutrons. Since they have a higher velocity, the magnetic fields created by the resonator coils have to be stronger and they have to be switched on and off faster, since the time a neutron is in the magnetic field of a single coil is significantly shorter.

For this reason, a new kind of power supply was created which is able to satisfy these increased demands. To realize the travelling-wave mode, every single coil is being supported separately by its individual power supply. This thesis essentially deals with these new, quite complex power supplies and presents a concept, from a hardware and software point of view, on how to manage these power supplies in order to enable a wavelength selection, either pulsed or permanent, in the range from cold to thermal neutrons.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Klassische Methoden zur Wellenlängenselektion und Pulsformung, dem sogenannten 'Choppen' von Neutronenstrahlen beruhen entweder auf periodischer Strahlunterbrechung durch rotierendes, teilweise neutronenabsorbierendes Material oder auf rotierenden Einkristallen, die gleichzeitig als Monochromatoren fungieren. Ein typisches Beispiel für eine mechanische Methode ist der sogenannte 'Scheibenchopper', der aus einer mit einem oder mehreren Schlitzen versehenen Scheibe aus neutronenabsorbierendem Material besteht. Versetzt man diese in Rotation, so wird der Strahl periodisch gepulst, da natürlich nur jene Neutronen den Chopper passieren können, für die zum Zeitpunkt ihres Eintreffens der Schlitz den Weg freigibt. Eine Wellenlängenselektion kann mit einer einzelnen Scheibe bei einem polychromatisch einfallenden Neutronenstrahl allerdings nicht vorgenommen werden, sondern erfordert mindestens zwei in einem bestimmten Abstand zueinander positionierten Scheiben, die mit exakt gleicher Drehzahl rotieren. Ein weiterer Nachteil solcher Chopper ist, dass die erzielbare Pulsbreite untrennbar von ihrer Drehzahl abhängt. Grundsätzlich will man schnell bewegende Teile im Versuchsaufbau aus Sicherheitsgründen eher vermeiden, aber bei modernen Vertretern dieser Art von Neutronen-Choppern werden bereits Drehzahlen bis zu etwa 25000 pro Minute erreicht [1] (bzw in Spezialanwendungen sogar noch darüber hinaus).

Bei der Wellenlängenselektion mittels Kristallen wird die Bragg-Bedingung für die Selektion der Wellenlänge ausgenutzt. Allerdings muss bei dieser Methode für jede Veränderung der gewünschten Wellenlänge sowohl der Kristall selbst als auch der gesamte restliche Versuchsaufbau gedreht werden, da der reflektierte Strahl seine Richtung verändert. Dazu ist das hier entstehende Spektrum von den speziellen Eigenschaften des verwendeten Kristalls abhängig und für einen vorgegebenen Kristallreflex fix definiert.

Das Konzept des hier vorgestellten magnetischen Wanderwellen-Spinresonators MONOPOL vermeidet solche Probleme von vornherein, da die Wellenlängenselektion nicht mechanisch, sondern durch Magnetfelder bewirkt wird. Die

selektierte Wellenlänge ist dabei von der Stärke dieser Magnetfelder abhängig, welche bei entsprechender Hardware softwareseitig kontrolliert werden kann. Darüber hinaus lässt sich die Auflösungsvermögen des Resonators jederzeit auf rein elektronischem Weg variieren. Zusätzlich kann der Resonator auch jederzeit mit einstellbarer Repeptitionsrate und Breite der Pulse gepulst betrieben werden, wodurch ein ursprünglich kontinuierlicher polychromatischer Strahl in periodische monochromatische Neutronenpakete zerlegt werden kann.

In dieser Arbeit wird ein Konzept für die Steuerung des MONOPOL-Wanderwellenresonators mit einem neu entwickelten, ziemlich komplexen aus bis zu 48 einzelnen Stromquellen bestehenden Stromversorgungssystem vorgestellt. Die einzelnen Stromversorgungen selbst wurden bereits im Rahmen von zwei Projektarbeiten getestet [2, 3]. Die vorliegende Arbeit soll nun erörtern, wie eine zeitkritische Steuerung realisiert werden kann, die eine Wellenlängenselektion bzw. eine Wellenlängenselektion inklusive Choppens an einem weißen Neutronenstrahl im Bereich von sehr kalten bis zu thermischen Neutronen zu erlaubt.

Dabei wird zunächst die Funktionsweise des Wanderwellenresonators zusammengefasst, bevor die benötigte Hardware zur Steuerung beschrieben wird. Daraufhin geben Assemblercode-Beispiele einen Einblick, welche programmtechnischen Möglichkeiten zur Steuerung des Wanderwellenresonators bestehen, bevor dann konkrete Tests an der Hardware durchgeführt werden, um diese auf ihre Funktionalität zu prüfen.

1.1 Eine historische und funktionelle Beschreibung von MONOPOL

Zunächst soll auf das Funktionprinzip von MONOPOL eingegangen werden. Dieser hat zwei Funktionen: Einerseits kann er aus einem polarisierten weißen Neutronenstrahl, in dem Neutronen unterschiedlicher Wellenlängen mit einer bestimmten spektralen Verteilung enthalten sind, einen bestimmten Bereich an Wellenlängen $\lambda \pm \Delta\lambda$ selektieren. In Abb. 1.1 ist der schematische Aufbau des MONOPOL-Experiments dargestellt.

Es sind also folgende zwei Betriebsmodi möglich:

- Wellenlängenselektor
- Wellenlängenselektor & Chopper

Eine Verwendung ausschließlich als Chopper ohne jegliche Wellenlängenselektion ist prinzipiell nicht möglich, was aber für die praktische Anwendung ohnehin nicht von Relevanz ist.

Monochromatische Neutronenstrahlen sind für eine Vielzahl von Experimenten der angewandten Neutronenstreuung, aber auch der Grundlagenforschung erforderlich, darunter das PERC (Proton Electron Radiation Channel)-Experiment [4], bei dem der Beta-Zerfall von Neutronen mit hoher Präzision untersucht wird und in dessen Rahmen das MONOPOL-Projekt gefördert wurde.

Die Methode zur Wellenlängenselektion von Neutronen mittels räumlicher magnetischer Spinresonanz wurde bereits 1963 von Drabkin et al. erstmals vorgeschlagen [6] und danach in weiterer Folge 1991 von Badurek und Jericha [5] entscheidend weiterentwickelt.

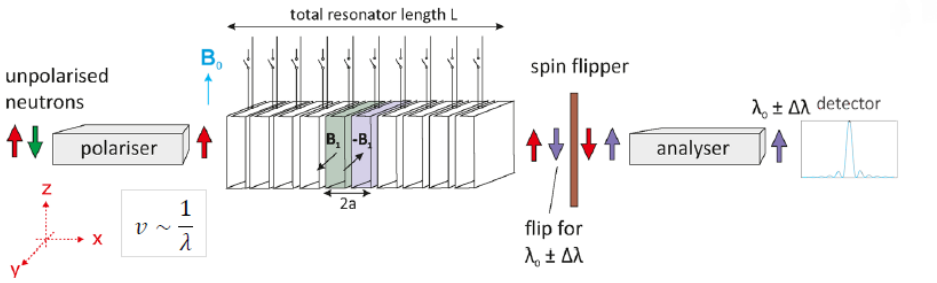

Abbildung 1.1: Schematische Darstellung eines Wanderwellenresonators zur Wellenlängenselektion, nach [7].

Dabei wird der anfänglich unpolarisierte, entlang der horizontalen x-Richtung propagierende polychromatische Neutronenstrahl zunächst durch einen Polarisator - meistens ein sogenannter Suppermirror - in die vertikale z-Richtung polarisiert. Entlang des Strahlwegs zwischen Polaristor und einem hinter dem Resonator angeordneten gleichartigen Supermirror-Analysator verhindert ein vertikales homogenes statisches Führungsmagnetfeld einen allfälligen Verlust an Strahlpolarisation. Im Bereich des Resonators ist diesem nunmehr'Selektorfeld' genannten vertikalen Feld $\vec{B}_0$ ein räumlich alternierendes, transversales Magnetfeld $\vec{B}_1(x)$ überlagert, welches von einer Reihe aufeinander folgender und jeweils nur aus einer einzelnen Windung bestehenden Aluminium-Spulen abwechselnd in y- und -y-Richtung erzeugt wird.

Das Neutron ist zwar elektrisch neutral, besitzt aber aufgrund seiner inneren Quark-Struktur dennoch ein anomales magnetisches Moment und wechselwirkt somit mit magnetischen Feldern. Sobald ein Neutron in den Resonator eintritt, vollführt es in den einzelnen Resonator-Spulen eine Larmorpräzession um die jeweils resultierenden lokalen Magnetfelder, die nun nicht mehr in die z-Richtung zeigen. Das im Laborsystem räumlich alternierende Magnetfeld $\vec{B}_1(x)$ führt im Ruhesystem des Neutrons zu einem zeitlich

oszillierenden Feld $\vec{B}_1(t) \equiv \vec{B}_1(x/v)$, dessen Frequenz nur von der Geschwindigkeit v des Neutrons abhängig ist. Wenn die Oszillationsfrequenz des transversalen Magnetfeldes mit der Larmorfrequenz der Neutronen übereinstimmt und die Amplitude so gewählt wird, dass der Larmor-Rotationswinkel für die gesamte Flugstrecke durch den Resonator einem ungeraden Vielfachen von π entspricht, findet in Analogie zur einer klassischen NMR-Anordnung gekreuzter Magnetfelder ein resonanter Spinflip-Prozess statt (für Details siehe z.B. [7]). Dieser wellenlängenabhängige Spinflip ist letztlich für die Wellenlängenselektion entscheidend. Denn ein zwischen Resonator und Analysator angebrachter wellenlängenunabhängiger, sogenannter 'Stromblatt'-Flipper bewirkt, dass nur die zuvor im Resonator schon einmal geflippten Neutronen den Analysator passieren können, während alle anderen blockiert werden. Selbstverständlich könnte eine derartige 'Dunkelfeld'-Konfiguration genauso realisiert werden, wenn der Stromblatt-Flipper vor dem Resonator angeordnet wäre.

Dabei kann für konstruktiv vorgegebene räumliche Periode des Resonators, dessen Resonanzfrequenz durch Variation der Stärke B_0 des vertikalen Feldes eingestellt und damit die gewünschte Wellenlänge selektiert werden. Die für die ausgewählte Resonanzwellenlänge λ_0 erforderliche Amplitude B_1 des alternierenden Feldes muss dann für die gegebene aktive Gesamtlänge L des Resonators entsprechend so gewählt werden, dass der insgesamt akkumulierte Larmor-Rotationswinkel ein ungerades Vielfaches von π beträgt.

Neben dem Selektieren der Wellenlänge soll der Resonator im gepulsten Modus auch als Chopper zur Erzeugung kurzer, periodischer Neutronenpulse genutzt werden. Dies kann einfach durch gezieltes Ein- und Ausschalten des transversalen Magnetfeldes erfolgen, da im Idealfall ohne den nur durch die Wechselwirkung mit diesem Feld möglichen resonanten Spinflip ja überhaupt keine Neutronen zum Detektor gelangen können.

Wird dabei der gesamte Resonator auf einmal ein- bzw. ausgeschaltet, so ist die kleinstmögliche Pulsbreite eines damit gechoppten Neutronenstrahls durch $\Delta t_{min} = L/v_0$ definiert, wobei L die aktive Gesamtlänge des Resonators angibt und v_0 die Geschwindigkeit für Neutronen der Resonanzwellenlänge λ_0 [7]. Ist es allerdings möglich, jede einzelne Spule unabhängig von allen anderen ein- und auszuschalten, so kann die kleinstmögliche Pulsbreite weiter auf $\Delta t_{min}^{TW} = a/v_0$ reduziert werden, wobei a die eine Dicke einer Spule ist, also einer Halbperiode des Resonators entspricht. Dies ist einer der Gründe, warum jede Spule separat mit Strom versorgt wird, denn nur dadurch es wird es möglich, die minimal erzielbare Dauer der Neutronenpulse im Prinzip auf das Zeitintervall zu reduzieren, das während des Durchquerens einer einzigen Spule vergeht.

Dies wäre allerdings auch möglich, würde man eine einzige Stromversorgung für alle Spulen benützen und nur mittels schneller elektronischer Schalter die jeweils gewünschten Spulen ein- und auszuschalten. Der Hauptgrund,

warum jede Spule hier von einer eigenen Stromversorgung gespeist wird, liegt in der Form des Spektrums, das entsteht wenn eine Wellenlängenselektion auf diese Art vorgenommen wird. Es lässt sich nämlich einfach zeigen (siehe [8]), dass die Form des Spektrums, die durch diese Art der Wellenlängenselektion entsteht, proportional zum Quadrat der Fouriertransformierten von B_1 ist. Fließt nun derselbe Strom durch jede Resonatorspule, so entspricht dies einer Rechteckfunktion von B_1, deren Fourier-Transformierte dann eine 'sinc^2-Funktion' ($\frac{sin^2(x)}{x^2}$) ergibt. Demzufolge treten im Spektrum neben dem Hauptmaximum unerwünschte störende Nebenmaxima auf, da auch die Polarisation von Neutronen abseits der gewünschten Resonanzwellenlänge λ_0 von den Oszillationen des transversalen Magnetfeldes beeinflusst werden.

Um dies zu verhindern, kann nun statt einem rechteckigen Verlauf von B_1 ein gaußförmiger Verlauf gewählt werden. Da die Fouriertransformierte einer Gaußfunktion wieder eine Gaußverteilung ergibt, treten dabei auf Kosten einer leichten Verbreiterung keine unerwünschten Nebenmaxima mehr auf. Somit ist es erforderlich, für jede der einzelnen Resonatorspulen einen unterschiedlichen Strom so einstellen zu können, dass das Transversalfeld über den gesamten Bereich des Resonator näherungsweise einen gaußförmigen Amplitudenverlauf aufweist.

1.2 Konzept der Stromversorgung der Spulen des Wanderwellenresonators

Der MONOPOL-Wanderwellenresonator besitzt 48 Einzelwindungs-Spulen aus Aluminium mit einer Dicke in Strahlrichtung von einem Zentimeter, um das notwendige alternierende Magnetfeld zu erzeugen. Um auch für thermische Neutronen eine Wellenlängenselektion bzw. eine Wellenlängenselektion inklusive Choppens zu ermöglichen, ist es erforderlich, Ströme von bis zu 25 Ampere innerhalb weniger Mikrosekunden zu schalten und zu stabilisieren.

Die von Andrzej Pelczar, dem Leiter der Elektronikabteilung des Atominstituts, konzipierte und im Rahmen von zwei Projektarbeiten [2, 3] beschriebene und getestete Stromversorgung zeigte sich in ersten Tests in der Lage, diese Anforderungen zu erfüllen. Dabei versorgt jede Stromversorgung eine der 48 Spulen mit dem für das Magnetfeld benötigten Strom, in der Endausführung werden daher 48 solcher Stromversorgungen für den Wanderwellenresonator benötigt. Abb. 1.2 zeigt eine Prototyp-Platine einer solchen Stromversorgung.

Der schematische Aufbau der Stromversorgung ist Abb. 1.3 zu entnehmen. Die mit PS1 bzw PS2 gekennzeichneten Blöcke sind zwei programmierbare Stromquellen, der Strombereich wurde dabei in einen niedrigen (0,2-5 Ampere) und einen hohen (5-20 Ampere) aufgeteilt, und die beiden Stromquellen sind jeweils für einen dieser Bereiche zuständig. Die Trennung der

Abbildung 1.2: Die von Andrzej Pelczar entwickelte Stromversorgung für den MONOPOL-Wanderwellenresonator. Am oberen Ende der Platine sind die Bauteile der Leistungselektronik zu sehen.

Strombereiche durch die Verwendung zweier Stromquellen erlaubt eine höhere Genauigkeit sorgen, denn normalerweise ist eine Stromquelle entweder nur bei hohen Stromstärken präzise und dafür ungenau bei niedrigeren Stromstärken, oder umgekehrt ist sie präzise bei niedrigen und ungenau bei hohen. Dies lässt sich vermeiden, wenn bei niedrigen Stromstärken die eine und bei hohen die andere Stromquelle eingeschaltet wird.

Die beiden Schottkydioden sorgen dafür, dass die Stromquellen sich nicht gegenseitig beeinflussen, machen es aber dennoch möglich, bei Bedarf die Ströme von beiden Stromquellen zu addieren, um so Ströme von bis zu 25 Ampere zu erreichen.

Der realen Last (Real Load) entspricht hierbei die Resonatorspule. Würde man direkt Strom über die Spule leiten, treten Stabilisationszeiten von mehreren hundert Mikrosekunden auf, bedingt durch den nicht direkt verbesserbaren Aufbau der programmierbaren Stromquelle. Dazu würden noch unerwünschte, auf die plötzliche Energieänderung im Stromkreis zurückzuführende Effekte, wie Stromspitzen direkt nach dem Einschalten auftreten.

Um dies zu verhindern, wird der Strom mittels MOSFET-Transistoren zunächst nur über einen Widerstand mit ähnlichen elektronischen Eigenschaften wie die Resonatorspule geleitet. Dieser Widerstand wird in der Folge als *Artificial Load* bezeichnet. Erst nachdem sich der Strom in der Artifical

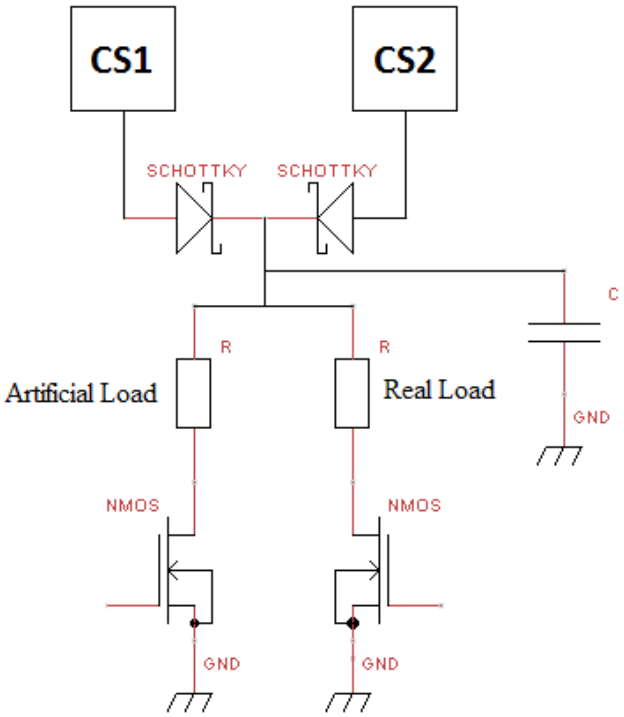

Abbildung 1.3: Schematische Darstellung der Stromversorgung des MONOPOL Wanderwellenresonators, die MOSFET-Transistoren werden extern (z.B. mittels Mikrocontroller) gesteuert.

Load stabilisiert hat, wird er mittels der beiden Transistoren über die Real Load, also die Resonatorspule, geleitet. Die unerwünschten Effekte, die normalerweise beim Einschaltvorgang unvermeidbar sind, treten hier nun nicht mehr auf und es erfolgt eine Stabilisierung des Stroms in der Resonatorspule binnen weniger Mikrosekunden, da die Gesamtenergie im System konstant bleibt. Details der mit einer Prototyp-Platine der Stromversorgung durchgeführten Tests zu diesem Schaltvorgang sind in [3] zu finden.

Da die hohen Ströme von bis zu 25 Ampere zu einer beträchtlichen Hitzeentwicklung führen, insbesondere wenn man bedenkt, dass der Wanderwellenresonator 48 dieser Stromversorgungen im Abstand von jeweils wenigen Zentimetern zueinander umfasst, wird eine Wasserkühlung benötigt, welche in [9] getestet wurde und in der Lage ist, die nötige Kühlleistung zu erbringen. Eine im Rahmen dieser Arbeit entstandene Aufnahme mit einer Wärmebildkamera ist in 1.4 zu sehen.

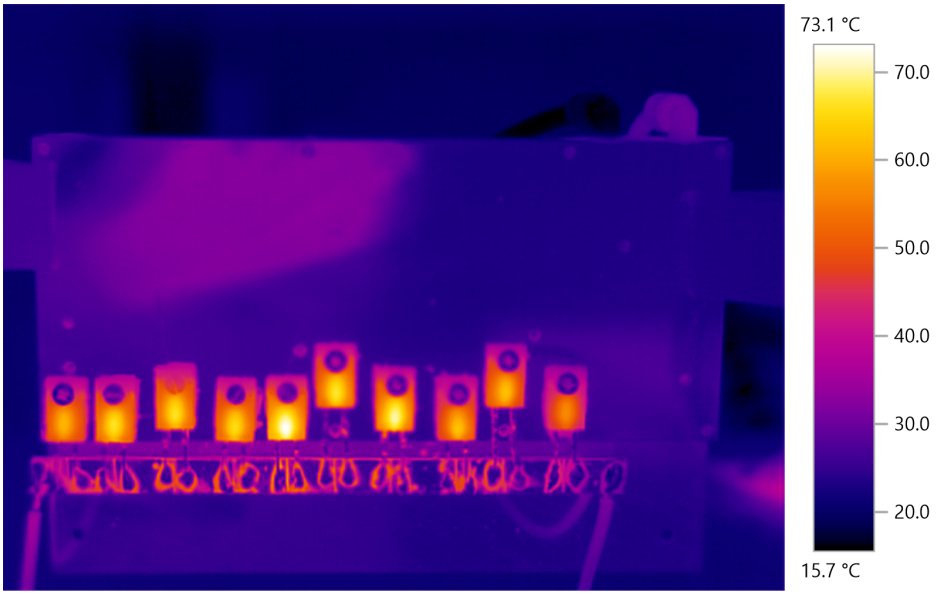

Abbildung 1.4: Aufnahme der Bauteile der Leistungelektronik mit einer Wärmebildkamera im Rahmen von Tests der Wasserkühlung.

Im Gegensatz zu dem in [9] verwendeten Prototyp des Kühlkörpers wurde für das finale Konzept der Kühlkörper eine Änderung vorgenommen. Statt Aluminium wird nun Kupfer als Material für die Kühlkörper verwendet. Dies bringt den großen Vorteil mit sich, dass Front- und Rückenplatte miteinander verlötet werden können, was die Gefahr von Wasserlecks drastisch verringert. Auch viele Bohrlöcher, welche im auf der Prototyp-Platine mit Al-Kühlkörper zur mechanischen Stabilisierung nötig waren, können nun eingespart werden, in Abb. 1.5 sind die beiden Versionen geneübergestellt. Ein Nachteil ist aber, dass die Kühlkörper aus Kupfer deutlich schwerer sind als diejenigen aus Aluminium, was für das Design eines stabilen Trägersystems für die Stromversorgungen berücksichtigt werden muss.

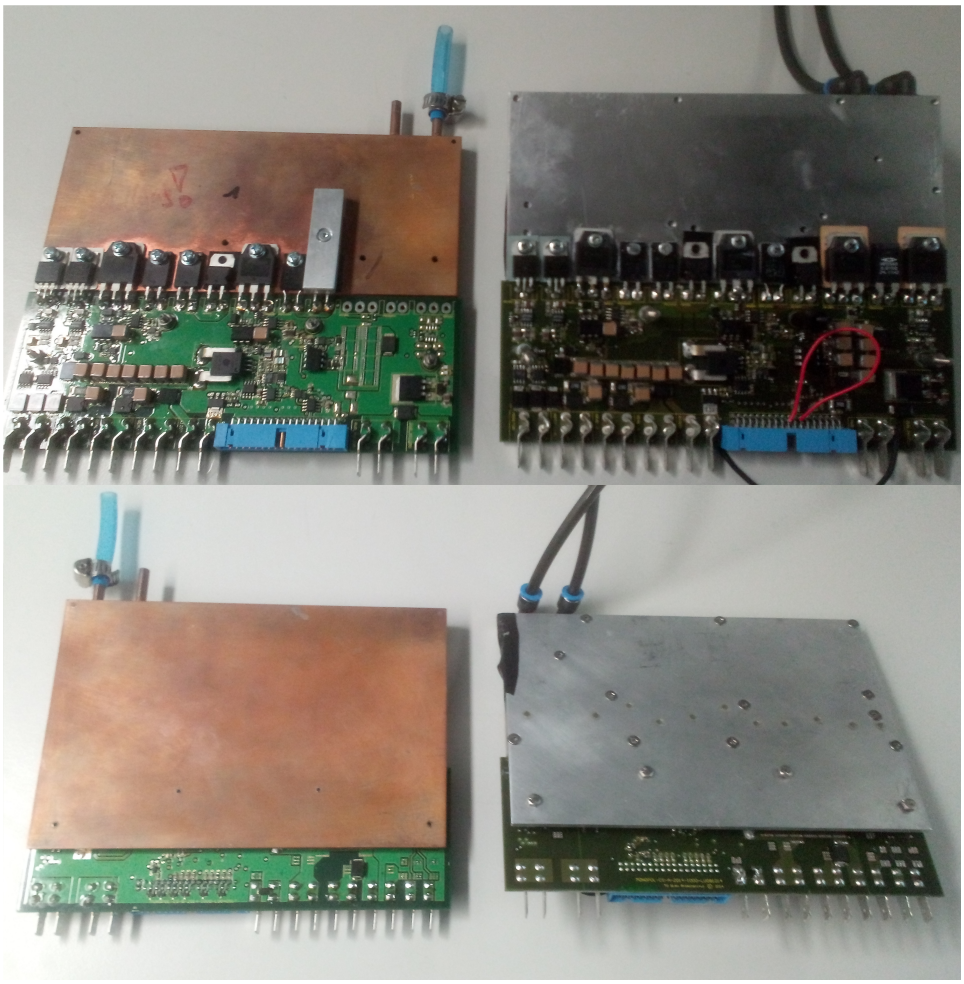

Abbildung 1.5: Zwei Stromversorgungen für den Wanderwellenresonator inklusive Kühlkörper. Links die neue Version mit Kühlkörper aus Kupfer, rechts die alte mit Aluminium. Anhand der Aufnahme der Rückseite ist gut zu erkennen, wie viele Schrauben bei der neuen Version eingespart werden können.

Kapitel 2

Schaltung der Stromversorgungen mittels Shiftregister

Neben den Stromversorgungen selbst ist auch ein Konzept notwendig, um die Spulen in genauem Timing zueinander ein- und auszuschalten. Um ein 1 cm langes Neutronenpaket des Strahls mit dem Magnetfeld quasi zu 'begleiten', soll immer genau nur diejenige Spule eingeschaltet sein, in der sich das Paket gerade befindet. Doch nicht nur dieser eine Extremfall sollte möglich sein, auch beliebige andere Muster - z.B. immer zwei Spulen aktiv, um ein 2 cm langes Paket zu begleiten – sollen erlaubt sein. Dazu ist es erforderlich, die zwei MOSFETs pro Stromversorgung simultan anzusteuern, um den Schaltvorgang wie gewünscht durchführen zu können.

Um das Magnetfeld einzustellen, aber auch um wichtige Systemparameter wie die Temperatur auszulesen, sind unterschiedliche Komponenten auf jeder Stromversorgung vorhanden, mit denen eine Kommunikation aufgenommen werden muss. Dies wird mittels eines Mikrocontrollers per I^2C-Kommunikation vorgenommen. Details zu den unterschiedlichen I^2C-Komponenten der Stromversorgung sind in [2] nachzulesen.

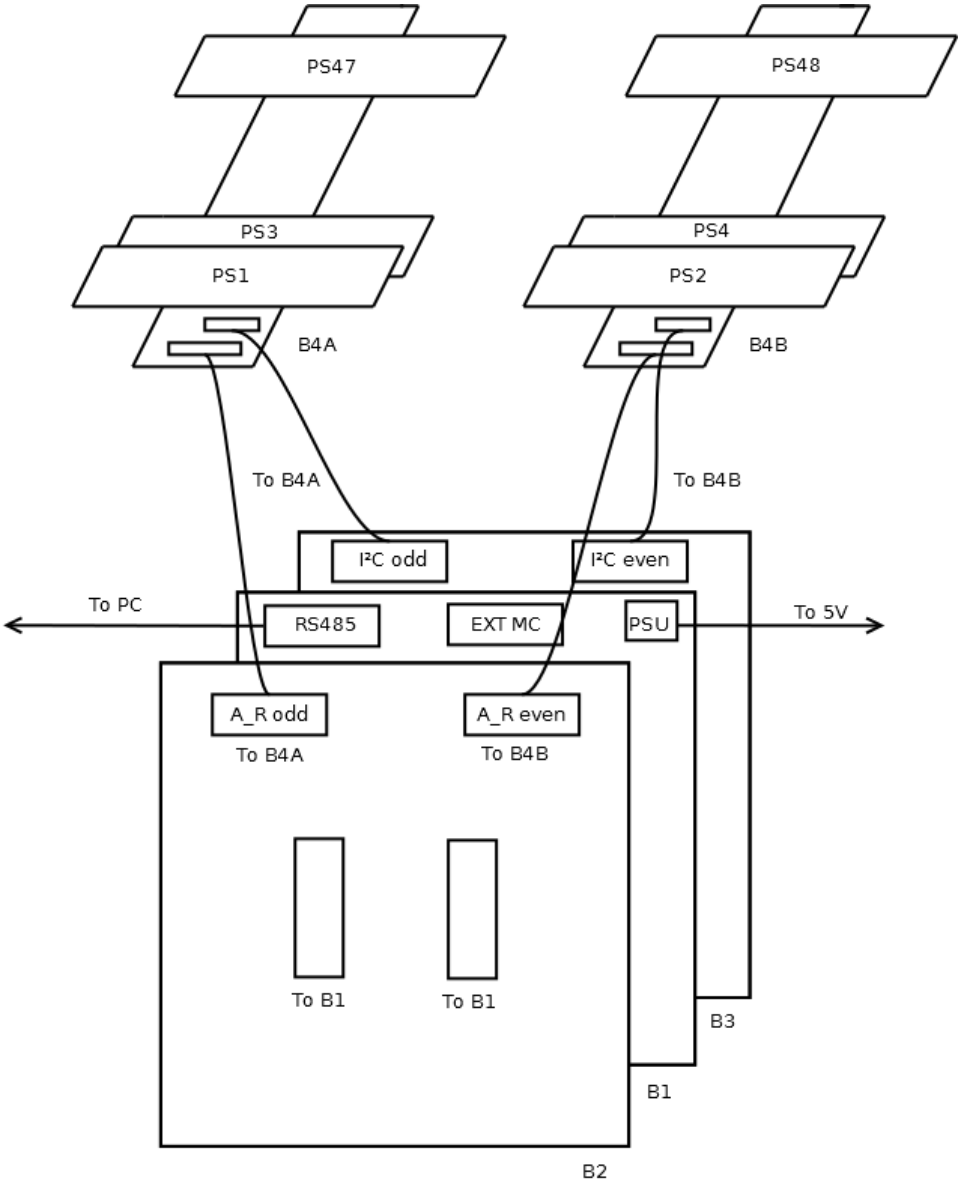

Abbildung 2.1: Die gesamte Systemarchitektur der Steuerung der Stromversorgungen des Wanderwellenresonators, welche das Resonatormagnetfeld erzeugen. B1-3 sind für die Schaltung der Stromversorgungen zuständig, über die Backplanes B4A und B4B werden die benötigten Signale zu den Stromversorgungen PS1-48 geführt.

Das Gesamtkonzept für die Steuerung ist in Abb. 2.1 zu sehen. Die drei Platinen im Vordergrund (B1, B2, B3) sind für die korrekte Schaltung der Stromversorgungen (PS1-PS48) zuständig, dabei gibt es folgende Arbeitsteilung zwischen diesen drei Platinen:

- B1, auch *Steuerplatine* genannt: Schaltet die jeweiligen Stromversorgungen im Timing entsprechend dem gewünschtem Muster mittels Shiftregistern ein und aus. Beherbergt auch den Mikrocontroller, welcher die Shiftregister auf dieser sowie auf Platine B2 ansteuert. Ebenfalls verbaut ist eine Schnittstelle für serielle Datenübertragung gemäß dem Industriestandard RS485, über den mit einem PC-Programm kommuniziert werden kann.

- B2, auch *I^2C-Platine* genannt: Verantwortlich für die korrekte I^2C-Kommunikation mit den Stromversorgungen über Shiftregister.

- B3, auch als *Verstärkerplatine* bezeichnet. Sie verstärkt die von den B1 Shiftregistern ausgegeben Signale bevor diese zu den Stromversorgungen gelangen.

Die drei Platinen sind durch unterschiedliche *Konnektoren* miteinander verbunden, eine genauere Beschreibung dieser drei Platinen ist in Unterkapitel 2.2 zu finden.

Die auf den Platinen angebrachten Shiftregister mit seriellem Eingang und parallelen Ausgängen werden realisiert, indem CPLDs (kurz für Complex Programming Logic Devices) entsprechend programmiert werden. Durch diese Shiftregister können mit nur wenigen Ausgängen des Mikrocontrollers viele Eingänge der Stromversorgungen simultan manipuliert werden, mehr dazu in Unterkapitel 2.1.

Die beiden Platinen B4A und B4B, die sogenannten *Backplanes*, sind nun dafür zuständig, die Signale der zuvor erwähnten Platinen B1-3, zu den Stromversorgungen zu befördern. Jede dieser beiden Platinen beherbergt dabei je 24 Stromversorgungen, aus Platzgründen werden diese je abwechselnd rechts und links entlang der Flugrichtung des Neutronenstrahls angeordnet, die Resonatorspulen sind dabei natürlich stets in der Mitte zwischen diesen beiden Platinen entlang dem Neutronenstrahl angeordnet. Die Backplane B4A beherbergt die ungerade nummerierten Stromversorgungen PS1,3,...,47, B4B die gerade nummerierten PS2,4,...,48. Auf diesen beiden Platinen befinden sich keine besonderen elektronischen Bauteile, sondern lediglich jeweils 26 Konnektoren, davon je einen für jede der 24 Stromversorgungen, einen weiteren, um per Kabel die Signale der I^2C-Platine empfangen zu können, sowie einen, um per Kabel eine Verbindung mit der Verstärkerplatine herzustellen, von welcher die verstärkten Signale der Steuerplatine ausgesendet werden.

2.1 Programmierung der CPLDs als Shiftregister

Wie bereits erwähnt, werden für die Platinen, welche die Stromversorgungen im gewünschten Timing schalten, Shiftregister mit seriellem Eingang und

parallelen Ausgängen benötigt. Um solche Shiftregister zu realisieren, werden Complex Programmable Logic Devices (kurz CPLD) passend programmiert.

Bei einem CPLD handelt es sich um eine programmierbare logische Schaltung, die Programmierung wird dabei in einem EEPROM (Electronically Erasable Programmable Read-only Memory) gespeichert und bleibt dadurch auch erhalten wenn keine Betriebsspannung anliegt. Die mögliche Komplexität der Schaltung ist dabei niedriger als etwa die eines FPGA (Field Programmable Gate Arryay)[1], für die Realisierung der hier benötigten Shiftregister allerdings ausreichend. Dazu geht beim FPGA die Programmierung verloren, wenn keine Betriebsspannung mehr anliegt, da diese im Gegensatz zu den CPLDs nicht in EEPROMs, sondern in anderen, flüchtigen Speicherchips, wie z.B. SRAMs (Static Random-Access Memory), gespeichert ist.

In unserem Fall werden spezielle ISP LSI2064 CPLD Chips [11] verwendet. Dabei handelt es sich um CPLDs mit einer Betriebsspannung von 5 V, einer Spannung, die deutlich höher ist als übliche Betriebsspannungen moderner CPLDs, die meist bei mindestens 1.8 V liegt. Jedoch ist diese höhere Betriebsspannung durchaus erwünscht, da die CPLDs dadurch deutlich robuster gegenüber elektrisch induzierten Störungen sind als vergleichbare Produkte mit niedrigerer Betriebsspannung.

Alle benötigten CPLDs werden als 16-bit Shiftregister mit seriellem Eingang und 16 parallelen Ausgängen für die Daten programmiert. Es gibt aber zwei leicht unterschiedliche Schaltpläne, da die Platine, die für die Schaltung der Stromversorgungen zuständig ist, für einige der CPLDs eine modifizierte Beschaltung benötigt, dies wird später in diesem Kapitel noch näher erläutert.

Die CPLDs können mittels der Software *ispLever Classic Project Navigator*, welche kostenlos auf der Website der Lattice Semiconductor Corporation heruntergeladen werden kann, sowie einem passendem Programmierkabel (im konkreten Fall das Modell HW-DLN-3C [12] programmiert werden. Dieses Programmierkabel wird dazu an einen 25-Pin PC-Port (ein ehemals üblicher Weg, um Drucker anzuschließen) des Computers angeschlossen. Dabei muss aber darauf geachtet werden, dass sich der PC-Port nicht im Modus für Drucker, sondern in dem für Peripherie befindet.

Mittels einer kleinen, in Abb. 2.2 gezeigten Platine, auf der drei Konnektor-Stecker und eine Halterung für den CPLD Chip montiert war, konnte dieser nun mittels Programmierkabel als Schiebregister programmiert werden. Ein Konnektorren ist für den Anschluss des Programmierkabels erforderlich und zwei weitere für das Testen eines bereits programmierten CPLDs. So ist einer dieser beiden Konnektor mit den Eingängen des CPLDs verbunden, über

[1] In [10] wird die Möglichkeit einer Schaltung der Stromversorgungen mittels FPGA behandelt.

welche etwa mit Hilfe eines Mikrocontrollers der Ablauf des Durchschiebens durch das Shiftregister gesteuert werden kann. Der dritte Konnektor wiederum ist mit den 16 Ausgängen des CLPD Chips verbunden, sodass mit einem Oszilloskop oder einem Logik Analysators die Ausgänge dahingehend überprüft werden können, ob das Shiftregister auch tatsächlich wie geplant funktioniert.

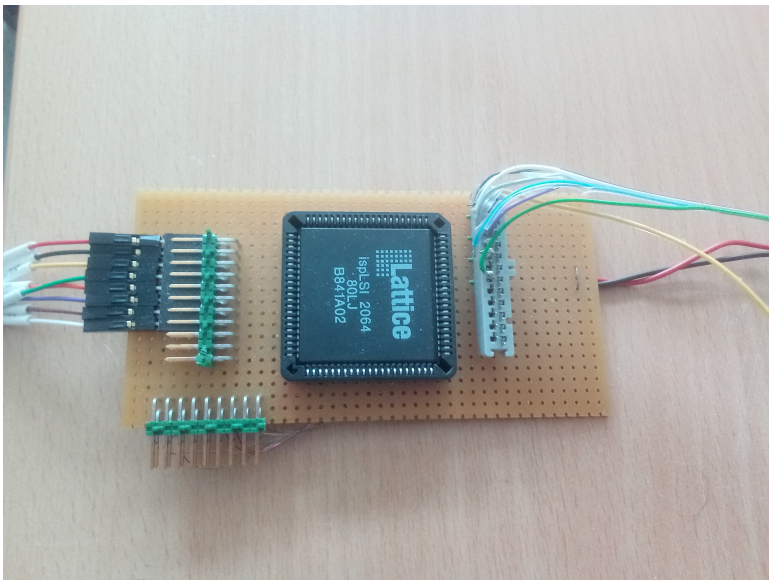

Abbildung 2.2: Die Platine über welche der CPLD programmiert werden kann: Links ist der Konnektor für das Programmierkabel zu sehen, rechts ein mit den Eingängen des CPLDs verbundener, sowie unten ein mit den Ausgängen des CPLD verbundener Konnektor. Mit Hilfe eines Oszilloskops oder eines Logik-Analysators lässt sich damit die korrekte Funktion des Shiftregisters unmittelbar nach erfolgter Programmierung des CLPD Chips überprüfen.

Um nun die gewünschte Schaltung des CPLDs zu programmieren, wird der im *ispLever Classic Project Navigator* enthaltene *Schematic Editor* verwendet, dabei handelt es sich um eine graphische Programmieroberfläche. Im Schematic Editor können unterschiedliche Schaltungen in den CPLD einprogrammiert werden, darunter logische Verknüpfungen (AND, OR, XOR....), Zähler, Flip-Flops, Multiplexer, oder auch arithmetische Verknüpfungen. Einen Überblick über die zur Verfügung stehenden Möglichkeiten gibt es in [13, 14].

Wie bereits erwähnt, werden für dieses Projekt zwei leicht unterschiedliche Programmierungen benötigt, eine davon ist in Abb. 2.3 zu sehen. Jedes der Shiftregister besteht aus 16 baugleichen Zellen. Zunächst soll anhand von Abb. 2.4 die Funktionsweise einer Zelle erklärt werden.

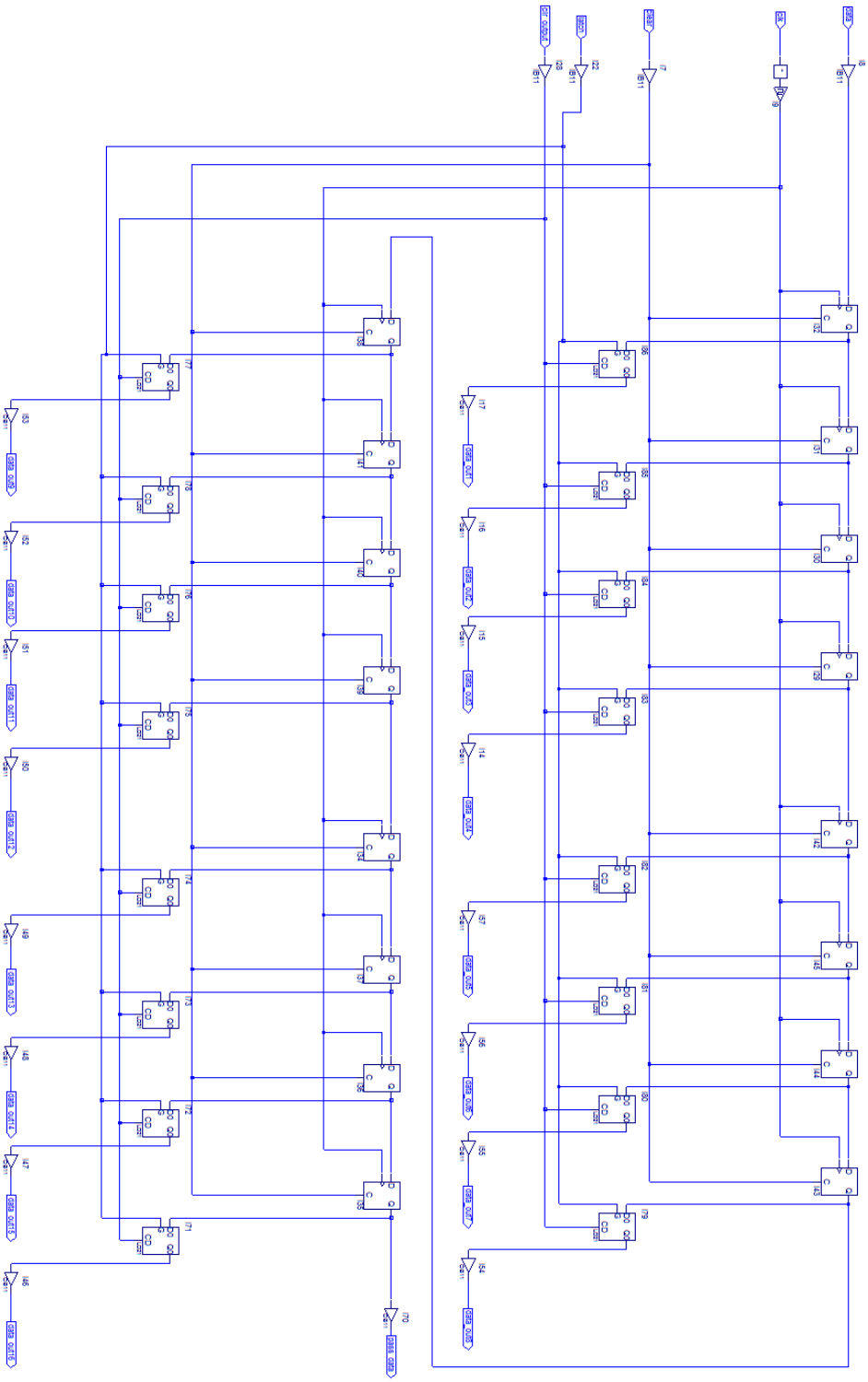

Abbildung 2.3: Der gesamte Schaltplan eines CPLD, das mit dem Schematic Editor als 16-Bit Shiftregister programmiert wurde.

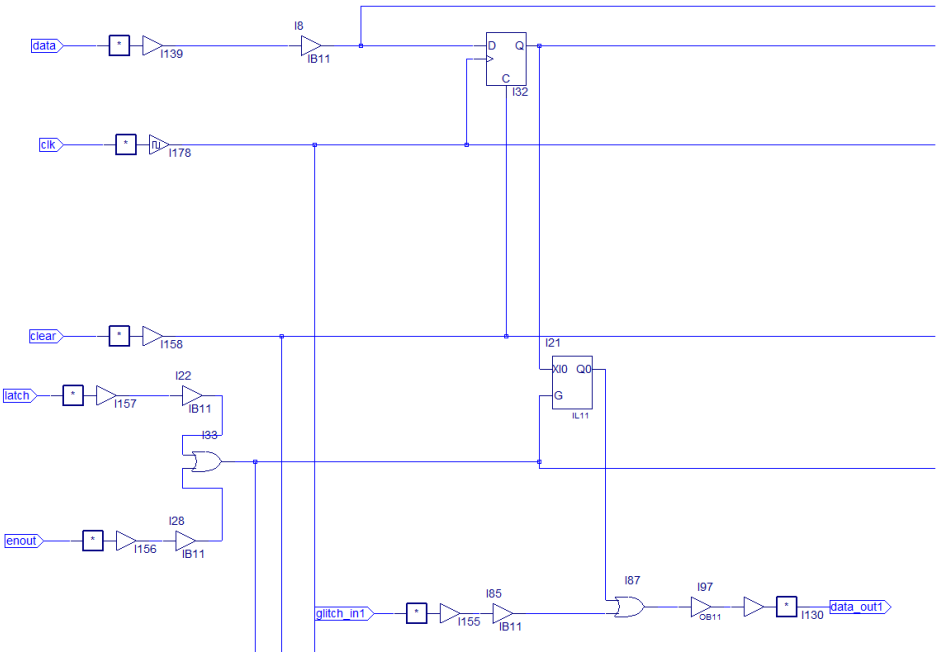

Abbildung 2.4: Eine von 16 Zellen des CPLD-Shiftregisters in der Ansicht des Schematic Editors des *ispLever Project Navigators*. Auf der linken Seite sind die Eingänge zu sehen, welche extern angesteuert werden müssen, um den Ablauf des Durchschiebens zu steuern.

Bei dem mit dem Marker I32 versehenen Bauteil handelt es sich um ein D-Flipflop, also um ein Bauteil, welches das am Dateneingang D anliegende Signal solange zwischenspeichert, bis am Eingang des Taktgebers (markiert durch ein Dreieck) ein Wechsel von 0 auf 1 erfolgt, woraufhin am Ausgang Q das zu diesem Zeitpunkt an Eingang D anliegende Signal ausgegeben wird. Dies gilt aber nur für den Fall, dass am Eingang C (Clear) keine Spannung anliegt, sollte hier Spannung anliegen, bleibt Ausgang Q immer auf 0. Auf diese Weise kann ein beliebiges Muster mit dem richtigen Timing weitergegeben werden.

Das Bauteil mit dem Marker IL11 ist ein sogenannter *latched Input* ('verriegelter Eingang'), dabei entscheidet das am Eingang G anliegende Signal, was am Ausgang Q0 ausgegeben wird. Liegt an G eine Spannung an, so wird das am Eingang XI0 anliegende Signal am Ausgang Q0 ausgegeben, ansonsten wird dasjenige Signal ausgegeben, welches am Eingang XI0 war als das letzte Mal Spannung an G anlag.

Die Shiftregister funktionieren nun folgenderweise: Das gewünschte Muster wird über den Eingang *data* Bit für Bit an den Eingang D des ersten

D-Flipflops gesendet. Bevor ein neues Bit gesendet wird, wird an jeden D-Flipflop parallel eine steigende Flanke des Taktgebers über den Eingang *clk* gesendet. Da mit jeder steigenden Flanke das anliegende Signal um einen Flipflop weitergeschoben wird, kann eine bestimmte Abfolge an Signalen, die zunächst seriell an den ersten D-Flipflop gesendet wurde, nach entsprechend vielen Takten parallel an den Q-Ausgängen der Flipflops ausgegeben werden

Sobald nun Spannung an den G-Eingängen der latched Inputs anliegt, wird dieses Signal schließlich an die Outputs der CPLDs übergeben. Auf diese Weise sind zwei Betriebsmodi möglich:

- Wird von Anfang an Spannung an die G-Eingänge angelegt, wird das gewünschte Muster bereits beim ersten Takt des Taktgebers an den Ausgängen ausgegeben. (transparenter Modus)
- Es wird zunächst ein gewünschtes Startmuster eingespeist und danach erst Spannung an die G-Eingänge angelegt, woraufhin das schon in den Flipflops gespeicherte Muster unmittelbar an die Ausgänge des CPLDs ausgegeben wird bevor es weitergeschoben wird. (latched Modus)

Zudem ist jeder Output des CPLD noch mit einer logischen OR-Verknüpfung mit den als *glitchin* bezeichneten Inputs verbunden. Wenn an einem solchen *glitchin*-Input Spannung anliegt wird folglich der Ausgang des CPLD auf HIGH gesetzt, egal in welchem Zustand sich der Ausgang des latched Inputs befindet. Dies ist wichtig für den Ablauf des Umschaltvorgangs zwischen Artificial und Real Load und wird genauer in Abschnitt 2.2.1 besprochen.

Wie in Abb. 2.4 zu sehen ist, gibt es ebenfalls eine logische ODER-Verknüpfung zwischen den Inputs *enout* und *latch*, die beide mit dem G-Eingang des latched Input verbunden sind.

So soll der Eingang *enout* dafür zuständig sein, den transparenten Modus zu aktivieren, dieser ist dabei ständig eingeschalten, der Eingang *latch* ist dagegen für den latched Modus zuständig und wird nur bei Bedarf auf HIGH gesetzt, wenn das voreingestellte Muster zu den Ausgängen des Shiftregisters gelangen soll.

Diese Inputs sind beide mit den Eingängen G aller latched Inputs verbunden, da üblicherweise vom Mikrocontroller nur einer der beiden Eingänge - je nach Modus - auf HIGH geschalten wird, sorgt das logische Oder dafür, dass das aktive Signal auch seinen Weg zu den G-Eingängen der latched Inputs findet, wenn der andere Eingang LOW ist.

In Abb. 2.5 ist der Schaltplan für die zweite Programmierung der CPLDs zu sehen, welche neben der vorher besprochenen Programmierung für die Steuerplatine gebraucht wird. Statt den latched Inputs der zuvor beschriebenen Schaltung (Kürzel IL11) kommen hier Bauteile mit dem Kürzel LD21

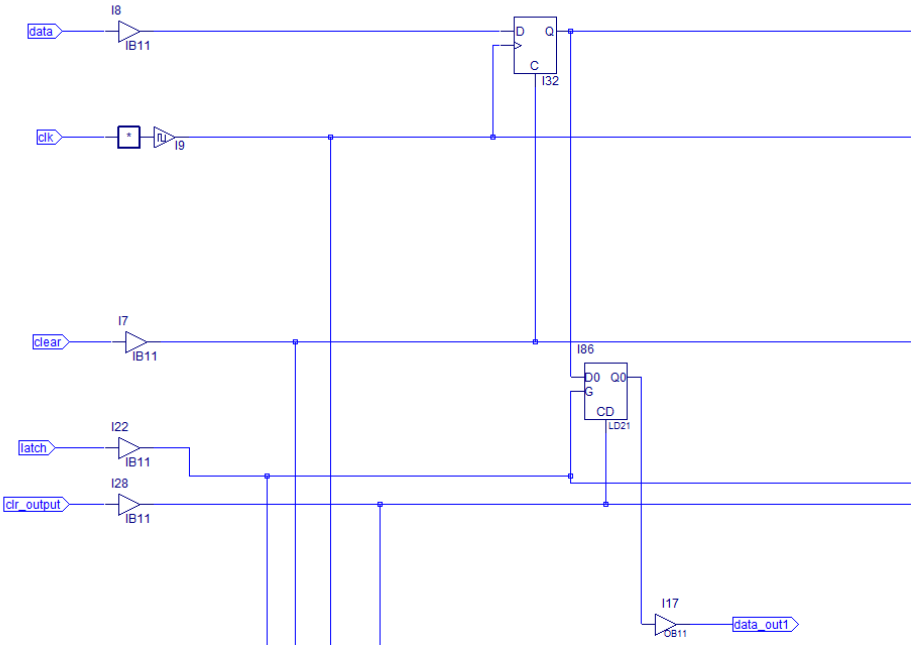

Abbildung 2.5: Eine der 16 Zellen der zweiten Art der Programmierung der CPLD-Shiftregister in der Ansicht des Schematic Editors des *ispLever Project Navigators*.

zum Einsatz, welche zusätzlich noch einen mit CD bezeichneten Extraeingang besitzen. Wird an CD Spannung angelegt, wird der Ausgang Q0 automatisch auf 0 gesetzt.

Zudem wurde der Eingang *enout* hier eingespart und durch *clroutput* ersetzt, welcher wiederum mit allen CD-Eingängen der latched Inputs verbunden ist und so dazu verwendet werden kann, alle Ausgänge gleichzeitig auf 0 zu setzen. Bis auf diese Unterschiede ist die Funktionsweise der beiden Shiftregister aber ident.

Wie die hier besprochenen Shiftregister dann genau eingesetzt werden, wird in Abschnitt 2.2.1 erklärt.

2.2 Die unterschiedlichen Platinen zur Steuerung der Stromversorgungen

In diesem Abschnitt sollen nun die einzelnen Platinen besprochen werden, welche benötigt werden, um die Stromversorgungen im gewünschten Timing zu schalten, eine I^2C-Kommunikation für die Einstellung der gewünschten Stromstärke vorzunehmen, oder um verschiedene Messdaten der Kompo-

nenten auszulesen. Entworfen wurden diese Platinen mit dem *Altium Designer Version 17*, unter Anleitung von Andrzej Pelczar. Mithilfe des Altium Designers können Platinen zunächst schematisch konzipiert werden, siehe Abb. 2.6. Sobald dann alle benötigten Komponenten schematisch korrekt angeordnet sind, kann ein tatsächliches Layout der Platine angefertigt werden, wie es in Abb. 2.7 zu sehen ist.

So können die benötigten Komponenten zunächst auf der Ober- und Unterseite der Platine angebracht werden und dann, je nach Komplexität der benötigten Verbindungen, die gewünschte Anzahl an Zwischenschichten eingestellt werden. In diesen Zwischenschichten können die benötigten Leitungen zwischen den einzelnen Komponenten dann verlegt werden, sie werden daher auch als 'Signalschichten' bezeichnet.

All die im Folgenden besprochenen Platinen besitzen dabei vier solcher Signalschichten, die ihrerseits mittels nichtleitender dielektrischer Schichten voneinander und von der Ober- und Unterseite isoliert werden. Benötigt man eine leitende Verbindung zwischen den Schichten benötigten, können sogenannte 'Vias' angebracht werden, die durch alle Schichten der Platine leiten.

Die Signalschichten wurden dabei in zwei Arten unterteilt: Betrachtet man die Platine mit Blick auf die Oberseite, so enthalten die Signalschichten 1 und 3 beinahe ausschließlich Verbindungen von oben nach unten, die Signalleitungen 2 und 4 dagegen beinahe nur Verbindungen von links nach rechts. Auf diese Weise soll einer eventuellen elektromagnetischen Kopplung zweier angrenzender Schichten vorgebeugt werden. Mittels der vorher besprochenen Vias können dabei beliebige Wege auf der Platine realisiert werden, die horizontalen bzw vertikalen Teilwege der Leitungen werden dabei auf den jeweiligen Schichten realisiert und mittels Via miteinander verbunden.

In den folgenden Abschnitten sollen die Platinen nun ihrer Aufgabe nach beschrieben werden. Sie sind so konzipiert, dass sie mittels passender Konnektoren direkt übereinander gesteckt werden können, wie in Abb. 2.8 gezeigt ist.

2.2.1 Die Steuerplatine

Diese Platine ist dafür verantwortlich, dass die Spulen mit dem korrekten Timing zueinander geschalten werden. Dazu werden insgesamt neun der CPLDs angebracht, von denen sechs wie in Abb. 2.4 und drei wie in Abb. 2.5 programmiert sind. Diese Platine benötigt eine Betriebsspannung von 5 V.

Die Eingänge der CPLDs werden dabei von einem 16-Bit PIC24FV32KA304 Mikrocontroller [15] angesteuert, der von einem ebenfalls auf der Platine verbauten 29.4912 MHz Oszillator [16] getaktet wird. Dies führt zu einer Zeit pro Takt des Oszillators von 33.91 ns. Da der Mikrocontroller zwei Takte pro Instruktion benötigt, führt dies wiederum zu einer Zeit pro Instruktion

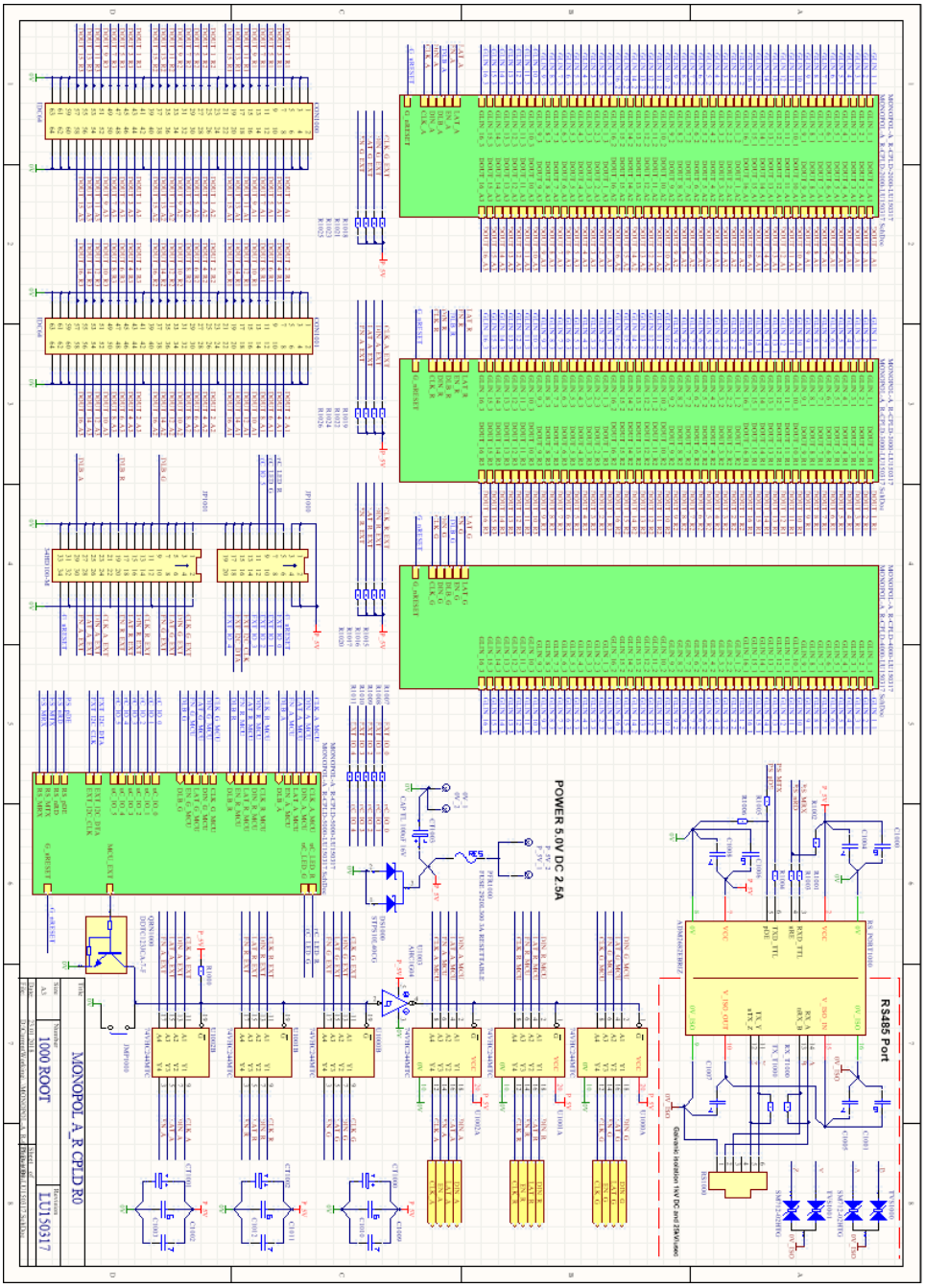

Abbildung 2.6: Der Schaltplan der Steuerplatine für die Stromversorgungen in der schematischen Ansicht des *Altium Designers*. Die grünen Blöcke im Bild sind eigens erstellte Schaltblöcke, die sich hierarchisch unter dem hier abgebildeten befinden.

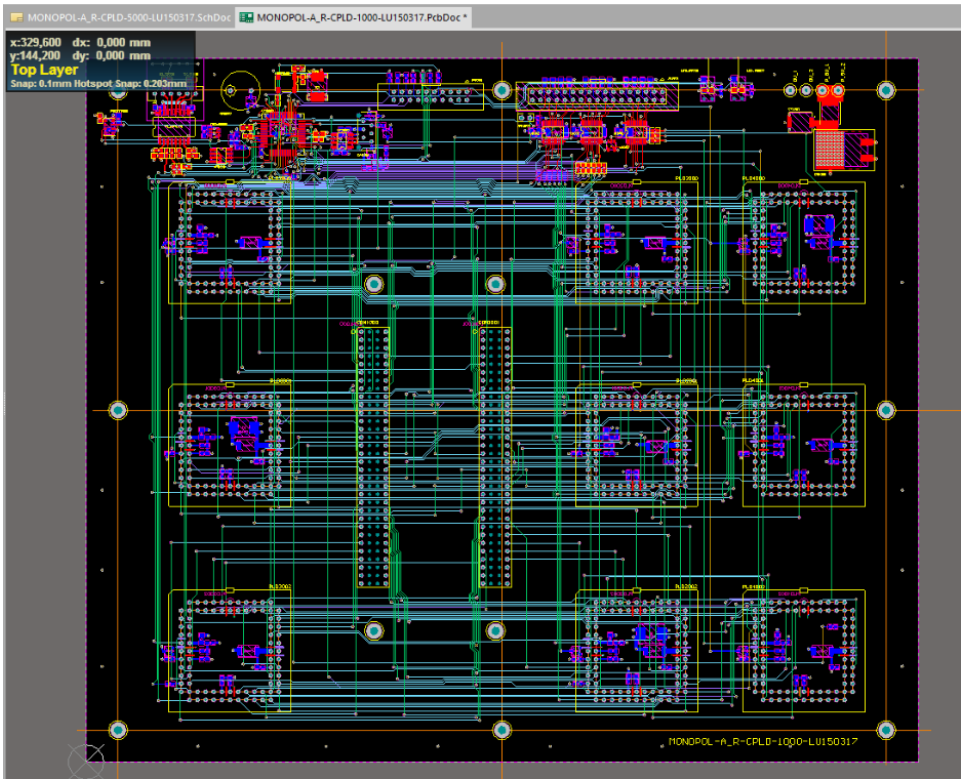

Abbildung 2.7: Das Layout der Steuerplatine in der 2D-Ansicht des *Altium Designers*, die unterschiedlichen Farben der leitenden Verbindungen zeigen an in welcher Schicht der Platine sich die jeweilige Leitung befindet.

von 67.82 ns. Dieser Mikrocontroller steuert auch die in Abschnitt 2.2.2 beschriebene Platine für die I^2C-Kommunikation und deren Shiftregister an. Dazu befindet sich auf der Steuerplatine ein Konnektor, welcher mit einem dazu passenden Gegenstück auf der I^2C-Platine verbunden wird. Ein weiterer Konnektor ist für den ICD 3 Debugger [17] notwendig, mit welchem der Mikrocontroller programmiert und die Programmierung auf Fehler untersucht werden kann ('*debuggen*'). Zusätzlich ist eine Verbindung für das UART (Universal Asynchronous Receiver Transmitter)-Modul des Mikrocontrollers vorhanden, mit der dieser mittels seriellem Port mit einem PC kommunizieren kann.

Außerdem sind sowohl ein Temperatursensor als auch ein Drucksensor eingebaut, welche beide mittels I^2C-Kommunikation vom Mikrocontroller ausgelesen werden können. Während der Temperatursensor für eine allgemeine Kontrolle der Hitzeentwicklung zuständig ist, ist der Drucksensor nur vonnöten, um auf ein etwaiges Leck hinweisen zu können, sollte der Wanderwellenresonator in Heliumumgebung benützt werden.

Weiters gibt es zwei Konnektoren, welche die Ausgänge der CPLD-Shift-

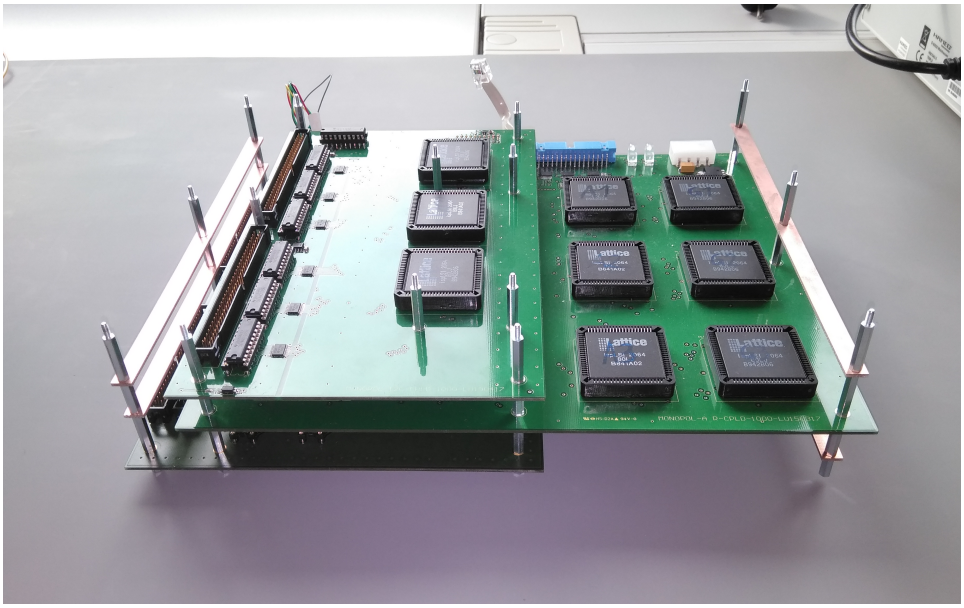

Abbildung 2.8: Foto der drei übereinander gestecken Platinen zur Kontrolle der MONOPOL Stromversorgungen. Oben befindet sich die Platine für die I^2C-Kommunikation, in der Mitte jene zur Ansteuerung der Stromversorgungen und der Regelung des Schaltablaufs der Resonatorspulen, unten die Verstärkerplatine, welche die ausgehenden Signale verstärkt und stabilisiert.

register mit den Stromversorgungen verbinden, einen für die ungerade nummerierten Stromversorgungen, sowie einen zweiten für die gerade nummerierten. Dies ist notwendig, da die Halterung für die Stromversorgungen aus zwei Teilen besteht, die sich auf einander gegenüber liegenden Seiten des Resonators befinden, wie schon in Abb. 2.1 angedeutet.

Steuerung der Stromversorgungen mittels Steuerplatine

Die Funktionsweise der Stromversorgung von Andrzej Pelczar für die Spulen des MONOPOL-Wanderwellenresonators ist in [2],[3] beschrieben. Sie basiert darauf, dass der gewünschte Strom zunächst über ein der Resonatorspule, auch *Real Load* genannt, ähnliches Bauteil geleitet wird, der sogenannten *Artificial Load*.

Die unerwünschten Effekte, wie Stromspitzen weit über dem eingestellten Strom sowie für dieses Projekt viel zu lange Stabilisierungszeiten von mehreren 100 μs, treten dann auf diesem Bauteil auf. Erst wenn sich der Strom auf dem Artificial Load stabilisiert hat, wird mittels MOSFET-Transistoren auf die Resonatorspulen umgeschalten.

Die in [3] durchgeführten Tests zeigen, dass mit dieser Methode Schaltzei-

ten im einstelligen Mikrosekundenbereich ohne unerwünschte Stromspitzen möglich sind. Die besten Ergebnisse ließen sich dabei erzielen, wenn beim ersten Schaltvorgang für etwa 600 ns sowohl Artificial als auch Real Load gleichzeitig mit Strom versorgt werden, siehe auch Abb. 2.9.

Abbildung 2.9: Der Schaltvorgang von *Artificial* auf *Real Load* mit ca. 600 ns Zeitspanne, in der über beide Loads Strom geleitet wird. Blau: Signallinie für die Artificial Load. Gelb: Signallinie für die Real Load. Grün: Stromverlauf über die Real Load.

Soll kein Strom mehr durch die jeweilige Resonatorspule fließen, wird von Real auf Artificial Load umgeschaltet, dies verkürzt auch die Abschaltzeit drastisch, was für das korrekte Mitbegleiten eines Neutronenpaketes über die Resonatorlänge ebenfalls sehr wichtig ist. Auch in weiterer Folge wird nur noch zwischen Artificial und Real Load hin- und hergeschaltet, wobei jeder dieser Schaltvorgänge nur mehr wenige Mikrosekunden dauert und die lange Stabilisierungszeit von mehreren hundert Mikrosekunden nur beim ersten Einschalten abgewartet werden muss.

Um diesen Schaltvorgang wie gewünscht zu ermöglichen, werden die neun CPLDs in drei Dreierblöcken angeordnet, wie in Abb. 2.10 verdeutlicht. Die Ausgänge der CPLDs lassen den Strom entweder über den Real oder Artificial Load fließen, bei dieser Darstellung sind dies die Ausgänge an den Seiten, R_E1-3 und R_O1-3 schalten dabei auf den Real Load, A_E1-3 und A_O1-3 auf den Artificial Load.

Die Unterteilung der Ausgänge in R_E und R_O bzw. A_E und A_O ist dem Aufbau der Halterung für die 48 Stromversorgungen geschuldet. Die Stromversorgungen sind dabei auf zwei Platinen, den Backplanes, angebracht, wobei abwechselnd eine Stromversorgung sich auf der einen und die nächste auf der anderen Seite befindet. Das führt dazu, dass beim Durchzählen eine Schiene die ungeraden (englisch *odd*, daher R_O, A_O), und

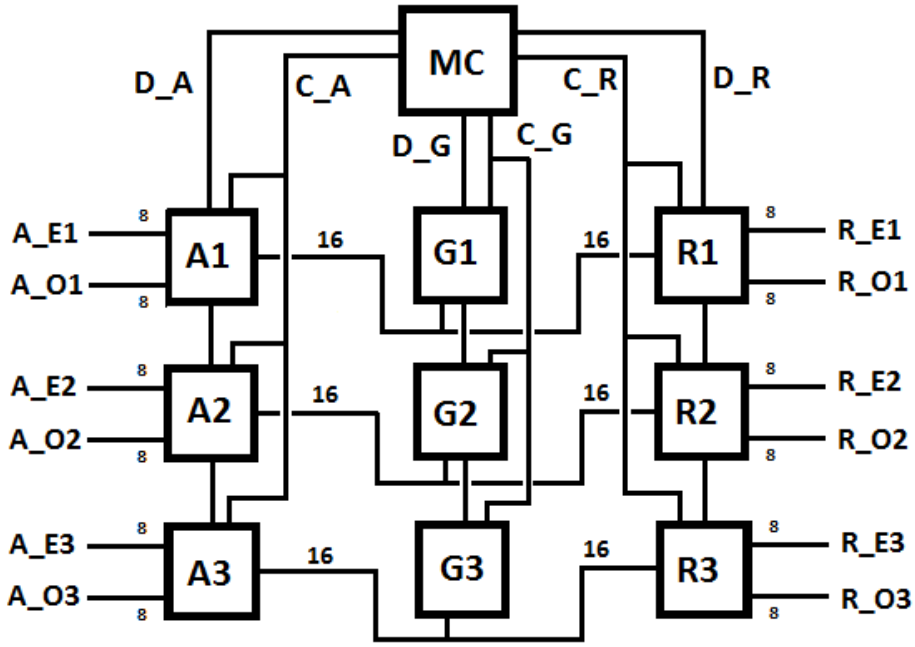

Abbildung 2.10: Schematische Darstellung der neun CPLDs der Steuerplatine (A1-3,R1-3,G1-3) angesteuert durch den 16-bit Mikrocontroller (MC). Die Nummern über den Datenlinien bezeichnen deren Breite in Bit, ohne Nummer ist eine Datenlinie nur ein Bit breit.

die gegenüberliegende die geraden (englisch *even*, daher R_E, A_E) Stromquellen hält.

Der Mikrocontroller fungiert hier einerseits als Taktgeber für drei Blöcke, andererseits versorgt er den seriellen Input der Shiftregister mit dem gewünschten Muster. Der Mikrocontroller gibt hierbei entweder 5 V, was als logisch HIGH interpretiert wird, oder 0 Volt, was logisch LOW entspricht, aus. Da allerdings aufgrund der unterschiedlichen Bauteile in elektronischen Schaltungen aus unterschiedlichsten Gründen Spannungsschwankungen auftreten können, wird schon ein Anstieg über 2.1 V als Wechsel von logisch LOW zu HIGH, und umgekehrt ein Spannungsabfall unter 0.8 V als ein Wechsel zu logisch LOW interpretiert.

Wie in Abb. 2.10 zu sehen ist, wird nur das erste der drei Shiftregister pro Block direkt vom Mikrocontroller mit Daten versorgt (Linien D_G, D_A, D_R), während der Takt parallel für jedes der drei Shiftregister verläuft (Linien C_G,C_A,C_R). Die zweiten Shiftregister jedes Blocks erhalten als Input die Daten, welche aus dem jeweils ersten Block herausgeschoben wurden, ebenso erhalten die dritten Shiftregister die aus dem zweiten Register geschobenen Daten.

Durch diese Schaltung wird aus drei 16-Bit Shiftregistern mit seriellem Input und parallelem Output ein 48-Bit Shiftregister mit seriellem Input und 48 parallelen Outputs. Die 48 Ausgänge des R-Blocks entscheiden dabei, ob der Strom über die Real Load fließt, die des A-Blocks hingegen, ob er über die Artificial Load fließt.

Der G-Block, das sogenannte *Glitch*-Shiftregister, ist nun für den Schaltvorgang von Artificial zu Real Load oder umgekehrt notwendig. Wie bereits erwähnt, soll beim Schaltvorgang für kurze Zeit der Strom über beide Loads geleitet werden, in Abb. 2.10 ist zu erkennen, dass die Outputs des *Glitch*-Shiftregisters sowohl mit dem R-Block als auch mit dem A-Block verbunden sind. Wie bereits in Unterkapitel 2.1 beschrieben, ist jeder Ausgang der Shiftregister, welche Real und Artificial Load ansteuern, über ein logisches ODER mit einem Eingang verbunden. Diese Eingänge werden wie in Abb. 2.11 gezeigt nun mit den Ausgängen des Glitch-Shiftregisters verbunden. Folglich wird der Strom sowohl über die Real als auch über die Artificial Load geleitet, wenn der dazugehörige Ausgang des Glitch-Shiftregisters unter Spannung gesetzt wird. Dieser Ausgang wird genau dann auf HIGH geschaltet, wenn ein Schaltvorgang im Gange ist, um zu ermöglichen, dass für eine Zeit von 600 ns Strom über beide Loads geleitet wird. Wie das genaue Timing aussieht, um so einen Schaltvorgang vornehmen zu können, wird in Abschnitt 3.3.1 besprochen.

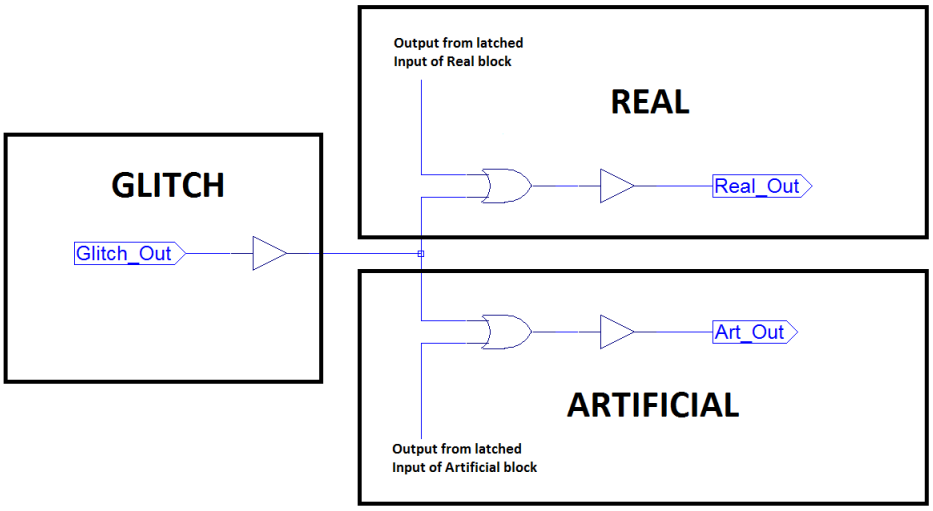

Abbildung 2.11: Logische Verknüpfung eines Ausganges des Glitch-Shiftregisters mit den Ausgängen der Artificial/Real-Shiftregister. Wenn der Ausgang Glitch_Out auf HIGH gesetzt wird, werden über das logische ODER auch die Ausgänge Real_Out und Art_Out auf HIGH gesetzt.

Da es sich um einen 16-Bit Mikrocontroller handelt, können theoretisch bis zu 16 Ausgänge des Mikrocontrollers simultan verändert werden, im vorliegenden diesem Fall soll es zumindest möglich sein, sechs Ausgänge simultan zu verändern, die Taktgeber und den Dateninput für die drei verschiedenen 48-Bit-Shiftregister.

Neben den Eingängen der CPLD-Shiftregister steuert der Mikrocontroller auch einige andere Eingänge der Stromversorgungen an, dies wird genauer in Kapitel 3 und insbesondere in Unterkapitel 3.1 besprochen.

2.2.2 Platine für die I²C-Kommunikation

Diese Platine sorgt dafür, dass die I²C-Kommunikation mit den einzelnen Stromversorgungen korrekt abläuft. Jede der 48 Stromversorgungen besitzt dabei folgende Komponenten, welche mittels I²C-Kommunikation entweder beschrieben oder ausgelesen werden können:

- Zwei **Digital-to-Analog-Converter** (kurz DAC): Durch Senden einer Bytefolge kann ein bestimmter Strom eingestellt werden, ein DAC ist dabei für die niedrigeren Ströme bis 5 Ampere zuständig, der andere für die höheren bis zu 20 Ampere. Beide können auch gleichzeitig zugeschaltet werden, um Ströme von bis zu 25 Ampere zu erreichen.
- Zwei **Analog-to-Digital-Converter** (kurz ADC): Ein ADC ist zuständig für die Messung der Spannung an der Resonatorspule, der andere misst über die Hallspannung eines einegebauten Hallsensors den über die Spule fließenden Strom. Mittels dieser Daten kann überwacht werden, ob die Spulen leitend mit der Stromversorgung verbunden sind, denn ein Anstieg der Spannung bei gleichzeitigem Abfall des Stroms würde auf einen mangelhaften Kontakt hinweisen.
- Zwei **Temperatursensoren**: Einer der Sensoren ist nahe an den Bauteilen angebracht, welche direkt auf dem Kühlkörper der Stromversorgung liegen, um die Hitzeentwicklung dort zu überwachen. Der andere ist weiter mittig angebracht, um die allgemeine Umgebungstemperatur zu messen.
- Ein **FRAM (Ferroelectric Random Access Memory)**: Ein nichtflüchtiger Datenspeicher, auf dem Kalibrationsdaten für die Stromversorgungen und Ähnliches gespeichert werden können.

Um diese unterschiedlichen I²C-Komponenten voneinander unterscheiden zu können, wird bei einer I²C-Kommunikation zunächst vom Master, in diesem Fall der Mikrocontroller, ein Adressbyte gesendet. Stimmt dieses Adressbyte mit der Adresse der I²C-Komponente überein, so besetzt diese den Datenbus und der Master kann eine Kommunikation mit dieser Komponente vornehmen.

Die oben erwähnten I^2C-Komponenten besitzen jeder eine eigene Adresse, sodass mit dem Senden des Adressbytes gewährleistet wird, dass der Master immer nur mit einer der Komponenten kommuniziert.

Allerdings gibt es jede dieser I^2C-Komponenten auf jeder der 48 Stromversorgungen. Würde man also beispielsweise das Adressbyte für den ersten Temperatursensor auf einmal an alle Komponenten schicken, würden sich 48 Temperatursensoren auf einmal melden und den Datenbus besetzen wollen. Um dies zu verhindern, ist diese I^2C-Platine konzipiert worden, die dafür sorgt, dass immer nur die Stromversorgung, mit der ein Kommunikationswunsch besteht, überhaupt die Möglichkeit hat, den Datenbus zu besetzen.

Um dies zu verwirklichen, werden auch hier die CPLD-Shiftregister benützt, siehe Abb. 2.12. Für diese Platine werden nur drei der 16-Bit Shiftregister benötigt, die wieder so miteinander verbunden werden, dass sie wie ein 48-Bit-Shiftregister mit seriellem Eingang und parallelem Ausgang funktionieren. Die Programmierung erfolgt dabei so wie schon in Abb. 2.5 gezeigt. Dabei werden nur zwei der Ausgänge des Mikrocontrollers (MC) benötigt, einer um die Daten zu übermitteln, die durch das Schieberegister geschoben werden sollen (D_I2C), sowie einer als Taktgeber für das Durchschieben (C_I2C).

Die Ausgänge sind dabei wieder in zwei Blöcke aufgeteilt, die aufgrund der Bauweise des Resonators jeweils eigene Konnektoren besitzen, einmal für die ungerade nummerierten Stromversorgungen und einmal für die gerade Nummerierten.

Die Funktionsweise ist dabei wie folgt: Jede Stromversorgung besitzt einen Eingang, welcher die I^2C-Kommunikation regelt. Diese Eingänge werden nun mit je einem Ausgang des Shiftregisters verbunden. Wird dieser Eingang auf HIGH gesetzt so gelangen die beiden für die I^2C-Kommunikation benötigten Linien, eine für die gesendeten Daten, eine als Taktgeber, bis zu den I^2C-Komponenten der Stromversorgung. Bleibt der Eingang auf LOW so wird die I^2C-Kommunikation mit der Stromversorgung abgeschnitten, die beiden Linien bleiben auf der Stromversorgung stets auf LOW, egal ob der Mikrocontroller eine Änderung vornimmt oder nicht. Um also immer genau die eine Komponente anzusprechen, die auch tatsächlich angesprochen werden soll, muss genau der Output des CPLD auf logisch HIGH geschaltet werden, der mit der gewünschten Stromversorgung verbunden ist, und alle anderen auf LOW.

Dies ist einfach realisierbar mit dem Durchschieben einer einzigen 1 durch das Shiftregister. Will man dabei mit einer Komponente von einer bestimmten Stromversorgung kommunizieren, so muss der Taktgeber diese 1 genau so oft weiterschieben, bis sie an dem Ausgang liegt, der mit der jeweils gewünschten Stromversorgung verbunden ist. Danach kann mittels Adressbyte die gewünschte Komponente auf der ausgewählten Stromversorgung angesprochen werden, ohne dass die baugleichen Komponenten auf den anderen

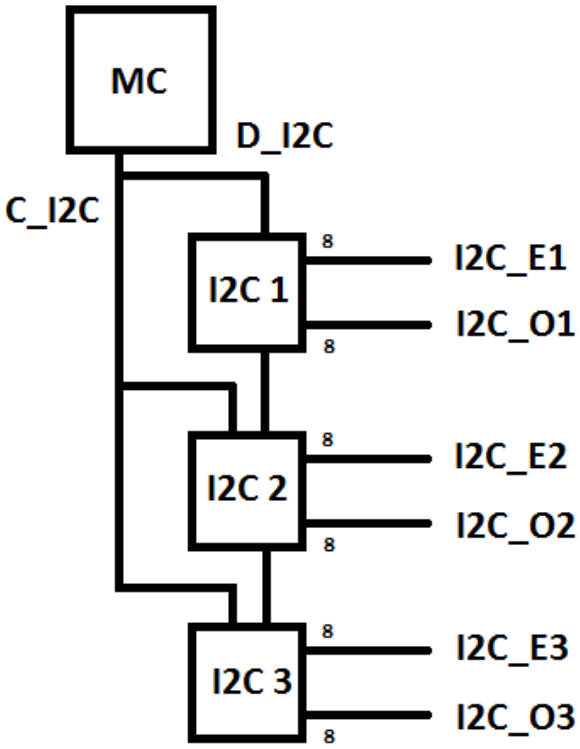

Abbildung 2.12: Schematische Darstellung der drei CPLDs der I^2C-Platine (I2C 1-3), angesteuert durch den Mikrocontroller(MC). Die Nummern bezeichnen die Breite der Datenlinien, ohne Nummer beträgt die Breite einer Datenlinie 1 Bit.

Stromversorgungen reagieren können.

Außer den Shiftregistern sind noch die beiden Konnektoren, die zu den Stromversorgungen führen notwendig, sowie ein Konnektor, der die I^2C-Platine mit der Steuerplatine verbindet, da auf dieser der Mikrocontroller sitzt, der sowohl die Shiftregister ansteuert als auch die I^2C-Kommunikation kontrolliert. Dazu sind noch einige Verstärker verbaut, welche dafür sorgen sollen, dass die Signale der I^2C-Kommunikation auch bei einer hohen Anzahl an angeschlossenen Komponenten ausreichend stark sind.

2.2.3 Verstärkerplatine

Diese dritte Platine sorgt für die Verstärkung und Stabilisierung der digitalen Signale, welche vom Mikrocontroller entweder direkt oder über die Shiftregister an die Stromversorgungen gesendet werden. Die Stabilisierung

ist hier aufgrund der hohen Anzahl an Kanälen nötig: alleine von den Shiftregistern auf der Steuerplatine laufen 96 Kanäle zu den Stromversorgungen. Dazu kommt noch ein durch die hohen Stromstärken bedingtes elektronisch störungsbegünstigendes Umfeld.

Die Stabilisierung soll dafür sorgen, dass keine Schwankungen der Signale auftreten, die so groß sind, dass sie fälschlicherweise als ein falscher Zustand vom Empfänger interpretiert werden. Wie bereits erwähnt liefert der Mikrocontroller Signale entweder mit 5 V (HIGH) oder 0 V (LOW). Von den Bauteilen wird dabei ein Spannungsanstieg auf über 2.1 V als Wechsel von LOW auf HIGH, und umgekehrt ein Senken der Spannung auf unter 0.8 V als Wechsel von HIGH auf LOW wahrgenommen. Wichtig ist also, dass keine Schwankungen in der Spannung auftreten, die zum unbeabsichtigten Über- bzw. Unterschreiten dieser beiden Spannungswerte führen.

Zwei unterschiedliche Arten der Signalübertragung wurden getestet:

- **Differentielle Signalübertragung**: Bei dieser wird jedes Signal auf zwei Datenlinien übertragen, wobei auf einer Datenlinie das tatsächliche Signal, auf der anderen das invertierte Signal übertragen wird. Ein Sender konvertiert das ursprüngliche Signal in die zwei Datenlinien, der Empfänger kann dann etwaige Störungen, die sowohl auf der normalen als auch der invertierten Leitung auftreten mittels Differenzbildung herausfiltern.

- **Transistor-Transistor-Logik** (kurz TTL): Auf Seiten des Senders sind zwei Dioden verbaut, eine mit 0.8 V, die andere mit 2.1 V Durchlassspannung. Liegt die anliegende Spannung unter 0.8 V, so sperren beide Dioden, die so sind mit einem Transistor verbunden sind, dass die Ausgangsspannung auf 0 gesetzt wird, sollten sie im Sperrzustand sein nicht leiten. Liegt die Spannung dagegen über 2.1 V, so setzt der Transistor die Ausgangsspannung auf 5 V. Auf diese Weise werden leicht abweichende Spannungen auf der Seite des Senders wieder auf 0 bzw 5 Volt gesetzt. Auf Seiten des Empfängers sorgt dagegen ein Operationsverstärker dafür, die Spannungen auf 0 bzw. 5 Volt zu stabilisieren.

Um diese Arten der Signalübertragung zu testen, wurde eine kleine Platine entworfen, auf der sich sowohl Sender als auch Empfänger beider Signalübertragungsarten befinden. Dazwischen wurde ein ca. 60 cm langer Kabelabschnitt mit einer Schleife zwischen Sender und Empfänger platziert, über den das Signal dann geführt wird. Der Versuchsaufbau ist in Abb. 2.13 zu sehen.

Der Hauptgrund für unerwünschte Oszillationen, die bei der Änderung des Signalpegels auftreten können, ist die Induktivität des Kabels, die natürlich mit der Länge variiert und von dessen Geometrie abhängt. Das Kabel, mit dem die unterschiedlichen Arten der Datenübertragung getestet wurden,

wurde daher in einer Länge gewählt, die voraussichtlich der Kabellänge im finalen Aufbau des Resonators entspricht.

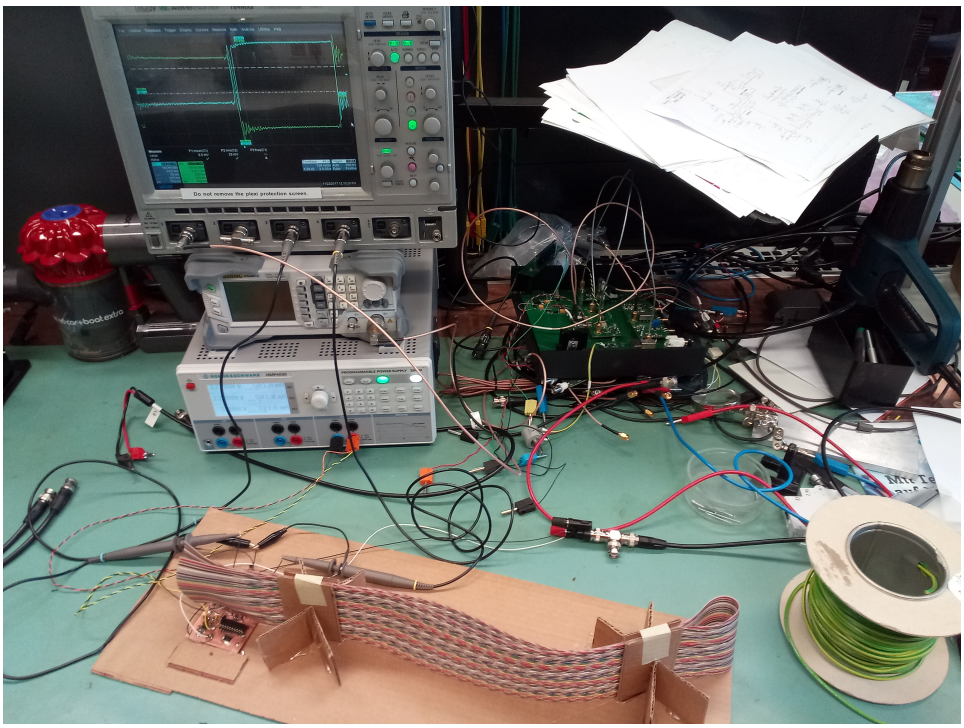

Abbildung 2.13: Versuchsaufbau zum Test der unterschiedlichen Arten der Signalstabilisierung. Links am Karton angebracht ist die Platine mit Sendern und Empfängern für die beiden Übertragungsarten. Deren Ausgänge wurden direkt mit dem Kabel verlötet.

Als Inputsignal wurde ein Rechtecksignal eines Funktionsgenerators verwendet, das Signal am Output wurde mittels Oszilloskop beobachtet. Die Messungen sind in Abb. 2.14 zu sehen. Die Signalübertragung mittels Transistor-Transistor-Logik liefert dabei hinreichend gute Ergebnisse, die Oszillationen an den Ecken des Rechteckssignals sind nicht groß genug, um von einem nachfolgenden Bauteil missinterpretiert zu weden. Bei der Übertragung mittels differentieller Logik treten hingegen Spannungsschwankungen auf, die ausreichen, dass der Empfänger diese als Änderung des Signals interpretieren kann.

Nach diesen Ergebnissen wurde bei der Signalübertragung die Transistor-Transistor-Logik verwendet, die dazugehörigen Sender wurden auf einer dritten Platine verbaut, der Verstärkerplatine. Jedes Signal, dass der Mikrocontroller direkt oder über die Ausgänge der Shiftregister an die Stromversorgungen sendet, wird zunächst von dem TTL-Sender verstärkt, und gelangt erst dann über ein Kabel zur dazugehörigen Backplane. An diesen Back-

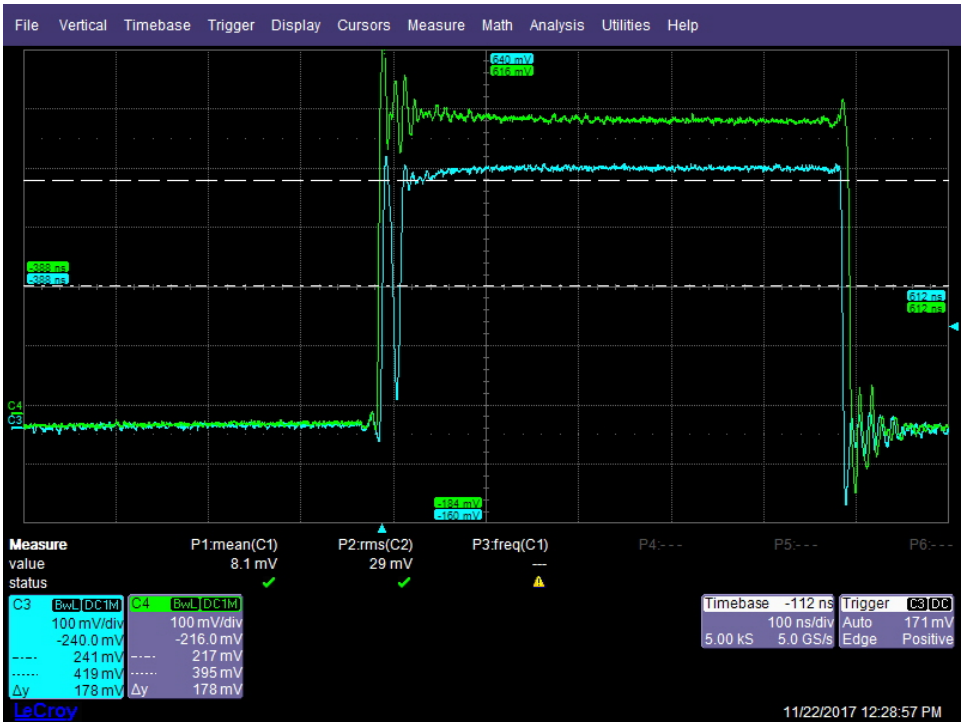

Abbildung 2.14: Messung des Signals nach Kodierung und Dekodierung mittels differentieller Datenübertragung (Linie 3, blau) bzw mittels Transistor-Transistor-Logik (TTL) (Linie 4, grün).

planes sind dann noch die TTL-Empfänger in der Nähe der Konnektoren für die Stromversorgungen angebracht, bevor das Signal dann schließlich zur jeweiligen Stromversorgung gelangt.

Kapitel 3

Steuerung durch Programmierung des Mikrocontrollers

Nachdem im vorherigen Kapitel die Hardware besprochen wurde, mit welcher die vielen Inputs der Stromversorgungen mit wenigen Outputs des Mikrocontrollers korrekt gesteuert werden können, soll nun auf die Programmierung des Mikrocontrollers selbst eingegangen werden, um einen korrekten Ablauf der Resonatorsteuerung ermöglichen zu können.

Dabei werden zunächst die Eingänge der Stromversorgungen besprochen, welche vom Mikrocontroller angesteuert werden und wie gewährleistet wird, dass bei den 48 Stromversorgungen stets nur die tatsächlich gewünschte Stromversorgung angesprochen wird und nicht versehentlich eine falsche oder mehrere auf einmal.

Danach wird der Mikrocontroller selbst noch einmal genauer besprochen und darauf eingegangen, wie die Outputs des Mikrocontrollers geschaltet werden müssen, damit die gewünschten Aktionen, wie etwa die Umschaltung zwischen Real und Artificial Load oder der Ablauf einer I^2C-Kommunikation korrekt durchgeführt werden.

3.1 Eingänge der Stromversorgung

Um das Schalten wie gewünscht durchzuführen, müssen nun einige Eingänge der Stromversorgung angesteuert werden. Diese Eingänge sowie zwei Ausgänge, deren Signale ebenfalls vom Mikrocontroller verarbeitet werden, sind in Abb. 3.1 zu sehen und werden nun kurz beschrieben.

nTEMP_ALARM sowie nOPEN sind die einzigen zwei Ausgänge der Stromversorgung, dabei handelt es sich um Alarme zweier I^2C-Komponenten.

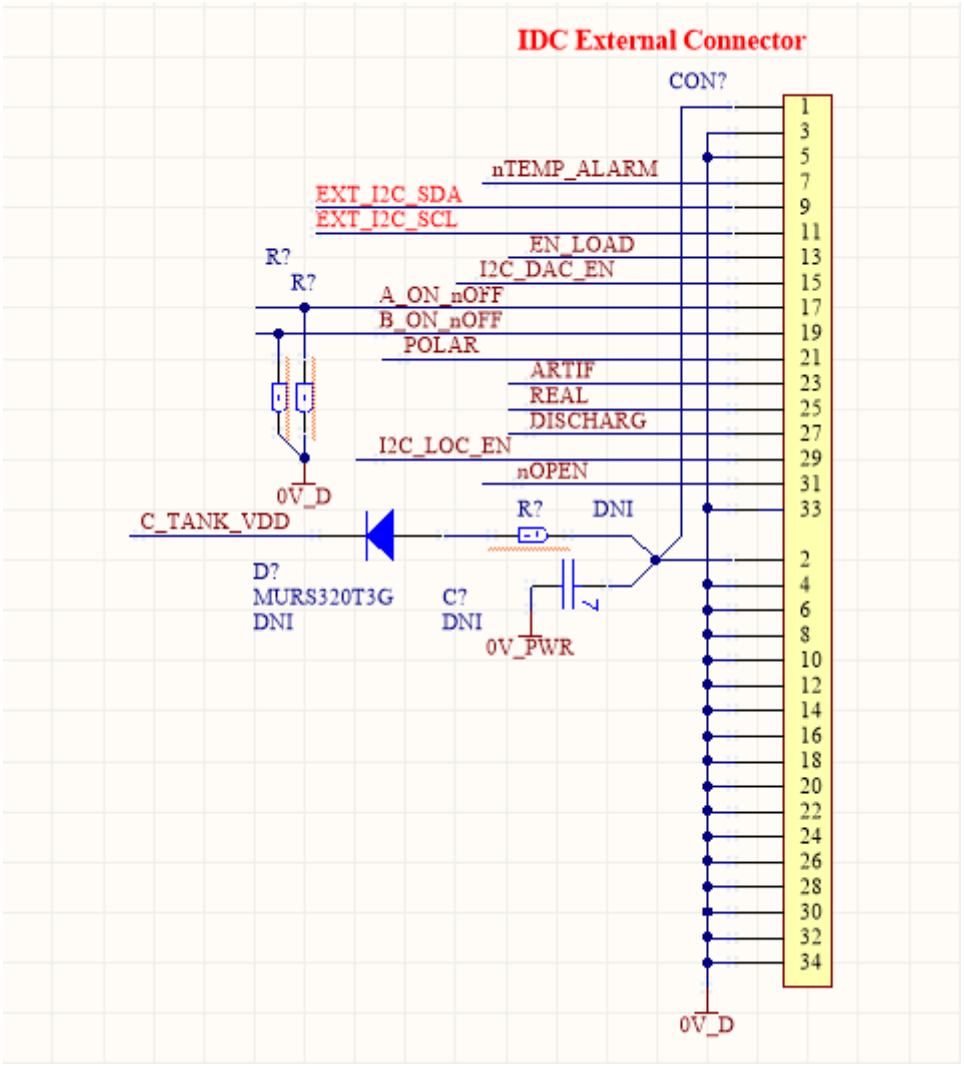

Abbildung 3.1: Der auf jeder der 48 Stromversorgungen montierte Konnektor mit den unterschiedlichen Ein- und Ausgängen in der schematischen Ansicht des *Altium Designers*.

nTEMP_ALARM ist dabei der Alarm des Temperatursensors: Wird eine vorher mittels I^2C-Kommunikation eingestellte Temperatur überschritten, wird dieser normalerweise im logisch HIGH Zustand befindliche Ausgang auf LOW herabgezogen. [1]

nOPEN ist der Alarm des Analog-To-Digital-Converters, welcher die an der Spule anliegende Spannung misst. Ist der Kontakt der Spule mit der Stromversorgung unterbrochen, kann dies über die Spannungsmessung de-

[1] Eingänge mit einem vorangestellten kleinen "n"sind immer invertiert, also aktiv im Zustand LOW.

tektiert werden, auch hier ist der Ausgang im Normalfall auf logisch HIGH und wird nur im Alarmfall auf LOW herabgezogen.

Aufgrund der begrenzten Anzahl an Pins des Mikrocontrollers werden diese beiden Alarme aller 48 Stromversorgungen auf einer einzigen Signalleitung verarbeitet, es können also im Falle eines Alarms Gegenmaßnahmen, wie das kontrollierte Herunterfahren des Resonators, ergriffen werden, allerdings kann nicht gesagt werden, von welcher Stromversorgung der Alarm kommt und um welchen Alarm es sich handelt.

Für den Schaltvorgang sind die Eingänge EN_LOAD, A_ON_nOFF, B_ON_nOFF, ARTIF und REAL verantwortlich. Die Eingänge EN_LOAD, A_ON_nOFF, B_ON_nOFF werden dabei direkt von einem Ausgang des Mikrocontrollers gesteuert, sie werden also für alle 48 Stromversorgungen gleichzeitig geschaltet.

A_ON_nOFF und B_ON_nOFF schalten die gewünschte Stromquelle ein, ist A_ON_nOFF auf HIGH, so ist die Stromquelle für 0.2-5 A aktiv, ist hingegen B_ON_nOFF auf HIGH, so ist die Stromquelle für 5-20 A aktiv, es können aber auch beide gleichzeitig eingeschaltet werden, um für Ströme bis zu 25 A zu erzielen.

Um nicht unnötig viele Ausgänge des Mikrocontrollers verwenden zu müssen, werden diese beiden Eingänge von nur einem einzigen Ausgang des Mikrocontrollers bedient. Es können also nur entweder alle Stromquellen bei allen Stromversorgungen eingeschaltet werden oder gar keine. Da der ausgegebene Strom sowieso mittels I^2C-Kommunikation pro Stromquelle eingestellt wird (siehe Unterabschnitt 2.2.2), ist dies ausreichend.

EN_LOAD wird ebenfalls direkt vom Mikrocontroller geschaltet. Solange dieser Eingang nicht auf HIGH gesetzt wird, kann kein Strom über eine der Loads fließen, auch wenn an den Eingängen A_ON_nOFF sowie B_ON_nOFF bereits Spannung anliegt.

Die Eingänge REAL und ARTIF steuern nun, über welche der beiden Loads der Strom fließt. Liegt Spannung an REAL an, so wird der Strom über die Real Load, also die Resonatorspule, geleitet, liegt Spannung an ARTIF an, so fließt der Strom über die Artificial Load. Wenn an beiden Eingängen Spannung anliegt, kann der Strom auch über beide Loads fließen, dies kommt während dem Schaltvorgang vom einer Load zur anderen vor.

Diese beiden Eingänge sind es auch, die nicht direkt vom Mikrocontroller angesteuert werden, sondern von den Ausgängen der CPLD-Shiftregister. Da jeder dieser Eingänge für jede der Stromversorgungen extra geschaltet werden müsste, bräuchte man 96 Ausgänge, was die Kapazität des Mikrocontrollers deutlich übersteigt. Zudem wäre es auch mit dieser Anzahl an Ausgängen nicht möglich, den Status aller 96 ARTIF und REAL Eingänge gleichzeitig zu verändern, sondern müsste dies hintereinander duchführen.

Da diese Eingänge nun aber an den parallelen Ausgängen des Shiftregisters liegen, kann mit steigender Flanke des Taktgebers, der an einem Ausgang des Ausgang des Mikrocontrollers zur Verfügung steht, das gesamte Muster an den Ausgängen um eine Stelle verschoben werden. Dies ist genau das, was benötigt wird, um ein Neutronenpaket durch schrittweises Schalten der aufeinanderfolgenden Spulen des Resonators zu begleiten.

Damit Strom über eine oder beide Loads fließen kann, müssen also zunächst EN_LOAD, A_ON_nOFF und B_ON_nOFF auf HIGH gesetzt werden, danach müssen noch ARTIF und REAL hochgeschaltet werden, je nachdem über welche der beiden Loads oder eventuell gleichzeitig über beide der Stromfluss gewünscht wird.

Bei EXT_I2C_SDA und EXT_I2C_SCL handelt es sich um die zwei für die I^2C-Kommunikation benötigten Signalleitungen. Auf EXT_I2C_SDA werden dabei die zu sendenden Daten übermittelt, während EXT_I2C_SCL der Taktgeber für die Kommunikation ist.

I2C_LOC_EN und I2C_DAC_EN agieren als Schlösser für die I^2C-Kommunikation. Erst wenn I2C_LOC_EN auf HIGH gesetzt wird, können über die I^2C-Signalleitungen Signale vom Mikrocontroller zu den I^2C-Komponenten, oder umgekehrt, gelangen.

Um die Digital-To-Analog-Converter anzusteuern, die den Strom einstellen, der über die Spulen fließt, einstellen, muss dann auch noch I2C_DAC_EN auf HIGH gesetzt werden. Dies soll zusätzliche Sicherheit bringen, damit nicht versehentlich eine Kommunikation mit einem der DACs durch einen Fehler z.B. im Adressbyte vorgenommen wird. Wann welche I^2C-Komponenten ansprechbar sind, ist in der folgenden Tabelle angeführt.

I2C_LOC_EN	I2C_DAC_EN	Kommunikation mit I^2C-Komponenten außer DAC	Kommunikation mit DAC
niedrig	niedrig	nein	nein
hoch	niedrig	ja	nein
niedrig	hoch	nein	nein
hoch	hoch	ja	ja

Die Eingänge I2C_LOC_EN der Stromversorgungen werden dabei von den Ausgängen des Shiftregisters der I^2C-Platine (siehe Unterabschnitt 2.2.2) gesteuert, die restlichen Eingänge sind direkt mit einem Ausgang des Mikrocontrollers verbunden.

Um eine Kommunikation vorzunehmen, wird eine einzelne 1 so oft weitergeschoben, bis diese bei der gewünschten Stromversorgung angelangt ist. Erst dann kann eine I^2C-Kommunikation mit dieser Stromversorgung aufgenommen werden.

Will man dazu noch durch Beschreiben des DAC einen neuen Wert für den Strom einstellen, so muss der Ausgang I2C_DAC_EN am Mikrocon-

troller auf HIGH gesetzt werden. Wie schon erwähnt, wird dieser Ausgang zwar für alle Stromversorgungen auf einmal geschaltet, dennoch kann, wie in obiger Tabelle ersichtlich, nur der DAC der eingestellten Stromversorgung angesprochen werden.

3.2 Der Mikrocontroller

Nun soll auf den Mikrocontroller näher eingegangen werden, der sich auf der Steuerplatine befindet und den Ablauf des Schaltens der Shiftregister steuert sowie die I^2C-Kommunikation vornimmt. wie bereits erwähnt, handelt es sich um einen PIC24FV32KA304 Mikrocontroller der Firma MIKROCHIP

Wie bereits in Abschnitt 2.2.1 erwähnt, fungiert hierbei als Taktgeber ein externer Oszillator, mit einer Taktfrequenz von 29.4912 MHz. Der hier verwendete Mikrocontroller benötigt dabei zwei Oszillatortakte, um einen Befehl auszuführen, die für einen Befehl benötigte Zeit liegt folglich bei 67,82 ns.

Es handelt sich dabei um einen 16-Bit Mikroprozessor, seine Register besitzen also üblicherweise eine Größe von 16 Bit, bzw zwei Bytes oder ein (Daten-)Wort. Der Mikrocontroller besitzt 16 Arbeitsregister und unter anderem Module für I^2C-Kommunikation und asynchrone serielle Kommunikation mittels UART (Universal Asynchronous Receiver Transmitter), welche für die Steuerung des Experiments von hoher Bedeutung sind. So werden mittels I^2C-Kommunikation verschiedene Komponenten der Stromversorgungen beschrieben und ausgelesen, das UART-Modul wiederum ermöglicht eine Kommunikation mit dem PC über einen seriellen Port unter Verwendung des Industriestandard RS-485.

Auch insgesamt fünf voneinander unabhängige 16-Bit Timer, die nach einer einstellbaren Zeit (von 0 bis zu 65535 Befehlszyklen) einen Interrupt auslösen können, sind vorhanden. Dies kann unter anderem dazu benützt werden, in regelmäßigen Abständen den Inhalt der unterschiedlichen Shiftregister weiter zu schieben. Da die Ausgänge dieser Shiftregister festlegen, ob Strom über die Resonatorspulen fließt und dadurch das Resonatormagnetfeld entsteht, kann der Timer also benutzt werden um ein Neutronenpaket in der gewünschten Geschwindigkeit mit dem Magnetfeld mitzubegleiten.

Die maximale Dauer zwischen zwei Interrupts des 16-Bit Timers liegt dabei bei 4.444 ms (= maximale Anzahl an Befehlszyklen $\times$ Dauer eines Befehlszyklus $= 2^{16} \times 67.817\,\text{ns}$), sollten längere Schaltzeiten gewünscht sein, können auch zwei der Timer gemeinsam im 32-Bit Modus betrieben werden, dann erhält man eine maximale Dauer von über 291.262 s ($= 2^{32} \times 67.817\,\text{ns}$) bevor der Timer den nächsten Interrupt auslöst.

Die Wellenlängenselektion ist im Wanderwellenmodus abgesehen von der Stärke des transversalen Magnetfeldes nur abhängig vom Zeitintervall, während dessen sich ein selektierter Neutronenpuls innerhalb einer Resonatorspule aufhält und bei seinem Durchflug durch den Resonator vom Transver-

salfeld 'begleitet' wird. Dieses Zeitintervall kann immer nur ein ganzzahliges Vielfaches der Zeit für einen Befehlszyklus sein. Das bedeutet aber in weiterer Folge, dass auch die mögliche Auswahl der Wellenlängen nicht kontinuierlich sein kann. So ist es etwa nicht möglich, einen Neutronenpuls von genau 2 Å zu selektieren, ein typischer Wert bei einem thermischen Neutronenspektrum. Dieser bewegt sich nämlich mit 1978 m/s und benötigt daher für die Durchquerung einer 1 cm breiten Resonatorspule 5.056 μs, was 74.55 Befehlstakten entspricht. Da aber nur ganzzahlige Vielfache der Befehlstaktdauer möglich sind, ist es also unmöglich genau diese Wellenlänge zu selektieren, werden hingegen 74 Takte abgewartet, so wird eine Wellenlänge von 1.985 Å ausgewählt, bei 75 Takten eine Wellenlänge von 2.012 Å.

Als Programmiersprache wird im vorliegenden Fall Assemblercode verwendet, die Programmieroberfläche bietet dabei das Programm MPLAB X IDE, ebenfalls entwickelt von MIKROCHIP. Es wurde die Version 3.51 verwendet.

Als Assembler, der dafür verantwortlich ist, die Assemblerbefehle in Maschinenspache zu übersetzen, wird der Assembler der XC 16 Compiler Toolchain verwendet. Ein Compiler ist im Gegensatz zu höheren Programmiersprachen dagegen nicht vonnöten, da ein solcher ja nur die Befehle höherer Betriebssprachen in Assemblerbefehle übersetzt.

Der Grund für die Verwendung von Assemblercode ist, dass Assemblerbefehle im Gegensatz zu Befehlen höherer Betriebssprachen vom Assembler eins zu eins in Maschinenbefehle übersetzt werden.

Folglich kann genau bestimmt werden, wie lang ein Befehlsblock dauert, indem einfach die Oszillatortakte abgezählt werden und dann mit der Dauer eines Befehlszyklus multipliziert werden. Besonders für den Schaltvorgang von Artificial zu Real Load und umgekehrt ist diese zeitkritische Komponente äußerst wichtig, durch die gewünschte Anzahl an Befehlen kann sichergestellt werden dass bei jedem Schaltvorgang stets dieselbe, genau festgelegte Schaltzeit vorliegt.

Im Folgenden wird darauf eingegangen wie dieser Schaltvorgang, sowie andere Funktionen wie die Kommunikation mittels einer seriellen Schnittstelle oder das Vornehmen einer I^2C-Kommunikation programmiertechnisch implementiert werden können.

Zunächst soll dabei allerdings noch kurz ein Überblick über die Assemblersprache gegeben werden. Im Gegensatz zu höheren Programmiersprachen ist diese dabei nicht einheitlich, sondern es ist für (fast) jedes Gerät ein anderer Befehlssatz vorhanden. Der gesamte Befehlssatz für den PIC24FV32KA304-Mikrocontroller ist in [15] zu finden.

Dabei gibt es drei Arten von Befehlen:

- **Wort-Befehle**: Da es sich um einen 16-Bit Mikrocontroller handelt, sind die Register üblicherweise 16 Bit (i.e. ein Wort bzw. zwei Bytes) breit, mit einem Wort-Befehl wird ein solches Register als ganzes manipuliert.
- **Bit-Befehle**: Bei diesen Operationen wird nur ein Bit getestet oder manipuliert.
- **Kontroll-Befehle**: Diese Befehle sind wichtig für die Struktur eines Programmes, so werden mit diesen Befehlen zum Beispiel Abzweigungen und Sprünge im Programmablauf vorgenommen.

In der folgenden Tabelle sind die wichtigsten Befehle der Programmiersprache aufgelistet. Neben dem eigentlichen Befehl stehen die Operanden, die der Befehl benötigt, wobei w für ein Arbeitsregister, f für ein beliebiges anderes Register, k für einen Skalar von 0 bis 65535, m und n für Skalare von 0 bis 15 und *add* für eine beliebige Programmadresse stehen.

Wort-Befehle	

mov f,wn
mov wn,f
mov wm,wn
	Kopiert den Inhalt eines beliebigen Registers in ein Arbeitsregister, den Inhalt eines Arbeitsregisters in ein beliebiges Register oder den Inhalt eines Arbeitsregisters in ein anderes Arbeitsregister.

clr f clr w	Löscht den Inhalt eines (Arbeits-)Registers.
Bit-Befehle	
bset f,n bset wm,n	Setzt das Bit eines (Arbeits-)Registers an der n-ten Stelle auf 1.
bclr f,n bclr wm,n	Setzt das Bit eines (Arbeits-)Registers an der n-ten Stelle auf 0.
btsc f,n btsc wm,n	Ist das Bit an der n-ten Stelle eines (Arbeits-)Registers 0, so wird der nächste Befehl übersprungen.
btss f,n btss wm,n	Ist das Bit an der n-ten Stelle eines (Arbeits-)Registers 1, so wird der nächste Befehl übersprungen.
btst f,n btst wm,n	Überprüft das Bit an der n-ten Stelle eines (Arbeits-)Registers, entsprechende Flags werden gesetzt.
Kontroll-Befehle	
goto add	Programm macht absoluten Sprung zur gewünschten Adresse, diese kann im Code auch als eine beliebige Zeichenfolge definiert sein.
bra add	Programm macht relativen Sprung zur gewünschten Adresse, diese kann im Gegensatz zum goto-Befehl auch mit einer Bedingung verbunden werden. So wird z.B. die Verzweigung beim Befehl bra Z,add nur durchgeführt, wenn die Z(Zero)-Flag gesetzt wurde, also das Ergebnis der letzten Operation 0 war.
call add	Ruft Subroutine an der angegebenen Adresse auf, diese kann im Code auch als eine beliebige Zeichenfolge definiert sein.
return	Kehrt von der Subroutine zurück zu der Stelle, an der sie zuvor mittels call-Befehl aufgerufen wurde.
repeat k	Wiederholt den nächsten Befehl k-mal.
nop	Nulloperation, es wird für die Dauer eines Befehlszyklus nichts ausgeführt. Dies kann verwendet werden um, falls erforderlich, eine bestimmte Zeit verstreichen zu lassen.

3.3 Codebeispiele

In diesem Unterkapitel wird nun anhand von Beispielen gezeigt, wie unterschiedliche Abläufe, die für einen korrekten Ablauf des Experiments nötig sind, mittels Assemblercode realisiert werden können. Dazu gehören wichtige Punkte wie der simultane Schaltvorgang von Artificial zu Real Load, bzw umgekehrt, mit einem bestimmten Muster, aber auch die Durchführung einer I^2C-Kommunikation oder einer Kommunikation mittels seriellem Port.

Für jeden dieser Punkte wird ein Codebeispiel präsentiert und der dazugehörige Assemblercode kommentiert, um ein besseres Verständnis der Abläufe zu ermöglichen.

In Kapitel 4 werden dann unterschiedliche Tests der Hardware präsentiert, welche alle mittels Programmen durchgeführt wurden, die hauptsächlich aus den hier präsentierten Codebeispielen in leicht abgeänderter Form bestehen.

3.3.1 Schaltvorgang

Nun soll beschrieben werden, wie ein beliebiges Muster über die Manipulation der Outputs des Mikrocontrollers durch die Shiftregister geschoben werden kann. In Abb. 3.2 sind die Pins des Mikrocontrollers mit den ihnen zugeteilten Funktionen zu sehen, der Mikrocontroller besitzt dabei drei Registerbänke, die Port A, B und C genannt werden. Mit den Wort-Befehlen können die Ports der einzelnen Registerbänke simultan beschrieben und gelöscht werden. Dies ist in weiterer Folge wichtig, da es dadurch möglich ist, gleichzeitig mehrere Eingänge der Shiftregister zu verändern.

Besonders wichtig ist dies bei Registerbank A, auf der sowohl die Taktgeber (Outputs CLK_G/A/R_MCU) als auch die Dateninputs (DIN_G/A/R) der drei Shiftregister liegen. Dadurch ist nämlich möglich, diese sechs Outputs simultan zu verändern und somit Daten gleichzeitig durch die unterschiedlichen Shiftregister zu schieben.

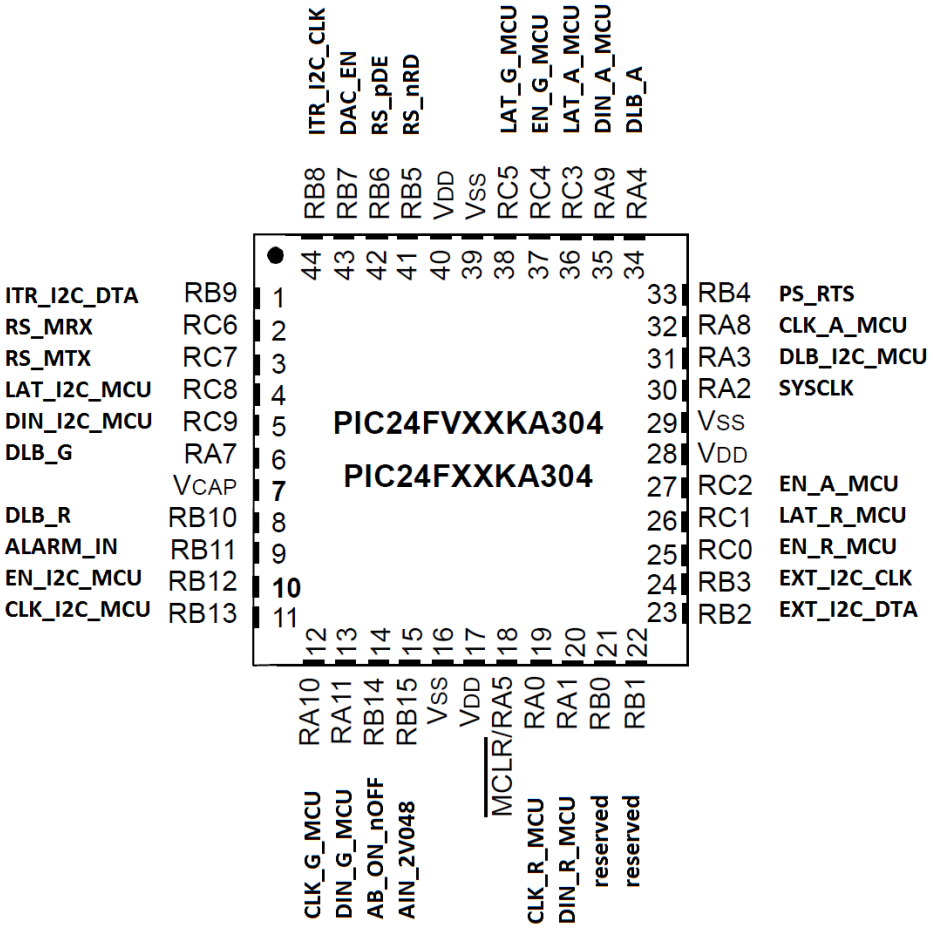

Abbildung 3.2: Die Pins des Mikrocontrollers schematisch dargestellt. *Außen:* vom User vergebene Namen der Pins. *Innen:* die Registerbank und die Position des Pins auf eben dieser.

Die restlichen Pins von Registerbank A sind dabei nur Inputs, sie werden bei einem Wort-Befehl nicht verändert, bei DLB_G/A/R[2] handelt es sich dabei um das aus den jeweiligen 48-Bit-Shiftregistern herausgeschobene Bit, es kann danach vom Programm überprüft werden, um sicherzustellen, dass das gewünschte Muster auch korrekt durchgeschoben wurde. Bei MCLR_ICSP handelt es sich um ein Master Clear, welches nur vom Programmierkabel während des Programmiervorgangs angesteuert wird.

Weitere, für den Schaltvorgang wichtige Outputs sind LAT_A/R/G_MCU, sie steuern die latched Inputs der Shiftregister an. Werden diese auf HIGH gesetzt, so gelangt das an den Flipflops anliegende Muster bis zu den Outputs des Shiftregisters (siehe Abschnitt 2.1).

Die Ausgänge EN_A/R_MCU sind mit LAT_A/R_MCU mittels logischem ODER verknüpft (siehe auch Abb. 2.4), ist also einer der beiden Outputs auf HIGH gesetzt, gelangt das Muster zu den Outputs. Will man von Anfang an das Muster an den Outputs anliegen haben, kann man gleich zu Anfang des Programms EN_A und EN_R auf HIGH setzen, soll aber zunächst ein bestimmtes Muster in den Flipflops gespeichert werden und erst später zu den Outputs gelangen, werden EN_A/R_MCU auf LOW geschaltet und zu gegebener Zeit LAT_A/R_MCU auf HIGH.

Der Ausgang CLROUT_G_MCU steuert den Eingang *clroutput* (siehe Abb. 2.5) an, welcher alle Ausgänge des Shiftregisters auf LOW setzt, das Muster selbst bleibt allerdings weiterhin in den Flipflops gespeichert und kann in weiterer Folge auch wieder an den Outputs ausgegeben werden.

Im Folgenden soll nun gezeigt werden, wie ein beliebiges Muster so durchgeschoben werden kann, dass das resultierende Magnetfeld in den Spulen ein Neutronenpaket mit einer bestimmten Geschwindigkeit mitbegleiten kann.

Dabei soll gewährleistet werden, dass sich das Magnetfeld so schnell wie möglich auf den Spulen stabilisieren kann. Dies wird wie bereits erwähnt dadurch realisiert, dass zunächst der Strom über einen der Resonatorspule ähnlichen Widerstand (Artificial Load) geleitet wird und erst danach von diesem auf die Spule (Real Load) umgeschaltet wird.

Dabei zeigte es sich, dass die besten Ergebnisse erzielt werden, wenn beim Schaltvorgang für ca. 600 ns der Strom über beide Loads geleitet wird. Auch für das Abschalten des Magnetfeldes konnten die besten Ergebnisse erreicht werden, wenn der Strom vom Real Load auf den Artificial Load geschalten wurde und dabei der Strom kurzzeitig über beide Loads geleitet wurde.

In Abb. 3.3 ist grafisch dargestellt, wie die Ausgänge des Mikrocontrollers geschaltet werden müssen, damit alle Schaltvorgänge korrekt ablaufen. Die Abbildung zeigt wie ein einfaches Muster von zwei aufeinanderfolgenden Einsen und abgesehen davon ausschließlich Nullen durch das Shiftregister geschoben wird, welches die Outputs des Real Loads ansteuert.

[2] DLB_R ist aus Platzgründen auf Registerbank B untergebracht

Auch wenn es durchaus möglich ist, komplexere Muster durch die Shiftregister zu schieben, beinhaltet dieses Muster schon die beiden möglichen Schaltvorgänge – Artificial zu Real Load und umgekehrt – und zeigt wie mit dem Glitch-Shiftregister umgegangen werden muss, um diese Schaltvorgänge korrekt vorzunehmen, ohne dabei den Strom über beide Loads zu leiten, wenn gerade kein Schaltvorgang vor sich geht.

Das Durchschieben eines solchen Musters ist dabei gleichbedeutend mit dem Mitbegleiten eines Neutronenpakets mit einem Magnetfeld in jeweils zwei aufeinander folgenden Spulen des Resonators.

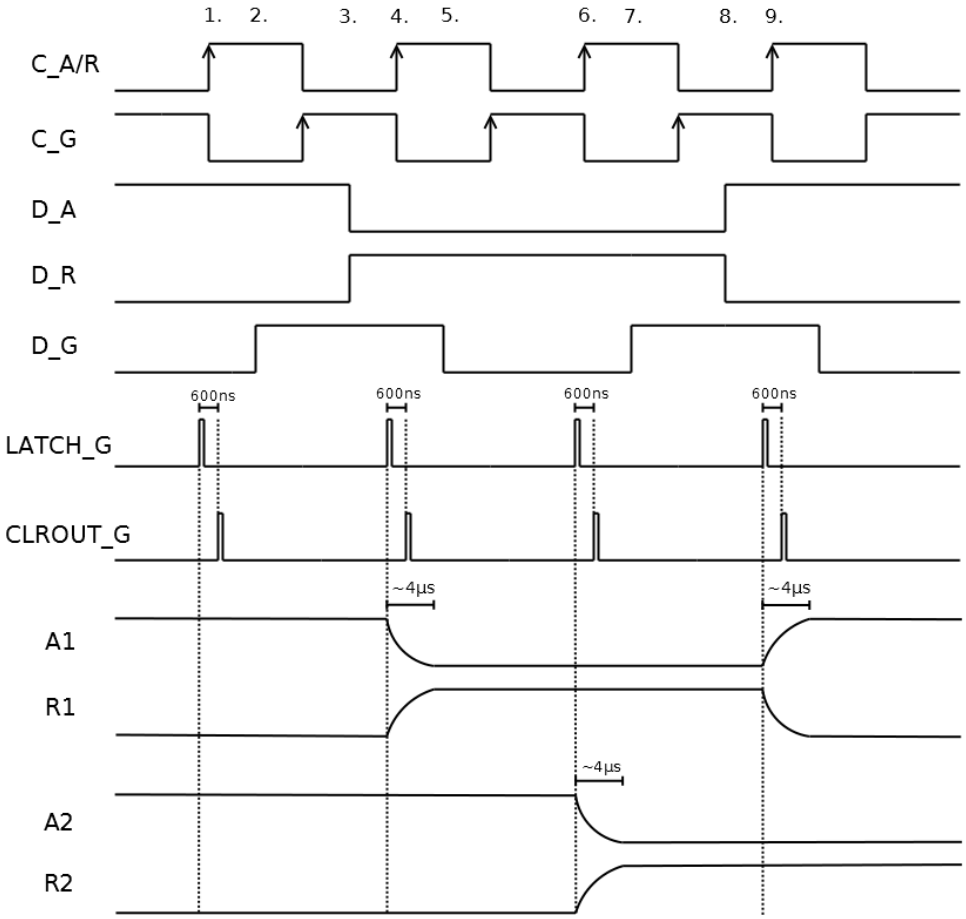

Abbildung 3.3: Schematische Darstellung, wie die unterschiedlichen Signallinien geschalten werden müssen, um ein einfaches Muster von zwei aufeinanderfolgenden Einsen in den Resonator zu schieben. Dies entspricht dem Mitbegleiten eines 2 cm dicken Neutronenpakets im Wanderwellenmodus des Resonators, es wird folglich sowohl ein Choppen des Neutronenstrahls und eine Wellenlängenselektion vorgenommen.

Folgende Punkte sollen nun nochmals kurz besprochen werden, um ein besseres Verständnis der zeitlichen Abfolgen zu ermöglichen:

Die Shiftregister funktionieren wie folgt: Sobald an den Uhren (C_A/R/G) eine steigende Flanke (mittels Pfeil noch extra gekennzeichnet) anliegt, wird ein Shift vorgenommen, dabei wird das in den Registern gespeicherte Muster um eine Stelle weitergeschoben und das zu dem Zeitpunkt der steigenden Flanke an der dazugehörigen Datenlinie anliegende Signal an die erste Stelle des Registers geschoben.

Das Muster wird in den Flipflops der Shiftregister gespeichert und nur dann an den Ausgängen ausgegeben, wenn der dazugehörige latched Input auf logisch HIGH liegt, dies geschieht, wenn einer der Eingänge ENOUT und LATCH oder beide auf logisch HIGH sind. Diese beiden Eingänge werden ebenfalls von Ausgängen des Mikrocontrollers angesteuert.

Die 48 Ausgänge der Artificial- und Real-Shiftregister bedienen dabei die Eingänge ARTIF und REAL der 48 Stromversorgungen (siehe Unterkapitel 3.1). Wenn ein Ausgang des Artificial-Shiftregister logisch HIGH ist, so wird bei der dazugehörigen Stromversorgung der Strom über die Artificial Load geleitet, dasselbe gilt für das Real-Shiftregister in Bezug auf die Real Load, also die Resonatorspulen.

Das Glitch-Shiftregister ist dagegen wie schon erwähnt für den Schaltvorgang von höherrangiger Bedeutung, jeder der 48 Ausgänge ist mit einer logischen ODER-Verknüpfung mit den dazugehörigen Ausgängen der Real- und Artificial-Shiftregister verbunden. Ist also ein Ausgang logisch HIGH, so wird die dazugehörige Stromversorgung den Strom gleichzeitig über Artificial und Real Load leiten, unabhängig davon, welchen Zustand die Ausgänge dieser beiden Shiftregister haben.

Da die ENOUT_A/R Ausgänge in diesem Beispiel ständig auf HIGH gesetzt sind, wird das in diesen Shiftregistern gespeicherte Muster sofort an den Ausgängen ausgegeben, anders verhält es sich dagegen beim Glitch-Register. Bei diesem wird das gespeicherte Muster nur dann an den Ausgänge ausgegeben, wenn LATCH_G auf HIGH geschaltet wird. Wie Abbildung 3.3 zu entnehmen ist, ist dies jeweils nur kurz vor der steigenden Flanke der Taktgeber C_A/R der Fall.

Folglich wird dieser Ausgang LATCH_G dazu benützt, während eines Schaltvorgangs den Strom über beide Loads zu leiten. Daraufhin wird das Muster in die Artificial- und Real-Shiftregister weitergeschoben und 600 ns, nachdem LATCH_G auf logisch HIGH gesetzt wurde, wird auch der Ausgang CLROUT_G auf HIGH gesetzt. Dieser setzt wiederum alle Ausgänge des Glitch-Registers auf LOW, ohne dass das gespeicherte Muster verloren geht, das wieder an die Ausgänge ausgegeben werden kann, sobald LATCH_G wieder auf HIGH gesetzt wird.

Natürlich muss darauf geachtet werden, dass der Ausgang des Glitch-Registers nur dann den Strom über beide Loads leitet, wenn auch tatsächlich ein Schaltvorgang zwischen Real und Artificial Load vorliegt. Dies wird

programmseitig dadurch gewährleistet, dass nur dann eine 1 in das Glitch-Register geschoben wird, wenn ein solcher Übergang zwischen Real und Artificial Load vorliegt. Ist dem nicht so, wird eine 0 gesetzt. Da die Ausgänge des Glitch-Registers mittels logischem ODER mit den Ausgängen der Artificial und Real Loads verknüpft sind, hat eine Null des Glitch-Registers keinerlei Konsequenzen auf die anderen Ausgänge.

Als Ausgangspunkt wird hier angenommen, dass zum Startzeitpunkt bereits alle Stromversorgungen Strom über die jeweiligen Artificial Loads leiten. Mit diesen zusätzlichen Infos soll nun Abb. 3.3 näher beschrieben werden, dabei sind die Zeitpunkte der folgenden Aufzählung zum besseren Verständnis im Diagramm noch einmal vermerkt. Bei den Datenlinien im Diagramm handelt es sich um digitale Spannungspegel, es gibt also nur die beiden logischen Zustände HIGH und LOW. Ausnahmen sind nur *A1*, *R1*, *A2* und *R2*, bei denen der tatsächliche Stromverlauf über die ersten beiden Real und Artificial Loads angezeigt wird.

1. Bei der ersten steigenden Flanke der Uhr C_A/R liegt die Datenlinie für die Artificial Load auf HIGH, jene für die Real Load auf LOW, folglich wird die erste Stromversorgung ab diesem Zeitpunkt beginnen, Strom über die Artificial Load zu leiten. Da noch nichts in das Glitch-Register geschoben wurde, liegen die Ausgänge auf 0, es wird also wie gewünscht nicht kurzzeitig Strom über beide Loads geleitet.

2. Bevor die erste steigende Flanke der Uhr C_G für das Glitch-Register auftritt, wird die Datenlinie für den Glitch D_G auf HIGH gesetzt, da bei der nächsten steigenden Flanke von C_A/R ein Wechsel von Artificial zu Real Load auftreten wird. Mit der steigenden Flanke wird nun diese 1 an die erste Stelle des Shiftregisters geschoben, da LATCH_G allerdings LOW ist, bleibt der Ausgang des Shiftregisters noch auf LOW.

3. Vor der steigenden Flanke von C_A/R wird die Datenlinie D_A auf LOW und D_R auf HIGH gesetzt, ein Übergang von Artificial auf Real Load wird vorbereitet.

4. Unmittelbar bevor diese Flanke stattfindet wird LATCH_G kurz hochgeschaltet, und da genau davor eine 1 in das Glitchregister geschoben wurde, wird der Strom ab diesem Zeitpunkt bei der ersten Stromversorgung über beide Loads geleitet. Die sofort darauf folgende Flanke von C_A/R sorgt daraufhin bei der ersten Stromquelle für den Wechsel von Artificial zu Real Load, die zweite Stromquelle dagegen beginnt ab diesem Zeitpunkt Strom über den Artificial Load zu leiten. Ca 600 ns nachdem LATCH_G auf HIGH desetzt wurde, wird CLROUT_G auf HIGH geschaltet, ab diesem Zeitpunkt fließt der Strom bei der ersten Stromquelle nur noch über die Real Load.

5. Da nach dem nächsteten Shift der Strom bei der ersten Stromquelle immer noch über die Real Load fließen soll und kein Schaltvorgang stattfindet, wird D_G auf LOW gesetzt, damit die steigende Flanke von C_G eine 0 in das Glitch-Register schiebt. Die 1, die sich zuvor auf der ersten Stelle befand, wird nun an die zweite Stelle geschoben.

6. Bei der nächsten Flanke von C_A/R tritt an der ersten Stromquelle keine Änderung auf, der Ausgang des Glitch-Registers ist LOW, also wird auch kein Strom kurzzeitig über beide Loads fließen. Hingegen erfolgt bei der zweiten Stromversorgung jetzt ein Übergang von Artificial auf Real Load, der genauso abläuft wie einen Zyklus zuvor bei der ersten Stromquelle. Bei der dritten Stromquelle wird dagegen nun Strom über die Artificial Load geleitet.

7. Nun soll der Übergang von Real auf Artificial Load bei der ersten Stromquelle stattfinden. Da nun auch hier kurzzeitig über beide Loads Strom geleitet werden soll, wird D_G auf HIGH gesetzt und mit der steigenden Flanke von C_G an die erste Stelle des Shiftregisters geschoben.

8. Für den Übergang von Real auf Artificial Load werden zunächst D_A auf HIGH und D_R auf LOW gesetzt.

9. Sobald LATCH_G auf HIGH gesetzt wird, fließt bei der ersten Stromquelle Strom über beide Loads, das Muster wird bei der steigenden Flanke von C_A/R um eine Stelle weitergeschoben und sobald CLROUT_G auf HIGH gesetzt wurde, fließt der Strom nur noch über die Artificial Load.

   Bei den weiteren Stromquellen geschieht jeweils das, was einen Zyklus zuvor bei der Stromquelle direkt davor geschah.

Auf die eben beschriebene Weise kann nun ein beliebiges Muster durch den Resonator geschoben werden, damit ein Neutronenpaket wie gewünscht über die gesamte Laänge des Resonators von einem Magnetfeld begleitet wird. Für das richtige Timing ist dabei das Timer-Modul des Mikrocontrollers zuständig, das nach einer gewissen Anzahl an Befehlstakten einen regelmäßigen Interrupt verursachen kann, der stets unmittelbar vor der steigenden Flanke von C_A/R stattfindet. So kann sichergestellt werden, dass die Zeit zwischen zwei aufeinander folgenden Flanken stets gleich ist, was genau dem Zeitintervall entspricht, während dem das Magnetfeld in einer der Resonatorspulen konstant bleiben soll.

Nun soll anhand eines Codebeispiels gezeigt werden, wie dieser Schaltvorgang mittels Assemblerprogrammierung realisiert werden kann. Zu beachten

ist hierbei, dass in diesem Beispiel davon ausgegangen wird, dass die Ströme, die über die Spulen fließen sollen, bereits eingestellt wurden. Wie dies abläuft, wird in Abschnitt 3.3.2 noch gezeigt werden.

Initialisierung

Zunächst müssen der Timer initialisiert sowie noch einige andere Vorbereitungen getroffen werden, damit der Schaltvorgang wie gewünscht ablaufen kann. Kommentare zum Code können nach einem Semikolon geschrieben werden und werden vom Assembler ignoriert.

Dezimalzahlen werden mit einem # vor der Zahl gekennzeichnet, Binärzahlen mit *#0b* und Hexadezimalzahlen mit einem *#0x*. Bei speziellen Registern haben die meisten Bits einen generischen Namen, dieser kann statt der Nummer des Pins verwendet werden. So wird *IFS0, #T1IF* vom Assembler als *IFS0,#3* interpretiert, die generischen Namen sowie Bedeutungen können in [15] nachgelesen werden.

```
clr TMR1          ;clear timer 1
clr T1CON
mov #1000,W0        ;1000*67.8 ns = 678us
mov W0,PR1          ;set timer 1 time out period
                    ;when PR1=TMR1=T1 interrupt
bclr IFS0,#T1IF     ;clear t1 interrupt flag IFS0,#T1IF
bset IEC0,#T1IE     ;set interrupt enable IEC0,#T1IE
```

In diesem Teil wird das Timer1 Modul vorbereitet und Interrupts von diesem ermöglicht. Das Modul funktioniert wie folgt: Sobald das TON-Bit des Kontrollregisters T1CON gesetzt wird, wird das Register TMR1 mit jedem Befehlstakt um 1 erhöht. Sobald der Inhalt von TMR1 mit dem des Registers PR1 übereinstimmt, wird die Interruptflag T1IF im Register IFS0 gesetzt. Ist auch das Bit T1IE im Register IEC0 gesetzt, findet ein Interrupt statt. [3]

Zunächst wird hier der Inhalt von TMR1 gelöscht, dies ist eine Vorsichtsmaßnahme für den Fall, dass das letzte Programm beendet wurde, als der Inhalt des Registers ungleich Null war. Danach wird das Kontrollregister T1CON ebenfalls gelöscht. Mit diesem Register können unterschiedliche Betriebsmodi des Timers festgelegt werden, in dem hier verwendeten Modus werden all diese Bits auf Null gesetzt. Außerdem befindet sich das TON-Bit in diesem Register, sobald dieses auf 1 gesetzt wird, inkrementiert sich das Register TMR1 mit jedem Befehl. Da dies zu diesem Zeitpunkt noch nicht erwünscht ist, wird dieses TON-Bit ebenfalls Null gesetzt.

[3] Die Flag T1IF wird unabhängig vom Status von T1IE gesetzt, man kann den Timer also falls gewünscht auch ohne Interrupt verwenden und den Status von T1IF an einem gewissen Programmpunkt abfragen.

Dann wird das Register PR1 vorbereitet, welches festlegt, nach wie vielen Takten die Interruptflag T1IF gesetzt werden soll. Im vorliegenden Beispiel findet alle 1000 Takte ein Interrupt statt, also alle 678 μs. Die Konstante wird zunächst in das Arbeitsregister W0 geladen und über dieses dann in das Register PR1. Vorsichtshalber wird die Interruptflag T1IF noch gelöscht und das Bit T1IE gesetzt, um Hardware-Interrupts durch den Timer 1 zu ermöglichen. Damit ist der Timer1 initialisiert und - sofern das TON-Bit im Kontrollregister gesetzt ist - nach jeweils 1000 Takten, oder alle 678 μs ein Interrupt ausgelöst.

```
mov #0b0000010000000000,W4   ;W4 is register for CLK_AR_LOW
mov #0b0000000100000001,W5   ;W5 is register for CLK_AR_HIGH

bset LATC,#0      ;enable Latch A/R (invisible Mode)
bset LATC,#2      ;comment out if latching

bset LATB,#14     ;set AB_ON_nOFF
```

Wie in Abb. 3.2 zu sehen ist, liegen sowohl die Taktgeber als auch die Datenlinien der drei Shiftregister auf Port A. Die Taktgeber sind dabei an den Pins A0 (C_R), A8 (C_A) sowie A10 (C_G) zu finden. Die beiden Arbeitsregister werden nun so initialisiert, dass sie, nachdem ihr Inhalt auf Port A kopiert wurde, diese Uhren wie in Abb. 3.3 manipulieren können.

Wird W4 auf Port A kopiert, so werden C_G auf HIGH und C_A sowie C_R auf LOW gesetzt. Wird hingegen W5 auf Port A kopiert, so werden C_A und C_R auf HIGH gesetzt und C_G auf LOW. So kann mittels abwechselndem Kopieren von W4 und W5 die gewünschte simultane Veränderung des Zustands der Uhren realisiert werden.

Daraufhin werden noch die Latches für Artificial- und Real-Shiftregister auf HIGH gesetzt, damit ein Muster, welches in die Shiftregister geschoben wird, sofort an den Ausgängen des Shiftregisters ausgegeben wird ('transparenter' Modus), die sich auf den Pins C0 und C2 befinden. Will man den Status eines Pins mit einer Bit-Operation ändern, so wird das Register LATx verwendet, wobei x für den jeweiligen Port steht. Folglich setzen die beiden Operationen die Pins C0 und C2 auf HIGH und die beiden Latches sind daraufhin ebenfalls auf logisch HIGH.

Daraufhin wird noch der Pin B14 auf HIGH gesetzt, der den Eingang AB_ON_nOFF der Stromversorgung ansteuert (siehe Abschnitt 3.1). Damit Strom über die Loads fließen kann muss dieser auf HIGH gesetzt werden.

Stabilisierung auf der Artificial Load

Bevor nun mittels Timer-Interrupt ein beliebiges Muster in die Shiftregister geschoben wird, soll der Strom zunächst ausschließlich über die Artificial

Loads geleitet werden, wo er sich stabilisieren kann. Diese lange Stabilisierung ist nur am Anfang notwendig, daraufhin wird nur noch zwischen Artificial und Real Load hin- und hergeschaltet, wobei nur sehr kurze Schaltzeiten auftreten.

Der grundlegende Ablauf ist wie folgt: Die Datenlinie für die Artificial Load wird auf HIGH gesetzt, jene für den Real Load auf LOW. Folgt nun eine steigende Flanke der Uhr C_A/R, so wird die erste Stromversorgung Strom über die Artificial Load leiten, wobei die unerwünschten Effekte, wie das Überschießen des Stroms sowie die lange Stabilisationszeit von mehreren hundert Mikrosekunden, auftreten. Diese Zeitspanne wird abgewartet, bevor mit der nächsten steigende Flanke der Uhr C_A/R die nächste Stromquelle beginnt, Strom über die Artificial Load zu leiten. Nach 48 Zyklen leiten somit alle 48 Stromversorgungen einen Strom über die Artificial Load und es wird mit dem nächsten Programmteil fortgefahren.

```
   call setlow       ;set Artificial Data
   mov W4, PORTA     ;put data in

   mov #48, W1       ;nr of coils used
   mov W1, counter

1:
   mov W5, PORTA     ;clock high
   nop
   nop
   nop
   nop
   mov W4, PORTA     ;and down

   repeat #10000
   nop

   dec counter       ;check counter
   bra Z,2f          ;zero? advance
   bra 1b            ;nonzero, again
2:
```

Zunächst wird die Subroutine *setlow* aufgerufen, welche die Register W4 und W5 dahingehend manipuliert, dass nach dem Kopieren auf Port A der Strom im nächsten Takt über die Artificial Load geleitet wird. Diese Subroutine befindet sich am Ende des Codes und wird nun kurz erklärt:

```
setlow:
    bset W4,#9      ;set A_DTA high on Wregs for high and low clock
    bset W5,#9
```

```
    bclr W4,#1      ;set R_DTA low on Wregs for high and low clock
    bclr W5,#1
    return
```

Hierbei werden die 9. Bits der Register W4 und W5 auf HIGH gesetzt, die 1. Pins dagegen auf LOW. Da die Datenlinie für die Real Load auf Pin A1 liegt und jene für die Artificial Load auf Pin A9, führt ein Kopieren der Arbeitsregister dazu, dass der Strom daraufhin über den Artificial Load fließt.

In weiterer Folge wird auch die Subroutine *sethigh* benützt, die genau gegensätzlich die Pins für die Real Load auf HIGH und jene für die Artificial Load auf LOW setzt.

```
sethigh:
    bset W4,#1      ;set R_DTA high on Wregs for high and low clock
    bset W5,#1
    bclr W4,#9      ;set A_DTA low on Wregs for high and low clock
    bclr W5,#9
    return
```

Nach dieser Subroutine wird zunächst W4 auf Port A kopiert, dies ist das Arbeitsregister, in dem C_A/R LOW ist, es findet also kein Shift durch diese Register statt. Die Datenlinie für das Artificial Register liegt aber nun bereits auf HIGH. Würde man direkt schon W5 auf Port A kopieren, würde die Datenlinie erst im Moment der steigenden Flanke auf HIGH gesetzt und Unsicherheit darüber bestehen, was nun tatsächlich an das Shiftregister übergeben wird.

Nun wird ein Zähler vorbereitet, welcher dafür sorgen soll, dass alle 48 Stromversorgungen korrekt initialisiert werden. Dafür wird einfach eine zuvor deklarierte Variable *counter* benützt. Es wird die Konstante 48 über das Arbeitsregister W1 auf *counter* kopiert.

Nach dem Marker *1:*, zu welchem mittels bra-Befehl gesprungen werden kann, folgt die erste steigende Flanke von C_A/R. W5 wird auf Port A kopiert, die Datenlinie für das Artificial Register liegt dabei auf HIGH, folglich beginnt die erste Stromversorgung Strom über die Artificial Load zu leiten. Nach vier folgenden '*nop*'-Befehlen, welche sicherstellen sollen, dass genug Zeit für die Shiftregister bleibt ist, um auf die steigende Flanke zu reagieren, wird W4 auf Port A kopiert, die Uhr C_A/R wird wieder LOW gesetzt, und die Datenlinien bleiben unverändert.

Nun folgen mittels '*repeat*' 10000 '*nop*'-Befehle mit einer Gesamtdauer von 678 μs, mehr als genug Zeit, damit sich der Strom in der Artificial Load stabilisieren kann.

Danach wird mittels *dec*-Befehl der Zähler um 1 dekrementiert, ergibt diese Operation Null, so wird dabei die *Z(Zero)-Flag* gesetzt. Diese wird

nun im nächsten Befehl als Bedingung benützt. Der Befehl *bra Z,2f*(forward) bewirkt im Code einen Sprung vor zum nächsten Marker *2:* falls die Z-Flag gesetzt wurde. Dies ist erst dann der Fall, wenn alle 48 Stromversorgungen initialisiert wurden und Strom über die Artificial Loads leiten.

Ist dies noch nicht der Fall, so wird der Befehl *bra 1b*(back) ausgeführt mit welchem zum Marker *1:* zurückgesprungen wird. Hier wird dann wieder die Uhr C_A/R auf HIGH gesetzt und der Inhalt des Shiftregisters dadurch einen Platz weitergeschoben. Da die Datenlinie des Artificial Loads auf HIGH und jene des Real Loads auf LOW liegt wird jedes Mal eine neue Artificial Load initialisiert.

Regelmäßiger Schaltvorgang mittels Timer-Interrupt

Nun werden noch letzte Vorbereitungen getroffen, bevor mittels Timer-Interrupt das Muster in regelmäßigen Abständen durchgeschoben wird.

```
   mov #0b001100110011, W1    ;pattern for coils

   btss W1,#0        ;check to see if glitchbits need to be set
   bra 1f

   bset LATA,#1
   bclr LATA,#9
   bset LATA,#11           ;Set glitchbits on pins
1:
   bclr LATA,#10           ;shift glitchbit in
   nop                     ;with setting clock down
   nop
   bset LATA,#10           ;and up again

   bset T1CON,#TON         ;start timer

loop:
   goto loop               ;loop forever
```

Zunächst wird das gewünschte Muster für die Real Loads auf das Arbeitsregister W1 kopiert. Der Befehl *btss W1,#0* überprüft das nullte Bit des Arbeitsregisters und überspringt den nächsten Befehl, sollte dieses Bit gesetzt sein. Dabei handelt es sich um jenes Bit, welches bei der nächsten steigenden Flanke in das Shiftregister für Artificial und Real Load geschoben wird.

Ist dieses Bit eine Null, muss das Datenbit für das Glitch-Register nicht gesetzt werden, da die erste Stromquelle dann weiterhin über die Artificial Load leitet und ein kurzzeitiges Leiten über beide Loads in diesem Fall

nicht erwünscht ist. Der Befehl *bra 1,f* überspringt daraufhin den nächsten Befehlsblock.

Ist dieses Bit allerdings gesetzt, so wird dieser *bra* Befehl übersprungen und ein Übergang von Artificial zu Real Load vorbereitet, dabei wird die Datenlinie für die Real Load auf HIGH (*bset LATA,#1*), die für die Artificial Load auf LOW (*bclr LATA,#9*), und jene für den Glitch ebenfalls auf HIGH (*bset LATA,#11*) gesetzt. Dies ermöglicht den gewünschten Schaltvorgang von Real auf Artificial Load inklusive kurzzeitigem Leiten über beide Loads beim nächsten Takt der Uhr.

Um dieses Bit nun an die erste Stelle des Glitch-Registers zu schieben, wird danach noch die Uhr, die sich auf Pin A10 befindet, LOW und wieder HIGH geschaltet. Nun ist alles korrekt vorbereitet und der Timer kann gestartet werden, das Programm befindet sich nun in einer Endlosschleife, die nur der Timerinterrupt unterbrechen kann.

Der Timer-Interrupt springt dabei zu der mit dem Marker *__T1Interrupt* markierten Stelle im Programmcode, an welcher der Schaltvorgang mit dem Shiftregister vorgenommen wird:

```
__T1Interrupt:

    call setglitchdata    ;look if glitch bits
                          ;need to be set for the next time
    call setglitch        ;use Glitch when rising the clock

;from here on clock is high

    rrnc W1,W1

    btsc W1,#0        ;is lowest bit 1
    call sethigh      ;yes, else skip

    btss W1,#0        ;is lowest bit 0
    call setlow       ;yes, else skip


    mov W4,PORTA


Int_end:
    bclr IFS0, #T1IF      ;clear Interrupt flag
    retfie
```

Zunächst wird die Routine *setglitchdata* aufgerufen, die das Glitch-Register für den nächsten Durchgang vorbereitet. Dies wird bereits zu diesem Zeit-

punkt gemacht, da beim Schaltvorgang (dieser findet in der direkt darauf folgenden Subroutine *setglitch* statt) W5, das die steigende Flanke für das Artificial- und Real-Register und die sinkende für das Glitch-Register beinhaltet, auf Port A kopiert wird. Die Daten für das Glitch-Register sollen schon bei der sinkenden Flanke korrekt gesetzt werden, damit keine Änderung der Daten stattfindet, wenn danach W4 kopiert wird und die steigende Flanke auftritt.

Diese Subroutine wird nun kurz erläutert.

```
setglitchdata:
    btst W1,#0      ;see if lowest bit is zero
    bra Z,1f        ;if it is branch to 1

;come here if first bit was 1!
    btst W1,#1      ;see if next bit is zero
    bra Z,3f        ;if it is branch to 3

;here if first bit 1, next bit 1
    bclr W4,#11     ;Clear glitchdata
    bclr W5,#11     ;for both
    return

1:    ;branch here if first bit was 0
    btst W1,#1      ;see if next bit is zero
    bra Z,2f

    ;here if first bit 0, next bit 1
    bset W4,#11     ;Set glitchdata
    bset W5,#11     ;for both
    return

2:  ;branch here if first bit 0, next bit 0
    bclr W4,#11     ;Clear glitchdata
    bclr W5,#11     ;for both
    return

3:  ;branch here if first bit 1, next bit 0
    bset W4,#11     ;Set glitchdata
    bset W5,#11     ;for both
    return
```

Hier werden die ersten beiden Bits des Arbeitsregisters W1 abgeglichen, in welchem das duchzuschiebende Muster gespeichert ist. Mithilfe des *btst*-Befehls werden diese beiden Bits untersucht und da diese Operation die

Z(Zero)-Flag setzt, wird *bra Z* verwendet, um je nach Ergebnis einen Vergleich der Bits möglich zu machen.

Je nach Ergebnis wird dann das elfte Bit von W4 und W5 - dieses steuert dann die Datenlinie für das Glitchregister an - entweder gesetzt, falls ein Schaltvorgang bevorsteht, die beiden Bits also unterschiedliche Zustände haben, oder gelöscht, falls beide Bits sich im selben Zustand befinden. Jeder Weg durch diese Funktion benötigt gleich viele Befehle, das zeitkritische Verhalten bleibt folglich erhalten.

Danach wird die Subroutine *setglitch* aufgerufen, die den Übergang von Artificial auf Real Load mithilfe der Glitch-Register realisiert, um im Falle eines Schaltvorgangs kurzzeitig über beide Loads zu leiten.

```
setglitch:
    bset LATC,#5   ;Latch the glitch
    nop
    nop
    nop
    mov W5, PORTA
    nop
    nop
    nop
    bset LATC,#4   ;clear the glitch with the reset
    bclr LATC,#5   ;unlatch
    bclr LATC,#4   ;clear the reset
    return
```

Sofort nach dem Aufrufen der Subroutine wird der Pin C5 auf HIGH gesetzt, dieser steuert den Latch des Glitch-Registers an, folglich wird das Muster welches im Register gespeichert ist, an den Ausgängen ausgegeben.

Ab jetzt leiten all jene Stromversorgungen, an denen der Ausgang des Glitch-Registers HIGH ist, über beide Loads. Die *nop*-Befehle sorgen dafür, dass die Schaltzeit entsprechend lang ist. Dazwischen wird noch das Arbeitsregister W5 auf Port A kopiert, was eine steigende Flanke für die Artificial- und Real-Register erzeugt und deren Inhalt um eine Stelle weiter schiebt, während dagegen die Uhr für das Glitch-Register auf 0 gesetzt wird.

Nach der Schaltzeit wird der Pin C4 auf HIGH gsetzt. Dieser ist mit dem *clrout*-Eingang des Glitch-Registers verbunden und setzt alle Ausgänge des Registers auf 0, sodass der Strom jetzt nur noch über eine der Loads fließt.

Daraufhin werden *Latch* und *Clear* wieder auf LOW gesetzt und die Funktion springt mittels *return* zurück ins Hauptprogramm:

```
    rrnc W1,W1

    btsc W1,#0       ;is lowest bit 1
    call sethigh     ;yes, else skip
```

```
    btss W1,#0          ;is lowest bit 0
    call setlow         ;yes, else skip


    mov W4,PORTA


Int_end:
    bclr IFS0, #T1IF    ;clear Interrupt flag
    retfie
```

Danach wird noch das im Arbeitsregister W1 gespeicherte Muster um eine Stelle weitergeschoben, die Funktion *rrnc W1,W1* (*rotate right no carry*) schiebt dabei den Inhalt des Registers um eine Stelle nach rechts, das rechts herausgeschobene Bit wird dabei am linken Ende wieder hineingeschoben. Nun werden die Arbeitsregister W4 und W5 für den nächsten Schaltvorgang vorbereitet, *sethigh* wird dabei aufgerufen, falls eine 1 als nächstes in das Shiftregister geschoben wird, welches die Real Loads ansteuert, *setlow* nur wenn eine 0 folgt.

Danach wird W4 auf Port A kopiert, dies bewirkt eine steigende Flanke bei der Uhr des Glitch-Registers und sorgt dafür, dass dieses um eine Stelle weitergeschoben sowie ein neues Bit hineingeschoben wird.

Zeitgleich findet die sinkende Flanke für die Uhren der Real und Artificial Shiftregister statt. Da kurz zuvor bereits die Arbeitsregister W4 und W5 für den nächsten Schaltvorgang aktualisiert wurden, werden die Daten für den nächsten Schaltvorgang bereits jetzt an den Ausgängen A1 und A9 eingestellt. Sie werden also nicht mehr verändert, wenn die steigende Flanke den nächsten Shift auslöst.

Nun ist ein Zyklus des Schaltvorganges vollzogen, bevor mittels *retfie*-Befehl (*return from Interrupt*) wieder in die Endlosschleife zurückgesprungen wird, muss noch die Interrupt Flag *T1IF* händisch gelöscht werden, damit der Timer sie beim nächsten Mal wieder setzen kann und so den nächsten Interrupt auslöst, der den nächste Schaltvorgang startet.

3.3.2 I²C-Kommunikation

Nun soll gezeigt werden wie die I²C-Kommunikation mit den unterschiedlichen Stromversorgungen und deren verschieden Komponenten (ADCs, DACs, Temperatursensoren, FRAM) möglich ist. Wie bereits erwähnt, besitzt jede der Stromversorgungen genau dieselben I²C-Komponenten, die auch alle dieselben Adressen besitzen.

Würden also alle Stromversorgungen an aktiven I²C-Datenlinien hängen, würden auf ein versendetes Adressbyte die Komponenten auf jeder Strom-

versorgung reagieren, um dies zu vermeiden, werden zwei Datenlinien verwendet, die wie Schlösser funktionieren, I2C_LOC_EN und I2C_DAC_EN (siehe Abschnitt 3.1).

Während I2C_LOC_EN HIGH sein muss, damit überhaupt ein Signal der I^2C-Datenleitung zu den Komponenten gelangt, muss zusätzlich auch I2C_DAC_EN auf HIGH gesetzt werden, will man mit den DACs kommunizieren, um eine neue Stromstärke einzustellen.

Um den Eingang I2C_LOC_EN anzusteuern, wird nun wieder ein 48-Bit Shiftregister verwendet, wieder bestehend aus drei hintereinander geschalteten CPLDs. Jeder der 48 Ausgänge dieses Shiftregisters ist dabei mit genau einer Stromversorgung verbunden und steuert den Eingang I2C_LOC_EN an. Um dann mit der gewünschten Stromquelle eine II^2C-Kommunikation aufnehmen zu können, wird eine einzelne 1 einfach so oft durch das Shiftregister geschoben, bis der Ausgang des Shiftregisters auf HIGH liegt, welche genau diese Stromquelle ansteuert.

Der Ausgang I2C_DAC_EN wird hingegen für alle Stromversorgungen auf einmal mit einem Ausgang des Mikrocontrollers angesteuert. Da dieser nur dann eine Auswirkung hat, wenn auch I2C_LOC_EN auf HIGH liegt, ist dies ausreichend um die Kommunikation mit nur einem einzigen DAC auf einer Stromversorgung zu gewährleisten.

Der PIC24FV32KA304 Mikrocontroller verfügt über zwei I^2C-Module, welche beide verwendet werden. Modul 1 verwendet dabei den Pin B9 als Datenlinie (SDA1) und B8 als Uhr (SCL1) und wird verwendet um die Komponenten anzusprechen, welche auf der Steuerplatine befestigt sind; ein Temperatursensor und ein Drucksensor. Um diese anzusprechen ist kein Shiftregister nötig, da keine Adressen doppelt vorhanden sind und keine Zweideutigkeiten auftreten können.

Das zweite Modul verwendet Pin B2 als Datenlinie (SDA2) und B3 als Uhr (SCL2) und führt zu den Stromversorgungen. Hier wird mittels Shiftregister ausgewählt, mit welcher Stromversorgung eine I^2C-Kommunikation aufgenommen werden soll.

Als Beispiel soll hier nun gezeigt werden, wie mittels Shiftregister alle 48 Stromversorgungen hintereinander auf einen bestimmten Stromwert eingestellt werden können, indem mittels I^2C-Kommunikation Daten an die DACs der Stromversorgungen geschickt werden.

Initialisierung

Dabei wird zunächst das zweite I^2C Modul initialisiert, das wie vorhin erwähnt die Kommunikation mit den Komponenten auf der Stromversorgung steuert:

```
init_I2C:     ;Set I2C 2 module
     bset I2C2CON, #I2CEN   ;enable I2C module

  mov #145,W0     ;Fosc 29,4912MHz, 100kHz I2C
  mov W0,I2C2BRG

init_I2C_Shiftreg
  mov #48,W0                ;move value of coils to the counter
  mov W0, coilcounter


  bset LATC,#8      ;Latch I2C-CPLDs
  bset LATC,#9      ;set Data high for the first shift
```

Das Register *I2C2CON* ist ein Kontrollregister für das I^{2}C 2 Modul, durch Setzen der einzelnen Bits dieses Moduls können unterschiedliche Betriebsmodi für das Modul ausgewählt werden, die genaue Beschreibung dieses Registers findet sich in [15]. In diesem Fall muss bei diesem Register allerdings nur das *#I2CEN* (I^{2}C Enable) Bit gesetzt werden, welches das Modul betriebsbereit macht. Daraufhin wird noch das *I2C2BRG* (I^{2}C Baud Rate Register) Register initialisiert, die Baudrate ist hierbei abhängig von der Geschwindigkeit des Oszillators, in diesem Fall 29.4912 MHz und der gewünschten Datenübertragungsrate der I^{2}C-Kommunikation, hier 100 kHz. Mithilfe der Formel (aus [15], S. 171) kann bestimmt werden, welcher Wert I2C2BRG zugeordnet werden soll:

$$I2C2BRG = \frac{F_{CY}}{F_{SCL}} - \frac{F_{CY}}{10^7} - 1$$

Hier ist F_{CY} die Frequenz, mit der eine Instruktion verarbeitet werden kann, also genau die halbe Frequenz des Oszillators, und F_{SCL} die für die I^{2}C-Kommunikation gewünschte Frequenz ist. Mit den eingesetzten Werten errechnet sich hier der Wert 145, welcher in das I2C2BRG-Register kopiert wird.

Daraufhin wird noch der Zähler *coilcounter* initialisiert, der wird in weiterer Folge dazu genützt wird, um die korrekte Anzahl an Shifts durch das I^{2}C-Shiftregister duchzuführen, die erforderlich ist, um mit jeder Stromversorgung eine Kommunikation aufnehmen zu können. Danach wird noch der Latch des Shiftregisters, der sich auf Pin C8 befindet auf HIGH gesetzt, da das in das Shiftregister geschobene Muster sofort an den Ausgängen ausgegeben werden soll. Auch die Datenlinie für das Shiftregister an Ausgang C9 wird auf HIGH gesetzt, bei der ersten steigenden Flanke des Taktgebers wird dann eine 1 in das Register geschoben.

Hauptprogramm

```
I2C_Loop:

   bset LATB,#13       ;set clock high

   mov #50, W0         ;set 1 AMP
   mov W0, DACHI
   mov #00, W0
   mov W0, DACLO
   call DAC0Write      ;write the DAC value

   mov #00, W0
   mov W0, DACHI
   mov #00, W0
   mov W0, DACLO
   call DAC1Write      ;to both

   bclr LATC,#9        ;data low
   bclr LATB,#13       ;clock low

   dec coilcounter     ;see if we have to shift I2C again
   bra Z,1f          ;if zero pass on
   goto I2C_Loop
1:

   bset LATB,#13     ;set clock high one last time to shift the 1 out

   repeat #10
   nop

   bclr LATB,#13
```

Zunächst findet eine steigende Flanke des Taktgebers für das I^2C-Shiftregister, welcher mit dem Ausgang B13 verbunden ist, statt. Dabei wird zunächst eine 1 in das Shiftregister geschoben. Da der Latch ebenfalls HIGH ist, wird der erste Ausgang des Shiftregisters auf logisch HIGH gesetzt und somit die I^2C-Kommunikation mit der ersten Stromversorgung ermöglicht.

Die Subroutinen *DAC0Write* und *DAC1Write* sind für die Kommunikation mit den beiden DACs der Stromversorgung zuständig, erstere ist zuständig für die Stromstärken bis zu 5 Ampere, die zweite für jene bis 20 Ampere. Wie diese Subroutinen programmiert sind, wird etwas später noch erklärt.

Die DACs sind dabei im 12-Bit Modus, da der Datenkanal für die I^2C-Kommunikation 8 Bit breit ist, werden jedem DAC 2 Bytes geschickt, wovon

die zweite Hälfte des zweiten Bytes vom DAC ignoriert wird. Die Subroutinen *DAC0Write* und *DAC1Write* senden den Inhalt des Registers *DACHI* als erstes Byte und den Inhalt von *DACLO* als zweites Byte an den jeweiligen DAC, die letzten 4 Bits von *DACLO* werden dabei ignoriert.[4]

Die Auflösung der unterschiedlichen Stromquellen kann dabei wie folgt berechnet werden: Wird der maximale Wert, bei diesen 12-Bit Komponenten wäre dies $2^{12} - 1 = 4095$, an die Stromversorgung gesendet, so wird der maximale Strom von der Komponente ausgegeben, dies wäre bei einem DAC 5 A, beim anderen 20 A. Durch eine einfache Division kann nun die minimale Schrittweite des Stroms eruiert werden, bei dem ersten DAC sind dies 1.22 mA, beim anderen 4.88 mA. Bei Experimenten mit dem MONOPOL-Wanderwellenresonator an sehr kalten Neutronen wurde ein linearer Umrechnungsfaktor von 9.45 μT/A gefunden [18]. Dies ergibt ein maximales Feld von 236.25 μT (bei 25 Ampere), sowie Schrittgrößen von ungefähr 46.1 nT bzw 11.5 nT.

Die Amplitude der horizontalen Feldoszillationen im Resonator muss auf die Stärke B_0 des vertikalen Selektorfelds abgestimmt sein, das die Resonanzwellenlänge λ_0 definiert. Für gegebenes Selektorfeld ergibt sich die Resonanzwellenlänge aus der Beziehung [5]

$$\lambda_0 = \frac{c}{a}\frac{1}{B_0} \ ,$$

wobei $a = 1$ cm die Breite einer Resonatorspule und $c = 6.783 \times 10^{-15}\ Tm^2$ eine Konstante ist.

Ein exakter Spinflip tritt ein, wenn der akkumulierte Larmorotationswinkel von Neutronen der Resonanzwellenlänge beim Durchqueren des Resonators ein ungerades Vielfaches von π beträgt

$$\beta\frac{L}{a} = 2k + 1 \qquad (k = 0, 1, 2, ...)$$

wobei L die Gesamtlänge des Resonators und k die Ordnung des Spinflips bezeichnen und der Parameter $\beta = 2B_1/\pi B_0$ vom Verhältnis der Amplidude B_1 der transversalen Feldsozillationen im Resonator zur Stärke B_0 des Selektorfeldes abhängt [5]. In niedrigster Ordnung ($k = 0$) ist die Wellenlängenauflösung des Resonators gegeben durch $(\Delta\lambda/\lambda)_{1/2} \simeq 1.6a/L = 0.8N$, wobei $N = L/2a$ die Gesamtzahl der räumlichen Perioden des Resonators ist [8].

[4] Wie alle Register des Mikrocontrollers sind *DACHI* und *DACLO* eigentlich 16 Bit Register, allerdings werden diese bei der I^2C-Kommunikation in ein 8 Bit Register kopiert, wobei der Kopiervorgang die oberen 8 Bits ignoriert, sodass *DACHI* und *DACLO* wie 8 Bit Register verwendet werden.

Anhand obiger Amplitudenbedingung kann die erforderliche Feldstärke B_1 für gegebene Resonatorabmessungen und Selektorfeldstärke unmittelbar berechnet werden. Um zum Beispiel mit dem MONOPOL-Resonator Neutronen mit einer Wellenlänge von 2 Å auszuwählen, wird ein Selektorfeld von $3.39\,\mu$T sowie ein Resonatorfeld B_1 von $110.99\,\mu$T benötigt.

Da nur ganzzahlige Vielfache der minimalen Schrittweite für den Strom des DACs eingestellt werden können, wird so immer ein gewisser kleiner Fehler auftreten. Um den Strom genau einstellen zu können, müsste der Wert 2407.59 an den größeren DAC gesendet werden (dies entspricht ca. 11.75 A), mit dem nächsthöheren Wert 2408 entsteht so ein relativer Fehler von 1.7×10^{-4}.

Die minimale Wellenlänge, welche ausgewählt werden kann, wird dabei von der technisch maximal möglichen Feldamplitude $B_{1,\max}$ bestimmt. Für MONOPOL entspricht diese einer minimalen Wellenlänge von circa 0.94 Å.

Im Codebeispiel wird nur ein Strom von knapp 1 A eingestellt, dazu wird als höheres Byte eine 50 an den DAC für die Ströme bis 5 A geschickt, alle restlichen Bytes haben den Wert Null.

Nachdem die Kommunikation mit den DACs abgeschlossen wurde, wird die Datenlinie für das Shiftregister nun auf 0 gesetzt. Da nur eine einzige 1 durch das Register geschoben werden soll, wird diese auch für den Rest des Programms LOW bleiben. Daraufhin wird auch die Uhr zunächst auf LOW gesetzt.

Nun wird der Zähler *clockcounter* dekrementiert, der darauf folgende Befehl *bra Z,1f* sorgt dafür, dass nach zum Marker *1* nach vor gesprungen wird, wenn alle 48 Stromversorgungen korrekt eingestellt sind. Ist dies noch nicht der Fall, so wird der Befehl *goto I2C_Loop* ausgeführt und wieder am Anfang der Schleife gestartet und die DACs der nächsten Stromversorgung angesteuert.

Sobald die Schleife 48 Durchläufe beendet hat, springt das Programm aus der Schleife, danach wird der Taktgeber des Shiftregisters noch einmal auf HIGH gesetzt, um die 1 vollständig aus dem Shiftregister zu schieben. Nach einer kurzen, mittels 10 *nop*-Befehlen realisierten Stabilisationszeit, wird die Uhr wieder auf LOW gesetzt.

Nun sollen noch die Subroutinen genauer betrachtet werden, dazu wird als Beispiel die Subroutine *DAC0Write* gezeigt, die bis auf die andere Adresse des DACs ident ist mit der Subroutine *DAC1Write*.

Die I^2C-Kommunikation läuft in diesem Fall wie folgt ab: Nach dem Startbefehl wird zunächst ein Adressbyte gesendet, die Komponente mit der passenden Adresse meldet sich dann per Handshake und besetzt den Datenbus, keine andere Komponente kann dann darauf zugreifen. Daraufhin wird ein Konfigurationsbyte versendet, welches wiederum mit einem Handshake

quittiert wird. Danach wird das erste Datenbyte verschickt und nach einem weiterem Handshake das zweite. Daraufhin wird ein Stopbefehl verschickt und der Datenbus wieder freigegeben.

```
DAC0Write:
;start I2C
    bset LATB,#7        ;unlock writing to the DAC


    bset I2C2CON,#SEN       ;send START condition
    call WaitI2C2
```

Zunächst wird der Port B7 auf HIGH gesetzt, dieser ist mit dem Eingang I2C_DAC_EN aller Stromversorgungen verbunden und ermöglicht eine Kommunikation mit den darauf verbauten DACs. Danach wird der Startbefehl gegeben, dies wird durch das Setzen des SEN-Bits im Kontrollregister *I2C2CON* durchgeführt, die Subroutine *WaitI2C2* sorgt dafür, dass abgewartet wird, bis ein das I^2C 2-Modul betreffender Vorgang fertig durchgeführt wird bevor das Programm weiterläuft.

```
    mov #0b00100100,W0      ;send address byte to write
    mov W0,I2C2TRN      ;move to transmit Register

1:
    btsc I2C2STAT,#TBF          ;wait for Conversion
    bra 1b

    call ACKPoll2
    call WaitI2C2
```

Nun wird das Adressbyte versendet, dieser DAC hat die 7-Bit Adresse '0010010', das achte Bit signalisiert, ob man etwas an die Komponente schicken (0) oder diese auslesen (1) will. Da in vorliegenden Fall gesendet werden soll, wird eine Null angehängt. Über das Arbeitsregister W0 wird das Byte in das Transferregister *I2C2TRN* kopiert. Sobald etwas in dieses Register kopiert wird, wird die TBF-Flag im Kontrollregister *I2C2STAT* gesetzt, ist das Byte fertig übertragen worden wird diese Flag wieder gelöscht.

Diese Flag wird benützt, um abzuwarten, bis das Byte fertig übertragen wurde bevor das Programm weiter fortschreitet. Erst wenn die TBF-Flag wieder 0 ist, wird mittels *btsc*(*bit test skip if clear*)-Befehl der *bra 1b* Befehl übersprungen, sollte die Kommunikation jedoch noch andauern, so wird nur wieder vor den *btsc*-Befehl zurückgesprungen.

Die Subroutine ACKPoll2 überprüft nun den Handshake nach der Übertragung eines Bytes, die Komponente sendet nach einer Datenübertragung

entweder ein ACK (Acknowledge) oder ein NACK (Not Acknowledge) zurück. Die Subroutine WaitI2C2 sorgt dafür dass abgewartet, wird bis dieser Handshake fertig abgewickelt wird bevor weiter fortgeschritten wird.

```
    mov #0b00110000 ,W0        ;write to and power up DAC
    mov W0,I2C2TRN             ;Send it

1:
    btsc I2C2STAT,#TBF                ;wait for Conversion
    bra 1b

    call ACKPoll2
    call WaitI2C2
```

Nun wird das Konfigurationsbyte versendet, welche Möglichkeiten es dabei gibt, kann in [19] nachgelesen werden. Hier wird ausgewählt, dem DAC neue Werte zu senden und ihn danach in Betrieb zu versetzen. Die Prozedur ist dabei wie bei jedem anderen Datenbyte, das versendet werden soll: Zunächst wird dieses in das Transferregister kopiert, danach abgewartet, bis dieses wieder leer ist, dann der Handshake mittels *ACKPoll2* durchgeführt und mit *WaitI2C2* abgewartet, bis dieser fertig abgewickelt ist.

```
    mov DACHI, W0             ;Load second byte which contains upper 8 bits
    mov W0,I2C2TRN             ;Send it
1:
    btsc I2C2STAT,#TBF                ;wait for Conversion
    bra 1b

    call ACKPoll2
    call WaitI2C2
```

Nun wird das erste Datenbyte versendet, im Hauptprogramm wurde dabei dafür gesorgt, dass der gewünschte Inhalt schon im Register *DACHI* abgespeichert wurde, dieses wird nun wie jedes andere Byte an den DAC gesendet und mit Handshake quittiert.

```
    mov DACLO, W0             ;Load third byte which contains lower 4 bits
    mov W0,I2C2TRN            ;Send it
1:
    btsc I2C2STAT,#TBF                ;wait for Conversion
    bra 1b

    call ACKPoll2
    call WaitI2C2
```

Nun noch das zweite Datenbyte, da der DAC im 12-Bit Modus agiert, sind dabei nur die ersten 4 Bits von Bedeutung. Das Register *DACLO* wurde ebenfalls im Hauptprogramm schon auf den korrekten Wert eingestellt und das Byte wird wie bisher übertragen.

```
bset I2C2CON,#PEN          ;Send STOP
call WaitI2C2

bclr LATB,#7          ;lock DAC again
return
```

Alle gewünschten Daten wurden dem DAC nun gesendet, nun wird noch der STOP-Befehl für die I^2C-Übertragung gesendet werden. Dies geschieht, indem das PEN-Bit des I2C2CON Registers gesetzt und mittels der *WaitI2C* Subroutine nun wieder abgewartet wird, bis der Stoppbefehl verarbeitet wurde. Nachdem dies geschehen ist, wird der Datenbus vom DAC wieder freigegeben und nachdem ein weiterer Startbefehl übermittelt wird, wird das nächste versendete Byte wieder als Adressbyte interpretiert und eine andere Komponente kann beschrieben oder ausgelesen werden.

Bevor mittels *return* zurück in das Hauptprogramm gesprungen wird, wird nun noch I2C_DAC_EN wieder auf LOW gesetzt, um die I^2C-Kommunikation von den DACs abzuschneiden.

Nun werden noch die hier benützen Subroutinen *WaitI2C2* und *ACKPoll2* kurz besprochen:

```
WaitI2C2:
1:
    btss IFS3,#MI2C2IF        ;check interrupt flag
    bra 1b
    bclr IFS3,#MI2C2IF
    return
```

Die Subroutine *WaitI2C2* besteht dabei aus einer Schleife, welche überprüft, ob die Master I2C2 Interrupt Flag (MI2C2IF) gesetzt wurde. Diese wird gesetzt, sobald ein Byte erfolgreich an eine Komponente übertragen oder von dieser empfangen wurde.

Sobald dies der Fall ist, überspringt der *btss IFS3,#MI2C2IF*-Befehl den *bra*-Befehl, welcher in der Schleife zurückspringt, daraufhin muss nur noch die Flag gelöscht werden und es kann mittels *return*-Befehl in das Hauptprogramm zurückgekehrt werden.

```
ACKPoll2:
    1:
    btsc I2C2STAT,#ACKSTAT   ;ACK? if so, skip next
```

```
bra 1b          ;NACK, poll again
return
```

Ähnlich verhält es sich mit der Subroutine *ACKPoll2*, hier wird das Bit ACKSTAT überprüft, dieses wird auf 0 gesetzt, sobald ein ACK (Acknowledge) von der Komponente gesendet wurde. Ist dies der Fall, wird die Schleife verlassen.

Hier ist zu beachten, dass diese Funktion davon ausgeht, dass die Komponente auch wirklich ein ACK und kein NACK (Not Acknowledge) senden wird, die Kommunikation also funktioniert. Sendet die Komponente ein NACK so wird die Schleife nie verlassen und das Programm muss händisch neu gestartet werden.

Dies ist für kleinere Versuche ausreichend, bei denen ein händischer Neustart nicht mit gröberen Komplikationen verbunden ist. In weiterer Folge kann angedacht werden, eine extra Routine bei einem NACK aufzurufen, die entsprechend reagiert, und etwa den Strom langsam herunter fährt, um dann überprüfen zu können, wo das Problem liegt.

3.3.3 UART-Kommunikation

Nun soll noch beschrieben werden, wie das UART-(Universal Asynchronous Receiver Transmitter) Modul des Mikrocontrollers verwendet werden kann, um mittels seriellem Port den Ablauf des Programms kontrollieren zu können. Für die Übertragung wird der Industriestandard RS485 verwendet, bei dem auf zwei Datenlinien das Signal einmal normal und einmal invertiert übertragen wird, dies erhöht die elektromagnetische Verträglichkeit des Systems.

Das folgende Codebeispiel zeigt, wie das Programm auf eine Datenübertragung des seriellen Ports reagieren kann und dabei zwischen unterschiedlichen gesendeten Zeichen unterscheidet. Das Graphical User Interface (GUI) wird von dieser Art der Datenübertragung Gebrauch machen, sobald das Assemblerprogramm die Initialisierung abgeschlossen hat, wird es auf eine Datenübertragung über diesen seriellen Port warten. Je nachdem welche Daten empfangen wurden, können unterschiedliche Schritte eingeleitet werden, so können etwa neue Informationen über Timing, Muster, oder auch eine neue Stromstärke übermittelt werden oder auch der Wunsch, die unterschiedlichen I^2C-Komponenten auszulesen.

Dabei können auch Daten vom Mikrocontroller zum GUI versendet werden, entweder, um Messdaten zu übermitteln oder auch um einen Handshake zu ermöglichen, nachdem Daten erfolgreich empfangen wurden.

Nun soll in einem kurzen Beispiel gezeigt werden, wie das UART-Modul initialisiert werden kann, um danach auf eine Datenübertragung zu warten. Sobald dann ein Byte über den Port versendet wurde, können, je nachdem

was übertragen wurde, unterschiedliche Maßnahmen ergriffen werden. Danach wird ein Byte zurückgesendet, um zu signalisieren, dass die Übertragung korrekt verarbeitet wurde.

Initialisierung

```
init_uart:
   bclr LATB,#6          ;positive enable for UART-transmission
   bclr LATB,#5          ;negative enable for UART-reception
```

Mit dem Marker *init_ uart* beginnt die Initialisierung des UART-Moduls. Die beiden Ausgänge B5 und B6 ermöglichen je nach ihrem Status entweder das Empfangen oder das Versenden mittels UART-Modul. Der Ausgang B5 ermöglicht dabei das Empfangen, wenn er Ausgang auf LOW gesetzt wird, Ausgang B6 dagegen ermöglicht das Versenden mit dem UART-Modul, wenn er auf HIGH gesetzt wird. Da zunächst etwas empfangen werden soll, werden daher beide Ausgänge auf LOW gesetzt.

```
   clr U1STA        ;clear UART(Transmit&Receive)-
                    ;-Status and Control Register

   BCLR IEC0,#U1RXIE ;disable UART reception Interrupt
   BCLR IEC0,#U1TXIE ;disable UART transmission Interrupt
   BCLR IEC4,#U1ERIE ;disable UART Error Interrupt
```

Danach wird der Inhalt des Statusregister des UART-Moduls U1STA zunächst gelöscht, mit diesem können einige Einstellungen bezüglich des Modus (8- oder 9-Bit, Parität, etc.) eingestellt werden, die genauen Einstellungen finden sich in [15]. Der Pin UTXEN des Registers muss in weiterer Folge noch auf HIGH gesetzt werden, um eine serielle Übertragung zu ermöglichen, dies geschieht allerdings erst später.

Es gibt drei Arten von Interrupts, die mit dem UART-Modul verbunden sind: Einen für das Empfangen eines Bytes, einen für das Versenden eines Bytes, und einen wenn ein Fehler auftritt.

Hier ist nicht erwünscht, dass einer dieser Interrupts einen Sprung aus dem Programm in die Interruptserviceroutine verursacht, deswegen werden die dazugehörigen *Enable*-Bits, die sich in den Kontrollregistern für die Interrupts IEC0 (Interrupt Enable Control) und IEC4 befinden, allesamt auf 0 gesetzt. Jedoch werden die dazugehörigen Flags für die Interrupts in jedem Fall gesetzt und können dazu benutzt werden, um den Programmablauf zu steuern, wie dies auch in diesem Beispiel an späterer Stelle geschehen wird.

```
   mov #47, W0       ;baudrate 19200=47,38400=23,57600=15,115200=7
   mov W0, U1BRG     ;set up baud rate register
```

```
bset U1MODE,#UARTEN        ;Set UARTEN Pin
nop
nop
nop
bset U1STA,#UTXEN          ;Set UTXEN Pin
```

Nun wird noch die Baudrate für die serielle Datenübertragung eingestellt, deren Berechnung in [15] beschrieben wird. Hier wird eine Baudrate von 19200 Symbolen pro Sekunde eingestellt.

$$U1BRG = \frac{F_{CY}}{16 \times BaudRate} - 1 \ ,$$

wobei F_{CY} die Frequenz ist, mit der eine Instruktion verarbeitet werden kann und damit exakt der halben Frequenz des Oszillators entspricht. Der errechnete Wert, in diesem Fall 47, wird dann in das für die Baudrate zuständige Register *U1BRG* kopiert. Daraufhin wird zunächst das UARTEN-Bit des Kontrollregisters *U1MODE* gesetzt und danach, nach einigen Zyklen Pause durch *nop*-Befehle, das #UTXEN Bit des Kontrollregisters *U1STA*. Nachdem diese beiden Bits in dieser Reihenfolge aktiviert wurden, ist das UART-Modul aktiv.

Abfrage durch Subroutine

Nun ist das UART-Model korrekt initialisiert, um über den seriellen Port Daten empfangen zu können. Hier wird nun als Beispiel eine Abfrage durchgeführt, welche unterschiedliche Bits eines Registers setzt, je nachdem welche Daten empfangen wurden. Durch eine Abfrage mit der Subroutine *UART_Check*, welche der Bits gesetzt wurden, können daraufhin unterschiedliche Teile des Programms gestartet werden.

```
UART_Check:
    clr UART_flag

2:
    btss U1STA,#URXDA      ;look if character received
    goto 2b
```

Am Anfang wird hierbei der Inhalt des Registers UART_flag gelöscht, dabei handelt es sich um das Register, bei welchem je nach empfangenen Daten unterschiedliche Bits gesetzt werden. Um bei einem mehrmaligem Aufrufen dieser Subroutine nicht die Bits, die davor gesetzt wurden 'mitzuschleppen', wird dieses register am Anfang gelöscht.

Daraufhin folgt die erste Schleife, die zunächst nur abfragt, ob überhaupt ein Zeichen über den seriellen Port empfangen wurde. Dafür wird

das URXDA-Bit des Statusregisters U1STA für das UART-Modul abgefragt. Sobald ein Zeichen empfangen wurde, wird dieses Bit gesetzt und der *btss*-Befehl überspringt den *goto 2b* Befehl. Ansonsten verbleibt das Programm solange in dieser Schleife, bis ein Zeichen empfangen wurde

```
mov U1RXREG,W0          ;read RXREG
mov W0, buffer
```

Nun wird der Inhalt von U1RXREG, in dem das empfangene Zeichen gespeichert wird, in das Arbeitsregister W0 kopiert. Da in weiterer Folge mittels einer XOR(*Exclusive Or*)-Abfrage festgestellt wird, welches Zeichen empfangen wurde, und diese Abfrage auch den Inhalt des Arbeitsregisters selbst verändern kann (und in den meisten Fällen wird), wird sein Inhalt zusätzlich noch in das Register *buffer* kopiert.

```
xor #97,W0          ;is it 'a'?
bra NZ,1f           ;if not, check next character
bset UART_flag,#0     ;set flag bit
return
```

Nun erfolgt die Abfrage, welches Zeichen verschickt wurde. Hierbei wird die Ascii-Codierung verwendet, bei welcher der Dezimalwert 97 das Zeichen 'a' codiert Nun wird mittels XOR-Abfrage überprüft, ob das in W0 gespeicherte Zeichen einem 'a', also dem Wert 97 entspricht. Dabei wird der Inhalt des Registers W0 wie folgt verändert: Haben zwei Bits der miteinander verglichenen Register denselben Wert, so wird dieses Bit in W0 auf 0 gesetzt, besitzen sie hingegen einen unterschiedlichen Wert, so wird es auf 1 gesetzt. Sind die Werte in den beiden verglichenen Registern identisch, so wird jedes Bit im Register W0 auf 0 gesetzt und dabei auch die Zero-Flag gesetzt. Die darauf folgende Abfrage *bra NZ,1f* springt zum nächsten *'1'*-Marker wenn das Ergebnis der letzten Operation des *xor*-Befehls, nicht 0 war, also die Inhalte der verglichenen Register nicht identisch waren, worauf die nächste Abfrage durchgeführt wird.

Ansonsten, also wenn die beiden Operanden gleich waren, wird das nullte Bit des Registers UART-flag gesetzt und mittels *return*-Befehl wieder aus der Subroutine herausgesprungen.

```
1:
mov buffer, W0
xor #98,W0          ;is it a 'b'?
bra NZ,1f           ;if not, check next character
bset UART_flag,#1     ;set flag bit
return

1:
```

```
    bset UART_flag,#7   ;it's something else
    return
```

Bevor die nächste Abfrage durchgeführt werden kann, wird nun der Inhalt des Bufferregisters in das Arbeitsregister W0 kopiert, da dessen Inhalt (höchstwahrscheinlich) vom *xor*-Befehl verändert wurde, das Register *buffer* dagegen enthält noch den unveränderten Inhalt der seriellen Übertragung. Danach kann wieder mittels *xor*-Befehl untersucht werden, ob es sich um das erwartete Zeichen handelt und ein anderes Bit des Registers gesetzt werden. Dieser Block kann, je nachdem wie viele unterschiedliche Zeichen abgefragt werden sollen, beliebig oft mit anderen abgefragten Werten wiederholt werden.

Nachdem so viele Zeichen wie gewünscht abgefragt wurden, wird am Ende der Subroutine noch das 7. Bit des UART-flag Registers gesetzt. Da das Programm bei einer gelungenen Abfrage schon davor mittels *return* aus der Subroutine springt, ist der Zustand dieses Bits nun ein Marker dafür, ob das übertragene Zeichen eines der abgefragten Zeichen ist.

Hauptprogramm mit Handshake

Nun soll gezeigt werden, wie mittels dieser Subroutine je nach empfangenem Zeichen unterschiedliche Programmabschnitte aufgerufen werden können. In diesem Fall werden als Beispiel nur unterschiedliche LEDs, die sich auf der Steuerplatine befinden, ein- und ausgeschaltet, dies kann jedoch durch beliebigen anderen Code ersetzt werden.

Dazu wird nachdem ein Zeichen über den seriellen Port empfangen wurde, ein Zeichen vom Mikrocontroller über den seriellen Port versendet, um zu signalisieren, dass die zuvor durchgeführte Übertragung erfolgreich war.

```
UART_Loop:

    call UART_Check
    bset LATB,#6          ;positive enable for UART_transmission
    bset LATB,#5          ;negative enable for UART_reception

   mov #65,W0
   mov W0,U1TXREG

1:
   btss U1STA,#TRMT       ;last transmission has completed?
   goto 1b
```

Zu Anfang wird die vorhin erwähnte Subroutine *UART_Check* aufgerufen, diese wartet auf eine Datenübertragung über den seriellen Port und setzt

je nach empfangenem Zeichen entsprechende Bits im Register *UART_flag*. Nachdem ein Zeichen empfangen wurde, soll nun eine Rückmeldung gegeben werden und ein Zeichen über den seriellen Port verschickt werden, dazu werden zunächst die beiden Pins B5 und B6 auf HIGH gesetzt. Dies ermöglicht dem Mikrocontroller das Versenden von Zeichen über das UART-Modul.

Danach wird über das Arbeitsregister W0 der zu übertragende Wert - in diesem Fall 65, dies ist ein 'A' im ASCII-Code – in das Register *U1TXREG* kopiert. Wird ein Wert in dieses Register kopiert, so wird dieser über das UART-Modul versendet, dabei wird nach einer erfolgreichen Übertragung das TRMT-Bit des Statusregisters *U1STA* gesetzt. Um erst im Code fortzufahren, wenn die Übertragung beendet wurde, wird dieses Bit abgefragt. Erst wenn es gesetzt wurde, springt der *btss*-Befehl über den *goto 1b*-Befehl und aus der Schleife.

```
btst UART_flag,#0
bra Z,1f
btg LATB,#7

1:
btst UART_flag,#1
bra Z,1f
btg LATB,#14

1:
```

Nun werden die Abfragen vorgenommen, je nach empfangenem Zeichen wurde in der Subroutine *UART_Check* entweder das nullte oder erste Bit des Registers *UART_flag* gesetzt. Mittels *btst*-Befehl werden diese Bits nacheinander abgefragt. Wenn das Bit nicht gesetzt ist, wird mittels *bra Z,1f* zur nächsten Abfrage gesprungen, ansonsten wird der nächste Befehlsblock ausgeführt.

Da in diesem Fall nur eine Befehlszeile ausgeführt wird, das Ändern des Zustands von Pin B7 bzw. B14, welche je mit einer LED verbunden sind, könnte in diesem Fall auch der Befehl *btss UART_flag,#0/#1* verwendet werden. Dieser würde dann genau das Ein- bzw. Ausschalten der LED überspringen, wenn das dazugehörige Bit nicht gesetzt wurde. Allerdings kann auf die hier im Beispiel verwendete Weise anstatt nur einer Zeile ein beliebig langer Befehlsblock eingeschoben werden, was für eine weiterführende Anwendung natürlich besser geeignet ist.

```
bclr LATB,#6          ;positive enable for UART-transmission
bclr LATB,#5          ;negative enable for UART-reception
clr U1TXREG
goto UART-Loop
```

Bevor nun wieder an den Anfang der Schleife zurückgesprungen wird um auf die nächste Datenübertragung zu warten, werden Pin B5 und B6 wieder auf LOW gesetzt, dies ermöglicht das Empfangen von Daten über den seriellen Port. Daraufhin wird noch der Inhalt des Registers U1TXREG gelöscht, über welches die Daten über das UART Modul versendet werden können und es wird wieder an den Anfang der Schleife gesprungen.

Diese serielle Kommunikation mit dem UART-Modul wird dazu benützt werden, den Resonator auf den gewünschten Betrieb einzustellen, sowie gewisse Messdaten auszulesen. Es können unter anderem

- die gewünschten Stromwerte für die Resonatorspulen an den Mikrocontroller übermittelt werden, welche dieser abspeichert und dann mittels I^2C-Kommunikation an die Stromversorgungen schickt. Dies legt die Stärke des Resonatormagnetfelds fest und damit die selektierte Wellenlänge.

- der gewünschte Modus des Resonators, Permanent- oder Wanderwellen-Modus, eingestellt werden. Beim Wanderwellenmodus wird dann auch noch die Information an den Mikrocontroller gesendet, wie breit der gewünschte monochromatische Neutronenpuls sein soll.

- vom Mikrocontroller mittels I^2C-Kommunikation ausgelesene Werte, wie die Temperatur an den Stromversorgungen oder die an den Spulen gemessene Spannung, empfangen werden.

Kapitel 4

Durchgeführte Tests an Stromversorgung und der Steuerung

Mittels unterschiedlicher Aneinanderreihung der im vorigen Kapitel besprochenen Befehlsblöcke, in je nach Situation leicht abgeänderter Form, ist es nun möglich gewesen unterschiedliche Tests der Steuerungsplatinen durchzuführen.

Dabei wurden zunächst die CPLDs selbst auf ihre Funktion als Shiftregister geprüft, bevor danach auch explizit mittels dieser Shiftregister der Schaltvorgang zwischen Artificial und Real Load vorgenommen wurde, zunächst bei einer Stromversorgung und in weiterer Folge auch mit beiden zum Zeitpunkt der Tests fertiggestellten Stromversorgungen.

4.1 Test der CPLD-Shiftregister

Zunächst wurde getestet, ob die in die CPLDs einprogrammierten Shiftregister auch wie gewünscht ein beliebiges Muster durchschieben und dieses an den Ausgängen korrekt ausgeben.

In diesem Fall wurden nur die digitalen Ausgänge der Shiftregister überprüft, welche sich auf der Steuerplatine befinden. Diese Signale werden wie in Abb. 4.1 zu sehen über zwei Konnektoren ausgegeben. Diese werden im fertigen Zustand über ein Kabel mit je einer der Backplanes verbunden. Folglich laufen über den einen Konnektor die Signale, welche auf der einen Seite für die ungerade nummerierten Stromversorgungen (die in Strahlrichtung 1.,3.,5. etc.) auf Artificial oder Real Load schalten, und über den anderen Konnektor jene für die geradzahlig nummerierten.

Der Zustand der Ausgänge wurde dabei mittels des Saleae Logic Pro 16 USB Logic Analyzer [20] untersucht. Mit diesem können bis zu 16

Ausgänge gleichzeitig über einen gewissen, vorher einstellbaren Zeitraum überwacht werden. Dabei lassen sich sowohl digitale, als auch analoge Signale überwachen. Im vorliegnden Fall können die Outputs der Shiftregister nur 0 oder 5 V ausgeben, es wird also im digitalen Modus gearbeitet, in welchem das Signal mit bis zu 50 Millionen Samples pro Sekunde abgetastet werden kann.

Mittels USB-Kabel kann der Logic Analyzer mit einem PC verbunden werden und mit dem dazugehörigen Programm Salae Logic (hier wurde die Version 1.2.12 verwendet) kann der gewünschte Modus (Digital, Analog), die Anzahl der Datenlinien, die Sampling-Rate sowie der Zeitraum, über welche die Samples genommen werden sollen, eingestellt werden.

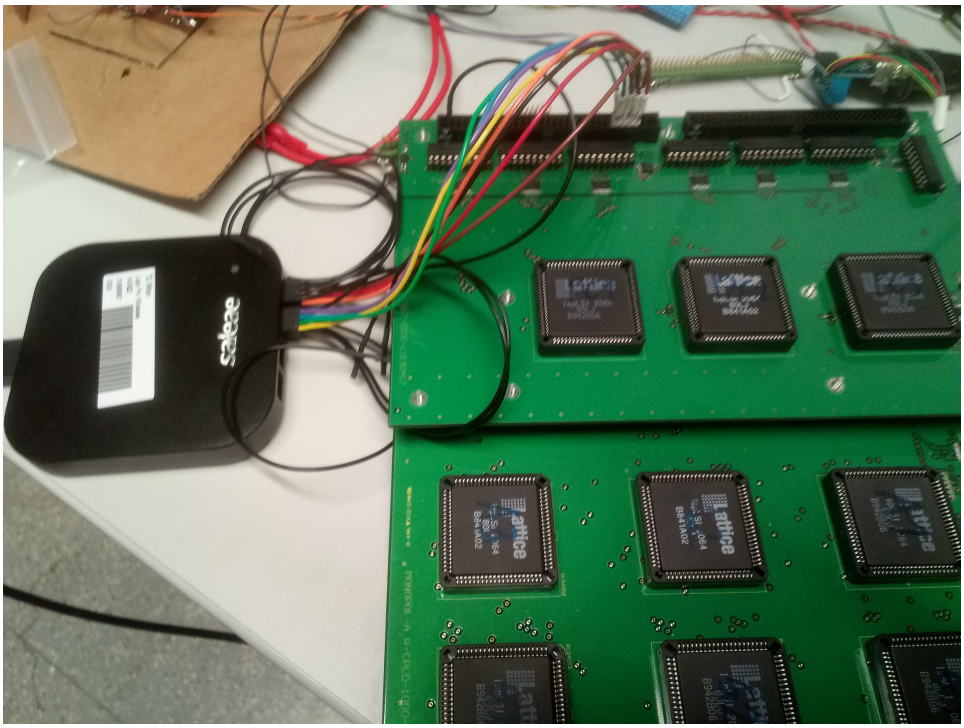

Abbildung 4.1: Versuchsaufbau zur Untersuchung der Signallinien, welche den Schaltvorgang steuern, mit Hilfe eines Logik Analysators.

Um die korrekte Funktion der Shiftregister zu überprüfen, wurde nun vom Mikrocontroller ein einfaches Muster gesendet, welches immer abwechselnd zwei Nullen und danach zwei Einser sendete, der Logic Analyser wurde dann an den unterschiedlichen Ausgängen der Shiftregister angebracht, um das von den Shiftregistern ausgegebene, parallele Signal zu überprüfen. Der Schaltvorgang lief dabei vom Programm her ab wie in Abschnitt 3.3.1 beschrieben, die Ergebnisse sind in Abb. 4.2 zu sehen.

Es wurde dabei eine Samplerate von 5 MHz im digitalen Modus für 2 Sekunden abgegriffen, der Timerinterrupt der den Schaltvorgang über die Änderung der dazugehörigen Uhren der Shiftregister verursachte, wurde alle 25 μs ausgelöst. Die Zeit, in der mittels Glitch sowohl über Artificial als auch über den Real Load geleitet wurde, betrug 0.4 μs.

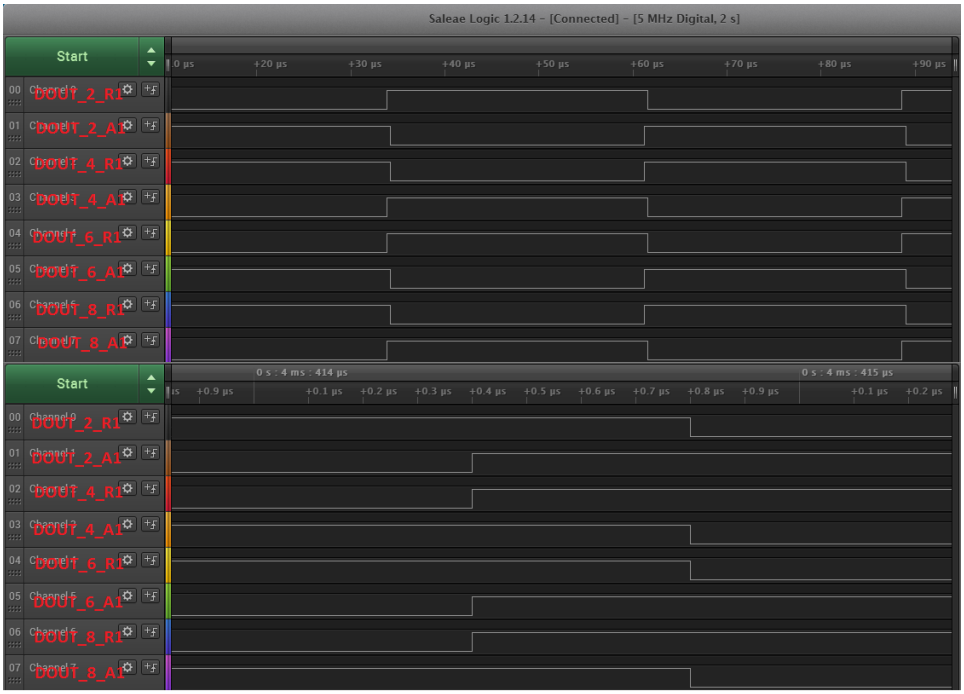

Abbildung 4.2: Der Schaltvorgang zwischen Artificial und Real Load in der Ansicht des Programms Salae Logic. Hierbei wurden die ersten vier Ausgänge der Artificial und Real Datenlinien auf dem Konnektor für die gerade nummerierten Stromversorgungen untersucht. *Oben:* Aufnahme über mehrere regelmäßige Perioden. *Unten:* Besonderer Fokus auf dem Schaltvorgang zwischen Artificial und Real Load.

Wie in den Abbildungen zu sehen ist, funktioniert der Schaltvorgang wie gewünscht. Um die Probleme zu verdeutlichen, die auftreten, wenn der Schaltvorgang ohne diesen Glitch abläuft, wurde auch noch ein Test durchgeführt, bei dem die Uhren des Artificial und Real Shiftregisters einen Offset von 0.4 μs haben und der Glitch nicht verwendet wird. Dieser Test ist in Abb. 4.3 zu sehen.

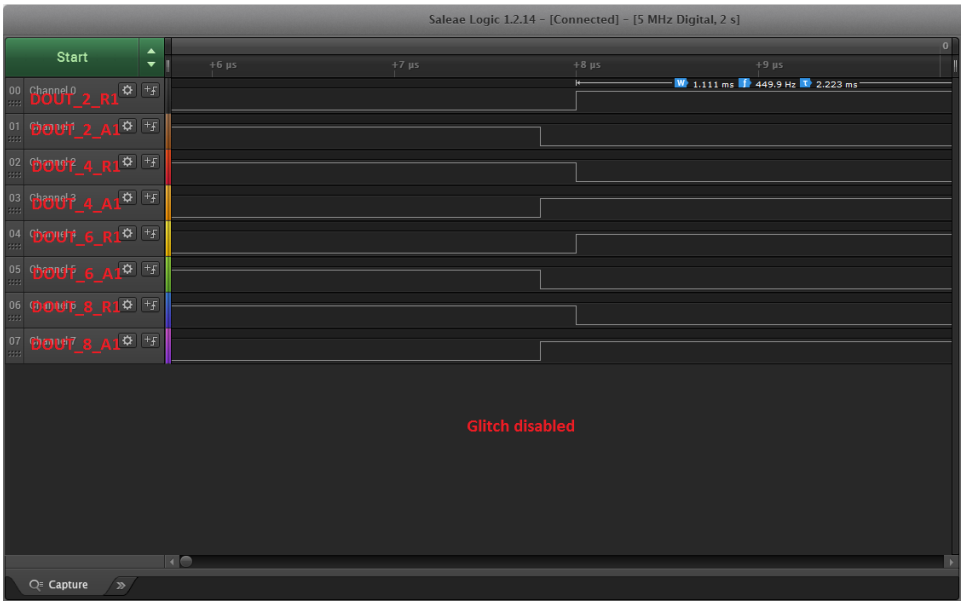

Abbildung 4.3: Der Schaltvorgang zwischen Artificial und Real Load mit Offset zwischen den Uhren der Artificial und Real Shiftregister.

In diesem Fall wird die Uhr, die den Takt für das Shiftregister für den Artificial Load vorgibt, 400 ns vor der anderen Uhr geschaltet, was dazu führt, dass bei einem Wechsel von Artificial zu Real Load der Schaltvorgang wie gewünscht vor sich geht, der Strom für diese Zeit also über beide Loads fließt. Das Problem tritt beim Wechsel von Artificial und Real Load auf, bei dem kurzzeitig über keinen der beiden Loads geleitet wird. Schaltet man hingegen die Uhr für den Real Load zuerst, erhält man das genau gegenteilige Verhalten.

Da die Uhren jede Zelle des Shiftregisters parallel ansteuern, ist auch ein je nach Situation abwechselndes Vorziehen der gerade passenden Uhr keine Option, auch hier wird es bei einem der beiden Schaltvorgänge immer Probleme geben.

4.2 Tests mit zwei Stromquellen

Nun sollten nicht nur die digitalen Signale, welche den Schaltvorgang bestimmen,überwacht werden, sondern der tatsächlich von der Stromversorgung kommende Strom über die Widerstände gemessen werden. Der Versuchsaufbau mit den zwei zum Zeitpunkt dieses Tests fertiggestellten Stromversorgungen ist in Abb. 4.4 dargestellt.

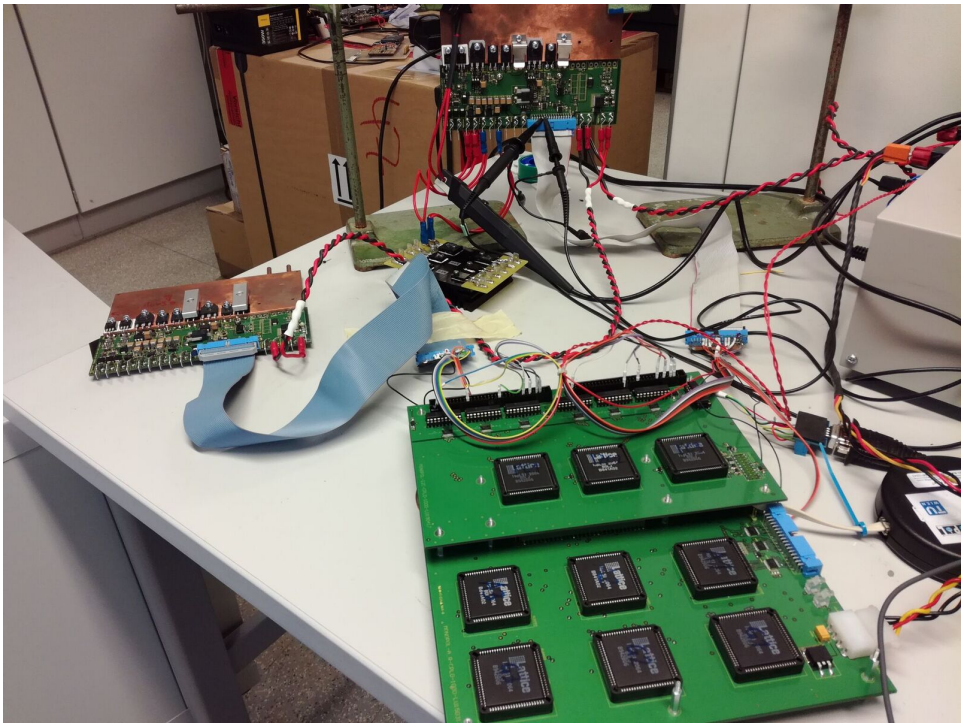

Abbildung 4.4: Versuchsaufbau für die Tests an zwei Stromquellen. Unten die für die Steuerung notwendigen Platinen mit den CPLD-Shiftregistern, oben die beiden Stromversorgungen, an denen getestet wurde. Da die Backplanes zu diesem Zeitpunkt noch nicht fertiggestellt waren, wurden die wichtigen Signallinien per händisch verlöteter Kabel vom Konnektor zu den Stromversorgungen verlegt.

Das für diese Tests benützte Programm wurde wie folgt aufgebaut: Direkt nach Start des Programms wurden per I^2C-Kommunikation die gewünschten Stromwerte eingestellt, dies geschah wie in Abschnitt 3.3.2 beschrieben, nur die Anzahl der Stromversorgungen, mit denen eine Kommunikation aufgenommen werden sollte, wurde dabei noch im Programmcode auf die zwei Vorhandenen verändert.

Daraufhin wird eine serielle Kommunikation mit dem UART-Modul vorgenommen, siehe Abschnitt 3.3.3. Das Programm wartet dabei darauf, dass

ein gewisses Zeichen über den seriellen Port empfangen wird. Sobald dieses Zeichen einlangt, wird ein Schaltvorgang vorgenommen. In dieser Schleife verbleibt das Programm dann bis zu einem Neustart, so kann Schritt für Schritt vom Benutzer ein Muster in der gewünschten Geschwindigkeit durchgeschoben werden und mittels Strommesszange der Stromverlauf sowie, falls gewünscht, noch mit dem Oszilloskop der Spannungsverlauf einiger Pins beobachtet werden.

Für die Messung des Stroms wurden zwei Strommesszangen verwendet, eine Tektronix TCP0030A [21] sowie eine Tektronix A6302 [22]. Das erste Modell ist dabei auf eine Verwendung mit dem hier verwendeten Tektronix MSO2000B Oszilloskop [23] ausgelegt und wird vom Oszilloskop auch beim dazugehörigen Kanal (Kanal 4 in Abb. 4.5) auch direkt in Ampere angezeigt. Beim anderen Modell wird dagegen nur die Hallspannung gemessen (Kanal 1 in Abb. 4.5), dabei können je nach Einstellung, 10 mV einer Stromstärke von 1-5 A entsprechen. In den hier vorgenommenen Messungen wurde dies stets so eingestellt, dass die beiden Kanäle dieselbe Skala besitzen, die Ampereanzahl für Kanal 4 ist also auch für Kanal 1 anwendbar.

Leider tritt bei diesen niedrigen Spannungen unvermeidlich ein sichtbares Hintergrundrauschen auf, auch das Zwischenschalten eines Rauschfilters konnte dieses nicht eliminieren. Für die Beobachtung des zeitlichen Ablaufs des Schaltvorgangs sind die Messergebnisse dennoch geeignet.

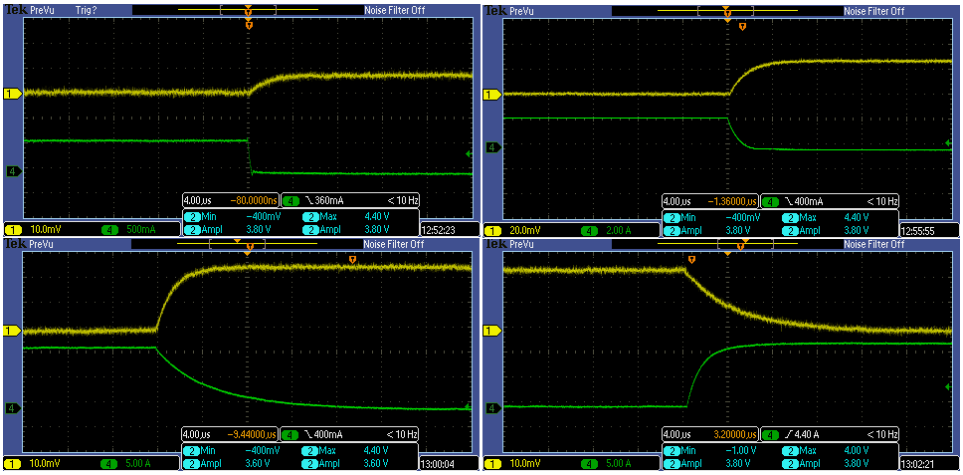

Abbildung 4.5: Aufnahmen des Schaltvorgangs mithilfe zweier Strommesszangen bei unterschiedlichen Stromstärken. Zu sehen sind die Schaltvorgänge (immer mit 600 ns Zeit, in der über beide Loads geleitet wurde) von Artificial Load (grün) auf Real Load (gelb) bei 0.5 A (oben links), 2.5 A (oben rechts), und 10 A (unten links), sowie der Schaltvorgang von Real Load auf Artificial Load bei 10 A (unten rechts). Die Zeit pro Skalenteilstrich beträgt 4 μs.

In Abb. 4.5 sind diese Schaltvorgänge bei unterschiedlichen Stromstärken zu sehen. Wie gut zu erkennen ist, können die gewünschten Stabilisationszeiten des Stroms von wenigen Mikrosekunden bei allen disen Stromstärken erreicht werden.

Neben dem Schaltvorgang von Artificial zu Real Load sollte auch noch untersucht werden, wie der zeitliche Verlauf des Stroms über zwei nacheinander angesteuerten Real Loads aussieht. Dazu wurde eine der Stromzangen bei der ersten Real Load angebracht, die andere bei der zweiten Real Load. Dieser Schaltvorgang ist in Abb. 4.6 zu sehen, hierbei wird gerade die erste Real Load abgeschaltet und das Muster gelangt gleichzeitig zur zweiten Real Load. Auch hier wurde dabei der Strom nur von der ersten Real Load auf die erste Artificial Load geschaltet und von der zweiten Artificial Load auf die zweite Real Load, da aber nur zwei Strommesszangen verfügbar waren, konnten allerdings nicht alle vier Ströme simultan gemessen werden.

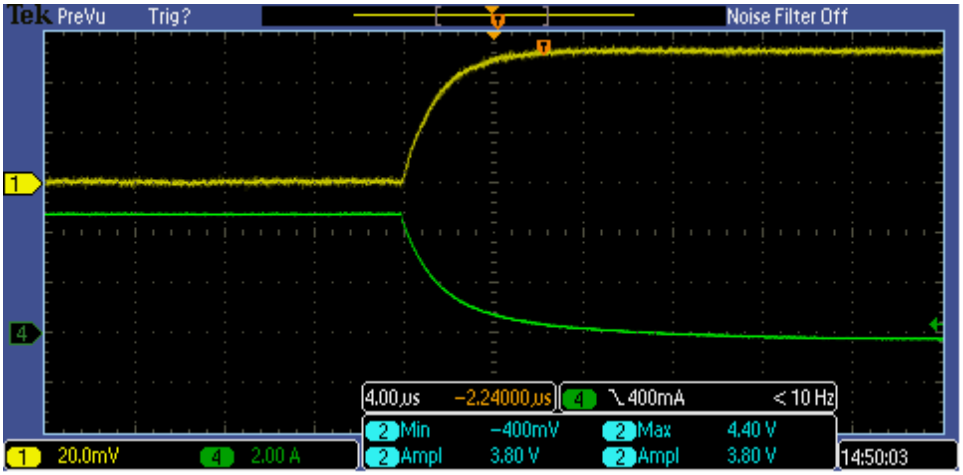

Abbildung 4.6: Messung des Schaltens von der ersten Real Load (grün) zur zweiten (gelb) bei 5 A.

Der Schaltvorgang verläuft vom Timing her wie erwartet und erfüllt die Anforderungen des Experiments an die Steuerung der Stromquellen, um ein Mitbegleiten eines Neutronenpakets durch den Resonator zu ermöglichen.

4.3 Tests mit längerem Kabel zur Überprüfung der Signalintegrität

Die bisherigen Tests wurden alle mit möglichst kurzen Kabeln durchgeführt, welche die digitalen Signale des Mikrocontrollers, über die der Schaltvorgang gesteuert wird, übertragen. Im Endzustand wird dies allerdings nicht möglich sein, jedes Signal wird zunächst mit einem längerem Kabel von der Steuerplatine zu der jeweiligen Backplane geleitet und muss dort dann auch noch

einen, von der Position der anzusteuernden Stromversorgung abhängigen, Weg zurücklegen.

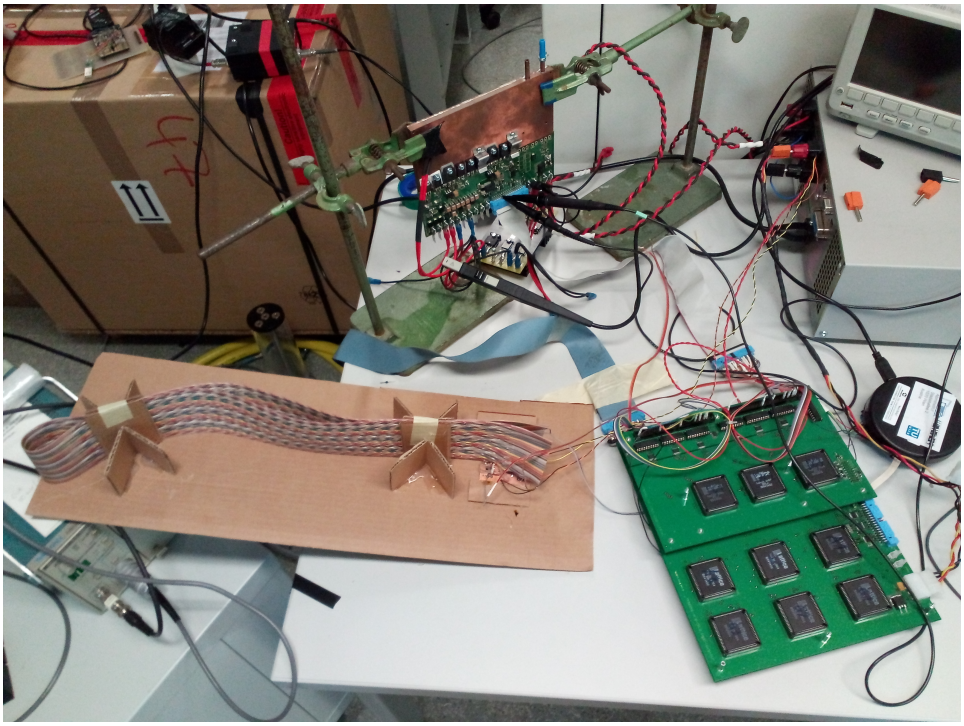

Abbildung 4.7: Aufbau zum Test der Signalintegrität über ein längeres Kabel. Das von der Steuerplatine (unten rechts) ausgegebene Signal gelangt zu der Platine, an der sowohl Sender als auch Empfänger der Signalverstärkung angebracht sind (unten links). Zwischen Sender und Empfänger wird das Signal noch über ein ca. 60 cm langes Kabel geleitet bevor es zur Stromversorgung (oben) gelangt.

Zur Stabilisierung des Signals wird Transistor-Transistor-Logik (TTL) verwendet. In ersten Tests mit einem Rechteckssignal über ein ca. 60 cm langes Kabel, siehe Abschnitt 2.2.3, liefert diese bereits hinreichend gute Ergebnisse. Nun soll neben diesem einfachen Rechteckssignal auch noch ein Schaltvorgang durchgeführt werden, um die Leistungsfähigkeit dieser Art der Signalverstärkung zu überprüfen. Der Versuchsaufbau ist dabei in Abb. 4.7 zu sehen. Da zum Zeitpunkt der Tests weder Verstärkerplatine noch Backplane fertiggestellt waren, wurde die Platine verwendet, welche bereits beim Test der unterschiedlichen Arten der Signalübertragung benützt wurde.

Für diese Tests konnte dasselbe Programm verwendet werden wie in Unterabschnitt 4.2, nur die Anzahl der angeschlossenen Stromversorgungen wurde dabei auf eins reduziert. Mit Hilfe von Oszilloskop und Strommesszange wurde der Stromverlauf über die Real Load, sowie die Signale, welche den

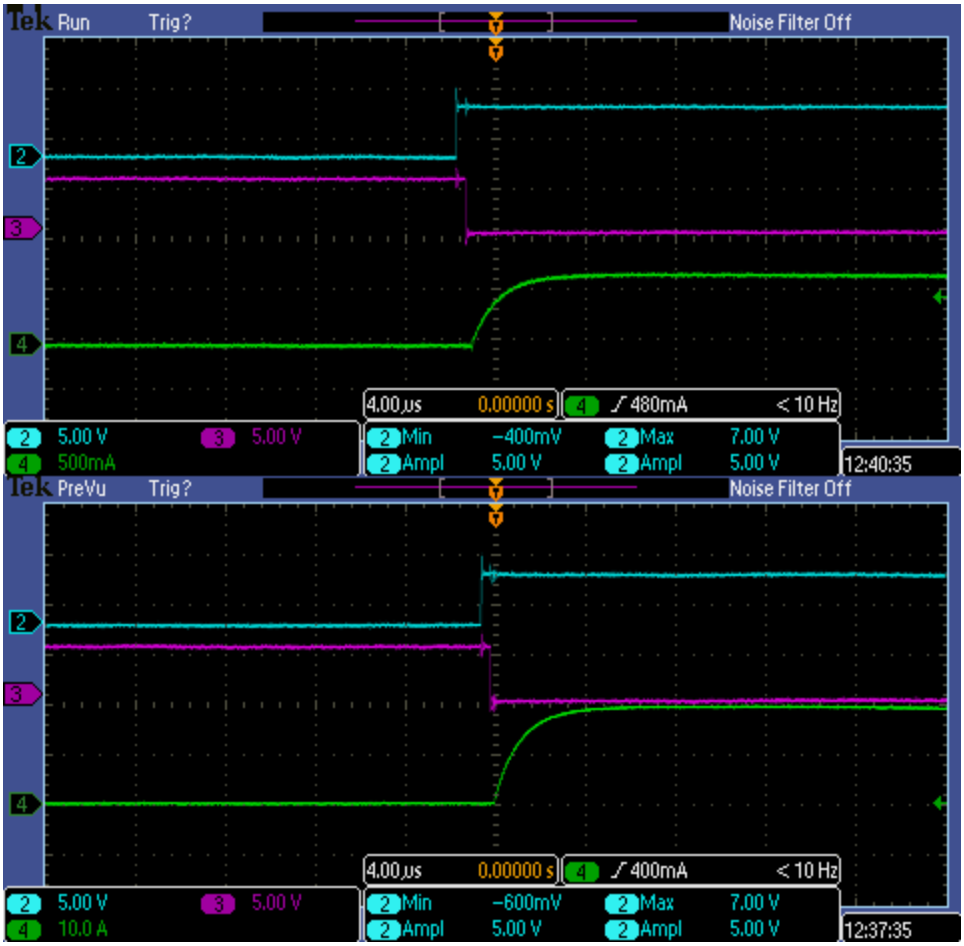

Abbildung 4.8: Messung von Strom über Real Load (Kanal 4) sowie Spannungsverlauf der Signalleitungen für Artificial Load (Kanal 3) und Real Load (Kanal 2) bei Leitung dieser Signale über ein längeres Kabel. Eingestellter Strom oben: 600 mA, unten: 20 A.

Wechsel zwischen Artificial Load und Real Load steuern, untersucht. Diese Messungen sind in Abb. 4.8 zu sehen. Dabei können kleinere Oszillationen zum Zeitpunkt des Schaltvorgangs beobachtet werden, sowie ein sogenannter 'Crosstalk', bei dem eine Änderung des Signals zu einer Spannungsänderung einer benachbarten Datenlinie führt. Dies geschieht jedoch in einem zu geringen Ausmaß, als dass es von einem Bauteil als unerwünschte Änderung des Zustands der jeweiligen Datenlinie wahrgenommen werden könnte. Folglich erfüllt diese Art der Stabilisierung ihren Zweck.

Kapitel 5

Fazit und Ausblick

Die in dieser Arbeit beschriebenen Tests zeigen, dass das hier präsentierte Konzept zur Steuerung der Stromversorgungen mittels als Shiftregister programmierter CPLDs in der Lage ist, die Stromversorgungen wie gewünscht zu schalten, um den Resonator sowohl im statischen, als auch im Wanderwellenmodus zu betreiben.

Auch wenn für die Tests an der Stromversorgung, welche im Rahmen dieser Arbeit durchgeführt wurden, nur zwei Stromversorgungen zur Verfügung standen, konnten über die Tests an den Ausgängen der Shiftregister mit dem Logik Analysator (siehe Unterabschnitt 4.1) gezeigt werden, dass die digitalen Ausgänge der Steuerplatine, welche die unterschiedlichen Stromversorgungen ansteuern, sich wie erwartet verhalten. Dadurch, dass die 48 Stromversorgungen baugleich zueinander sind, darf davon ausgegangen werden, dass diese auch gleich auf die ihren Ablauf steuernden digitalen Signale reagieren.

Auch der Schaltvorgang zwischen Artificial Load und Real Load kann mit diesem Konzept der Steuerung realisiert werden. Wie in Unterabschnitt 4.2 zu sehen ist, können Stabilsationszeiten des Stroms von wenigen Mikrosekunden realisiert werden. Diese kurzen Zeiten sind unbedingt nötig, um die Wellenlängenselektion inklusive Choppen des Strahls bei thermischen Neutronen, wie sie im TRIGA Reaktor des Atominstituts in Wien zur Verfügung stehen, erfolgreich realisieren zu können. Dabei ist zu beachten, dass die Schaltzeit stark von der Induktivität des Widerstands abhängt, über den der Strom fließt. Wie schon in [3] festgestellt, kann alleine schon durch die Länge der Kabel, welche zu den Spulen führen, die Stabilisationszeit stark verändert werden. Bei der Konstruktion der Aufhängung für die Stromversorgungen sollte darauf geachtet werden, dass die Kabel zwischen Stromversorgung und Resonatorspule so kurz wie möglich sind, um die Induktivität so klein wie möglich zu halten und dementsprechend kurze Stabilisationszeiten zu erzielen.

Das kurzzeitige Leiten des Stroms über beide Loads, das ein unerwünschtes Übersteuern des Stroms beim Schaltvorgang vermeidet, kann mithilfe der in der Steuerung implementierten Glitch-Register vorgenommen werden. Liegt ein Ausgang dieser Register auf HIGH, so wird unabhängig vom Status der Ausgänge der anderen Shiftregister Strom über beide Loads geleitet. Durch korrektes Schalten der unterschiedlichen Eingänge dieses Shiftregisters, siehe Abschnitt 3.3.1, kann ein Ablauf des Schaltvorgangs realisiert werden, bei dem während eines kurzen Zeitintervalls von ca. 600 ns, Strom über beide Loads geleitet wird.

Derzeit wird im Rahmen einer Projektarbeit von Maja Sajatovic ein Graphical User Interface (kurz GUI) entwickelt, mit welchem die für den Resonator notwendigen Parameter eingestellt werden können. Damit soll es für den Benützer intuitiv möglich sein, über das Einstellen der gewünschten Resonanzwellenlänge sowie des gewünschtes Modus des Resonators (statisch, Wanderwelle) die Stromversorgungen zu initialisieren, ohne sich dabei mit dem genauen Ablauf der unterschiedlichen Ausgänge der Steuerung beschäftigen zu müssen.

Die Kommunikation mit dem GUI findet über die serielle Schnittstelle statt. Hierbei werden zunächst vom dem GUI zugrunde liegenden Programm die für den Mikrocontroller relevanten Werte berechnet. So können aus der gewünschten Resonanzwellenlänge die benötigten Werte für das Resonatormagnetfeld berechnet werden, da dieses vom über die Spulen fließenden Strom abhängt, können daraus die Werte abgeleitet werden, die an die DACs der Stromversorgungen geschickt werden müssen. Auch andere Parameter, wie das Muster, welches durch die Shiftregister geschoben werden soll, wird hier ausgewählt, oder auch der Takt, mit der dieses Muster weitergeschoben werden muss, damit die Neutronenpakete im Wanderwellenmodus beim Durchflug durch den Resonator vom transversalen Magnetfeld mitgbegleitet werden.

Dann werden diese Werte über die serielle Schnittstelle an den Mikrocontroller gesendet, dieser nimmt dann zum Beispiel, wie in Abschnitt 3.3.2 beschrieben, die I^2C-Kommunikation mit den Stromversorgungen vor, um die passenden Stromwerte einzustellen. Andere Parameter, wie das Muster oder die Zeiteinheit nach der das Muster um eine Stelle weitergeschoben wird[1], können vom Mikrocontroller direkt in die dazugehörigen Register kopiert werden.

Welcher Betriebsmodus gewünscht ist, wird ebenfalls per serieller Schnittstelle übermittelt damit vom Mikrocontroller entsprechende Schritte eingeleitet werden können. So muss etwa beim statischen Modus nur das gewünschte Muster in das Shiftregister geschoben und dieses dann zum Startzeitpunkt mittels Latch zu den Ausgängen geleitet werden.

[1] Dies entspricht dem Wert, mit dem der Timer-Interrupt ausgelöst wird, siehe Abschnitt 3.3.1.

Beim Wanderwellenmodus, bei dem das Muster in gleichmäßigen Zeitabständen weitergeschoben wird, um ein Mitbegleiten des Neutronenpakets zu ermöglichen, wurden noch zwei Extraparameter hinzugefügt, die in Abb. 5.1 als Zeitintervalle *t1* und *t2* bezeichnet sind. Der bis jetzt erwähnte Wanderwellenmodus geht davon aus, dass ein Muster ständig durch den Resonator geschoben wird, was dem Fall $t1 = t2 = 0$ entspricht. Nun können aber auch andere Modi gewünscht sein. *t1* ist hier das Zeitintervall, während dem der Resonator eingeschalten bleibt und das Muster nicht weitergeschoben wird. Damit können Neutronenpulse generiert werden, die länger sind als L/v_0. (L ist die gesamte Resonatorlänge, v_0 die zur selektierten Wellenlänge invers proportionale Neutronengeschwindigkeit). Es könnte etwa auch gewünscht sein, dass zwischen einzelnen Neutronenpulsen keine Änderung der Neutronenpolarisation durch den Resonator erfolgen soll, dazu kann das Zeitintervall *t2* benützt werden, während dem der Resonator inaktiv ist und auch nichts weitergeschoben wird.

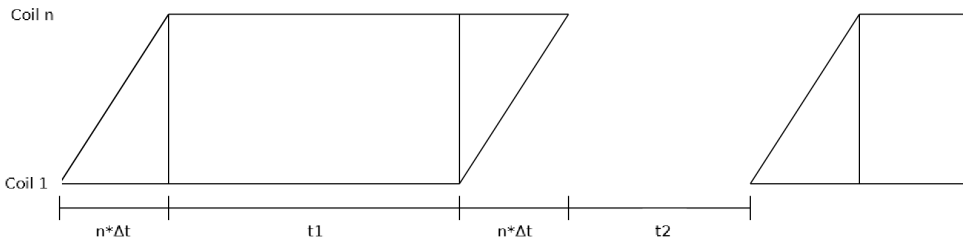

Abbildung 5.1: Der Wanderwellenmodus des Resonators mit den Extraparametern *t1* und *t2*. Auf der x-Achse ist die Zeit, auf der y-Achse die Spulen in der Reihenfolge des Einschaltens aufgetragen, befindet sich ein Punkt im Diagramm innerhalb des Parallelogramms, so ist die Spule in diesem Moment und damit das Resonatormagnetfeld eingeschaltet. Als Beispiel wird hier gezeigt, wie ein Muster aus n (die Anzahl der Spulen) Einsern in den Resonator geschoben wird, nach einer Zeit $\Delta t^* n$ ist der gesamte Resonator eingeschaltet und das Durchschieben wird für eine Zeitdauer *t1* unterbrochen, dann wird für eine Zeit $\Delta t^* n$ das Muster durch Nullen ersetzt und das Resonatormagnetfeld so schrittweise abgeschaltet. Dann wird für eine Zeitdauer *t2* pausiert, während der die Neutronen im Resonator keinen Spinflip durchmachen und demzufolge auch nicht vom Analysator durchgelassen werden.

Das bereits in der Einleitung erwähnte PERC-Experiment benötigt beispielsweise Pulse unterschiedlicher Länge, dies kann durch das Einstellen des Musters, bei Pulslängen von a/v_0 bis L/v_0 bzw bei noch längeren Pulszeiten über das Einstellen von *t1* geschehen. Zwischen diesen Pulsen soll es auch eine gewisse Zeitspanne geben, bei der gar keine Neutronen zum Experiment gelangen, dies geschieht über das Einstellen von *t2*.

Zusätzlich zum GUI wird derzeit im Rahmen einer weiteren Projektarbeit von Martin Wess eine Oberfläche zur Steuerung des gesamten Experiments entwickelt, dabei geht es um das korrekte Timing zueinander von Detektor und den Resonatorspulen sowie um das Einstellen des Selektormagnetfeldes B_0 und die Steuerung der Wasserkühlung. Das GUI hingegen wird ausschließlich dazu verwendet, das alternierende Resonatormagnetfeld B_1 mit den gewünschten Parametern einzustellen.

Der Informationsaustausch mit diesem Steuerungsprogramm, mit dem das Timing der unterschiedlichen Komponenten des Resonators zueinander abgestimmt werden kann, findet über zwei I/O-Pins des auf der Steuerplatine verbauten Mikrocontrollers statt. So gibt es einen Eingang des Mikrocontrollers, welcher als Trigger verwendet werden kann, um ab diesem Zeitpunkt Strom über die Resonatorspulen zu leiten, sowie einen Ausgang, der auf HIGH gesetzt wird, sobald die Resonatorspulen aktiv sind, und so dem Steuerungsprogramm einen Startzeitpunkt liefern kann. Welcher dieser Modi benützt werden soll, ist vom Benutzer zu definieren und wird über das GUI dem Mikrocontroller über die serielle Schnittstelle mitgeteilt.

Zusätzlich sind derzeit für den finalen Betrieb des Resonators auch noch einige Hardwarekomponenten in Arbeit. So sind zwar mittlerweile die Platinen aller 48 Stromversorgungen bestückt, müssen aber noch mit den Kühlkörpern verbunden werden. In Arbeit sind außerdem noch die Verstärkerplatine, sowie die beiden Backplanes (siehe Abb. 2.1). Ein Konzept für die Aufhängung der Stromversorgungen und Backplanes wird derzeit ebenfalls entwickelt, auch das Kühlsystem ist noch in Arbeit, erste Tests mit niedrigen Strömen sollten mit dem finalen Resonator aber auch ohne dieses möglich sein.

Um das gesamte System mit Strom zu versorgen, wurden außerdem industrielle Netzteile erworben, die in der Lage sind, bei einer Spannung von 12 V bis zu 1500 A auszugeben, jede der einzelnen Stromversorgungen kann dabei bei 12 V bis zu 25 A liefern. Wenn alle 48 Stromversorgungen auf maximaler Strombelastung laufen, entspricht dies einem Gesamtstrom 1200 A und einer Gesamtleistung von 14.4 kW. Um eine dermaßen hohe Belastung zu vermeiden, werden zwei solcher Hochleistungenetzgeräte Netzgeräte verwendet, wovon jedes 24 Stromversorgungen mit Leistung versorgt und daher nur maximal 600 A bei 12 V, also nur mehr 7.2 kW liefern muss.

Da ein derart großer Strom selbst wieder ein Magnetfeld erzeugt, welches für Störungen im Resonatorbetrieb sorgen könnte, wurde bei der Zuleitung zwischen Netzteilen und Stromversorgung auf eine Art überdimensionales Koaxialkabel gesetzt, welches dieses Magnetfeld abschirmen soll. Dabei handelt es sich um eine Cu-Stange mit 30 mm Durchmesser, welche sich inmitten eines Cu-Rohrs mit 50 mm Außendurchmesser und 6 mm Wanddicke befindet. Diese Stange wird dabei auf 12 V gesetzt, das Rohr selbst wird geerdet. Mittels unterschiedlich langer Schrauben, die in das Rohr und - mit einer

Isolierung versehen - in die Kupferstange eingeschraubt werden, können die Stromversorgungen mit den beiden Netzteilen leitend verbunden werden. Die Dimensionen von Kupferstange und Rohr wurden dabei so gewählt, dass der Querschnitt so groß ist, dass keine nennenswerte Erwärmung dieser Rohre durch den darüber geleiteten Strom zu befürchten ist[2].

Im Rahmen einer weiteren Diplomarbeit von Christopher Burger-Scheidlin werden die Tests des MONOPOL-Wanderwellenresonators fortgeführt werden, dabei sollen nach Fertigstellung der restlichen Hard- und Software auch erste Experimente am seit kurzem verfügbaren *Weißen Thermischen Neutronenstrahl* des TRIGA-Reaktors duchgeführt werden. Dabei müssen neben dem Test der Resonatorspulen auch noch die anderen Komponenten des Resonators getestet werden, wie etwa der Polarisationsgrad der beiden Polarisatoren oder die Effizienz des zusätzlich im Strahlengang erforderlichen Breitband-Spinflippers, welche für das erfolgreiche Durchführen einer Wellenlängenselektion ebenso unverzichtbar sind.

[2] Querschnitt Cu-Stange 706 mm^2, Querschnitt Cu-Rohr 828 mm^2.

Literaturverzeichnis

[1] G. Ehlers, A. Podlesnyak, J. L. Niedziela, *The new cold neutron chopper spectrometer at the Spallation Neutron Source: Design and performance*, Review of Scientific Instruments 82.8 (2011) 085108.

[2] A. Frank, *Software-Entwicklung und Test der Steuerungselektronik für den Neutronenresonator* MONOPOL, Bachelorarbeit, Technische Universität Wien, 2015.

[3] A.Frank, *Test der Steuerungselektronik für den Neutronenresonator* MONOPOL, Projektarbeit, Technische Universität Wien, 2017.

[4] D. Dubbers et al. , *A clean, bright, and versatile source of neutron decay products*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 596 (2008), 238-247.

[5] G. Badurek, E. Jericha, *Upon the versatility of spatial neutron magnetic spin resonance*, Physica B 335 (2003) 215-218.

[6] G. M. Drabkin, V.A. Trunov, V.B. Runov, *Static magnetic field analysis of a polarized neutron spectrum*, JETP 27 (1963) 194.

[7] C. Gösselsberger, *Entwicklung eines Wanderwellen-Neutronen-Spinresonators*, Dissertation, Technische Universität Wien, 2012.

[8] M.M. Agamalyan, G.M. Drabkin, V.I. Sbitnev, *Spatial spin resonance of polarized neutrons. A tunable slow neutron filter*, Physics Reports 168 (1988) 265-303.

[9] S. Ecker, *Development of a cooling concept for the operation of a spatial magnetic neutron spin resonator with thermal neutrons*, Projektarbeit, Technische Universität Wien, 2016.

[10] F. Linhardt, *Neuentwurf der Steuerung des* MONOPOL*-Experiments*, Projektarbeit, Technische Universität Wien, 2016.

[11] *LSI 2064/A In-System Programmable High Density PLD*, URL=`www.latticesemi.com`.

[12] *Programming Cable User's Guide*, url=www.latticesemi.com/en/Products/DevelopmentBoardsAndKits/ProgrammingCablesforPCs.

[13] *ispLSI Macro Library Reference Manual*, URL=www.latticesemi.com.

[14] *Generic Macro Library Reference Manual*, URL=www.latticesemi.com/view_document?document_id=36516.

[15] *PIC24FV32KA304 FAMILY Manual*, URL=http://ww1.microchip.com/downloads/en/DeviceDoc/30009995e.pdf.

[16] *CTS Model CB3 & CB3LV HCMOS/TTL Clock Oscillator Datasheet*, URL=https://www.mouser.com/ds/2/96/008-0256-0-786323.pdf.

[17] *MPLAB® ICD 3 In-Circuit Debugger User's Guide*, URL=ww1.microchip.com/downloads/en/DeviceDoc/50002081B.pdf.

[18] R. Raab, *Weiterentwicklung eines Wanderwellen-Neutronenspin-resonators für sehr kalte Neutronen*, Diplomarbeit, Technische Universität Wien, 2013.

[19] *LTC2631 Single 12-/10-/8-Bit I2C VOUT DACs with 10ppm/C Reference*, URL=http://cds.linear.com/docs/en/datasheet/2631fd.pdf.

[20] *Saleae Logic Pro 16 USB Logic Analyzer*, URL=downloads.saleae.com/specs/Logic+Pro+16+Data+Sheet.pdf.

[21] *Tektronix 30 A AC/DC Current Probe TCP0030A Datasheet*, URL=https://www.tek.com/datasheet/30-ac-dc-current-probe.

[22] *Tektronix A6302 & A6302XL 20 Ampere AC/DC Current Probes*, URL=https://www.tek.com/a6302-manual/a6302-a6302xl-instructions-0.

[23] *Mixed Signal Oscilloscopes MSO2000B Series, DPO2000B Series Datasheet*, URL=https://www.tek.com/sites/default/files/media/media/resources/MSO2000B-DPO2000B-Mixed-Signal-Oscilloscope-Datasheet_3GW-28413-0.pdf.