Online-Bestimmung von Betriebsparametern an Van-de-Graaff-Ionenbeschleunigern

Masterarbeit

eingereicht am Fachbereich
ELEKTROTECHNIK
der Hochschule für Technik und Wirtschaft Dresden (FH)

zur Erlangung des akademischen Grades

Master of Science

vorgelegt
von
Holger Lange

Vorwort

Das Institut für Ionenstrahlphysik und Materialforschung des Forschungszentrums Rossendorf beschäftigt sich mit dem Einsatz von Ionenstrahlen in der Materialforschung und bei der Untersuchung von Halbleitern. So werden Forschungsarbeiten zur Oberflächenanalyse, zur Abscheidung dünner Schichten und zur Modifizierung von Festkörperoberflächen durchgeführt. Beispielsweise können Halbleitermaterialien mit Hilfe eines Ionenstrahls gezielt in Ort und Stärke dotiert werden, wodurch geringere Strukturweiten realisierbar sind als es zur Zeit durch übliche Lithographieverfahren möglich ist. Um noch feinere Strukturen analysieren oder fertigen zu können, werden immer höhere Anforderungen an die Qualität des eingesetzten Ionenstrahls gestellt. Besonders die Stabilität der Strahlenenergie während der laufenden Untersuchungen ist ein wichtiges Merkmal bei der Durchführung oben genannter Arbeiten. Aus diesem Grund gibt es Bestrebungen seitens des Forschungszentrums, Methoden zu entwickeln, mit denen die Stabilität der Ionenstrahlenergie weiter verbessert werden kann.

Im Rahmen dieser Masterarbeit wurde mir die Aufgabe übertragen, ein Verfahren zu entwickeln, mit dem es möglich sein sollte, während des Betriebes (online) eines Van-de-Graaff-Ionenbschleunigers Parameter zu erfassen, welche dessen arbeitspunktabhängige Eigenschaften widerspiegeln. Dazu sollen verschiedenartige Testsignale über eine bestehende Reglerstruktur dem System zugeführt und deren Auswirkungen ausgewertet werden. Anhand dieser ermittelten Faktoren kann in weiterführenden Arbeiten ein Algorithmus entwickelt werden, welcher auftretende Störungen erkennt und ihnen entgegenwirkt.

Im ersten Teil dieser Arbeit wird die Erstellung eines Softwaremodells des Ionenbeschleunigers beschrieben, mit dessen Hilfe geeignete Methoden zur Parameterschätzung entworfen und getestet werden sollen. Dabei wird die Funktionsweise der einzelnen Teilsysteme, in welche das Verhalten des Beschleunigers untergliedert wurde, erläutert und die daraus entstandene Modellstruktur beschrieben. Ausgewählte Simulationen und deren Vergleich mit gemessenen Signalverläufen bestätigen die gewählten Funktionen, mit denen das Verhalten nachgebildet wurde. In Abschnitt 2 wird ein mögliches Verfahren zur Bestimmung der Übertragungsfunktion von zwei gekoppelten Regelsystemen hergeleitet, über die eine aktive Beeinflussung der Betriebsparameter möglich ist.

Für die technische Realisierung des Verfahrens wird ein Mikrocontroller genutzt. Wie im Abschnitt 3 anhand von erstelltem Quellcode beschrieben, generiert der Controller während des laufenden Betriebs des Beschleunigers verschiedene Testsignale, misst parallel dazu die zur Berechnung benötigten Größen und bestimmt fortlaufend die Übertragungsfunktionen der beiden Teilsysteme.

Für das Vertrauen, welches mir das Forschungszentrum zur Bearbeitung dieser Aufgabe entgegengebracht hat, sowie für die persönliche Unterstützung durch meinen institutsseitigen Betreuer Herrn Dr. Bürger und durch Herrn Professor Wächter von der HTW-Dresden möchte ich mich sehr bedanken. Des weiteren gilt mein Dank meinen Eltern, die mir das gesamte Studium durch ihre Förderung ermöglicht haben.

Holger Lange Dresden, im November 2004

Inhaltsverzeichnis

Verzeichnis der Formelzeichen und Abkürzungen

ADW	Analog-Digital-Wandler	
c_n	komplexe Amplitude	
C	Kapazität	[F]
C_{En}	Kapazität zwischen Säule und Masse	[F]
CPU	Capacitive-Pick-Up (dt: Kapazitive Messelektrode)	
C_{RRn}	Kapazität zwischen zwei benachbarten Ringen der Säule	[F]
CS	Chip Select	
den	Nennerpolynom einer kontinuierlichen Übertragungsfunktion (denominator)	
$dend$	Nennerpolynom einer diskreten Übertragungsfunktion (discrete denominator)	
D	Durchgriff (Triode)	
DAW	Digital-Analog-Wandler	
EDK	Evaluation Development Kit	
$EMSP$	Energiemagnetseparator	
f_0	Frequenz der Grundschwingung	[Hz]
f_{Band}	Bandumlauffrequenz	[Hz]
f_{core}	Taktfrequenz des Mikrocontrollers	[Hz]
f_{periph}	Taktfrequenz der Peripherie des Mikrocontrollers	[Hz]
F_B	Übertragungsfunktion des Bandes	
F_{CPU}	Übertragungsfunktion der CPU	
F_{KT}	Übertragungsfunktion der Koronatriode	
FZR	Forschungszentrum Rossendorf	
$G(s)$	Übertragungsfunktion im Laplacebereich	
$G(z)$	Übertragungsfunktion im $\mathcal{Z}$-Bereich	
i_{infl_band}	vom Band in die Säule influenzierter Strom	[A]
i_{infl_t}	vom Band in das Terminal influenzierter Strom	[A]
I_a	Anodenstrom (Triode)	[A]
$I_{b\_ab\_a}$	zur Basis fließender Strom auf Bandaußenseite	[A]
$I_{b\_auf\_a}$	von Basis fließender Strom auf Bandaußenseite	[A]
I_{bi}	Basisstrom im Bandinneren	[A]
I_{entl}	Entladestrom	[A]

I_k	Kathodenstrom (Triode)	[A]
I_{lade}	Ladestrom	[A]
I_{lade_ist}	Ladestrom zum Vergleich im Ladestromregler	[A]
$I_{t\_ab\_a}$	vom Terminal fließender Strom auf Bandaußenseite	[A]
$I_{t\_auf\_a}$	zum Terminal fließender Strom auf Bandaußenseite	[A]
I_{ti}	Terminalstrom im Bandinneren	[A]
ISR	Interruptserviceroutine	
j	imaginäre Einheit	
k_h	Faktor der horizontalen Ladungsbewegung auf dem Band	
k_{ia}	Stromkoeffizient der Bandaußenseite	
k_{ii}	Stromkoeffizient des Bandinneren	
k_u	Spannungskoeffizient	[A/V]
k_v	Faktor der vertikalen Ladungsbewegung in das Band	
KT	Koronatriode	
LS	Ladestrom	
num	Zählerpolynom einer kontinuierlichen Übertragungsfunktion (numerator)	
$numd$	Zählerpolynom einer diskreten Übertragungsfunktion (numerator discrete)	
N	Anzahl der Abtastwerte	
PC	Personal Computer	
PLL	Phase-Locked Loop (Phasengekoppelte Regelschleife)	
R	Widerstand	[Ω]
R_{i_entl}	Innenwiderstand des Entladesystems	[Ω]
R_{i_lade}	Innenwiderstand des Ladesystems	[Ω]
R_n	Widerstand zwischen zwei benachbarten Ringen der Säule	[Ω]
RAM	Random Access Memory (Arbeitsspeicher)	
s	Laplace-Variable	
S	Steilheit (Triode)	[A/V]
t	Zeit	[s]
T	Periodendauer	[s]
T_a	Abtastzeitintervall	[s]
T_{Band}	Periodendauer eines Bandumlaufes	[s]
T_e	Zeitdifferenz in der PLL	[s]
T_{ion}	Laufzeit der Ionen in der Koronatriode	[s]

T_{TGRA}	Dauer einer Zählperiode in der PLL	[s]
u	Spannung	[V]
U_0	Schwellenspannung; Urspannung	[V]
U_a	Anodenspannung (Triode)	[V]
U_B	Eingangsspannung des Bandsystems	[V]
U_{cpu}	Spannung an CPU	[V]
U_{em_sp}	Eingangsspannung der Regler	[V]
U_g	Gitterspannung (Triode)	[V]
$U_{i\_lade\_ist}$	Spannungsäquivalent des Ladestroms	[V]
$U_{i\_lade\_soll}$	Spannungsvorgabe für den Ladestrom	[V]
U_{int}	Ausgangsspannung des integrierenden Reglers	[V]
U_k	Kathodenspannung (Triode)	[V]
U_{KT}	Eingangsspannung der Koronatriode	[V]
U_{lade}	Ladespannung	[V]
U_{st}	Steuerspannung (Triode)	[V]
U_t	Spannung am Terminal	[V]
U_{thr}	Schwellenspannung im Lade- und Entladesystem	[V]
U_{thr_kt}	Schwellenspannung der Koronatriode	[V]
$U_{u\_lade\_soll}$	Spannungsvorgabe für Ladespannung	[V]
v	Verstärkungsfaktor	
v_{int}	Integrationskonstante des integrierenden Reglers	
v_{rausch}	Verstärkungsfaktor für Störstrom	
z	komplexe Variable im $\mathcal{Z}$-Bereich	
Z	Störgröße im Regelkreis	
$\underline{Z}$	komplexe Zahl; komplexer Widerstand	
$\mathcal{Z}$	Bildbereich der Z-Transformation	
γ	Verhältnis zwischen auf- und abfließendem Strom im Terminal	
$\lambda(x)$	Linienladung	[C]
$\lambda(x)_n$	Linienladung nach der Berechnung	[C]
$\lambda(x)_v$	Linienladung vor der Berechnung	[C]
τ	Zeitkonstante	[s]
$\varphi_{F_{KT}}$	Phasenwinkel der Übertragungsfunktion der Koronatriode	
φ_{F_B}	Phasenwinkel der Übertragungsfunktion des Bandes	
ω	Kreisfrequenz	[Hz]

1 Funktionsanalyse und Modellbildung

Für eine exakte Verhaltensbeschreibung des Van-de-Graaff-Ionenbeschleunigers durch ein Modell ist es notwendig, nicht nur die allgemeine Funktionsweise sondern auch die speziellen Eigenschaften des Beschleunigers, welcher im Forschungszentrum Rossendorf (FZR) betrieben wird, zu kennen.
Im ersten Teil der Arbeit soll der Aufbau und die Wirkungsweise dieses Systems analysiert und gleichzeitig die Modellbildung der einzelnen Teilsysteme erläutert werden.

1.1 Van-de-Graaff-Ionenbeschleuniger

Ein Van-de-Graaff-Beschleuniger ist ein Teilchenbeschleuniger, dessen Spannungserzeuger nach dem Prinzip des Van-de-Graaff-Generators arbeitet und nach seinem gleichnamigen Entwickler benannt ist.
Der Grundgedanke eines solchen Generators liegt darin, dass zur Erzeugung der Beschleunigungsspannung ein mechanisch angetriebenes, elektrisch schwach leitendes Endlosband genutzt wird, auf welches durch eine Beladeeinheit positive Ladungsträger aufgebracht und zu einem Hochspannungsterminal befördert werden (Bild 1.1) [1], [2]. Dort entlädt sich das Band an einer Entladeeinheit und die transportierten Ladungsträger fließen auf eine Terminalelektrode ab. Dadurch entsteht an dieser ein positives Potential gegen Masse, welches am Beschleuniger im Rossendorfer Forschungszentrum bis zu zwei Megavolt betragen kann. Diese Hochspannung wird über eine Widerstandskette entlang eines Ionenstrahlrohres so aufgeteilt, dass die Teilspannungen auf den dazwischengeschalteten Potentialringen (in

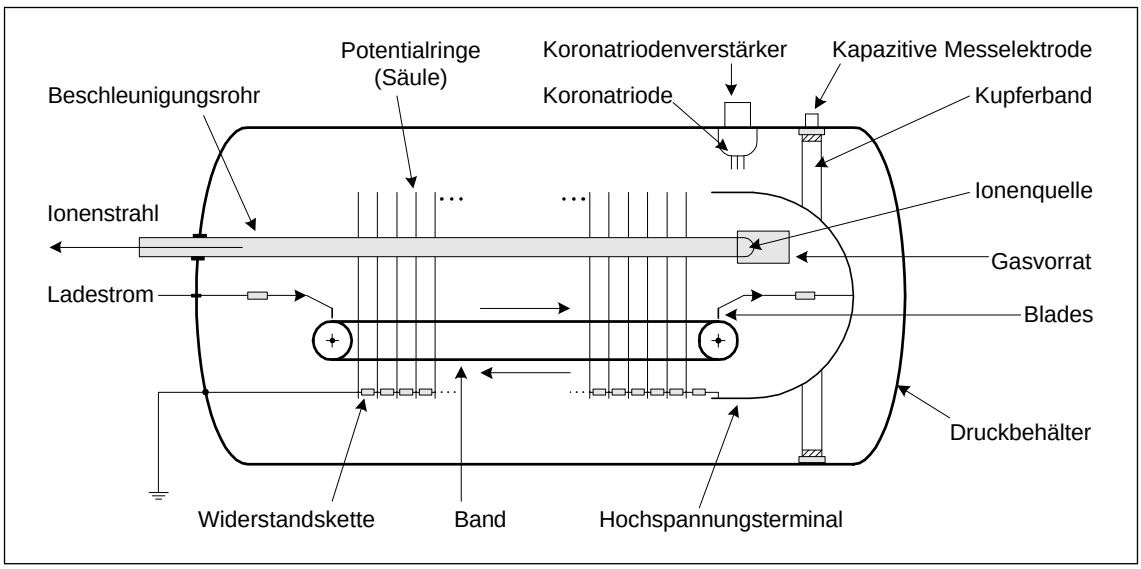


Bild 1.1: Schematischer Aufbau des Beschleunigers (nach [2])

ihrer Gesamtheit Säule genannt) ein elektrisches Feld aufbauen, welches eine gleichmäßige Beschleunigung der Ionen gewährleistet. Die Ionen werden von einer separaten Quelle emittiert, im Bereich der Säule beschleunigt und gelangen durch ein mehrere Meter langes Rohrsystem zum so genannten 'Target', welches mit diesen Ionen beschossen wird. Auf dem Weg dahin erfährt der Ionenstrahl mehrere Beeinflussungen durch verschiedene Regelsysteme, welche die strahlenoptischen Eigenschaften verändern und korrigieren.
Die maximale Höhe der Terminalspannung wird in erster Linie durch die auftretende elektrische Feldstärke und die damit in Zusammenhang stehende Durchschlagfestigkeit des umgebenden Mediums begrenzt. Aus diesem Grund befindet sich das Generatorsystem in einem mit Gas befüllten Druckbehälter. Abhängig vom verwendeten Gas und vom herrschenden Druck kann die Durchschlagsfestigkeit vom Terminal in Richtung des Druckbehälters wesentlich vergrößert werden. Damit ergibt sich auch ein Zusammenhang zwischen der Hochspannung und dem auf das Band geleiteten Ladestrom, welcher nicht größer gewählt werden darf, als zum Erreichen der maximalen Terminalspannung nötig ist.

1.2 Modellentwurf mittels MATLAB und SIMULINK

Das Verhaltensmodell des Beschleunigers wurde mit der Software MATLAB und der implementierten Simulationsumgebung SIMULINK erstellt.
MATLAB ist ein numerisches Berechnungsprogramm, mit dessen Hilfe es möglich ist, durch einfache Befehle oder mit selbstprogrammierten Funktionen komplexe Berechnungen durchzuführen. Des weiteren steht eine Simulationsumgebung mit grafischer Oberfläche zur Verfügung, mit der man durch Blockschaltbilder und Signalpfade Verhaltensbeschreibungen erstellen und simulieren kann. Ein großer Vorteil ist, dass es eine Interaktion zwischen MATLAB und SIMULINK gibt, sodass ein bestehendes Modell von MATLAB aus parametrier- und steuerbar ist. Das vereinfacht zum Beispiel die mehrfache Simulation des Modells mit verschiedenen Parametersätzen und die mathematische oder auch grafische Auswertung der erhaltenen Ergebnisdaten, die in Form von Vektoren oder Matrizen vorliegen.
Da der zu entwerfende Algorithmus zur Bestimmung der Betriebsparameter später von einem Mikrocontroller abgearbeitet wird, wurde es für sinnvoll erachtet, das Modell, welches diese Funktionalität beinhalten soll, auch in zeitdiskreten Schritten arbeiten zu lassen. Dafür stehen in den Blockbibliotheken von SIMULINK, den so genannten Blocksets, diskrete Funktionsblöcke zur Verfügung, welche mit der festgelegten Simulationsschrittweite arbeiten. Die für das entworfene Modell gewählte Taktzeit beträgt 1 ms. Dieses Zeitintervall wurde als kleinste Zeiteinheit aus der Bandumlaufzeit von $1/f_{Band} \approx 175\,\text{ms}$ abgeleitet (siehe auch Abschnitt 1.5). Da das Modell erstellt wurde, um mit Hilfe von Simulationsrechnungen neue Erkenntnisse gewinnen zu können und es demzufolge sehr oft mit unterschiedlichen Parametersätzen analysiert werden soll, ist es wichtig einen Kompromiss

zwischen der benötigten Zeitauflösung bzw. Genauigkeit und dem daraus resultierenden Rechenaufwand zu finden. Mit ersten Simulationen einfacher Teilsysteme zu Beginn der Arbeit wurde die Taktzeit von 1 ms als oben benannter Kompromiss bestätigt. Als Rechner für die durchzuführenden Simulationen wurde ein handelsüblicher Personal Computer verwendet (Prozessor: AMD Athlon XP 2400+; Arbeitsspeicher: 512 MByte).
Um das in der Realität vorkommende zeitkontinuierliche Übertragungsverhalten verschiedener Einzelsysteme in einem zeitdiskret simulierten Modell nachbilden zu können, mussten die entsprechenden Übertragungsfunktionen in den $\mathcal{Z}$-Bereich transformiert werden. Dies geschah mit Hilfe der bilinearen Transformation, welche im Anhang A erläutert wird. In diesem Abschnitt sind auch alle anderen im Modell genutzten Übertragungsfunktionen mit ihrer entsprechenden Transformierten aufgeführt.
Da MATLAB um sehr viele Funktionsbibliotheken (Toolboxen) und SIMULINK um viele Blocksets erweitert werden kann, darf nicht davon ausgegangen werden, dass auf jedem Rechnersystem die gleichen Funktionen vorhanden sind und genutzt werden können. Aus diesem Grund wurde auf spezielle Funktionen und SIMULINK-Blöcke aus gesonderten Bibliotheken verzichtet. Alle Systeme wurden aus grundlegenden Funktionen aufgebaut, die in jeder Basisversion von MATLAB/SIMULINK zur Verfügung stehen. Damit wird erreicht, dass dieses Modell flexibel und unabhängig auf verschiedenen Plattformen verwendbar ist.
Ein erster Schritt zur detaillierten Verhaltensbeschreibung ist die Unterteilung des Gesamtsystems in eine Hierarchie von Haupt- und Untersystemen. Bild 1.2 (auch Bild B.1 in Anhang B) zeigt eine grobe Gliederung auf oberster Ebene. Unterteilt ist das Gesamtsystem in ein Ladestromregelsystem, ein Generatorsystem, in dem sich das Band befindet, in das Hochspannungsterminal und in einen Block, in welchem mehrere Regelsysteme vereint sind.

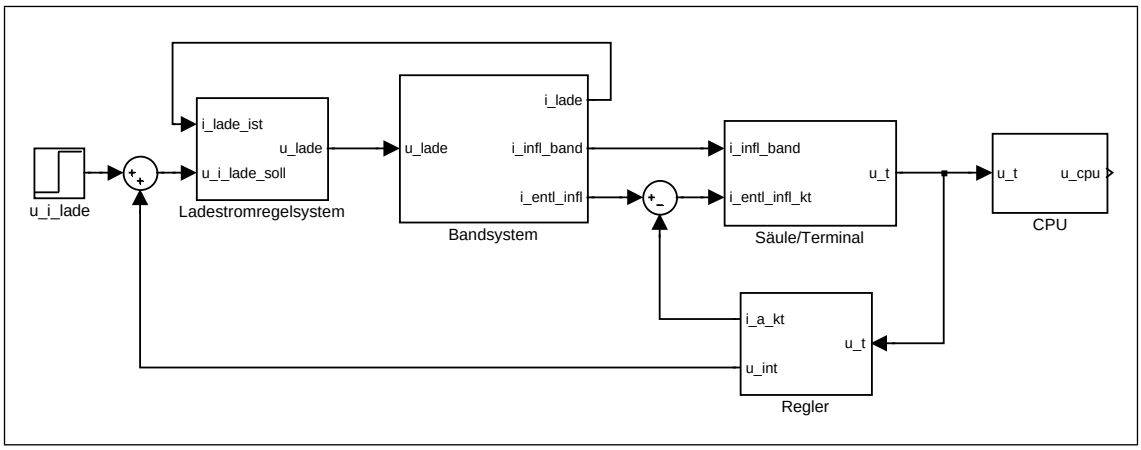


Bild 1.2: Oberste Gliederungsebene des Gesamtsystems in SIMULINK

1.3 Ladestromregelsystem

Das Potential des Terminals und damit auch die Qualität des Ionenstrahls hängt stark von der Stabilität des Ladestroms ab. Abweichungen des Ladestroms von seinem Sollwert durch Unregelmäßigkeiten und Störungen wirken sich direkt auf den Ionenstrahl aus. Deswegen ist es erforderlich, den Ladestrom durch eine Regelung zu stabilisieren und Störeinflüsse auszugleichen. Wesentliche Störungen sind zum einen die schleichende Verschiebung des Arbeitspunktes in Abhängigkeit von der Umgebungstemperatur und vom Gasdruck im Behälter und zum anderen die unregelmäßige Struktur des Bandes und die damit in Verbindung stehenden sofortigen Rückwirkungen auf den Lade- bzw. Entladestrom im Terminal.

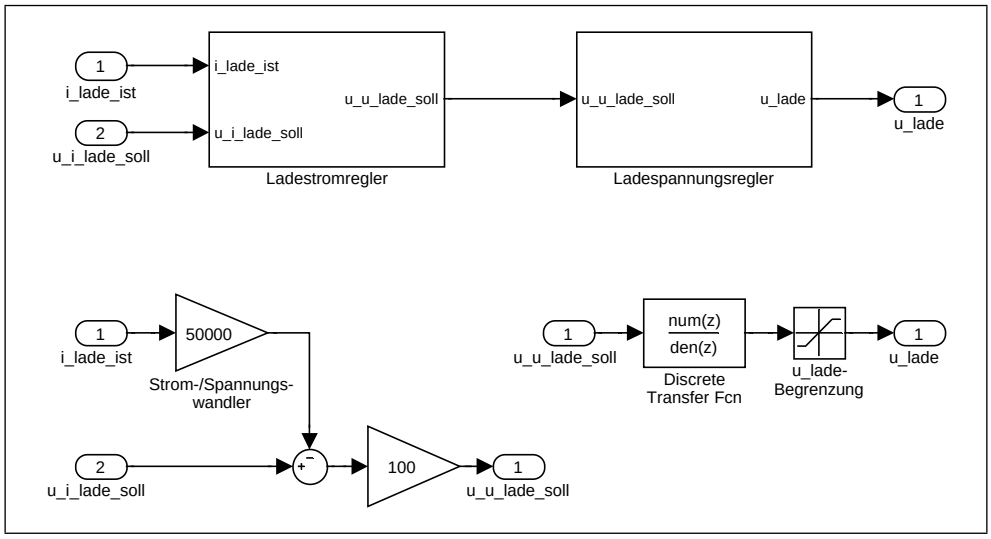


Bild 1.3: Unterteilung des Ladestromregelsystems

Das Ladestromregelsystem besteht aus einem Ladestromregler und einem Ladespannungsregler (Bild 1.3). Der Ladestromregler hat die Aufgabe, den rückgekoppelten Ladestrom i_lade_ist durch einen Widerstand von 50 kΩ in eine Spannung u_i_lade_ist zu wandeln und mit dem Vorgabewert zu vergleichen, welcher durch die Spannung u_i_lade_soll repräsentiert wird. Die Differenz zwischen den beiden Werten wird mit dem Faktor 100 verstärkt und an den Ladespannungsregler weitergeleitet (u_u_lade_soll). Dieser besitzt ein proportionales Übertragungsverhalten mit Verzögerung 1. Ordnung von 20 ms und vervielfacht die Spannungsvorgabe um das Zweitausendfache. Im Modell ist dies mit dem Block Discrete Transfer Fcn gekennzeichnet (siehe Abschnitt A Gleichung A.2). Ein Begrenzer sorgt dafür, dass die Ladespannung nicht über 20 kV ansteigt. Die geregelte Ladespannung u_lade wird weitergeleitet zum Ladesystem, in welchem der eigentliche Ladestrom erzeugt und auf das Band gebracht wird.

1.4 Belade- und Entladesystem

Die Untergruppen des Belade- und Entladesystems befinden sich an Positionen innerhalb des Beschleunigers, welche etwa eine halbe Bandlänge voneinander entfernt sind, und spiegeln im Modell hauptsächlich die physikalischen Vorgänge an der Stelle wider, an welcher der Ladestrom auf das Band geleitet und der Entladestrom wieder vom Band abgenommen wird. Der Übergang zum Band wird mittels sehr dünner Ladeplättchen (Blades) realisiert, über die der Strom fließt. Während des Betriebs des Generators 'schweben' die Blades in geringem Abstand über dem Band, das heißt, sie berühren es im Idealfall nicht. Durch Unebenheiten im Band geschieht dies aber dennoch in Einzelfällen und verursacht kurzzeitige Störungen im Lade- und Entladestrom.
Die physikalischen Vorgänge an den Übergangsstellen werden durch vereinfachte aber für die Modellierung ausreichende Gleichungen beschrieben [3]. Da das Prinzip der Blades auf beiden Seiten angewandt wird, ähneln sich die Gleichungen.

$$I_{lade} = \quad k_u(U_{lade} - U_{thr} - I_{lade} \cdot R_{i\_lade}) \quad - k_{ia}(I_{b\_auf\_a} + I_{b\_ab\_a}) - k_{ii} \cdot I_{bi} \qquad (1.1)$$

$$I_{entl} = \qquad -k_u(U_{thr} + I_{entl} \cdot R_{i\_entl}) \qquad + k_{ia}(I_{t\_auf\_a} + I_{t\_ab\_a}) + k_{ii} \cdot I_{ti} \qquad (1.2)$$

Gleichung 1.1 stellt den Sachverhalt dar, dass die vom Ladespannungsregler kommende Spannung (U_{lade}) zunächst den Wert einer Schwellenspannung (U_{thr}) übersteigen muss, bevor ein Strom auf das Band fließen kann. Zusätzlich wird der Wert einer Spannung subtrahiert, welche sich infolge des durch den Innenwiderstand fließenden Ladestroms ($I_{lade} \cdot R_{i\_lade}$) aufbaut. Der Multiplikator k_u wird als Spannungskoeffizient bezeichnet und besitzt die Einheit A/V. Erreicht die Ladespannung einen ausreichend großen Wert, fließen Ladungsträger ($I_{b\_auf\_a}$) auf das Band. Diese bauen sogleich ein elektrisches Feld auf, welches den nachfolgenden Ladungsträgern (I_{lade}) bis zum Erreichen eines dynamischen Gleichgewichts entgegenwirkt, und werden weiter zum Terminal hin bewegt. Da an der Entladestelle nicht alle Ladungsträger vom Band abgeleitet werden, transportiert sie das Band wieder mit zurück zur Beladestelle (Basis). Dort wirken diese Ladungen ($I_{b\_ab\_a}$) erneut dem Ladestrom entgegen. Die Indizes der angegebenen Größen geben den Ort und die Richtung der Wirkung an. So steht der Index b für den Ort der Basis (Beladung) und der Index t für das Terminal, in dem sich die Entladestelle befindet. Die Bezeichnungen *auf* und *ab* geben die Richtungen der Ladungsträger, die durch das Band bewegt werden, an. Entweder sie werden zum Terminal hinauf oder zur Basis hinab transportiert. Trotz dem das Band aus einem Material mit hohem spezifischen Widerstand besteht, verteilen sich die Ladungsträger auf der Bandoberfläche und zusätzlich auch in das Bandinnere hinein. Aus diesem Grund setzt sich die oben genannte Gegenwirkung aus dem elektrischen Feld, welches auf der Bandoberfläche entsteht (k_{ia}), und aus dem elektrischen Feld, welches im Bandinneren ensteht (k_{ii}), zusammen. Daraus folgt die dritte Indizierung: a bezeichnet die

Bandaußenseite und i das Bandinnere. Der so genannte Stromkoeffizient k_{ia} bzw. k_{ii} ist einheitenlos und gibt die Stärke der Gegenwirkung an. In Bild 1.4 sind die Bezeichnungen der Ströme nochmals grafisch dargestellt.

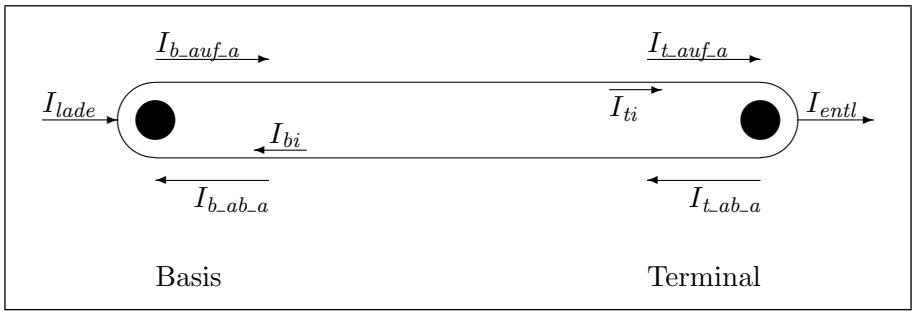


Bild 1.4: Indizierung der Ströme auf dem Band

Gemäß dem Knotenpunktsatz für Ströme können die folgenden mathematischen Beziehungen aufgestellt werden:

$$I_{b\_auf\_a} = I_{b\_ab\_a} + I_{lade} \qquad \text{und} \tag{1.3}$$

$$I_{t\_ab\_a} = I_{t\_auf\_a} - I_{entl}\,. \tag{1.4}$$

Als Verhältnis zwischen dem am Terminal ankommenden und dem nach der Entladestelle auf dem Band verbleibenden Strom wird die Größe γ mit

$$\gamma = \frac{I_{t\_auf\_a}}{I_{t\_ab\_a}} \tag{1.5}$$

eingeführt.
Der Entladevorgang am Terminal wird mit Gleichung 1.2 beschrieben. Der hier ankommende Strom $I_{t\_auf\_a}$ muss so groß sein, dass er die bestehende Schwellenspannung U_{thr} und die entgegenwirkende Spannung, welche über dem Innenwiderstand ($I_{entl} \cdot R_{i\_entl}$) abfällt, übersteigt. Im Gegenteil zur Basis wirkt sich das elektrische Feld der Ladungsträger auf dem Band begünstigend aus und treibt den Entladestrom an. Auch hier wird diese Wirkung auf die der Bandoberfläche (k_{ia}) und auf die des Bandinneren (k_{ii}) aufgeteilt.

Verhaltensbeschreibung mittels SIMULINK

Für den Modellentwurf wird die Ladestromgleichung 1.1 mit Hilfe von 1.3 nach I_{lade} aufgelöst und es ergibt sich eine mathematische Beziehung, welche sich relativ einfach mit SIMULINK nachbilden lässt:

$$I_{lade} = \frac{k_u(U_{lade} - U_{thr}) - 2k_{ia} \cdot I_{b\_ab\_a} - k_{ii} \cdot I_{bi}}{1 + k_u \cdot R_{i\_lade} + k_{ia}}\,. \tag{1.6}$$

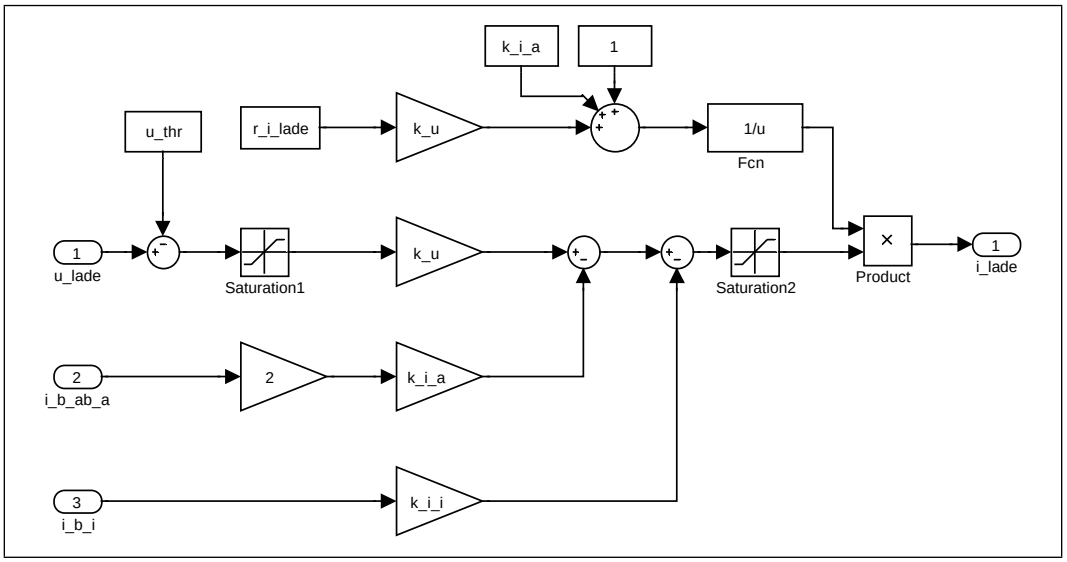


Bild 1.5: Modellierung der physikalische Vorgänge an der Beladestelle

Es entsteht eine rückkopplungsfreie Struktur, wie sie in Bild 1.5 dargestellt ist. Hinzugekommen sind lediglich zwei Saturation-Blöcke, die sicherstellen, dass wirklich nur dann ein positiver Strom fließen kann, wenn die Ladespannung die Schwellenspannung und die Gegenwirkung vom Band überwunden hat. Gleichung 1.2 wurde nach dem Entladestrom I_{entl} aufgelöst:

$$I_{entl} = \frac{2k_{ia} \cdot I_{t\_auf\_a} - k_u \cdot U_{thr} + k_{ii} \cdot I_{ti}}{1 + k_u \cdot R_{i\_entl} + k_{ia}} \tag{1.7}$$

und ergibt eine ähnliche Struktur (Bild 1.6) wie die vorherige.

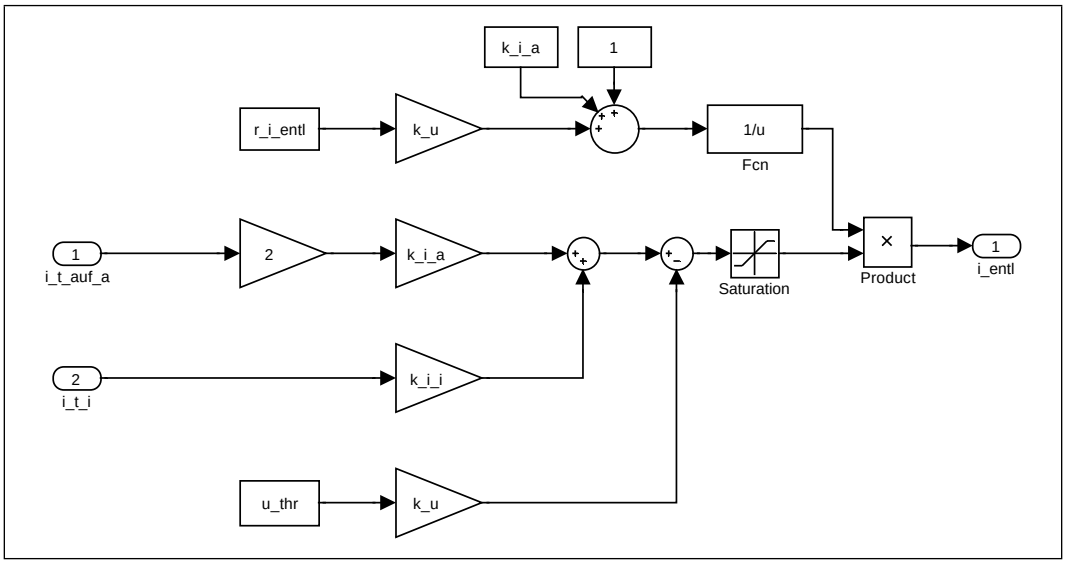


Bild 1.6: Modellierung der physikalische Vorgänge an der Entladestelle

1.5 Band

Das umlaufende Generatorband aus elektrisch schwach leitendem Material mit einer Länge von 3,05 m befördert die positiven Ladungsträger mit einer Geschwindigkeit von $17,4\,\frac{\mathrm{m}}{\mathrm{s}}$ vom Ladesystem zur Entladestelle. Die daraus resultierende Bandumlauffrequenz beträgt 5,7 Hz.

Die unregelmäßige Bandstruktur und insbesondere der Klebestoß sowie die mechanischen Schwingungen des Bandes und die Eigenbewegung der darauf befindlichen Ladungsträger tragen zu einem großen Teil zur Modulation des Ladestroms auf dem Band bei. Die dadurch verursachten kurzzeitigen Störungen müssen bei der Betrachtung des Ladungsträgertransports unbedingt berücksichtigt werden. Bild 1.7 zeigt einen kurzen Ausschnitt aus einer Messung des Ladestroms bei konstanter Spannungsvorgabe im Ladestromregler. Es sind deutlich die periodisch wiederkehrenden Störeinflüsse zu erkennen.

Ein anderer zu beachtender Effekt ist der, dass die Ladungsträger sich in das Bandinnere bewegen und somit einerseits nicht mehr für den Entladevorgang zur Verfügung stehen jedoch andererseits von dem Inneren des Bandes her die Einspeisung des Ladestroms und die Ableitung an der Entladestelle beeinflussen (siehe auch Stromkoeffizient k_{ii} in Gleichung 1.1 und 1.2). In Bild 1.7 ist dieser Vorgang anhand des oberen Diagramms deutlich zu erkennen. Die Langzeitmessung des Ladestroms wurde mit offener Regelschleife durch-

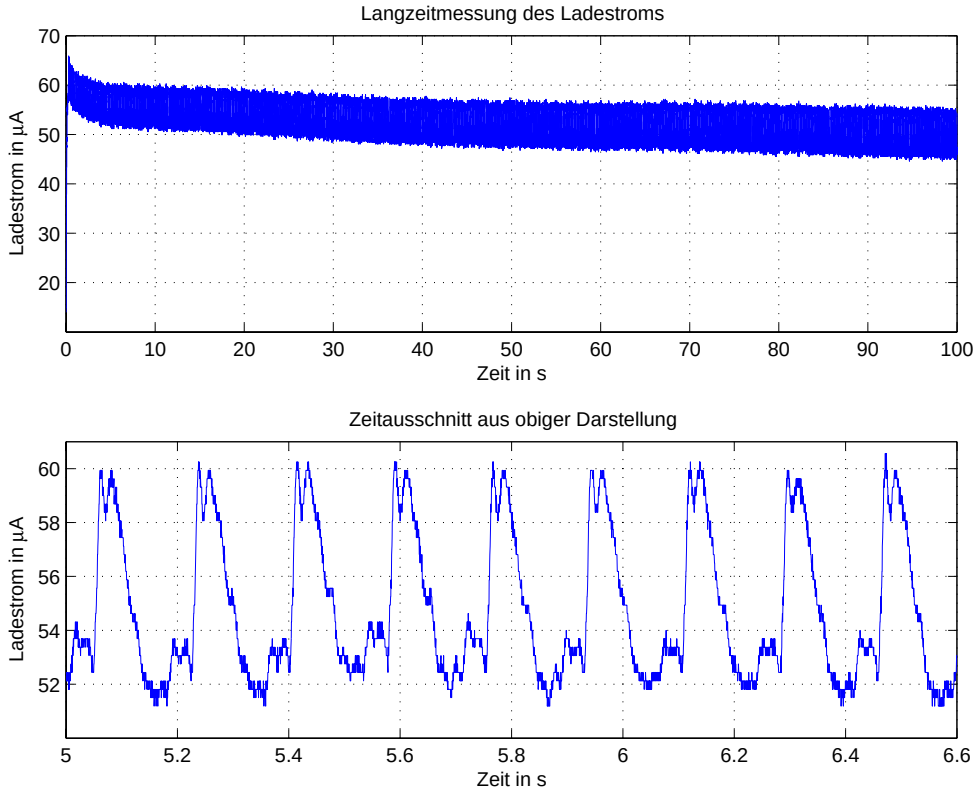


Bild 1.7: Messung des Ladestroms auf dem Band nach dem Einschalten

geführt, um diesen Effekt sichtbar zu machen. Nachdem sich der Strom auf der Bandoberfläche zwischen Lade- und Entladestelle eingeschwungen hat, driften die Ladungsträger in das Innere und wirken dem Ladestrom entgegen, der nicht nachgesteuert wird und sich damit immer weiter verringert.
Um diese Eigenschaften des Bandes zu modellieren, wurde es in mehrere Segmente unterteilt, welche alle die gleichen Merkmale aufweisen. Bei der oben angegebenen Geschwindigkeit und der Länge des Bandes ergibt sich eine Umlaufzeit von ca. 175 ms. Wird das Band in 35 gleich lange Abschnitte geteilt, so ergibt sich eine Laufzeit pro Bandsegment von genau 5 ms, was einer Abschnittslänge von 8,71 cm entspricht. Es wurde festgestellt, dass die Genauigkeit der Simulation im Verhältnis zur benötigten Rechenzeit bei dieser Gliederung ausreichend ist. Die Festlegung der Bandeinteilung basiert darüber hinaus auf dem Zusammenhang, dass der entsprechende Zeitabschnitt ein ganzzahliges Vielfaches der vorher festgelegten Simulationsschrittweite von einer Millisekunde ist. Dadurch wird erreicht, dass alle Bandsegmente identische Eigenschaften aufweisen. Eventuell auftretende numerische Störeffekte durch sich ständig ändernde Berechnungszeitpunkte während der Simulation können dadurch ausgeschlossen werden. Die Struktur, die das Verhalten eines solchen Bandsegments nachbilden soll, ist in Bild 1.8 dargestellt.

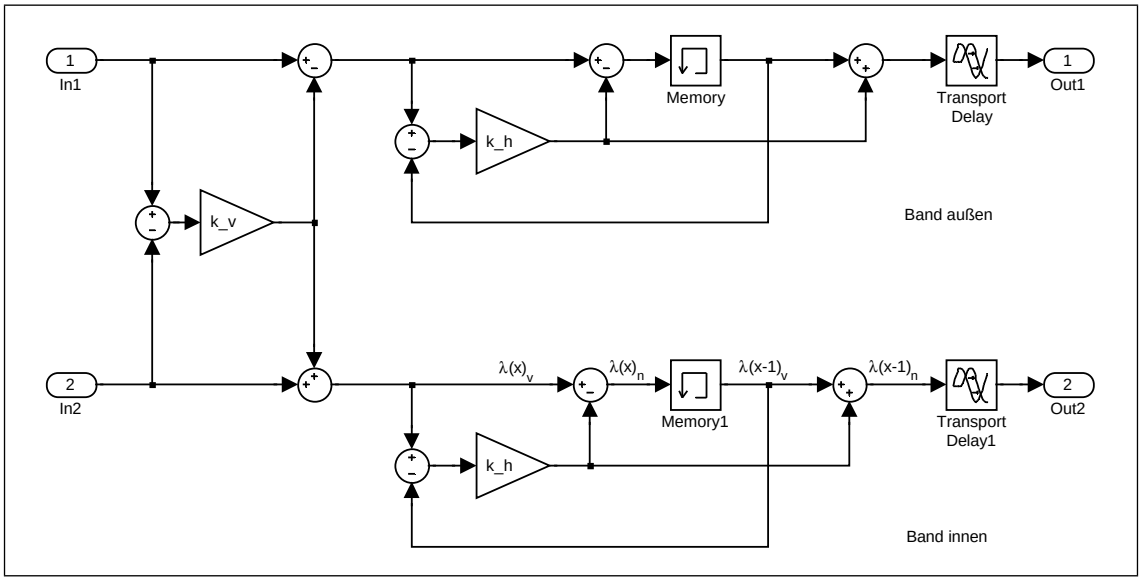


Bild 1.8: SIMULINK-Struktur des Bandsegments

Am Eingang dieses Subsystems wird die vertikale Bewegung der Ladungsträger in das Band modelliert. Dazu wird die Differenz der Ladungen zwischen der Oberfläche und dem Bandinneren gebildet und anschließend mit negiertem Vorzeichen den Eingangswerten jeweils wieder zugeführt. Überwiegen zum Beispiel die Ladungen auf der Oberfläche, so wird die Differenz von dieser subtrahiert und zum Bandinneren addiert. Mit dem Faktor k_v wird die Stärke dieses Effektes angepasst.

Der Gedanke, welcher zu dem erläuterten Prinzip führt, ist der, dass das Band aus zwei Schichten besteht, zwischen denen sich die Ladungsträger bewegen. Abhängig von ihrer Konzentration können sich die Ladungsträger von der Außenseite in das Innere des Bandes oder auch in umgekehrter Richtung verteilen, bis das Bandinnere im zeitlichen Mittel feldfrei ist. Das bedeutet, dass mit einem konstant aufgebrachten Ladestrom nach einiger Zeit ein Zustand erreicht wird, in dem die Stärke des Stroms im Inneren des Bandes gleich der Stärke des Stroms auf der Außenseite ist. Die Zeitdauer bis zum Erreichen dieses Zustandes ist von der Vorbeladung des Bandes abhängig und kann im Falle eines unbeladenen Bandes nach dem Einschalten des Ladestromes mehrere Minuten dauern (siehe Bild 1.7). Dadurch ist auch bedingt, dass kurzzeitige Schwankungen und Störungen des Ladestroms keinen Einfluss auf diesen Vorgang haben. Über den zeitlichen Verlauf der Entladung des Bandes können keine Angaben gemacht werden, da diese im Gegensatz zur Beladung passiv geschieht. Nach dem Abschalten des Ladestromes können nach kurzer Zeit auch keine Ladungsträger mehr von der Bandoberfläche über das Entladesystem zum Terminal fließen. Das bedeutet, dass ein Restpotential auf und im Band verbleibt, welches sich theoretisch nur über die Antriebsrollen des Bandes zur Masse hin entladen kann.
Der zweite Teil des in Bild 1.8 dargestellten Subsystems bildet die horizontale Bewegung bzw. die Diffusion der Ladungsträger zwischen verschiedenen Ladungsdichten auf dem Band nach. Dabei wird zur Vereinfachung des Sachverhalts von einer Linienladung ausgegangen. Während der Simulation wird zwischen zwei aufeinander folgenden Ladungen $\lambda(x)$ und $\lambda(x-1)$ wiederum eine Differenz gebildet, die von der 'aktuellen' Ladung subtrahiert und zur vorhergehenden addiert wird. Die mathematische Beschreibung lautet

$$\lambda(x)_n = \quad \lambda(x)_v \quad - k_h(\,\lambda(x)_v - \lambda(x-1)_v) \tag{1.8}$$

$$\lambda(x-1)_n = \quad \lambda(x-1)_v \quad + k_h(\,\lambda(x)_v - \lambda(x-1)_v)\,, \tag{1.9}$$

wobei $\lambda(x)_v$ den Wert der Ladung am Ort x vor der Berechnung und $\lambda(x)_n$ den Wert nach der Berechnung angibt. Die Differenz der Ladungen wird mit einem Faktor k_h belegt, welcher die Intensität für diesen Vorgang festlegt. In Bild 1.9 ist die Simulation des separierten Subsystems zu sehen. Es wurde nur das Band ohne alle anderen Systeme simuliert. Die Faktoren k_v und k_h sind hier so übertrieben groß gewählt worden, dass die Effekte in der simulierten Zeit von 50 s sichtbar werden. Als Anregung wurde ein Rechteckimpuls gewählt, mit dem sich besonders die Diffusion der Ladungsträger auf der Bandoberfläche (Flankenverschleifung) simulieren lässt. Die blaue Kurve stellt den Strom auf der Bandoberfläche und die rote Kurve den des Bandinneren dar. Es ist deutlich ein Ausgleich zwischen beiden zu beobachten. Als eher lang andauernder Vorgang ist die horizontale Verteilung zu sehen. Aus dem Rechteckimpuls wird ein sinusähnliches Signal, bis es sich im weiteren Verlauf der Simulation weiter verflacht und sich einem konstanten Wert angleicht, d. h. die Ladungsdichte auf dem Band ist überall gleich und konstant.

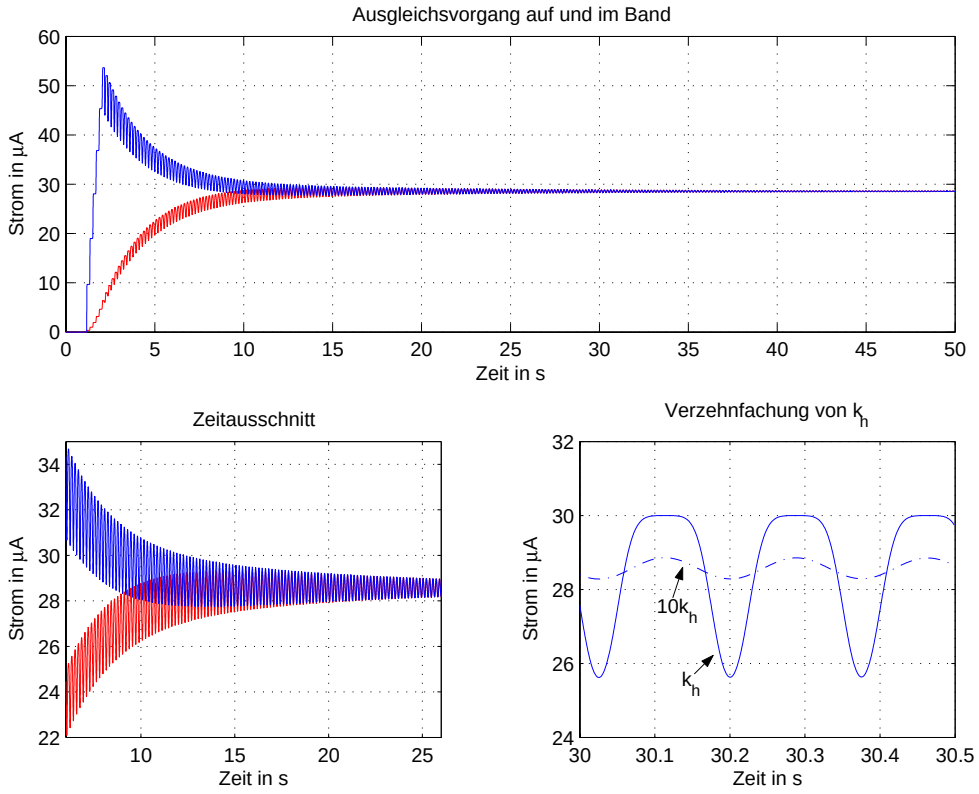


Bild 1.9: Simulation der vertikalen und horizontalen Ladungsträgerverteilung

Eine eingesetzte Verzögerung (Memory) um einen Simulationstaktschritt ist für die Bereitstellung der Vorgängerladung notwendig, bevor die Differenz mit der 'aktuellen' Ladung gebildet und jeweils vorzeichennegiert den beiden Ladungen aufsummiert wird. Die anschließende Zeitverzögerung im Block Transport Delay beträgt 4 ms und nicht wie oben aufgeführt 5 ms, was durch den Verzug um einen Taktschritt, d. h. 1 ms, im Memory-Block bedingt ist. Damit wird erreicht, dass sich bei einer Anzahl von 35 gleichartigen Bandabschnitten eine Gesamtverzögerung von 175 ms ergibt, was genau einer Bandumlaufzeit entspricht. Von den 35 in einer Ringstruktur angeordneten Bandsegmenten (Bild 1.11 bzw.

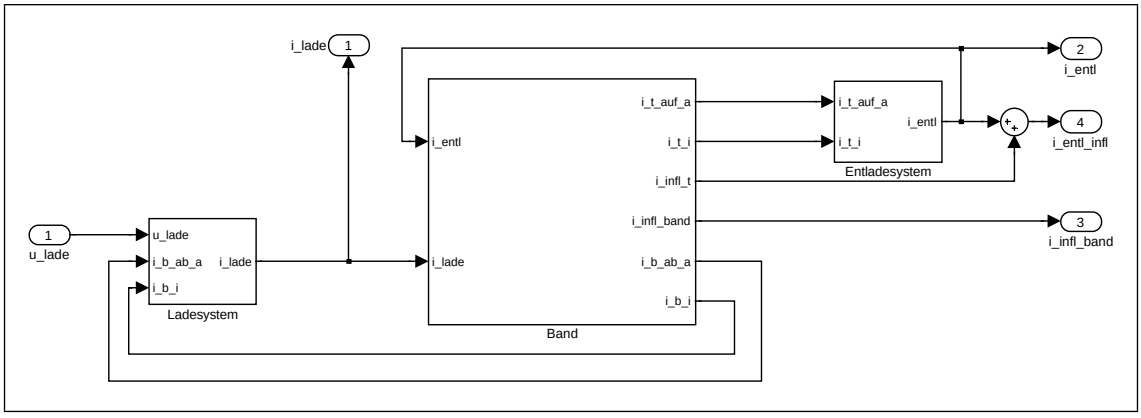


Bild 1.10: Signalverbindungen des Bandsystems

Bild B.3 im Anhang B) befinden sich 14 zwischen der Be- und Entladestelle. Sechs Segmente (52,25 cm) bilden den Abschnitt nach, in dem sich das Band durch die Hochspannungselektrode bewegt. Die verbleibenden 15 Teile liegen in Richtung des Stromflusses gesehen zwischen der Ent- und Beladestelle. Hervorzuheben sind die Abzweigungen der verschiedenen Ströme zwischen den Segmenten, besonders die zur Modellierung der Gegenwirkung im Ladesystem benötigten. Die Signalverläufe i_infl_band(x) werden in Abschnitt 1.6 erläutert. Bild 1.10 bzw. Bild B.2 im Anhang B veranschaulicht die Position des Bandsubsystems im Modell und die Signalverläufe zu anderen Einheiten.

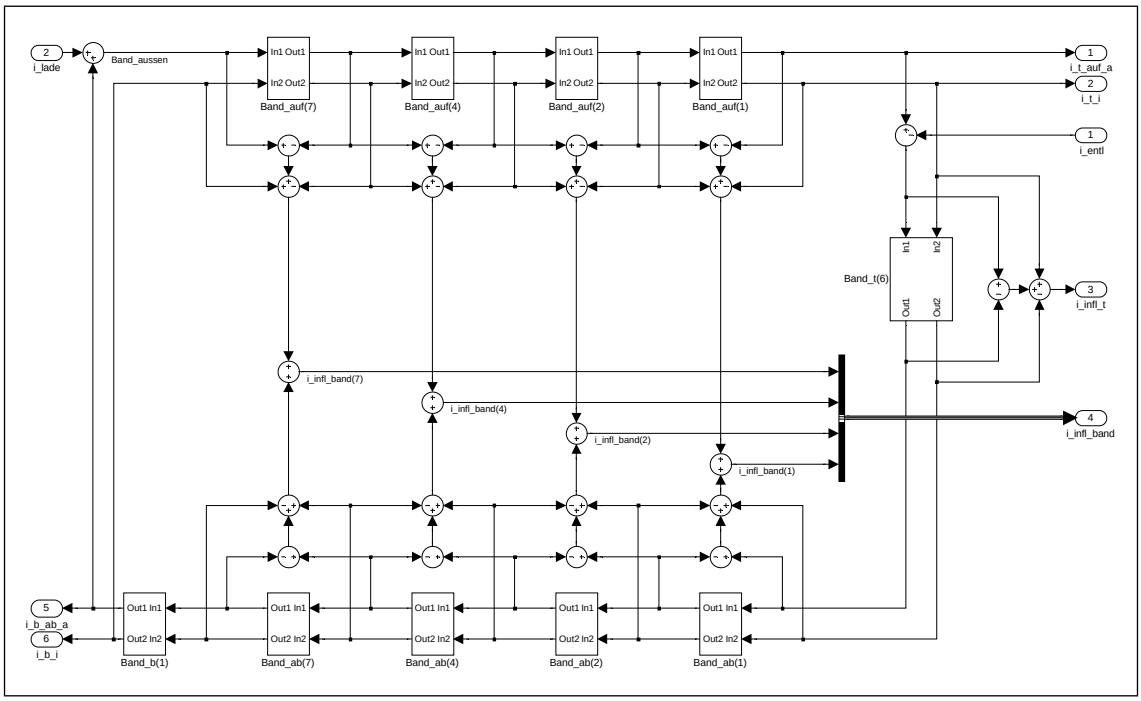


Bild 1.11: Anordnung und Verteilung der Bandsegmente

1.6 Säule

Wie bereits in Abschnitt 1.1 erwähnt, ist das Rohr, in dem die Ionen beschleunigt werden, von einer Widerstandskette umgeben, welche die im Terminal erzeugte Hochspannung so aufteilt, dass die Ionen durch das entstehende elektrische Feld gleichmäßig beschleunigt werden. Dazu sind im Abstand weniger Zentimeter Metallringe zu einer zylinderförmigen Säule nacheinander angebracht, in welcher sich sowohl das Beschleunigungsrohr als auch das Band befinden. Zwei benachbarte Ringe sind nur durch einen Widerstand miteinander verbunden, bilden aber auch Kapazitäten zwischeneinander und jeweils einzeln zur Druckbehälterwand – d. h. zur Masse hin – aus, deren Größen bestimmt sind von der gegebenen Ringgeometrie. Ein Ersatzschaltbild dieser Beschreibung ist in Bild 1.12 dargestellt.
Jeder Ring ist wiederum mit einem Teil des ebenfalls segmentierten Beschleunigerrohres elektrisch kontaktiert. Dadurch wird erreicht, dass sich in jeder Ringebene bzw. in jedem

Rohrsegment ein homogenes elektrisches Feld aufbaut und sich durch die Widerstandsstufung gleichmäßig entlang des Rohres verteilt.
Mit Hilfe von Messungen der Spannung U_{cpu} an der kapazitiven Messelektrode (siehe Abschnitt 1.7) und der Spannung u_i_lade_ist im Ladestromregelsystem ließ sich der Frequenzgang des Generators bestimmen. Es waren einzelne Maxima zu erkennen, die jeweils genau zwischen den Frequenzen des Bandumlaufs lagen und nicht sofort erklärbar waren (Bild 1.23 im Abschnitt 1.11). Durch eine detailliertere Verhaltensbeschreibung der Säule sollte dieser Effekt mit Hilfe des Modells nachgebildet und verifiziert werden. Es wurde angenommen, dass der Strom, den das Band transportiert, durch seine influenzierende Wirkung Verschiebungsströme auf die eben beschriebene Ringstruktur einfließen lässt. Ein Verschiebungsstrom stellt eine Veranschaulichung der Tatsache dar, dass ein sich zeitlich änderndes elektrisches Feld eine Änderung des elektrischen Flusses durch die Fläche, auf welche das elektrische Feld einwirkt, zur Folge hat. Dabei werden von diesem Strom keine Ladungen transportiert, er hat aber die gleiche Wirkung wie ein konduktiver Strom. Er verursacht in Abhängigkeit von seiner Frequenz eine Potentialverschiebung auf der Säule und somit rückwirkend auch auf dem Terminal. Ist die Summe der einzelnen Influenzströme groß, bewirkt dies eine Anhebung der Spannung am Terminal. Ein Maximum erreicht der Verschiebungsstrom bei der halben Bandumlauffrequenz, da hier genau eine Halbwelle des bei der Frequenzgangmessung sinusförmig aufgebrachten Ladestroms über die gesamte Bandlänge moduliert wird. Bei einer Sinusschwingung, die dem ganzzahligen Vielfachen der Bandumlauffrequenz entspricht, heben sich die Auswirkungen der beiden entgegengesetzten Halbwellen auf und der beschriebene Effekt ist nicht zu beobachten.

Verhaltensbeschreibung mittels SIMULINK

Der erste Ansatz zur Modellierung der Säule und des Influenzstroms ist die Einteilung des Bandes in mehrere Abschnitte, von denen die jeweilige influenzierende Wirkung quantitativ bestimmt werden kann. Eine Einteilung, die mit der Anzahl der Bandsegmente übereinstimmt, ist wegen des sehr hohen Rechenaufwands während der Simulation nicht gewählt worden. In Bild 1.11 ist die Aufteilung in zwei mal vier verschieden große Bereiche zu sehen. Die Bezeichnungen dieser Abschnitte lauten Band_auf(n), wobei der Wert in Klammern die Anzahl der einbezogenen Bandsegmente angibt. Die Simulationsergebnisse zeigen, dass diese Teilung eine verwendbare Genauigkeit liefert.
Da die sich gegenüberliegenden Bandabschnitte Band_auf(n) und Band_ab(n) eine gemeinsame Wirkung auf ein und die selbe Stelle der Säule haben, ist das auf- und ablaufende Band gleich unterteilt. Die feiner werdende Gliederung in Richtung des Terminals ist damit begründet, dass die Influenzströme in dessen Nähe eine direkte und damit größere Wirkung haben. Die sechs Bandsegmente, die dem Terminal zugeordnet sind, werden bei dieser Unterteilung nicht beachtet, da deren Verschiebungsstrom direkt auf das Terminal

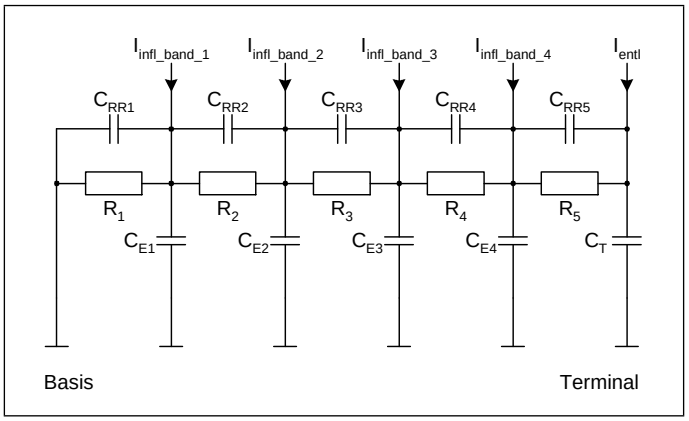


Bild 1.12: Elektrisches Ersatzschaltbild der Säule

einwirkt und somit in dieser Eigenschaft dem Entladestrom gleichkommt. Im Bild 1.10 ist zu sehen, dass der Influenzstrom des Terminals direkt zum Entladestrom addiert wird.
Der einzelne Influenzstrom bzw. die Änderung des elektrischen Flusses innerhalb eines Bandabschnitts und damit die Wirkung auf einen Teil der Säule wird durch die Differenz zwischen dem hinein- und hinausfließenden Strom gebildet. Dieser wird mit dem Strom des gegenüberliegenden Abschnittes summiert und zur Modellstruktur der Säule (Bild 1.13 bzw. Bild B.4 in Anhang B) geleitet. Dieses Subsystem bildet die in Bild 1.12 gezeigte Ersatzschaltung in SIMULINK nach. Über jeder Kapazität C_{En} (Ring → Erde) stellt sich ein Potential ein, welches vom Säulenstrom und damit auch von den Einzelströmen der benachbarten Elemente abhängig ist. Der Strom $i_{C_{En}}$ berechnet sich aus der Summe der Einzelströme:

$$i_{C_{En}} = i_{infl\_band\_n} + i_{C_{RR(n+1)}} + i_{R_{n+1}} - i_{C_{RRn}} - i_{R_n} \,. \tag{1.10}$$

Mit der Strom-Spannungsbeziehung

$$i = C\,\frac{\mathrm{d}u_C}{\mathrm{d}t} \qquad \text{bzw.} \qquad \frac{\mathrm{d}u_C}{\mathrm{d}t} = \dot{u}_C = \frac{i}{C} \tag{1.11}$$

ergeben sich an den Kondensatoren die Teilströme

$$i_{C_{RRn}} = C_{RRn} \left(\dot{u}_{C_{En}} - \dot{u}_{C_{E(n-1)}}\right) \qquad \text{bzw.} \tag{1.12}$$

$$i_{C_{RR(n+1)}} = C_{RR(n+1)} \left(\dot{u}_{C_{E(n+1)}} - \dot{u}_{C_{En}}\right) \,, \tag{1.13}$$

und durch die Widerstände fließen die Teilströme

$$i_{R_n} = \frac{u_{C_{En}} - u_{C_{E(n-1)}}}{R_n} \qquad \text{und} \tag{1.14}$$

$$i_{R_{n+1}} = \frac{u_{C_{E(n+1)}} - u_{C_{En}}}{R_{n+1}} \,. \tag{1.15}$$

Mit diesen Gleichungen lässt sich das Verhalten in SIMULINK nachbilden. Durch die Verwendung eines Discrete-Time Integrator stehen sowohl die Spannung $u_{C_{En}}$ als auch deren Ableitung $\dot{u}_{C_{En}}$ zur Verfügung. Zwei benachbarte Segmente dieser Struktur bilden die Gleichung 1.10 nach, in welche die Gleichungen 1.12 bis 1.15 eingesetzt wurden.

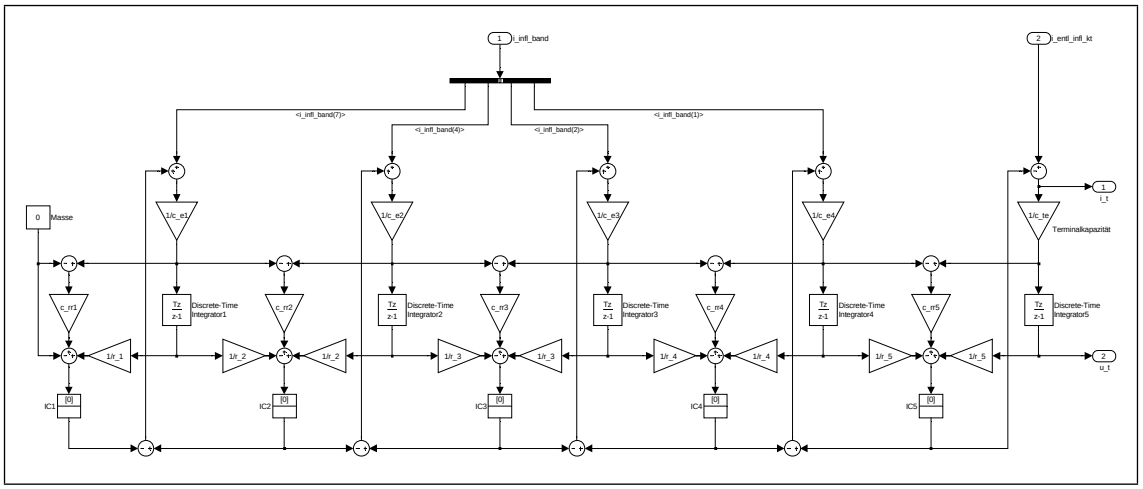


Bild 1.13: Modellstruktur der Säule

1.7 Kapazitive Messelektrode

Ein Möglichkeit, die Spannungsschwankungen am Terminal zu quantifizieren, besteht darin, über eine so genannte Capacitive-Pick-Up (CPU) die Wechselanteile der Hochspannung zu messen. Realisiert wird die CPU mittels eines metallischen Bandes, welches an der Innenseite des Druckbehälters direkt auf der Höhe des Terminals angebracht ist. Eine Isolierung trennt das Metallband vom Behälter. Durch diese Anordnung ergibt sich ein theoretisch frequenzunabhängiger kapazitiver Spannungsteiler zwischen Terminal, Metallstreifen und Behälter:

$$\frac{U_{cpu}}{U_t} = \frac{C_1}{C_1 + C_2}\,. \tag{1.16}$$

Da aber der nachgeschaltete Verstärker einen Widerstand zur Masse besitzt, ergibt sich für diese Anordnung doch eine Zeitkonstante in ihrem Übertragungsverhalten:

$$\frac{U_{cpu}}{U_t} = \frac{C_1}{C_1 + C_2} \cdot \frac{j\omega\tau}{1 + j\omega\tau} \qquad \text{mit} \qquad \tau = R(C_1 + C_2)\,. \tag{1.17}$$

Die Grenzfrequenz wurde zu 0,7 Hz ermittelt. Daraus ergibt sich, dass an dieser Elektrode nur die Wechselanteile der Hochspannung am Terminal gemessen werden können. Zusätzlich werden diese Schwankungen, welche mehrere hundert Volt betragen können, durch den kapazitiven Spannungsteiler um den Faktor 2700 verringert. Das beschriebene Hochpass-Verhalten der Übertragungsfunktion wird im $\mathcal{Z}$-Bereich durch die Gleichung A.3 im Anhang A für die Simulation mit SIMULINK nachgebildet. Die Einordnung der CPU in das Gesamtmodell ist in Bild 1.2 zu sehen.

1.8 Systemregler

Um die Spannungsschwankungen an der Hochspannungselektrode auszugleichen, stehen zwei verschiedenartige Regler zur Verfügung (Bild 1.14). Die Koronatriode ist in der Lage,

innerhalb sehr geringer Zeit kurzzeitige Spannungsabweichungen am Terminal zu glätten. Der zweite Regler wirkt durch sein integrierendes Verhalten langzeitstabilisierend auf den Ladestrom und somit auf den gewählten Arbeitspunkt. Die Regelabweichung der Terminalspannung liefert ein den Reglern vorgeschaltetes System, welches im Modell vereinfacht durch einen Differenzverstärker (em_sp) nachgebildet wird.

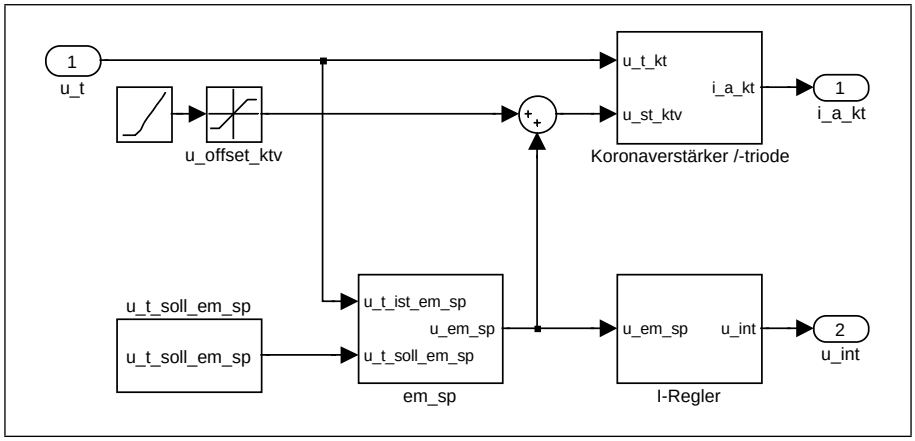


Bild 1.14: Ansteuerung der Koronatriode und des I-Reglers

1.8.1 Koronatriode

Eine von außen in den Druckbehälter eingeführte Elektrode bildet mit dem Hochspannungsterminal zusammen eine Anordnung, die einer Triode gleichkommt. Dabei hat das Terminal die Funktion der Anode, die eingeführte Elektrode die der Kathode und das Gitter der Triode ist an Masse angeschlossen. Durch die gezielte Modulation der Kathodenspannung U_k ist es möglich, negative Ladungsträger (Kathodenstrom I_k) zur positiv geladenen Hochspannungselektrode zu leiten und damit deren Spannung U_t (bzw. die Anodenspannung U_a) zu verändern (Bild 1.15):

$$I_k = S(D \cdot U_t - U_k)\,. \tag{1.18}$$

Die Röhrengleichung 1.18 gibt den allgemeinen mathematischen Zusammenhang der Triode in Gitterbasisschaltung an. Die Faktoren S und D sind Röhrenkenngrößen und stellen

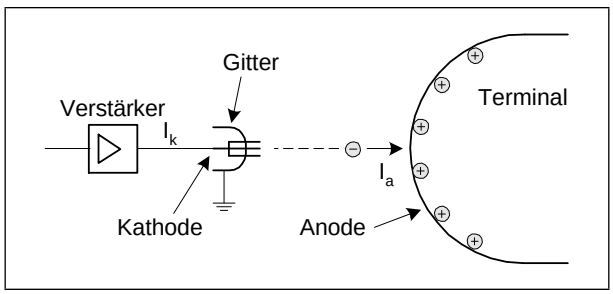


Bild 1.15: Schematischer Aufbau der Koronatriode

zum einen die Steilheit und zum anderen den Durchgriff der Triode dar:

$$S = \left.\frac{\Delta I_a}{\Delta U_g}\right|_{U_a=konst} \qquad \text{und} \qquad D = \left.\frac{\Delta U_g}{\Delta U_a}\right|_{I_a=konst} . \tag{1.19}$$

Die Stärke des auf dem Terminal ankommenden Anodenstroms I_a ist eine Funktion des Kathodenstroms I_k und der Ionenlaufzeit T_{ion} zwischen Kathode und Anode bzw. Terminal, und wird durch

$$I_a = \frac{1}{T_{ion}} \int_{T_{ion}} I_k(t-\tau)\,\mathrm{d}\tau \tag{1.20}$$

dargestellt. Das bedeutet, dass der Integrator die Strecke zwischen beiden Elektroden nachbildet und die Wirkung des Anodenstroms auf das Terminal durch die Subtraktion vom Terminalstrom modelliert wird. Da sich das beschriebene System in einem mit Gas gefüllten Druckbehälter befindet, wird bei dieser Modellvorstellung davon ausgegangen, dass die Elektronen nicht wie im Vakuum gleichmäßig beschleunigt werden, sondern sich mit konstanter Geschwindigkeit in einem homogenen elektrischen Feld bewegen.

Verhaltensbeschreibung mittels SIMULINK

Da die Kathodenspannung der Triode durch einen Verstärker bereitgestellt wird, muss sein Übertragungsverhalten in der Modellierung mit berücksichtigt werden. Dazu wird die linearisierte Ersatzschaltung des Verstärkers als spannungsgesteuerte Spannungsquelle angenommen, welche ein inhärentes Tiefpassverhalten besitzt. Dieses wird durch den komplexen Widerstand $\underline{Z}$ dargestellt. Die mathematische Beschreibung dieser Ersatzschaltung lautet:

$$U_k = U_0 - \underline{Z} \cdot I_k \qquad \text{mit} \qquad U_0 = U_{st} \cdot v\,, \tag{1.21}$$

wobei U_{st} die Summe aus der Vorgabe für den Arbeitspunkt der Triode (u_offset_ktv) und der modulierenden Steuerspannung (u_em_sp) ist. Wird Gleichung 1.18 nach U_k aufgelöst und in obige Gleichung eingesetzt so ergibt sich

$$U_0 - \underline{Z} \cdot I_k = \frac{S \cdot D \cdot U_t - I_k}{S}\,. \tag{1.22}$$

Sollte diese Gleichung mit SIMULINK modelliert werden, so würde eine algebraische Schleife im Signalpfad auftreten, welche zum Startzeitpunkt der Simulation nicht zu lösen ist. Abhilfe verschafft eine Funktion namens Algebraic Constraint, die von SIMULINK für solche Problemstellungen vorgesehen ist. Voraussetzung für die Anwendung dieses Blocks ist die Umstellung der zuletzt aufgeführten Gleichung nach Null

$$0 = S(U_0 - D \cdot U_t - \underline{Z} \cdot I_k) + I_k \tag{1.23}$$

und das Einsetzen des Blocks in die entstandene Signalschleife. Somit liefert diese Funktion am Ausgang immer Werte, die dafür sorgen, dass der Eingang des Blocks immer gleich

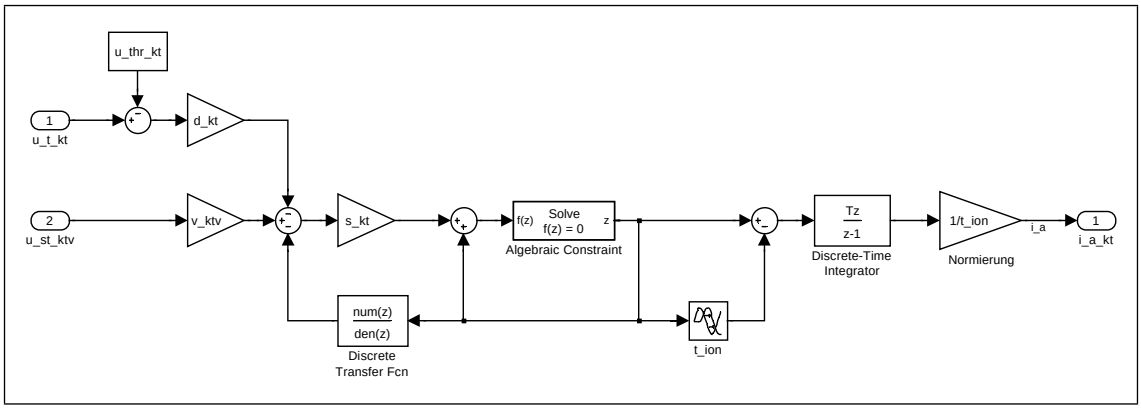


Bild 1.16: Koronatriode inkl. Verstärker

Null gesetzt wird. In Bild 1.16 bzw. in Bild B.5 im Anhang B ist die Nachbildung dieses Sachverhalts gezeigt. Zusätzlich wird noch eine Schwellenspannung (u_thr_kt) eingebunden, welche erst von der Terminalspannung überwunden werden muss, damit diese einen Einfluss auf die Koronatriode ausüben kann. Das Übertragungsverhalten des komplexen Widerstandes $\underline{Z}$ wird mit dem Block Discrete Transfer Fcn realisiert. Die Herleitung dieser Funktion ist im Anhang A (Gl. A.4) aufgeführt. Die Indizes *kt* und *ktv* beziehen sich jeweils auf die Verwendung der Größen, entweder für die Koronatriode oder für den Koronatriodenverstärker. Die Funktion der Gleichung 1.20 wird mit den Block Discrete-Time Integrator gestaltet. Eine verzögerte Subtraktion des Stroms I_k vor dem Integrator bewirkt, dass dieser nur so lange den Wert des Stroms aufsummiert, wie sich die Elektronen zwischen Kathode und Anode befinden (T_{ion}). Die Normierung des Anodenstroms auf die Ionenlaufzeit wird mit einem Verstärkerblock realisiert.

1.8.2 Integrierender Regler

Um einen gewählten Arbeitspunkt stabil zu halten, wird für die Regelung des Ladestroms ein integrierender Regler genutzt. Als Eingangsgröße erhält er die gleiche Spannung u_em_sp wie die Koronatriode, reagiert aber durch seine Integrationskonstante v_int sehr viel langsamer auf diese Änderungen. Somit lässt sich die Spannungsvorgabe für den Ladestrom u_i_lade_soll langsam und ohne Überschwingungen nachführen. In SIMULINK ergibt sich eine Struktur, welche in Bild 1.17 dargestellt ist. Der Integrator ist durch ein Discrete-Time Integrator realisiert, mit einer Begrenzung der Ausgangsspannung u_int von $\pm 5\,\mathrm{V}$.

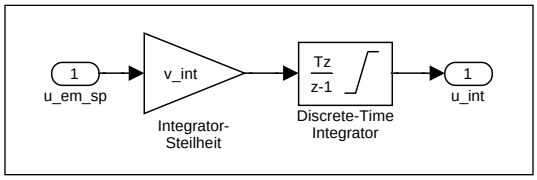


Bild 1.17: Integrierende Reglerstruktur

1.9 Rauschen

Das Modell des Van-de-Graaff-Generators stellt bisher eine idealisierte Verhaltensbeschreibung des Systems dar. Um eine möglichst getreue Nachbildung des Generators zu erhalten, ist es deshalb unbedingt notwendig, seine inhärenten Störquellen zu berücksichtigen. Ein Großteil der Störungen wird durch die periodisch wiederkehrenden Inhomogenitäten des Bandes verursacht, weshalb es als ausreichend erachtet wurde, nur diese nachzubilden und dem Modell zuzuführen. Dazu wurde aus einer Messreihe des Ladestroms ein Zeitausschnitt extrahiert und mit Hilfe einer Mittelwertbildung der Gleichanteil beseitigt. Das Ergebnis ist in Bild 1.18 dargestellt.

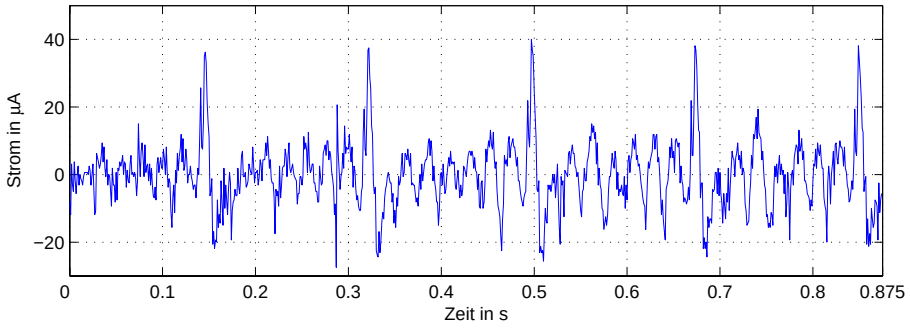


Bild 1.18: Periodische Störung des Ladestroms

Die Länge des Zeitabschnitts beträgt 0,875 ms und entspricht damit exakt dem Fünffachen der Bandumlaufzeit. Theoretisch würde auch das Störsignal eines Bandumlaufs ausreichen um die Periodizität zu modellieren. Aber wie im Bild zu sehen ist, unterscheiden sich nachfolgende Perioden teilweise in ihrem Verlauf, sodass diese Differenzen mit einem längeren Zeitausschnitt zusätzlich berücksichtigt werden können. Eine Beschränkung auf fünf Perioden soll die Größe des Datensatzes und damit den Bedarf an Arbeitsspeicher während der Simulation klein halten. Der ermittelte Datensatz, bestehend aus Zeit- und Amplitudenvektor, wird an den in Bild 1.19 bzw. in Bild B.6 im Anhang B zu sehenden Punkten jeweils dem Ladestrom I_{lade} und dem Strom vor dem Entladesystem $I_{t\_auf\_a}$ während der Simulationslaufzeit fortlaufend aufsummiert. Mit dem Faktor v_rausch kann die Intensität des Rauschsignals eingestellt werden. Die Quantität der eingespeisten Störleistung kann mit Hilfe des Leistungsdichtespektrums überprüft werden [4]. Dazu wurden am Van-de-Graaff-Generator das Leistungsdichtespektrum des Ladestroms und das der CPU-Spannung gemessen und jeweils als Sollwert der modellseitigen Spektren übernommen. Durch die Variation des Parameters v_rausch wurden im Modell mehrere Leistungsdichtespektren berechnet und auf ihre Verwendungsfähigkeit hin überprüft. Eine hinreichend genaue Nachbildung ist in Bild 1.20 dargestellt. Es zeigt im oberen Teil das Spektrum der CPU-Spannung und im unteren Teil das des Ladestroms. Der jeweils blaue Kurvenverlauf ist das am Generator über einhundert Messungen gemittelte Spektrum und der rote

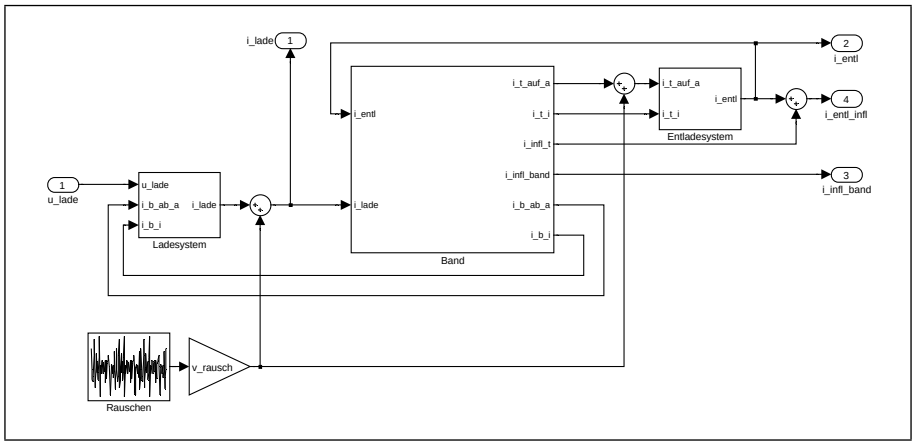


Bild 1.19: Einspeisepunkte des Störstroms

Kurvenverlauf das am Modell berechnete. Auf eine Mittelung des Modellspektrums wurde verzichtet, da die Simulation hierfür ebenfalls einhundert mal mit jeweils veränderten Störparametern durchgeführt werden müsste. Trotz der einmaligen Berechnung des Leistungsdichtespektrums am Modell sind die Anteile, die mit Bandumlauffrequenz und deren ganzzahligen Vielfachen zur Gesamtstörung beitragen, deutlich zu erkennen. Die teilweise erheblichen Abweichungen der modellierten Spektralverteilungen könnten zum einen durch die bereits angesprochene Mittelung mehrerer Messungen und andererseits durch eine sehr viel größere Detailtiefe im Modell verringert werden. Auf diesen hohen Aufwand soll aber verzichtet werden, da die bisher erreichten Ergebnisse eine ausreichend hohe Genauigkeit aufweisen. Da für die weitere Arbeit hauptsächlich die CPU-Spannung von Bedeutung ist, wurde auf die Nachbildung des Leistungsdichtespektrums dieser Spannung ein höherer Wert gelegt.

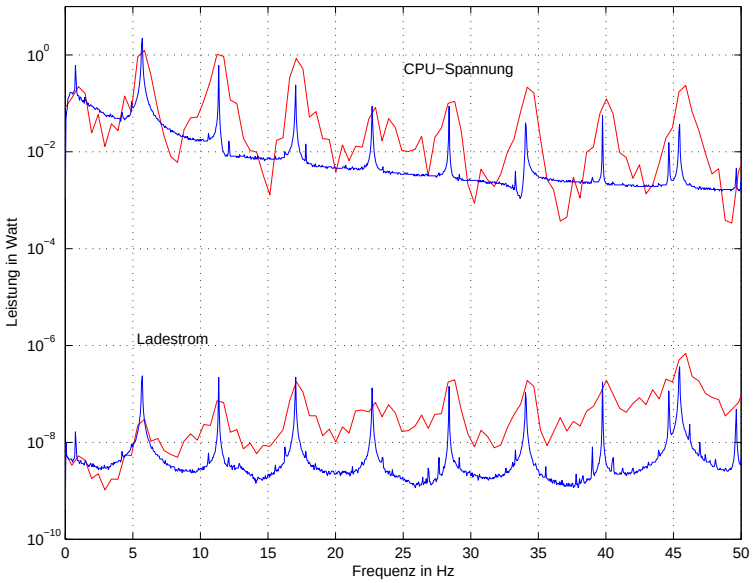


Bild 1.20: Gemessene und modellierte Leistungsdichtespektren

1.10 Koeffizientenbestimmung

Ausgehend von den Gleichungen 1.1 und 1.2 aus dem Abschnitt 1.4 können die bereits beschriebenen Koeffizienten bestimmt werden. Stellt man verschiedene Arbeitspunkte am Beschleuniger ein und misst die dazugehörigen Größen des Ladestroms und der Ladespannung, so erhält man durch lineare Regression der gemessenen Werte eine statische Kennlinie, welche einen ähnlichen Verlauf aufweist wie er in Bild 1.21 qualitativ dargestellt ist. Es ist deutlich zu sehen, dass erst eine bestimmte Ladespannung ($U_{lade} = U_0$) aufgebracht werden muss, bevor ein Ladestrom und damit auch ein Entladestrom fließen kann. Diese Spannung ergibt sich aus der Addition der Modellgleichungen 1.1 und 1.2. Da im Arbeitspunkt $U_{lade} = U_0$ noch kein Lade- und Entladestrom fließen kann, sind somit auch alle anderen in den Gleichungen vorkommenden Ströme gleich Null. Damit ergibt sich:

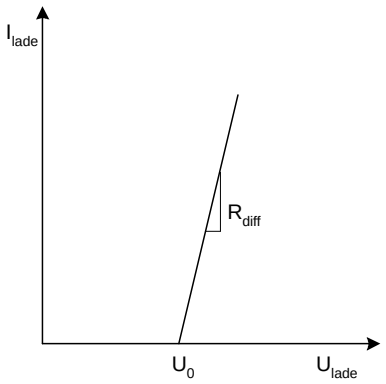


Bild 1.21: Ladestrom-Kennlinie des Beschleunigers

$$0 = k_u(U_0 - U_{thr}) + k_u(-U_{thr}) \quad (1.24)$$

$$= U_0 - 2 \cdot U_{thr} \quad (1.25)$$

$$U_{thr} = \frac{U_0}{2}\,. \quad (1.26)$$

Durch den gemessenen Wert $U_0 = 8,372\,\mathrm{kV}$ ist damit $U_{thr} = 4,186\,\mathrm{kV}$.
Der Wert von k_u bestimmt innerhalb des Modells welcher Ladestrom sich bei einer vorgegebenen Ladespannung ohne Gegenwirkung einer Ladung auf dem Band einstellt. Da im Arbeitspunkt des eingeschwungenen Systems gilt, dass I_{lade} gleich I_{entl} ist, können zur Ermittlung des Faktors k_u in den Gleichungen 1.1 und 1.2 der Wert von I_{entl} durch I_{lade} ersetzt, und die daraus resultierenden Gleichungen wiederum addiert werden. Die Terme, welche mit k_{ia} und k_{ii} bewertet sind, können bei der Betrachtung eines festen Arbeitspunktes vernachlässigt werden. Sie repräsentieren das dynamische Verhalten und bestimmen nicht die Einstellung eines bestimmten Arbeitspunktes, sondern lediglich die Zeitdauer bis zum Erreichen des Gleichgewichtszustands. Es ergeben sich folgende Zusammenhänge:

$$I_{lade} + I_{entl} = k_u(U_{lade} - U_{thr} - I_{lade} \cdot R_{i\_lade}) + k_u(-U_{thr} - I_{entl} \cdot R_{i\_entl}) \quad (1.27)$$

$$2 \cdot I_{lade} = k_u(U_{lade} - 2 \cdot U_{thr} - I_{lade}(R_{i\_lade} + R_{i\_entl}))\,. \quad (1.28)$$

Umgestellt nach k_u folgt:

$$k_u = \frac{2 \cdot I_{lade}}{U_{lade} - 2 \cdot U_{thr} - I_{lade}(R_{i\_lade} + R_{i\_entl})} \,. \tag{1.29}$$

Für einen typischen Arbeitspunkt von $60\,\mu$A bei 13,5 kV, welcher aus der statischen Kennlinie gewählt werden kann, erhält man mit $R_{i\_lade} = 25\,\mathrm{M\Omega}$ und $R_{i\_entl} = 6,75\,\mathrm{M\Omega}$:

$$k_u = \frac{2 \cdot 60\,\mu\mathrm{A}}{13,5\,\mathrm{kV} - 2 \cdot 4,186\,\mathrm{kV} - 60\,\mu\mathrm{A}(25\,\mathrm{M\Omega} + 6,75\,\mathrm{M\Omega})} \tag{1.30}$$

$$k_u = 37,2324\,\frac{\mu\mathrm{A}}{\mathrm{kV}} \,. \tag{1.31}$$

Übernimmt man diese Koeffizienten in das erstellte Modell, so ist zu sehen, dass damit ähnliche Arbeitspunkte eingestellt werden können wie sie sich auch im Betrieb des Beschleunigers zeigen.
Werden mit Hilfe der Lade- bzw. Entladegleichungen die restlichen Faktoren ermittelt, ergibt sich für k_{ia} der Wert 0,3878. Es stellt sich im Folgenden aber immer die triviale Lösung $k_{ii} = 0$ ein, weswegen der dafür benutzte Lösungsweg nicht aufgeführt werden soll. Es besteht die Vermutung, dass diese Parameter, welche das dynamische Verhalten des Systems widerspiegeln, unter Verwendung der statischen Kennlinie nicht genau genug ermittelt werden können. Überlegungen gehen dahin, dass diese Faktoren über die Auswertung des gemessenen Frequenzganges und unter Nutzung des Parameters γ (vgl. Gleichung 1.5) bestimmt werden könnten. Dieser Ansatz wurde aber im Rahmen dieser Arbeit nicht weiter verfolgt. Für alle Simulationen wurde festgesetzt, dass $k_{ia} = 0,3878$ und der Faktor k_{ii} ein Zehntel von k_{ia} beträgt.

1.11 Simulationsergebnisse

In diesem Abschnitt der Arbeit sollen zwei ausgewählte Simulationsergebnisse vorgestellt werden, welche die Funktionalität des entworfenen Modells bestätigen.

Arbeitspunkteinstellung

In Bild 1.22 ist zu sehen, wie sich der Entladestrom (rote Kurve) dem Ladestrom (blaue Kurve) angleicht und sich dadurch ein stabiler Arbeitspunkt einstellt. Es ist deutlich zu erkennen, dass erst eine bestimmte Zeit lang ein Ladestrom auf das Band fließen muss, bis sich darauf so viele Ladungen befinden, dass die Schwellenspannung an der Entladestelle überschritten wird und ein Entladestrom fließen kann. Die deutliche Stufung des Entladestroms ist durch die schubweise Ladungsakkumulation bei jedem einzelnen Bandumlauf bedingt. Der untere Teil des Bildes stellt einen Ausschnitt aus dem obigen Verlauf dar, welcher zeigen soll, dass die im Abschnitt 1.5 beschriebene Eigenschaft der Verringerung des Ladestroms, (bedingt durch die Gegenwirkung der ins Bandinnere gelangten Ladungsträger) in das Modell implementiert wurde. Dieser Vorgang zeigt sich aber in der Simulation

in Größenordnungen, welche weit unter denen des realen Prozesses liegen. Es ist möglich, diesen Effekt durch die Koeffizienten k_{ia} und k_{ii} zu beeinflussen, dies unterliegt aber dem im vorhergehenden Abschnitt beschriebenen Problem, dass die beiden Faktoren noch nicht genau ermittelt werden konnten.

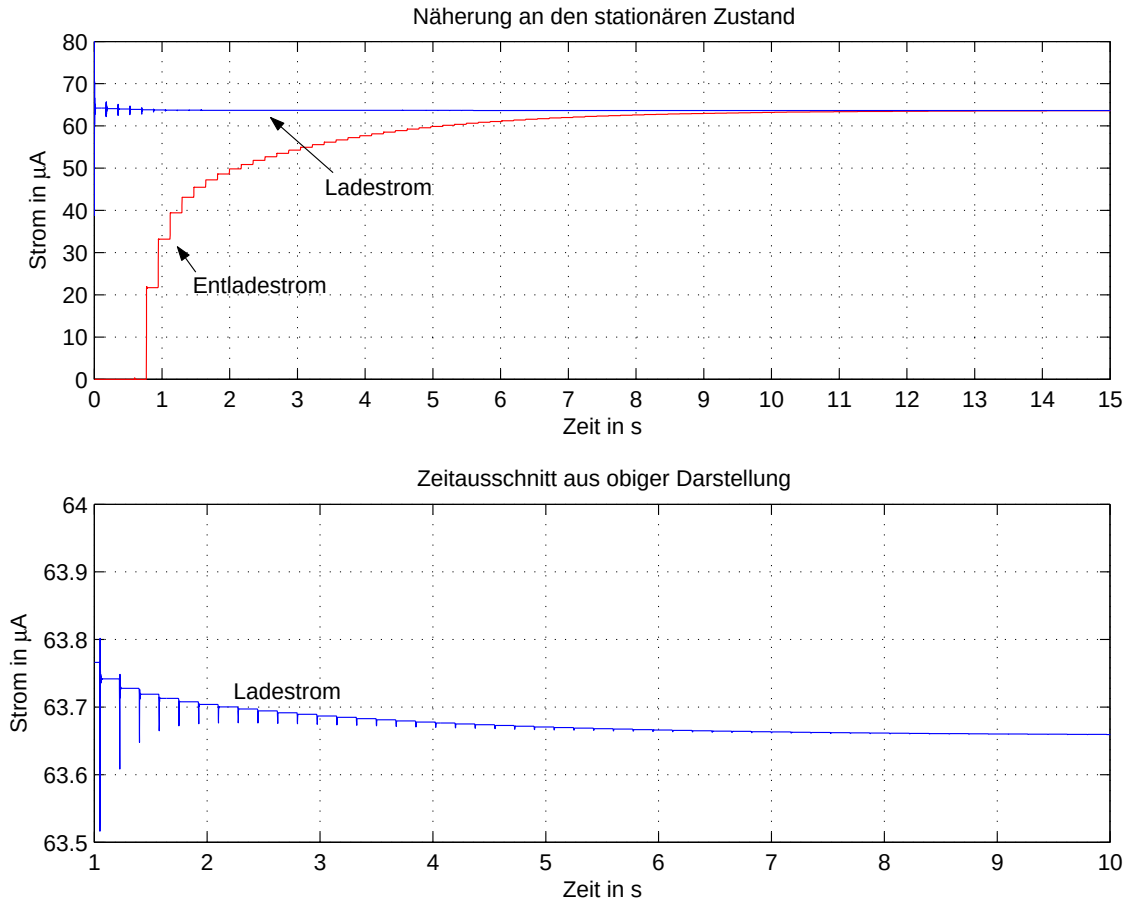


Bild 1.22: Lade- und Entladestrom

Frequenzgang

Im zweiten Beispiel (Bild 1.23) ist der gemessene (blaue Kurve) und der simulierte Frequenzgang (rote Kurve) dargestellt. Damit die Merkmale des Frequenzganges deutlicher erkennbar sind, wurden die gemessenen Daten auf die Übertragungsfunktion der Säule normiert, d. h. deren Einfluss eliminiert, da diese mit ihrem Tiefpassverhalten die Messdaten deutlich beeinflusst [3]. Im Gegensatz zum ersten Beispiel wurde in diesem Fall der Wert des Faktors k_{ia} angepasst, damit der dargestellte Frequenzgang als Simulationsergebnis ermittelt werden konnte, und unterstreicht damit wiederum das Problem der eindeutigen Bestimmung der verwendeten Koeffizienten. Mit Darstellung 1.23 soll lediglich veranschaulicht werden, dass der charakteristische Verlauf des Frequenzganges durch die Struktur des Modells nachgebildet werden kann. Es sind deutlich die schon im Abschnitt 1.6 beschrie-

benen Maxima zu erkennen, welche jeweils zwischen den Frequenzen des Bandumlaufs von 5,7 Hz liegen.

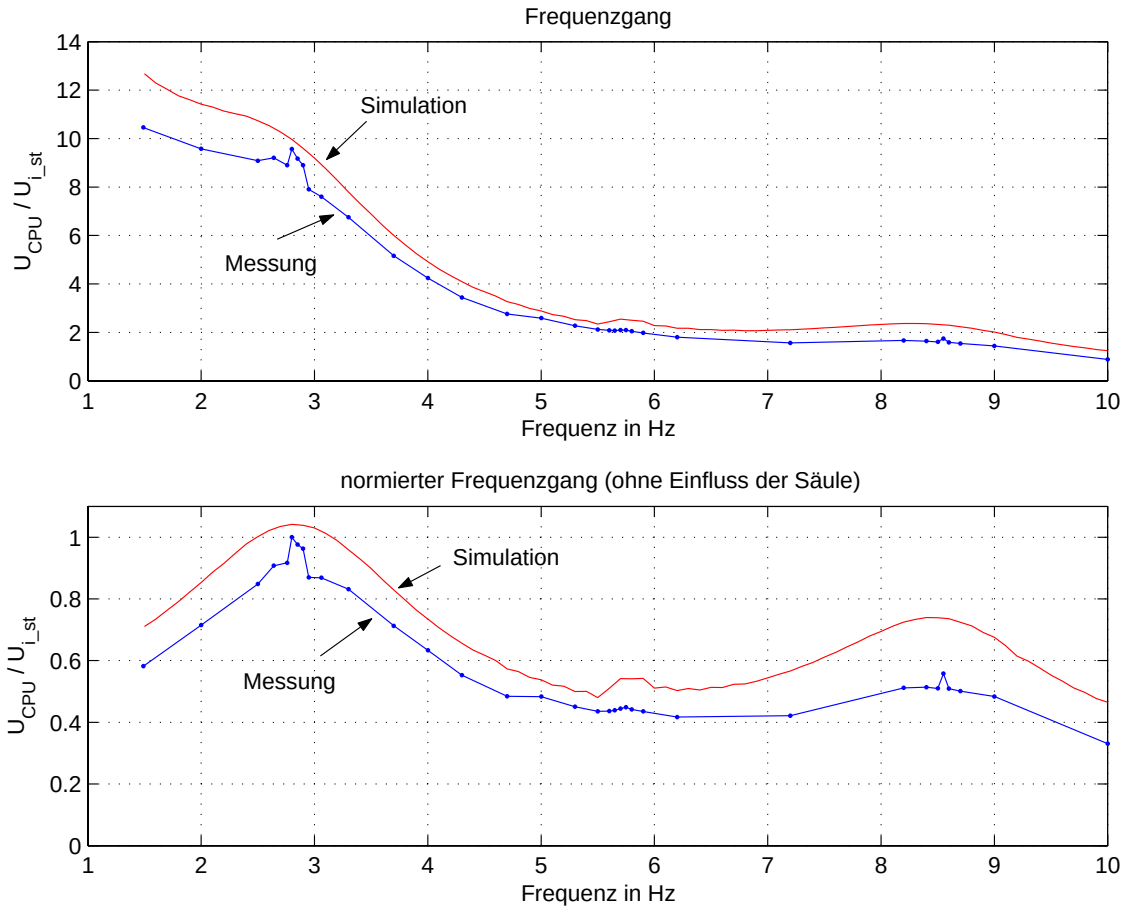


Bild 1.23: Gemessener und simulierter Freuenzgang

2 Online-Bestimmung von Betriebsparametern

Zu den Betriebsparametern, welche während des Betriebs (online) des Ionenbeschleunigers bestimmt werden sollen, zählt das Übertragungsverhalten von Teilsystemen als eine Grundlage für alle weiteren Betrachtungen.

2.1 Übertragungsfunktionen

Wie schon in Abschnitt 1.8 gezeigt, lässt sich das gesamte System durch zwei Regler steuern. Das ist zum einen der Ladestromregler und zum anderen die Regelung über die Koronatriode. Da diese die einzigen Punkte sind, an welchen sich aktiv in das Verhalten des Systems eingreifen lässt, ist es notwendig, das jeweilige Übertragungsverhalten der beiden Teilsysteme zu bestimmen. Dazu sollen nacheinander Testsignale mit verschiedenen Frequenzen in die Signalpfade eingekoppelt und die Reaktion am Terminal bzw. an der kapazitiven Messelektrode (CPU) gemessen werden. Hierdurch erhält man die Übertragungsfunktion zwischen der Beladestelle des Bandes und dem Terminal und die Übertragungsfunktion zwischen der Koronatriode und dem Terminal (Bild 2.1). Zu beachten ist dabei die gegenseitige Beeinflussung der Regler. Eine Änderung an einem Eingang hat eine Reaktion über das Terminal und den jeweiligen Regler am anderen Eingang des Systems zur Folge. Damit ist es nicht möglich die Übertragungsfunktionen unabhängig voneinander zu bestimmen. Es ist deshalb erforderlich, nicht die generierten Testsignale zur Bestimmung der Übertragungsverhaltens zu nutzen, sondern die tatsächlich eingespeisten Signale an jedem Eingang. Diese müssen gemessen und mit dem Ausgangssignal am Terminal korrelativ ausgewertet werden.

2.2 Berechnung der Übertragungsfunktionen

Die folgenden Berechnungen setzen voraus, dass der Beschleuniger als ein lineares System betrachtet werden kann und dass die Störgröße Z vernachlässigbar klein ist. Das System soll so stimuliert werden, dass jeweils nur an einem Eingang ein Signal anliegt:

$$U_B(\omega)\,\big|_{U_{KT}(\omega)=0} \quad \text{und} \quad U_{KT}(\omega)\,\big|_{U_B(\omega)=0}\ . \tag{2.1}$$

Daraus ergeben sich zwei Arbeitspunkte bei der Frequenz ω, die durch ein lineares Gleichungssystem dargestellt werden können:

$$\underline{U}_{T1}(\omega) = \underline{F}_B(\omega)\cdot\underline{U}_{B1}(\omega) + \underline{F}_{KT}(\omega)\cdot\underline{U}_{KT1}(\omega) \tag{2.2}$$

$$\underline{U}_{T2}(\omega) = \underline{F}_B(\omega)\cdot\underline{U}_{B2}(\omega) + \underline{F}_{KT}(\omega)\cdot\underline{U}_{KT2}(\omega)\,. \tag{2.3}$$

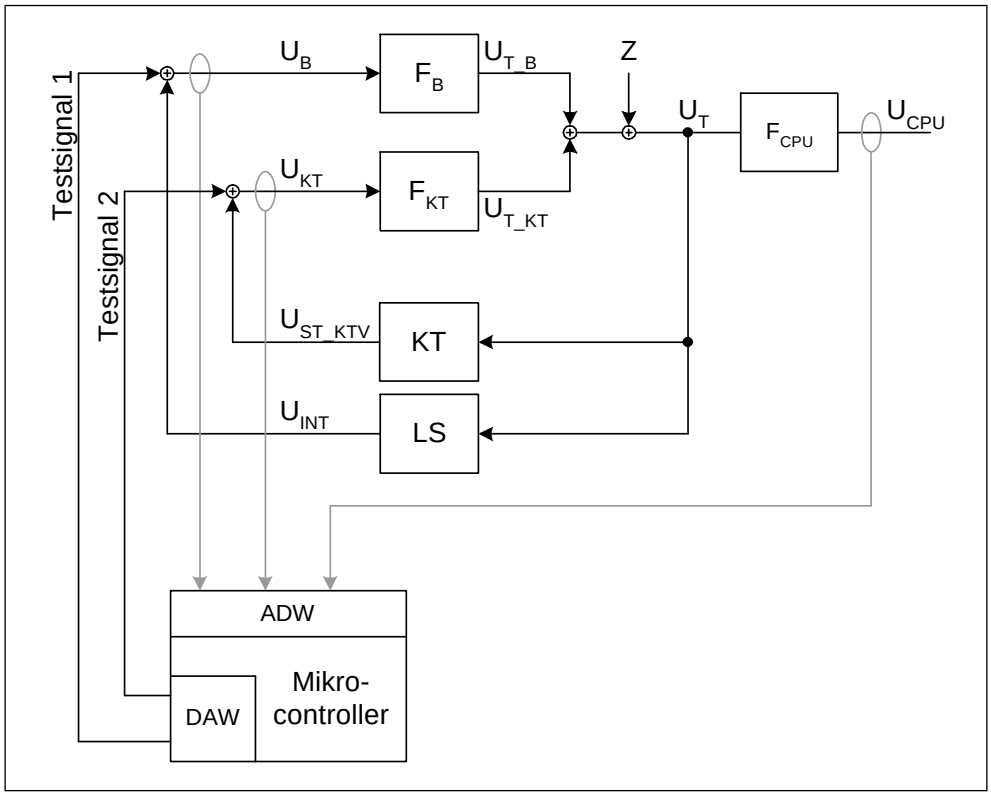


Bild 2.1: Schema zur Bestimmung der Übertragungsfunktionen

Aufgelöst nach den zu ermittelnden Übertragungsfunktionen ergeben sich die zwei Gleichungen

$$\underline{F}_{KT}(\omega) = \frac{\underline{U}_{T1}(\omega)\cdot\underline{U}_{B2}(\omega) - \underline{U}_{T2}(\omega)\cdot\underline{U}_{B1}(\omega)}{\underline{U}_{KT1}(\omega)\cdot\underline{U}_{B2}(\omega) - \underline{U}_{KT2}(\omega)\cdot\underline{U}_{B1}(\omega)} \quad \text{und} \tag{2.4}$$

$$\underline{F}_{B}(\omega) = \frac{\underline{U}_{T2}(\omega)\cdot\underline{U}_{KT1}(\omega) - \underline{U}_{T1}(\omega)\cdot\underline{U}_{KT2}(\omega)}{\underline{U}_{KT1}(\omega)\cdot\underline{U}_{B2}(\omega) - \underline{U}_{KT2}(\omega)\cdot\underline{U}_{B1}(\omega)}\,, \tag{2.5}$$

wobei die Indizes 1 und 2 die Arbeitspunkte angeben, in denen die Spannungen gemessen worden sind. Diese Größen stellen komplexe Werte dar, welche im Folgenden, der besseren Übersicht wegen, ohne den Zusatz (ω) und ohne Unterstreichung verwendet werden. Es soll gelten:

$$\underline{Z}(\omega) \mathrel{\widehat{=}} Z = \mathrm{Re}\{Z\} + j\,\mathrm{Im}\{Z\}\,. \tag{2.6}$$

Jede verwendete Größe lässt sich auch durch die komplexe Form einer Fourier-Reihe darstellen, welche keinen Gleichanteil und nur einen Summationsterm der gegebenen Frequenz besitzt. Dies soll als Ansatz für die folgenden Betrachtungen dienen [5], [6]. Aus der allgemeinen Fourier-Reihe

$$f(t) = \sum_{n=-\infty}^{\infty} c_n\, e^{jn\omega t} \tag{2.7}$$

ergibt sich somit

$$f(t) = c_n(\cos(n\omega t) + j\sin(n\omega t)) \tag{2.8}$$

mit dem komplexen Fourierkoeffizienten

$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\, e^{-jn\omega t}\,\mathrm{d}t \tag{2.9}$$

$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\cos(n\omega t)\,\mathrm{d}t - j\,\frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\sin(n\omega t)\,\mathrm{d}t \tag{2.10}$$

$$\text{bzw.}\quad c_n = \frac{2}{N}\sum_{n=0}^{N-1} f(nT_a)\cos\left(n\frac{2\pi}{N}\right) - j\,\frac{2}{N}\sum_{n=0}^{N-1} f(nT_a)\sin\left(n\frac{2\pi}{N}\right). \tag{2.11}$$

Durch diese Darstellung ist die Bedeutung des gewählten Ansatzes für die messtechnische Praxis zu erkennen. Wird jeder gemessene Wert $f(t)$ mit dem entsprechenden Cosinus- und Sinuswert multipliziert und über eine Periode integriert bzw. summiert, ergeben sich der Real- und der Imaginärwert des komplexen Fourierkoeffizienten. Mit dieser Methode werden die Messwerte in komplexe Größen transformiert, mit denen sich das Übertragungsverhalten eines Systems wesentlich einfacher bestimmen lässt [7].
Werden die komplexen Zahlen in Gleichung 2.4 und 2.5 durch ihre Real- und Imaginärwerte ersetzt und nach den Rechenvorschriften der komplexen Zahlen zusammengefasst, ergeben sich Gleichungen, nach denen sich die Übertragungsfunktionen mit den gemessenen Werten berechnen lassen. Da diese Gleichungen unübersichtlich groß sind, werden sie, wie im Anhang C zu sehen ist, in ihren Zähler und Nenner aufgeteilt. Die Übertragungsfunktionen werden dann allgemein mit den Gleichungen

$$F_{KT} = \frac{\mathrm{Re}\{\mathit{Zähler\_F_{KT}}\} + j\,\mathrm{Im}\{\mathit{Zähler\_F_{KT}}\}}{\mathrm{Re}\{\mathit{Nenner}\} + j\,\mathrm{Im}\{\mathit{Nenner}\}} \quad \text{und} \tag{2.12}$$

$$F_B = \frac{\mathrm{Re}\{\mathit{Zähler\_F_B}\} + j\,\mathrm{Im}\{\mathit{Zähler\_F_B}\}}{\mathrm{Re}\{\mathit{Nenner}\} + j\,\mathrm{Im}\{\mathit{Nenner}\}} \tag{2.13}$$

dargestellt. Aufgelöst in Betrag und Phase ergeben sich die Gleichungen

$$|F_{KT}| = \sqrt{\frac{\mathrm{Re}\{\mathit{Zähler\_F_{KT}}\}^2 + \mathrm{Im}\{\mathit{Zähler\_F_{KT}}\}^2}{\mathrm{Re}\{\mathit{Nenner}\}^2 + \mathrm{Im}\{\mathit{Nenner}\}^2}} \tag{2.14}$$

$$|F_B| = \sqrt{\frac{\mathrm{Re}\{\mathit{Zähler\_F_B}\}^2 + \mathrm{Im}\{\mathit{Zähler\_F_B}\}^2}{\mathrm{Re}\{\mathit{Nenner}\}^2 + \mathrm{Im}\{\mathit{Nenner}\}^2}} \tag{2.15}$$

$$\varphi_{F_{KT}} = \arctan\left(\frac{\mathrm{Im}\{\mathit{Zähler\_F_{KT}}\}}{\mathrm{Re}\{\mathit{Zähler\_F_{KT}}\}}\right) - \arctan\left(\frac{\mathrm{Im}\{\mathit{Nenner}\}}{\mathrm{Re}\{\mathit{Nenner}\}}\right) \tag{2.16}$$

$$\varphi_{F_B} = \arctan\left(\frac{\mathrm{Im}\{\mathit{Zähler\_F_B}\}}{\mathrm{Re}\{\mathit{Zähler\_F_B}\}}\right) - \arctan\left(\frac{\mathrm{Im}\{\mathit{Nenner}\}}{\mathrm{Re}\{\mathit{Nenner}\}}\right) \tag{2.17}$$

welche unter Verwendung der gemessenen Werte numerisch gelöst werden können.

Die Bestimmung der Übertragungsfunktionen kann durch einen Mikrocontroller erfolgen und sollte in der Praxis wie folgt ablaufen: In einer ersten Messung darf keiner der beiden Eingänge stimuliert werden. Durch das Messen der CPU-Spannung zu diesem Zeitpunkt wird die inhärente Störung Z ermittelt, welche bei allen nachfolgenden CPU-Messwerten subtrahiert wird. Damit wird das 'Grundrauschen' aus diesen Größen eliminiert und es wird erreicht, dass die Störgröße Z nicht in die durchzuführenden Berechnungen mit eingeht. Anschließend wird nur ein Eingang durch ein Cosinussignal mit einer bestimmten Frequenz stimuliert, der andere Eingang bleibt unbeeinflusst. Nach der Einschwingzeit des Systems werden alle benötigten Werte gemessen und gespeichert. Nachfolgend wird das selbe Testsignal auf den anderen Eingang geschaltet und nach der Einschwingzeit werden wiederum alle Messungen durchgeführt. Mit den ermittelten Werten kann nun die bereits gezeigte Berechnung entsprechend den Gleichungen 2.14 bis 2.17 durchgeführt werden. Um das Übertragungsverhalten zu bestimmen, wird dieser Ablauf für verschiedene Frequenzen wiederholt. Das Vorgehen, dass immer nur ein Eingang mit dem Testsignal beschalten wird, hat den Vorteil, dass die damit erreichten Arbeitspunkte weit auseinander liegen und die berechenbaren Ergebnisse genauer sind.

3 Parameterbestimmung durch einen Mikrocontroller

Die im vorhergehenden Abschnitt aufgestellten Gleichungen zur Bestimmung der Übertragungsfunktionen sollen während des Betriebs des Beschleunigers von einem Mikrocontroller berechnet werden. Zusätzlich sollen auch die beiden Testsignale vom Mikrocontroller generiert werden.

Für diese Aufgabe wurde durch das Forschungszentrum Rossendorf ein 32 Bit-Mikrocontroller mit der Bezeichnung SH 7047 von der Firma RENESAS gewählt. Er besitzt 12 KByte internen, 256 KByte externen Arbeitsspeicher und 256 KByte Festwertspeicher. Der interne Arbeitstakt wird aus einem externen Oszillator abgeleitet und beträgt $f_{core} = 4 \cdot 11,0592\,\text{MHz} = 44,2368\,\text{MHz}$, wobei aber die peripheren Komponenten innerhalb des Controllers, wie z. B. Analog-Digital-Wandler und Zähler, mit $f_{periph} = 2 \cdot 11,0592\,\text{MHz} = 22,1184\,\text{MHz}$ betrieben werden. Der Controller befindet sich innerhalb eines so genannten Evaluation-Development-Kit (EDK), welches im eigentlichen Sinne herstellerseitig nur für die Entwicklung von Anwendungen vorgesehen ist. Darauf sind alle Anschlüsse des Mikrocontrollers auf zwei Anschlussleisten geführt, eine serielle Schnittstelle angebracht und eine Programmierschnittstelle montiert. Über diese Schnittstelle wird der Controller von einem PC aus über einen Emulator (E10A-USB Emulator) programmiert und der entwickelte Quellcode während der Laufzeit getestet. Bild 3.1 zeigt das verwendete EDK 7047.

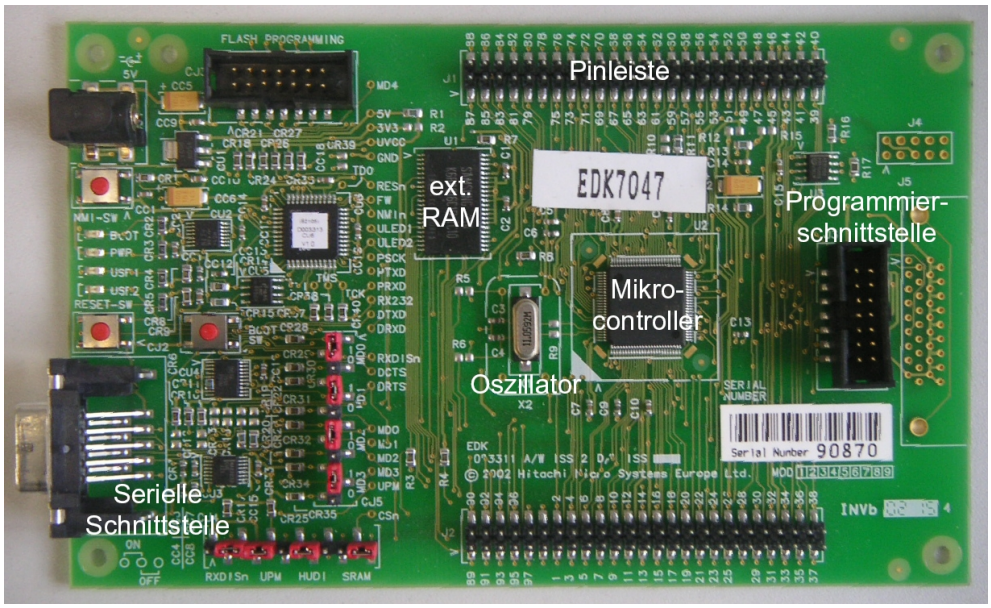


Bild 3.1: Mikrocontroller SH 7047 innerhalb des EDK

Da der Mikrocontroller keine internen Digital-Analog-Wandler (DAW) besitzt, mussten für die Arbeit externe Wandler genutzt werden, die sich auf einer Platine befinden, welche auf die Pinleisten des EDK gesteckt wird (Bild D.1 im Anhang D). Eine zusätzliche externe Schutz- und Anpassschaltung für die analogen Signaleingänge des Controllers und die dafür benötigte Spannungsversorgung bilden zusammen mit der aufgesteckten Platine die in Bild D.2 im Anhang D gezeigte Experimentieranordnung, die von Mitarbeitern des Forschungszentrums gefertigt wurde.
Da der beschriebene Mikrocontroller bei dieser Arbeit zum ersten Mal im Forschungszentrum Rossendorf zum Einsatz kam, war es ein Teil der Arbeit, die Hard- und Software der Entwicklungsumgebung für dieses EDK 7047 zu installieren und in Betrieb zu nehmen. Um anderen Mitarbeitern für ihre zukünftigen Arbeiten den Einstieg in den Umgang mit dem Mikrocontroller zu erleichtern, wurde eine Dokumentation erstellt, welche sich im Anhang F befindet und eine CD zusammengestellt, welche die Software der Entwicklungsumgebung und die genannte Dokumentation beinhaltet.

3.1 Taktsystem

Um die Parameter des Beschleunigers mit Hilfe eines Mikrocontrollers bestimmen zu können, müssen beide über ein gemeinsames Taktregime miteinander gekoppelt sein. Dabei soll die Umlaufzeit des Generatorbandes den Grundtakt darstellen, aus dem sich andere Taktzeiten für einzelne Subsysteme ableiten lassen. Dazu wird ein Impuls verwendet, den das Generatorband bei jedem Umlauf im Beschleuniger generiert. Nach diesem Takt werden alle Vorgänge im Mikrocontroller mittels einer phasengekoppelten Regelschleife (englisch: Phase-Locked Loop bzw. PLL) ausgeführt. Diese ist notwendig, da innerhalb einer Bandumlaufperiode, d. h. zwischen zwei Bandimpulsen an 128 äquidistanten Zeitpunkten die drei zur Berechnung notwendigen Größen gemessen und an 256 Zeitpunkten das Testsignal generiert werden sollen. Damit wird erreicht, dass die höchste Frequenz (fünffache Bandumlauffrequenz), welche bei dieser Arbeit von Interesse ist, mindestens 25 mal pro Periode abgetastet wird und ein gleichfrequentes Testsignal 50 Stützstellen besitzt. Ändert sich der zeitliche Abstand zwischen den Impulsen durch Störungen des Bandlaufes, so ist es die Aufgabe der PLL, die 128 Zeitpunkte neu zu berechnen und so innerhalb des Controllers ein zum Generatorband phasenstabiles Taktsystem zu schaffen. Zur Realisierung einer PLL wird ein Zähler benötigt, der 128 mal bis zu einem durch die PLL berechneten Wert zählt. Beim Erreichen dieses Wertes werden durch einen Interrupt immer wieder die Analog-Digital-Wandler zur Messung der Spannungen gestartet. Der Zähler wird auf Null zurückgesetzt und beginnt wieder zu zählen. Nach 128 Zählerperioden wird der Abstand zwischen dem Bandimpuls und dem letzten Interrupt durch einen zweiten gleichschnellen Zähler gemessen. Aus dieser Differenz berechnet die PLL einen neuen Wert, bis zu welchem der Zähler in der nächsten Periode 128 mal zählen soll. Tritt

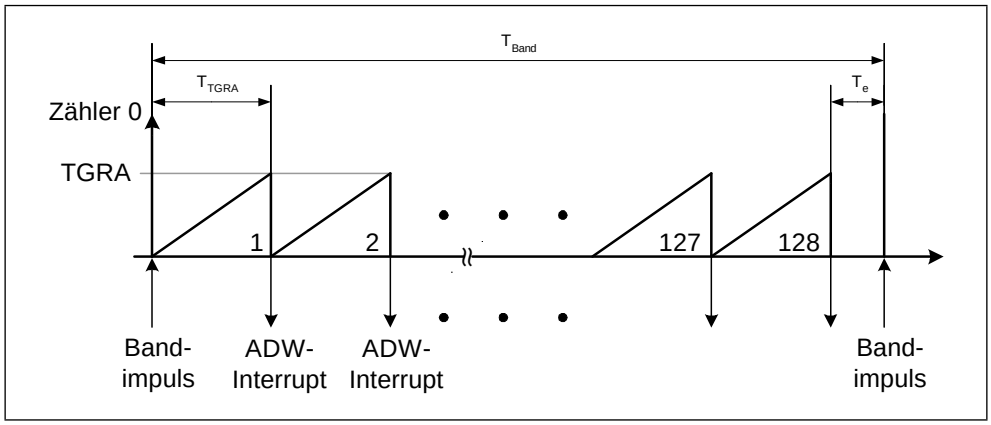


Bild 3.2: Erzeugung von 128 äquidistanten Zeitpunkten durch die PLL

der letzte Interrupt eher auf als der Bandimpuls, muss der Zählwert vergrößert werden - andernfalls verringert er sich. In Bild 3.2 ist dieses Prinzip grafisch dargestellt. Dabei gibt T_{Band} die Zeit zwischen zwei Bandimpulsen von ca. 175 ms an. Im Idealfall beträgt demzufolge eine Zählerperiode $T_{TGRA} = 1,367\,\text{ms}$ und die Abweichung T_e ist Null. Bei einer gewählten Taktung des Zählers von

$$\frac{f_{periph}}{16} = \frac{22,1184\,\text{MHz}}{16} = 1,3824\,\text{MHz} \tag{3.1}$$

ergibt sich ein Grenzwert bei 1890 Zählschritten, der im Register TGRA gespeichert wird. Mit der eingestellten Zählerfrequenz von 1,3824 MHz lässt sich durch den Differenzzähler theoretisch eine Abweichung $T_e = 0,7223\,\mu\text{s}$ erkennen. Der theoretisch geringste Zeitausgleich durch die PLL beträgt, verursacht durch die 128 Zählvorgänge, $128 \cdot 0,7223\,\mu\text{s} = 92,454\,\mu\text{s}$, welcher die Regelung ausreichend genau arbeiten lässt.

Signalsynthese

Die zur Anregung des Systems benötigten Testsignale werden neben den durchzuführenden Messungen vom Mikrocontroller generiert. Dazu ist es notwendig, diesen Prozess in das vorher beschriebene Taktregime einzubinden. Zu den Zeitpunkten, an denen die Messungen durchgeführt werden, ist es nicht möglich, ein Signal mittels des Digital-Analog-Wandlers zu erzeugen. Dies geschieht zeitlich versetzt, aber dennoch in äquidistanten Zeitabschnitten, während der Zähler der PLL aufwärts zählt (Bild 3.3). Erreicht dieser Zähler die Werte, welche einem Viertel (TGRC) bzw. drei Viertel (TGRD) des Endwertes (TGRA) entsprechen, löst er einen Interrupt zur Signalerzeugung aus, wird dabei aber nicht zurückgesetzt. Die Inhalte für die Zählregister werden von der PLL berechnet und eine Sinustabelle liefert die notwendigen Werte für den Digital-Analog-Wandler.

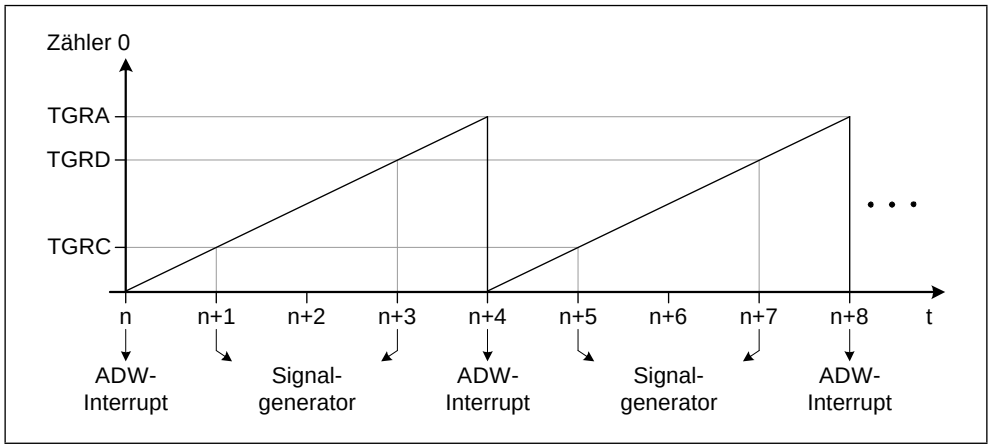


Bild 3.3: Interrupterzeugung für Signalsynthese

Look-Up-Table

Die Berechnung der notwendigen Cosinuswerte zu den angegebenen Zeitpunkten für das Testsignal dauert mit einem üblichen C-Programmierbefehl ca. 400 μs und belastet damit die Recheneinheit des Mikrocontroller unnötig. Ein Versuch mit Hilfe einer so genannten Look-Up-Table und einer linearen Interpolation zwischen den gespeicherten Werten, wie sie im Abschnitt E.8 als Quellcode dargestellt ist, brachte nur eine mäßige Verringerung der aufzuwendenden Zeit auf ca. 300 μs. Deshalb sollte eine Tabelle verwendet werden, welche nur die tatsächlich benötigten Werte ohne aufwändige Zwischenrechnungen bereitstellt. Da für alle weiteren Berechnungen mit den gemessenen Größen ebenfalls Cosinus- und Sinuswerte benötigt werden, wurden für jeden Zeitabschnitt an den Stellen $n, n+1, n+2, \cdots$ (vgl. Bild) Sinuswerte berechnet und in der Tabelle gespeichert (es ergeben sich 512 Einträge pro Bandumlauf). Da jedoch zusätzlich auch die erste Subharmonische der Bandumlauffrequenz zu beachten ist (Abschnitt 1.6), beinhaltet die Tabelle 1024 Einträge, welche ein Sinussignal mit der Frequenz $f_0 = f_{Band}/2$ abbilden. Die Periodendauer dieses Signals reicht über zwei Bandumläufe und stellt somit die Basisfunktion dar, aus welcher sich Funktionen mit einem ganzzahlig Vielfachen dieser Frequenz ableiten lassen. Dafür sorgt ein Index, der je nach gewählter Frequenz inkrementiert wird und so immer die benötigten Sinuswerte aus der Tabelle innerhalb einer Zeit von unter 5 μs liefert. Ein Cosinuswert wird aus der selben Tabelle mittels eines zweiten Index ermittelt, welcher einen Offset von 256 ($\pi/2$) aufweist.

3.2 Programmierung

Im folgenden Teil der Arbeit sollen einzelne Algorithmen und Funktionen des Quellcodes beschrieben werden, welcher für den Mikrocontroller mit Hilfe von [8] entwickelt wurde. Die Programmierung des Mikrocontrollers erfolgt mittels der Programmiersprache C. Da-

bei wird der Quellcode durch die frei verfügbaren Entwicklungswerkzeuge (Compiler, Linker,...) der 'GNU Toolchain' in Maschinencode übersetzt [9]. Dies erleichtert das Entwickeln und Testen (Debuggen) des Quellcodes erheblich, da der gesamte Programmablauf, wie es bei C üblich ist, in übersichtliche Funktionen unterteilt werden kann.
Nach dem Einschalten oder einem Reset erfolgt eine Hardwareinitialisierung des Mikrocontrollers wie sie im Anhang E.1 als Quellcode beschrieben ist. Dabei werden grundlegende Einstellungen bezüglich der Pinkonfiguration, der Arbeitsweise der einzelnen Zähler und des Analog-Digital-Wandlers sowie der Interruptprioritäten vorgenommen.

Phasengekoppelte Regelschleife

Durch die Freigabe des externen Interrupts des Bandimpulses wird gleichzeitig auch die PLL gestartet. Der Quellcode der PLL ist im Abschnitt E.2 aufgeführt. Durch die Messung des zeitlichen Abstands zwischen zwei Bandimpulsen mit Hilfe eines Zählers wird zu Beginn ein Basiswert für den PLL-Zähler ermittelt (#pragma interrupt INT_IRQ0). Dieser wird im Register TGRA gespeichert und anschließend erfolgt der Start des Zählers. Bei jedem erreichen des in TGRA gespeicherten Wertes wird ein Interrupt ausgelöst (#pragma interrupt INT_MTU0_TGIA0), welcher die Anzahl der Zählperioden durch das inkrementieren von index_t0 verfolgt. Nach 128 Perioden wird geprüft, ob dieser oder der Interrupt des Bandimpulses eher aufgetreten ist. Dazu wird in jeder Interruptserviceroutine (ISR) getestet, ob der Korrekturzähler schon aktiviert wurde (if ((P_MTU34.TSTR.BIT.CST & 0x2) == 0x0)). Wenn nicht, trat dieser prüfende Interrupt eher als der andere auf und startet demzufolge den Korrekturzähler. Der darauf folgende Interrupt hält den Korrekturzähler an und ermittelt seinen Zählerstand, welcher den Abstand zwischen den beiden Ereignissen darstellt (-zaehler_t1_tgia oder zaehler_t1_irq0). Die Serviceroutine ruft die Funktion pi_regler() auf und übergibt den ermittelten Zählwert. Der PI-Regler berechnet unter Verwendung der Trapezannäherung für den Integralteil einen neuen Wert für das Register TGRA (Abschnitt E.3). Es wird geprüft, ob sich die berechneten Zwischenwerte innerhalb von festgelegten Grenzen befinden und bevor das Register TGRA geladen wird und der Zähler startet werden zusätzlich noch die Werte für die Register TGRC und TGRD berechnet. Die Funktion und die Interruptserviceroutine wird nach dem Beenden dieser Berechnungen verlassen.

Signalsynthese

Wie schon zuvor beschrieben, wird zu den Zeitpunkten, an denen der PLL-Zähler die Werte der Register TGRC und TGRD erreicht, jeweils ein Interrupt ausgelöst (#pragma interrupt INT_MTU0_TGIC0 und #pragma interrupt INT_MTU0_TGID0). In Folge dessen wird die Funktion signalgenerator() (Abschnitt E.4) ausgeführt. In dieser bestimmt ein

Index welcher Wert aus der Sinustabelle für die Signalgenerierung verwendet werden soll. Dieser Index wird bei einer höheren gewählten Frequenz um ein ganzzahlig Vielfaches schneller inkrementiert und durchläuft damit, im Gegensatz zum einmaligen Durchlauf bei der Basisfrequenz f_0, die Tabelle mehrmals. Um eine Sinus-Periodizität gewährleisten zu können, wird der Index binär maskiert. Wird ein Feldindex berechnet, welcher größer als 1023 ist, wird durch diese Maske der Wert 1024 überdeckt (Bild 3.4). Der verbleibende Rest ist sogleich der aktuell zu verwendende Feldindex.

Mit Ausnahme des ersten Funktionsaufrufes wird der für den nächsten Interrupt benötigte Signalwert schon berechnet, sodass beim folgenden Funktionsaufruf dieser Wert nur noch an den DAW geleitet werden muss. Damit wird die Zeitspanne zwischen dem Auftreten des Interrupts und der Aktivierung des DAW erheblich verkürzt. Konvertiert von 32 Bit (float) in 16 Bit (short) und mit einem Offset des halben Wertebereiches versehen, wird dieser Wert an die Funktion PutDataOutDAC() übergeben (Abschnitt E.5). Der Offset wird benötigt, um trotz des unipolaren DAW ein bipolares Signal mit einer Amplitude von $\pm 2,5\,\mathrm{V}$ ausgeben zu können. In der aufgerufenen Funktion wird die 16 Bit breite Zahl in zwei 8 Bit-Zahlen getrennt und einzeln über die serielle Schnittstelle mit der Bezeichnung SCI 4 an den externen DAW gesendet. Die Auswahl zwischen mehreren möglichen externen Digital-Analog-Wandlern wird über das Setzen eines Chip-Select (CS) gesteuert.

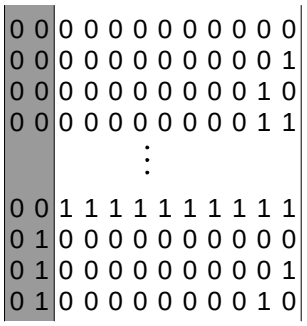


Bild 3.4: Modulo 1024

Ringspeicher

Die Berechnung des in Abschnitt 2.2 beschriebenen komplexen Fourier-Koeffizienten erfolgt im Mikrocontroller durch einen so genannten Ringspeicher und die Summenbildung über alle darin gespeicherten Werte (Abschnitt E.6). Dazu wird vom ADW nach jeder Wandlung ein Interrupt ausgelöst, in welchem die Ergebnisse mit einem entsprechenden Cosinus- und Sinuswert multipliziert und einzeln in eines von sechs Feldern mit jeweils 256 ($\widehat{=}$ zwei Bandumläufe) Elementen gespeichert werden. Für die Auswahl des Feldelements sorgt ein Index, welcher um 1 inkrementiert wird und bei einem Zählerstand von 256 wieder auf 0 gesetzt wird. Da immer nur die aktuellen Summen der Werte der letzten zwei Bandumläufe von Interesse sind, wird vor der Addition eines neuen Messwertes der Wert subtrahiert, welcher zwei Umläufe zuvor an exakt der gleichen Bandposition aufgezeichnet wurde.

Look-Up-Table

Die Generierung der Einträge dieser Tabelle kann im Rahmen der Hardwareinitialisierung des Mikrocontrollers erfolgen. Dieses Vorgehen hat aber den Nachteil, dass ein Feld mit 1024 Elementen erzeugt und in den internen Arbeitsspeicher abgelegt wird. Da die Einträge vom Typ float, d. h. 32 Bit-Zahlen sind, würde ein Speicherbedarf von $4\,\text{Byte} \cdot 1024 = 4\,\text{KByte}$ entstehen. Das entspricht einem Drittel des insgesamt zur Verfügung stehenden internen Arbeitsspeichers. Deswegen wurde ein Feld mit 1024 konstanten Elementen vom Typ float definiert und mit einem Vektor, der mit Hilfe von MATLAB generiert wurde, initialisiert. Dieser Vektor wurde in eine eigene Header-Datei gespeichert und wird beim Kompilieren des Quellcodes in diesen eingefügt. Damit wird erreicht, dass dieses Feld in den 256 KByte großen Festwertspeicher des Mikrocontroller geschrieben werden kann und die RAM-Ressourcen somit nicht belastet.

Berechnungen

Die bisher beschriebenen Teilfunktionen laufen permanent auf dem Mikrocontroller. Gestartet und untereinander synchronisiert werden sie durch auftretende Interrupte. Das bedeutet, dass z. B. die PLL und somit auch der ADW ständig arbeitet und aus den gemessenen Werten fortlaufend eine Summe gebildet wird. Darauf aufbauend, ist es möglich, eine Rahmenfunktion zu gestalten, welche einen vollständigen Messzyklus steuert und im Anschluss daran die Übertragungsfunktionen der Teilsysteme des Beschleunigers berechnet. Dazu müssen lediglich die Zeitdauer der Anregung des Systems definiert und die Zeitpunkte ausgewählt werden, an denen die jeweiligen Summen zur Berechnung herangezogen werden. In der Funktion ablauf() (Abschnitt E.7) ist eine Zählschleife definiert, welche die am Ende des Abschnitts 2.2 beschriebenen Aktionen für verschiedenen Frequenzen gleichsam wiederholt und die Zwischenergebnisse über die serielle Schnittstelle SCI 3 an einen angeschlossenen PC überträgt. Auf dem PC können diese Daten mit Hilfe des 'Windows Hyper Terminal' dargestellt werden.

4 Zusammenfassung und Ausblick

Um die gestellte Aufgabe dieser Masterarbeit bearbeiten zu können, war es ein erster Schritt, die Funktionsweise eines Ionenbeschleunigers, insbesondere des Van-de-Graaff-Beschleunigers im Forschungszentrum Rossendorf, zu erarbeiten. Neben theoretischen Betrachtungen geschah dies anhand einer Vielzahl von durchgeführten Messungen auch experimentell. Mit diesen gewonnenen Wissen wurde ein Softwaremodell mittels der Software MATLAB und mit der Simulationsumgebung SIMULINK erstellt, wie es im ersten Abschnitt der Arbeit beschrieben wird. Eine Verifikation des Modells wurde durch den Vergleich von zuvor aufgenommenen Messreihen mit den Simulationsergebnissen erreicht. Dabei lieferte das Softwaremodell schon während seiner Entwicklung Erkenntnisse über das Verhalten des Beschleunigers, welche dem Fachpersonal des Forschungszentrums noch nicht bekannt waren. Das hatte zur Folge, dass die bestehenden Vorstellungen über Teilfunktionen des Beschleunigers erweitert oder korrigiert werden mussten und die Vorgehensweise zur Bearbeitung der Aufgabenstellung mehrfach verändert wurde. Als eine Schwierigkeit stellte sich das Nachbilden des dynamischen Verhaltens des Systems heraus. Dieses wurde im Modell in weniger komplexe Teilfunktionen gegliedert und deren Verhalten durch geeignete Parameter beschrieben. Bis zur Fertigstellung der vorliegenden Arbeit war es jedoch noch nicht möglich, durch Berechnungen oder durch Simulation einen *vollständigen* Parametersatz zu erhalten, mit welchen es möglich ist, das dynamische Verhalten nachzubilden und zu dimensionieren. Die in den bisher durchgeführten Simulationen gewählten Parameter basieren auf Ansätzen der Koeffizientenbestimmung, welche dafür geeignet scheinen, aber noch nicht vollständig aufgelöst werden konnten. Die Simulationsergebnisse bestätigen im Vergleich mit den aufgenommenen Messreihen die gewählten Ansätze und zeigen eine Übereinstimmung im charakteristischen Verhalten des Ionenbeschleunigers.
In Abschnitt 2 der vorliegenden Arbeit wurde ein Algorithmus zur näherungsweisen Bestimmung von Übertragungsfunktionen der Teilsysteme theoretisch hergeleitet und in Abschnitt 3 wurde gezeigt, wie dieser in einen Mikrocontroller implementiert wurde. Das Ziel bestand darin, dem Beschleuniger im laufenden Betrieb (online) zusätzliche Testsignale zuzuführen, deren Auswirkungen zu messen und diese mit korrelativen Methoden auszuwerten. Dazu wurden mehrere Teilfunktionen programmiert, welche zum einen für die Steuerung des Mikrocontrollers und zum anderen für die Generierung von Testsignalen und für das Messen der Auswirkungen notwendig sind. In einem Rahmenprogramm werden die jeweiligen Messungen gesteuert und die ermittelten Werte für die darauf folgende Berechnung bereitgestellt. Diese einzelnen Funktionen sind in Abschnitt 3.2 aufgeführt und beschrieben. Eine erste Prüfung des implementierten Algorithmus zeigt, dass der für diese Aufgabe ausgewählte Mikrocontroller die benötigte Rechenleistung aufbringt, um den Algorithmus in Echtzeit abarbeiten zu können.

In zukünftigen Arbeitsschritten sollte der Mikrocontroller innerhalb eines realen Prozesses getestet werden. Die dafür notwendigen Bedingungen wurden innerhalb der vorliegenden Masterarbeit geschaffen. Ein ausstehender Punkt ist die Veränderung der Testsignale in Abhängigkeit des Signal-Rausch-Abstandes der gemessenen Größen. Diese Aufgabe könnte in gleichmäßigen zeitlichen Abständen zusätzlich vom Mikrocontroller übernommen werden. Ein dafür geeigneter Algorithmus und eine zweckmäßige Initialisierung der verwendeten Größen sollten mit Hilfe des erstellten Modells gefunden werden. Dafür muss das Modell um die im Mikrocontroller ablaufenden Algorithmen erweitert werden. In Vorbereitung dieser Erweiterung wurden im Rahmen der erstellten Arbeit die benötigten Schnittstellen im Modell eingerichtet und der theoretische Hintergrund ausgearbeitet.

5 Erklärung

Ich versichere an Eides statt, dass ich die beiliegende Masterarbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe.
Ferner gestatte ich der Hochschule für Technik und Wirtschaft Dresden (FH), die beiliegende Masterarbeit unter Beachtung insbesondere urheber-, datenschutz- und wettbewerbrechtlicher Vorschriften für Lehre und Forschung zu nutzen.
Mir ist bekannt, dass für die Weitergabe oder Veröffentlichung der Arbeit die Zustimmung der HTW Dresden (FH) sowie der an der Aufgabenstellung und Durchführung der Arbeit unmittelbar beteiligten Partnereinrichtungen erforderlich ist.

Holger Lange Dresden, im November 2004

Anhang A: Bilineare Transformation

Um zeitkontinuierliches Übertragungsverhalten in zeitdiskret simulierten Modellen nutzen zu können, ist es notwendig, diese in den $\mathcal{Z}$-Bereich zu transformieren. Aus mehreren verschiedenen Methoden wurde für das entworfene Modell die der bilinearen Transformation genutzt. Bei dieser Vorgehensweise kommt eine Substitution der komplexen Variablen s mit einem Ausdruck, der die komplexe Variable z beinhaltet, zum Einsatz. Die Herleitung der Methode wird in [10] beschrieben. Konkret lautet die Substitution

$$s = \frac{2}{T}\frac{z-1}{z+1}\,, \tag{A.1}$$

wobei T die Simulationsschrittweite angibt. Das bedeutet, dass die berechneten Werte in der diskreten Übertragungsfunktion abhängig von der Taktzeit sind und für jede Änderung dieser Zeit neu berechnet werden müssen. Eine Funktion in MATLAB [11] erleichtert dieses Vorgehen und lautet

```
[numd, dend] = c2dm(num, den, T, 'tustin') .
```

Die Funktion `continuous-to-discrete` benötigt als Eingabe das Zähler- und Nennerpolynom der kontinuierlichen Übertragungsfunktion, die als Vektoren `num` und `den` vorliegen müssen. Die Taktzeit `T` ist im entworfenen System stets 1 ms und die Option `'tustin'` bezeichnet die zu verwendende bilineare Transformationsmethode. Als Ergebnis erscheinen die Rückgabewerte wiederum als Vektoren, `numd` und `dend`.
Im folgenden werden alle im Modell vorkommenden Übertragungsfunktionen jeweils in ihrer zeitkontinuierlichen und zeitdiskreten Form angegeben. Es werden alle durch MATLAB darstellbaren Nachkommastellen angegeben, da sich bei der Simulation gezeigt hat, dass diese auf die Genauigkeit der Nachbildung des Verhaltens einen beträchtlichen Einfluss haben.

- Ladespannungsregler

$$G(s) = \frac{2000}{0.02\,s + 1}$$

$$G(z) = \frac{48.78048780487805\,z + 48.78048780487805}{z - 0.95121951219512} \tag{A.2}$$

- Capacitive-Pick-Up-Elektrode

$$G(s) = \frac{(2\pi \cdot 0.7/2700)\, s}{(2\pi \cdot 0.7)\, s}$$

$$G(z) = \frac{0.37032827067405 \cdot 10^{-3}\, z - 0.37032827067405 \cdot 10^{-3}}{z - 0.99977266163989} \tag{A.3}$$

- Koronatriodenverstärker

$$G(s) = \frac{1}{0.001\, s + 1}$$

$$G(z) = \frac{0.33\overline{3}\, z + 0.33\overline{3}}{z - 0.33\overline{3}} \tag{A.4}$$

Anhang B: Modellstrukturen in SIMULINK

B.1 Oberste Gliederungsebene

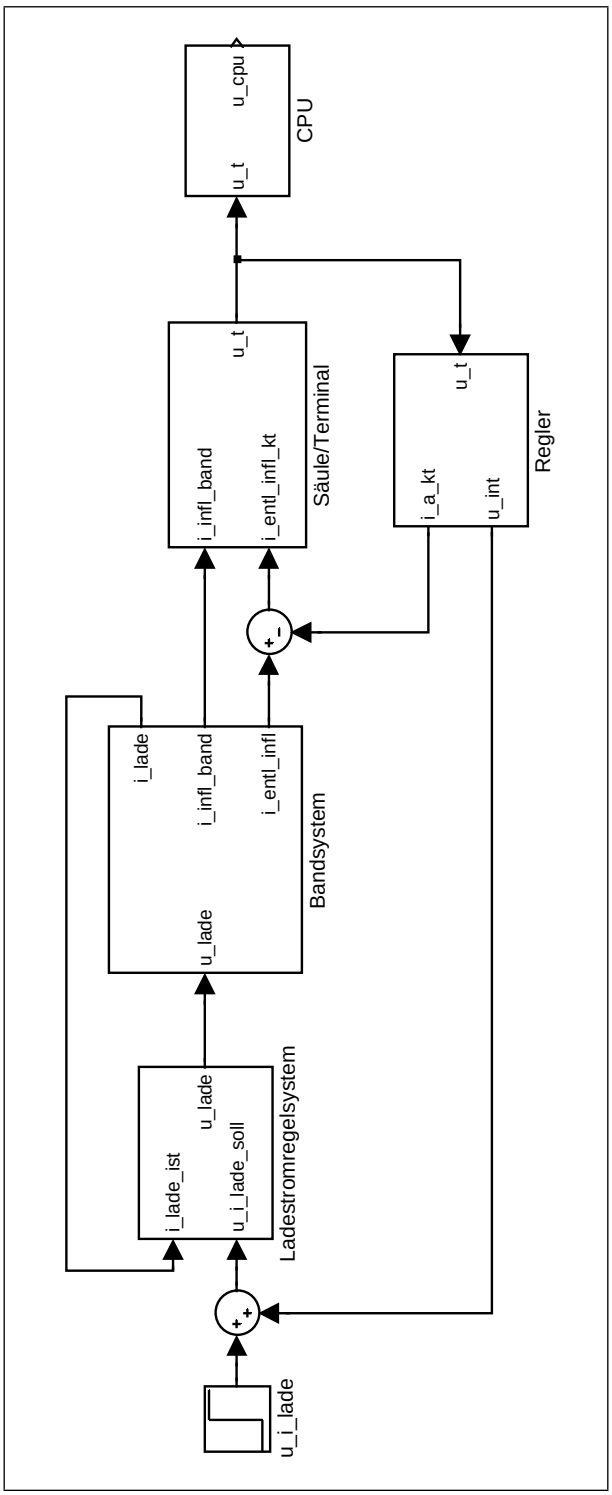


Bild B.1: Oberste Gliederungsebene des Gesamtsystems in SIMULINK

B.2 Bandsystem

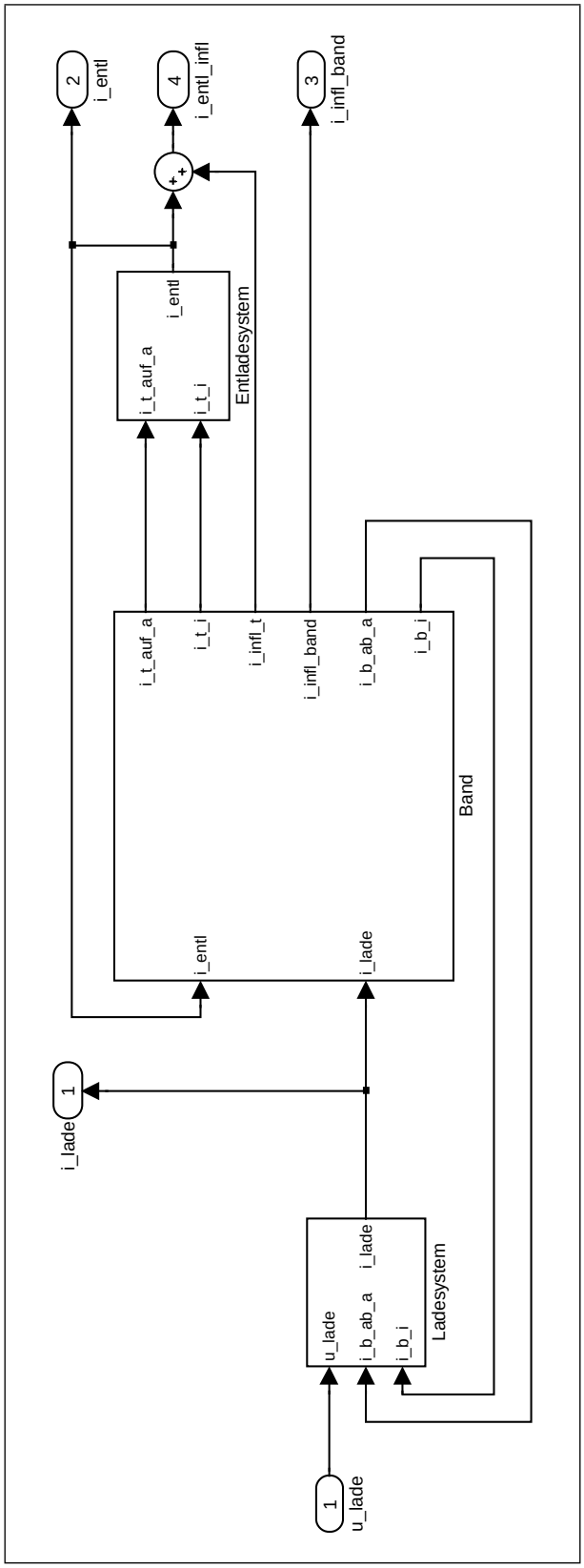


Bild B.2: Signalverbindungen des Bandsystems

B.3 Anordnung der Bandsegmente

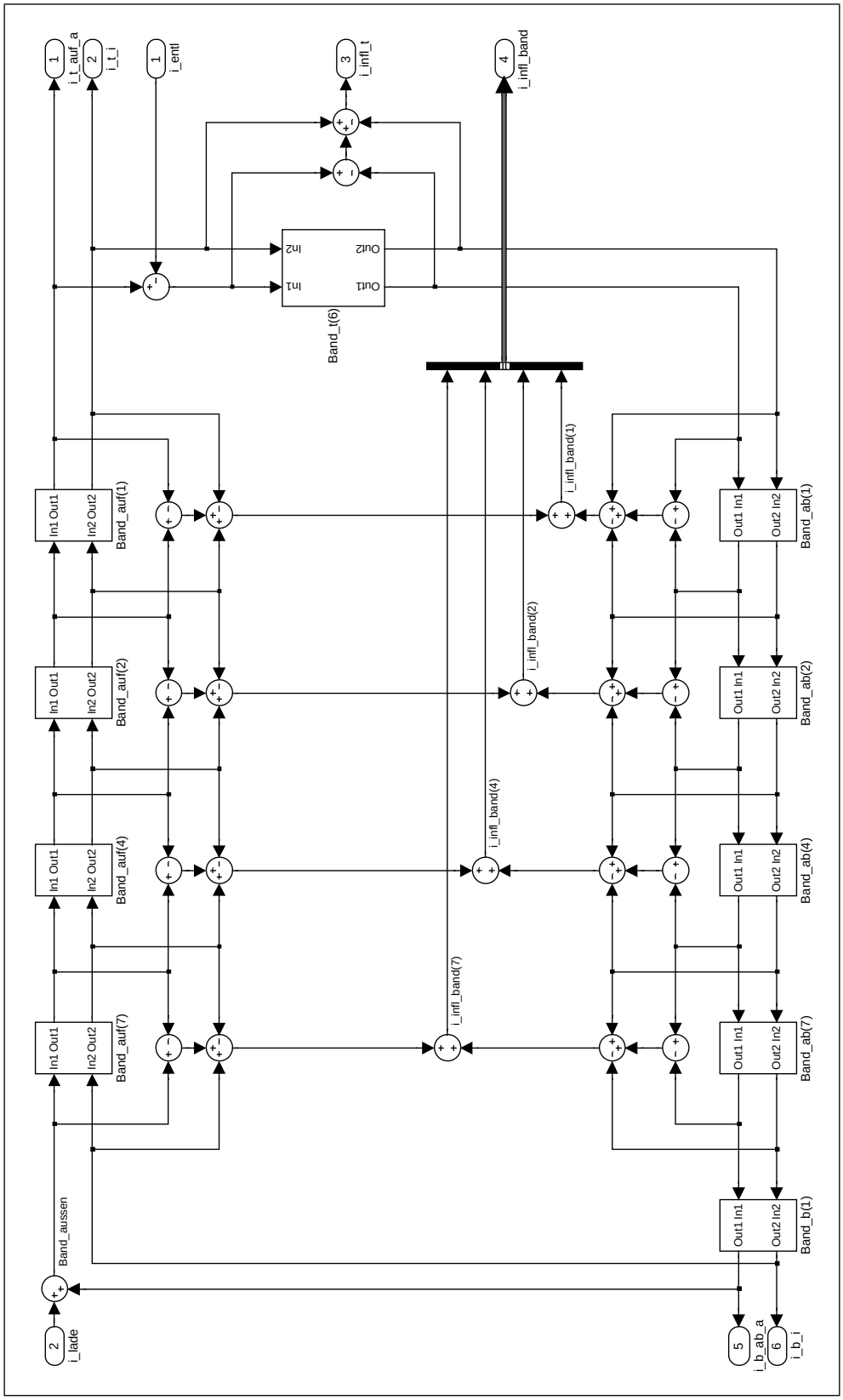


Bild B.3: Anordnung und Verteilung der Bandsegmente

B.4 Modellierung der Säule

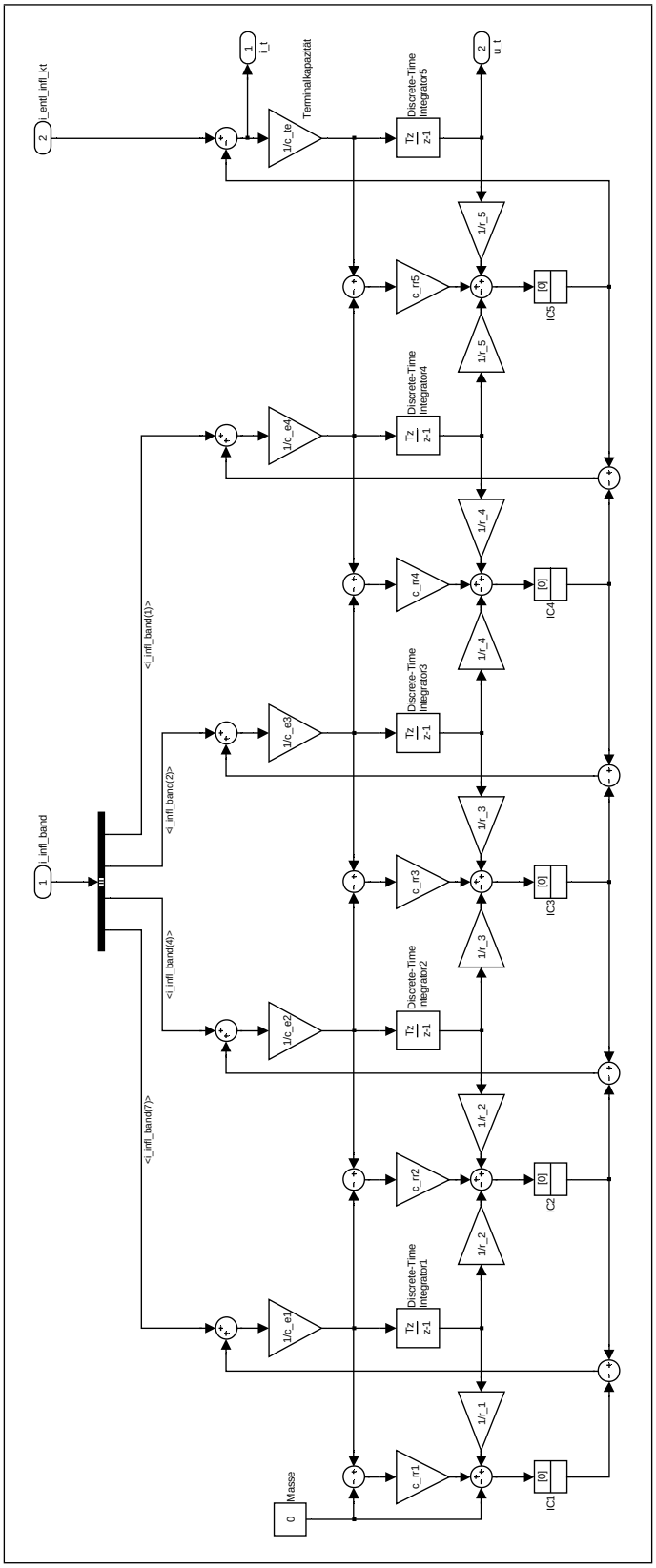


Bild B.4: Modellstruktur der Säule

B.5 Koronatriode

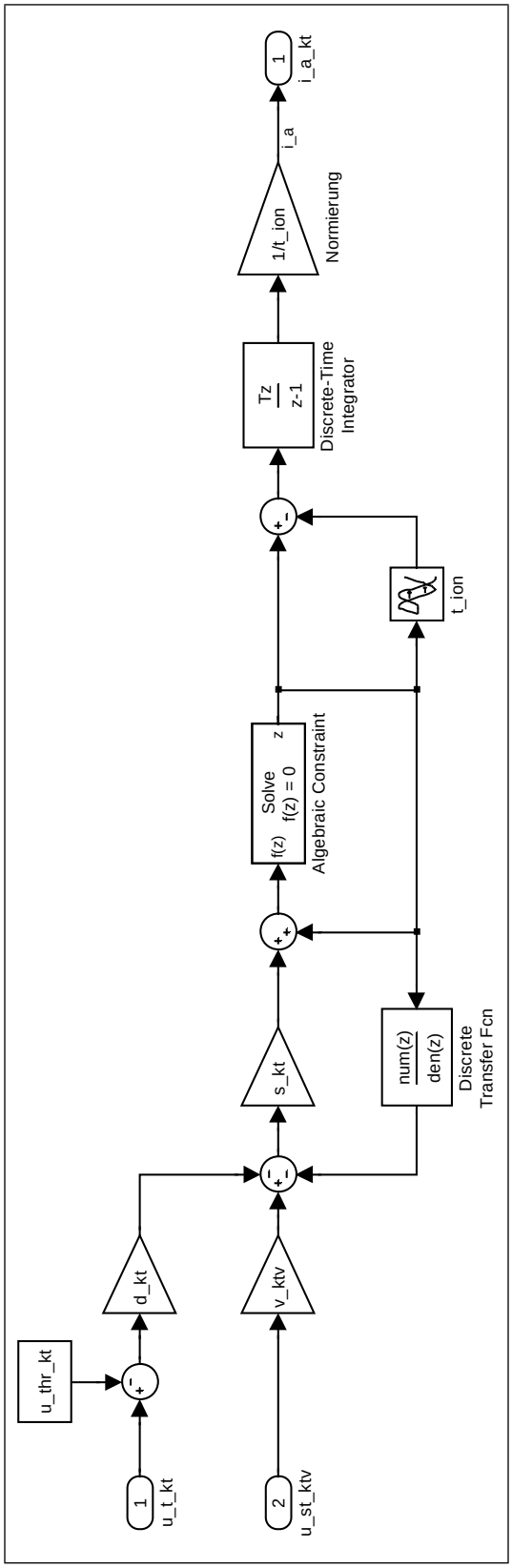


Bild B.5: Koronatriode inkl. Verstärker

B.6 Störstrom

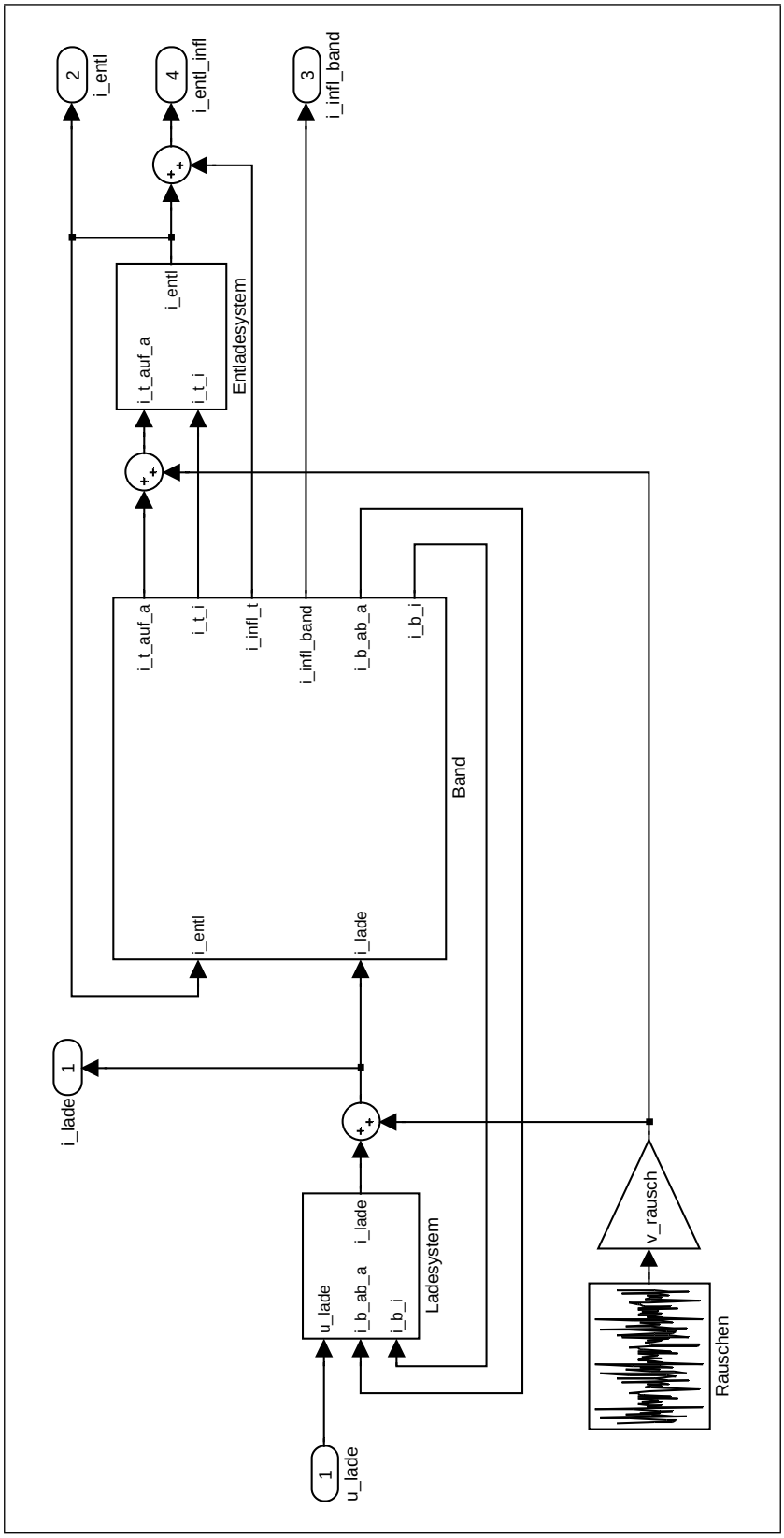


Bild B.6: Einspeisepunkte des Störstroms

Anhang C: Gleichungen der Übertragungsfunktionen

$$
\begin{aligned}
\mathit{Zähler\_F_{KT}} = \quad & \mathrm{Re}\{U_{T1}\} \cdot \mathrm{Re}\{U_{B2}\} - \mathrm{Im}\{U_{T1}\} \cdot \mathrm{Im}\{U_{B2}\} \\
- \quad & \mathrm{Re}\{U_{T2}\} \cdot \mathrm{Re}\{U_{B1}\} + \mathrm{Im}\{U_{T2}\} \cdot \mathrm{Im}\{U_{B1}\} \\
+ j\,(\quad & \mathrm{Re}\{U_{T1}\} \cdot \mathrm{Im}\{U_{B2}\} + \mathrm{Re}\{U_{B2}\} \cdot \mathrm{Im}\{U_{T1}\} \\
- \quad & \mathrm{Re}\{U_{T2}\} \cdot \mathrm{Im}\{U_{B1}\} - \mathrm{Re}\{U_{B1}\} \cdot \mathrm{Im}\{U_{T2}\} \,)
\end{aligned} \tag{C.1}
$$

$$
\begin{aligned}
\mathit{Zähler\_F_{B}} = \quad & \mathrm{Re}\{U_{T2}\} \cdot \mathrm{Re}\{U_{KT1}\} - \mathrm{Im}\{U_{T2}\} \cdot \mathrm{Im}\{U_{KT1}\} \\
- \quad & \mathrm{Re}\{U_{T1}\} \cdot \mathrm{Re}\{U_{KT2}\} + \mathrm{Im}\{U_{T1}\} \cdot \mathrm{Im}\{U_{KT2}\} \\
+ j\,(\quad & \mathrm{Re}\{U_{T2}\} \cdot \mathrm{Im}\{U_{KT1}\} + \mathrm{Re}\{U_{KT1}\} \cdot \mathrm{Im}\{U_{T2}\} \\
- \quad & \mathrm{Re}\{U_{T1}\} \cdot \mathrm{Im}\{U_{KT2}\} - \mathrm{Re}\{U_{KT2}\} \cdot \mathrm{Im}\{U_{T1}\} \,)
\end{aligned} \tag{C.2}
$$

$$
\begin{aligned}
\mathit{Nenner} = \quad & \mathrm{Re}\{U_{KT1}\} \cdot \mathrm{Re}\{U_{B2}\} - \mathrm{Im}\{U_{KT1}\} \cdot \mathrm{Im}\{U_{B2}\} \\
- \quad & \mathrm{Re}\{U_{KT2}\} \cdot \mathrm{Re}\{U_{B1}\} + \mathrm{Im}\{U_{KT2}\} \cdot \mathrm{Im}\{U_{B1}\} \\
+ j\,(\quad & \mathrm{Re}\{U_{KT1}\} \cdot \mathrm{Im}\{U_{B2}\} + \mathrm{Re}\{U_{B2}\} \cdot \mathrm{Im}\{U_{KT1}\} \\
- \quad & \mathrm{Re}\{U_{KT2}\} \cdot \mathrm{Im}\{U_{B1}\} - \mathrm{Re}\{U_{B1}\} \cdot \mathrm{Im}\{U_{KT2}\} \,)
\end{aligned} \tag{C.3}
$$

Anhang D: Entwicklungsumgebung des Mikrocontrollers

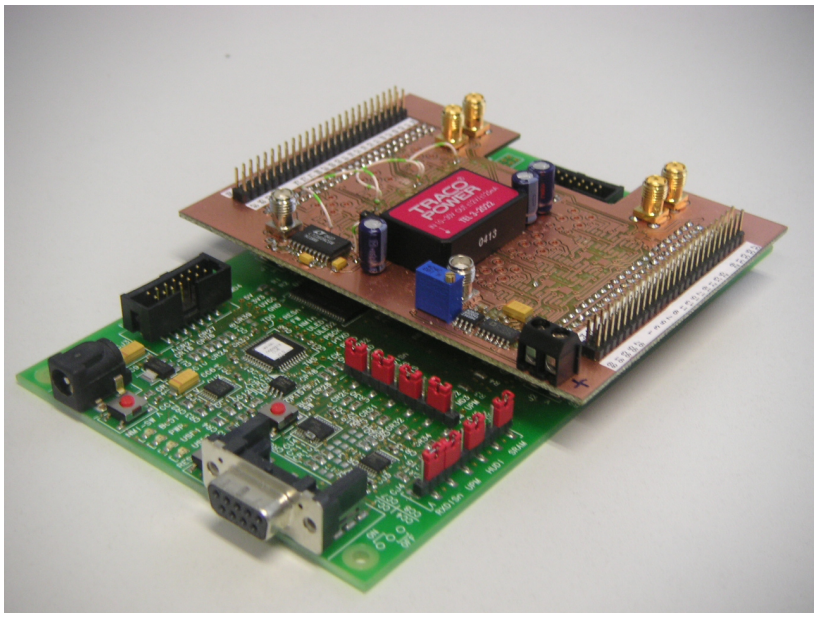


Bild D.1: EDK 7047 mit zusätzlichen Digital-Analog-Wandlern

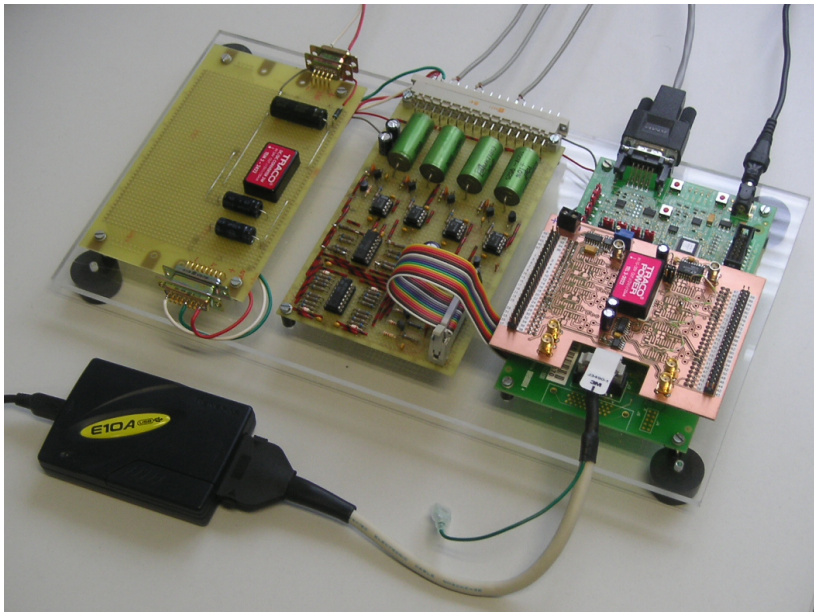


Bild D.2: Experimentierboard mit E10A-USB Emulator

Anhang E: Quellcode

E.1 Hardwareinitialisierung

```
void  hw_initialise (void)
{
    set_imask(0xF);                    // setzen der Interruptmaske auf 15 in CPU-SR[I3...I0] (Seite:13)

/**********************************
* PLL
**********************************/

    //-------- externer Interruptpin IRQ0 (Pin 71)

    P_INTC.ICR1.BIT.IRQ0S  = 1;        // externer Interrupt an IRQ0 reagiert auf Signalflanken (Seite:68)
    P_INTC.ICR2.BIT.IRQ0ES = 01;       // externer Interrupt an IRQ0 reagiert auf steigende SF (Seite:69)
    P_INTC.IPRA.BIT.IRQ0   = 0xD;      // Interruptlevel von IRQ0 = 13 (Seite:72)

    //-------- Timer für PLL

    P_STBY.MSTCR2.BIT.MSTP13 = 0;      // Aktivieren des Timermoduls(MTU) (Seite:576)
    P_MTU34.TSTR.BYTE   = 0x00;        // Stoppen aller Timer im TSTR (Seite:176)
    P_MTU0.TSR_0.BYTE   = 0xC0;        // löschen aller Timer0 interrupt flags (Seite:172)
    P_MTU1.TSR_1.BYTE   = 0xC0;        // löschen aller Timer1 interrupt flags (Seite:172)
    P_INTC.IPRD.BIT.TGI_0  = 0xD;      // Interruptlevel von TGRA0 = 13 (Seite:72)

    //-------- Timer0 zur Erzeugung der 128 Takte

    P_MTU0.TMDR_0.BYTE  = 0xc2;        // Timer0 im PWM mode 1 (Seite:150)
    P_MTU0.TCR_0.BYTE   = 0x22;        // TGRA0 setzt Zähler zurück (compare match)
                                       // steigende Flanke des Zählerclock
                                       // clockfreq.-->phi0/16 (Seite:146;149;165)
    P_MTU0.TIER_0.BIT.TGIEA = 1;       // Interrupt für TGRA_0 freischalten (Seite:171)
    P_MTU0.TIER_0.BIT.TGIEC = 1;       // Interrupt für TGRC_0 freischalten (Seite:171)
    P_MTU0.TIER_0.BIT.TGIED = 1;       // Interrupt für TGRD_0 freischalten (Seite:171)


    P_MTU0.TIORH_0.BYTE = 0x52;        // Signal an TIOC0A (PIN2) bei compare match (Seite:155)
                                       // TGRA0 setzt Pin auf '1'; TRGB0 setzt Pin auf '0'

    P_MTU0.TCNT_0       = 0x0000;      // Zählerstand von T0 löschen
    P_MTU0.TGRA_0       = 0x0000;      // TGRA0 auf 0 (Seite:175)
    P_MTU0.TGRB_0       = 0x0080;      // TGRB0 für Impulsweite des 128er Zählers (Seite:175)
    P_MTU0.TGRC_0       = 0x0000;      // TGRC0 für Zeitpunkte des Sinusgenerators(Seite:175)
    P_MTU0.TGRD_0       = 0x0000;      // TGRD0 für Zeitpunkte des Sinusgenerators(Seite:175)


    //-------- Timer1 zur Messung der Phasendifferenz

    P_MTU1.TMDR_1.BYTE  = 0xc0;        // alle Timerregister von T1 im Normalmodus (Seite:150)
    P_MTU1.TCR_1.BYTE   = 0x03;        // Zähler wird nicht automatisch zurückgesetzt
                                       // steigende Flanke des Zählerclock
                                       // Taktfrequenz phi0/64(Seite:146;149;163)
    P_MTU1.TCNT_1       = 0x0000;      // Zählerstand von T1 löschen


/**********************************
* SCI3 Setup
**********************************/

    P_STBY.MSTCR1.BIT.MSTP19 = 0;      // Aktivieren der SCI3 (Seite:575)
    P_SCI3.SCR.BYTE = 0x00;            // Senden und Empfangen an SCI3 gestoppt (Seite:319)
    P_SCI3.SSR.BYTE &= 0x00;           // löschen aller Flags (Seite:321)
    P_SCI3.SMR.BYTE = 0x00;            // Asynchronous mode; P_NONE;1;8 (Seite:318)
    P_SCI3.BRR      = 5;               // Baudrate (Seite:324)
    P_SCI3.SCR.BYTE = 0x20;

/**********************************
* SCI4 Setup
**********************************/

    P_STBY.MSTCR1.BIT.MSTP20 = 0;      // Aktivieren der SCI4 (Seite:575)
```

```
    P_SCI4.SCR.BYTE = 0x00;              // Senden und Empfangen an SCI4 gestoppt (Seite: 319)
                                         // SCK4 als clock output
    P_SCI4.SSR.BYTE &= 0x00;             // löschen aller Flags (Seite: 321)
    P_SCI4.SMR.BYTE = 0x80;              // Clocked synchronous mode (Seite: 318)
    P_SCI4.SDCR.BYTE = 0xFA;             // MSB wird zuerst geladen (Seite: 324)
    P_SCI4.BRR      = 5;                 // Baudrate (Seite: 324)


/**********************************
* ADC
**********************************/

    P_STBY.MSTCR2.BIT.MSTP4 = 0;         // Aktivieren des ADC0 (A/D0) (Seite:576)
    P_MTU0.TIER_0.BIT.TTGE = 1;          // Interrupt für ADC von TGRA0 (Seite:170)
    P_INTC.IPRG.BIT.AD01 = 0xA;          // Interruptlevel von ADC0 und ADC1 = 10 (Seite:72)

    P_AD.ADCR_0.BIT.ADST = 0;            // Stoppen des ADC0 (Seite:369)
    P_AD.ADCR_0.BYTE     = 0xE7;         // Trigger; single-cycle-scan (Seite:369)
    P_AD.ADTSR.BYTE      = 0x02;         // Trigger von MTU (Seite:370)
    P_AD.ADCSR_0.BYTE    = 0x5A;         // Interrup enable; 4-channel scan mode (Seite:367)


/**********************************
* Pin Function Controller
**********************************/

    //-------- Pin 2 (TIOC0A) für Timer0

    P_PORTE.PECRL2.BIT.PE0MD = 01;       // Multiplexer an Port PE0 ist auf TIOC0A (Seite:504)
    P_PORTE.PEIORL.BIT.PE0IOR = 1;       // Port PE0 (Pin 2) als Ausgang (Seite:502)

    //-------- Pin 4 für 128* Timer0

    P_PORTE.PECRL2.BIT.PE2MD = 00;       // Multiplexer an Port PE2 ist auf PE2 (Seite:504)
    P_PORTE.PEIORL.BIT.PE2IOR = 1;       // Port PE2 (Pin 4) als Ausgang (Seite:502)

    //-------- Pin 6 für Tests

    P_PORTE.PECRL2.BIT.PE4MD = 00;       // Multiplexer an Port PE4 ist auf PE4 (Seite:504)
    P_PORTE.PEIORL.BIT.PE4IOR = 1;       // Port PE4 (Pin 6) als Ausgang (Seite:502)

    //-------- Pin 71 (IRQ0) als Interrupteingang

    P_PORTA.PACRL3.BIT.PA2MD2 = 0;       // Multiplexer an Port PA2 ist auf IRQ0 (Seite:498)
    P_PORTA.PACRL2.BIT.PA2MD = 3;
    P_PORTA.PAIORL.BIT.PA2IOR = 0;       // Port PA2 als Eingang (Seite:495)

    //-------- Pin 18 (LED1) und Pin 20 (LED2)

    P_PORTE.PECRL1.BIT.PE12MD = 00;      // Multiplexer an Port PE12 ist auf PE12 (Seite:504)
    P_PORTE.PEIORL.BIT.PE12IOR = 1;      // Port PE12 (LED1) als Ausgang (Seite:502)

    P_PORTE.PECRL1.BIT.PE14MD = 00;      // Multiplexer an Port PE14 ist auf PE14 (Seite:504)
    P_PORTE.PEIORL.BIT.PE14IOR = 1;      // Port PE14 (LED2) als Ausgang (Seite:502)

    //-------- Serielle Schnittstelle SCI3

    P_PORTA.PACRL3.BIT.PA9MD2 = 1;       // SCI3 Pins freischalten (Seite:497)
    P_PORTA.PACRL1.BIT.PA9MD = 1;
    P_PORTA.PACRL3.BIT.PA8MD2 = 1;
    P_PORTA.PACRL1.BIT.PA8MD = 1;

    //-------- Serielle Schnittstelle SCI4

    P_PORTB.PBCR1.BIT.PB4MD2 = 1;        // SCI4/SCK4 als Funktion an Pin 52 (Seite: 497)
    P_PORTB.PBCR2.BIT.PB4MD = 11;
    P_PORTB.PBIOR.BIT.PB4IOR = 1;        // Port PB4 (SCK4) als Ausgang (Pin 52)(Seite: 499)

    P_PORTB.PBCR1.BIT.PB3MD2 = 1;        // SCI4/TxD4 als Funktion an Pin 54 (Seite: 497)
    P_PORTB.PBCR2.BIT.PB3MD = 11;

    P_PORTB.PBCR1.BIT.PB2MD2 = 1;        // SCI4/RxD4 als Funktion an Pin 55 (Seite: 497)
    P_PORTB.PBCR2.BIT.PB2MD = 11;

    //-------- ext. DAC/ADC

    P_PORTE.PECRL1.BIT.PE8MD = 00;
    P_PORTE.PEIORL.BIT.PE8IOR = 1;       // Port PE8 (Pin 10) als Ausgang (DAC_CS1) (Seite:502)
```

```
    P_PORTE.PEDRL.BIT.PE8DR = 1;        // DAC_CS1 auf '1'

    P_PORTE.PECRL1.BIT.PE9MD = 00;
    P_PORTE.PEIORL.BIT.PE9IOR = 1;      // Port PE9 (Pin 12) als Ausgang (DAC_CS2) (Seite:502)
    P_PORTE.PEDRL.BIT.PE9DR = 1;        // DAC_CS2 auf '1'

    P_PORTE.PECRL1.BIT.PE11MD = 00;
    P_PORTE.PEIORL.BIT.PE11IOR= 1;      // Port PE11 (Pin 17) als Ausgang (DAC_CS3) (Seite:502)
    P_PORTE.PEDRL.BIT.PE11DR = 1;       // DAC_CS3 auf '1'

    P_PORTE.PECRL1.BIT.PE13MD = 00;
    P_PORTE.PEIORL.BIT.PE13IOR= 1;      // Port PE13 (Pin 19) als Ausgang (DAC_CS4) (Seite:502)
    P_PORTE.PEDRL.BIT.PE13DR = 1;       // DAC_CS4 auf '1'

    P_PORTE.PECRL1.BIT.PE15MD = 00;
    P_PORTE.PEIORL.BIT.PE15IOR= 1;      // Port PE15 (Pin 21) als Ausgang (LDAC) (Seite:502)
    P_PORTE.PEDRL.BIT.PE15DR = 1;       // LDAC auf '1'

    P_PORTD.PDCRL1.BIT.PD8MD0= 0;
    P_PORTD.PDCRL2.BIT.PD8MD1= 0;
    P_PORTD.PDIORL.BIT.PD8IOR= 1;       // Port PD8 (Pin 76) als Ausgang (ADC_RC) (Seite:502)
    P_PORTD.PDDRL.BIT.PD8DR= 1;         // ADC_RC auf '1'

    P_PORTB.PBCR1.BIT.PB5MD2 = 0;
    P_PORTB.PBCR2.BIT.PB5MD = 00;
    P_PORTB.PBIOR.BIT.PB5IOR = 1;       // Port PB5 (Pin 51) als Ausgang (ADC_CS) (Seite: 499)
    P_PORTB.PBDR.BIT.PB5DR= 0;          // ADC_CS auf '0'
}
```

E.2 PLL

```
#pragma interrupt INT_IRQ0                          // Interrupt des Bandumlaufimpuls
void INT_IRQ0(void)
{
    long  zaehler_t1_irq0 ;
    static short  start_t0  = 0;

    switch ( start_t0 )
    {
    case 0 :     start_t0  = 1;                     // erster Impuls wird vernachlässigt
                 break;


    case 1 :     P_MTU34.TSTR.BIT.CST |= 0x2;       // Zähler T1 starten mit Takt von phi0/64
                 start_t0  = 2;
                 break;


    case 2 :     P_MTU34.TSTR.BIT.CST &= 0x5;       // Zähler T1 stoppen
                 zaehler_t1_irq0  = P_MTU1.TCNT_1; // Zählerstand von Timer1 übernehmen

                 if (P_MTU1.TSR_1.BIT.TCFV = 1) // Overflowflag T1
                 {   P_MTU0.TGRA_0 = (zaehler_t1_irq0 + 0xFFFF)/32;} // Überlauf
                 else
                 {   P_MTU0.TGRA_0 = (zaehler_t1_irq0)/32;}

                 P_MTU1.TCR_1.BYTE = 0x02;          // Taktfrequenz von T1 auf phi0/16 erhöhen
                 P_MTU1.TCNT_1 = 0x0000;            // Zählerstand von T1 auf Null setzen
                 P_MTU1.TSR_1.BYTE = 0xC0;          // alle FLags löschen
                 start_t0  = 3;
                 break;


    case 3 :     P_MTU34.TSTR.BIT.CST |= 0x1;       // Start von Timer0 (Seite:176)
                 start_t0  = 4;
                 break;


    default :    zaehler_t1_irq0  = P_MTU1.TCNT_1; // Zählerstand von Timer1 übernehmen
                 P_INTC.ISR.BIT.IRQ0F = 0;          // IRQ0 Flag löschen ( Seite:71)

                 if ((P_MTU34.TSTR.BIT.CST & 0x2) == 0x0)// Zähler T1 läuft noch nicht (Seite:176)
                 {
                     P_MTU1.TSR_1.BYTE = 0xC0;  // alle Flags löschen
                     P_MTU1.TCNT_1 = 0x0000;    // Zählerstand von T1 auf Null setzen
                     P_MTU34.TSTR.BIT.CST |= 0x2; // Zähler T1 starten
                 }
                 else
                 {
                     P_MTU34.TSTR.BIT.CST &= 0x5; // Zähler T1 stoppen

                      pi_regler ( zaehler_t1_irq0 );     // Berechnung

                     P_MTU1.TCNT_1 = 0x0000;    // Zählerstand von T1 auf Null setzen
                     P_MTU1.TSR_1.BYTE = 0xC0;  // alle Flags löschen
                 }
                 bandumlaufzaehler ++;
                 break;
    }//switch ( start_t0 )

}//INT_IRQ0(void)
```

```
#pragma interrupt INT_MTU0_TGIA0            // Timer0 Interrupt (TGRA0 compare match)
void INT_MTU0_TGIA0(void)
{
    long  zaehler_t1_tgia ;
    static short index_t0 = 0;

     zaehler_t1_tgia  = P_MTU1.TCNT_1;      // Zählerstand von Timer1 übernehmen
    P_MTU0.TSR_0.BYTE &= 0xFE;              // output compare Flag TGFA0 löschen (Seite:174)

    if (index_t0 == 0x007F)                 // hat Zähler T0 128 (0-127) Zählerdurchläufe?
    {
        P_MTU34.TSTR.BIT.CST &= 0x6;        // Stopp von Timer0 (Seite:176)
        P_MTU0.TCNT_0 = 0x0000;             // Zählerstand von T0 löschen

        P_PORTE.PEDRL.BIT.PE2DR ^= 1;       // Port PE2 (Pin 4) (Seite:515)

        if ((P_MTU34.TSTR.BIT.CST & 0x2) == 0x0)// läuft Zähler T1 ? (Seite:176)
        {
            P_MTU1.TSR_1.BYTE = 0xC0;       // alle FLags löschen
            P_MTU1.TCNT_1 = 0x0000;         // Zählerstand von T1 auf Null setzen
            P_MTU34.TSTR.BIT.CST |= 0x2;    // Zähler T1 starten
        }
        else
        {
            P_MTU34.TSTR.BIT.CST &= 0x5;    // Zähler T1 stoppen

             pi_regler (- zaehler_t1_tgia );// Berechnung

            P_MTU1.TCNT_1 = 0x0000;         // Zählerstand von T1 auf Null setzen
            P_MTU1.TSR_1.BYTE = 0xC0;       // alle FLags löschen
        }
        index_t0 = 0;                       // Index zurücksetzen
    }
    else
    {   index_t0 ++;}                       // Index inkrementieren

}//INT_MTU0_TGIA0
```

E.3 PI-Regler

```
void pi_regler (long e_t1)
{
    short sinustakt1 , sinustakt2 ;

    long y_gesamt;

    float y_proportional , y_integral ;

    static float y_integral_mem = 0, e_t1_v = 0, reset_index_cos_sin = 0;

    const float K_INTEGRAL = 0.0003, K_PROPORTIONAL = 0.003;

    const float INTEGRAL_GRENZE_OBEN = 15, INTEGRAL_GRENZE_UNTEN = -15;

    const float Y_GRENZE_OBEN = 15, Y_GRENZE_UNTEN = -15;

    const short T0_GRENZE_OBEN = 2200, T0_GRENZE_UNTEN = 1600;

    y_proportional = e_t1 * K_PROPORTIONAL;                    // Proprtionalanteil

    y_integral = (e_t1 + e_t1_v ) * 0.5 * K_INTEGRAL + y_integral_mem;   // Integralanteil (Trapezregel)

    if ( y_integral < INTEGRAL_GRENZE_UNTEN) // Grenzen des Integralanteils
    {   y_integral = INTEGRAL_GRENZE_UNTEN;}
    else
    {   if ( y_integral > INTEGRAL_GRENZE_OBEN)
        {   y_integral = INTEGRAL_GRENZE_OBEN;}
        else
        {   y_integral = y_integral ;}
    }

    y_integral_mem = y_integral ;
    e_t1_v = e_t1;

    y_gesamt = (long)(round( y_proportional + y_integral ));

    if (y_gesamt < Y_GRENZE_UNTEN)              // Grenzen der Stellgröße
    {   P_MTU0.TGRA_0 += Y_GRENZE_UNTEN;}
    else
    {   if (y_gesamt > Y_GRENZE_OBEN)
        {   P_MTU0.TGRA_0 += Y_GRENZE_OBEN;}
        else
        {   P_MTU0.TGRA_0 += y_gesamt;}
    }

    if (P_MTU0.TGRA_0 < T0_GRENZE_UNTEN)   // Grenzen des Zählers T0
    {   P_MTU0.TGRA_0 = T0_GRENZE_UNTEN;
        P_PORTE.PEDRL.BIT.PE12DR = 1;}         // Port PE12 (Pin 18) (LED1 anschalten) (Seite:515)
    else
    {   if (P_MTU0.TGRA_0 > T0_GRENZE_OBEN)
        {   P_MTU0.TGRA_0 = T0_GRENZE_OBEN;
            P_PORTE.PEDRL.BIT.PE12DR = 1;}     // Port PE12 (Pin 18) (LED1 anschalten) (Seite:515)
        else
        {   P_MTU0.TGRA_0 = P_MTU0.TGRA_0;
            P_PORTE.PEDRL.BIT.PE12DR = 0;}     // Port PE12 (Pin 18) (LED1 ausschalten)(Seite:515)
    }

    sinustakt1 = P_MTU0.TGRA_0 /4;              // Zählerstand für Takt des Signalgenerators
    sinustakt2 = sinustakt1 *3;                 // Zählerstand für Takt des Signalgenerators

    P_MTU0.TGRC_0 = sinustakt1;
    P_MTU0.TGRD_0 = sinustakt2;

    //----- Generatorsignal
     reset_index_cos_sin ++;
    if ( reset_index_cos_sin == 2)              // Index wird aller zwei Bandumläufe
        {                                       // zurückgesetzt (Synchronisation)
            index_cos = COS_INDEX;
            index_sin = SIN_INDEX;
             reset_index_cos_sin = 0;
        }

    P_MTU0.TCNT_0 = 0x0000;                     // Zählerstand von T0 löschen
    P_MTU34.TSTR.BIT.CST |= 0x1;                // Start von Timer0 (Seite:176)
}// pi_regler ()
```

E.4 Signalsynthese

```
#pragma interrupt INT_MTU0_TGIC0      // ISR für INT_MTU0_TGIC0 (compare match)
void INT_MTU0_TGIC0(void)
{
    P_MTU0.TSR_0.BYTE &= 0xFB;        // output compare Flag TGFC0 löschen (Seite:173)
     signalgenerator ();              // Wert aus Sinustabelle über DAC ausgeben
}


#pragma interrupt INT_MTU0_TGID0      // ISR für MTU0_TGID0 (compare match)
void INT_MTU0_TGID0(void)
{
    P_MTU0.TSR_0.BYTE &= 0xF7;        // output compare Flag TGFD0 löschen (Seite:173)
     signalgenerator ();              // Wert aus Sinustabelle über DAC ausgeben
}


void  signalgenerator (void)
{
    static  short  k = 0;
    static  short  DataOut =0;

    float  ampl = 0.5;
    float  funktionswert ;

    switch (k)
    {
    case 0:  index_cos = index_cos + 2 *n_f0 ;          // Index inkrementieren
             index_sin = index_sin + 2 *n_f0 ;

             funktionswert = sinus( index_cos );        // Wert aus Sinustabelle holen
             funktionswert *= ampl;

             DataOut = (short)(funktionswert /LSB);     // Wert für DAC mit LSB berechnen
             DataOut += 0x8000;

             if (generator_dac_on == 1)
                 {   PutDataOutDAC(DataOut, dac_channel);}
             k++;

    default :     if (generator_dac_on == 1)
                      {   PutDataOutDAC(DataOut, dac_channel);}
    }

    index_cos = index_cos + 2 *n_f0 ;                   // Index inkrementieren
    index_sin = index_sin + 2 *n_f0 ;

    index_cos &= 0x03FF;                                // Maske für Modulo 1024
    index_sin &= 0x03FF;                                // damit schaltet Index nach einer Periode
                                                        // immer wieder auf Null
    funktionswert = sinus( index_cos );                 // Wert aus Sinustabelle holen
    funktionswert *= ampl;

    DataOut = (short)(funktionswert /LSB);              // Wert für DAC mit LSB berechnen
    DataOut += 0x8000;                                  // Offset für unipolare Darstellung
}
```

E.5 Ausgabe über den DAW

```
void PutDataOutDAC(short DataOut, short dac_switch)
{
    short DataOut_HI, DataOut_LO;

    DataOut_HI = HIBYTE(DataOut);
    DataOut_LO = LOBYTE(DataOut);

    P_SCI4.SCR.BYTE |= 0x20;              // SCI4-Senden starten (Seite: 319)

    // ------ HIBYTE senden ------------------------------

    while ( !( P_SCI4.SSR.BIT.TDRE) )     // warten bis TDRE=1 (Seite: 321)
    {   ;   }

    switch (dac_switch)                   // Auswahl des DAC
    {
    case 1 :    P_PORTE.PEDRL.BIT.PE8DR = 0;  // CS1 schalten Port PE8 (Pin 10) (Seite: 515)
                break;
    case 2 :    P_PORTE.PEDRL.BIT.PE9DR = 0;  // CS2 schalten Port PE9 (Pin 12) (Seite: 515)
                break;
    case 3 :    P_PORTE.PEDRL.BIT.PE11DR = 0; // CS3 schalten Port PE11 (Pin 17)(Seite: 515)
                break;
    case 4 :    P_PORTE.PEDRL.BIT.PE13DR = 0; // CS4 schalten Port PE13 (Pin 19)(Seite: 515)
                break;
    default :   P_PORTE.PEDRL.BIT.PE8DR = 0;  // CS1 schalten Port PE8 (Pin 10) (Seite: 515)
                break;
    }

    P_SCI4.TDR = DataOut_HI;              // HIBYTE in Register laden (Seite: 317)

    P_SCI4.SSR.BIT.TDRE = 0;              // Datenfreigabe für Übertragung (Seite: 321)


    // ------ LOBYTE senden ------------------------------

    while ( !( P_SCI4.SSR.BIT.TDRE) )     // warten bis TDRE=1 (Seite: 321)
    {   ;   }

    P_SCI4.TDR = DataOut_LO;              // LOBYTE in Register laden (Seite: 317)

    P_SCI4.SSR.BIT.TDRE = 0;              // Datenfreigabe für Übertragung (Seite: 321)

    while ( !( P_SCI4.SSR.BIT.TEND) )     // warten bis Daten gesendet worden sind (Seite: 323)
    {   ;   }

    P_SCI4.SCR.BYTE &= 0xDF;              // SCI4-Senden stoppen (Seite: 319)

    switch (dac_switch)                   // Auswahl des DAC
    {
    case 1 :    P_PORTE.PEDRL.BIT.PE8DR = 1;  // CS1 schalten Port PE8 (Pin 10) (Seite: 515)
                break;
    case 2 :    P_PORTE.PEDRL.BIT.PE9DR = 1;  // CS2 schalten Port PE9 (Pin 12) (Seite: 515)
                break;
    case 3 :    P_PORTE.PEDRL.BIT.PE11DR = 1; // CS3 schalten Port PE11 (Pin 17)(Seite: 515)
                break;
    case 4 :    P_PORTE.PEDRL.BIT.PE13DR = 1; // CS4 schalten Port PE13 (Pin 19)(Seite: 515)
                break;
    default :   P_PORTE.PEDRL.BIT.PE8DR = 1;  // CS1 schalten Port PE8 (Pin 10) (Seite: 515)
                break;
    }

    P_PORTE.PEDRL.BIT.PE15DR = 0;         // LDAC schalten Port PE15 (Pin 21) (Seite: 515)
    P_PORTE.PEDRL.BIT.PE15DR = 1;
}
```

E.6 Ringspeicher

```
#pragma interrupt INT_ADI0                    // Interrupt des ADC0
void INT_ADI0(void)
{
    static short i = 0;

    short analyse_index_cos = 0, analyse_index_sin = 0;

    static float summe_ks_cos = 0, summe_cs_cos = 0, summe_ls_cos = 0;

    static float summe_ks_sin = 0, summe_cs_sin = 0, summe_ls_sin = 0;

    float sinuswert , cosinuswert ;

    analyse_index_cos = index_cos - n_f0 ;    // Index dekrementieren da in
    analyse_index_sin = index_sin - n_f0 ;    // signalgenerator () schon nächster
                                              // Index berechnet

    P_AD.ADCSR_0.BIT.ADF = 0;                 // Flag löschen

//------- Ringspeicher

    cosinuswert = sinus( analyse_index_cos ); // Wert aus Sinustabelle holen
    sinuswert   = sinus( analyse_index_sin ); // Wert aus Sinustabelle holen

    //------- Realteil

    summe_ks_cos -= koronaspannung_cos[i];
    summe_cs_cos -= cpuspannung_cos [i];
    summe_ls_cos -= ladespannung_cos [i];

    koronaspannung_cos[i] = P_AD.ADDR0.BIT.AD *cosinuswert;
    cpuspannung_cos  [i] = P_AD.ADDR1.BIT.AD *cosinuswert;
    ladespannung_cos  [i] = P_AD.ADDR2.BIT.AD *cosinuswert;

    summe_ks_cos += koronaspannung_cos[i];
    summe_cs_cos += cpuspannung_cos [i];
    summe_ls_cos += ladespannung_cos [i];


    //------- Imaginärteil

    summe_ks_sin -= koronaspannung_sin[i];
    summe_cs_sin -= cpuspannung_sin  [i];
    summe_ls_sin -= ladespannung_sin  [i];

    koronaspannung_sin[i] = P_AD.ADDR0.BIT.AD *sinuswert;
    cpuspannung_sin  [i] = P_AD.ADDR1.BIT.AD *sinuswert;
    ladespannung_sin  [i] = P_AD.ADDR2.BIT.AD *sinuswert;

    summe_ks_sin += koronaspannung_sin[i];
    summe_cs_sin += cpuspannung_sin  [i];
    summe_ls_sin += ladespannung_sin  [i];


    //------- Zuordnung an globale Variablen

    re_ks = summe_ks_cos;                     // Realwerte
    re_cs = summe_cs_cos;
    re_ls = summe_ls_cos;

    im_ks = summe_ks_sin;                     // Imaginärwete
    im_cs = summe_cs_sin;
    im_ls = summe_ls_sin;


    i ++;
    if (i == 256){i = 0;}

}//INT_ADI0
```

E.7 Berechnungen

```
void ablauf(void)
{
    float re_rauschen_cpu = 0, im_rauschen_cpu = 0;

    float re_ks1 = 0, re_cs1 = 0, re_ls1 = 0;

    float im_ks1 = 0, im_cs1 = 0, im_ls1 = 0;

    float re_ks2 = 0, re_cs2 = 0, re_ls2 = 0;

    float im_ks2 = 0, im_cs2 = 0, im_ls2 = 0;

    float re_zaehler_kt , im_zaehler_kt ;

    float re_zaehler_band , im_zaehler_band;

    float re_nenner , im_nenner;

    float betrag_kt , betrag_band, betrag_nenner;

    float f_kt_berechnung , f_band_berechnung;

    float phi_zaehler_kt , phi_zaehler_band , phi_nenner;

    float phi_kt_berechnung , phi_band_berechnung;

    float f_kt   [10] = {0}, f_band[10] = {0};

    float phi_kt   [10] = {0}, phi_band[10] = {0};

    short anzahl_frequenzen    = 5;
    short anzahl_bandumlaeufe = 64;

for (n_f0 = 1; n_f0 <= anzahl_frequenzen; n_f0++)
{
    //--------- messen des Rauschens an CPU

     sprintf ((unsigned char*) StringBuffer ," Messung bei Frequenz: %3d *2.85 Hz\r\n", n_f0);
    PutStrSCI3(StringBuffer );

     sprintf ((unsigned char*) StringBuffer ," Kein  Testsignal  an Kanal 1 und Kanal 2 \r\n");
    PutStrSCI3(StringBuffer );

    generator_dac_on   = 0;       // Testsignale  abschalten
    bandumlaufzaehler = 0;

    while (bandumlaufzaehler != anzahl_bandumlaeufe)
        { ; }

    re_rauschen_cpu = re_cs /128; // Messwerte zuordnen und normieren
    im_rauschen_cpu = im_cs /128;

    //--------- Cosinus auf Kanal1 und messen der Eingänge und der CPU

     sprintf ((unsigned char*) StringBuffer ," Testsignal  an Kanal 1 \r\n");
    PutStrSCI3(StringBuffer );

    dac_channel        = 1;       // DAC für Kannal1 auswählen
    generator_dac_on   = 1;       // Testsignalausgabe  anschalten
    bandumlaufzaehler = 0;

    while (bandumlaufzaehler != anzahl_bandumlaeufe)    // warten
        { ; }

    re_ks1 = re_ks /128;          // Messwerte zuordnen und normieren
    re_cs1 = re_cs /128;
     re_ls1 = re_ls /128;

    im_ks1 = im_ks /128;
    im_cs1 = im_cs /128;
     im_ls1 = im_ls /128;

    re_cs1 -= re_rauschen_cpu;  // Rauschen von gemessener CPU-Spannung subtrahieren
    im_cs1 -= im_rauschen_cpu;
```

```
//--------- Cosinus auf Kanal2 und messen der Eingänge und der CPU

    sprintf ((unsigned char*) StringBuffer ," Testsignal an Kanal 2 \r\n");
    PutStrSCI3(StringBuffer );

    dac_channel        = 2;      // DAC für Kannal2 auswählen
    generator_dac_on   = 1;      // Testsignalausgabe anschalten
    bandumlaufzaehler = 0;

    while (bandumlaufzaehler != anzahl_bandumlaeufe)    // warten
        { ; }

    re_ks2 = re_ks /128;         // Messwerte zuordnen und normieren
    re_cs2 = re_cs /128;
    re_ls2 = re_ls /128;

    im_ks2 = im_ks /128;
    im_cs2 = im_cs /128;
    im_ls2 = im_ls /128;

    re_cs2 -= re_rauschen_cpu;// Rauschen von gemessener CPU-Spannung subtrahieren
    im_cs2 -= im_rauschen_cpu;

//--------- DAC abschalten

    dac_channel        = 1;      // DAC für Kannal1 auswählen
    generator_dac_on   = 0;      // Testsignalausgabe abschalten
    bandumlaufzaehler = 0;

    sprintf ((unsigned char*) StringBuffer ," Messungen beendet \r\n");
    PutStrSCI3(StringBuffer );

    sprintf ((unsigned char*) StringBuffer ," Berechnung fuer Frequenz: %3d *2.85 Hz\r\n", n_f0 );
    PutStrSCI3(StringBuffer );

//--------- Berechnung durchführen

    //--------- Zähler für Ü.-funktion von Koronatriode (f_kt)

    re_zaehler_kt = re_cs1 * re_ls2 - im_cs1 *im_ls2 - re_cs2 * re_ls1 + im_cs2 *im_ls1;
    im_zaehler_kt = re_cs1 *im_ls2 + re_ls2 *im_cs1 - re_cs2 *im_ls1 - re_ls1 *im_cs2;

    //--------- Zähler für Ü.-funktion vom Band (f_band)

    re_zaehler_band = re_cs2 *re_ks1 - im_cs2 *im_ks1 - re_cs1 *re_ks2 + im_cs1 *im_ks2;
    im_zaehler_band = re_cs2 *im_ks1 + re_ks1 *im_cs2 - re_cs1 *im_ks2 - re_ks2 *im_cs1;

    //--------- Nenner für beide Ü.-funktionen

    re_nenner = re_ks1 * re_ls2 - im_ks1 *im_ls2 - re_ks2 * re_ls1 + im_ks2 *im_ls1;
    im_nenner = re_ks1 *im_ls2 + re_ls2 *im_ks1 - re_ks2 *im_ls1 - re_ls1 *im_ks2;

    //--------- Betragsfunktionen

    betrag_kt     = sqrt( re_zaehler_kt * re_zaehler_kt + im_zaehler_kt * im_zaehler_kt );
    betrag_band   = sqrt( re_zaehler_band *re_zaehler_band + im_zaehler_band *im_zaehler_band );
    betrag_nenner = sqrt(re_nenner *re_nenner + im_nenner *im_nenner);

    f_kt_berechnung   = betrag_kt   /betrag_nenner;
    f_band_berechnung = betrag_band /betrag_nenner;

    //--------- Winkel

    phi_zaehler_kt   = atan(im_zaehler_kt   / re_zaehler_kt );
    phi_zaehler_band = atan(im_zaehler_band / re_zaehler_band );
    phi_nenner       = atan(im_nenner       /re_nenner );

    phi_kt_berechnung   = phi_zaehler_kt   - phi_nenner;
    phi_band_berechnung = phi_zaehler_band - phi_nenner;

//--------- Ergebniss speichern

    f_kt   [n_f0] = f_kt_berechnung;
    f_band [n_f0] = f_band_berechnung;

    phi_kt [n_f0] = phi_kt_berechnung;
```

```
    phi_band[n_f0] = phi_band_berechnung;

    sprintf((unsigned char*)StringBuffer," Ergebnisse: \r\n");
    PutStrSCI3(StringBuffer);

    sprintf((unsigned char*)StringBuffer," f_kt= %10.2f f_band= %10.2f \
                          phi_kt= %4.2f phi_band= %4.2f \r\n\n\n",\
                          f_kt[n_f0], f_band[n_f0], phi_kt[n_f0], phi_band[n_f0]);
    PutStrSCI3(StringBuffer);

}//for (n_f0 = 1; n_f0 <= anzahl_frequenzen; n_f0++)

//-------- Ausgabe aller Ergebnisse

for (n_f0 = 1; n_f0 <= anzahl_frequenzen; n_f0++) {
    sprintf((unsigned char*)StringBuffer," Messung bei Frequenz: %3d *2.85 Hz\r\n", n_f0);
    PutStrSCI3(StringBuffer);

    sprintf((unsigned char*)StringBuffer," f_kt= %10.2f f_band= %10.2f \
                          phi_kt= %4.2f phi_band= %4.2f \r\n\n",\
                          f_kt[n_f0], f_band[n_f0], phi_kt[n_f0], phi_band[n_f0]);
    PutStrSCI3(StringBuffer);

}//for (n_f0 = 1; n_f0 <= anzahl_frequenzen; n_f0++)

sprintf((unsigned char*)StringBuffer," Messung beendet \r\n");
    PutStrSCI3(StringBuffer);
}//void ablauf(void)
```

E.8 Look-Up-Table

```
float   sinus_interpolation (float x)
{
    short x0, x1;
    float y0, y1, delta_x, sinus_x;

    while (x > 2*PI)                    // Periodizität des Sinus
        {  x-= 2*PI;}

    x *= 100;

    x0 = floor(x);                      // abrunden für Feldindex (Vorgänger)
    x1 = x0 +1;                         // Feldindex des Nachfolgers

    y0 = sinustabelle [x0];             // Vorgängerstützstelle
    y1 = sinustabelle [x1];             // Nachfolgerstützstelle

    delta_x = x-x0;

    sinus_x = y0 + (y1-y0) *delta_x;    // lineare Interpolation zw. Stützstellen

    return sinus_x;
}
```

Anhang F: Dokumentation zu SH-2 EDK 7047

Dokumentation zu SH-2 EDK 7047 und HEW 3 (GNU Toolchain)

erstellt von: H. Lange

14. November 2004

- Installation von HEW 3
- Einrichten des E10A-USB Emulator
- Installation der GNU Toolchain
- Beispielprojekte

Vorwort

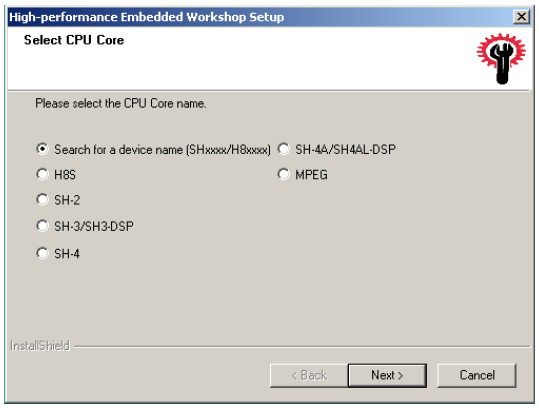
Diese Dokumentation hat das Ziel, das Arbeiten mit dem Mikrocontroller SH-2 / 7047 bzw. mit dem Evaluationboard SH-2 EDK 7047 zu erleichtern. Das Board wird zusammen mit dem E10A-USB Emulator betrieben, welcher ein On-Chip-Debugging über die USB-Schnittstelle eines PCs ermöglicht. Auf das Debugging mit Hilfe des Hardwaremonitors HMON wird im Folgenden nicht eingegangen.
Es wird die Installation der zum Betrieb benötigten Softwarekomponenten und das Erstellen von eigenen Projekten gezeigt. Anhand von Beispielen sollen einfache Funktionen wie z. B. das Blinken von LEDs und die Datenkommunikation über die serielle Schnittstelle demonstriert und das Debuggen des Quellcodes ermöglicht werden.

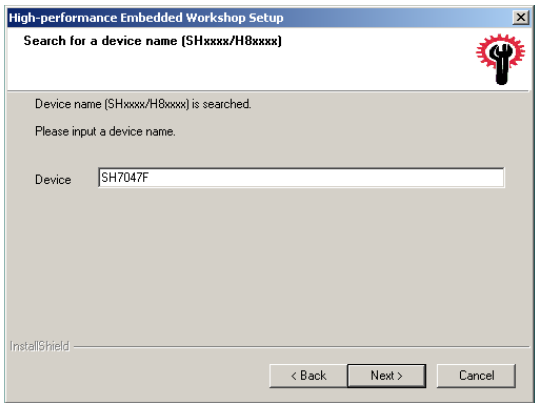
Installation der benötigten Software

Installiert werden muss die folgende Software:

- **HEW 3:** hew3ntc (no toolchain)
- **E10A-USB Emulator:** V.1.07 Release 00
- **GNU Toolchain:** gnushv0402-elf

Die Installation von HEW 3 (High-performance Embedded Workshop) kann in ein beliebiges Verzeichnis erfolgen und benötigt keine speziellen Angaben. Auf die mitgelieferte Toolchain von Hitachi wird verzichtet, da ihre Lizenz zeitlich begrenzt ist.
Das Einbinden des E10A-USB Emulator geschieht über die Installation der Software von der mitgelieferten CD der Firma RENESAS. Während der Installation wird der HEW 3 automatisch erkannt und die benötigte Software in das gleiche Verzeichnis kopiert. Es muss lediglich eine Einstellung bezüglich des verwendeten Gerätes gemacht werden. Hier sollte der verwendete Mikrocontroller bzw. das EDK 7047 direkt spezifiziert werden: SH 7047 F.

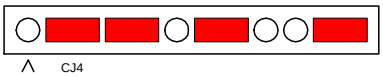


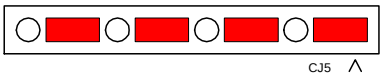


Bei der Installation der GNU Toolchain wird wiederum die bereits vorhandene Software des HEW 3 erkannt. Das Installationsverzeichnis muss mit dem des HEW 3 übereinstimmen. Weitere spezielle Einstellungen sind nicht erforderlich. Die GNU Toolchain wird bezüglich des HEW 3 automatisch konfiguriert.

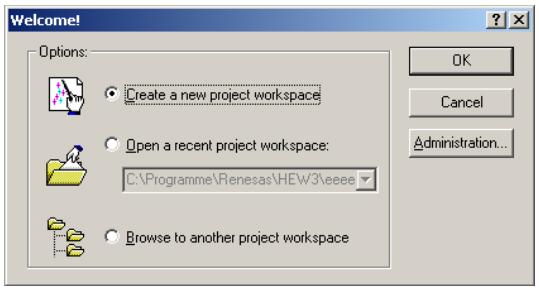
Ein neues Projekt

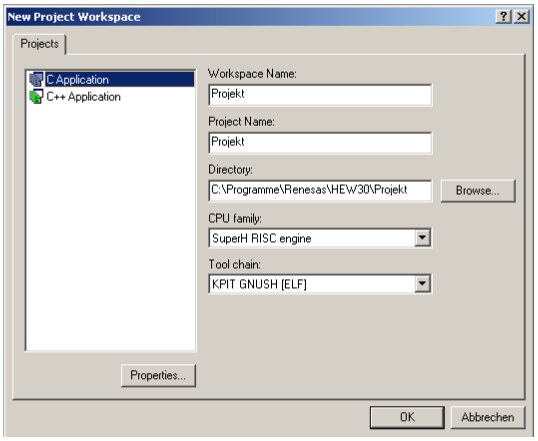
Nach dem Anschließen des E10A-USB Emulators an den PC und an das EDK 7047 wird dieser automatisch erkannt und kann sofort genutzt werden. Zusätzlich wurde während der Erstinstallation auf dem EDK 7047 an der Buchse J3 der Kontakt 2 durchtrennt, da dieser nicht beschalten werden darf. Die Jumper auf dem EDK 7047 sollten wie folgt gesteckt sein:





Startet man den HEW 3 fragt dieser ob ein neues Projekt angelegt oder ein bereits vorhandenes geöffnet werden soll. Im Sinne dieser Anleitung wird ein neues Projekt erstellt:





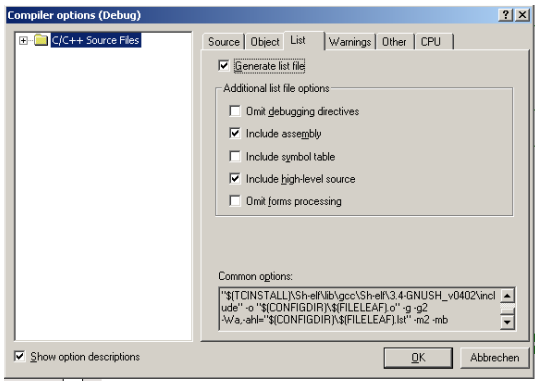
Ein Workspace- bzw. ein Projektname und das zu verwendende Verzeichnis kann beliebig gewählt werden. Als CPU-Family wird die *SuperH RISC engine* und als Toolchain die *KPIT GNUSH[ELF]* angegeben. Als Projektvorlage dient in diesem Beispiel eine leere C-Applikation. Im folgenden Dialogfenster wählt man *SH-2* als CPU-Typ. Alle anderen Einstellmöglichkeiten können ohne Veränderung übernommen werden. Der Verbindungsaufbau zum Emulator kann erst einmal abgebrochen werden.
Da es keine (gute) Möglichkeit gibt, bestehende Projekte zu ex- bzw. zu importieren, wird ein etwas unkonventioneller Weg gegangen, um eine der vorliegenden Beispielanwendungen testen zu können. In den Ordner des soeben angelegten Projektes werden die *.c und *.h -Dateien aus einem der Beispielprojekte kopiert und damit die bereits vorhan-

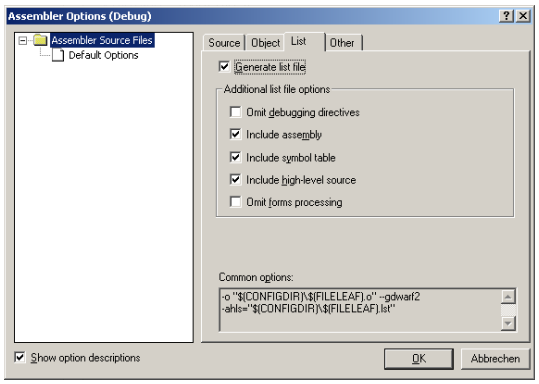
denen Dateien überschrieben. Mit *Project* → *Add Files...* werden die dazugekommenen Dateien in das Projekt eingegliedert. Folgende Einstellungen müssen noch vorgenommen werden:

Debug und *Debug session* einstellen: 

.

Desweiteren folgt: *Options* → *Compiler...* und *Options* → *Assembler...*



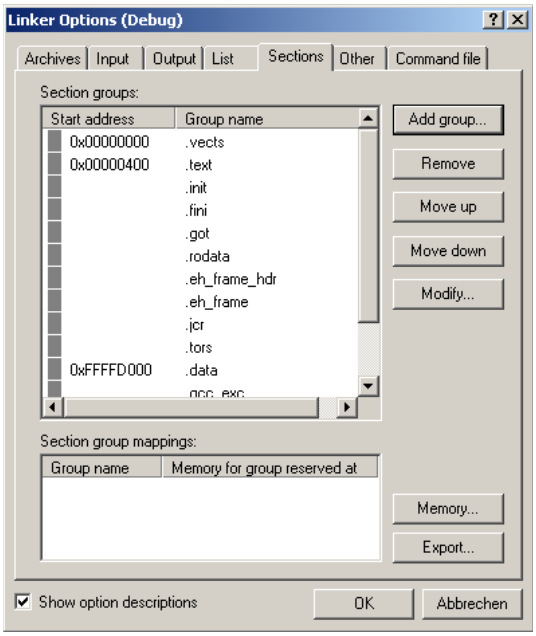


Alle anderen Einstellungen können beibehalten werden.

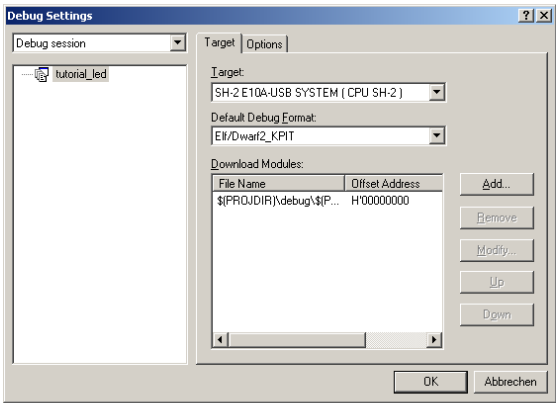
Unter *Options* → *Linker...* werden die Speicherbereiche auf dem Mikrocontroller definiert, in welche der Maschinencode geladen wird.

Die Aufteilung lautet:

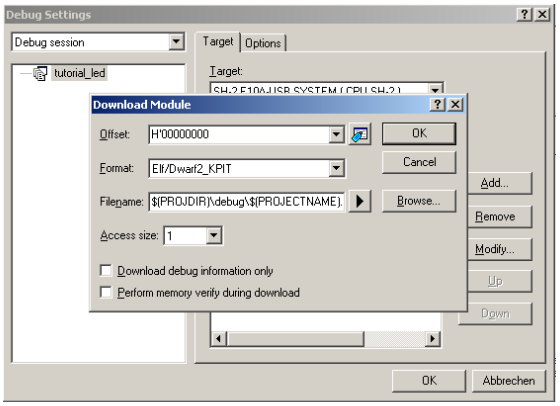
- Interruptvektortabelle
  .vects → 0x 0000 0000 (Flash)
- Quellcode
  .text → 0x 0000 0400 (Flash)
- Variablen
  .data → 0x FFFF D000 (interner RAM)
- Stack
  .stack → 0x FFFF FFFC (interner RAM)



Eine weitere wichtige Konfiguration legt fest, welche Datei in den Mikrocontroller geladen wird. Über das Menü *Options* → *Debug Settings...* wird das Zielgerät, das Debugformat und die zu verwendende Datei ausgewählt. Folgende Einstellungen sind für die GNU Toolchain und den Emulator notwendig:

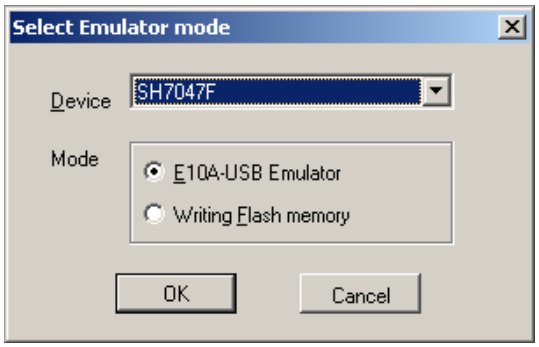


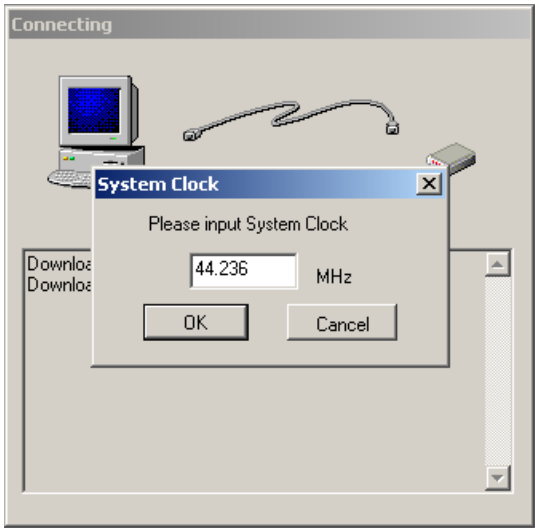
- SH-2 E10A-USB SYSTEM (CPU SH-2)
- Elf/Dwarf2_KPIT



- H'00000000
- Elf/Dwarf2_KPIT
- $(PROJDIR)\debug\$(PROJECTNAME).x

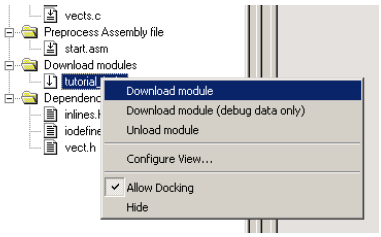
Nach *File* → *Save Workspace*... und *File* → *Save Session*... kann das so konfigurierte Projekt nun mit dem Befehl *Build all* kompiliert werden.
Nach erfolgreichem Kompilieren wird eine Verbindung zum Emulator und somit auch zum EDK 7047 aufgebaut. Über das Menü *Build* → *Debug* → *Connect* oder mit erscheint folgender Dialog:





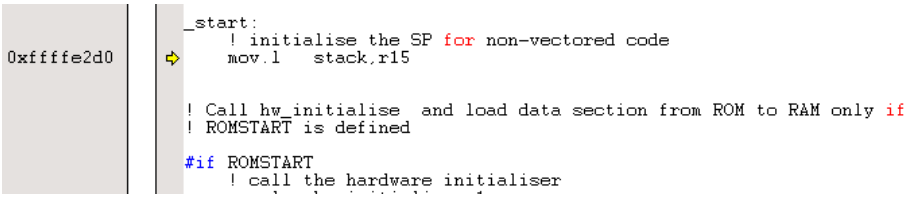
Wenn der geforderte manuelle Reset am EDK 7047 keine Wirkung zeigt, kann man die wiederholte Aufforderung ohne Folgen ignorieren. Der einzugebende Wert *System Clock* ist nicht die Taktfrequenz des externen Oszillators auf dem EDK 7047 (11,0592 MHz). Der Oszillatortakt wird im Mikrocontroller für die CPU vervierfacht (44,236 MHz) und für die peripheren Komponenten (Timer, ADC, ...) verdoppelt (22,118 MHz). Einzugeben ist die CPU-Frequenz. Nach unterschiedlich langer Wartezeit wird die Verbindung hergestellt und die vorher ausgewählte Datei *Projektname.x* kann in den Mikrocontroller geladen werden. Dafür gibt es zwei Varianten: Zum einen kann die Datei nach jedem

Kompilieren automatisch geladen werden (*Options* → *Debug Settings*. . . → *Options* → *Download modules after build*) und zum anderen mit einem Rechtsklick auf die Datei in der Verzeichnisstruktur mit anschließender Auswahl von *Download module*.

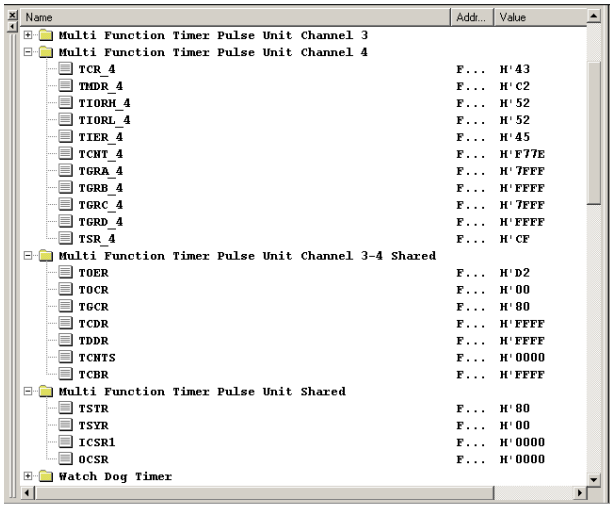


Beispielprojekte

In Abhängigkeit von dem Beispiel, welches in das selbst angelegte Projekt kopiert worden ist, stehen verschiedene Anwendungen zur Verfügung. Im **Tutorial LED** werden die zwei LEDs auf dem EDK 7047 angesteuert. Zum einem durch die Funktion Flashing_LED(), in welcher eine Zählschleife programmiert ist, und zum anderen mit Hilfe des Timers4 und zwei zugeordneten Interrupts. Der Timer wird mit einer vorgegebenen Taktung aufwärts gezählt, bis sein Zählerstand mit dem Wert, welcher im Register TGRB steht, übereinstimmt. Nach diesem compare match wird der Zähler auf Null zurückgesetzt und wieder gestartet. In den Registern TGRA und TGRC sind wiederum Vorgaben gespeichert, die der Zähler im Laufe der Zeit überstreift. Ein solcher compare match setzt den Timer nicht zurück, löst aber jeweils ein Interrupt aus. In den zugehörigen Interruptserviceroutinen wird die jeweilige LED über TIOC4x angesteuert und gleichzeitig wird der Wert im Register TGRx verringert, sodass sich die Blinkfrequenz ständig ändert. Für weitere Erläuterungen sollten die Kommentare im Quellcode ausreichen. Die in Klammern angegebenen Seitenzahlen beziehen sich auf das Hardware-Manual des Mikrocontrollers. Zum Starten der Funktionen wird im ersten Schritt über das Menu *Debug* → *Reset CPU* oder mit  die CPU und alle Register zurückgesetzt. Der Programm-Counter sollte nun an der Adresse der Startroutine stehen  .

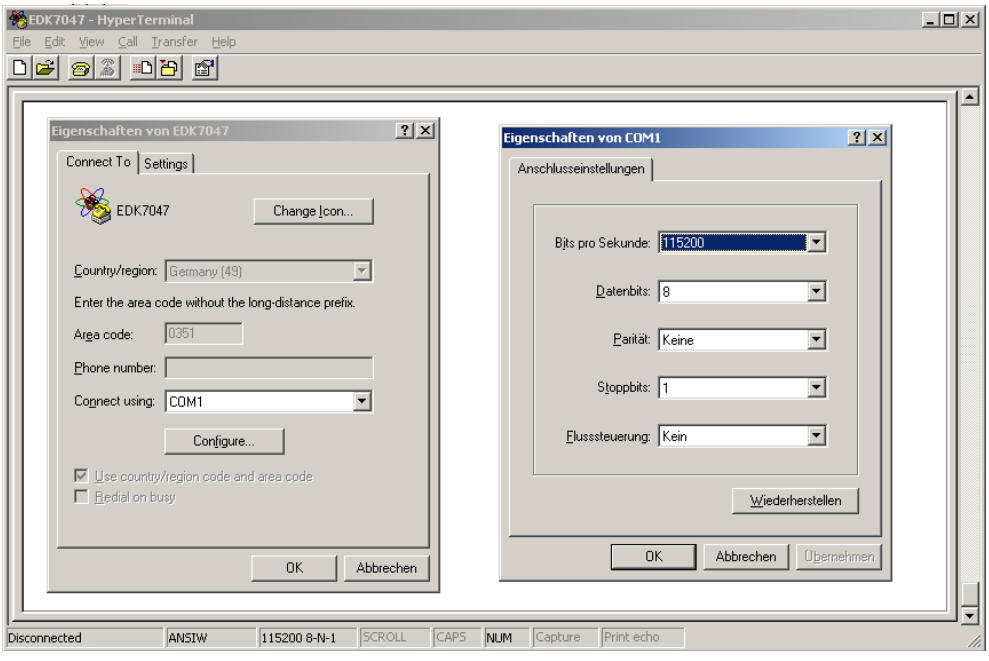


Mit *Debug* → *Go* oder mit  wird die Applikation gestartet. Zum Debuggen können Breakpoints gesetzt, in Einzelschritten  die programmierten Anweisungen getestet und mit Hilfe von  die Registerinhalte des Mikrocontroller geprüft werden.

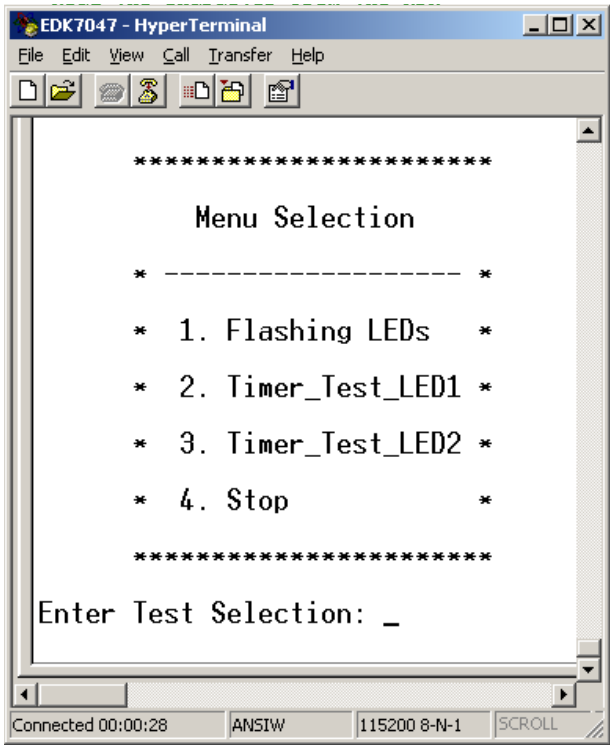


Sollte der HEW 3 einmal abstürzen, beendet man ihn mit dem Windows Task Manager, unterbricht die USB-Verbindung zwischen PC und Emulator und beginnt von vorn.

Im **Tutorial Serial** ist eine ähnliche Funktionalität wie die eben beschriebene programmiert worden. Im Unterschied zum ersten Programm wird ein grafisches Auswahlmenu über die serielle Schnittstelle ausgegeben, über welches auch die Steuerung der LEDs erfolgt. Dazu muss vorher noch das Windows-HyperTerminal konfiguriert werden. Die Einstellungen müssen mit denen im Quellcode (main.c) angegebenen Werten übereinstimmen.



Nach dem Aktivieren des HyperTerminals wird die zuvor in den Controller geladene Applikation wieder mit und gestartet. Im Fenster des Terminals erscheint folgendes Menu:



Mit Eingabe der Ziffern können die LEDs durch Starten der jeweiligen Routine ausgewählt und gesteuert werden.
Ausgehend von diesen Beispielprojekten können eigene Applikationen programmiert werden. Die Dateien main.c und hwinit.c können auch komplett neu geschrieben werden. Die Dateien vects.c und iodefine.h müssen bei jedem neuen Projekt verwendet werden, da diese zum einen die Interruptvektortabelle zum anderen die Registerstrukturen enthalten, welche speziell auf das EDK 7047 angepasst worden sind. In vects.c werden alle verwendeten Interrupts an die jeweilige Stelle eingetragen und in iodefine.h ist es möglich, genaue Registerbezeichnungen und deren Größe zu erfahren. Eine eigene Zusammenstellung von Dateien kann mit *Project* → *Create Project Type* in eine Vorlage für andere Projekte überführt werden.

Quellen- und Literaturverzeichnis

[1] Hellborg, R.: Electrostatic Accelerators. North-Holland Publishing Co. Amsterdam, im Druck

[2] Simonyi, K.: Physikalische Elektronik. Akadémiai Kiadó, Budapest, 1972

[3] Bürger, W.: On the Dynamic Behaviour of a Charging System of a Van de Graaff Belt Generator. Forschungszentrum Rossendorf, 2003

[4] Proakis, J. G., Salehi, M.: Grundlagen der Kommunikationstechnik. Pearson Studium, München, 2004

[5] Eichhorn, K., Hosemann, G., Schneider, E.: Digitale Messungen von Wechsel- und Drehstromgrößen für die Netzschutz- und Leittechnik. In: Elektrie, Berlin 44 1990 7

[6] Lobos, T.: Digitale Filteralgorithmen zur Echtzeitermittlung der Grundschwingung von Spannungen, Strömen und symmetrischer Komponenten. In: Elektrie, Berlin 46 1992 1

[7] Guillaume, P., Schoukens, J., Pintelon, R.: On the Use of Signals with a Constant Signal-to-Noise Ratio in the Frequency Domain. In: IEEE Transactions on Instrumentation and Measurement, 39(6), 1990

[8] RENESAS Technology Corp.: Hardware Manual SH 7047, 2003
URL: `www.renesas.com`

[9] KPIT Cummins Infosystems Limited: KPIT GNU Tools and Support
URL: `www.kpitgnutools.com`

[10] Girod, B., Rabenstein, R., Stenger, A.: Einführung in die Systemtheorie. B. G. Teubner Stuttgard, 1997

[11] Hoffmann, J.: Matlab und Simulink - Beispielorientierte Einführung in die Simulation dynamischer Systeme. Addison-Wesley, München, 1998