Mikrocontrollerbasierte Regelung der Hochfrequenzamplitude des ELSA Booster-Synchrotrons

Jens-Peter Thiry

Diplomarbeit in Physik
angefertigt im Physikalischen Institut

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
Bonn

Juni 2011

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Referent: PD Dr. Wolfgang Hillert
Koreferent: Prof. Dr. K. Brinkmann

Inhaltsverzeichnis

Einleitung

Die Elektronen-Stretcher-Anlage (ELSA) des Physikalischen Instituts der Universität Bonn ist seit 2004 einer der Mittelpunkte des DFG-Sonderforschungsbereiches Transregio 16 „Subnuclear Structure of Matter – Elektromagnetische Anregung subnuklearer Systeme“, der sich mit der Wechselwirkung und inneren Struktur von Hadronen beschäftigt. Dazu werden Experimente mit Photonen durchgeführt, die über kohärente Bremsstrahlung wahlweise aus unpolarisierten oder polarisierten Elektronen erzeugt werden. Zusammen mit einem polarisierten Target können Doppelpolarisationsexperimente durchgeführt werden.

Abbildung 0.1 zeigt einen Lageplan der gesamten Beschleunigeranlage. Diese kann in drei Beschleunigungsstufen eingeteilt werden: Im Bereich des Injektors befinden sich zwei Elektronenquellen, eine Quelle für polarisierte und eine für thermische Elektronen. Beide liefern Elektronen mit einer Energie von $50\,\mathrm{keV}$. Daran schließt sich der Linearbeschleuniger LINAC 2 an, der die Elektronen auf eine Energie von $26\,\mathrm{MeV}$ beschleunigt. In der zweiten Stufe, dem Booster-Synchrotron, findet die Vorbeschleunigung auf typischerweise $1{,}2\,\mathrm{GeV}$ statt. Die sich daran anschließende dritte Stufe, der sogenannte ELSA-Stretcherring, beschleunigt schließlich auf die Endenergie von maximal $3{,}5\,\mathrm{GeV}$.

Das Booster-Synchrotron ist bereits seit 1967 in Betrieb und hat zunächst viele Jahre eine Vielzahl von Experimenten mit beschleunigten Elektronen versorgt. Da die Ablenkmagnete synchron mit der Frequenz des Netzstroms erregt werden, findet die Beschleunigung im Synchrotron mit $20\,\mathrm{ms}$-Zyklen statt. Die Extraktionszeit beträgt dabei weniger als $1\,\mathrm{ms}$. Das Verhältnis der Zeit, in der dem Experiment Elektronen zur Verfügung stehen, zur Gesamtzyklusdauer ist demnach kleiner als $5\,\%$. Diese Situation hat sich mit dem Aufbau des ELSA-Stretcherrings erheblich verbessert. Dieser akkumuliert typischerweise bis zu 21 Injektionen aus dem Synchrotron. Der umlaufende Strahlstrom wird dann anschließend im Subsekundenbereich beschleunigt. Danach wird mittels Resonanzextraktion langsam extrahiert. So lassen sich Tastverhältnisse von über $75\,\%$ erreichen.

Als schneller Injektor für den ELSA-Stretcherring wird das Synchrotron nach wie vor als unverzichtbarer Bestandteil der gesamten Anlage benötigt. Insbesondere sollen in naher Zukunft in ELSA höhere umlaufende Elektronenströme beschleunigt werden, um für die Experimente höhere Ereignisraten zu erzeugen. Dazu muss vor allem das Hochfrequenzsystem des Synchrotrons in der Lage sein, die für höhere Ströme benötigte Leistung optimal zur Verfügung zu stellen, ohne dass dabei nicht tolerierbare Stahlverluste auftreten.

Der eigentliche Beschleunigungsvorgang im Synchrotron findet mittels eines dreizelligen Hohlraumresonators statt, in den dazu Hochfrequenz-Leistung mit der Eigenfrequenz des Resonators eingekoppelt wird. Dadurch bilden sich longitudinale elektrische Felder aus, die beschleunigend auf die den Resonator durchfliegenden Elektronen wirken. Da die Eigenfrequenz des Resonators stark von seiner geometrischen Form abhängt, kann die Eigenfrequenz durch in den Resonator einfahrbare Kupferstempel abgestimmt werden. Insbesondere Temperaturänderungen verschieben die Eigenfrequenz des Resonators, was zu unerwünschten Reflexionen von Hochfrequenzleistung führt. Daher soll eine automatisierte Stempelregelung in Betrieb genommen werden, die diese Reflexionen minimiert.

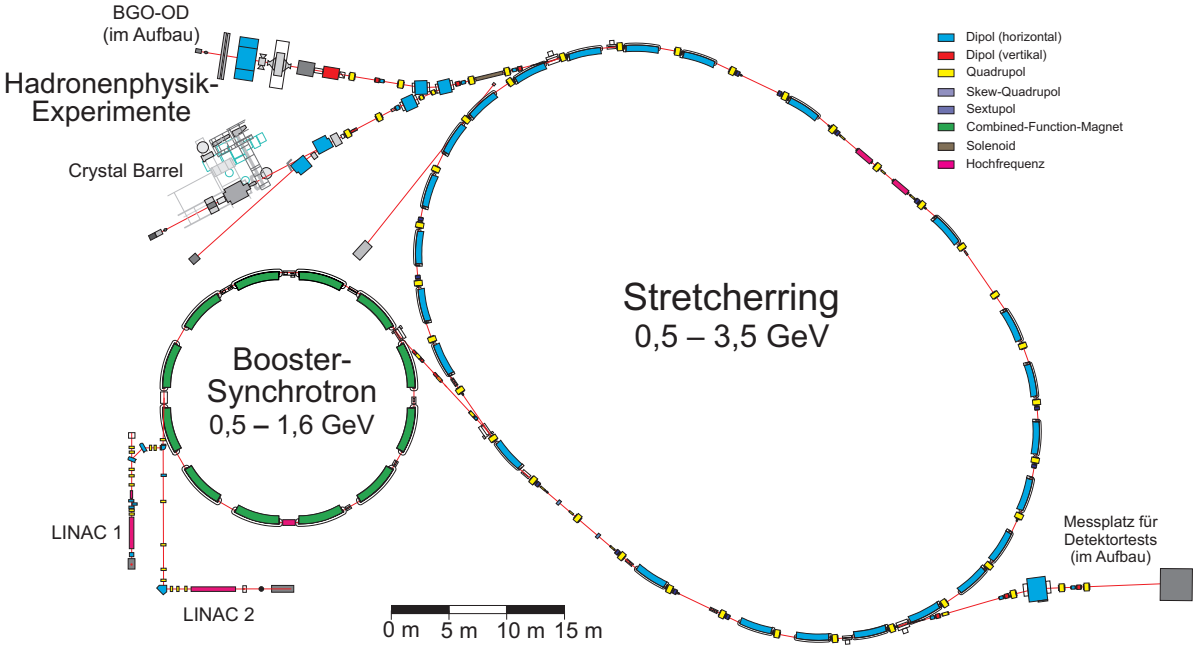


Abbildung 0.1.: Lageplan der ELSA Beschleunigeranlage.

Bei einem Synchrotron müssen die zu beschleunigenden Teilchen stets auf einer fest vorgegebenen Bahn umlaufen. Dazu ist es erforderlich, die Höhe der beschleunigenden elektrischen Feldstärke mit den ablenkenden magnetischen Feldern zu synchronisieren. Diese Synchronisation wird am Booster-Synchrotron mit Hilfe eines sogenannten Programmgenerators hergestellt, indem dieser die Feldstärke im Hohlraumresonator an die zeitliche Änderung der Magnetfeldstärke anpasst. Der Programmgenerator soll modernisiert werden, so dass er digital durch das Kontrollsystem der Beschleunigeranlage gesteuert werden kann. Zusätzlich wird auch hier der Einsatz eines Reglers zur Stabilisierung der Hochfrequenzamplitude gefordert.

Es lassen sich somit zwei Ziele formulieren, die in dieser Arbeit realisiert werden sollen:

1. Eine Regelung der Eigenfrequenz des Hohlraumresonators soll in Betrieb genommen werden.

2. Ein neuer Programmgenerator für die Hochfrequenzamplitude soll aufgebaut und mit einem Regler versehen werden.

Kapitel 1.

Beschleunigung von Elektronen in einem Synchrotron

Zur Beschleunigung von Elektronen in Kreisbeschleunigern werden elektromagnetische Wellen in Hohlraumresonatoren eingekoppelt, die dort longitudinal[1] schwingende elektrische Wechselfelder bilden. In diesem Kapitel werden zunächst die im Hinblick auf die Ansteuerung des gesamten Hochfrequenzsystems relevanten Grundlagen der Beschleunigung mit Hohlraumresonatoren zusammengefasst. Dabei wird sich auf ultra-relativistische Elektronen beschränkt.

1.1. Hohlraumresonatoren

Ein Hohlraumresonator kann allgemein als eine von leitenden Wänden umschlossene Struktur definiert werden, in der sich stehende elektromagnetische Wellen ausbilden können. Für solche Strukturen existieren spezielle Konfigurationen des elektromagnetischen Feldes, die bestimmte Orts- und Frequenzbedingungen erfüllen und dabei gleichzeitig elektromagnetische Energie in der Struktur speichern. Ein Hohlraumresonator ermöglicht daher die Ausbildung von stehenden Wellen mit bestimmten festen Frequenzen. Regt man einen Hohlraumresonator mit seiner Resonanzfrequenz an, so wird sich ein elektromagnetisches Feld aufbauen und so Energie gespeichert. Man nennt diesen Vorgang Resonanz, dieser ist dabei durch die auftretenden Verluste in den Resonator-Wänden beschränkt. Im Folgenden werden einige wichtige Zusammenhänge im Hinblick auf die Beschleunigung mit Hohlraumresonatoren eingeführt. (vergleiche. [Jac98, Kap. 8.2-8.8]).

1.1.1. Moden

Das elektromagnetische Feld in einem Hohlraumresonator wird durch die Lösungen der Maxwellgleichungen für das elektrische Feld $\vec{E}(\vec{r},t)$ und das magnetische Feld $\vec{H}(\vec{r},t)$ innerhalb des Hohlraums angegeben. Auf den leitenden Oberflächen können keine dazu parallel ausgerichteten elektrischen Felder existieren, dasselbe gilt für senkrecht zu den Oberflächen stehende magnetischen Felder. Diese Randbedingungen führen zusammen mit der Ausbildung von stehenden Wellen zu zwei Klassen von erlaubten Feldkonfiguration, welche als Moden[2] bezeichnet werden:

[1] Als longitudinale Richtung wird die Strahlrichtung bezeichnet. Bei Verwendung von Zylinderkoordinaten (ρ,φ,z) wird als die longitudinale Richtung die z-Koordinate verwendet.

[2] Als Moden (engl. eigenmode) bezeichnet man die stationären Eigenschaften von Wellen hinsichtlich ihrer räumlichen Energieverteilung.

TM-Mode: Transversal-magnetische Mode keine longitudinalen magnetischen Felder

TE-Mode: Transversal-elektrische Mode keine longitudinalen elektrischen Felder

Zur Beschleunigung werden elektrische Felder in longitudinaler Richtung benötigt, daher scheiden TE-Moden aus. Für einen zylinderförmigen Resonator mit Länge L und Radius R existieren abzählbar viele Frequenzen von TM-Moden:

$$w_{mnp} = c \cdot \sqrt{\left(\frac{j_{mn}}{R}\right)^2 + \left(\frac{\pi p}{L}\right)^2}. \tag{1.1}$$

Hierbei ist j_{mn} die n-te Nullstelle der m-ten Besselfunktion und mit c ist die Lichtgeschwindigkeit bezeichnet. Die dabei auftauchenden ganzzahligen Indizes m, n und p bezeichnen die einzelnen Moden. Anschaulich beschreiben diese Indizes die Anzahl der Knotenpunkte der elektromagnetischen Felder in einer bestimmten Koordinatenrichtung. Der Index n beschreibt die Anzahl der Knoten in radialer Richtung, m beschreibt die Anzahl der Perioden in φ-Richtung und p die Anzahl der longitudinalen Nulldurchgänge.

Für die Beschleunigung wird nur die niedrigste Frequenz, die Grundmode (TM_{010}-Mode) benutzt [Wil96, S. 181]. Abbildung 1.1 zeigt die zugehörige Feldverteilung für eine TM_{010}-Mode in longitudinaler und radialer Richtung. Die Feldverteilung ist radialsymmetrisch ($m = 0$), in longitudinaler Richtung homogen ($p = 0$) und weist in radialer Richtung nur einen Knoten auf ($n = 1$). Die Frequenz ist nach Gleichung (1.1) nur vom Radius abhängig. Zusammen mit der ersten Nullstelle der nullten Besselfunktion $j_{01} \approx 2{,}405$ ergibt sich für die Resonanzfrequenz der Mode:

$$\omega_{010} = \omega_0 = c \cdot \frac{2{,}405}{R}. \tag{1.2}$$

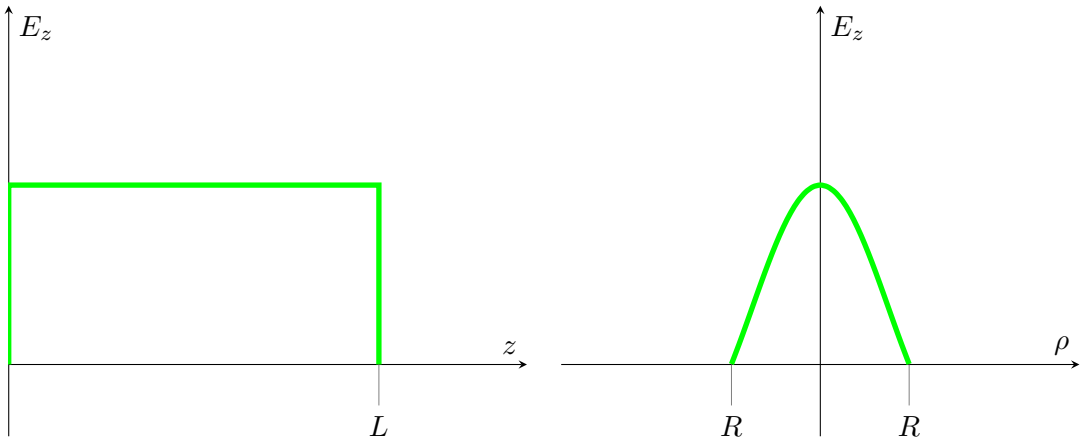


Abbildung 1.1.: Schematischer Verlauf der elektrischen Feldstärke einer TM_{010}-Mode in einem zylinderförmigen Hohlraumresonator mit Länge L und Radius R . Links ist der Verlauf in longitudinaler Richtung, rechts der Verlauf in radialer Richtung gezeigt.

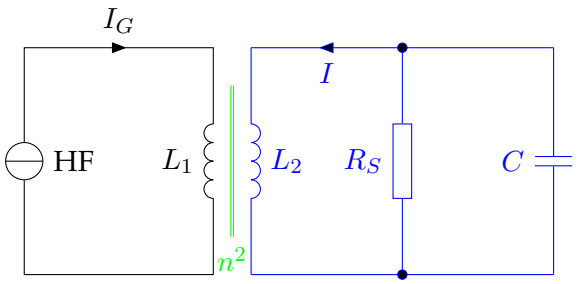


Abbildung 1.2.: Ersatzschaltbild eines Hohlraumresonators (Parallelschwingkreis aus L_2,R_S und C) zusammen mit der Einkopplung externer Hochfrequenzleistung (linker Schaltkreis). Die Erzeugung der Hochfrequenzleistung wird symbolhaft durch eine Stromquelle repräsentiert. Die Einkopplung in den Hohlraumresonator geschieht induktiv (L_1 und L_2, Übersetzungsverhältnis $1 : n^2$). (zur Herleitung vgl. [Pus09, S. 20])

1.1.2. Resonatoreigenschaften: Shuntimpedanz und Güte

Das Integral über die elektrischen Feldstärke längs der Resonatorachse ergibt die Potenzialdifferenz:

$$U_{\text{Res}} = \int_0^L E_0(z)\mathrm{d}z. \tag{1.3}$$

Wird die Leistung P_{HF} in einen Resonator eingespeist, so kann die entstehende Spannung U_{Res} mit der sogenannten Shuntimpedanz R_{S} verknüpft werden [Wil96, S. 184]:

$$U_{\text{Res}} = \sqrt{2P_{\text{HF}}R_{\text{S}}}. \tag{1.4}$$

Bei resonanter Anregung beschreibt die Shuntimpedanz anschaulich die ohmschen Verluste des Resonators. Des Weiteren führt man die Güte Q als Verhältnis der im zeitlichen Mittel gespeicherten Energie zu den Energieverlusten während einer Anregungsperiode ein [Jac98, S. 371]:

$$Q = \omega_0 \frac{\text{gespeicherte Energie}}{\text{Energieverluste}}. \tag{1.5}$$

Bei der Einkopplung von Hochfrequenz in einen Resonator zeigt sich ein frequenzabhängiges Verhalten. Als ein gutes Modell für einen Hohlraumresonator kann ein elektrischer Parallelschwingkreis, der aus einer Kapazität, einer Induktivität und einem ohmschen Widerstand aufgebaut ist, verwendet werden. Eine Übersicht dazu und die Herleitung der hier wiedergebenden Zusammenhänge wird in [Pus09, Kap. 3.4] gegeben.

1.1.3. Ersatzschaltbild eines Hohlraumresonators

In Abbildung 1.2 ist ein solches Ersatzschaltbild gezeigt; der rechte Teil der Schaltung repräsentiert den Resonator. Die Einkopplung der Hochfrequenzleistung ist in diesem Modell symbolisch durch einen Transformator mit dem Übersetzungsverhältnis $1 : n^2$ dargestellt. Im linken Schaltungsteil ist schließlich die Quelle der Hochfrequenz als Stromquelle symbolisiert.

Die frequenzabhängige Impedanz[3] des Parallelschwingkreises ist (vgl. [Lee05, S. 349]):

$$\mathbf{Z}(\omega) = \frac{1}{\frac{1}{R_\mathrm{S}} + \mathbf{i}\omega C + \frac{1}{\mathbf{i}\omega L}} \tag{1.6}$$

und kann auch durch die Güte Q ausgedrückt werden:

$$\mathbf{Z}(\omega) = \frac{R_\mathrm{s}}{1 + \mathbf{i}2Q\frac{(\omega_0 - \omega)}{\omega_0}} \tag{1.7}$$

Reflexionsfaktor

Bei der Einkopplung von Hochfrequenzwellen in einen Resonator kann ein Teil der Welle reflektiert werden. Dies lässt sich mit Hilfe der Leitungstheorie beschreiben. Reflexion tritt dabei immer dann auf, wenn die Impedanzen der Leitung und die der Ein- oder Ausgänge nicht übereinstimmen. Der Reflexionsfaktor wird als das Verhältnis der Amplituden von hin- und zurücklaufender Welle definiert: (vgl. [Hof97, S. 46]):

$$\rho = \frac{\hat{U}_-}{\hat{U}_+}. \tag{1.8}$$

Dabei ist $\hat{U}_-$ die Amplitude der zurücklaufenden Welle, $\hat{U}_+$ dementsprechend die der hinlaufenden. Beachtet man, dass sich die Impedanz des Hohlraumresonators mit dem Übertragungsverhältnis $1 : n^2$ transformiert, so kann der Reflexionsfaktor an der Einkopplung als das Verhältnis von Generatorimpedanz $\mathbf{Z}_\mathrm{G}$ zur Impedanz des Resonators $\acute{\mathbf{Z}}_\mathrm{C} = \frac{\mathbf{Z}_\mathrm{C}}{n^2}$ ausgedrückt werden (vgl. [Hof97, S. 46]):

$$\rho = \frac{\acute{\mathbf{Z}}_\mathrm{C} - \mathbf{Z}_\mathrm{G}}{\acute{\mathbf{Z}}_\mathrm{C} + \mathbf{Z}_\mathrm{G}}. \tag{1.9}$$

Üblicherweise beträgt die Impedanz des Generatorzweiges $\mathbf{Z}_\mathrm{G} = 50\,\Omega$. Die Einkopplung wird so konstruiert, dass bei Anregung mit der Resonanzfrequenz ω_0 keine Reflexion auftritt ($\mathbf{Z}_\mathrm{G} = \acute{\mathbf{Z}}_\mathrm{C}$). Da die Impedanz des Hohlraumresonators frequenzabhängig ist, wird bei nicht resonanter Anregung Reflexion auftreten.

1.1.4. Amplituden und Phasenverlauf bei resonanter Anregung

Im Hinblick auf spätere Kapitel dieser Arbeit ist der Verlauf der Resonanzkurve und insbesondere der Verlauf der Phase zwischen der anregenden Hochfrequenz und des elektrischen Feldes längs der Resonatorachse von Bedeutung.

Sowohl der Betrag der Beschleunigungsspannung in einem Hohlraumresonator als auch die Phase zwischen anregender Hochfrequenz und dem elektrischen Feld sind abhängig von der

[3] Mit Impedanz (lat. impedire „hemmen“, „hindern“) wird der Wechselstromwiderstand bezeichnet, der als das Verhältnis von elektrischer Spannung an einem elektrischen Bauelement zum aufgenommenem Strom definiert ist.

anregenden Frequenz (vgl. [Sch06]):

$$|U(\omega)| = \frac{U_0}{\sqrt{1 + Q^2 \left(\frac{\omega}{\omega_0} - \frac{\omega_0}{\omega}\right)^2}}, \tag{1.10}$$

$$\phi(\omega) = \arctan\left(Q\left(\frac{\omega_0}{\omega} - \frac{\omega}{\omega_0}\right)\right). \tag{1.11}$$

In Abbildung 1.3 sind beide Zusammenhänge für verschiedene Güten gezeigt. Resonatoren, die für die Teilchenbeschleunigung benutzt werden, haben sehr hohe Güten im Bereich von 1×10^4, [Wil96, S. 183] weswegen auch insbesondere der Phasenverlauf (siehe rechter Teil von Abbildung 1.3) sehr steil verläuft. Weicht die anregende Frequenz von der Resonanzfrequenz ab, so ergibt sich eine deutlich messbare Änderung der relativen Phase. Dieser Effekt kann für Regelzwecke benutzt werden. Dies wird in Kapitel 3 zur Regelung der Resonanzfrequenz des Hohlraumresonators benutzt.

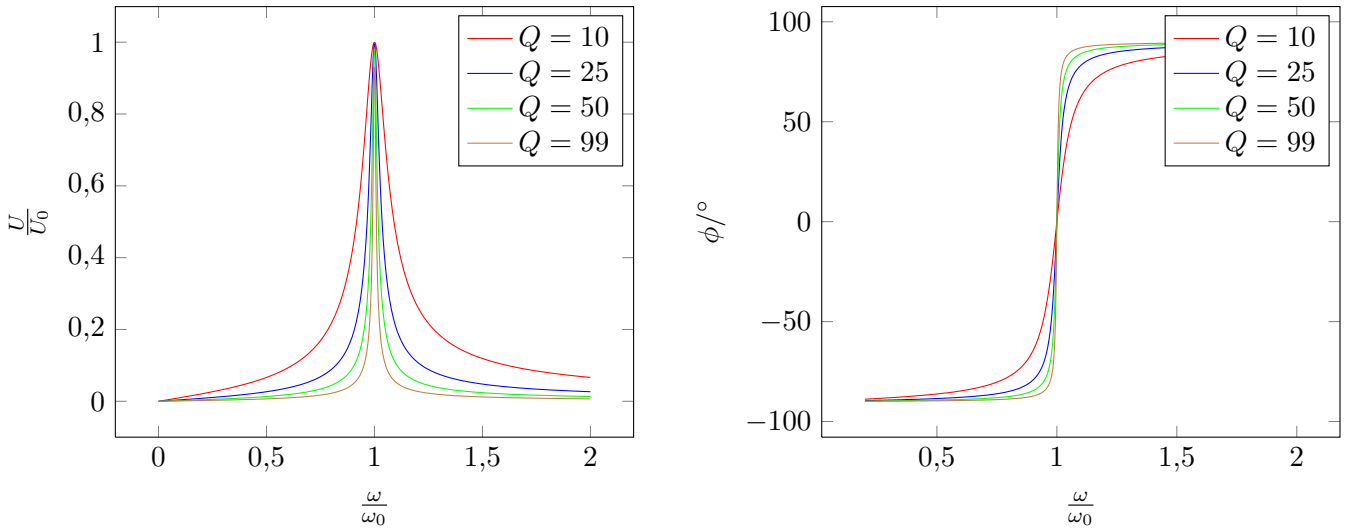


Abbildung 1.3.: Verlauf von Amplitude (links) und relativer Phase (rechts) in Abhängigkeit von der anregenden Frequenz ω. Die Kurven sind für die jeweils angegebenen Güten nach den Gleichungen (1.10) und (1.11) berechnet worden.

1.1.5. Energiegewinn in einem Hohlraumresonator

Der Energiegewinn einer durch die TM_{010}-Mode des Resonators beschleunigte Ladung q lässt sich durch die Integration der elektrischen Feldstärke berechnen (vergleiche auch Gleichung (1.3)):

$$\Delta W = q \int E_z(t)\, \mathrm{d}z \tag{1.12}$$

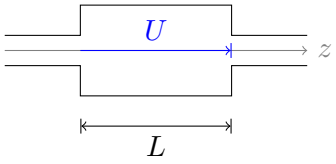


Abbildung 1.4.: Ein zylinderförmiger Hohlraumresonator mit Länge L.

Unter der Annahme, dass die Feldstärke in longitudinaler Richtung homogen ist, lässt sich deren Zeitabhängigkeit explizit mit der anregenden Frequenz ω_{HF} angeben:

$$E(t) = E_z \sin\left(\omega_{\mathrm{HF}}\, t + \varphi\right). \tag{1.13}$$

Mit φ wird die Phase der Elektronen zur Amplitude des beschleunigenden Feldes beschrieben.

Strahlkopplungsfaktor

Hier wird beispielhaft ein zylinderförmiger Hohlraumresonator mit der Länge L betrachtet (vgl. Abbildung 1.4). Die longitudinale Feldstärke im Hohlraumresonator ändert sich zeitlich mit der Frequenz der Anregung ω_{HF}. Diese Änderung führt dazu, dass der tatsächliche Energiegewinn der Elektronen im Vergleich zu dem, der sich aus (1.12) bei zeitlich konstantem E_z ergäbe, geringer ausfällt. Das Verhältnis dieses reduzierten Energiegewinns zum maximal möglichen Energiegewinn wird Strahlkopplungsfaktor genannt [Pir95]. Die Flugzeit wird durch die Geschwindigkeit ausgedrückt, wobei bei ultra-relativistischen Elektronen die Lichtgeschwindigkeit verwendet werden kann:

$$\Delta W = qE_z \int_{-\frac{L}{2}}^{\frac{L}{2}} \sin\left(\omega_{\mathrm{HF}} \frac{z}{c} + \varphi\right) \mathrm{d}z. \tag{1.14}$$

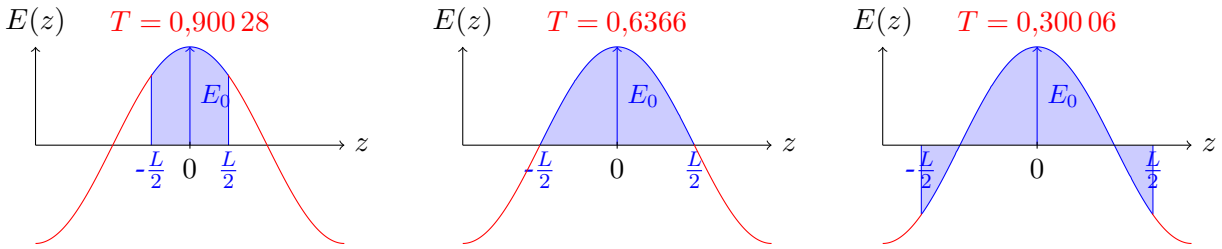
Hierbei wird über die Länge L des Hohlraumresonators integriert. Der maximal mögliche Energiegewinn beträgt $E_0 \cdot q \cdot L \cdot \sin\varphi$. Aus dem Verhältnis zu (1.14) erhält man den Strahlkopplugsfaktor T (vgl. [Pir95]):

$$T = \frac{\sin\left(\frac{\omega_{\mathrm{HF}} \cdot L}{2 \cdot c}\right)}{\frac{\omega_{\mathrm{HF}} \cdot L}{2 \cdot c}}. \tag{1.15}$$

Der Energiegewinn lässt sich unter Berücksichtigung des Strahlkopplungsfaktors nun schreiben als:

$$\Delta W = q \cdot U_0 \cdot T \sin\varphi. \tag{1.16}$$

In Abbildung 1.5 ist für $\varphi = \frac{\pi}{2}$ die Hochfrequenzamplitude für eine Periodenlänge gezeigt. Von links nach rechts sind verschiedene Resonatorlängen eingezeichnet. Der sich daraus ergebene Strahlkopplungsfaktor ist nach Gleichung (1.15) berechnet worden. In Abbildung (1.6) ist der Strahlkopplungsfaktor gegen die Resonatorlänge gezeichnet. Der Strahlkopplungsfaktor ist auf Grund der kürzeren Verweildauer der Elektronen für kleinere Resonatorlängen am größten. Allerdings muss ein Kompromiss bei der Wahl der Länge gefunden werden, da die



(a) $L = \frac{\lambda}{4}$ **(b)** $L = \frac{\lambda}{2}$ **(c)** $L = \frac{3\lambda}{4}$

Abbildung 1.5.: Der Strahlkopplungsfaktor T für drei unterschiedliche Längen des Resonators, jeweils nach Gleichung (1.15) berechnet.

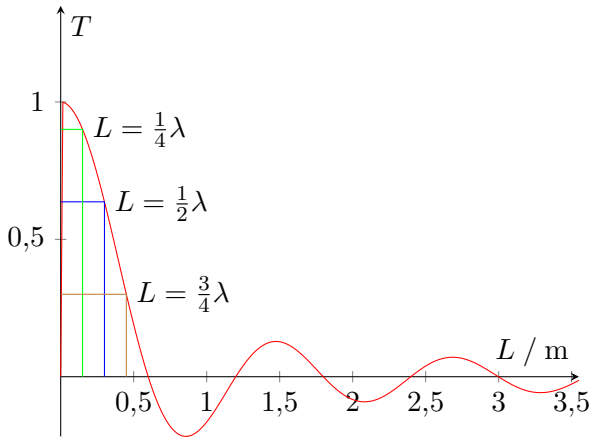


Abbildung 1.6.: Strahlkopplungsfaktor in Abhängigkeit von der Resonatorlänge, berechnet mit w=500 MHz.

Beschleunigungsspannung proportional zur Länge des Resonators ist (vgl. Abschnitt 2.2.2).

Zusammen mit $E = pc$ erhält man aus Gleichung (1.16) den gesuchten Ausdruck für die Impulsänderung pro Umlauf:

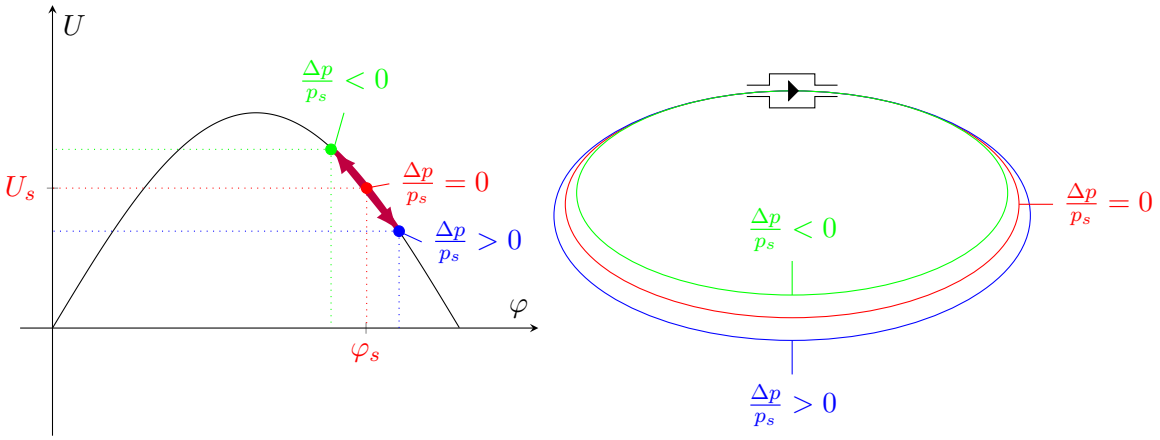
$$\Delta p \cdot c = q \cdot U_0 \cdot T \sin \varphi. \tag{1.17}$$

1.2. Longitudinale Dynamik

Da die elektrische Feldstärke im Hohlraumresonator mit der Hochfrequenz oszilliert, bildet sich diese Zeitstruktur auf die zu beschleunigenden Elektronen ab. Es entstehen Teilchenpakete, die im zeitlichen Abstand der Periodendauer der Hochfrequenzschwingung den Beschleuniger durchlaufen. Die Elektronen innerhalb der Pakete führen zeitliche Schwingungen aus, die im Folgenden betrachtet werden.

1.2.1. Synchrotronschwingung

Ein ultra-relativistisches geladenes Teilchen mit Impuls $\vec{p}$ wird in einem homogenen und zur Ausbreitungsrichtung senkrecht orientierten Magnetfeld $B_\perp$ auf eine Kreisbahn mit dem



(a) Spannung im Resonator in Abhängigkeit von der Phase des Elektronenpaketes zur elektrischen Feldstärke

(b) Bahnlängenänderung aufgrund von relativen Impulsabweichungen.

Abbildung 1.7.: Synchrotronschwingung (vgl. Erläuterungen im Text)

Radius R_s abgelenkt:

$$p(t) = e \cdot R \cdot B_\perp(t). \tag{1.18}$$

Als Sollimpuls p_s wird derjenige Impuls definiert, der nach dieser Gleichung mit der aktuellen Ablenkmagnetfeldstärke den Sollradius R_S ergibt. Ein Elektron mit Sollimpuls wird im ablenkenden Magnetfeld per Definition genau die Sollbahn zurücklegen. Ein Elektron, dessen Impuls kleiner als der Sollimpuls ist, wird bei gleichem Magnetfeld eine kürzere Bahn zurücklegen und daher nach einem Umlauf früher im Resonator ankommen. Gegenteiliges gilt für Elektronen deren Impuls größer als der Sollimpuls ist (vgl. Abbildung 1.7b).

Wird die sogenannte Sollphase φ_s auf der fallenden Flanke der Resonatorspannung gewählt (vgl. Abbildung 1.7a), so stellt sich eine automatische Fokussierung ein. Elektronen mit zu kleinem Impuls ($\frac{\Delta p}{p_\mathrm{s}} < 0$ grüne Kurven) treten früher in den Hohlraumresonator ein, so dass auf diese eine höhere Beschleunigungsspannung wirkt. Dadurch erhalten sie mehr Impulsgewinn als Elektronen mit Sollimpuls; der fehlende Impuls wird also kompensiert. Umgekehrtes gilt für Elektronen mit zu großem Impuls ($\frac{\Delta p}{p_\mathrm{s}} > 0$ blaue Kurven). In der Folge treten um die Sollphase φ_s Schwingungen auf, welche Synchrotronschwingungen genannt werden.

Frequenz der Synchrotronschwingung

Die Schwingung kann für kleine Abweichungen um die Sollphase und unter Vernachlässigung von Dämpfung linearisiert werden:

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\Delta\varphi + \omega_\mathrm{sync}^2 \Delta\varphi = 0, \tag{1.19}$$

dabei wird mit ω_sync die Eigenfrequenz der Schwingung bezeichnet (aus [Hin97, S. 290]):

$$\omega_\mathrm{sync} = \omega_u \sqrt{\frac{h}{2\pi E} e U_0 \cos\varphi_\mathrm{s} \left(\frac{1}{\gamma^2} - \alpha\right)}. \tag{1.20}$$

Die Größen $E, h, \alpha, \gamma, \varphi_\mathrm{s}$ können dabei als nahezu zeitlich konstant angesehen werden. Die Harmonischenzahl

$$h = \frac{\omega_\mathrm{HF}}{\omega_\mathrm{U}} \tag{1.21}$$

ist als das Verhältnis zwischen der Umlauf-Frequenz ω_u zur Resonatorfrequenz ω_HF definiert, und mit α wird der sogenannte Momentum-Compaction-Faktor bezeichnet. Dieser ist als das Verhältnis von relativer Bahnlängenänderung zur relativen Impulsänderung definiert:

$$\frac{\Delta L}{L} = \alpha \frac{\Delta p}{p}. \tag{1.22}$$

Er beschreibt den Effekt, dass aus einer Impulsabweichung aufgrund der Ablenkung in den Dipolmagneten eine Änderung der zurückgelegten Bahnlänge folgt. Das Verhalten der Frequenz der Synchrotronschwingung wird demnach durch den Ausdruck $U_0 \cos \varphi_\mathrm{s}$ bestimmt:

$$\omega_\mathrm{sync} \propto \omega_\mathrm{u} \sqrt{U_0 \cos \varphi_s}. \tag{1.23}$$

1.2.2. Robinson-Dämpfung

Die Umlaufzeit eines Teilchenpaketes ist abhängig von der mittleren Energie der Elektronen innerhalb des Paketes (vgl. Abschnitt 1.2.1). Beim Durchgang durch einen Hohlraumresonator induziert ein mit der Frequenz ω_U umlaufendes Elektronenpaket dort selbst ein elektrisches Feld, welches die zur Verfügung stehende Spannung reduziert[4] (vgl. [Bau05]). Aufgrund der hohen Güte des Resonators wird vor allem die Höhere harmonische $h \cdot \omega_\mathrm{U}$ der Umlauffrequenz in dem Resonator die TM_{010}-Grundmode anregen.

Die Frequenzabhängigkeit der Resonatorimpedanz (vgl. Gleichung (1.7)) kann zur Dämpfung der Synchrotronschwingungen benutzt werden (vgl. [Wie95, S. 200 ff]). Dazu muss der Resonator kapazitiv[5] verstimmt werden. In Abbildung 1.8 ist der Realteil der Resonatorimpedanz gegen die anregende Frequenz als qualitativer Verlauf gezeigt. Die Resonanzfrequenz ω_0 ist wegen der kapazitiven Verstimmung kleiner als die Anregungsfrequenz $\omega_\mathrm{HF} = h \cdot \omega_\mathrm{U}$. Ein Teilchenpaket mit Sollenergie E_{B_0} und Umlauffrequenz ω_U wird beim Durchflug durch den Resonator an der Impedanz $Z(\omega_\mathrm{HF})$ Energie verlieren. Ein Paket, dessen Energie größer ist als die Sollenergie ($E_B > E_{B_0}$), hat eine kleinere Umlauffrequenz als ein Paket mit Sollenergie (vgl. Abschnitt 1.2.1 und Abbildung 1.7 zur Synchrotronschwingung). Ein solches Paket erfährt einen Energieverlust entsprechend der Impedanz $Z(\omega_>)$:

$$Z(\omega_>) > Z(\omega_<)$$

Somit ist der Energieverlust größer als der eines Paketes mit Sollenergie (vgl. Abbildung 1.8). Umgekehrtes gilt für Pakete, deren Energie kleiner als die Sollenergie ist ($E_B < E_{B_0}$). Bei kapazitiver Verstimmung sind die Energieschwingungen demnach gedämpft.

[4] Dieser Effekt wird Beam-Loading genannt. Beam-Loading, engl. etwa Strahl-Belastung. Damit ist der Effekt gemeint, dass der Elektronenstrahl den Resonator belastet, was sich als Veränderung der Resonatorimpedanz und Reduzierung der Resonatorspannung auswirkt.

[5] Bei kapazitiver Verstimmung wird die Impedanz des Resonators zu größeren kapazitiven Anteilen verschoben. Dies ist gleichbedeutend mit einer Verkleinerung der Resonanzfrequenz.

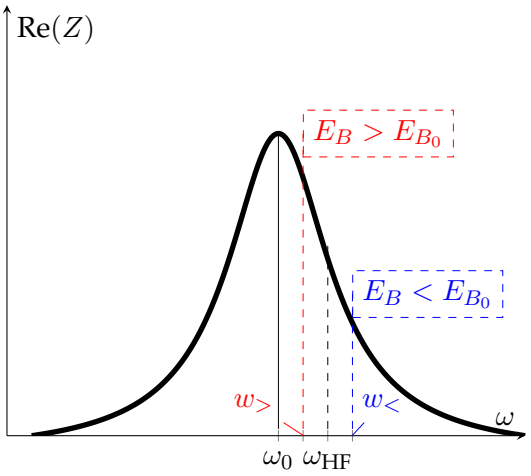


Abbildung 1.8.: Gezeigt ist der qualitative Verlauf des Realteils der Resonator-Impedanz in Abhängigkeit von der Frequenz der Anregung ω. Die Resonanzfrequenz des Resonators ω_0 ist hier kleiner als die anregende Frequenz ω_{HF} gewählt (sogenannte kapazitive Verstimmung). Weiterhin sind die Frequenzen für ein Paket mit Sollenergie und für zwei ausgewählte Pakete mit positiver ($E_B > E_{B_0}$) bzw. negativer ($E_B < E_{B_0}$) Energieabweichung gezeigt.

Diese von der Energie abhängige Dämpfung der Synchrotronschwingung wird Robinson-Dämpfung genannt (nach K. W. Robinson: vgl. [Rob64]). Die hier gegebene anschauliche Darstellung genügt, um den Vorgang prinzipiell zu verstehen und die Notwendigkeit abzuleiten, den Resonator kapazitiv zu verstimmen. Insbesondere muss beachtet werden, dass bei induktiver Verstimmung der gegenteilige, unerwünschte Effekt eintritt: Die Schwingung würde angeregt. Daher wird in Abschnitt 3.2.4 beschrieben wie der Holhraumresonator zur Robinson-Dämpfung kapazitiv verstimmt wird.

1.3. Synchronisation des elektrischen Feldes

Bei einem Synchrotron müssen sich die zu beschleunigenden Teilchen stets auf einer fest vorgegebenen Kreisbahn bewegen. Dazu ist es erforderlich, die elektrische Feldstärke in den Beschleunigungsstrecken und die Änderung der Feldstärke in den Ablenkmagneten ständig aufeinander abzustimmen. Es soll nun der zur Synchronisation nötige Zusammenhang zwischen der Änderung des Impulses der Teilchen und der Änderung der Ablenkmagnetfeldstärke hergeleitet werden.

1.3.1. Notwendige Impulsänderung

Während der Umlaufzeit t_{u} ändert sich die Magnetfeldstärke der Ablenkmagnete um den Anteil $\Delta B = \int_0^{t_{\mathrm{u}}} \dot{B}\,\mathrm{d}t$ und für die Änderung des Sollimpulses gilt:

$$\dot{p}_{\mathrm{s}}(t) = eR\dot{B}(t)\ . \tag{1.24}$$

Um den Energiegewinn pro Umlauf mit der Änderung der Ablenkmagnetfeldstärke zu synchronisieren, muss diese Bedingung zu jedem Zeitpunkt erfüllt werden. Die während eines

Umlaufes auftretenden Energieverluste müssen zusätzlich ausgeglichen werden.

1.3.2. Berechnung der Synchrotronstrahlungsverluste

Der pro Umlauf auftretende Energieverlust durch Synchrotronstrahlung beträgt (vgl. [Wal92, Gleichung (8)]:

$$\Delta E = \frac{e^2}{3\varepsilon_0}\frac{\beta^3\gamma^4}{R}.$$

Ersetzt man den Impuls in Gleichung (1.18) durch den relativistisch korrigierten Ausdruck: $p = \beta\gamma m_0 c$, erhält man den Energieverlust pro Umlauf in Abhängigkeit der Ablenkmagnetfeldstärke :

$$\begin{aligned}\Delta E =& \frac{e^2}{3\varepsilon_0}\left(\frac{p}{m_0 c}\right)^4 \frac{1}{R\beta} \\ =& \frac{e^6 R^3}{3\varepsilon_0 (m_0 c)^4} B^4 \qquad \text{mit } \beta \approx 1\end{aligned} \tag{1.25}$$

1.3.3. Synchronisationsbedingung

Ist die Umlaufzeit t_U deutlich kleiner als die Zeitkonstante, mit der sich das ablenkende Magnetfeld ändert, kann angenommen werden, dass die zeitliche Änderung des Magnetfeldes während eines Umlaufes konstant ist[6]. Die notwendige Änderung des Sollimpulses pro Umlauf kann dann aus Gleichung (1.24) erhalten werden:

$$\Delta p_\mathrm{s} = e \cdot R \cdot t_\mathrm{U} \cdot \dot{B}. \tag{1.26}$$

Der Impulsgewinn in einem Resonator wurde in Gleichung (1.17) in Abhängigkeit von der Resonatorspannung U_0 angeben. Gleichsetzen mit (1.26) und Umformen nach U_0 liefert den gesuchten Zusammenhang zwischen $\dot{B}$ und $U_{\dot{B}}$:

$$U_{\dot{B}} = \underbrace{\frac{c \cdot R \cdot t_\mathrm{U}}{T \sin\varphi}}_{c_{\dot{B}}} \dot{B}. \tag{1.27}$$

Einen Ausdruck für die notwendige Änderung der Beschleunigungsspannung zur Kompensation der Synchrotronstrahlung erhält man durch Gleichsetzen der Gleichung für den Energiegewinn im Resonator (1.16) mit Gleichung (1.25):

$$U_{B^4} = \underbrace{\frac{e^5 R^3}{3\varepsilon_0 (m_0 c)^4 \cdot T \cos\varphi}}_{:=c_{B^4}} B^4. \tag{1.28}$$

[6] Beim Bonner Booster-Synchrotron beträgt die Umlaufzeit $t_\mathrm{U} = 232{,}16\,\mathrm{ns}$. Das Magnetfeld ändert sich mit einer Frequenz von $f = 50\,\mathrm{Hz}$, was einer Zeitkonstante von $t_\mathrm{mag} = 20\,\mathrm{ms}$ entspricht. Es darf also davon ausgegangen werden, dass die zeitliche Änderung der Magnetfeldstärke während eines Umlaufes konstant ist.

Zusammengesetzt ergibt sich folgende Vorschrift:

$$\boxed{U = c_{\dot{B}} \cdot \dot{B} + c_{B^4} \cdot B^4} \tag{1.29}$$

Als zentrales Thema dieser Arbeit (Kapitel 4) wird eine Schaltung aufgebaut, welche die Beschleunigungsspannung im Resonator nach dieser Vorschrift ansteuert. Dazu wird die Änderung der Ablenkmagnetfeldstärke gemessen und daraus die vierte Potenz des aktuellen Magnetfeldes errechnet.

Kapitel 2.

Das Bonner Booster-Synchrotron

Seit 1967 ist am Physikalischen Institut der Universität Bonn ein $2{,}5\,\mathrm{GeV}$-Elektronen-Synchrotron in Betrieb [Alt+68], welches verschiedene Experimente der Teilchenphysik mit Elektronen versorgte. Ab 1982 wurde der ELSA-Stretcherring gebaut und seit 1987 wird das Synchrotron nur noch als Injektor für diesen genutzt [Hil]. Typischerweise findet die Injektion in den ELSA-Ring bei $1{,}2\,\mathrm{GeV}$ statt. Tatsächlich wurden Modifikationen vorgenommen, die es nur noch ermöglichen, Elektronen im Synchrotron bis zu einer Energie von maximal $1{,}6\,\mathrm{GeV}$ zu beschleunigen. Somit erfüllt das Synchrotron die Aufgabe eines Vorbeschleunigers für den eigentlichen ELSA-Stretcherring und wird daher auch als Booster[1]-Synchrotron bezeichnet.

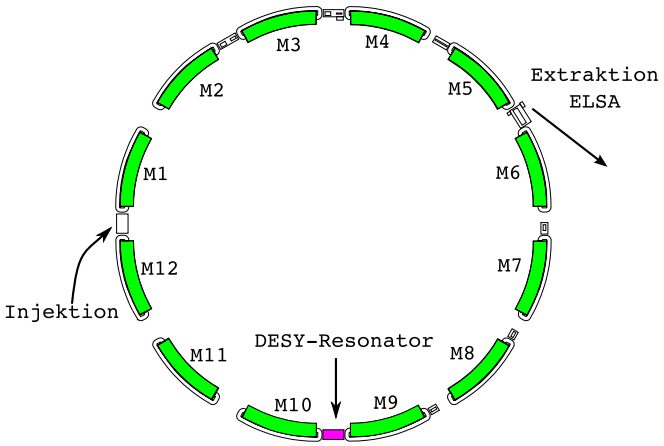


Abbildung 2.1.: Schematische Darstellung der Magnetstruktur des Synchrotrons.

[1] engl. verstärkern, vergrößern

B-Feld bei $2{,}3\,\mathrm{GeV}$	B_{max}	$1{,}003\,\mathrm{T}$
Umfang	L_{U}	$69{,}5\,\mathrm{m}$
Orbit-Radius	R	$7{,}65\,\mathrm{m}$
Umlauffrequenz	f_{U}	$4{,}3074\,\mathrm{MHz}$

Tabelle 2.1.: Einige wichtige Parameter des Booster-Synchrotrons.

2.1. Synchrotron-Magnete

Zwölf Magnete lenken den Elektronenstrahl auf eine Kreisbahn mit einem Umfang von ca. 70 Metern ab; dabei sind den Dipolfeldern der Ablenkmagnete zusätzlich noch Quadrupolfeldanteile zur Fokussierung überlagert. Bereits bei der Erzeugung des Elektronenstrahls hat dieser eine unvermeidbare Divergenz, und da die Elektronen in dem Synchrotron mehrere tausend Male[2] umlaufen, sind fokussierende Elemente unverzichtbar. Auf einen horizontal fokussierenden (F) Quadrupolanteil folgt jeweils ein defokussierender (D) Quadrupolanteil. Die Bereiche der Driftstrecken zwischen den fokussierenden Elementen werden dabei mit (O) bezeichnet. Eine Einheitszelle des Synchrotrons lässt sich so als O/2 (FD) O/2 identifizieren (vgl. [Hin97, S. 55]).

In Abbildung 2.2 ist ein Querschnitt durch einen Synchrotronmagneten gezeichnet. Zur Erregung des magnetischen Feldes sind in das C-förmige Magnetjoch vier Spulen mit jeweils 9 Wicklungen angebracht. Alle vier Spulen sind in Reihe geschaltet. Damit ergibt sich ein ohmscher Widerstand von $14\,\mathrm{m\Omega}$ und eine Induktivität von $30{,}9\,\mathrm{mH}$ für die Spulen eines Magneten. Um die maximale Magnetfeldstärke von $1{,}003\,\mathrm{T}$ zu erzeugen, wird ein Strom von $1380\,\mathrm{A}$ benötigt, weshalb die Spulen wassergekühlt sind.[Alt+68]

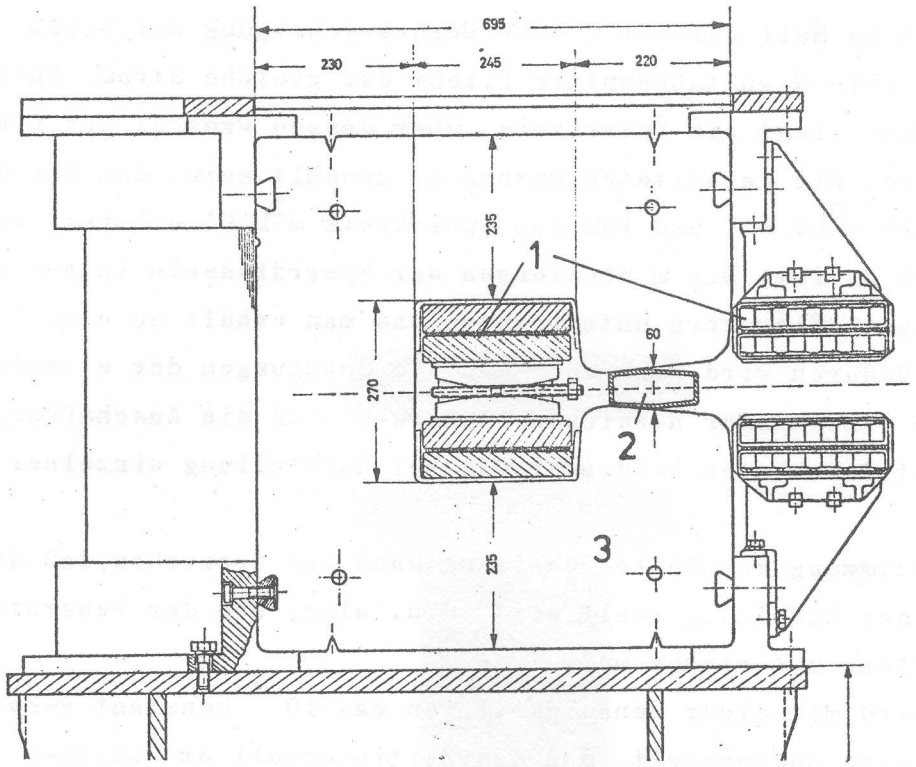


Abbildung 2.2.: Querschnitt durch einen Synchrotronmagneten [Alt+68, S. 13]
1. Magnetspulen, 2. Strahlkammer, 3. Magnetjoch.

[2] Die Umlaufzeit beträgt beim Booster-Synchrotron $232{,}16\,\mathrm{ns}$. Die Zeit zwischen Injektion und Extraktion beträgt $10\,\mathrm{ms}$, womit sich die Umlaufzahl zu $43\,073$ ergibt.

2.1.1. Zeitlicher Verlauf der Magnetfeldstärke

Wie in Kapitel 1.3 festgestellt wurde, muss der zeitliche Verlauf der magnetische Feldstärke in den Ablenkmagneten und die Beschleunigung der Elektronen angepasst sein. Dazu wird bei dem Booster-Synchrotron die Änderung der Ablenkmagnetfeldstärke durch das $50\,\mathrm{Hz}$-Stromnetz vorgeben; die Beschleunigungsspannung in dem Hohlraumresonator muss dann dieser Änderung folgen. Zunächst soll hier nun der zeitliche Verlauf der Magnetfeldstärke beschrieben werden.

Die Elektronen werden im LINAC auf $26\,\mathrm{MeV}$ vorbeschleunigt. Daraus ergibt sich, dass die Feldstärke in den Ablenkmagneten zum Zeitpunkt der Injektion in das Synchrotron $11{,}3\,\mathrm{mT}$ betragen[3] muss. Während der sich anschließenden Beschleunigungsphase auf die Endenergie von typischerweise $1{,}2\,\mathrm{GeV}$ muss das Feld bis zum Zeitpunkt der Extraktion das Maximum von $0{,}523\,\mathrm{T}$ erreicht haben. Bis zur nächsten Injektion wird die Feldstärke in den Ablenkmagneten wieder auf den zur Injektion notwendigen Wert heruntergefahren.

2.1.2. Stromversorgung der Magnete

Um den oben skizzierten Feldverlauf zu erzeugen, muss die Stromstärke in den Spulen der Ablenkmagnete kontrolliert geändert werden. Dazu ist die Stromversorgung der Magnete durch ein resonantes Netzwerk[4] realisiert worden, das netzsynchron mit der Frequenz des Stromnetzes von $f = 50\,\mathrm{Hz}$ schwingt. Die negative Halbwelle des Wechselstroms wird in den Spulen nicht benötigt, weswegen man dem Wechselstrom einen Gleichstrom überlagert:

$$I(t) = I_{\mathrm{DC}} - I_{\mathrm{AC}} \cos \omega t. \tag{2.1}$$

Die Amplituden I_{DC} und I_{AC} können beide geregelt werden, dabei wird ein fest vorgegebenes Verhältnis von $k = \frac{I_{\mathrm{AC}}}{I_{\mathrm{DC}}} \approx 1.0689$ automatisch eingehalten[5]. Der zeitliche Verlauf des Erregerstroms lässt sich dann mit k schreiben als:

$$I(t) = I_{\mathrm{DC}} \left(1 - k \cos \omega t\right). \tag{2.2}$$

Soll die Extraktionsenergie verändert werden, wird eine neue Gleichstromamplitude vorgegeben, die Wechselstromamplitude wird dann durch die Regelautomatik auf einen Wert nach obiger Gleichung (2.2) eingestellt. Die Summe der beiden Stromanteile hat dann als Maximum die Stromstärke, die dem benötigten Magnetfeld zum Extraktionszeitpunkt entspricht. Es ist hilfreich, den Erregerstrom in jedem Zyklus kurzzeitig negativ werden zu lassen. So ist sichergestellt, dass auch das magnetische Feld Nulldurchgänge hat. Aus diesem Grund wählt man k leicht größer als Eins. In Abbildung 2.3 ist der zeitliche Verlauf des Spulenstromes exemplarisch für eine Extraktionsenergie von $1{,}2\,\mathrm{GeV}$ dargestellt.

[3] vgl. Gleichung (1.4), als Radius R wurde der Magnetradius $R = 7{,}65\,\mathrm{m}$ verwendet. (vgl. Tabelle 2.1)

[4] Ein sogenanntes White-Netzwerk: Dabei werden Gruppen von Magnetspulen zusammen mit Kondensatoren zu Reihenschwingkreisen zusammengeschaltet. Diese Gruppen sind dann wiederum mit weiteren Kondensatoren über eine Drossel zu einem Parallelschwingkreis verschaltet. Die Resonanzfrequenz beträgt jeweils $f = 50\,\mathrm{Hz}$ [Alt+68, Kap. 4.5] Weiterführende Literatur: [WSON56][Bot90].

[5] Die Regelung funktioniert dabei nach folgendem Prinzip: Da der Wechselstromanteil etwas größer als der Gleichstromanteil ist, ergibt sich für eine kurze Zeit ein negativer Strom. Die Dauer zwischen den beiden Nulldurchgängen wird durch Änderung der Wechselstromamplitude auf $2{,}3\,\mathrm{ms}$ eingeregelt, was dem geforderten Verhältnis $k \approx 1{,}0689$ entspricht (vgl. Abbildung 2.3)

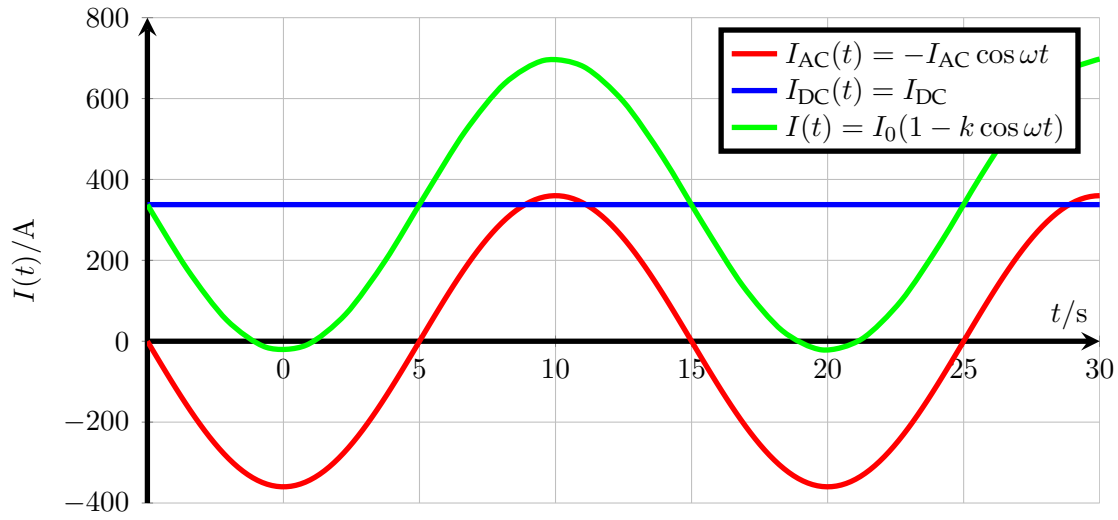


Abbildung 2.3.: Zeitlicher Verlauf des Stroms in den Magnetspulen für eine Extraktionsenergie von $1{,}2\,\mathrm{GeV}$.

2.1.3. Aus dem Magnetfeld abgeleitete Größen

Um die einzelnen Teile der Beschleunigeranlage ELSA zeitlich aufeinander abzustimmen, wird ein Referenzzeitpunkt für jeden Beschleunigungszyklus benötigt. Dazu wird ein Signal aus dem Nulldurchgang des magnetischen Feldes in den Ablenkmagneten des Synchrotrons abgeleitet. Dieses legt den Zeitpunkt des Nulldurchganges und damit einen definierten Referenzzeitpunkt fest. Des Weiteren wird für die Synchronisation von beschleunigendem elektrischen und ablenkenden magnetischen Feld ein zu $\dot{B}$ proportionales Signal benötigt (vgl. Kapitel 1.3 und Gleichung (1.24)). Beide Signale können direkt aus den Magnetfeldern der Dipolmagnete des Synchrotrons abgeleitet werden.

Ein zu $\dot{B}$ proportionales Spannungssignal wird in einer in einem der Magnetjoche angebrachten Spule induziert und zur Verfügung gestellt. Das Signal, welches den Nulldurchgang anzeigt, wird mit einem sogenannten Peaking-Strip[6] erzeugt.

Peaking-Strip

In einem der Synchrotron-Magnete ist eine Vorrichtung zur Detektion der Nulldurchgänge des magnetischen Feldes installiert. Die Funktionsweise dieser sogenannten Peaking-Strips soll nun beschrieben werden.

In ein Glasröhrchen ist ein kurzer Permalloy-Draht[7] (Durchmesser $0{,}05\,\mathrm{mm}$, Länge $25\,\mathrm{mm}$) eingebracht [Alt+68, Kap. 4.4]. Permalloy hat eine sehr hohe Permeabilität und gleichzeitig eine kleine magnetische Koerzitivfeldstärke. Die magnetische Permeabilität μ ist ein Tensor, der die magnetische Flussdichte B mit der magnetischen Feldstärke H verknüpft:

$$\vec{B} = \mu \vec{H}$$

[6] strip: englisch Streifen. Peaking-Strip könnte mit Verstärkungsstreifen übersetzt werden.
[7] Eine Legierung aus $80\,\%$ Ni und $4{,}5\,\%$ Mo.

Sie beträgt bei Permalloy typischerweise einige $10^5\ \frac{\mathrm{H}}{\mathrm{m}}$. In Abbildung 2.4 ist eine Hystereseschleife beispielhaft für eine Peaking-Strip-Anordnung skizziert. Diese Hysteresekurve stellt den Zusammenhang zwischen dem sich mit der Frequenz $f = 50\,\mathrm{Hz}$ harmonisch ändernden magnetischen Feldstärke der Ablenkmagnete $\vec{H}$ und der Flussdichte $\vec{B}$ in dem Peaking-Strip dar. Durch die sehr hohe Permeabilität verläuft die Hysteresekurve nahezu rechteckig.

Unter der magnetischen Koerzitivfeldstärke versteht man diejenige magentische Feldstärke, die benötigt wird, um einen Stoff vollständig zu entmagnetisieren. Da bei Permalloy die magnetische Koerzitivfeldstärke klein ist, wird die Hystereseschleife zusätzlich sehr schmal, daher gerät die Flussdichte $\vec{B}$ schon bei sehr kleinen magnetischen Feldstärken $\vec{H}$ in Sättigung. Bei den in Abbildung 2.4 mit H_1 und H_2 bezeichneten magnetischen Feldstärken tritt Sättigung auf, die Koerzitivfeldstärke ist mit H_c gekenzeichnet.

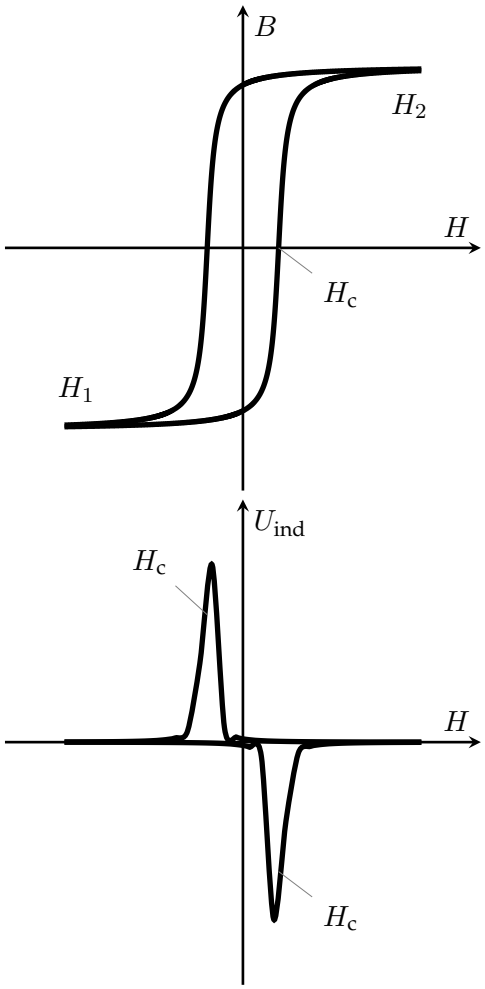


Abbildung 2.4.: Schematische Darstellung der Hystereseschleife in einem Peaking-Strip. H bezeichnet die magnetische Feldstärke, die durch den Strom in den Spulen des Ablenkmagneten erzeugt wird. B ist die magnetische Flussdichte in dem Peaking-Strip. Bei H_1 und H_2 tritt Sättigung auf, H_c ist die Koerzitivfeldstärke.
Im unteren Teil ist der Verlauf der Spannung, die in einer zusätzlich angebrachten Spule induziert wird gezeigt. (vgl. [Kel51][Nys59])

Um das Glasröhrchen ist eine Spule mit $N = 400$ Windungen angebracht, durch die zeitliche Änderung der magnetischen Flussdichte B wird in dieser Spule eine Spannung induziert:

$$\begin{aligned} U_{\text{ind}} &= -N\frac{\mathrm{d}\Phi}{\mathrm{d}t} \\ \Phi &= \int \vec{B}\,\mathrm{d}A \\ \Longrightarrow U_{\text{ind}} &= -N\dot{B}\cdot A \end{aligned}$$

Da der Permalloy-Draht schon bei kleiner äußeren magnetischen Feldstärke H in Sättigung gerät, wird nur während eines sehr kurzen Zeitraumes um den Nulldurchgang des Feldes der Ablenkmagnete überhaupt eine Spannung in der Spule induziert. Die Höhe der Spannung ist abhängig von $\dot{B}$, diese ist am Punkt H_c am größten. Man erhält daher den im unteren Teil von Abbildung 2.4 dargestellten Spannungsverlauf. Das Maximum des Spannungspulses wird nicht zum Zeitpunkt des Nulldurchganges, sondern zum Zeitpunkt der maximalen Änderung von B erzeugt. Die Zeitdifferenz zwischen dem Spannungspuls und dem eigentlichen Nulldurchgang ist durch die Koerzitivfeldstärke H_c vorgegeben. Das Signal dieser Spule wird durch eine Verstärkerschaltung aufbereitet und in einen TTL[8] kompatiblen Rechteckpuls umgewandelt.

2.2. Beschleunigungskomponenten

Im Folgenden sollen die für die Beschleunigung wichtigen Komponenten des Synchrotrons vorgestellt werden.

Die Hochfrequenzleistung wird in einem kommerziellen Fernsehsender erzeugt, dort in einem Klystron[9] verstärkt und über eine Koaxiale-Rohrleitung zum Resonator transportiert. Für das letzte Stück vor dem Resonator wird dabei auf einen Rechteckhohlleiter übergegangen, aus dem die Hochfrequenz über ein Keramikfenster in den Resonator eingekoppelt wird. [Sti67, Kap. 4]

2.2.1. Erzeugung der Hochfrequenzleistung

Um aus dem Synchrotron effektiv in den Beschleunigerring ELSA injizieren zu können, müssen beide Beschleuniger mit identischer Frequenz arbeiten, und zusätzlich muss die Phasenbeziehung zwischen den beschleunigenden Feldern beider Beschleuniger starr sein. Als Frequenz der für die Beschleunigung benötigten alternierenden Spannung sind $499{,}670\,\mathrm{MHz}$ festgelegt worden.

Ein Signalgenerator erzeugt ein sinusförmiges Signal mit einer einstellbaren Frequenz von $f = 499{,}670\,\mathrm{MHz}$ mit einer festen Amplitude von $10\,\mathrm{mW}$. Dieses Signal wird aufgeteilt und den Hochfrequenzverstärkern des Synchrotrons und denen des ELSA-Beschleunigers zugeführt (vgl. Abbildung 2.5). Dadurch, dass beide Hochfrequenzsysteme aus einer gemeinsamen Quelle gespeist werden, wird die geforderte Frequenz- und Phasenkopplung erfüllt.

[8] TTL =Transistor-Transistor-Logik

[9] Ein Klystron ist eine Elektronenröhre zur Verstärkung von elektromagnetischen Mikrowellen.

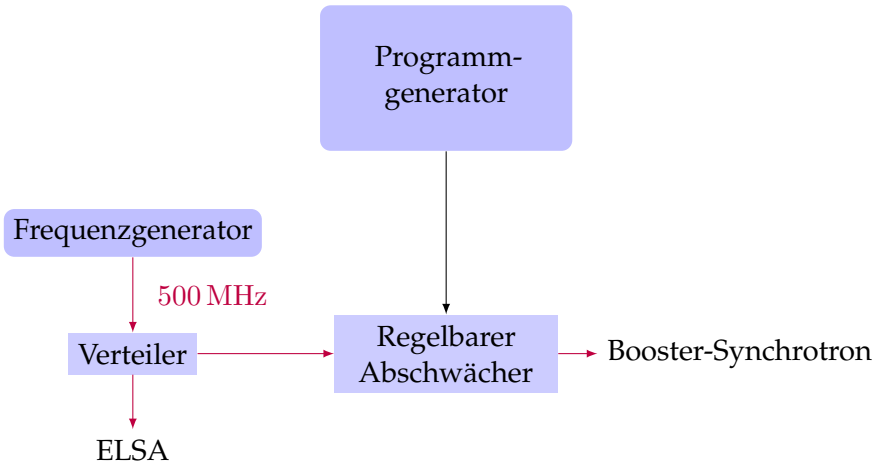


Abbildung 2.5.: Schematische Darstellung der Verteilung der Hochfrequenz.

Für das Synchrotron wird das $500\,\mathrm{MHz}$[10] Signal von einem einstellbaren Abschwächer gedämpft und anschließend von einem Festverstärker vorverstärkt. Dieses Signal wird dann mit einem Klystron auf die endgültige Leistung gebracht. Durch den Abschwächer wird die Eingangsamplitude des Verstärkers mit dem HF-Programm moduliert; die in den Resonator eingespeiste Leistung wird dadurch ebenso moduliert. Dieses Programm (vgl. Abschnitt 1.3.3) wird durch einen Programmgenerator (vgl. Kap. 4) erzeugt und stellt die notwendige Synchronisation zum Ablenkmagnetfeld her.

In seiner ursprünglichen Konfiguration konnte das Synchrotron bis zu einer Endenergie von $2{,}5\,\mathrm{GeV}$ beschleunigen. Dazu wird eine Hochfrequenzleistung im Resonator von etwa $80\,\mathrm{kW}$ benötigt. Als Verstärker wurde damals ein Fernsehsender aufgebaut, der mit einem speziell ausgewählten Klystron als Endstufe ausgestattet wurde. Im gepulsten Betrieb konnten bis zu $100\,\mathrm{kW}$ erzeugt werden. Heute wird im Standard-Betrieb nur bis $1{,}2\,\mathrm{GeV}$ beschleunigt, die benötige Leistung übersteigt dabei nie ca. $3\,\mathrm{kW}$[11].

2.2.2. Der Hohlraumresonator des Booster-Synchrotrons

Als Resonator wird ein dreizelliger, zylindersymmetrischer Holhraumresonator aus Kupfer verwendet. Ein Schnitt ist in Abbildung (2.6) gezeigt. Damit die Elektronen beim Durchqueren des Resonators in jeder der drei Zellen eine beschleunigende Spannung erfahren, muss der Phasenvorschub beim Durchgang durch eine Zelle $\Delta\varphi = \pi$ betragen. Die Zellenlänge l muss also jeweils der halben Wellenlänge entsprechen, bei $500\,\mathrm{MHz}$ ergibt sich daraus für die Längen

[10] Zur besseren Lesbarkeit wird hier vereinfachend von $500\,\mathrm{MHz}$ gesprochen. Die tatsächliche Frequenz wird während des Betriebes unter Umständen leicht geändert.

[11] Eine kalibrierte Leistungsmessung steht nicht zur Verfügung. Der hier angegebene Wert von $3\,\mathrm{kW}$ bezieht sich auf die Messung an einem Richtkoppler. Eine nachgeschaltete Messschaltung erzeugt daraus eine zur Leistung proportionale Spannung. Die Kenntnis des Zusammenhangs zwischen dieser Spannung und der tatsächlichen Leistung ist verloren gegangen, es existiert eine Umrechnung, die fest in einer SPS (Speicher-Programmierbare-Steuerung) einprogrammiert wurde. Ob diese Skalierung allerdings noch korrekt ist, kann zur Zeit nicht überprüft werden.

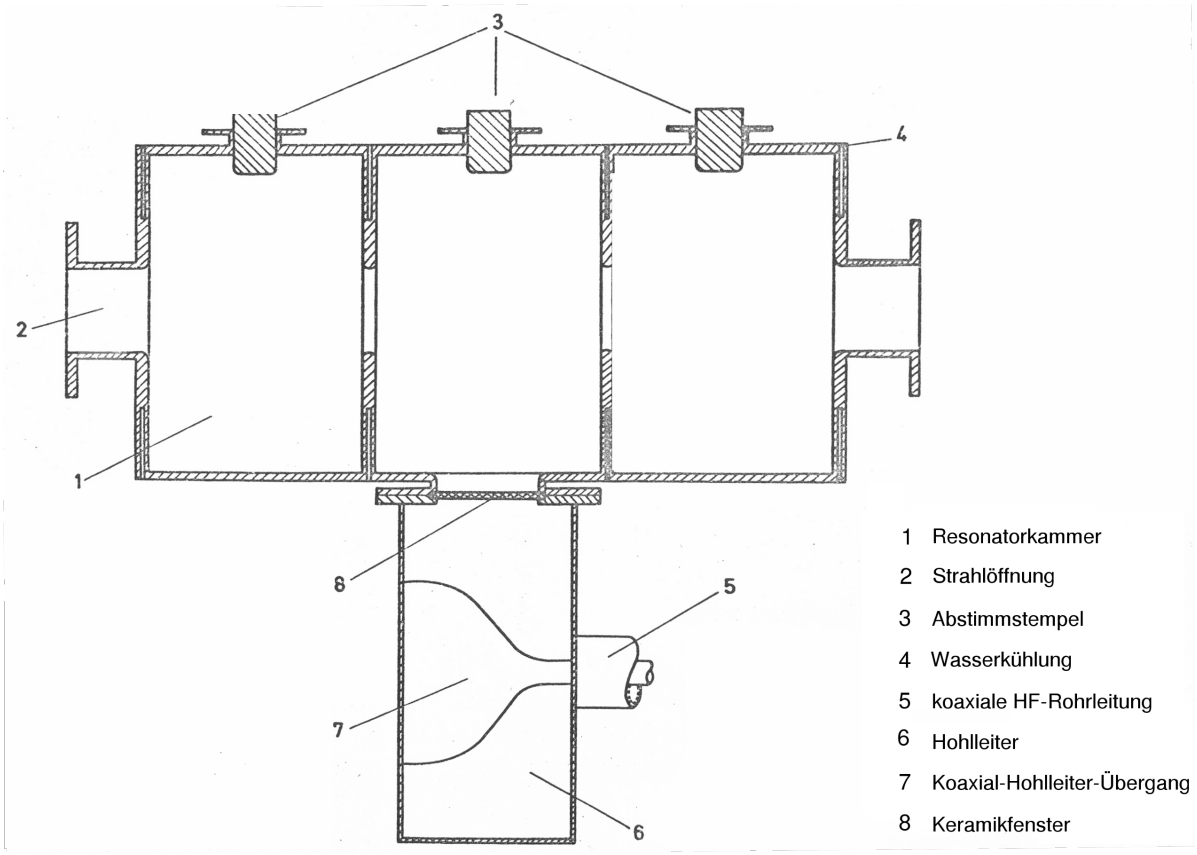


Abbildung 2.6.: Schnitt durch den Hohlraumresonator des Bonner Booster-Synchrotrons. [Sti67, S. 10]

der Resonatorzellen:

$$l = \frac{\lambda}{2} = \frac{\frac{c}{f_{\mathrm{HF}}}}{2} = 0{,}30\,\mathrm{m} \tag{2.3}$$

Aus der Bedingung $l = \frac{\lambda}{2}$ folgt direkt für den Strahlkopplungsfaktor: (vgl. Abschnitt 1.1.5 und Gleichung (1.15))

$$T = 0{,}6366 \tag{2.4}$$

Einkopplung

Die Hochfrequenzleistung wird über einen an die mittlere Zelle angekoppelten Rechteckhohlleiter eingespeist (vgl. [Sti67, S. 9]). Die drei Zellen sind kapazitiv miteinander gekoppelt. In jede der drei Zellen ist eine Koppelschleife eingebracht, die einen kleinen Teil der Leistung der jeweiligen Zelle nach außen koppelt und dort per koaxialem Kabel für Messzwecke zur Verfügung stellt.

Abstimmung der Resonanzfrequenz

Die in Abschnitt 1.1.2 eingeführten Größen Shuntimpedanz R_S und Güte Q wurden in [Sti67, Kap. 2.2 und Kap. 2.3] bestimmt:

$$Q_0 \approx 32\,000$$
$$R_S \approx 10\,\frac{\mathrm{M\Omega}}{\mathrm{m}}$$

Die hohe Güte des Resonators bedingt eine schmale Resonanzkurve (vgl. Abbildung 1.3). Dies hat zur Folge, dass schon kleine Abweichungen $\Delta\omega = \omega_0 - \omega$ zwischen anregender Frequenz und Resonanzfrequenz zum Einbruch des elektrischen Feldes führen. Da die Impedanz des Resonators ebenso von der Abweichung $\Delta\omega$ abhängt, führen kleine Abweichungen auch zu ansteigenden Reflexionen an der Einkopplung (vgl. Abschnitt 1.1.3 und Gleichung (1.7)). Die Resonanzfrequenz selbst ist allein abhängig von der Geometrie des Resonators (Gleichung (1.2)). Thermische Ausdehnungen haben daher einen direkten Einfluss auf die Resonanzfrequenz, was bedeutet, dass die Temperatur des Resonators aktiv stabilisiert werden muss. Zusätzlich bietet die Kontrolle der Temperatur eine direkte Möglichkeit, die Resonanzfrequenz zu steuern.

In jede der drei Zellen des Resonators kann ein Kupferstempel eingefahren werden. Die dadurch einstellbare Veränderung der Geometrie hat wiederum die Änderung der Resonanzfrequenz zur Folge. Die Stempel sind wassergekühlt und können einzeln über Schrittmotoren verfahren werden. Beide Verfahren werden standardmäßig zur Steuerung der Resonanzfrequenz verwendet.

Kapitel 3.

Inbetriebnahme einer automatischen Resonanzregelung mit Abstimmstempeln

Die Resonanzfrequenz des Hohlraumresonators kann, wie in Abschnitt 2.2.2 beschrieben, durch Stempel verändert werden. Mit einer Regelung ist es möglich, die aufgrund von Temperaturschwankungen auftretende Änderung der Resonatorgeometrie und die damit einhergehende Änderung der Resonanzfrequenz zu kompensieren. Zu Beginn dieser Arbeit war bereits eine Stempelsteuerungskomponente installiert, mittels derer zwar zwei der drei Abstimmstempel ferngesteuert verfahren werden konnten, aber noch keine Regelung der Resonanzfrequenz umgesetzt wurde.

3.1. Abstimmung der Resonanzfrequenz

Im Folgenden werden zunächst die zwei zur Verfügung stehenden Verfahren zur Abstimmung des Resonators, Temperaturregelung und Verschieben von Kupferstempeln, genauer untersucht. Im Anschluss daran wird die Funktion der Regelautomatik für die Stempel erläutert, mit der die Resonanzfrequenz konstant gehalten werden kann. Schließlich wird die erfolgreiche Inbetriebnahme an einigen Messergebnissen gezeigt.

3.1.1. Abstimmung durch Temperaturregelung

Abbildung 3.1 zeigt die Temperaturabhängigkeit der Resonanzfrequenz. Die aufgetragene Änderung der Resonanzfrequenz Δf bezieht sich auf die Grundfrequenz $f_0 = 499{,}667\,\mathrm{MHz}$:

$$\Delta f = f - f_0$$

Um die Abhängigkeit zu bestimmen, muss die Soll-Temperatur variiert und die davon abhängige Resonanzfrequenz gemessen werden. Das Verschwinden der relativen Phase gilt dabei als Kriterium, ob die Anregung des Resonators mit der Resonanzfrequenz geschieht (vgl. Abschnitt 1.1.4, Gleichung (1.11)). Für jeden Messpunkt muss dazu nach folgendem Schema vorgegangen werden:

- Soll-Temperatur variieren,
- Warten, bis sich die neue Ist-Temperatur eingestellt hat,
- Anregende Frequenz variieren, bis die relative Phase erneut Null wird. Dabei schwankt der Messwert der Phase bei erreichen des Optimums um den Nullpunkt. Die Resonanzfrequenz lässt sich daher nur in einem gewissen Toleranzbereich einstellen. Empirisch zeigt sich, dass die Frequenz mit etwa $\pm 1{,}5\,\mathrm{kHz}$ Ungenauigkeit eingestellt werden kann.

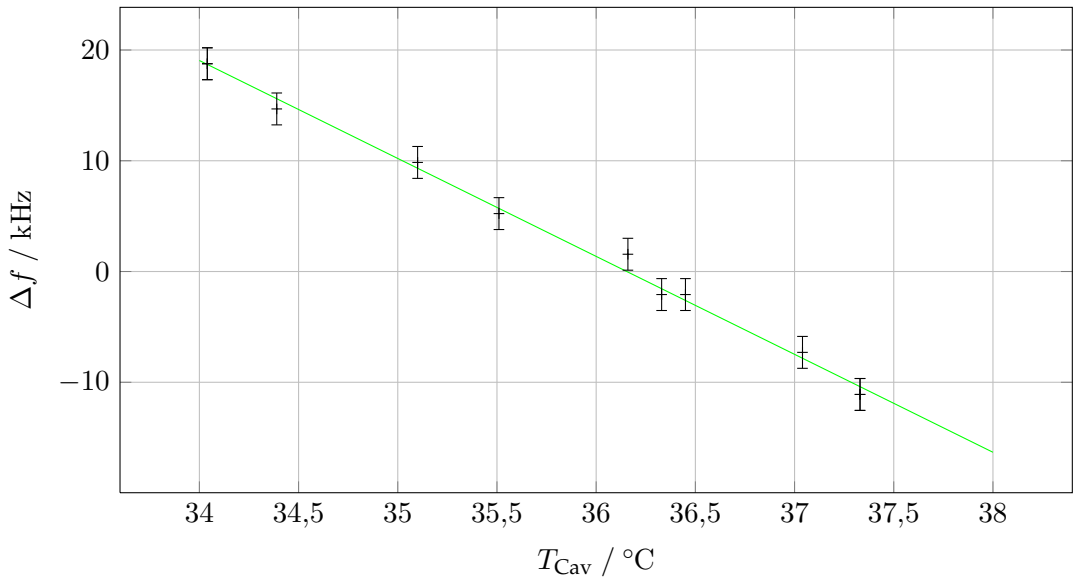


Abbildung 3.1.: Temperaturabhängigkeit der Resonanzfrequenz des im Booster-Synchrotron verwendeten Hohlraumresonators. Gezeigt ist die Abweichung von der standardmäßig verwendeten Frequenz $f_0 = 499{,}669\,\mathrm{MHz}$.

Die Analyse der Daten zeigt, dass ein linearer Zusammenhang anzunehmen ist. Ein durchgeführter Geradenfit[1] ergibt als funktionalen Zusammenhang:

$$\Delta f\,(T) = (-8{,}84 \pm 0{,}20)\,\frac{\mathrm{kHz}}{^\circ\mathrm{C}} \cdot T + (319 \pm 7{,}2)\,\mathrm{kHz} \tag{3.1}$$

Dieses Verhalten lässt sich mit der thermischen Ausdehnung des aus Kupfer gefertigten Resonators erklären. Die thermische Ausdehnung kann linear mit dem materialspezifischen Ausdehnungskoeffizienten ausgedrückt werden. Diese Ausdehnung überträgt sich direkt auf die, nur von der Geometrie abhängige Resonanzfrequenz (vgl. Gleichung (1.2)). Nach [Rei10, S. 8] kann die Veränderung der Resonanzfrequenz mit der Temperaturänderung verknüpft werden:

$$\Delta f = -\alpha \Delta T \cdot f_0.$$

Wird der Ausdehnungskoeffizient von Kupfer verwendet ($\alpha = 16{,}5 \times 10^{-6}\,\frac{1}{^\circ\mathrm{C}}$ vgl. [Rei10]) so erhält man bei einer Temperaturzunahme um $1\,^\circ\mathrm{C}$:

$$\Delta f = -8{,}245\,\mathrm{kHz}.$$

Dieses Ergebnis ist gut verträglich mit der oben durchgeführten Messung. Die Temperaturabhängigkeit der Resonanzfrequenz lässt sich also mit der temperaturbedingten Ausdehnung des Resonators erklären. Für den Hohlraumresonator des Synchrotrons wird eine Temperaturstabilisierung verwendet. Auf Basis eines Soll/Ist-Vergleichs wird das Kühlwasser des

[1] Dazu wird ein numerisches Verfahren benutzt, das die quadratischen Abweichungen der fehlergewichteten Messpunkte zur anzupassenden Funktion minimiert.

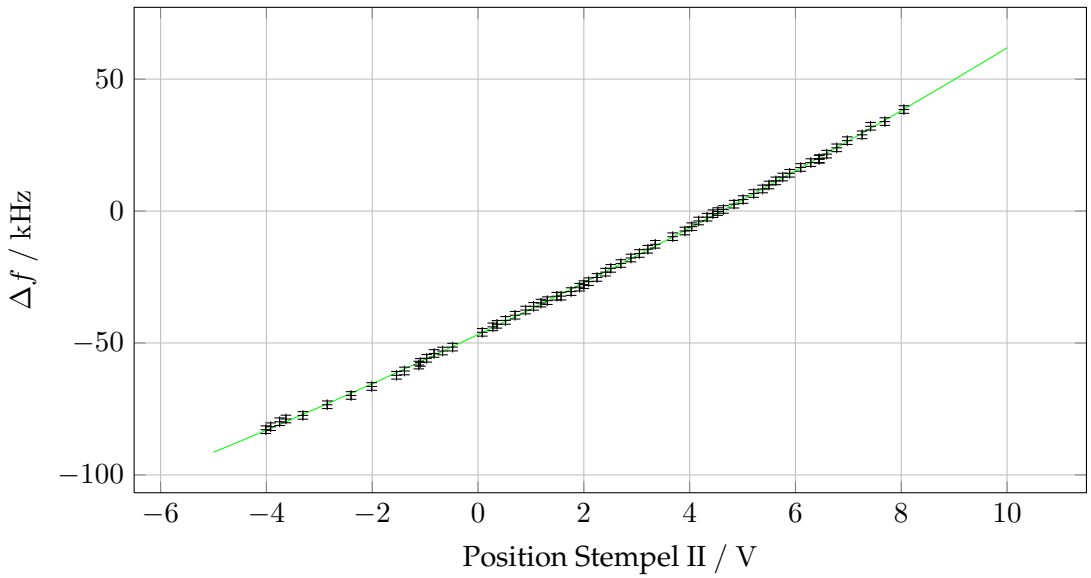


Abbildung 3.2.: Abweichung der Resonanzfrequenz in Abhängigkeit von der Position der zwei Stempel. Die Position des Stempels II wird als Spannung über einem Potentiometer abgegriffen, welches fest mit der Stempelposition verschoben wird.

Resonators mit warmem Wasser gemischt, so dass eine konstante Temperatur gewährleistet wird. Gegenwärtig wird als Solltemperatur $T_{\text{Soll}} = 36{,}5\,^{\circ}\text{C}$ verwendet.

3.1.2. Abstimmung mit Kupferstempeln

Die eingesetzte Stempelsteuerung kann maximal zwei der drei Stempel verfahren, standardmäßig werden diese dabei symmetrisch bewegt. Da nur zwei Stempel angesteuert werden können, werden aus Symmetriegründen nur die beiden äußeren Stempel angesteuert, der Stempel der mittleren Zelle wird fixiert und nicht zur Steuerung benutzt. Für den dritten Stempel muss daher eine optimale Position gefunden werden in der dieser dann verbleibt. Diese Fragestellung wird in Abschnitt 3.2.3 behandelt.

Analog zum vorherigen Abschnitt wird die Abhängigkeit der Resonanzfrequenz von der Position der Stempel vermessen. Die Stempelposition kann nur indirekt gemessen werden: Über einem Potentiometer, das mechanisch an den Verfahrweg des Stempels gekoppelt ist, fällt eine Spannung ab, die linear zur Auslenkung des Stempels ist. Da die tatsächliche Position des Stempels im eingebauten Zustand nicht nachprüfbar ist, wird im Folgenden anstelle der Position in Metern immer die entsprechende Spannung in Volt verwendet[2]. Für den Resonanzabgleich werden beide Stempel synchron angesteuert. Es genügt daher die Position nur eines Stempels anzugeben[3].

Die Ergebnisse einer Messung sind in Abbildung 3.2 dargestellt. Dabei wurden beide Stempel über den gesamten zur Verfügung stehenden Weg verschoben. Der Spannungsabfall an dem Potentiometer kann nicht als rein linear angenommen werden, insbesondere in der Nähe

[2] Auch im Kontrollsystem der Beschleunigeranlage wird die Stempelposition daher nur in Volt angezeigt.

[3] Es zeigt sich, dass die Anzeige der Stempelspannung des ersten Stempels unzuverlässig ist, daher wird generell die Position des zweiten Stempels benutzt.

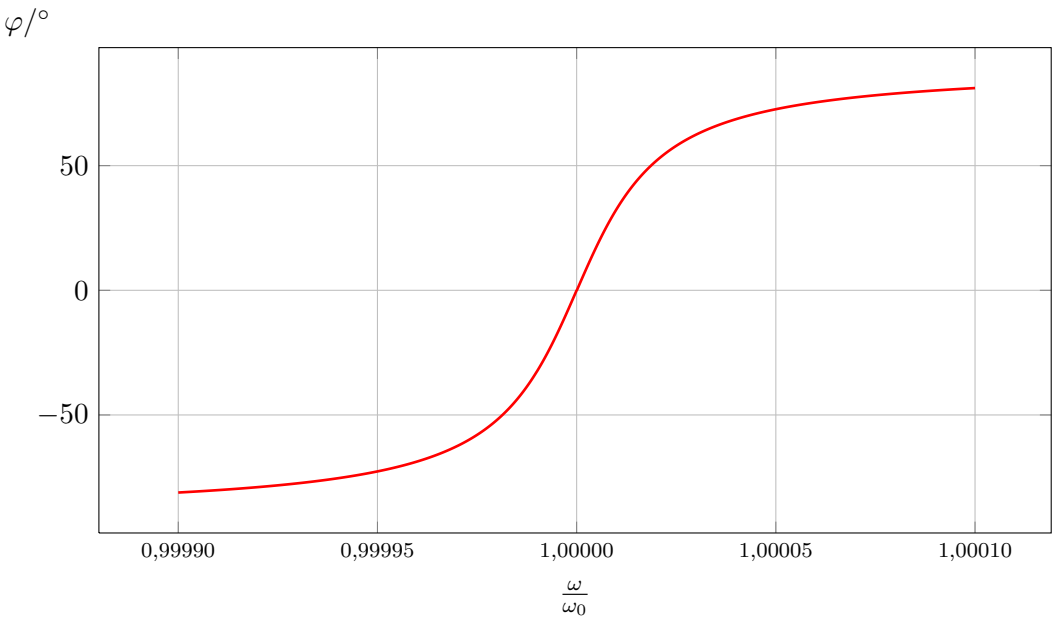


Abbildung 3.3.: Verlauf der relativen Phase in Abhängigkeit der anregenden Frequenz ω. Für die Berechnung wurde dabei die Güte des verwendeten Resonators $Q = 32\,000$ eingesetzt. (vgl. Abschnitt 1.1.4 und Abbildung 1.3).

der Endpositionen weist der Spannungsabfall quadratische Anteile auf. Der gesuchte Zusammenhang lässt sich daher aus einer linearen Funktion, der ein leichter quadratischer Anteil überlagert wird, gewinnen:

$$f(U) = a + b \cdot U + c \cdot U^2 \tag{3.2}$$

Eine durchgeführte Anpassung der Funktion $f(U)$ an die gemessenen Daten ergibt:

$$\Delta f(U) = -\left(46\,801 \pm 91\right)\mathrm{Hz} + \left(9570 \pm 31\right)\frac{\mathrm{Hz}}{\mathrm{V}} \cdot U + \left(129 \pm 6\right)\frac{\mathrm{Hz}}{\mathrm{V}^2} \cdot U^2. \tag{3.3}$$

Durch Verschieben der Stempel lässt sich die Resonanzfrequenz über einen Bereich von $\Delta f \approx 100\,\mathrm{kHz}$ verstimmen.

Zusammen mit der Temperaturstabilisierung stehen damit zwei unterschiedliche Werkzeuge zur Verfügung, um den Resonator abzustimmen.

3.2. Stempel-Regelautomatik

Im Abschnitt über den Phasengang eines Parallelschwingkreises bei resonanter Anregung (vgl. Abschnitt 1.1.4) wurde gezeigt, dass die Phase zwischen der Anregung und der Resonanzamplitude bei Anregung mit der Resonanzfrequenz exakt Null beträgt. Bei hohen Güten wird der Phasenverlauf zusätzlich sehr steil. Das Verschwinden der relativen Phase ist also ein guter Indikator für das Vorliegen von resonanter Anregung. In Abbildung 3.3 ist der Verlauf der relativen Phase für die Güte des verwendeten Resonators gezeigt: ($Q = 32\,000$ [Sti67, S. 12]). Beim Queren der Resonanzfrequenz tritt zusätzlich ein Vorzeichenwechsel auf, daher lässt

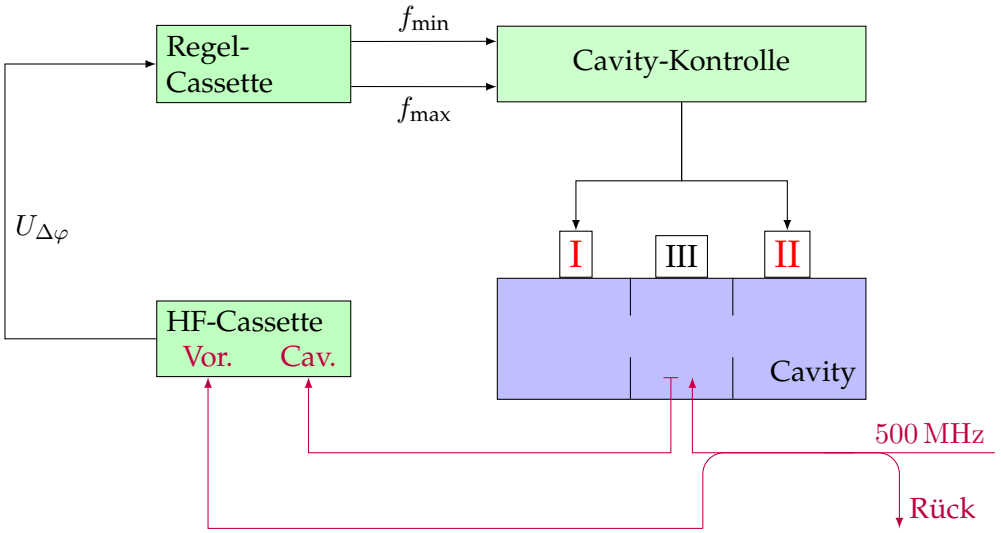


Abbildung 3.4.: Vereinfachte Darstellung der Signale zur automatischen Regelung der Resonanzfrequenz. In der HF-Cassette wird die zur Phasendifferenz proportionale Gleichspannung $U_{\Delta\varphi}$ erzeugt. Die Regel-Cassette erzeugt abhängig von $U_{\Delta\varphi}$ Spannungspulse. Diese werden in die Cavity-Kontrolle geleitet und dort in Pulse für die Schrittmotoren von Stempel I und II umgewandelt. Stempel III kann nicht angesteuert werden.

sich die relative Phase ideal als Regelgröße verwenden.

Die elektronische Funktionsweise der Stempelsteuerung soll nicht Gegenstand dieser Arbeit sein, hier wird nur ein kurzer Überblick über die prinzipielle Arbeitsweise gegeben (vgl. dazu Abbildung 3.4). Ziel der Schaltung ist es, ein von den Amplituden der Eingangsgrößen unabhängiges, zur relativen Phase proportionales Gleichspannungssignal $U_{\Delta\varphi}$ zu erzeugen. Dieses Signal wird in einer Regelautomatik, abhängig vom Vorzeichen der Phasenspannung, in Regelpulse umgewandelt. Die Zahl der Pulse pro Sekunde wird dabei dynamisch an den Betrag der Abweichung angepasst, so dass eine größere Abweichung mehr Pulse pro Sekunde zur Folge hat. Diese Pulse werden schließlich in Ansteuerpulse für die Schrittmotoren der Stempel umgewandelt. Damit führt dieser kontinuierliche Prozess zur Minimierung der relativen Phase und somit zur automatischen Optimierung der Resonanzfrequenz.

3.2.1. Eingangsgrößen der Regelautomatik

Aus einem Richtkoppler[4], der im Rechteckhohlleiter direkt vor der Einkopplung in den Hohlraumresonator angebracht ist, wird ein kleiner Anteil der zum Resonator hin laufenden Hochfrequenzleistung entnommen und über ein Koaxialkabel der Regelschaltung zugeführt. Dieses Signal wird mit `Vorlauf` bezeichnetet, es weist die gleiche Phasenlage wie die der Anregung auf. Außerdem wird aus einem direkt daneben angebrachten zweiten Richtkoppler ein Teil der an der Einkopplung reflektierten Leistung ausgekoppelt.

Aus der mittleren Zelle des Hohlraumresonators wird mit der dort angebrachten Messschleife ein kleiner Teil der Leistung in dieser Zelle über ein zweites Koaxialkabel der Regelung

[4]Der Richtkoppler ist ein Bauteil der Hochfrequenztechnik und dient dazu, aus einer Leitung einen Teil der darin laufenden elektromagnetischen Wellen richtungsabhängig abzuzweigen.

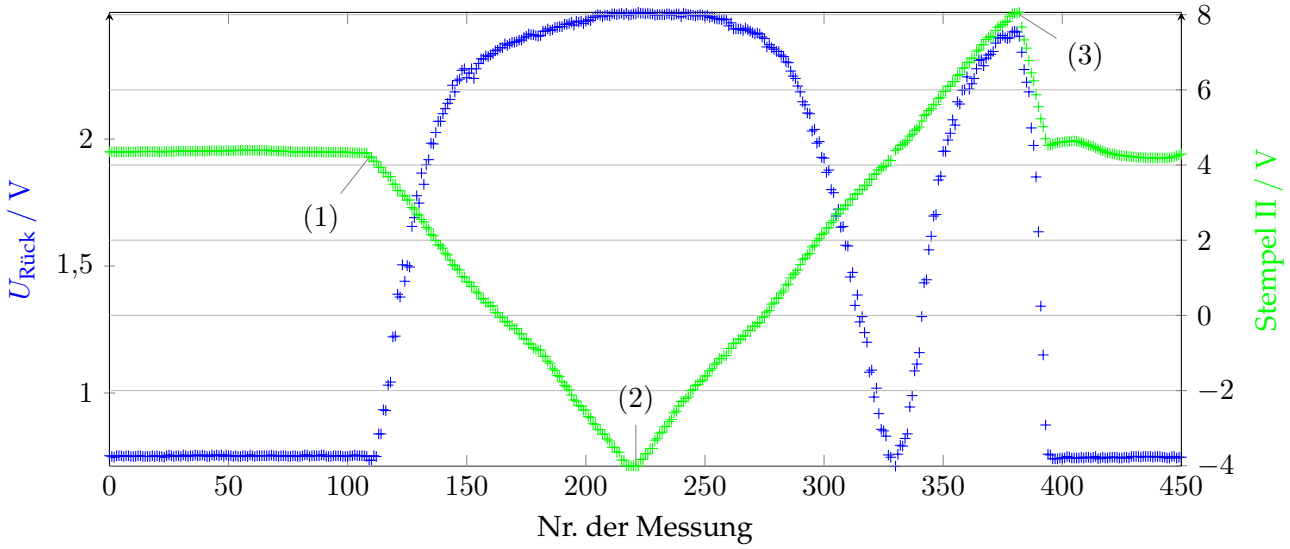


Abbildung 3.5.: Gezeigt ist die zur reflektierten Leistung proportionale Spannung $U_{\text{Rück}}$ (in blau) und die zur Position des 2. Stempels proportionale Spannung (in grün). Bis (1) ist die Regelung eingeschaltet. Danach wurden beide Stempel bis zur minimalen (2) von dort bis zur maximalen (3) Position verfahren. Anschließend wurde die Regelung wieder aktiviert.

zugeführt. Dieses Signal wird `Cavity`-Signal[5] genannt und weist die gleiche Phasenlage wie das Hochfrequenzfeld im Resonator auf. Die Koaxialkabel werden parallel geführt, um die relative Phase beider Signale nicht zu beeinflussen. Mit einem einstellbaren Phasenschieber kann der dennoch immer auftretende Laufzeitunterschied beider Signalzweige ausgeglichen werden, das dazu nötige Vorgehen wird im folgenden Abschnitt genauer beschrieben. Alle Signale werden dem Kontrollsystem der Beschleunigeranlage digitalisiert zur Verfügung gestellt.

3.2.2. Inbetriebname der Regelung

Bevor die Regelautomatik in Betrieb genommen werden kann, müssen die Signalzweige `Vorlauf` und `Cavity` hinsichtlich ihrer Phasendifferenz kalibriert werden. Die Phasendifferenz der Signale direkt am Eingang der Regelautomatik muss der Differenz entsprechen, die zwischen dem Hinlaufenden Hochfrequenzfeld und dem Feld im Resonator vorliegt. Nur dann kann aus dem Verschwinden der Phase auf das Vorliegen von resonanter Abstimmung geschlossen werden.

Um diese Kalibrierung vorzunehmen, kann die Stempelsteuerung selbst benutzt werden. Dazu wird Hochfrequenzleistung in den Hohlraumresonator eingespeist und die an der Einkopplung reflektierte Leistung beobachtet. Mit Hilfe der Stempel wird der Hohlraumresonator verstimmt, bis das Minimum der reflektierten Leistung gefunden wird. Nach den Überlegungen im Abschnitt zum Reflexionsfaktor (Abschnitt 1.1.3) wird die reflektierte Leistung bei resonanter Anregung minimal; im optimalen Fall tritt keine Reflexion mehr auf. Hat man eine Stempeleinstellung gefunden, bei der die Reflexion minimal wird, ist der Resonator folglich

[5] Cavity (engl.) Hohlraumresonator.

bestmöglich auf die anregende Frequenz abgestimmt. Ein in den `Vorlauf`-Signalzweig fest eingebauter Phasensteller wird nun so eingestellt, dass die zur Phasendifferenz direkt proportionale Spannung $U_{\Delta\varphi}$ Null wird[6]. Nach erfolgreicher Durchführung dieses Verfahrens sind beide Signalzweige im Phasengang aneinander angepasst und die Spannung $U_{\Delta\varphi}$ entspricht der tatsächlichen Phasendifferenz.

Anschließend wird die gefundene Einstellung überprüft. In Abbildung 3.5 ist das Ergebnis einer solchen Messung graphisch dargestellt. Während der gesamten Messung wird in den Hohlraumresonator Hochfrequenz mit einer nominellen Leistung von $1{,}5\,\mathrm{kW}$ eingespeist und auf diesem Wert konstant gehalten. Zunächst ist die Resonanzregelung über einen längeren Zeitraum eingeschaltet. Für diesen Zeitraum ist die reflektierte Leistung konstant. Um zu zeigen, dass dieses die minimal auftretende Leistung ist werden anschließend die Stempel, bei ausgeschalteter Resonanzregelung über ihren gesamten Bereich verfahren. Wie erwartet steigt das Signal der reflektierten Leistung an. Erst wenn während des Rückwegs die Stempel erneut die Ausgangsposition durchfahren, wird diese wieder minimal. Da die Stempel danach weiter fahren, steigt die reflektierte Leistung wieder an. Nachdem die maximale Position erreicht wurde, wird die Resonanzregelung eingeschaltet und die Regelung fährt die Stempel so schnell wie möglich zurück in eine Position, bei der die relative Phase verschwindet. Dies ist die Ausgangsposition und die reflektierte Leistung ist erneut minimal.

Anhand der gezeigten Abbildung lässt sich belegen, dass zum einen der Phasenabgleich der Signalzweige `Vorlauf` und `Cavity` korrekt durchgeführt wurde, zum anderen zeigt sich, dass die Resonanzregelung funktioniert.

3.2.3. Studien zu Stempel III

Mit der installierten Stempelsteuerung können lediglich zwei Stempel angesteuert werden. Daher muss entschieden werden, auf welche Position der dritte Stempel dauerhaft eingestellt werden soll. Die optimale Position des dritten Stempels muss gefunden werden. Dazu wird untersucht, welchen Einfluss dieser auf die reflektierte Leistung bei resonanter Anregung hat.

Analyse des Einflusses von Stempel III auf die reflektierte Leistung

Der Resonator wird konstant mit $1{,}5\,\mathrm{kW}$ Hochfrequenzleistung angeregt. Dabei bleibt die Regelung eingeschaltet, die Stempel I und II werden also automatisch so eingestellt, dass die relative Phase zwischen `Vorlauf` und `Cavity` Signal verschwindet; der Resonator wird also auf die anregende Frequenz abgestimmt. Nun wird der dritte Stempel langsam in den Resonator eingefahren[7]. Dies geschieht schrittweise, wobei nach jedem Schritt gewartet wird, bis die Resonanzregelung mit den beiden äußeren Stempeln auf die Bewegung des dritten Stempels reagiert hat. In Abbildung 3.6 ist gezeigt, wie die reflektierte Leistung und die Position des zweiten Stempels auf die neuen Positionen des dritten Stempels reagieren: Zu Beginn der Messung ist Stempel III völlig aus dem Resonator heraus gefahren. Da der automatische

[6] Eine oft schnellere Methode mit vergleichbarem Resultat benutzt den Phasensteller direkt: Dazu wird die Resonanzregelung aktiviert und der Phasensteller variiert. Nach jeder Variation muss kurz gewartet werden, bis die Resonanzregelung die Phasendifferenz ausgeglichen hat. Es ist wiederum das Minimum der reflektierten Leistung zu suchen. Ist dieses gefunden, so ist der Phasensteller automatisch optimal eingestellt.

[7] Um den dritten Stempel ansteuern zu können, wurde eine weitere Stempelsteuerung provisorisch installiert. Da diese nicht an das Kontrollsystem der Beschleunigeranlage angeschlossen werden konnte, liegen leider keine digitalen Werte der Position von Stempel II vor.

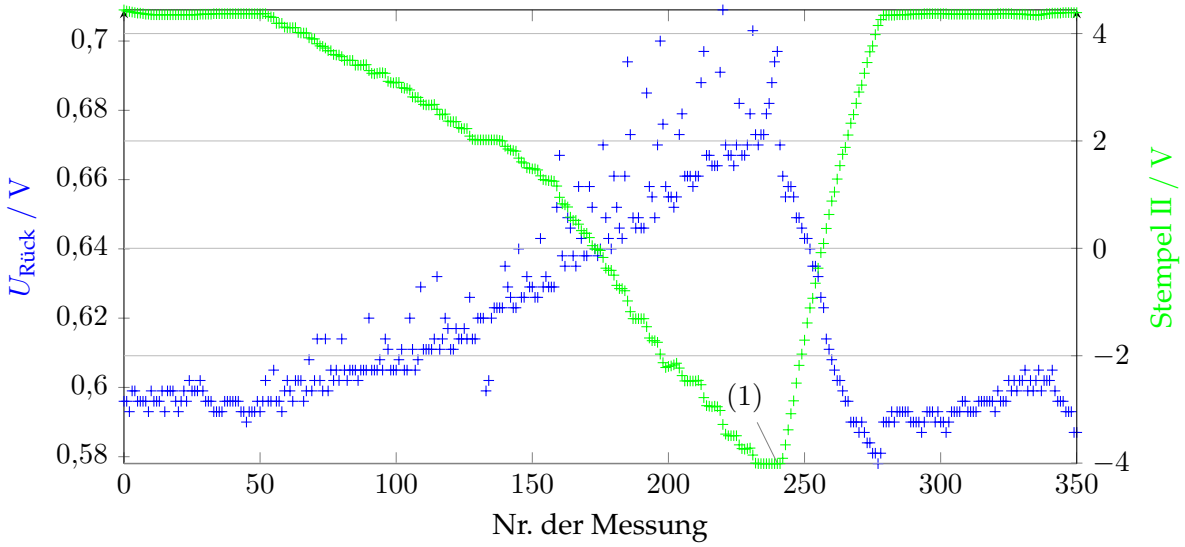


Abbildung 3.6.: Einfluss des mittleren Stempels (Stempel III). Die automatische Resonanzregelung ist eingeschaltet und der dritte Stempel wird schrittweise in den Resonator hinein gefahren. Nach jedem Schritt wird gewartet, bis die Regelung eine neue Position für die zwei anderen Stempel gefunden hat, bei welcher der Resonator wieder abgestimmt ist. Gezeigt sind hier die zu der an der Einkopplung reflektierten Leistung proportionale Spannung U_{Refl} in blau und die zur Position von Stempel II proportionale Spannung in grün. Ab (1) wurde der dritte Stempel wieder schnell aus dem Resonator heraus gefahren. (Die Position von Stempel III ist hier nicht gezeigt.)

Resonanzabgleich mit den beiden anderen Stempel eingeschaltet ist, bleibt die reflektierte Leistung konstant. Dann wird der dritte Stempel wie beschrieben schrittweise in den Resonator hineingefahren. Die Reglung reagiert auf diese Verstimmung des Resonators, indem die beiden anderen Stempel herausgefahren werden. Der Resonator ist wieder abgestimmt. Allerdings ist zu beobachten, dass die reflektierte Leistung nicht mehr das absolute Minimum erreicht. In der Tat lässt sich beobachten, dass bei weiterem Fahren von Stempel III das erreichbare Minimum der reflektierten Leistung kontinuierlich ansteigt. Daher wird der mittlere Stempel dauerhaft auf seiner minimalen Position belassen.

3.2.4. Verstimmung des Resonators zur Robinson-Dämpfung

In Abschnitt 1.2.2 wurde die Robinson-Dämpfung eingeführt und die Forderung, den Resonator kapazitiv zu verstimmen, abgeleitet. Um diese Verstimmung dauerhaft einzustellen, wird der in den `Vorlauf`-Singnalzweig eingebaute Phasensteller um einige Grad verstellt. Bei eingeschalteter Resonanzregelung wird diese zusätzliche relative Phase ausgeglichen. Im Ergebnis wird der Resonator von der Automatik nicht mehr auf die Resonanzfrequenz, sondern auf eine um die vorgenommene Verstellung abweichende Frequenz abgestimmt. Die Wahl der Verstellung hängt von zwei Punkten ab: Die Verstimmung darf nicht zu groß gewählt werden, sonst treten an der Einkopplung unzulässige Reflexionen auf. Außerdem muss das Vorzeichen der Verstellung so gewählt werden, dass der Resonator zu kleineren Eigenfrequenzen ω_0 verstimmt wird, andernfalls würde man eine induktive Verstimmung vornehmen. Diese würde die Synchrotronschwingung anregen und zu Strahlverlust führen.

Aus Abbildung 3.3 ist ersichtlich, dass die Verstellung negativ vorgenommen werden muss, denn ein negatives $\Delta\varphi$ wird von der Automatik, durch Verstimmung des Resonators zu kleineren Frequenzen ω_0, ausgeglichen – was der geforderten kapazitiven Verstimmung entspricht. Um die Größenordnung der Verstellung zu kontrollieren, wird ein Signalzweig mit einem Leitungsstück verlängert, dessen Phasenverzögerung bekannt ist. Die Erzeugung der zur relativen Phasen proportionalen Spannung geschieht in der Stempelsteuerungseinheit derart, dass wenn in den `Cavity`-Signalzweig ein Leitungsstück mit der Phasenverzögerung σ eingebaut wird, daraus eine positive Spannung $U_{\Delta\varphi} \hat{=} + \sigma$ entsteht. Für die geforderte kapazitive Verstimmung wird daher ein $12{,}5\,\mathrm{mm}$ langes SMA-Verlängerungsstück mit ungefähr $10°$ Phasengang (bei $499{,}669\,\mathrm{MHz}$) in den `Cavity`-Zweig eingebaut. Anschließend wird mit dem Phasensteller die zur Phasendifferenz proportionale Spannung $U_{\Delta\varphi}$ wieder auf Null eingestellt. Danach wird das Verlängerungsstück wieder entfernt. Wird die Resonanzregelung nun wieder aktiviert, so regelt diese jetzt auf eine neue Resonanzfrequenz, die einer um $-10°$ verschobenen relativen Phase entspricht; der Resonator ist kapazitiv verstimmt. Um die Größe der Frequenzverschiebung abschätzen zu können, wird diese aus der Gleichung für die relative Phase (1.11) berechnet:

$$\begin{aligned} \tan\varphi &= Q \cdot \left(\frac{\omega_0}{\omega} - \frac{\omega}{\omega_0} \right) \\ \Longrightarrow \frac{\omega}{\omega_0} &= \frac{\sqrt{\tan^2\varphi + 4Q^2} - \tan\varphi}{2Q} \end{aligned}$$

Einsetzen von $Q = 32\,000$, $\omega_0 = 2\pi \cdot 499{,}669\,\mathrm{MHz}$ und $\varphi = 10°$ liefert für die Abweichung von der Resonanzfrequenz:

$$\omega_0 - \omega \approx 1{,}4\,\mathrm{kHz} \tag{3.4}$$

3.3. Status der Regelung

Abschließend ist in Abbildung 3.7 anhand einiger Messwerte gezeigt, wie mit der Regelung der Stempel die auftretenden Temperaturschwankungen erfolgreich kompensiert werden. Bei eingeschalteter Temperaturstabilisierung mit vorgebender Soll-Temperatur $T_{\mathrm{Soll}} = 36{,}5\,°\mathrm{C}$ ist eine langsame Oszillation der Ist-Temperatur um den Sollwert mit einer Zeitkonstante in der Größenordnung von 10 Minuten zu beobachten. Die Ursache dieser Schwankung liegt vermutlich in der aktiven Stabilisierung der Temperatur, was allerdings noch nicht weiter untersucht worden ist.

Für die ersten 20 Minuten der Messung war die Stempelautomatik eingeschaltet. Die Regelung kompensiert die mit der Temperaturänderung einhergehende Änderung der Resonanzfrequenz. Dies ist anhand der Messwerte der Position von Stempel II nachvollziehbar. Die reflektierte Leistung und ebenso die Leistung im Resonator bleibt konstant. Nach etwa 20 Minuten wurde die Regelung der Stempel ausgeschaltet, die Temperaturschwankung überträgt sich nun direkt auf die Leistungen. Nachdem die Regelung erneut eingeschaltet wurde, wird die Oszillation wieder kompensiert.

Die Resonanzregelung ist erfolgreich in Betrieb genommen und kann von nun an standardmäßig im Beschleunigerbetrieb eingesetzt werden.

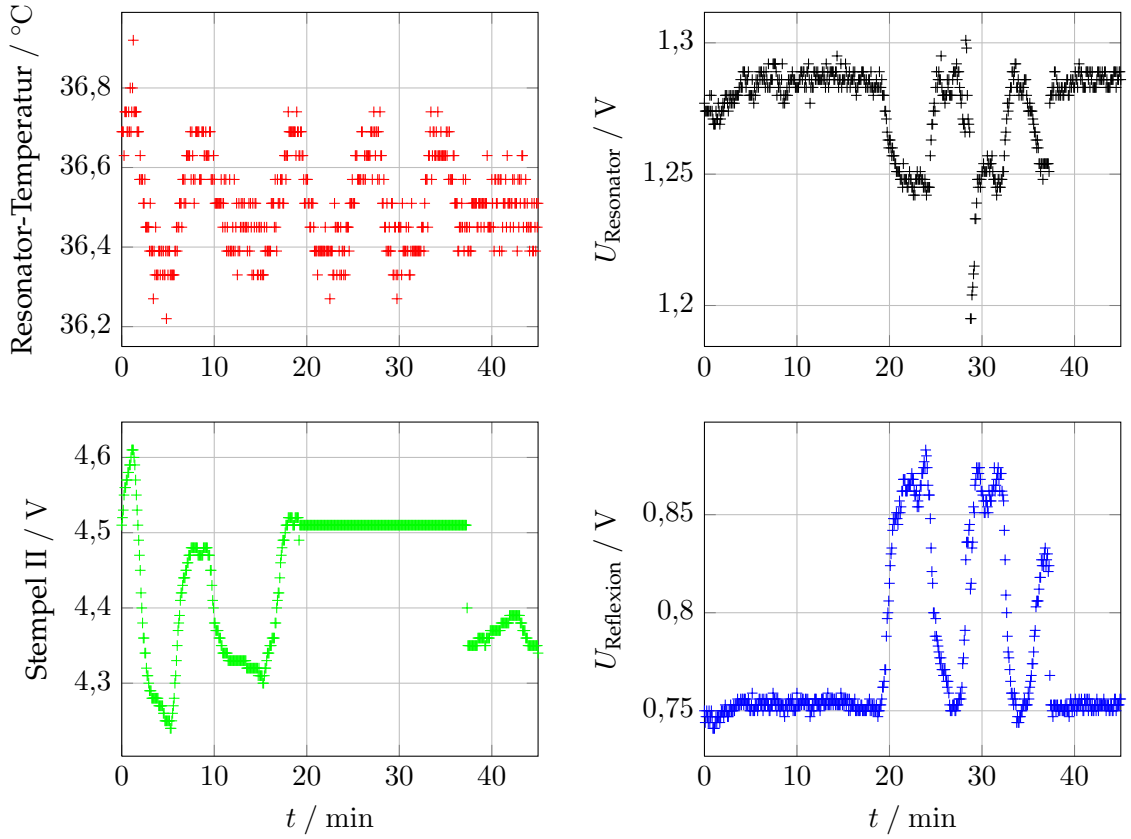


Abbildung 3.7.: Oben links ist die Entwicklung der Resonator-Temperatur gezeigt, die Temperatur oszilliert langsam, die Periodenlänge beträgt ungefähr 10 min.
Unten links ist die Position des zweiten Stempels gezeigt, für die ersten 20 Minuten und ab Minute 35 ist die Resonanzregelung eingeschaltet und die Stempel kompensieren die Temperaturschwankung. Dies kann in den beiden Graphen auf der rechten Seite beobachtet werden: Während die Regelung aktiv ist, bleiben die reflektierte Leistung und die Leistung im Resonator beide konstant. Ist die Regelung der Stempel ausgeschaltet (ab etwa Minute 20) überträgt sich die Temperaturschwankung direkt auf diese Größen.

Kapitel 4.

Aufbau eines neuen HF-Programmgenerators

Gegenstand dieses Kapitels ist der Aufbau einer neuen elektronischen Schaltung zur Ansteuerung der Feldstärke im Hohlraumresonator des Synchrotrons. Die Feldstärke muss ständig an die ablenkende Magnetfeldstärke angepasst werden (vgl. Abschnitt 1.3), um die notwendige Synchronisation herzustellen. Da beim Synchrotron die Ablenkmagnete synchron mit dem $50\,\mathrm{Hz}$-Stromnetz erregt werden, ist die zeitliche Änderung der Ablenkmagnetfeldstärke fest vorgegeben. Die neu aufzubauenden Schaltung soll, die Feldstärke im Resonator daher angepasst an die Änderung der Ablenkmagnetfeldstärke regeln. Als Ausgangsgröße wird ein Spannungsverlauf ausgegeben, der proportional zur Soll-Feldstärke im Resonator ist. Dieser Spannungsverlauf wird HF-Programm genannt, die Schaltung dementsprechend HF-Programmgenerator.

Zunächst wird der theoretische Verlauf des HF-Programms näher betrachtet, danach wird auf die Ansteuerung der Hochfrequenz-Anlage eingegangen. Abschließend wird dann die neu aufgebaute Schaltung detailliert beschrieben.

4.1. Theoretische Berechnung des Programms

Bereits in Abschnitt 1.3 wurde mit Gleichung (1.29) die grundsätzliche Vorschrift zur Herstellung der geforderten Synchronisation abgeleitet:

$$U = c_{\dot{B}} \cdot \dot{B} + c_{B^4} \cdot B^4$$

Aus den Gleichungen 1.27 und 1.28 lassen sich weiterhin Vorschriften für die Vorfaktoren $c_{\dot{B}}$ und c_{B^4} gewinnen:

$$c_{\dot{B}} = \frac{c \cdot R \cdot t_U}{T} \frac{1}{\sin \varphi_1} \tag{4.1}$$

$$c_{B^4} = \frac{e^5 R^3}{3\epsilon_0 \left(m_0 \cdot c\right)^4 \cdot T} \frac{1}{\sin \varphi_2} \tag{4.2}$$

Setzt man die Parameter aus Tabelle 2.1 in diese Gleichung ein, ergeben sich folgende Vorfaktoren:

$$c_{\dot{B}} = 2445{,}32\,\frac{\mathrm{V\,s}}{\mathrm{T}}, \tag{4.3}$$

$$c_{B^4} = 705\,847{,}14\,\frac{\mathrm{V}}{\mathrm{T}^4}. \tag{4.4}$$

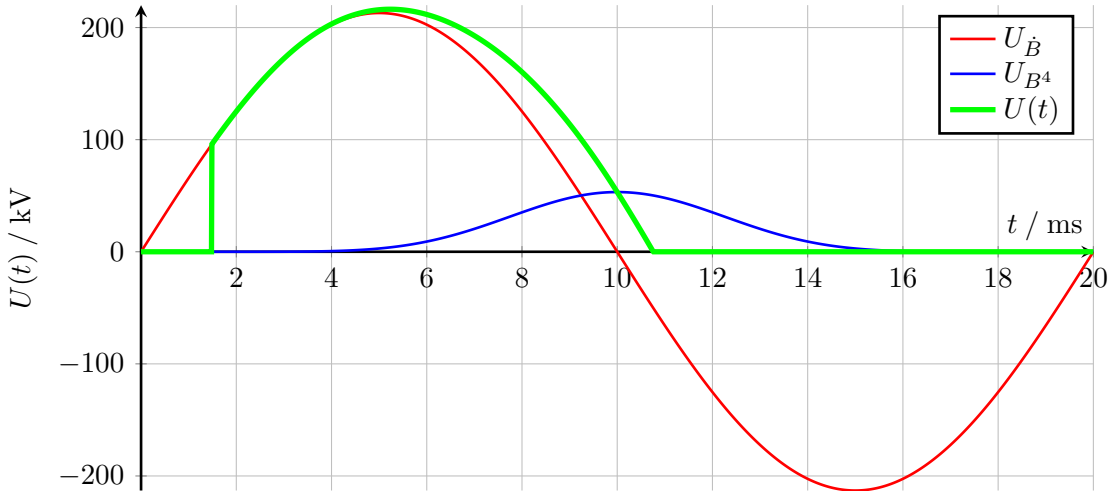


Abbildung 4.1.: Berechnete HF-Beschleunigungsspannung für eine Extraktionsenergie von $1{,}2\,\mathrm{GeV}$.

Dabei wurden die in [Net68, S. 6] angebenden Sollphasen

$$\varphi_1 = 20^\circ \quad \text{und} \quad \varphi_2 = 45^\circ$$

verwendet. Die standardmäßig verwendete Extraktionsenergie von $1{,}2\,\mathrm{GeV}$ verlangt im Maximum eine ablenkende Magnetfeldstärke von $0{,}52\,\mathrm{T}$. In Abbildung 4.1 ist für diese Bedingungen die benötigte Beschleunigungsspannung gezeigt. Da in das Synchrotron über mehrere Umläufe injiziert wird, darf bis zum Abschluss der Injektion keine Hochfrequenz in den Resonator eingespeist werden. Daher wird das HF-Programm erst nach Abschluss der Injektion gestartet.

Der geforderte Spannungsverlauf ist abhängig von der maximalen Magnetfeldstärke und daher abhängig von der Extraktionsenergie. In Abbildung 4.2 sind die Spannungsverläufe für verschiedene Extraktionsenergien gezeigt. Mit Anwachsen der Energie wächst vor allem der Anteil, der für die Kompensation der Synchrotronstrahlungsverluste verantwortlich ist.

4.2. Ansteuerung der Hochfrequenzsendeanlage

Mit dem aufzubauenden Programmgenerator soll die Resonatorspannung geregelt werden. Über die Shuntimpedanz ist diese mit der in den Resonator eingespeisten Leistung verknüpft (siehe Gleichung (1.4)). Zur Ansteuerung wird daher die Eingangsleistung des Klystrons mit dem HF-Programm moduliert. Die verstärkte Ausgangsleistung wird dann zum Resonator geleitet und wie beschrieben dort eingekoppelt. In Abbildung 4.3 ist diese Ansteuerung schematisch dargestellt: Nach der Verteilung der Hochfrequenz muss diese am Eingang der Schaltung zuerst verstärkt werden. Anschließend wird das Signal durch einen einstellbaren Abschwächer geleitet, dieser wird von der Programmgenerator-Schaltung angesteuert und moduliert die Hochfrequenzleistung. Das Ausgangssignal wird dann erneut zur Pegelanpassung verstärkt und ist nun so verstärkt worden, dass es auf den Leistungsteil gegeben werden

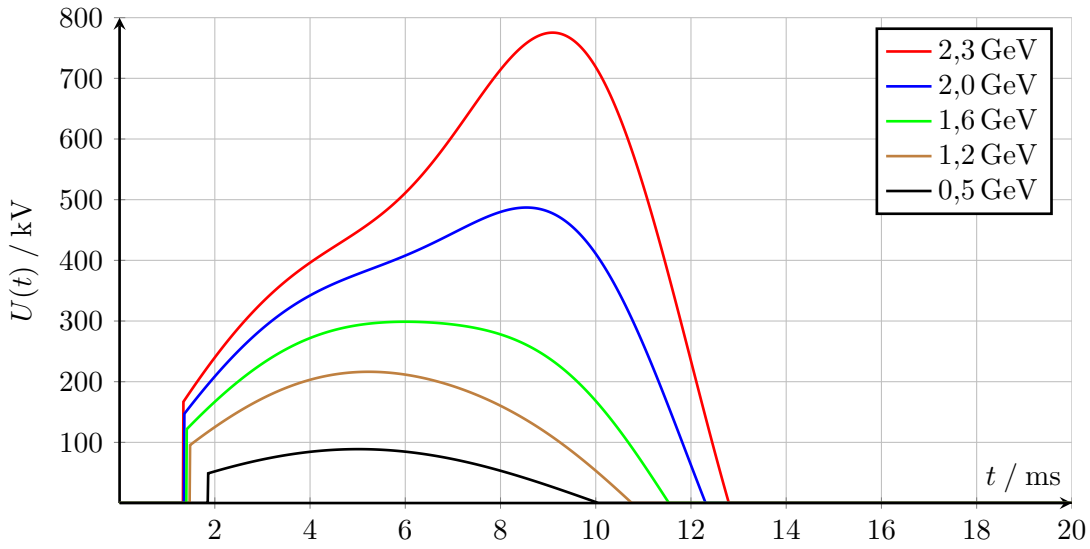


Abbildung 4.2.: Berechnetes Amplitudenprogramm für verschiedene Extraktionsenergien.

kann. Mit der Ansteuerung des Abschwächers kann so die Ausgangsleistung des Klystrons in einem Bereich von $0\,\mathrm{kW}$ bis weit über $10\,\mathrm{kW}$ gesteuert werden.

Fällt die Stromversorgung der Verstärker aus, so schwächen die Verstärker die Signale jeweils um ca $16\,\mathrm{dB}$ ab. Selbst wenn der Abschwächer in diesem Fall so angesteuert wird, dass nicht abgeschwächt wird, liegt am Eingang der Leistungsstufe des Klystrons ein Pegel an, der sicher kleiner als $-48\,\mathrm{dBm}$ ist. So ist sichergestellt, dass bei Ausfall der Stromversorgung keine Leistung am Ausgang des Klystrons anliegt.

4.3. Regelung der Beschleunigungsspannung

Die elektrische Feldstärke im Hohlraumresonator soll geregelt werden. Dazu wird die von dem Programmgenerator vorgegebene *Soll*-Feldstärke mit der *Ist*-Feldstärke im Hohlraumresonator verglichen. Aus der Abweichung von *Soll*- und *Ist*-Größe wird dann eine Ansteuerungsspannung, die sogenannte *Stell*-Größe, für den regelbaren Abschwächer abgeleitet. Durch die Änderung der *Stell*-Größe ändert sich die Ansteuerung der Regelstrecke und damit wiederum die *Ist*-Größe. Der Regler ist korrekt eingestellt, wenn er die Abweichung von *Soll*- und *Ist*-Wert möglichst schnell ausgleicht.

Als Regler wird ein analog aufgebauter PI-Regler[1] verwendet. Die Funktionsweise eines solchen Reglers ist schematisch in Abbildung 4.4 dargestellt. Die Differenz aus *Soll*- und *Ist*-Wert wird in zwei getrennten Schaltungsteilen verstärkt. Im Proportionalteil wird die Differenz um einen einstellbaren Faktor k_p verstärkt. Der Integrationsteil liefert das zeitliche Integral der Abweichung, wobei die Integrationszeitkonstante[2] t_i eingestellt werden kann. Die Ausgangsgrößen der beiden Schaltungsteile werden dann addiert und bilden die neue Stellgröße. Die Parameter k_p und t_i müssen für den vorliegenden Anwendungsfall optimiert

[1] PI-Regler: Proportional-Integral-Regler.
[2] Diese Zeitkonstante ist gleichbedeutend mit einem einstellbaren Vorfaktor vor dem Integral.

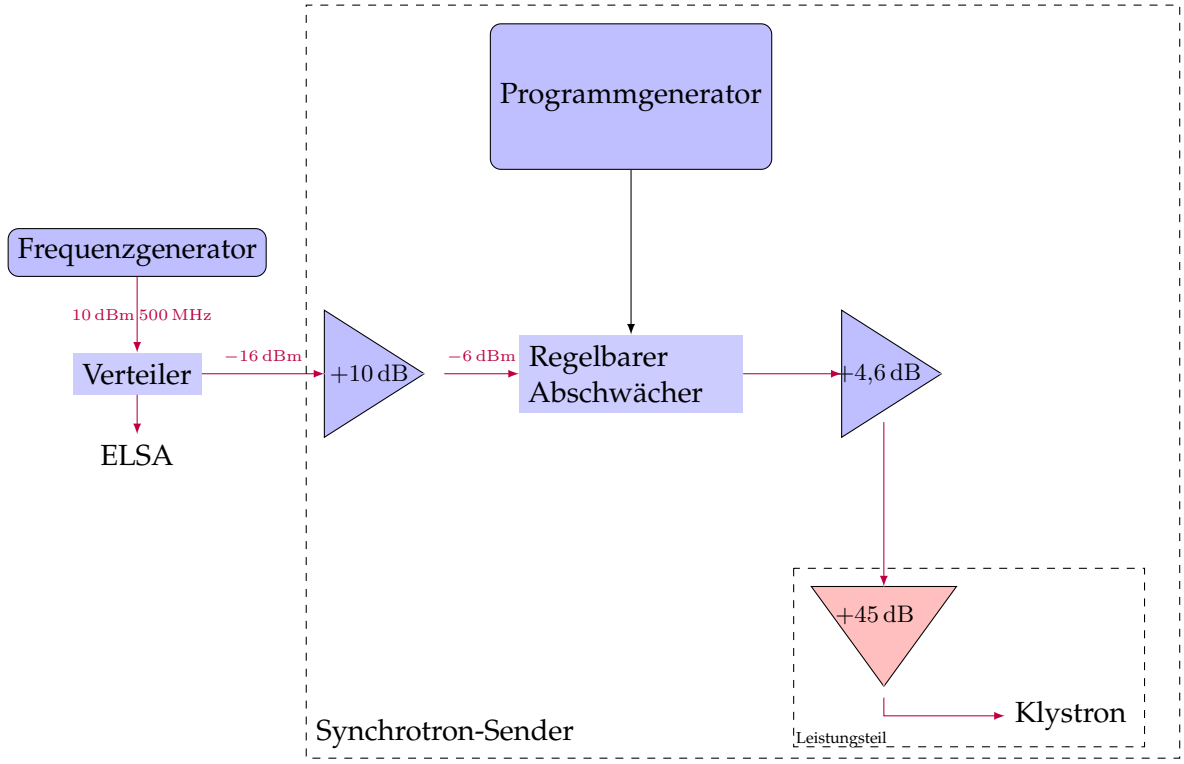


Abbildung 4.3.: Schematische Darstellung der Verteilung der Hochfrequenz.

werden. Ein optimal eingestellter Regler steuert die Regelstrecke so an, dass die Abweichung, auch bei sich ändernder *Soll*-Größe, möglichst rasch verschwindet.

Im vorliegenden Fall wurde der Regler empirisch optimiert. Dazu wird ein rechteckiger *Soll*-Wertverlauf vorgeben und die Antwort der *Ist*-Größe beobachtet. Die Parameter k_p und t_i werden einzeln so optimiert, dass sich ein möglichst rechteckiger *Ist*-Verlauf einstellt. Wird einer der Parameter zu groß gewählt, führt die *Ist*-Größe Schwingungen aus, was unerwünscht ist. Wird eine der Größen jedoch zu klein gewählt, ist der Anstieg des *Ist*-Signals zu flach. Die Prozedur wird bei verschiedenen *Soll*-Werten, die Ausgangsleistungen am Klystron zwischen $0{,}5\,\mathrm{kW}$ und maximal $6\,\mathrm{kW}$ entsprechen, wiederholt.

In Abbildung 4.5 ist gezeigt, wie der PI-Regler eingestellt wurde. Auf den rechteckig vorgebenden *Soll*-Wertverlauf reagiert der Regler mit dem gezeigten Verlauf der *Stell*-Größe.

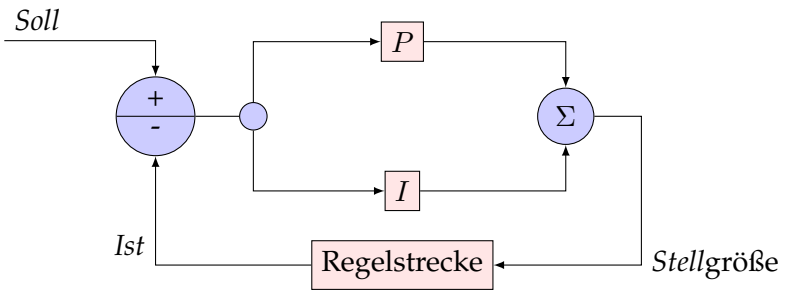


Abbildung 4.4.: Schematische Darstellung der Funkionsweise eines Proportional-Integral-Reglers.

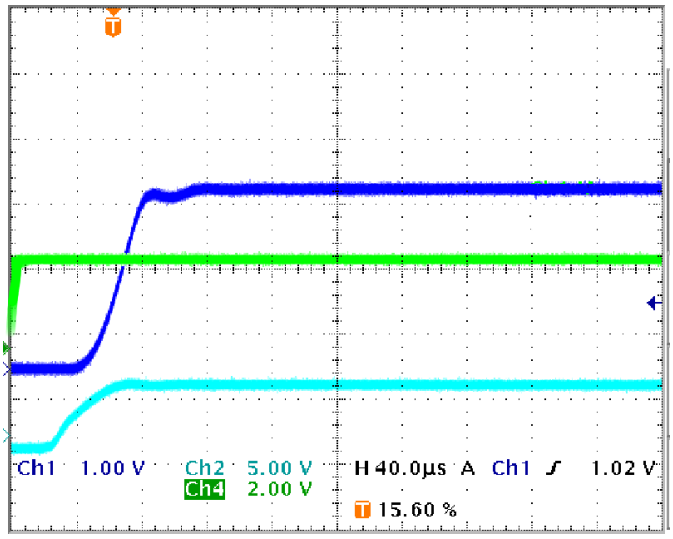


Abbildung 4.5.: Antwort des PI-Reglers auf einen vorgegebenen rechteckigen *Soll*-Wertverlauf (in Grün). Der *Ist*-Wert, die zur Leistung im Resonator proportionale Spannung $U_{\text{Cav.}}$, ist in (Blau) gezeigt. Die untere Kurve (in Cyan) stellt die zur Vorlaufleistung proportionale Spannung U_{Vor} dar. Dies ist die *Stell*-Größe des PI-Reglers.

Optimiert wird der gezeigte *Ist*-Wert-Verlauf, der möglichst rechteckig sein soll. Wird einer der beiden Regelparameter in dieser Situation verändert, gerät der *Ist*-Wert sofort in Schwingung. Ausgehend von den so gefundenen Einstellungen der beiden Parameter, werden diese weiter optimiert. Dazu wird z.B. k_{p} verkleinert, t_{i} wird dann weiter vergrößert, um die Reduktion von k_{p} zu kompensieren. Danach kann t_{i} weiter vergrößert werden, bis sich ein möglichst steiler Anstieg einstellt und weiteres Vergrößern von t_{i} zu Schwingungen führt. Auch die andere Richtung muss untersucht werden. Nach endlich vielen Iterationen stellt sich eine, soweit mit empirischen Mitteln erreichbare, optimierte Situation ein.

4.4. Die Programmgenerator-Schaltung

Die Vorfaktoren des HF-Programms und die gesamte Steuerung sollen durch das Kontrollsystem der Beschleunigeranlage vorgegeben werden können. Daher muss die Schaltung selbst digital realisiert werden. Dies bedeutet vor allem auch, dass Integration und Potenzierung digital durchgeführt werden. Es wurde daher entschieden die Berechnung auf einem Mikrocontroller[3] durchzuführen. Dementsprechend muss die Eingangsspannung digitalisiert werden. Das berechnete Programm schließlich muss wieder in eine analoge Spannung umgewandelt werden.

Als Ausgangspunkt für die neu aufzubauende Schaltung wird eine Experimentierplatine mit zwei Mikrocontrollern benutzt. Ausgehend von dieser Platine wird die Schaltung als Prototyp schrittweise aufgebaut. Nachdem die gesamte Schaltung und die Programme der Mikrocontroller erfolgreich im Labor getestet wurden, konnte die Produktion einer gedruckten

[3]Ein Mikrocontroller ist ein kleiner Rechner, der in einem einzigen integrierten Halbleiter-Chip aufgebaut ist. Neben dem Prozessorkern sind auch der Speicher und programmierbare Ein- und Ausgänge zur Anbindung an die Peripherie integriert.

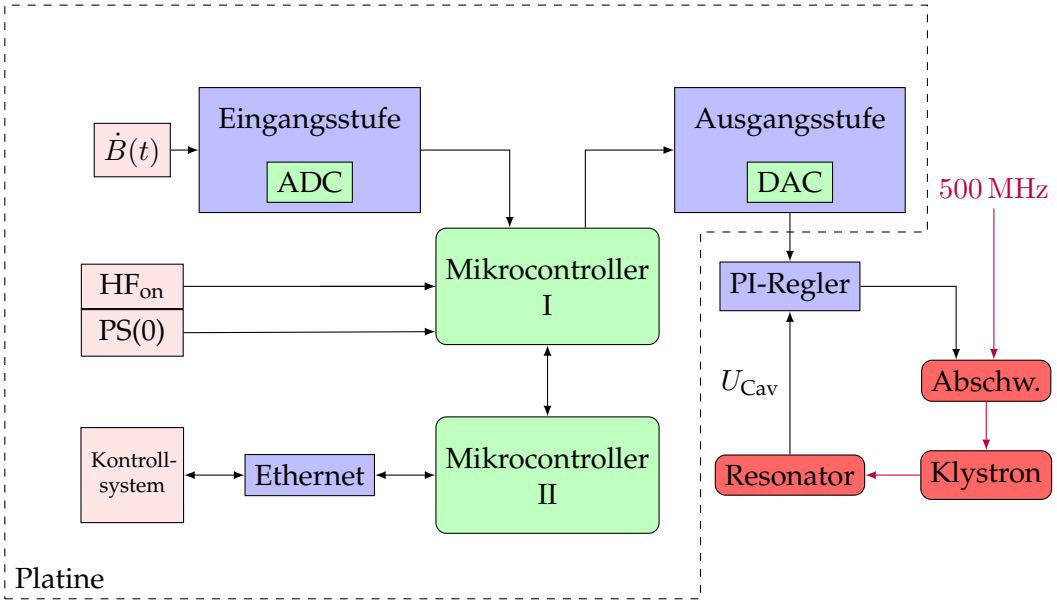


Abbildung 4.6.: Schematische Darstellung der Schaltungsmodule der Programmgenerator Schaltung (innerhalb des gestrichelt gekennzeichneten Bereichs). Zusätzlich ist symbolisch die Ansteuerung der Hochfrequenzleistungsstufe mit einem PI-Regler gezeigt.

Schaltung in Auftrag gegeben werden. Dazu muss zunächst aus dem elektrischen Schaltplan des Prototypen ein Layout für die gedruckte Schaltung entwickelt werden. Eine Ansicht des Layouts befindet sich im Anhang B, der elektrische Gesamt-Schaltplan der Platine ist ebenfalls dort eingefügt.

Als Eingangsgröße für die Berechnung des HF-Programms steht eine zur zeitlichen Änderung des Magnetfeldes proportionale Spannung zur Verfügung (vgl. Abschnitt 2.1.3). Zusätzlich wird eine zur vierten Potenz der Magnetfeldstärke proportionale Größe benötigt. Diese muss durch Integration und anschließender Potenzierung aus der Eingangsspannung gewonnen werden. Die beiden Werte werden dann mit Vorfaktoren gewichtet zusammen mit einem Offset aufaddiert und bilden das HF-Programm.

Um den Nullpunkt des Integrals festzulegen, steht ein Signal zur Verfügung, das den Nulldurchgang des Magnetfeldes anzeigt: **PS(0)**[4] (vgl. Abschnitt 2.1.3). Die Ausgabe des HF-Programms darf erst starten, wenn die Injektion abgeschlossen ist. Dazu wird vom Kontrollsystem ein TTL-Signal erzeugt, das diesen Zeitpunkt anzeigt. (**HF$_{on}$**)

Abbildung 4.6 zeigt die einzelnen Komponenten der Schaltung in ihrer logischen Verknüpfung. Anhand dieser Abbildung werden nun im Folgenden die einzelnen Schaltungsteile vorgestellt.

4.4.1. Eingangsstufe

Da die Eingangsspannung abhängig von der maximalen Magnetfeldstärke einen weiten Amplitudenbereich von 40 V bei einer Extraktionsenergien von 500 MeV bis ca. 150 V bei 1,6 GeV hat, wird das Analogsignal vor der Digitalisierung über einen einstellbaren Spannungsteiler skaliert. Ein elektrischer Schaltplan der Eingangsstufe ist in Abbildung 4.7 gezeigt. Das Eingangssignal

[4] PS(0): Peakingstrip Signal. Ein TTL-Signal, das den Nulldurchgang des Magnetfeldes anzeigt.

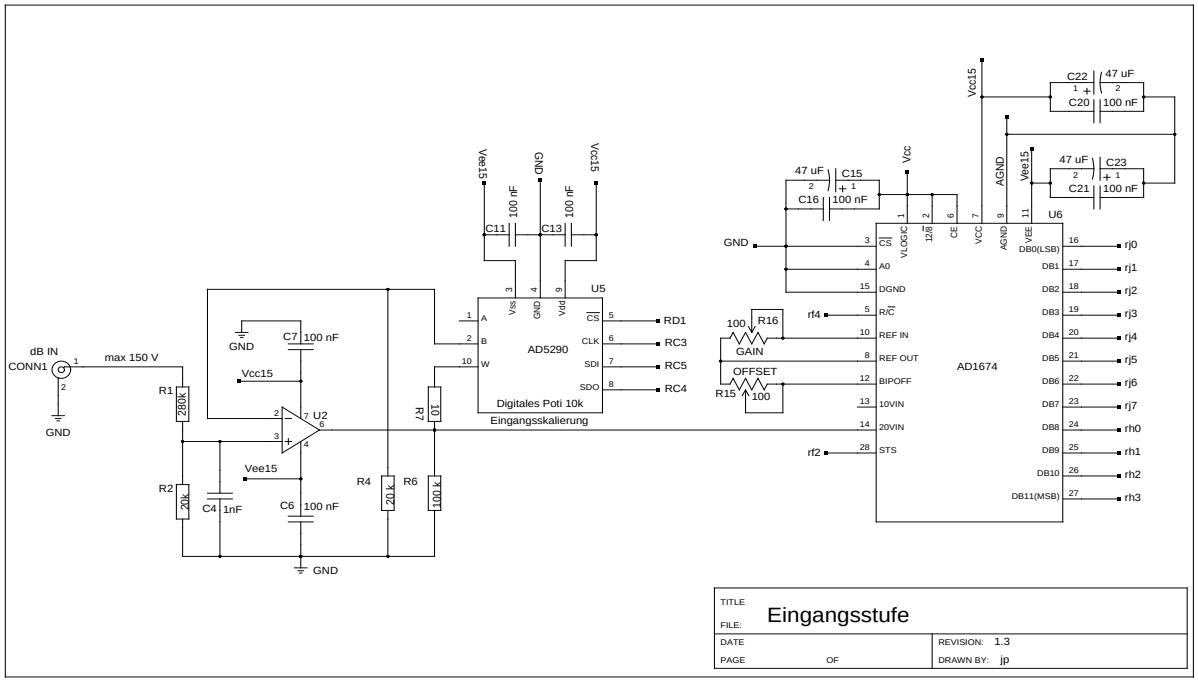


Abbildung 4.7.: Schaltplan der Eingangsstufe

wird zunächst auf den hochohmigen Eingangsspannungsteiler aus den Widerständen R_1 und R_2 gegeben, dabei wird dieses um den Faktor

$$\frac{R_2}{R_1 + R_2} = \frac{20\,\mathrm{k\Omega}}{20\,\mathrm{k\Omega} + 20\,\mathrm{k\Omega}} = \frac{1}{15}$$

geteilt. Bei einer maximal zulässigen Spannung von $150\,\mathrm{V}$ entspricht das gerade einer Spannung von $U_2 = 10\,\mathrm{V}$ am Ausgang des Spannungsteilers. Dies ist die maximal mögliche Spannung, die der ADC[5] der Eingangsstufe noch digitalisieren kann. Der sich daran anschließende, nicht invertierende Verstärker (U_2 mit dem Rückkopplungszweig: R_7, U_5 und R_4) verstärkt diese Spannung einstellbar. Die Verstärkung ν kann mit dem digitalen Potentiometer U_5 eingestellt werden (vgl. z.B. [HH89, S. 178f]) :

$$\nu = 1 + \frac{R_7 + U_5}{R_4}.$$

Das digitale Potentiometer U_5 kann in 256 Schritten zwischen $0\,\Omega$ und $10\,\mathrm{k\Omega}$ über den SPI-Bus der Platine eingestellt werden (vgl. den folgenden Abschnitt über Mikrocontroller II). Über dieses digitale Potentiometer lässt sich die Verstärkung von $\nu = 1$ bis $\nu = 1{,}5$ einstellen. Dadurch besteht die Möglichkeit, die Verstärkung der analogen Eingangsstufe digital kontrolliert einzustellen, was die Auflösung des sich daran anschließenden ADC auch für Signale kleinerer Amplitude vergrößert.

Der ADC U_6[6] digitalisiert Eingangsspannungen bis maximal $10\,\mathrm{V}$ und stellt das Ergebnis

[5] ADC ist die übliche Abkürzung für einen Analog-Digital-Konverter, ein Baustein der analoge Signale in digital vearbeitbare Datenworte umformt.

[6] AD1674: $12\,\mathrm{bit}$ parallel auslesbarer Analog-Digital-Konverter.

der Digitalisierung mit 12 bit Auflösung an den Ausgängen 16 bis 27 zur Verfügung. Diese werden direkt dem Mikrocontroller I, der die Programmberechnung durchführt, zugeführt.

4.4.2. Mikrocontroller

Die Programmierung der Mikrocontroller wird separat in Abschnitt 4.5 behandelt. Hier wird die Aufgabe der Mikrocontroller erläutert und die Verknüpfung mit den anderen Schaltungsmodulen dargestellt.

Mikrocontroller I

Die eigentliche Berechnung des HF-Programms findet auf dem Mikrocontroller I[7] statt. Der ADC der Eingangsstufe ist parallel an diesen Mikrocontroller angeschlossen. Eine neue Digitalisierung muss vom Mikrocontroller gestartet werden (Pin 5 am ADC). Der Abschluss der Digitalisierungen wird dem Mikrocontroller vom ADC dann über den Pin 28 angezeigt. 10 µs nach Start einer Digitalisierung liegt ein neuer gültiger Wert vor. Das analoge Eingangssignal kann also von dem ADC mit einer Frequenz von maximal 100 kHz abgetastet werden. Nachdem ein neuer digitaler Ausgangswert aus dem Eingangswert berechnet wurde (nach weiteren 70 µs) wird dieser auf den DAC[8] der Ausgangsstufe geben. Gesteuert wird der Mikrocontroller I von dem zweiten Mikrocontroller II[9]. Beide sind über den SPI-Bus der Platine miteinander verbunden.

Mikrocontroller II

Als *Master* des SPI-Busses gibt dieser Mikrocontroller die Taktfrequenz vor und entscheidet, mit welchem Bauteil die Kommunikation stattfindet. Mikrocontroller II muss daher regelmäßig den Ethernet-Treiber-IC[10] über den SPI-Bus abfragen und, falls Daten für die Schaltung anliegen, darauf reagieren. Außerdem programmiert dieser Mikrocontroller die beiden digitalen Potentiometer und steuert Mikrocontroller I.

SPI-Bus

Das Serial Peripheral Interface (kurz SPI) ist ein Datenbus zur seriellen Übertragung digitaler Daten. Ein Gerät muss die Bus-Steuerung übernehmen und gibt die Taktung vor; dieses Gerät wird *Master* genannt. Der Bus wird durch vier Leitungen aufgebaut:

SCK (engl. Serial Clock) der vom *Master* ausgegebene Takt.

MOSI (engl. Master Out, Slave In) Ausgangsleitung des *Master*

MISO (engl. Master In, Slave Out) Eingangsleitung des *Master*

CS (engl. Chip Select) eine Leitung pro Endgerät, über die der *Master* auswählt, mit welchem Endgerät kommuniziert werden soll.

[7] Mikrocontroller I: PIC18F8680: 8 bit Mikrocontroller mit 25 MHz.
[8] DAC: Digital-Analog-Konverter; ein Baustein, der digitale Datenworte in analoge Spannungswerte konvertiert.
[9] Mikrocontroller II: PIC18F6680: 8 bit-Mikrocontroller mit 25 MHz.
[10] Ethernet: Ein Standard, der eine Technologie für lokale Netzwerke beschreibt. Sowohl die Software (Beschreibung des Protokolls und der Kodierung) als auch die Hardware (Treiber, Kabel usw.) sind spezifiziert.

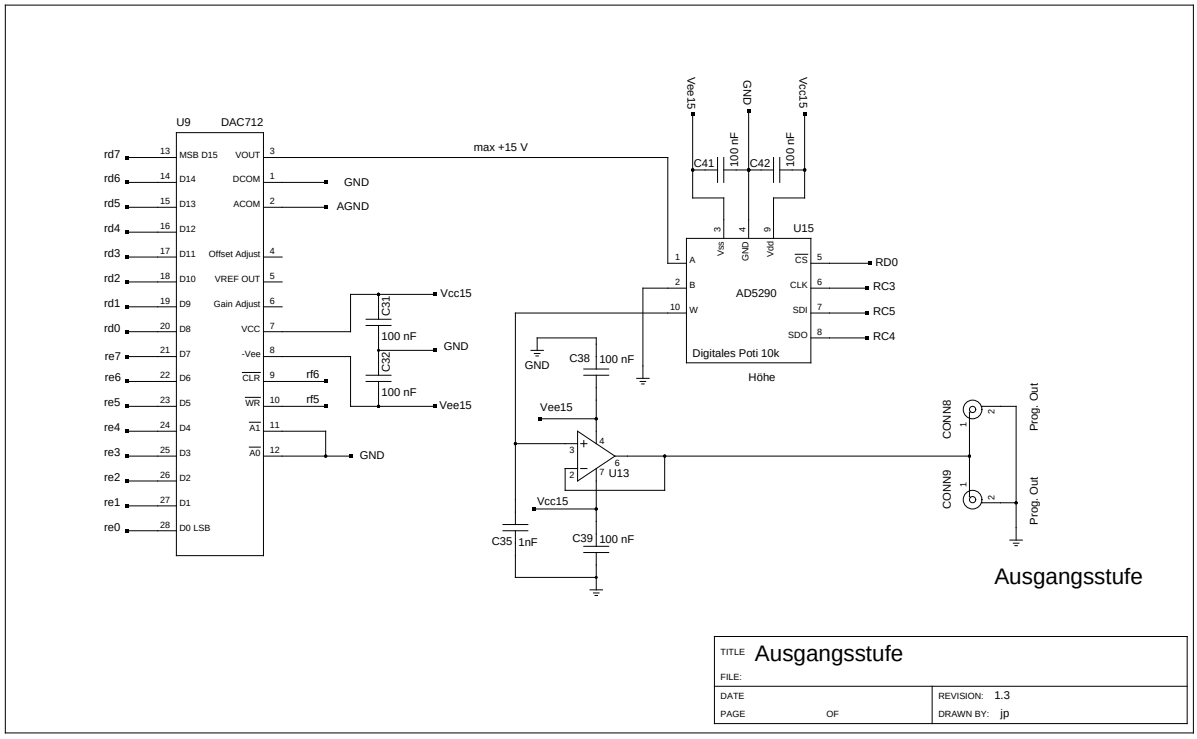


Abbildung 4.8.: Schaltplan der Ausgangsstufe

Alle Endgeräte teilen sich die drei Bus-Leitungen (SCK, MOSI und MISO). Als Taktfrequenz sind $1{,}6\,\mathrm{MHz}$ eingestellt worden. Mikrocontroller II fungiert als *Master*-Gerät, insgesamt sind vier SPI-Endgeräte angeschlossen:

- Mikrocontroller I
- Ethernet-Treiber (vgl. Abschnitt 4.4.4)
- Digitales Potentiometer der Eingangsstufe
- Digitales Potentiometer der Ausgangsstufe

4.4.3. Ausgangsstufe

Das berechnete HF-Programm wird als digitale Größe vom Mikrocontroller I zur Ausgangsstufe geleitet. Abbildung 4.8 zeigt den elektrischen Schaltplan der Ausgangsstufe: Der digitale Wert des HF-Programms wird mit $16\,\mathrm{bit}$ Auflösung parallel in den DAC geleitet (Pins D_0 bis D_{15}) und dort in eine Ausgangsspanung (Pin 3 am DAC) im Bereich von $\pm 15\,\mathrm{V}$ konvertiert. Die Ausgangsspanung wird an einem digitalen Potentiometer (U_{15}) aufgeteilt, wobei 256 Schritte die Ausgangsspannung auf $0\,\%$ bis $100\,\%$ abbilden. Dadurch wird ermöglicht das berechnete HF-Programm herunter zu skalieren, um die Leistung des Klystrons zu steuern und dabei die Proportionalität des HF-Programms beizubehalten. Aus historischen Gründen wird dieses Potentiometer *Höhe* genannt.

Zur Impedanzanpassung wird das Ausgangssignal auf einen Verstärker (U_{13}) mit Verstärkung $\nu = 1$ gegeben.

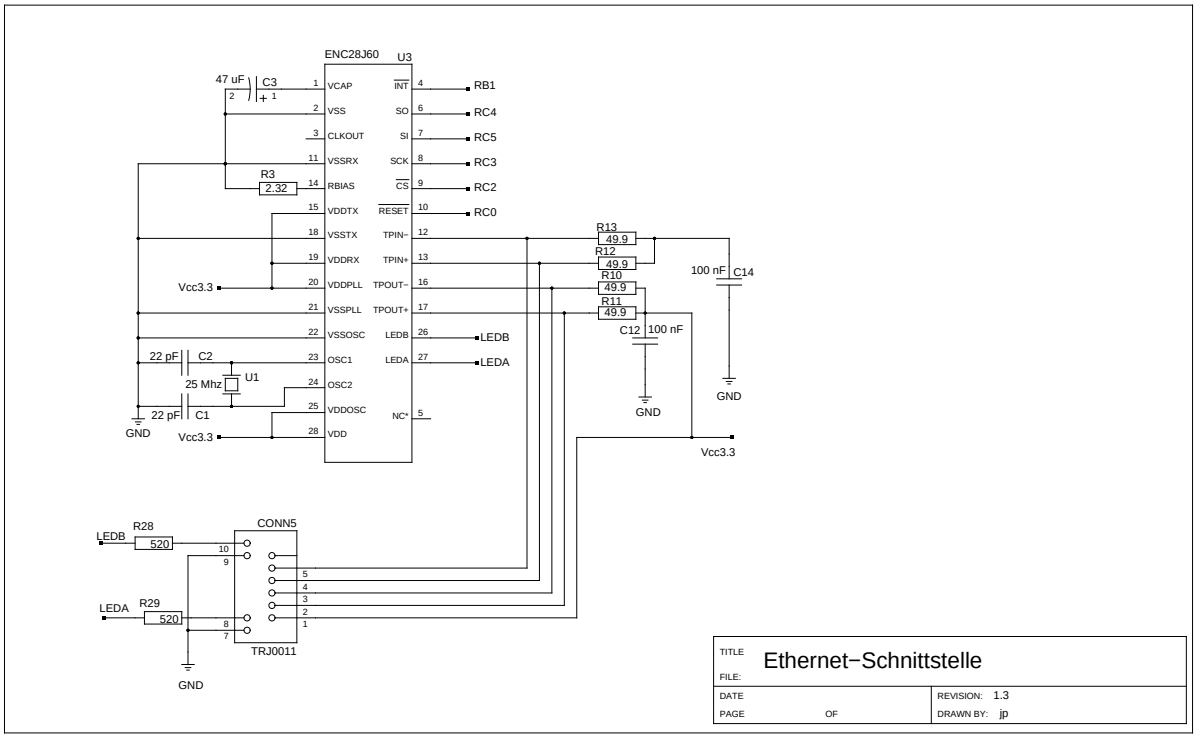


Abbildung 4.9.: Schaltplan der Ethernet-Schnittstelle.

4.4.4. Ethernet-Schnittstelle

Die Anbindung an das Kontrollsystem wird über Ethernet realisiert. Der Schaltplan dieses Teils ist in Abbildung 4.9 gezeigt. Dieser besteht im Wesentlichen aus dem Treiber-IC, welches die Hardware-Spezifikation des Ethernet-Standard herstellt. Der hier verwendete IC (ENC28J60) wird über den SPI-Bus mit Mikrocontroller II verbunden, welcher die gesamte Softwarerealisierung des Ethernet-Stacks[11] übernehmen muss. Der Treiber IC übernimmt nur die direkte Hardwareebene.

4.5. Software

Gegenstand dieses Kapitels sind die für die beiden Mikrocontroller entwickelten Programme. Dabei wird hier nicht weiter auf die Architektur und Programmierung von Mikrocontrollern eingegangen.

Prinzipiell werden Mikrocontroller in einer spezifischen Maschinensprache programmiert, die für jeden Mikrocontroller einen speziellen Satz von Befehlen beinhaltet. In diesem Fall jedoch werden die Programme in der Programmiersprache C geschrieben. Ein spezieller Compiler[12] übersetzt dann die in C geschriebenen Programme in den Maschinencode der verwen-

[11] Stack: (englisch) Stapel. Hier wird speziell mit Ethernet-Stack der Teil der Software beschrieben, der die standardkonforme Abarbeitung aller Anfragen, die per Ethernet an die Schaltung übermittelt werden, vornimmt.

[12] Ein Compiler ist ein Computerprogramm, das ein in einer Hochsprache geschriebenes Programm, in eine andere, üblicherweise maschinennähere Sprache übersetzt. Hier wird ein unfreier Compiler der Firma Microchip eingesetzt: der C18-Compiler.

deten Mikrocontroller. Dabei kann im wesentlichen ANSI-C[13] verwendet werden. Trotzdem muss beim Programmieren beachtet werden, dass für Mikrocontroller-Programmierungenen einige Einschränkungen gelten: Der Speicherplatz ist begrenzt, die Anbindung an den Peripherie muss spezifisch auf den jeweiligen Mikrocontroller angepasst werden und die Datenbreite beträgt nur 8 bit. Vor allem wenn die Ausführungsgeschwindigkeit relevant ist, empfiehlt es sich, die Ausgabe des Compilers zu untersuchen und das C-Programm hinsichtlich der Geschwindigkeit gezielt zu optimieren (vgl. hierzu besonders den folgenden Abschnitt).

Listings[14] der Programme in ihrer aktuellen Version sind als Referenz im Anhang A angeben. Die Programme sind im Quelltext umfangreich kommentiert worden, im Folgenden werden daher nur zentrale Bausteine der Programme detaillierter erläutert.

4.5.1. Programm zur Berechnung des HF-Programms: Mikrocontroller I

In Listing (A.1) ist das C-Programm des Mikrocontroller I angeben, dessen Aufgabe darin besteht, das HF-Programm zu berechnen. Im Wesentlichen arbeitet das Programm eine Endlosschleife ab, die den Zyklus Digitalisieren → Berechnen → Analogwertausgabe ständig durchläuft. Die Abtastrate wird über einen Software-Timer fest vorgegeben.

Hauptschleife

Der Code der Hauptschleife ist in Listing (A.1) in den Zeilen 344 bis 386 zu finden. Auf wichtige Blöcke in dieser Schleife soll nun genauer eingegangen werden:

Im Kopf der Schleife wird der Wert von `Timer0` überprüft. Nur wenn dieser größer als 12 ist steigt das Programm in die Berechnung ein.

```
344
345   while(1)  // ADC->DAC loop: Main loop
346     {
347           if(TMR0L >=(unsigned) 13)  // ca 80 µs
348          {
349            TMR0H=0;  // Set Timer0 to zero
350            TMR0L=0;
```

Die Zahl 13 wurde empirisch gefunden; dazu wird das Programm so geändert, dass bei jedem Eintritt in die Schleife ein Ausgangspin des Mikrocontrollers seinen Status auf logisch *Eins* setzt, am Ende der Schleife wird dieser Pin wieder auf *Null* gesetzt. Von außen kann dann mit einem Oszilloskop dieser Pin beobachtet werden. Die Rate, mit der die Schleife ausgeführt wird, wird als optimal eingestellt angesehen, wenn dieser Pin für etwa 1 % der Zeit eines jeden Zyklus *Null* ist, also bleiben maximal 1 % der gesamten Zeit als Reserve übrig. So ist vor allem sichergestellt, dass die Zeit zwischen zwei Integrationsschritten konstant ist.

Als nächster Schritt wird ein neuer analoger Wert digitalisiert. Dazu werden zwei Makros benutzt: `Convert_my_adc()` legt die zum Starten einer neuen Konvertierung benötigte Flanke an PIN 5 des ADC, `Busy_my_adc()` ist mit PIN 28 des ADC belegt, welcher das Ende der Digitalisierung signalisiert. Die niedrigen 8 bit-Ausgangsleitungen des ADC sind am Mikrocontroller mit `PORTJ` verbunden, die 4 hochwertigen mit `PORTH`:

[13] Ein Standard, der die Sprache C definiert.

[14] Listing, englisch: Programmausdruck. Hier ist damit die Wiedergabe des Programmcodes gemeint. Hierbei werden bestimmte Syntaxelemente speziell hervorgehoben.

```
353           Convert_my_adc();                 // start conversion on extern ADC
354           while(Busy_my_adc());             // waiting for ADC
355           db= PORTJ + PORTH*256 -0x7ff ;    // read extern ADC ; 0x7ff is 0V at ADC.
```

Als Nächstes wird das Eingangssignal aufgearbeitet. Die in Abschnitt 2.1.3 erläuterte Spule liefert ein invertiertes Signal, also eine zu $-\dot{B}$ proportionale Spannung[15]. Daher wird das Vorzeichen der Variable `db` zunächst invertiert. In den darauffolgenden Zeilen wird die Eingangsgröße kleiner skaliert. Dies ist notwendig, da auf dem Mikrocontroller in sinnvoller Geschwindigkeit nur Rechenoperationen mit maximal 16 bit-Variablen durchgeführt werden können. Alle verwendeten Variablen sind daher maximal als 16 bit-Werte definiert [16].

Die Auflösung des ADC beträgt 12 bit, die digitalisierte Eingangsgröße wird daher zunächst um 4 bit nach rechts verschoben, so dass diese mit maximal 8 bit dargestellt werden kann. Sich daran anschließende Rechenoperationen mit weiteren 8 bit Größen erzeugen dann sicher keinen 16 bit-Überlauf. Es muss beachtet werden, dass beim Verschieben von Speicherinhalten die interne Darstellung vorzeichenbehafteter Größen vom Compiler nicht beachtet wird. Vorzeichenbehaftete Größen werden intern in der Zweierkomplement-Darstellung gespeichert. Dabei belegt das höchstwertige Bit die Information des Vorzeichens (vgl. z.B. [Tie02]). Daher muss beim Verschieben von negativen Zahlen eine Korrektur vorgenommen werden (Zeile 363). Dazu wird in der Variable `tmp` das Vorzeichen vor der Verschiebung gesichert:

```
357           db*=-1;                       // Extern dbIn has wrong sign, so toggle it
358
359           tmp= db < 0;   // is db negative ?
360           db  >>= 4;     // division by 2^4;
361
362           // if db was negative then correct Two's complement
363           if (tmp) db |= 0b1111000000000000;
```

Um das Integral zu berechnen, müssen die aktuellen Magnetfeldstärken addiert und mit der Zeit zwischen zwei Integrationen multipliziert werden. Da mit einer festen Rate digitalisiert wird, ist die Zeit zwischen jeder Addition identisch und kann aus der Addition heraus faktorisiert werden. Um das Integral zu berechnen, werden die digitalisierten Werte summiert, der gemeinsame Faktor wird aus Rechenzeitgründen weggelassen. Er muss bei der Festlegung des Gewichtungsfaktors für den B^4-Anteil berücksichtigt werden.

```
366           integral += (db) ;       // integration
```

Dabei bleibt die Integrationskonstante allerdings unbestimmt. Ein von dem Peaking-Strip (vgl. Abschnitt 2.1.3) erzeugter Puls signalisiert dem Mikrocontroller per Interrupt[17] den Nulldurchgang des Magnetfeldes. Zu diesem Zeitpunkt wird daher der Wert des Integrals auf Null gesetzt, womit das Integral inklusive Integrationskonstante vorliegt:

```
373           if (PROG_STATUSbits.ps) // new integral
374             {
375               integral=0;
376               PROG_STATUSbits.ps=0;
377             }
```

[15] Es ist unklar wieso das Vorzeichen falsch erzeugt wird, vermutlich müssen nur die Anschlüsse direkt an der Pick-Up Spule getauscht werden. Ob die Belegung absichtlich so gewählt wurde, ist unbekannt.

[16] Hilfsvariablen sind, wo immer möglich, nur als 8 bit-Variablen definiert.

[17] (engl. to interrupt, unterbrechen) Unterbrechung eines Programms, um eine andere Funktion abzuarbeiten. Bei Mikrocontrollern wird ein Interrupt meistens durch äußere Ereignisse ausgelöst.

Die Variable `PROG_STATUSbits.ps` wird in einer Interruptfunktion auf logisch Eins gesetzt, falls ein Signal des Peaking-Strips anlag.

Am Ende der Schleife wird das HF-Programm berechnet. Dazu wird zunächst B^2 berechnet, das Ergebnis wird um $8\,\mathrm{bit}$ verschoben um bei der anschließenden Berechnung von B^4 keinen $16\,\mathrm{bit}$-Überlauf zu erzeugen. Die Größe B^4 wird ebenfalls um $8\,\mathrm{bit}$ verschoben, um bei der Berechnung des HF-Programms keinen Überlauf zu erhalten:

```
379             b2=(b*b)    >> 8;
380             b4=(b2*b2)  >> 8;
381             prog=actuals.cdb*db+actuals.cb4*b4+actuals.c0;
382
383             LATE = prog ;                 // DAC low bits
384             LATD = (char)(prog >> 8);     // DAC high bits
385             }
386     }
```

Hierbei ist die Variable `b` die noch einmal um $6\,\mathrm{bit}$ verschobene Größe `integral`. Die Variablen `actuals.cdb`, `actuals.cb4` und `actuals.c0` werden über den SPI-Bus durch Mikrocontroller II an dieses Programm übermittelt und sind die einstellbaren Vorfaktoren des HF-Programms (vgl. Abschnitt 4.1). Der DAC ist mit `PORTE` und `PORTD` verbunden, wobei auf letzteren die hochwertigen Bit gelegt werden müssen.

Einschalten des DAC und Vorgabe der Dauer

Zunächst bleibt der DAC noch ausgeschaltet, da das HF-Programm erst nach Abschluss der Injektion eingeschaltet werden darf. Dazu wird von außen eine Signalflanke ($\mathbf{HF_{on}}$) an PIN 58 des Mikrocontroller I gelegt. Diese Flanke löst einen speziellen Interrupt aus, der zum einen den DAC einschaltet und zum anderen einer Timer startet. Wenn dieser Timer abgelaufen ist, löst dieser einen weiteren Interrupt aus, durch den der DAC wieder ausgeschaltet wird. Die Laufzeit dieses Timers kann vom Kontrollsystem vorgegeben werden und legt die Dauer des HF-Programms fest. Die Interrupts sind in Listing A.1 in den Zeilen 67 bis 125 definiert.

Funktion: `void SpiTask()`

Die Steuerung und Kommunikation wird von Mikrocontroller II, dem SPI-*Master*, kontrolliert: Dazu sind beide Mikrocontroller jeweils mit einem SPI-Modul[18] ausgestattet. Wird dieses Modul aktiviert, werden vier bestimmte Pins des Mikrocontrollers als SPI-Bus Leitungen (vgl. Abschnitt 4.4.2) konfiguriert. Durch das SPI-Modul kann die Implementierung der Kommunikation via SPI-Bus komfortabel realisiert werden:

Wird an Mikrocontroller I über die Datenleitung ein Datenbyte geschickt, wird durch sein SPI-Modul der Interrupt (`PIR1bits.SSPIF`) ausgelöst. Dieser ruft die Funktion `SpiTask()` auf, welche das Datenbyte einlesen muss. Die Funktion wertet anschließend das gelesene Byte aus und legt das weitere Verhalten fest; zum Beispiel müssen bei einigen Kommandos weitere Bytes gelesen werden, oder aber der DAC wird ein- oder ausgeschaltet. Die Funktion ist im Listing A.1 in den Zeilen 134 bis 216 dokumentiert, eine Übersicht über alle definierten Kommandos kann auch in der Definitions-Datei *spi.h* (vgl. Listing A.3) gefunden werden.

[18] Unter einem Modul ist hier eine direkt auf dem Chip implementierte Hardware-Komponente zu verstehen. Das SPI-Modul übernimmt dabei autark die Steuerung des SPI-Interfaces.

Funktion: `passive_read()` und `passive_send()`

Das Lesen und Schreiben auf dem SPI-Bus muss auf Mikrocontroller I passiv realisiert werden, da der SPI-*Master* den Bus-Takt vorgibt. Die Funktion `passive_read()` liest das nächste Byte von der Datenleitung des SPI-Bus:

```
242 unsigned char passive_read(void)   // read from SPI. Master has SPI CLock
243 {
244   unsigned char read;
245   while(!PIR1bits.SSPIF);
246   read=SSPBUF;
247   PIR1bits.SSPIF=0;
248   return ( read );
249 }
```

Die Funktion wartet, bis das SPI-Modul des Mikrocontrollers das Interrupt-Bit für ein SPI-Ereignis gesetzt hat, was anzeigt, dass 8 bit eingelesen wurden. Danach liegt in `SSPBUF` das neu eingelesene Byte, dieses gibt die Funktion schließlich an den Aufrufer zurück.

Die Funktion wird als passiv implementiert bezeichnet, da ein Aufruf der Funktion so lange wartet, bis 8 bit gelesen wurden. Die Funktion ist also darauf angewiesen, dass aktiv Daten gesendet werden. Sollte der SPI-*Master* diese nicht senden, oder den Takt-Zyklus nicht bedienen, wartet diese Funktion allerdings endlos[19]. Die Funktion `passive_send()` ist analog aufgebaut (vgl. die Zeilen 251 bis 257 in Listing A.1)

4.5.2. Der Microchip-TCP/IP-Stack

Das Programm von Mikrocontroller II steuert zum einen alle SPI-Endgeräte, zum anderen ist es für die Implementierung des Ethernet-Stacks verantwortlich. Für letzteres wird der von der Firma Microchip entwickelte frei zur Verfügung stehende TCP/IP-Stack[20] eingesetzt. Bei Rechnernetzen wird die Kommunikation in einer großen Zahl von Protokollen festgelegt. Jedes Gerät, das an einem solchen Netz teilnehmen möchte, muss diese Protokolle implementieren und dabei dem Standard folgen. Da in diesem Fall ein schon vollständig implementierter Stack eingesetzt werden kann, muss die Implementierung nicht selbst vorgenommen werden.

Der TCP/IP-Stack von Microchip liegt in Form etlicher C-Programme und Definitionsdateien vor. Dadurch ist es möglich, nur die Teile des Stacks, die für das vorliegende Projekt notwendig sind, zu integrieren:

- IP.c
- UDP.c
- ENC28J60.c
- ARP.c

[19] Um zu verhindern, dass in einer solchen Situation die Ausführung des Programms nicht fortgesetzt wird, kann ein sogenannter Watchdog aktiviert werden. Dabei handelt es sich um einen intern im Mikrocontroller, unabhängig von der Programmausführung laufenden Timer, der nach einer einstellbaren Zeit abläuft. Der Timer kann im Programmcode regelmäßig zurückgesetzt werden. Bleibt das Rücksetzen aus, so läuft der Timer ab und löst einen Reset aus.

[20] Transmission Control Protocol/Internet Protocol (TCP/IP) ist eine Familie von Netzwerkprotokollen und wird wegen ihrer großen Bedeutung für das Internet auch als Internetprotokollfamilie bezeichnet (aus [Wik11a]). In dem vorliegenden Fall wird die TCP-Schicht des Protokolls nicht benötigt.

- DHCP.c
- ICMP.c
- Helpers.c
- Tick.c
- StackTsk.c

In IP.c wird das IP implementiert, das die Verbindung von Mikrocontroller II über das Netzwerk zum Kontrollsystem der Beschleunigeranlage realisiert. UDP.c implementiert das UDP[21] das in diesem Projekt verwendete Datenaustausch-Protokoll. Der Code in ENC28J60.c stellt die Schnittstelle zum verwendeten Ethernet-Treiber dar und realisiert dadurch die Hardwareanbindung an das Netzwerk. Die Dateien ARP.c, DHCP.c und ICMP.c definieren drei weitere Protokolle, die bei der Verwendung von IP implementiert sein müssen. Helpers.c und Tick.c stellen Funktionen zur Verfügung, die generell benötigt werden. StackTask.c schließlich muss von jedem Projekt, das den Stack benutzt, eingebunden werden. Hier wird die Funktion `void StackTask(void)` definiert, diese muss vom Hauptprogramm des Mikrocontrollers regelmäßig aufgerufen werden, wodurch alle anstehende Aufgaben des Stacks abgearbeitet werden.

4.5.3. Mikrocontroller II

Auch Mikrocontroller II führt eine Endlosschleife aus, die in Listing A.2 in den Zeilen 417 bis 447 definiert wird. Zu Anfang wird die oben erwähnte Stack-Funktion ausgeführt:

```
423     StackTask();
```

Direkt danach wird überprüft, ob ein UDP-Datenpaket empfangen wurde, und falls dies der Fall ist, wird das Datenpaket eingelesen und in `ProcessMyUDP(udp_command)` verarbeitet:

```
425     if(UDPIsGetReady(my_udp_socket))
426       {
427         DEBUG_GREEN; // got valid UDP Packet (from ELSA-controll system)
428         if(UDPGetArray((BYTE*) &udp_command, sizeof(COMMAND_PACKAGE)) )
429           {
430             ProcessMyUDP(udp_command);
431           }
432         else // something went terribly wrong
433           {
434             RESET_PROG;   // Reset Prog
435             Reset();      // Self-Reset
436           }
437       }
```

Die Funktion `UDPGetArray` liest Bytes, die mittels UDP übertragen wurden, in `udp_command` ein. Dieses Objekt ist vom Typ `COMMAND_PACKAGE` und definiert die Zuordnung der gelesen Bytes in ein Kommando-Byte und zwei Daten-Bytes:

[21] Das User Datagram Protocol, kurz UDP, ist ein minimales, verbindungsloses Netzwerkprotokoll, das zur Transportschicht der Internetprotokollfamilie gehört.(aus [Wik11b])

```
typedef struct COMMAND_PACKAGE_T
{
  unsigned char cmd; // 8 byte command
  unsigned char dummy;
  int data; // 16 bit data
} COMMAND_PACKAGE;
```

Die Funktion `ProcessMyUDP` verarbeitet das eingelesene Kommando (vgl. Zeilen 108 bis 148 in Listing A.2). Wenn das gesendete Kommando eines der SPI-Potentiometer in der Eingangs- oder Ausgangsstufe (vgl. Abschnitt 4.4.1 oder 4.4.3) programmieren soll, wird das erste Datenbyte an das entsprechende SPI-Potentiometer gesendet. In allen anderen Fällen wird das Kommando und die Datenbytes zur anschließenden Übermittlung gesichert:

```
110   myspi.command=udp_command.cmd; // got command from UDP
111   myspi.SPInew=1;       // we got a command  from UDP so a SPI action is needed
```

`myspi` ist vom Type `MYSPI`. Diese Struktur wird zum Versenden von SPI-Kommandos benutzt und ist wie folgt definiert:

```
typedef struct MYSPI {
  unsigned SPInew:1;
  unsigned command:7;
} MYSPI;
```

Im weiteren Ablauf der Hauptschleife wird die Funktion `SpiTask()` aufgerufen, die falls `myspi.SPInew` gesetzt ist, die SPI Kommunikation vornimmt (vgl. dazu die Zeilen 249 bis 375 in Listing A.2).

4.6. Anbindung an das Kontrollsystem

Die Eingabemaske zur Steuerung des Programmgenerators ist in Abbildung 4.10 gezeigt. Zur Anbindung an die Benutzeroberfläche des ELSA-Kontrollsystems (vgl. [T95; Pic95]) wird ein Programm benötigt, welches die angezeigten Parameter mit der Hardware verknüpft. Hier konnte auf schon bestehende Software zurückgegriffen werden[22], die allerdings an die spezifischen Anforderungen angepasst wurde. Dieses Programm verbindet sich mit dem Kontrollsystem und meldet dort Parameter an. Zusätzlich wird in der Struktur `parameter_cb[]` eine Zuordnung von Parameternamen zu den zugehörigen SPI-Kommandos (vgl. Listing A.3 *spi.h*) vorgenommen.

Listing 4.1: Zuordnung der Parameternamen zu ihren SPI-Kommandos.

```
PARAMETER_CB parameter_cb[] = {
  {"SYN_HF_PROG.ENABLE_DC", TEIN},
  {"SYN_HF_PROG.SIGN_DC", T_SIGN},
  {"SYN_HF_PROG.RESET_DC", S_RESET },
  {"SYN_HF_PROG.NORM_DC", S_NORM},
  {"SYN_HF_PROG.SIMUL_DC", SIMUL },
  //   {"SYN_HF_PROG.COMMAND_SC", },
  {"SYN_HF_PROG.SDB_AC", S_CDB},
  {"SYN_HF_PROG.SCB4_AC",S_CB4 },
```

[22] Momentan hat dieses Programm den Namen conn.c und wird auf *elsapc84* ausgeführt. Ein komplettes Listing dieses Programms ist nicht angefügt, hier werden nur kurze Ausschnitte gezeigt.

```
  {"SYN_HF_PROG.SC0_AC", S_C0},
  {"SYN_HF_PROG.SDAUER_AC",S_DAUER },
  {"SYN_HF_PROG.SHOEHE_AC",S_HOEHE },
  {"SYN_HF_PROG.SCALE_AC", S_IN_SCALER},
        {NULL}
};
```

Wird einer dieser Parameter zum Beispiel durch eine Benutzereingabe geändert, so wird die Funktion `standard_callback` ausgeführt. Der info-Wert des Parameters, also das zugeordnete SPI-Kommando, und der neue Wert des Parameters werden beim Aufruf der Funktion übergeben.

Listing 4.2: Definitionen der Funktion: `standard_callback`

```
int standard_callback(int pid, void *info, int ndata, int *data)
{
        unsigned char buffer[4];
        unsigned short x;

        buffer[0] = *(int *)info;
        buffer[1] = 0;

        x = (short) (*data);

        memcpy(buffer+2, &x, 2);
        send_data(buffer, 4);
        return 0;
}
```

Das erste Byte der Variable `buffer[]` wird auf den zugeordneten Werder des SPI-Kommandos gesetzt, das zweite auf Null. In die letzten zwei Bytes wird der neue Wert des Parameters gespeichert. Durch den Aufruf der Funktion `send_data` werden dann schließlich diese vier Bytes per UDP an die Programmgenerator-Schaltung gesendet.

4.6.1. Ein Beispiel: Einschalten des Programmgenerators

Die Ansteuerung des Programmgenerators durch das ELSA-Kontrollsystem soll anhand der Abläufe eines beispielhaften Vorgangs erläutert werden. Wird in der Eingabemaske der Button zum Einschalten betätigt, so werden durch die Funktion `send_data`, wie oben beschrieben, folgende vier Byte versendet: 7 | 0 | 1 | 0 . Das erste Byte ist dabei auf den info-Wert des Parameters des Einschalten-Button gesetzt worden, der in der zentralen Definitionsdatei *spi.h* als 7 festgelegt wird:

```
8 #define TEIN      7       // Toggle EIN/AUS
```

Mikrocontroller II

Bei der Abarbeitung der Endlosschleife in Mikrocontroller II werden vier Bytes des anliegenden UDP-Datenpakets in die Variable `udp_command` eingelesen und an die Funktion `ProcessMyUDP` weitergegeben (vgl. Zeilen 108 bis 148 in Listing A.2). Diese stellt fest, dass ein SPI Kommando (1. Byte 7) an Mikrocontroller I gesendet werden muss und sichert dieses dazu in der Variable `myspi.command`. Anschließend wird das Kommando ausgewertet. Da es sich um das

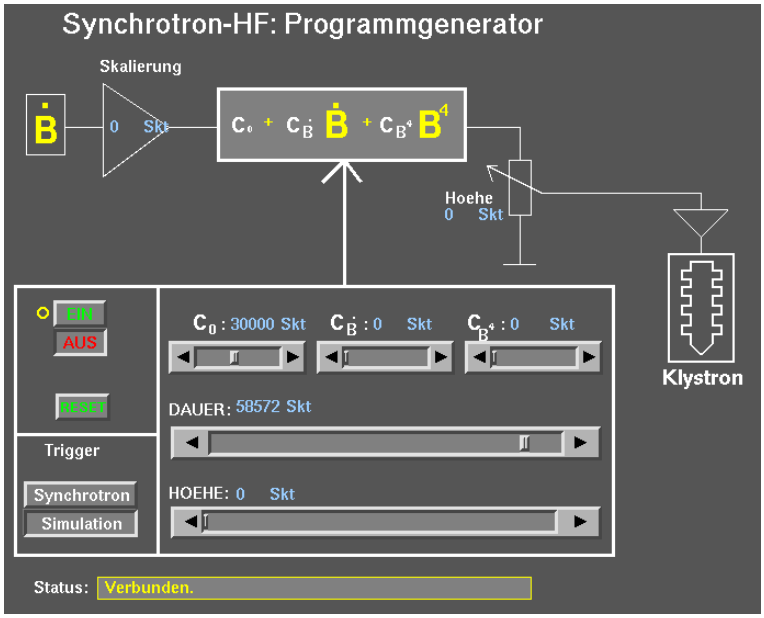


Abbildung 4.10.: Kontrollsystemmenu zur Steuerung des Programmgenerators

Kommando zum Ein- oder Ausschalten handelt, wird das erste Datenbyte des UDP-Pakets: 1 zusätzlich in der Variable `set_ein` gespeichert (0 = Aus; 1 = EIN). Als nächstes wird die Funktion `SpiTask()` aufgerufen. Diese wertet das SPI-Kommando aus.

```
281       switch (myspi.command)
282         {
283         case TEIN:
284           spi_send(TEIN);
285           spi_send(set_ein);
286           myspi.command=G_ACTUALS;
287           myspi.SPInew=1;
288           break;
```

Für das Ein-/Ausschalten-Kommando wird dann zunächst das Kommando TEIN mittels `spi_send` an Mikrocontroller I gesendet. Direkt danach wird `set_ein` versendet. Danach ist die Bearbeitung durch Mikrocontroller II abgeschlossen.

Mikrocontroller I

Sobald das erste Byte mit dem SPI-Kommando versendet wird, wird bei Mikrocontroller I ein SPI-Interrupt ausgelöst. Dieser ruft direkt die Funktion `SpiTask`[23] auf, welche das eingelesene SPI-Byte auswertet.

```
150       switch(myspi.command)
151         {
152         case TEIN:                          // Toggle EIN/AUS
153           if(passive_read())
154             {
```

[23] Die Funktion `SpiTask` wird auf beiden Mikrocontrollern implementiert, und darf hier nicht mit der Funktion von Mikrocontroller II verwechselt werden.

```
155               PROG_STATUSbits.ein=1;
156             }
157           else
158             {
159               PROG_STATUSbits.ein=0;
160               LATFbits.LATF6=0;          // Disable DAC
161               TRISHbits.TRISH5=1;        // Extern LED: Prog OK is now Input
162             }                            // => LED stays off
163           break;
```

Bei Anliegen des TEIN-Kommandos wird direkt ein weiteres Byte vom SPI-Bus gelesen (vgl. Zeile 155). Dieses wird ausgewertet, falls dabei eine 1 gelesen wurde wird das globale Bit `PROG_STATUSbits.ein` gesetzt. Sollte der Wert 0 gesendet worden sein wird dieses Statusbit auf 0 gesetzt und anschließend der DAC ausgeschaltet.

Der DAC darf hier nicht direkt eingeschaltet werden. Dies geschieht ausschließlich in der Interruptfunktion, die auf das HF-Einschalten-Signal reagiert (vgl. Zeilen 90 bis 100 in Listing A.1) . Nur wenn das Status-Bit gesetzt ist, wird dort der DAC eingeschaltet, andernfalls bleibt er deaktiviert.

Ergebnisse

Das Ziel der vorliegenden Arbeit war es, die Ansteuerung des Hochfrequenzsystems des Booster-Synchrotrons für den zukünftigen Betrieb mit hohen Strahlströmen zu verbessern. Dazu wurde eine aktive Regelung der Resonanzfrequenz des Hohlraumresonators in Betrieb genommen und die Hochfrequenzamplitude im Resonator stabilisiert.

Für die Regelung der Resonanzfrequenz wurde die vorhandene Steuerung der Abstimmstempel um die für die Regelung notwendigen Komponenten erweitert. Die dazu benötigten Hochfrequenz-Signale wurden identifiziert und der Regelung zugeführt. Die durchgeführten Messungen belegen die Funktion der Regelung; die dabei entdeckte langsame zeitliche Oszillation der Resonatortemperatur kann durch die Stempelregelung erfolgreich kompensiert werden. Erstmals steht damit eine Regelung zur Verfügung, die während des Beschleunigerbetriebs den Resonator automatisch auf die anregende Frequenz abgestimmt hält und so konstante Betriebsbedingungen garantiert. Die Regelung kann von nun an im Standardbetrieb eingesetzt werden.

Die Regelung der Hochfrequenzamplitude wurde völlig neu aufgebaut. Dazu wurde zunächst der theoretisch benötigte zeitliche Verlauf der Hochfrequenzamplitude berechnet und eine neue Schaltung auf Basis von Mikrocontrollern entwickelt. Diese erzeugt das für die Synchronisation der Felder notwendige HF-Programm. Ein installierter analoger Proportional-Integral-Regler stabilisiert die Amplitude. Für die Anbindung an das Kontrollsystem der Beschleunigeranlage wurde eine Ethernet-Schnittstelle aufgebaut; im Betrieb zeichnet sich diese durch schnelle und robuste Kommunikation aus. Die letzte analoge Einheit im Kontrollraum der Beschleunigeranlage konnte damit erfolgreich durch eine digital ansteuerbare Schaltung ersetzt werden. Damit steht nun eine einstellbare Ansteuerung der Hochfrequenzamplitude zur Verfügung.

Die empirische Einstellung optimaler Vorfaktoren für das HF-Programm steht noch aus. Dazu wird idealerweise eine Messung der Synchrotronschwingung benötigt. Allerdings muss die dazu notwendige Messapparatur zunächst noch konzipiert und aufgebaut werden. Zusätzlich wäre eine Kalibrierung der Leistungspegel der Richtkoppler wünschenswert. Zumindest die Ausgangsleistung des Klystronverstärkers sollte gemessen und kalibriert im Kontrollsystem angezeigt werden.

Anhang

Anhang A.

Programm-Listings

A.1. HF-Programm Berechnung

Listing A.1: Programm des PIC18F8680: Programmberechnung

```
/* PIC 18 Program for: Synchrotron-HF-Programmgenerator
 * Last modified 2011-05-24 by J-P Thiry
 * Main Loop:
 *  read external ADC into dB, and calculate appropriate RF-program: c + c_db * dB + c_b4 * B^
 *  set extern DAC with actual program value
 * Extern Triggers:
 *  PS(0) : TTL: B=0: Start new Integration
 *  PS(HF): TTL: Extern Signal: Start RF
 * Timers:
 *  Timer0: sampling rate of main loop
 *  Timer1: Time of active RF-program
 *  Timer2: Helper for Simulation mode: ca. 50 Hz
 * this PIC is controlled by annother PIC: communication via SPI
 * this PIC is a SPI-slave
 * refer to spi.h for defintions of SPI commands
 * second PIC on Board is SPI-Master
 */


#include <p18f8680.h>
#include <timers.h>
#include <delays.h>
#include <math.h>
#include "../test-eth/spi.h"
#pragma config OSC = HS
#pragma config OSCS = OFF
#pragma config PWRT = OFF   // Power on timer
#pragma config BOR = ON
#pragma config BORV = 20
#pragma config WDT = OFF
#pragma config WDTPS = 64
#pragma config MODE =MC
#pragma config WAIT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF

#define Busy_my_adc() PORTFbits.RF2
#define Convert_my_adc()  LATFbits.LATF4=1;   LATFbits.LATF4=0;   Nop();   Nop();
LATFbits.LATF4=1;   Nop();   Nop();
#define Read_my_adc() PORTJ + PORTH*256

```

```
41 #define DEBUG_YELLOW() LATGbits.LATG3=~PORTGbits.RG3;
42 #define DEBUG_RED() LATGbits.LATG0=~PORTGbits.RG0;
43 #define DEBUG_ORANGE() LATGbits.LATG4=~PORTGbits.RG4;
44
45
46 void high_isr (void);
47 int do_ad(void);
48 void SpiTask(void);
49 void passive_send(char comma);
50 unsigned char passive_read(void);
51 int getDBinMax(void);
52
53 volatile int dbinmax=0;
54
55 SPI_ACTUALS actuals;
56 SPI_ACTUALS *actuals_ptr = &actuals;
57
58 #pragma code high_vector=0x08
59 void interrupt_at_high_vector(void)
60 {
61 _asm GOTO high_isr _endasm
62 }
63 #pragma code
64
65 #pragma interrupt high_isr
66 void high_isr (void)   // Interrupt Service Routine
67 // SPI Interrupt  (SPI-Master send command)
68 // Extern Interrupts
69 //  PORTB0: PS(0)
70 //  PORTB1: PS(HF)
71 // Timer Interrupts:
72 //  Timer1: stop RF !
73 //  Timer2: for simulation Mode only
74
75 {
76   INTCONbits.GIEH = 0;   // Interrupts off
77   if( PIR1bits.SSPIF)    // SPI Interrupt ?
78     {
79       SpiTask();
80       PIR1bits.SSPIF=0;   // clear Interrupt Flag
81     }
82
83   if(INTCONbits.INT0IF ) // PORTB0 interrupt: PS(0) // start new integration
84     {
85       PROG_STATUSbits.ps=1;
86       INTCONbits.INT0IF=0;
87     }
88
89   if(INTCON3bits.INT1IF ) // PORTB1 interrupt: PS(HF) // RF = ON
90     {
91       if(PROG_STATUSbits.ein)
92         {
93           LATFbits.LATF6=1;     // Enable DAC
94           TRISHbits.TRISH5=0;   // Extern LED: Prog OK ist output
95         }
96       WriteTimer1(actuals_ptr->dauer);    // start Dauer
97       INTCON3bits.INT1IE=0;               // deaktivate extern interrupt PORTB1 : RF trigger
```

```
98        INTCON3bits.INT1IF=0;                // Reset PORTB1 interrupt
99      }
100
101   if(PIR1bits.TMR1IF)          // Timer1 Interrupt (16 Bit overflow); stop RF !
102      {
103        LATFbits.LATF6=0;       // Disable DAC
104        PIR1bits.TMR1IF=0;      // Reset Interruptflag
105        INTCON3bits.INT1IE=1;   // enable extern interrupt PORTB1 : RF trigger
106      }
107
108   if(PIR1bits.TMR2IF)         // Timer2 Interrupt (Reached PR2); Only used for
109                               //         simulation mode: reanable DAC
110      {
111        PIR1bits.TMR2IF=0;           // Reset Interruptflag
112        if(PROG_STATUSbits.ein)
113          {
114            LATFbits.LATF6=1;      // Enable DAC
115            TRISHbits.TRISH5=0;    // Extern LED: Prog OK ist output
116          }
117        WriteTimer1(actuals_ptr->dauer);     // start Dauer
118        INTCON3bits.INT1IE=0;                // deaktivate extern interrupt PORTB1 : RF trigger
119        INTCON3bits.INT1IF=0;                // Reset PORTB1 interrupt
120        WriteTimer2(0);
121      }
122
123    INTCONbits.GIEH = 1;        // Interrupts on
124  }
125
126  int do_ad(void)
127  {
128    Convert_my_adc();                    // start conversion on extern ADC
129    while(Busy_my_adc());                // waiting for ADC
130    return( (Read_my_adc()-0x7ff) );     // read extern ADC ; 0x7ff is 0V at ADC.
131  }
132
133  // read command from SPI an execute  it
134  void SpiTask()
135  {
136    MYSPI myspi;
137    char magic_counter=0;
138    char *tmp;
139    unsigned char i;
140
141    myspi.command=SSPBUF; // read Byte from SPI Buffer
142
143    do { // Read MAGICs and increment magic_counter
144      // MAGIC is a SPI dummy command, used for synchronization
145      magic_counter++;
146    } while ((unsigned) MAGIC == (myspi.command=passive_read()));
147
148    if(magic_counter > 5)      // got 5 or more MAGICs
149      {
150        switch(myspi.command)
151          {
152          case TEIN:                      // Toggle EIN/AUS
153            if(passive_read())
154              {
```

```
155               PROG_STATUSbits.ein=1;
156             }
157           else
158             {
159               PROG_STATUSbits.ein=0;
160               LATFbits.LATF6=0;         // Disable DAC
161               TRISHbits.TRISH5=1;       // Extern LED: Prog OK is now Input
162             }                           // => LED stays off
163           break;
164         case 0:
165           break;
166         case SIMUL:   // Simulates PORTB1 Interrupt
167           PROG_STATUSbits.simul ^=1;
168           if(PROG_STATUSbits.simul)
169             {
170               OpenTimer2(TIMER_INT_ON & T2_PS_1_16 & T2_POST_1_16); // ca. 50 Hz for simulation
171               WriteTimer2(0);
172             }
173           else CloseTimer2();
174           break;
175         case AUS:
176           PROG_STATUSbits.ein=0;
177           LATFbits.LATF6=0;             // Disable DAC
178           TRISHbits.TRISH5=1;           // Extern LED: Prog OK is now Input
179           break;                        // => LED stays off
180         case EIN:
181           PROG_STATUSbits.ein=1;
182           break;
183         case S_RESET:
184           Reset();  // asm reset command
185           break;
186         case S_CDB: // SPI-Master sends CDB
187           actuals_ptr->cdb=passive_read();
188           break;
189         case S_CB4: // SPI-Master sends CB4
190           actuals_ptr->cb4=passive_read();
191           break;
192         case S_C0: // SPI-Master sends C0
193           actuals_ptr->c0=passive_read()+passive_read()*256;
194           break;
195         case S_DAUER: // SPI-Master sends DAUER
196           actuals_ptr->dauer=passive_read()+passive_read()*256;
197           break;
198         case G_ASK_PROG_DBINMAX:// SPI-Master asks for CDB, so send it
199           passive_send((actuals_ptr->dbinmax)>> 8);
200           passive_send(actuals_ptr->dbinmax);
201           break;
202         case S_PROG_FIND_DBINMAX:
203           actuals_ptr->dbinmax=getDBinMax();
204           break;
205         case G_ACTUALS: // SPI master asks for actuals, so send them
206           actuals_ptr->status=*((char *)prog_status_ptr);  // get Status Byte
207           tmp=(char *) actuals_ptr;                        // cast of actuals Pointer
208           for(i=0;i<sizeof(SPI_ACTUALS);i++)
209             {
210               passive_send( *(tmp++));                    // send actuals
211             }
```

```
212           break;
213         }
214       myspi.command=0;
215     }
216 }
217
218
219 int getDBinMax(void)     // find Maximum dB input Signal and return it
220 {
221     int db;
222   char dummy;
223   char i=0;
224   INTCONbits.GIEH = 0;  // Interrupts off
225   LATFbits.LATF6=0;     // Disable DAC
226   dbinmax=0;
227   WriteTimer1(0);       // Reset Timer1
228   while(i<50)
229     {
230       while(TMR1H <=(unsigned) 254)    // scans as long as possible dB input Signal for maximum
231         {
232           db = do_ad() ;               // Read extern ADC
233           if (db > dbinmax && db <2048) dbinmax=db;
234           dummy=TMR1L;                 // Read Timer Low Bytes => TMR1H gets valid
235         }
236       i++;
237     }
238   INTCONbits.GIEH = 1;    // Interrupts on
239   return dbinmax;
240 }
241
242 unsigned char passive_read(void)  // read from SPI. Master has SPI CLock
243 {
244   unsigned char read;
245   while(!PIR1bits.SSPIF);
246   read=SSPBUF;
247   PIR1bits.SSPIF=0;
248   return ( read );
249 }
250
251 void passive_send(char comma) // send Byte to SPI. Master has Clock
252 {
253   PIR1bits.SSPIF = 0;       // Clear interrupt flag
254   SSPBUF = comma;           // write byte to SSP1BUF register
255   while(!PIR1bits.SSPIF);   // wait until bus cycle complete
256   PIR1bits.SSPIF = 0;       // Clear interrupt flag
257 }
258
259 void main(void)
260 {
261   static unsigned char time;
262   static unsigned char tmp;
263   static int db;
264   static int b;
265   static unsigned int b2;
266   static unsigned int b4;
267   static int integral;        // 16 Bit signed
268   static int prog;
```

```
269
270   PROG_STATUSbits.ein=0;
271   PROG_STATUSbits.simul=0;
272   actuals_ptr->c0=0;
273   actuals_ptr->cdb=0;
274   actuals_ptr->dauer=0;
275   actuals_ptr->dbinmax=0;
276   TRISHbits.TRISH5=0;
277   TRISG=0;
278
279   for (tmp=0; tmp < (unsigned )25; tmp++) // Signals Startup
280     {
281       LATHbits.LATH5 ^=1;
282       Delay10KTCYx(20);
283     }
284
285
286   TRISBbits.TRISB0=1; //< PORTB0 is input for PS     : new integral
287   TRISBbits.TRISB1=1; //< PORTB1 is input for PS-HF : gate
288
289   TRISC=0; // set C,D and E as Outputs
290   TRISCbits.TRISC5=0; // PORTC5: SPI: SDO
291   TRISCbits.TRISC4=1; // PORTC4: SPI: SDI
292   TRISCbits.TRISC3=1; // PORTC1: SPI: SCK, Slave has clock input
293   TRISFbits.TRISF7=1; // PORTF7: SPI: SS, Slaveselect
294   TRISD=0;
295   TRISE=0;
296   PORTG=0;
297
298   LATC=0; // set C,D and E zero
299   LATD=0;
300   LATE=0;
301
302   ADCON1=0xf; // disbale internal ADC
303
304   //Extern ADC
305   TRISFbits.TRISF4=0; // R/aC   Read and anti-convert
306   TRISFbits.TRISF2=1; // Status
307   TRISH=0xf;          // set H0-4 and J as inputs (read from ADC)
308   TRISJ=255;
309
310   // DAC
311   TRISFbits.TRISF5=0;
312   TRISFbits.TRISF6=0;
313   LATFbits.LATF5=0;
314   LATFbits.LATF6=0;
315
316   OpenTimer0(TIMER_INT_OFF & T0_16BIT & T0_SOURCE_INT & T0_PS_1_32);
317   OpenTimer1(TIMER_INT_ON & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_8); // Dauer
318   INTCONbits.INT0IE=1;     // aktivate extern interrupt RB0 : new integral
319   INTCON3bits.INT1IE=1;    // aktivate extern interrupt RB1 : RF trigger
320   INTCON2bits.INTEDG0=1;   // Define Edges of triggers
321   INTCON2bits.INTEDG1=1;
322   INTCONbits.PEIE=1;       // aktivate periphery interrupts
323   INTCONbits.GIEH = 1;     // aktivate global interrupts
324   PIE1bits.TMR1IE = 1;     // Timer1 Interrupt
325   PIE1bits.TMR2IE = 1;     // Timer2 Interrupt
```

```
326
327
328  WriteTimer0(0);                        // Timer0 initialization: Task of Timer0: sampling rate
329  WriteTimer1(actuals_ptr->dauer);       // Timer1 initialization: Task of Timer 1: Dauer
330
331  PR2=254; // Timer2 overflow Register: 254 = ca. 50 Hz;
332
333 // Configure SPI Module
334 SSPCON1bits.SSPEN = 0;// start configuration by disabling
335 SSPSTATbits.SMP = 0;
336 SSPSTATbits.CKE = 1;  // CKP is 1 so this sets data transmit on rising SCK edge
337 SSPSTATbits.BF  = 0;  // set Buffer Full status to "SSBUF empty"
338 SSPCON1=5;            // defines SPI Mode and Frequenzy, compatible to ENC28J60
339 PIE1bits.SSPIE=1;     // SPI Interrupt
340 PIR1bits.SSPIF = 0;   // clear the SSP interrupt flag
341 SSPCON1bits.SSPEN = 1;// complete configuration by re-enabling
342
343 TRISHbits.TRISH5=1;   // Disable Status LED
344
345  while(1) // ADC->DAC loop: Main loop
346    {
347         if(TMR0L >=(unsigned) 13) // ca 80 $\micro s$
348        {
349          TMR0H=0; // Set Timer0 to zero
350          TMR0L=0;
351
352          // get new analog value from extern ADC:
353          Convert_my_adc();               // start conversion on extern ADC
354          while(Busy_my_adc());           // waiting for ADC
355          db= PORTJ + PORTH*256 -0x7ff ;  // read extern ADC ; 0x7ff is 0V at ADC.
356
357          db*=-1;                      // Extern dbIn has wrong sign, so toggle it
358
359          tmp= db < 0;  // is db negative ?
360          db  >>= 4;    // division by 2^4;
361
362          // if db was negative then correct Two's complement
363          if (tmp) db |= 0b1111000000000000;
364
365          LATHbits.LATH5 =1;       // Toggle extern LED: Prog. Ok
366          integral += (db) ;       // integration
367          LATHbits.LATH5 =0;       // Toggle extern LED: Prog. Ok
368          tmp=integral < 0;        // is integral negative ?
369          b=integral >> 6;         // division by 2^6
370          // if b was negative correct Two's complement
371          if(tmp) b |= 0b1111110000000000;
372
373          if (PROG_STATUSbits.ps) // new integral
374            {
375              integral=0;
376              PROG_STATUSbits.ps=0;
377            }
378
379          b2=(b*b)   >> 8;
380          b4=(b2*b2) >> 8;
381          prog=actuals.cdb*db+actuals.cb4*b4+actuals.c0;
382
```

```
383             LATE = prog ;                   // DAC low bits
384             LATD = (char)(prog >> 8);       // DAC high bits
385             }
386     }
387 }
```

A.2. SPI-Master

Listing A.2: Programm des PIC18F6680: SPI-Master, Kommunikation und Steuerung

```
1  /* PIC 18 Program for: Synchrotron-HF-Programmgenerator:
2   * Last modified 2011-05-24 by J-P Thiry
3   *      Device U11; Kommunikation und Steuerung
4   * This is the SPI Master: four Slaves on Board:
5   *   1. U3: ENC28J60: Ethernet driver. Connects the  ELSA Control-System with
6   *           this Board
7   *   2. U8: PIC18F8860: Programmberechnung:
8   *   3. U5: AD? : Digital Poti for input scaling
9   *   4. U15: AD?: Digital Poti ("Hoehe") for output scaling
10  *
11  * The Michroship TCP/IP Stack is used, so this source file depends on:
12  *    HardwareProfile.h, my_udp.h TCPIPConfig.h UDP.h IP.h spi.h Tick.h TCPIP.h
13  *    test-eth.c StackTsk.c ARP.c ENC28J60.c DHCP.c UDP.c Tick.c ICMP.c IP.c
14  *    Helpers.c Delay.c
15  * look into spi.h for defintions of commands
16  */
17
18
19 #include <p18f6680.h>
20 #define THIS_IS_STACK_APPLICATION //define as entry point
21
22 #include <tcpi/TCPIP.h> //include TCPIP headers
23 #include "spi.h"
24 #include "my_udp.h"
25
26 // 1. SPI-Slave: ENC28J60 Ethernet Driver
27 #define ENC_SPI_OFF  LATCbits.LATC2=0;
28 #define ENC_SPI_ON  LATCbits.LATC2=1;
29
30 // 2. SPI-Slave: PIC18F8860: calculation of RF-Program
31 #define PROG_SPI_ON   LATCbits.LATC1=0;
32 #define PROG_SPI_OFF  LATCbits.LATC1=1;
33
34 // Pin C7 is wired to Reset-Pin on U8.
35 #define RESET_PROG  LATCbits.LATC7=0; Nop(); Nop(); Nop(); Nop(); LATCbits.LATC7=1;
36
37 // 3. SPI-Slave: Digital-Poti for input-scaling
38 #define IN_SCALER_ON   LATDbits.LATD1=0;
39 #define IN_SCALER_OFF  LATDbits.LATD1=1;
40
41 // 4. SPI-Slave: Digital-Poti for output-scaling: Hoehe
42 #define PROG_HOEHE_ON   LATDbits.LATD0=0;
43 #define PROG_HOEHE_OFF  LATDbits.LATD0=1;
44
45
```

```
46 // define two LEDs for output
47 #define DEBUG_GREEN  LATGbits.LATG0=~PORTGbits.RG0;
48 #define DEBUG_YELLOW LATGbits.LATG2=~PORTGbits.RG2;
49
50 void InterruptServiceLow(void);
51 char SpiTask(void);
52
53 #pragma code InterruptVectorLow = 0x8
54 void InterruptVectorLow(void)
55 {
56   _asm
57     goto InterruptServiceLow
58     _endasm
59     }
60 #pragma code /* return to the default code section */
61 #pragma interrupt InterruptServiceLow
62
63 void InterruptServiceLow()
64 {
65   TickUpdate();
66 }
67
68 volatile char global_error;
69 volatile char do_push=0;
70
71 static TICK t = 0;
72
73 static DWORD dwLastIP = 0;
74 BYTE * my_byte;
75
76 static UDP_SOCKET my_udp_socket;
77 static UDP_PORT my_udp_port;
78
79 static int set_cdb=0;
80 static int set_cb4;
81 static int set_c0;
82 static int set_dauer;
83 static int set_hoehe;
84 static int set_scaler;
85 static char set_ein=0;
86
87
88 // IP-Stack
89 APP_CONFIG AppConfig; //holds TCPIP setup, refer to TCPIPConfig.h
90 static ROM BYTE SerializedMACAddress[6] = {MY_DEFAULT_MAC_BYTE1,
91                                            MY_DEFAULT_MAC_BYTE2,
92                                            MY_DEFAULT_MAC_BYTE3,
93                                            MY_DEFAULT_MAC_BYTE4,
94                                            MY_DEFAULT_MAC_BYTE5,
95                                            MY_DEFAULT_MAC_BYTE6};
96
97 static void InitHardware(void);
98 static void panic(char);
99 static void PushActuals(void);
100 static void get_dbINmax(void);
101 static void do_Norm(void);
102 static void do_reset_prog(void);
```

```
103
104 MYSPI myspi;
105 volatile SPI_ACTUALS spi_actuals;
106 volatile SPI_ACTUALS *spi_actualsptr = &spi_actuals;
107
108 char ProcessMyUDP(COMMAND_PACKAGE udp_command)
109 {
110   myspi.command=udp_command.cmd; // got command from UDP
111   myspi.SPInew=1;       // we got a command  from UDP so a SPI action is needed
112   switch(udp_command.cmd)
113     {
114     case TEIN:
115       set_ein=(unsigned char) udp_command.data;
116       break;
117     case S_C0:
118       set_c0=(int) udp_command.data;
119       break;
120     case S_CDB:
121       set_cdb=(unsigned char) udp_command.data;
122       break;
123     case S_CB4:
124       set_cb4=(unsigned char) udp_command.data;
125       break;
126     case S_DAUER:
127       set_dauer=udp_command.data;
128       break;
129     case S_HOEHE:
130       set_hoehe =(unsigned char) udp_command.data;
131       PROG_HOEHE_ON;
132       spi_send(set_hoehe); // send new hoehe to hoehe POTI
133       spi_actualsptr->hoehe=set_hoehe;
134       myspi.SPInew=0;
135       PROG_HOEHE_OFF;
136       PushActuals();       // tell controll-system
137       break;
138     case S_IN_SCALER:
139       set_scaler =(unsigned char)udp_command.data;
140       IN_SCALER_ON;
141       spi_send(set_scaler); // send new input Scaler to IN-SCALER-Poti
142       spi_actualsptr->scaler=set_scaler;
143       myspi.SPInew=0;
144       IN_SCALER_OFF;
145       PushActuals();  // tell control-system#
146       break;
147       }
148 }
149
150 void PushActuals(void)
151 {
152   if(UDPIsPutReady(my_udp_socket)) // Push actuals to Control-System
153     {
154       if(UDPPutArray((BYTE*) spi_actualsptr, sizeof(SPI_ACTUALS)) != sizeof(SPI_ACTUALS))
155         {
156           panic(1); // something wrong
157         }
158     }
159   else panic(2);  // obscure Error happend
```

```
160   UDPFlush();
161 }
162
163
164 void panic(char error_code)
165 {
166   global_error=error_code;
167   while(1)
168     {
169       TICK pt;
170       if(TickGet() - pt >= TICK_SECOND/5ul){ //blink Yellow LED with 5 Hz
171                                              //endless Loop
172         pt = TickGet();
173         DEBUG_YELLOW;
174       }
175     }
176 }
177
178 static void do_reset_prog(void)
179 {
180   RESET_PROG; // wired Reset on PROG
181 // loop until valid read from PROG, that means PROG did a successful reset
182   do
183     {
184       myspi.command=G_ACTUALS;
185       myspi.SPInew=1;
186       SpiTask(); // Read new Actuals from PROG
187     }       while(spi_actuals.status == 0xff || spi_actuals.c0 != 0);
188 PushActuals(); // tell Control-System
189 }
190
191 static void get_dbINmax(void)
192 {
193   myspi.command=S_PROG_FIND_DBINMAX;
194       myspi.SPInew=1;
195       SpiTask(); // Ask Prog MCU to find dbINmax
196       DelayMs(250); // wait for PROG
197       DelayMs(250);
198       DelayMs(250);
199       myspi.command=G_ASK_PROG_DBINMAX;
200       myspi.SPInew=1;
201       SpiTask();// Ask Prog MCU for dbINmax value
202       myspi.SPInew=0;
203 }
204
205 static void do_Norm(void) // calls get_dbINmax and calculate new value for IN_SCALER
206 {
207   long unsigned int tmp=0;
208   char i=0;
209   spi_actualsptr->dbinmax=0;
210   while(spi_actualsptr->dbinmax==0)
211     {
212       if( i > 4) {
213         do_reset_prog();
214         return;
215       }
216       i++;
```

```
217       IN_SCALER_ON;
218       spi_send(0);
219       IN_SCALER_OFF;
220       get_dbINmax();
221     }
222
223
224   // scaler_max is 255.
225   // dbinmax_max is 2048  (12 bit with sign)
226   tmp=(1044480/spi_actualsptr->dbinmax)-510; // magic calculation of new IN_SCALER
227   // if dbINmax gets near 2048 but well below 2048 we have found best new IN_SCLAER
228   if(tmp> (unsigned) 254) tmp=254;
229   set_scaler=tmp;
230   IN_SCALER_ON;
231   spi_send((char) set_scaler); // set the scaler
232   IN_SCALER_OFF;
233   spi_actualsptr->scaler=(char) set_scaler;
234   myspi.SPInew=0;
235   get_dbINmax(); // verify new dbINmax
236 }
237
238 unsigned char spi_read(void)
239 {
240   unsigned char TempVar;
241   Delay10TCYx(16);
242   spi_send(0);
243   while(!PIR1bits.SSPIF);     // wait until bus cycle complete
244   PIR1bits.SSPIF=0;
245   TempVar= SSPBUF;
246   return (TempVar);
247 }
248
249 char spi_send(char comma)
250 {
251   unsigned char TempVar;
252   PIR1bits.SSPIF = 0;     // Clear interrupt flag
253   SSPBUF = comma;         // write byte to SSP1BUF register
254   while(!PIR1bits.SSPIF);  // wait until bus cycle complete
255   TempVar=SSPBUF;
256   return ( 0 );              // if WCOL bit is not set return non-negative
257 }
258
259  // if no new SPI command arrived over UDP (from Control-System)
260  // this function immediately returns, else: Process SPI command char SpiTask()
261 char SpiTask(void)
262 {
263   unsigned char i;
264
265   if(myspi.SPInew) // else return
266     {
267       myspi.SPInew=0;
268       PROG_SPI_ON;
269       myspi.SPInew=0;
270       spi_send(MAGIC);
271       spi_send(MAGIC);
272       spi_send(MAGIC);
273       spi_send(MAGIC);
```

```
274         spi_send(MAGIC);
275         spi_send(MAGIC);
276         spi_send(MAGIC);
277         spi_send(MAGIC);
278         spi_send(MAGIC);
279         spi_send(MAGIC);
280
281         switch (myspi.command)
282           {
283           case TEIN:
284             spi_send(TEIN);
285             spi_send(set_ein);
286             myspi.command=G_ACTUALS;
287             myspi.SPInew=1;
288             break;
289           case EIN:
290             spi_send(EIN);
291             myspi.command=G_ACTUALS;
292             myspi.SPInew=1;
293             break;
294           case AUS:
295             spi_send(AUS);
296             myspi.command=G_ACTUALS;
297             myspi.SPInew=1;
298             break;
299           case SIMUL:
300             spi_send(SIMUL);
301             myspi.command=G_ACTUALS;
302             myspi.SPInew=1;
303             break;
304           case S_CDB:
305             spi_send(S_CDB);
306             spi_send((unsigned char)set_cdb);
307             myspi.command=G_ACTUALS;
308             myspi.SPInew=1;
309             break;
310           case S_CB4:
311             spi_send(S_CB4);
312             spi_send((unsigned char)set_cb4);
313             myspi.command=G_ACTUALS;
314             myspi.SPInew=1;
315             break;
316           case S_C0:
317             spi_send(S_C0);
318             spi_send((char) (set_c0 >> 8) );
319             spi_send((char) (set_c0));
320             myspi.command=G_ACTUALS;
321             myspi.SPInew=1;
322             break;
323           case S_DAUER:
324             spi_send(S_DAUER);
325             spi_send(high_byte(set_dauer));
326             spi_send(low_byte(set_dauer));
327             myspi.command=G_ACTUALS;
328             myspi.SPInew=1;
329             break;
330           case S_RESET:
```

```
331           do_reset_prog();
332           myspi.command=G_ACTUALS;
333           myspi.SPInew=1;
334           break;
335         case T_SHOW_B:
336           spi_send(T_SHOW_B);
337           myspi.command=G_ACTUALS;
338           myspi.SPInew=1;
339           break;
340         case T_SIGN:
341           spi_send(T_SIGN);
342           myspi.command=G_ACTUALS;
343           myspi.SPInew=1;
344           break;
345         case G_ACTUALS:
346           spi_send(G_ACTUALS);
347           for(i=0;i<sizeof(SPI_ACTUALS);i++)
348             {
349               *(( (char *) spi_actualsptr)+i)=spi_read();
350             }
351           spi_actuals.hoehe=set_hoehe;
352           spi_actuals.scaler=set_scaler;
353           do_push=1;
354           break;
355         case S_NORM:
356           do_Norm();
357           do_push=1;
358           break;
359         case G_DBINMAX:
360           myspi.SPInew=0;
361           get_dbINmax();
362           break;
363         case  G_ASK_PROG_DBINMAX:
364           spi_send(G_ASK_PROG_DBINMAX);
365           spi_actualsptr->dbinmax=spi_read()+spi_read()*256;
366           do_push=1;
367           break;
368         case S_PROG_FIND_DBINMAX:
369           spi_send(S_PROG_FIND_DBINMAX);
370           break;
371         }
372
373       PROG_SPI_OFF;
374     }
375   return 0;
376 }
377
378 void main(void){
379   COMMAND_PACKAGE udp_command;
380   int i;
381   NODE_INFO remote_node;
382
383   TRISG=0;
384   LATG=0;
385   TRISG=0;
386
387   InitHardware(); //setup hardware
```

```
388
389   i=20;
390   while(i>1) // signals Startup
391     {
392       DEBUG_YELLOW;
393       DelayMs(20);
394       i--;
395     }
396
397   LATGbits.LATG2=0; // Yellow LED
398   TickInit();    //setup the tick timer
399
400   //setup the TCPIP stack config variable
401   AppConfig.Flags.bIsDHCPEnabled = FALSE;
402   AppConfig.Flags.bInConfigMode = FALSE;
403   AppConfig.MyIPAddr.Val = MY_DEFAULT_IP_ADDR_BYTE1 | MY_DEFAULT_IP_ADDR_BYTE2<<8ul | MY_DEFAU
404   AppConfig.DefaultIPAddr.Val = AppConfig.MyIPAddr.Val;
405   AppConfig.MyMask.Val = MY_DEFAULT_MASK_BYTE1 | MY_DEFAULT_MASK_BYTE2<<8ul | MY_DEFAULT_MASK_
406   AppConfig.DefaultMask.Val = AppConfig.MyMask.Val;
407
408   StackInit();  //setup the stack
409
410   // open UDP socket <-- conect ELSA-Controllsystem on this socket !
411   my_udp_socket = UDPOpen(1200,NULL,1200);
412   if(my_udp_socket == INVALID_UDP_SOCKET) panic(3);
413
414   spi_actualsptr->scaler=0;
415   spi_actualsptr->hoehe=0;
416
417   while(1){ //never ending loop
418     if(TickGet() - t >= TICK_SECOND){ //Blink Link-OK every Second
419       t = TickGet();
420       DEBUG_YELLOW;
421     }
422
423     StackTask();
424
425     if(UDPIsGetReady(my_udp_socket))
426       {
427         DEBUG_GREEN; // got valid UDP Packet (from ELSA-controll system)
428         if(UDPGetArray((BYTE*) &udp_command, sizeof(COMMAND_PACKAGE)) )
429           {
430             ProcessMyUDP(udp_command);
431           }
432         else // something went terribly wrong
433           {
434             RESET_PROG;  // Reset Prog
435             Reset();     // Self-Reset
436           }
437       }
438
439     SpiTask(); // if no new SPI command arrived over UDP (from Control-System)
440                // this function immediately returns, else: Process SPI command
441
442     if(do_push) // did SpiTask asked for pusshing of actuals ?
443       {
444         PushActuals(); // do so
```

```
445         do_push=0;
446       }
447   }
448 }
449
450
451
452 //configures the PIC hardware
453 static void InitHardware(void){
454
455   // set TRIS
456   TRISCbits.TRISC1=0; // PROG_SPI
457   TRISCbits.TRISC7=0; // RESET_PROG
458   TRISCbits.TRISC2=0; // PORTC5: SPI: SS Slaveselect ENC
459   TRISCbits.TRISC0=0; // PORTC5: ENC RESET
460   TRISFbits.TRISF6=0; // DEBUG_YELLOW
461   TRISGbits.TRISG0=0; // DEBUG_GREEN
462   TRISGbits.TRISG2=0; // DEBUG_RED
463   TRISD=0;            // PORTD LEDs
464
465   PROG_HOEHE_OFF;
466   IN_SCALER_OFF;
467   RESET_PROG;
468
469   INTCONbits.GIEH = 1;    // Interrupts on
470   INTCONbits.PEIE = 1;    // Interrupts on
471   INTCONbits.TMR0IE=1;
472   DEBUG_GREEN;
473   PROG_SPI_OFF;
474   ENC_SPI_ON;
475   LATCbits.LATC0=1;    // ENC RESET
476 }
```

A.3. spi.h

Listing A.3: Definitionen für SPI

```
1  // Version 1.1
2  // must be included in every program
3
4  // Define SPI commands
5  #define AUS       3       // AUS
6  #define EIN       4       // EIN
7  #define SIMUL     5       // Simulation Mode: Trigger Signals are simulated
8  #define TEIN      7       // Toggle EIN/AUS
9
10 #define G_ACTUALS 36      // We are asked for actual values
11
12 //#define AMAGIC    42
13 #define T_SHOW_B 50
14 #define T_SIGN   51
15 #define MAGIC    85       // MAGIC-Token, used for synchronization of SPI comunication
16
17 #define G_DBINMAX 94      // ELSA-Controllsystem asks for dBinMax
18 #define G_ASK_PROG_DBINMAX 93  // Ask SPI_Slave for dBinMax
```

```
19
20 #define S_RESET  95         // call Reset
21
22 #define S_NORM   96         // call do_Norm: calls get_dbINmax and calculate new value for IN_S
23 #define S_PROG_FIND_DBINMAX 121 // ask SPI-Slave to start search of dBinMax
24
25 #define S_IN_SCALER 122     // Set Input-Scaler
26 #define S_HOEHE  123        // Set Hoehe
27 #define S_DAUER  124        // Set Dauer
28 #define S_C0     125        // Set C0
29 #define S_CB4    126        // Set CB4
30 #define S_CDB    127        // Set CDB
31
32 // 8 Bit Status Register
33  volatile struct PROG_STATUS_t {
34   unsigned ein:1;
35   unsigned sign:1;
36   unsigned show_B:1;
37   unsigned simul:1;
38   unsigned ps:1;
39   unsigned hf_inhibit:1;
40 } PROG_STATUSbits ;
41
42 struct PROG_STATUS_t * prog_status_ptr=&PROG_STATUSbits;
43
44
45 // 8 Bit SPI Register
46 typedef struct MYSPI {
47   unsigned SPInew:1;
48   unsigned command:7;
49 } MYSPI;
50
51
52
53 typedef struct SPI_ACTUALS {
54   unsigned char cdb;
55   unsigned char cb4;
56   int c0;
57   int dbinmax;
58   unsigned int dauer;
59   unsigned char status;
60   unsigned char hoehe;
61   unsigned char scaler;
62   unsigned char dummy;
63 } SPI_ACTUALS;
64
65 void do_my_reset(void);
66 char spi_send(char comma);
67 unsigned char spi_read(void);
68
69
70 char high_byte(int var)
71 {
72   return (char) (var >> 8);
73 }
74
75 char low_byte(int var)
```

```
76 {
77 return (char) var;
78 }
```

A.4. my_udp.h

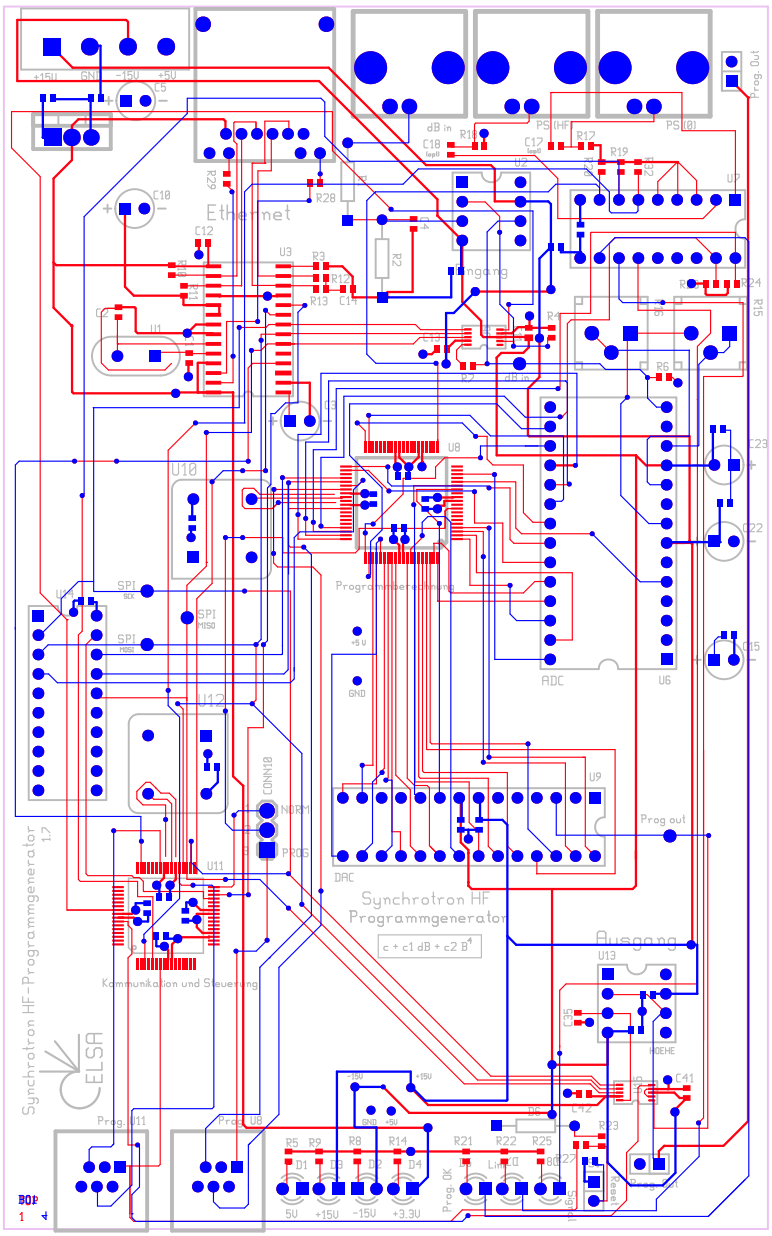
Listing A.4: Definitionen für UDP

```
1  #ifndef __MY_UDP_H
2  #define __MY_UDP_H
3
4
5
6  typedef struct COMMAND_PACKAGE_T
7  {
8    unsigned char cmd; // 8 byte command
9    unsigned char dummy;
10   int data; // 16 bit data
11 } COMMAND_PACKAGE;
12
13
14 // Warning: Muss auch in Kontrollsystem-Programmen implementiert werden. mit an die Architektur ang
15 typedef struct ACTUAL_VALUES_T
16 {
17   unsigned char cdb;
18   unsigned char cb4;
19   int c0;
20   unsigned int dauer;
21   int dbinmax;
22   unsigned char status;
23 } ACTUAL_VALUES;
24
25 #endif
```

Anhang B.

Programmgenerator Platine

B.1. Layout der Platine



B.2. Schaltplan

.

Literatur

[Alt+68] K.H. Althoff u. a. »The 2.5 GeV Electron Synchrotron of the University of Bonn«. In: *Nuclear Instruments and Methods* 61.1 (1968), S. 1 –30. ISSN: 0029-554X. DOI: DOI:10.1016/0029-554X(68)90443-6. URL: http://www.sciencedirect.com/science/article/B73DN-471XPWR-R7/2/17131ec35337688b196d9e190f459529 (siehe S. 15–18).

[Bau05] P. Baudrenghien. »LOW-LEVEL RF SYSTEMS FOR SYNCHROTRONS, Part II:High intensity. Compensation of beam-induced effects«. In: *RADIO FREQUENCY ENGINEERING*. Hrsg. von J. Milrd. CERN ACCELERATOR SCHOOL. 2005 (siehe S. 11).

[Bot90] W Bothe. »Resonant excitation of synchrotron magnets«. In: DESY-M-90-01 (1990), 45 p (siehe S. 17).

[HH89] Paul Horowitz und Winfield Hill. *The Art of Electronics*. 2. Aufl. Cambridge University Press, Juli 1989. ISBN: 0521370957 (siehe S. 41).

[Hil] W. Hillert. »The Bonn Electron Stretcher Accelerator ELSA: Past and Future«. In: *The European Physical Journal A* 28.S1 (), S. 139–148. ISSN: 1434-6001. DOI: 10.1140/epja/i2006-09-015-4. URL: http://dx.doi.org/10.1140/epja/i2006-09-015-4 (siehe S. 15).

[Hin97] Frank Hinterberger. *Physik der Teilchenbeschleuniger und Ionenoptik*. 1. Aufl. Springer-Verlag GmbH, Apr. 1997. ISBN: 3540612386 (siehe S. 10, 16).

[Hof97] Michael H. W. Hoffmann. *Hochfrequenztechnik: Ein systemtheoretischer Zugang (Springer-Lehrbuch) (German Edition)*. Springer, 1997. ISBN: 9783540616672 (siehe S. 6).

[Jac98] John David Jackson. *Classical Electrodynamics*. 3. Auflage. John Wiley & Sons, Dez. 1998. ISBN: 9780471309321 (siehe S. 3, 5).

[Kel51] Joseph M. Kelly. »Magnetic Field Measurements with Peaking Strips«. In: *Review of Scientific Instruments* 22.4 (1951), S. 256–258. DOI: 10.1063/1.1745904. URL: http://link.aip.org/link/?RSI/22/256/1 (siehe S. 19).

[Lee05] S. Y. Lee. *Accelerator Physics*. 2. Aufl. World Scientific Pub Co, Jan. 2005. ISBN: 9789812562005 (siehe S. 6).

[Net68] Hans Netter. »Programmierung und Regelung der HF-Feldstärke in den Beschleunigungssrecken des Bonner $2{,}5\,\mathrm{GeV}$-Elektronensynchrotrons«. Diplomarbeit. Universität Bonn, Juni 1968 (siehe S. 36).

[Nys59] Hans M. Nysäter. »Accurate Magnetic Field and Field Gradient Measuring Instrument for Dynamic Low Fields in a Synchrotron Magnet«. In: *Nuclear Instruments and Methods* 4.1 (1959), S. 44 –49. ISSN: 0029-554X. DOI: DOI:10.1016/0029-554X(59)90044-8. URL: http://www.sciencedirect.com/science/article/B73DN-470NYD9-4KC/2/e12cd2a9e5ff4ad730f41ba7e83e97b4 (siehe S. 19).

[Pic95] M. Picard. »Entwurf, Entwicklung und Inbetriebnahme eines verteilten Rechnerkontrollsystems zur Steuerung der Elektronen-Stretcher-Anlage ELSA, unter besonderer Berücksichtigung der Extraktion im Nachbeschleunigungsbetrieb bis 3,5 GeV«. Doktorarbeit. Universität Bonn, 1995 (siehe S. 50).

[Pir95] Werner Pirkl. »Longitudinal Beam Dynamics«. In: *CAS - CERN Accelerator School : 5th Advanced Accelerator Physics Course* (1995), S. 233–257. URL: http://cdsweb.cern.ch/record/302486/files/p233.pdf?version=1 (siehe S. 8).

[Pus09] Thorsten Pusch. »Lagemessung des extrahierten Strahls am Elektronenbeschleuniger ELSA mittels Hochfrequenzresonatoren«. Diplomarbeit. Universität Bonn, Okt. 2009 (siehe S. 5).

[Rei10] C. Reinsch. »Pulsformung am Linearbeschleuniger I der Beschleunigeranlage ELSA«. Bacherlor-Arbeit. Universität Bonn, Aug. 2010 (siehe S. 26).

[Rob64] K. W. Robinson. *Stability of Beam in Radiofrequency System*. Cambridge, MA: MIT, 1964 (siehe S. 12).

[Sch06] Schindl. *Instabilities*. 2006. URL: http://cdsweb.cern.ch/record/941317 (siehe S. 7).

[Sti67] H. E. Stier. »Das Hochfrequenz-Beschleunigungssystem des Bonner 2,3 GeV Electronensynchrotrons«. Diplomarbeit. Universität Bonn, Juli 1967 (siehe S. 20, 22, 23, 28).

[T95] Götz T. »Entwicklung und Inbetriebnahme eines verteilten Rechnerkontrollsystems zur Steuerung der Elektronen-Stretcher-Anlage ELSA, unter besonderer Berücksichtigung der Anforderungen des Nachbeschleunigungsbetriebes bis 3,5 GeV«. Doktorarbeit. Universität Bonn, 1995 (siehe S. 50).

[Tie02] Christoph Schenk Ulrich Tietze. *Halbleiter-Schaltungstechnik*. 12. Berlin: Springer, 2002. ISBN: 3-540-42849-6 (siehe S. 46).

[WSON56] M G White, F C Schoemaker und G K O Neill. »A 3 Bev High Intensity Proton-Synchrotron«. In: (1956) (siehe S. 17).

[Wal92] R. P. Walker. »SYNCHROTRON RADIATION«. In: *CAS - CERN Accelerator School : Fifth General Accelerator Physics Course* (1992), S. 437–460 (siehe S. 13).

[Wie95] Helmut Wiedemann. *Particle Accelerator Physics II: Nonlinear and Higher-Order Beam Dynamics*. 1. Aufl. Springer, Mai 1995. ISBN: 3540575642 (siehe S. 11).

[Wik11a] Wikipedia. *Transmission Control Protocol/Internet Protocol — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 26. Mai 2011]. 2011. URL: http://de.wikipedia.org/w/index.php?title=Transmission_Control_Protocol/Internet_Protocol&oldid=88971070 (siehe S. 48).

[Wik11b] Wikipedia. *User Datagram Protocol — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 26. Mai 2011]. 2011. URL: `http://de.wikipedia.org/w/index.php?title=User_Datagram_Protocol&oldid=87628545` (siehe S. 49).

[Wil96] Klaus Wille. *Physik der Teilchenbeschleuniger und Synchrotronstrahlungsquellen. Eine Einführung*. 2., überarb. u. erw. A. Teubner Verlag, 1996. ISBN: 3519130874 (siehe S. 4, 5, 7).

Danksagung

Ich danke Herrn PD Dr. Wolfgang Hillert für die Übertragung der spannenden Aufgabe und die Möglichkeit das Thema frei zu bearbeiten. Ausdrücklich möchte ich mich für das in mich gesetzte Vertrauen bedanken, an für den Betrieb der Anlage notwendigen Komponenten selbstbestimmt arbeiten zu dürfen.

Mein Dank gilt Herrn Prof. Dr. Kai-Thomas Brinkmann für die Übernahme des Korreferates.

Andreas Dieckmann stand mir immer für Fragen und Entscheidungsfindungen zur Verfügung. Seine unkomplizierte Art und uneingeschränkte Hilfsbereitschaft waren eine wichtige Stütze.

Ebenfalls bedanken möchte ich mich bei Frank Frommberger, der mir jederzeit bereitwillig Fragen zum Kontrollsystem beantwortet hat und mich bei der Programmierung der Anbindung tatkräftig unterstützt hat.

Außerdem gilt mein Dank Hans Bücking für seine umfangreiche Unterstützung bei der Realisierung der Hardware.

Außerdem danke ich der gesamten ELSA-Manschaft für die nette Aufnahme und bereitwillige Unterstützung. Alle anfallenden, zum Teil auch lästige Arbeiten wurden immer umgehend erledigt. Dafür danke ich ausdrücklich Herrn F. G. Engelmann, H. Schug, P. Mahlberg und insbesondere Herrn Klaus-Peter Faßbender, der mit mir viele Stunden alte Kabel nachverfolgen musste.

Weiterhin möchte ich mich für die umfangreiche Hilfe bei der Korrektur des Textes dieser Arbeit bei folgenden Personen herzlich bedanken: Oliver Boldt, Nikolas Heurich, Dennis Proft, Thorsten Pusch, André Roth, Manuel Schedler, Sven Zander, Rebecca Zimmermann, und Thomas Perlitius und Doro Fendel. Eure Hilfe hat wesentlich dazu beigetragen diesen Text in seine vorliegende Form zu bringen.

Zum Schluss möchte ich die wichtige Rolle von freien Software-Projekten hervorheben. Eine Vielzahl von Programmen und Projekten wurden bei der Programmierung, bei der Erstellung von Schaltplänen und des Platinenlayouts sowie bei der Anfertigung dieser Arbeit eingesetzt. Ohne die beeindruckende freiwillige Arbeit der vielen tausend Menschen, die diese Projekte aktiv entwickeln, wären viele der Aufgaben nur schwer lösbar gewesen. Nachfolgend eine unvollständige Liste wichtiger Projekte: `GNU/Linux`, `Debian`, `gcc`, `GNU geda Tools`, `pcb`, `gschem`, `latex`, `gnuplot`, `PGF/TikZ`, `pgfplots`, `gimp`, `inkscape`, `...`