Fachhochschule Pforzheim
Fachbereich 2 Elektrotechnik / Informationstechnik

Entwicklung des Bedienteils für einen Signalgenerator
Firmware für Signalgenerator und Bedienteil

Dokumentation der Projektarbeit
im 5. Semester

WS 2006/07

Erstellt von Thomas Gulden

Betreut von Prof. Dr.-Ing. Wolf-Henning Rech

Inhalt

1. Übersicht

Für einen vorhandenen Prototyp eines Signalgenerators soll ein Microcontroller zur Steuerung ausgewählt und ein passendes Bedienteil entwickelt werden, damit dieser für verschiedene Versuche im HF-Labor der Hochschule eingesetzt werden kann.

Die vorliegende Dokumentation beschreibt die Entwicklung und Konzeption dieser Komponenten ausgehend von einer Bestandsaufnahme an der bereits bestehenden Hardware und anhand der Eigenschaften, die das Gerät später aufweisen soll.

Anschließend folgt eine Beschreibung der hard- und softwaretechnischen Umsetzung.

Besonderer Wert wird dabei auf einen modularen Aufbau gelegt, so dass die einzelnen Ausbaustufen schrittweise vorgestellt werden können und das System mit geringem Einarbeitungsaufwand gewartet und erweitert werden kann.

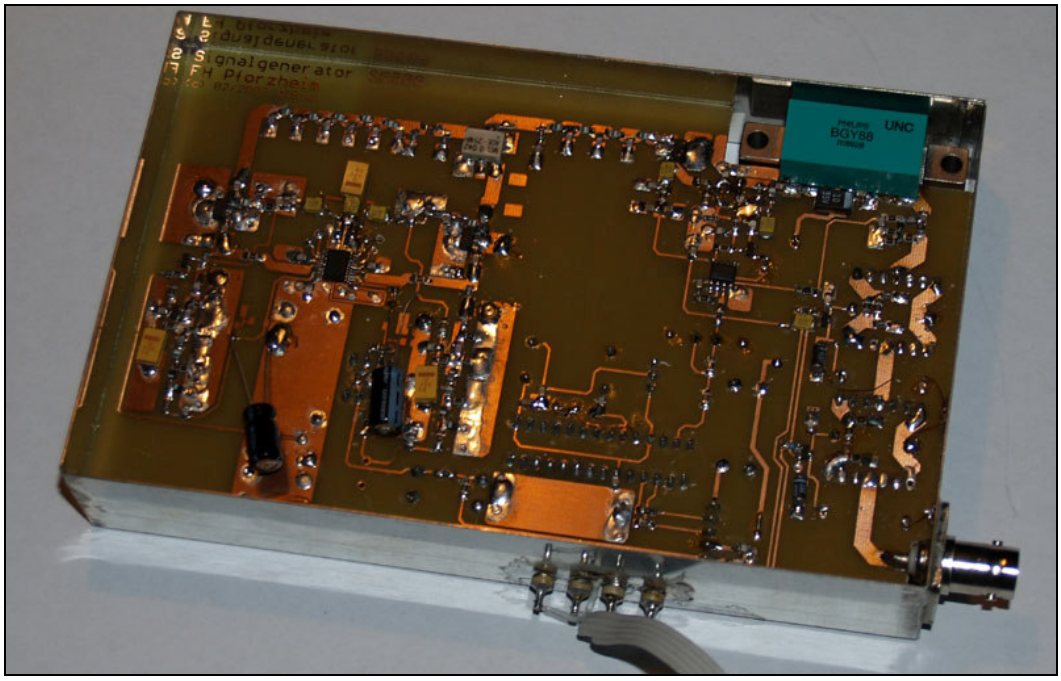


Abbildung 1 : Der Prototyp des Signalgenerators im Originalzustand

2. Bestandsaufnahme

2.1 Aufbau & Funktionsweise

Bei dem vorliegenden Prototyp des Signalgenerators handelt es sich um eine Eigenentwicklung an der Fachhochschule Pforzheim.

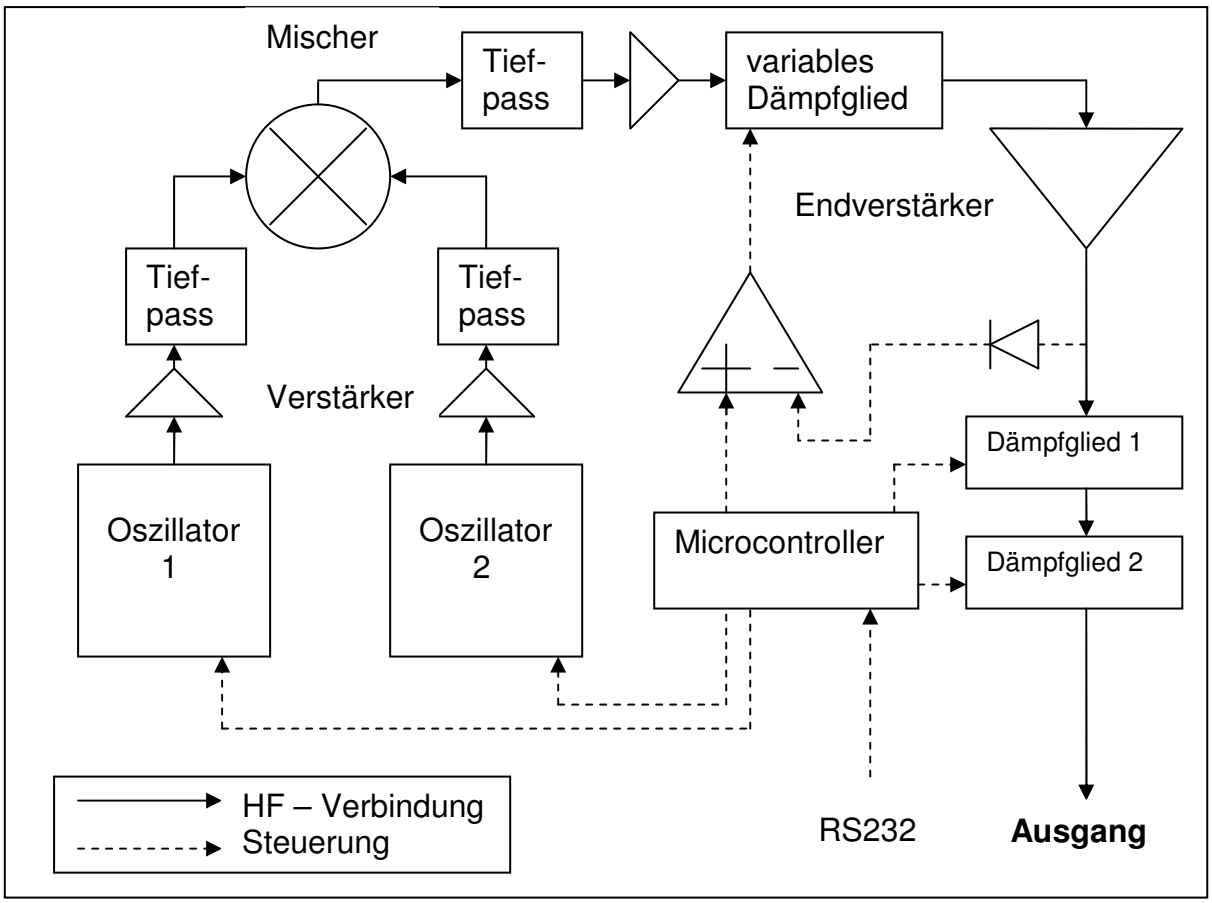


Abbildung 2 : Blockschaltbild des Signalgenerators

Der Signalgenerator deckt einen Frequenzbereich von 40 .. 500 MHz ab, welcher durch Mischung zweier Oszillatoren, die mittels PLL (Phase Locked Loop) stabilisiert sind, erzeugt wird.
Ein Oszillator wird mittels der PLL fest auf eine Frequenz von 1 GHz eingestellt während der zweite Oszillator je nach geforderter Ausgangsfrequenz zwischen 1040 und 1500 MHz programmiert werden kann.
Die Ausgangssignale der Oszillatoren werden durch einen Treiber gepuffert und durchlaufen anschließend ein Tiefpassfilter um Oberwellen zu unterdrücken.
Die gefilterten Signale werden einem passiven Mischer zugeführt und das Differenzsignal aus beiden Oszillatoren über ein Tiefpassfilter selektiert.
Dieses Signal im Bereich von 40 .. 500 MHz wird von einem Treiber gepuffert, dem ein variables Dämpfungsglied (PIN-Dioden) folgt, über welches die Ausgangsleistung mit einem Dynamikumfang von etwa 18 dB regelbar ist.

Dem Dämpfungsglied folgt ein Breitband-Verstärkungsmodul, welches einen maximalen Ausgangspegel von etwa +18 dBm bereitstellt.
Um den Frequenzgang des Systems zu kompensieren, folgt diesem ein Diodendetektor, welcher eine Spannung liefert, welche proportional zur Ausgangsleistung des Moduls ist und über einen Komparator den PIN-Diodenabschwächer regelt.
Als Referenzsignal des Komparators dient eine Spannung aus dem Microcontroller, durch welche die Soll-Leistung vorgegeben wird.
Um die Ausgangsleistung in weiteren Schritten variieren zu können, folgen abschließend zwei Festwertabschwächer, die mittels Relais realisiert sind und Dämpfungen von 15 bzw. 25 dB aufweisen.

Die Steuerung dieser Hardware, im Folgenden als „HF-Teil“ bezeichnet, erfolgt durch ein Bedienteil, welches über eine serielle Schnittstelle mit dem HF-Teil verbunden wird.

Das Bedienteil besteht im Wesentlichen aus einem alphanumerischen Display mit 2 Zeilen à 16 Zeichen, einem Drehgeber zum Einstellen der Frequenz und Amplitude und 5 Tastern, über welche die Frequenz- oder Amplitudeneinstellung und die zu ändernde Stelle im Display ausgewählt und der HF-Ausgang an-/abgeschaltet wird. Zudem kann das Bedienteil über einen Jumper in den Kalibriermodus versetzt werden, mit dem sich die Amplitude abgleichen lässt.

2.2 Sichtkontrolle und Funktionstest

Bei der ersten Sichtkontrolle wurden folgende Mängel am Prototyp entdeckt:

1. Zu der Software des auf dem Prototyp verbauten PIC Microcontrollers gab es keinerlei Dokumentation
2. Etliche Bauteile waren nicht bestückt oder deren Wert war in der Dokumentation nicht angegeben
3. Jedes Exemplar des Prototyps war unterschiedlich bestückt
4. Einige Bauteile hatten den Lötprozess nicht unbeschadet überstanden und mussten ersetzt werden

Nach Anlegen der Betriebsspannung meldete sich der Microcontroller mit „Bereit“ im Terminalprogramm und die Oszillatoren schwangen an.

Ein weiterer Funktionstest war nicht möglich, da nicht bekannt war, mit welchen Kommandos die Firmware im PIC gesteuert werden konnte.
Ein Hochfrequenz – Signal konnte zu diesem Zeitpunkt auch nicht festgestellt werden, da einer der Treiberverstärker nach den Oszillatoren seinen Dienst verweigerte.
Die aktuellen Bauteilwerte sind der Originaldokumentation des HF-Teils zu entnehmen

3. Entwurf

3.1 Anforderungen an den HF-Teil

Da der Signalgenerator als „low cost“ Alternative zu Geräten von Agilent oder Rohde&Schwarz gedacht ist, müssen verständlicherweise einige Abstriche bezüglich der Performance gemacht werden.
Um dennoch sinnvoll mit den Geräten arbeiten zu können, wurden folgende Forderungen gestellt:

- Frequenz in 100 kHz – Schritten zwischen 40 und 500 MHz einstellbar
- Amplitude in 0,1 dB – Schritten regelbar
- Serielle Schnittstelle für die Steuerung
- Ablegen der Kalibrierdaten im Speicher auf der HF-Platine um einen Austausch der Bedienteile ohne Rekalibrierung zu ermöglichen

3.2 Anforderungen an das Bedienteil

Das Bedienteil stellt später die Brücke zwischen dem Signalgenerator und dem Benutzer dar, daher sollte die Bedienung so intuitiv und selbsterklärend wie möglich sein, weshalb der Funktionsumfang auf das Wesentliche reduziert ist:

- Alphanumerisches Display welches Frequenz und Amplitude anzeigt und bei Bedarf durch die Kalibrierung führt
- Drehgeber zum Einstellen von Frequenz bzw. Amplitude
- Taster, mit denen ausgewählt werden kann, ob die Amplitude oder Frequenz verstellt werden kann
- Taster, mit denen man die zu ändernde Stelle (0,1MHz, 1Mhz, 10MHz-Stelle) auswählen kann, um eine Grob- und Feineinstellung zu ermöglichen
- Taster, mit dem das Ausgangssignal an- und abgeschaltet werden kann
- Ein Jumper auf der Platine des Bedienteils, mit dem in den Kalibriermodus gewechselt werden kann; dieser wird nicht als Taster auf die Frontseite herausgeführt, um den Signalgenerator vor versehentlichem Fehlkalibrieren zu sichern

4. Überlegungen zur Steuerung des HF-Teils

4.1 Auswahl eines geeigneten Microcontrollers

Entgegen der anfänglichen Intention, einen 8051-kompatiblen Microcontroller zu verwenden, mit dem bereits im Microcontroller – Labor Erfahrungen gesammelt wurden, fiel die Entscheidung zugunsten eines Vertreters der PIC-Familie aus dem Hause Microchip.
Entscheidend hierfür waren vor allem der bereits vorhandene Sockel für einen PIC 16F870 bzw. den pinkompatiblen 16F876, die Verfügbarkeit und weite Verbreitung von PIC-Controllern und der gute Support durch zahlreiche Anwender, die bereits mit dieser Prozessorfamilie gearbeitet haben.
Um die Programmierung modular und einfach nachvollziehbar zu machen und die Einarbeitungszeit zu minimieren, kam die Programmierung in Assembler-Sprache nicht in Betracht.
Stattdessen sollte der Controller in der Hochsprache C programmierbar sein.
Als Entwicklungsumgebung wurde im Folgenden der C-Compiler der Firma CCS verwendet, welcher durch eine überdurchschnittlich gute Dokumentation und viele Beispieldateien überzeugen konnte.

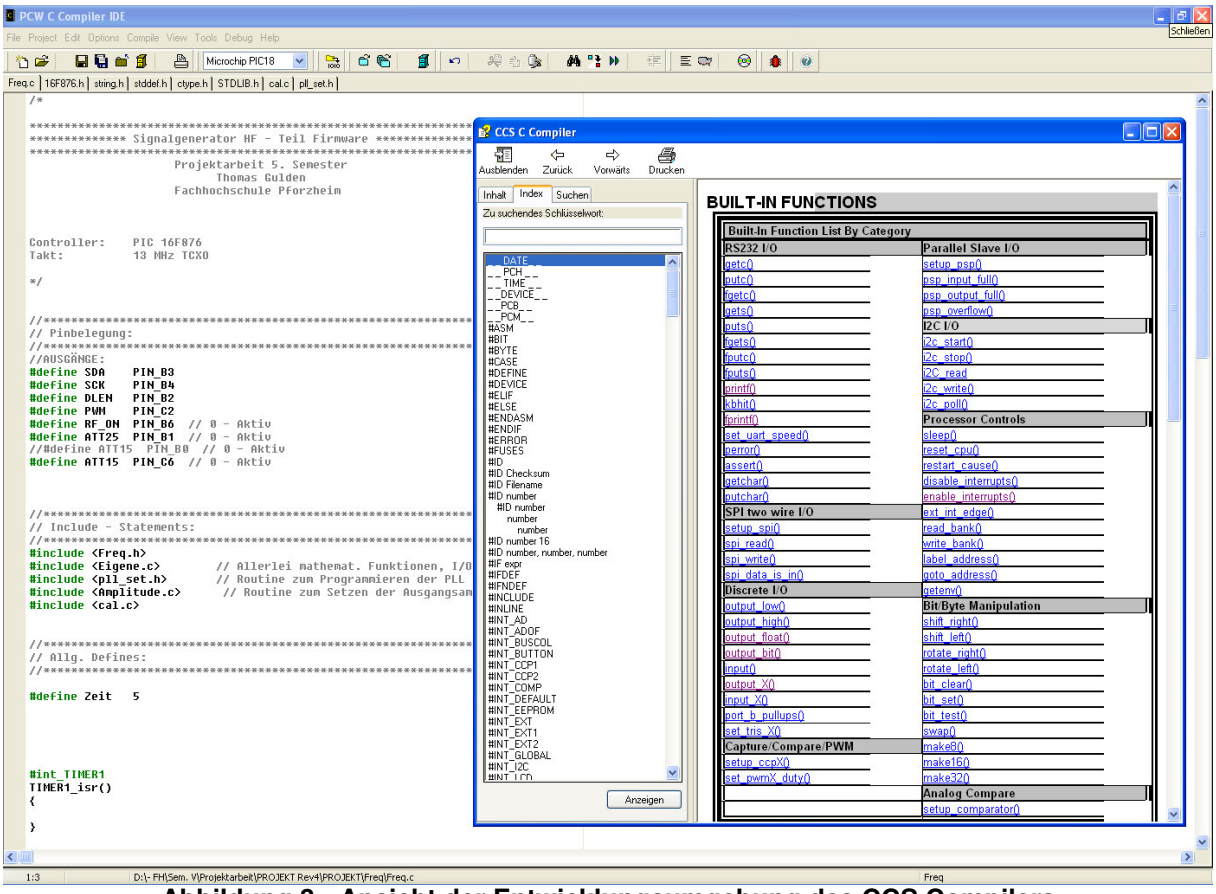


Abbildung 3 : Ansicht der Entwicklungsumgebung des CCS Compilers

Letztendlich wurde ein PIC 16F876 im HF-Teil eingesetzt, da dieser ein relativ großes internes EEPROM, mehr RAM und 8k Programmspeicher enthält, während der 16F870 lediglich 2k Programmspeicher beinhaltet.
Somit sind ausreichend Ressourcen zum Ablegen der Kalibriertabelle und für die Frequenzberechnungen vorhanden.

Dieser Controller hat einen Flash-Programmspeicher, wodurch er mehrmals programmiert werden kann, was in Verbindung mit einer „In System Programming“-Schnittstelle spätere Firmware-Updates ermöglicht.

4.2 Aufgaben des Microcontrollers

Der Microcontroller hat folgende Aufgaben:

- Einlesen der seriellen Schnittstelle
- Umrechnen des empfangenen Frequenz-Datenwortes in die Werte, mit welchen die Register des PLL-Bausteins geladen werden
- Emulation des SPI-Interfaces zum PLL-Baustein in Software
- Auswertung der gewünschten Soll-Amplitude, Einstellen des PIN-Abschwächers und setzen der Relais-Abschwächer unter Beachtung der im EEPROM abgelegten Korrekturwerte

4.3 Die serielle Kommunikation

Die serielle Kommunikation erfolgt prinzipiell nach dem RS232-Standard, jedoch ohne die Spannungen von TTL-Niveau (0V / 5V) auf das nach dem RS232-Standard geforderte Potential von mindestens +/- 5V zu wandeln.

Das Protokoll ist als „Klartext“ realisiert, was die Fehlersuche mittels Terminalprogrammen erheblich vereinfacht.
Somit war es möglich, zunächst den HF-Teil betriebsfertig zu machen und erst danach mit der Konzeption des Bedienteils zu beginnen.

Folgende Datenworte finden Verwendung:

Bedienteil ➔ Synthesizer:

- Fxxxxxx mit xxxxxx = Sollfrequenz in kHz
- Axxx mit xxx = Sollamplitude in (dBm * 10)
- Rx mit x = 0 ➔ RF off oder x = 1 ➔ RF on

Dies sind die Steuerkommandos, welche im normalen Betrieb vorkommen.
Zum Kalibrieren werden zudem folgende Kommandos verarbeitet:

Bedienteil ➔ Synthesizer:

- G maximal mögliche Amplitude bei aktuell eingestellter Frequenz ausgeben, d.h. alle Dämpfglieder sind abgeschaltet
- H 15 dB Dämpfglied kalibrieren: Bei der Mittenfrequenz f=280 MHz wird lediglich das 15 dB Dämpfglied zugeschaltet
- I 25 dB Dämpfglied kalibrieren: Bei der Mittenfrequenz f=280 MHz wird lediglich das 25 dB Dämpfglied zugeschaltet
- Xxxx mit xxx = Amplitude in (dBm * 10) mit zugeschaltetem 15dB Dämpfglied
- Yxxx mit xxx = Amplitude in (dBm * 10) mit zugeschaltetem 25dB Dämpfglied
- Zxxx mit xxx = Amplitude in (dBm * 10) ohne zugeschaltete Dämpfglieder zur Kalibrierung der Ausgangsleistung über der Frequenz

Ausserdem existert zu Debugzwecken und zur Aufnahme der Kalibrierkurve des PIN-Dämpfglieds noch das Kommando „Dxxxx“ mit xxxx ϵ 0 .. 1023.
Dabei wird der übergebene Zahlenwert direkt in das Register des PWM-Ausgangs geschrieben, welcher als Referenz für das PIN Dämpfglied dient.
Ein Wert von 1023 entspricht minimaler Durchgangsdämpfung.

Jedes Kommando wird durch das „Carriage Return“ – Zeichen („\r“) terminiert.

Durch das einfache Protokoll kann als mögliche Erweiterung beispielsweise ein LabVIEW Programm geschrieben werden, wodurch der Signalgenerator vom Computer aus steuerbar wird oder es kann bei Bedarf ein Sweepmodus realisiert werden.
Selbstverständlich muss in diesem Fall ein Pegelwandler auf RS232-Potential, zum Beispiel ein MAX232, vorgeschaltet werden, um eine Beschädigung des Microcontrollers zu verhindern.

4.4 Die PLL

Bei der PLL (= Phase Locked Loop) handelt es sich um einen „Dual Frequency PLL Synthesizer“ des Typs LMX2336 von National Semiconductor.
Diese enthält in einem 16poligen SMD-Gehäuse zwei unabhängige PLL-Synthesizer mit einer gemeinsamen Referenzfrequenz.

Der eine Synthesizer ist dabei für einen Frequenzbereich von 200 MHz bis 2 GHz ausgelegt, der andere für einen Bereich von 50 MHz bis 1,1 GHz.

Letzterer wird fest auf eine Sollfrequenz von 1 GHz eingestellt, während der Erstgenannte zwischen 1040 MHz und 1,5 GHz variiert werden kann.

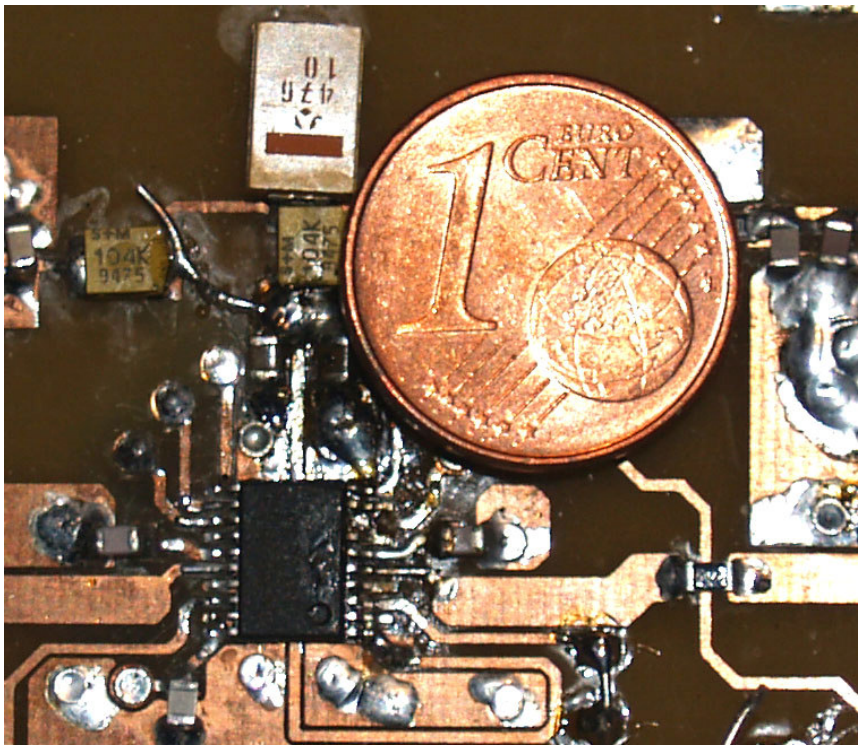


Abbildung 4 : PLL-IC „LMX2336“

4.4.1 Funktionsweise der PLL

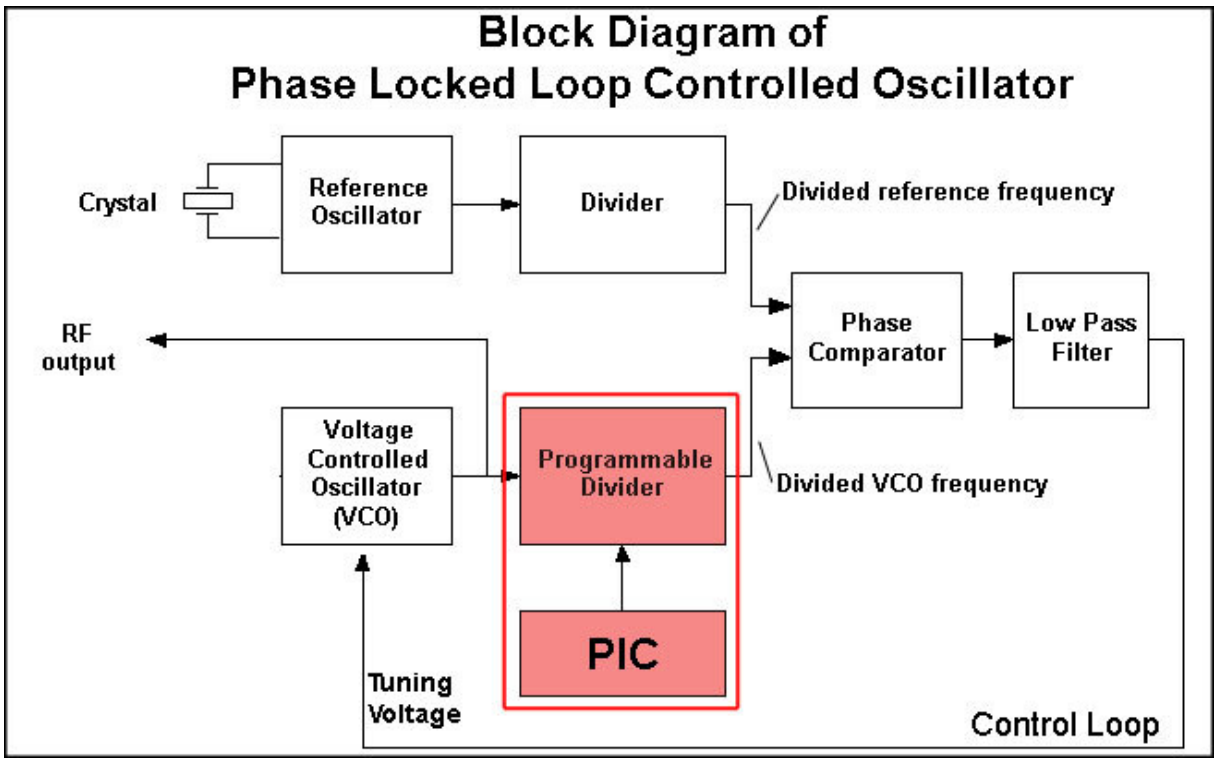


Abbildung 5 : Funktionsweise einer Phase Locked Loop

Eine PLL benötigt zwei Eingangssignale:

- Eingangssignal des Oszillators, der geregelt werden soll:
  Ein Oszillator, dessen Frequenz proportional zu einer angelegten Abstimmspannung ist (VCO, Voltage Controlled Oscillator)
- Referenztakt aus dem Referenzoszillator

Durch programmierbare Frequenzteiler (Divider, Vorteiler, Prescaler) werden beide Eingangssignale auf eine gemeinsame Vergleichsfrequenz heruntergeteilt und die Phasenlage dieser beiden Signale im Phasenvergleicher (Phase Comparator) verglichen.
„Läuft“ die Phase des zu regelnden Oszillators weg, wird dieser über die Regelschleife nachgestimmt.
Durch die PLL kann somit die heruntergeteilte Vergleichsfrequenz eines abstimmbaren Oszillators, der keine gute Langzeitstabilität hat, phasenstarr an die heruntergeteilte Vergleichsfrequenz eines Referenzoszillators, welcher nur einen sehr schmalen Frequenz-Variationsbereich, dafür aber eine ausgezeichnete Langzeitstabilität aufweist, gekoppelt werden, wodurch der abstimmbare Oszillator die Stabilität des Referenzoszillators aufweist.

Da der Ausgang des Phasenvergleichers nur die Werte „0“ und „1“ kennt (entweder die Phase des regelbaren Oszillators hinkt der Referenzphase hinterher oder sie eilt dieser voraus) muss an dessen Ausgang ein Tiefpassfilter geschaltet werden, welches als Integrator wirkt.

Je tiefer die Grenzfrequenz dieses Filters ist, desto langsamer reagiert der nachzuregelnde Oszillator auf das Signal des Phasenvergleichers; dies hat zwar einerseits zur Folge, dass es beispielsweise nach dem Einschalten relativ lange braucht, bis sich der regelbare Oszillator auf seinen Sollwert eingestellt hat, andererseits wird aber auch das Rauschen besser unterdrückt, was für ein schmales, phasenrauscharmes Ausgangssignal des Oszillators sorgt.
Soll dieser Oszillator nun aber in großen Schritten abstimmbar sein, wie es im Signalgenerator der Fall ist, ist ein schmaler Tiefpass unbrauchbar, da immer relativ lange abgewartet werden müsste, bis sich die Sollfrequenz eingestellt hat.
Für große Abstimmschritte dimensioniert man das Filter daher etwas breiter; dies verschlechtert zwar das Phasenrauschen und bedingt leichte „Überschwinger“ bei großen Frequenzsprüngen, dafür ist der Oszillator aber schnell abstimmbar, was in dieser Anwendung relevanter ist.

Ein potentieller Optimierungsansatz wäre es, Filter umschaltbarer Bandbreite einzusetzen, die während des Abstimmvorgangs (also beim Frequenzwechsel) auf die große Bandbreite schalten und wenn die Sollfrequenz erreicht ist, auf die schmale Bandbreite zurückwechseln.
Der PLL-Baustein wäre sogar für eine solche Funktion vorgesehen:
Er ist mit einer sog. „Fast Lock“-Funktion ausgestattet, mit welcher man während des Abstimmvorgangs den Strom, welcher der Phasenvergleicher am Ausgang liefert, erhöht werden kann.
Nach erfolgter Abstimmung wird der Strom dann wieder zurückgeregelt, was den gleichen Effekt wie die umschaltbaren Filter zur Folge hat.

Da dies aber in der Anwendung im HF-Labor nicht gefordert ist, wird es vorerst nicht implementiert und könnte eine mögliche Verbesserung des Systems darstellen.

Als Referenzoszillator kommt ein temperaturkompensierter Quarzoszillator (TCXO), wie er beispielsweise in Mobiltelefonen Verwendung findet, zum Einsatz.

4.4.2 Ansteuerung der PLL

Der PLL-Baustein wird über ein serielles Zweidraht-Interface angesteuert.
Über eine Leitung wird dabei der Takt, über die andere werden die Daten übertragen.
Des Weiteren muss während des Übertragens von Daten an die PLL deren „Latch Enable“- Pin aktiviert werden, damit das IC die Daten annimmt, wodurch mehrere Zweidraht-Geräte an einem gemeinsamen Bus betrieben werden können, da die Auswahl des gewünschten Teilnehmers über die „Enable“-Leitung erfolgt.

4.4.3 Setzen der PLL-Register

Die PLL verfügt über mehrere Register, welche die Vorteiler programmieren.
Das Setzen dieser Register erfolgt analog für beide Kanäle der PLL.

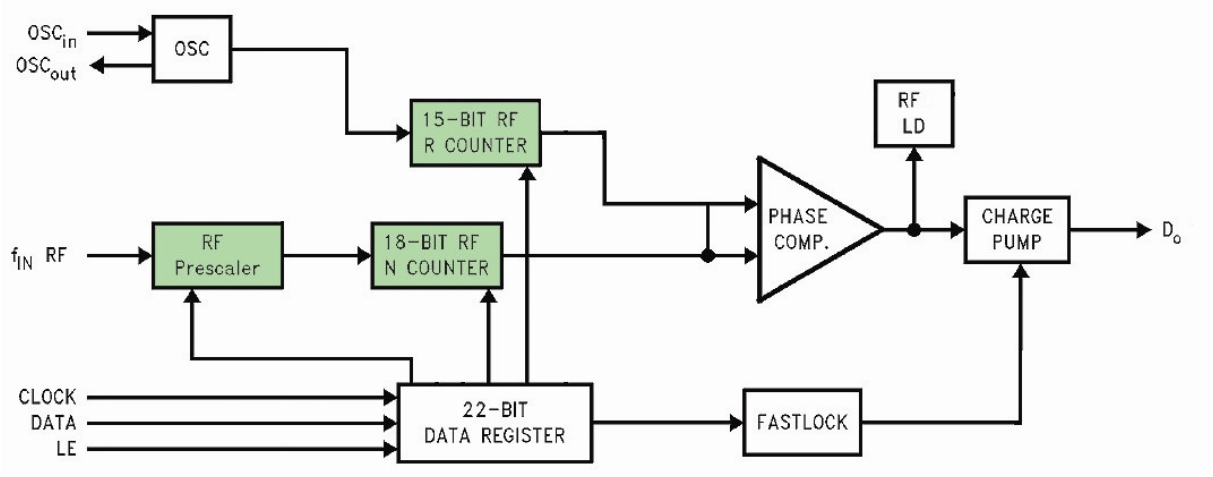


Abbildung 6 : Aufbau eines PLL-Kanals

Um eine Frequenz einzustellen, muss das korrekte Datenwort in drei Register geschrieben werden:

- „R counter“: Faktor, durch den das Referenzsignal geteilt wird
- „Prescaler“: Kann auf 64 oder 128 gesetzt werden
- „N counter“: Faktor, um den das vorgeteilte Signal zum 2. Mal geteilt wird; dieser besteht aus einem 11 Bit Counter „B“ und einem 7 Bit Counter „A“

Die Sollfrequenz des abstimmbaren Oszillators ergibt sich nach folgender Formel in Abhängigkeit von den verschiedenen Teilern:

$$fvco = \left[(P * B) + A\right] * \frac{Fref}{R}$$

Formel 1 : PLL Frequenzberechnung

Um später ein Abstimmraster von 100 kHz zu ermöglichen, muss die Vergleichsfrequenz zu 100 kHz gewählt werden.
Da der verwendete TXCO eine Frequenz von 13 MHz hat, muss folglich das „R“ – Register dezimal mit dem Wert 130 geladen werden:

$$\frac{Fref}{R} = \frac{13000kHz}{130} = 100kHz$$

Formel 2 : Abstimmraster

Aus Formel 1 : PLL Frequenzberechnung ergibt sich also, dass die Frequenz des abstimmbaren Oszillators immer ein Vielfaches des Abstimmrasters ist.

4.5 Amplitudenregelung

Die Regelung der Amplitude erfolgt über zwei Festwertabschwächer und ein variables Dämpfungsglied, welches durch PIN-Dioden realisiert ist.

4.5.1 Festwertabschwächer

Die Festwertabschwächer sind mit Relais aufgebaut, welche einen Transistor als Treiber haben, da der Microcontroller maximal 25 mA je Port-Pin liefern kann.
Da die Relais nur die Schaltzustände „an“ und „aus“ annehmen können, wird hierfür lediglich ein binärer Ausgang des Controllers pro Relais benötigt.
Anhand der empfangenen Sollamplitude schaltet der Controller die Dämpfglieder zu oder überbrückt diese.

4.5.2 Variabler Abschwächer mit PIN-Dioden

Beim PIN-Diodenabschwächer handelt es sich um ein spannungsgesteuertes Dämpfungsglied, bei dem die Einfügedämpfung umso größer wird, je weiter man den Strom durch die Dioden absenkt.
Angesteuert wird der PIN-Dämpfungssteller über einen PWM-Ausgang (Pulsweitenmodulation) des Microcontrollers, dem ein Tiefpass zur Integration nachgeschaltet ist.
Somit kann die Sollamplitude mittels der Steuerspannung vom Controller vorgegeben werden.
Um den Frequenzgang der Zwischenverstärker und vor allem des Endstufenmoduls, welches für einen Bereich von 40 .. 500 MHz spezifiziert ist und damit das bandbegrenzende Bauelement in diesem Design ist, weitgehend zu kompensieren, befindet sich zwischen dem PWM-Ausgang des Microcontrollers und dem Regeleingang des PIN-Dämpfgliedes ein Komparator, dessen invertierender Eingang mit einem Diodendetektor verbunden ist, der einen Teil der Hochfrequenz am Ausgang des Endstufenmoduls auskoppelt und gleichrichtet.
Diese Spannung ist proportional zur Ausgangsleistung. Wird eine Sollspannung vom Controller vorgegeben und beim Frequenzwechsel steigt die Ausgangsleistung geringfügig an, so wird auch die Spannung am invertierenden Eingang des Komparators größer, wodurch die Regelspannung des PIN-Abschwächers abgesenkt wird, was zu einer Reduzierung der Ausgangsleistung führt.
Das Dämpfungsverhalten des PIN-Dämpfungsgliedes selbst ist weitgehend frequenzunabhängig:

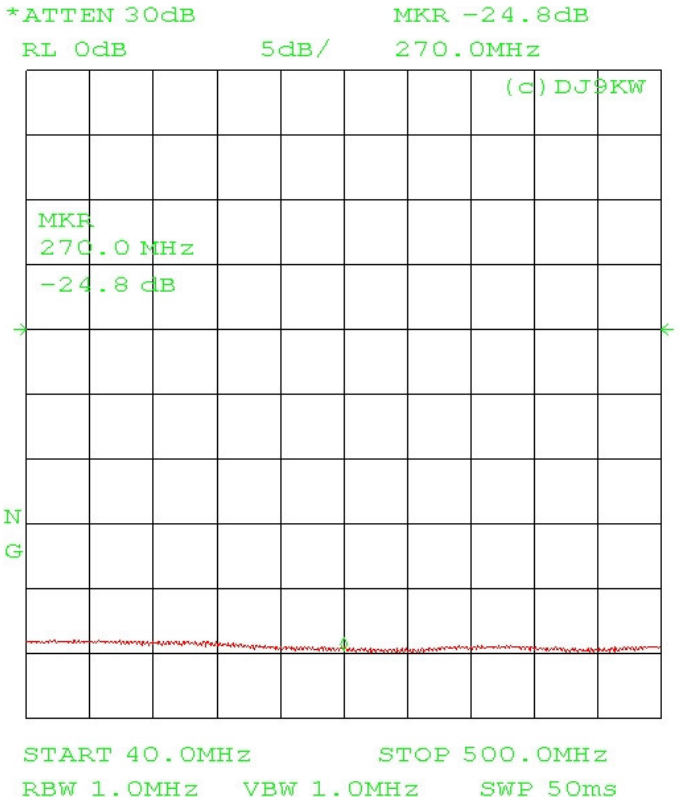


Abbildung 7 : Gemessene Durchgangsdämpfung bei einer Solldämpfung von 29 dB

In der Praxis hat sich dies bestätigt; der Pegelverlauf über der Frequenz ist nahezu gerade, wie man in der linken Abbildung erkennen kann.
Die Solldämpfung von 29 dB, welche per Software eingestellt wurde, wird auf 0,5dB genau über den gesamten Frequenzbereich von 40 .. 500 MHz eingehalten.

Die vertikale Auflösung beträgt 5dB / Division

4.5.3 Amplitudenregelung

Die Regelung der Ausgangsamplitude erfolgt in vier Stufen:

- Nur das PIN-Dämpfglied
- 15dB Fest-Dämpfglied + PIN-Dämpfglied
- 25dB Fest-Dämpfglied + PIN-Dämpfglied
- 15dB Fest-Dämpfglied + 25dB Fest-Dämpfglied + PIN-Dämpfglied

Folgende Schritte sind zum Einstellen der Amplitude nötig:

1. Einlesen der Sollamplitude P_{soll} vom Bedienteil über die serielle Schnittstelle
2. Einlesen der Maximalamplitude P_{max} (d.h. die Ausgangsamplitude bei 0dB Dämpfung) bei der eingestellten Frequenz aus der Kalibriertabelle im EEPROM
3. Berechnen der Dämpfung:

$$Dämpfung = P_{max} - P_{soll}$$

Formel 3 : Berechnung der Dämpfung

4. Durch Vergleichen der berechneten Dämpfung mit den in der Kalibriertabelle abgespeicherten Werten für das 15dB und 25dB Dämpfglied die Dämpfglieder entsprechend setzen und verbleibende Dämpfung berechnen
5. Die verbleibende Dämpfung an die Steuerfunktion für das PIN-Dämpfglied übergeben

Voraussetzung ist, dass zuvor eine vollständige Kalibrierung des Gerätes vorgenommen wurde.

Beispiel:

P_{soll} = -5 dBm
P_{max} = 16 dBm
Dämpfung des 15 dB Dämpfglieds = 15,3 dB
Dämpfung des 25 dB Dämpfglieds = 25,8 dB

3.: Dämpfung = $P_{max} – P_{soll}$ = 16 dBm – (-5 dBm) = 21 dB
4.: 21 dB < 25,8 dB ➔ 25 dB Dämpfglied nicht nötig
21 dB > 15,3 dB ➔ 15 dB Dämpfglied wird zugeschaltet
Darüber hinaus benötigte Dämpfung: 21 dB – 15,3 dB = 5,7 dB
5.: Das PIN-Dämpfglied auf eine Dämpfung von 5,7 dB einstellen

Beispiel 1 : Berechnung der Dämpfung aus gegebenem Sollwert

4.6 Kalibrierung

Aufgrund der Exemplarstreuungen und dynamischen Eigenschaften der Bauelemente muss das Gerät kalibriert werden, um die angestrebte Genauigkeit zu erreichen.
Dies betrifft insbesondere den Amplitudengang des gesamten Systems über der Frequenz und die Dämpfung der Abschwächer.

4.6.1 Kalibrierung der maximalen Ausgangsamplitude über der Frequenz

Zu Beginn muss die Ausgangsamplitude des Systems mit minimaler Einfügungsdämpfung ermittelt werden, d.h. die Festwertabschwächer sind abgeschaltet und das PIN-Dämpfungsglied wird auf eine Einfügungsdämpfung von 0 dB programmiert.
Da im EEPROM nur begrenzt viel Speicherplatz zur Verfügung steht, wurde als Kompromiss zwischen Genauigkeit und Speicherbedarf beschlossen, alle 5 MHz eine Stützstelle zu bestimmen.
In der Praxis wird der Ausgang des Signalgenerators mit einem Leistungsmesser verbunden und das Kalibrierprogramm gestartet.
Das Programm beginnt bei 40 MHz und inkrementiert die Frequenz in 5 MHz - Schritten, wobei die jeweils gemessene Ausgangsleistung eingegeben und im EEPROM abgespeichert wird.
Somit ist dem Generator bekannt, wie groß die maximal mögliche Leistung bei jeder Frequenz ist und die erforderliche Dämpfung für eine bestimmte Ausgangsleistung kann gemäß *Beispiel 1* in *Abschnitt 4.5.3* bestimmt werden.

4.6.2 Kalibrierung der Festwertabschwächer

Die Kalibrierung der Festwertabschwächer erfolgt analog zu der des PIN-Dämpfglieds:
Um die Durchgangsdämpfung des jeweiligen Abschwächers zu bestimmen, muss zunächst eine Kalibrierung des PIN-Dämpfglieds vorliegen, damit das System die Maximalamplitude ohne zugeschaltete Dämpfglieder im Speicher abgelegt hat,
Nun wird zunächst nur das 15dB-Dämpfglied und anschließend nur das 25dB-Dämpfglied zugeschaltet.
Mit einem Leistungsmesser am Ausgang des Generators wird die Ausgangsamplitude ermittelt und am Bedienteil eingestellt, worauf die Steuersoftware aus der Maximalamplitude und der gemessenen Ausgangsamplitude die Dämpfung errechnet und im EEPROM ablegt.

5. Entwicklung der Software des HF-Teils

Um die Übersichtlichkeit bei einem umfangreichen Projekt wie diesem zu gewährleisten, sollten einige Regeln bei der Entwicklung des Programmcodes berücksichtigt werden:
Es ist nicht möglich, den ganzen Programmcode „am Stück“ zu entwickeln; vielmehr entsteht das Gesamtprogramm aus vielen einzelnen Bausteinen, die zum Schluss zusammengefügt werden.
Diese Bausteine sind modular einsetzbar und können für sich getestet werden, was die Entwicklung einfacher und rationeller macht, gleichzeitig aber auch die Wiederverwendbarkeit des Programmcodes in anderen Projekten gewährleistet.
Daher ist es sinnvoll, die einzelnen Programmteile als getrennte Funktionen zu realisieren und diese in jeweils eigene Dateien zu speichern.
Dies wären im HF-Teil beispielsweise Dateien für

- die Berechnung der PLL-Steuerworte und die Programmierung der PLL
- die Amplitudenregelung
- die Kalibrierroutine
- häufig benötigte arithmetische Funktionen
- häufig benötigte Funktionen zur ASCII-Zeichenkettenbearbeitung

Somit können einige Programmbestandteile wie beispielsweise neu implementierte arithmetische Funktionen aus dem HF-Teil direkt für die Software des Bedienteils wiederverwendet werden.

5.1 Begriffsdefinition : Der Datentyp „string“

Während der Variablentyp „string“ in Programmiersprachen wie Delphi oder Turbo Pascal zu den Standardtypen gehört, wird man ihn unter C vergeblich suchen.
Es handelt sich hierbei um eine Variable, in welcher eine ASCII-Zeichenkette abgelegt werden kann.
Unter C gibt es hingegen nur den Datentyp „char“, welcher einen einzelnen Buchstaben aufnehmen kann.
Soll unter C mit Zeichenketten gearbeitet werden, so ist es erforderlich, mehrere „char“-Variablen (eng. Char = Buchstabe) zu einem „Array of characters“, zu Deutsch einem „Buchstabenfeld“, aneinanderzureihen.
Diesen Datentyp werde ich im Folgenden der Einfachkeit halber als „string“ bezeichnen, wie es in anderen Programmiersprachen üblich ist.
Damit der Compiler erkennt, wann eine Zeichenkette zu Ende ist, wird dieser die sog. „Nullterminierung“ (“\0“, ASCII-Zeichen 0) angehängt, welche als reserviertes Steuerzeichen dient.

5.2 Auswertung der Steuerkommandos

Zunächst muss der Microcontroller auf den Empfang von Steuerworten warten. Diese besitzen Buchstaben als führende Steuerzeichen und danach den zu übernehmenden Wert, wie bereits in *Abschnitt 4.3* Die serielle Kommunikation erläutert wurde.

Um die Information zu verwerten muss der Microcontroller anhand des führenden Buchstaben erkennen, um was für einen Befehl es sich handelt; anschließend wird der führende Buchstabe entfernt und die dahinter stehende Zeichenkette in eine Zahl umgewandelt.
Die Auswertung des Steuerzeichens erfolgt mittels eines „switch-case“ – Statements; wird ein ungültiges Zeichen empfangen, so wird dieses ignoriert und weiter auf ein gültiges Kommando gewartet.

Leider wurde die mit dem CCS-Compiler ausgelieferte Funktion zum Empfang von Zeichenketten via RS232 den Ansprüchen nicht gerecht, da die empfangenen Daten auf eine Länge von sechs Zeichen abgeschnitten wurden, wodurch es nötig war, eine eigene Funktion zum Einlesen zu entwickeln.
Die Funktion zum umwandeln von Zeichenketten in Dezimalzahlen war ebenfalls suboptimal, da sie erstens nicht für Zahlen größer 65536 ausgelegt war und zusätzlich nur für vorzeichenbehaftete Zahlen gedacht war, was den Gültigkeitsbereich auf -32786 .. 32787 einschränkte.
Da die Sollfrequenz in der Einheit Kilohertz übertragen werden soll, um aufwendige Komma-Arithmetik zu sparen, muss der Wertebereich mindestens Ganzzahlen bis 500000 umfassen.
Somit ist es nicht ausreichend, die Vorzeicheninterpretation abzuschalten; es muss ein größerer Datentyp gewählt werden.
Der nächstgrößere Bereich kann durch 32Bit Integer – Zahlen abgedeckt werden; der Wertebereich für nicht-vorzeichenbehaftete Zahlen reicht hier von 0 bis 4294967296.
Da der Compiler von Haus aus nicht mit den nötigen Funktionen für solch große Zahlen ausgestattet ist, wurde eine geeignete Funktion unter Zuhilfenahme der 16Bit Umwandlungsfunktion geschrieben.

Um auch bei der Amplitude auf Kommazahlen verzichten zu können, werden die Sollpegel in (dBm * 10) übertragen, d.h. aus -10,5 dBm wird die Zahl -105.
Bei der Amplitude ist somit eine Darstellung mittels vorzeichenbehafteten 16Bit Ganzzahlen ausreichend.
Zudem bietet dieser Datentyp gegenüber einer Kommadarstellung, welche 32 Bit Speicher je Zahl benötigt, dass nur zwei 8Bit-Speicherplätze je Kalibrierdatum im EEPROM belegt werden, wodurch eine feinstufigere Kalibrierung möglich wird.

5.3 Das SPI-Interface

Um den PLL-Baustein programmieren zu können, müssen die Daten über ein SPI-Interface an diesen übertragen werden.
Hierbei handelt es sich um ein serielles Interface, welches drei Leitungen benötigt:

- Die seriellen Daten
- Der Takt
- Die „Latch Enable“ – Leitung

Das „Latch Enable“ hat die Aufgabe, den Baustein, für welchen die gesendeten Daten bestimmt sind, auszuwählen und dessen Datenempfänger zu aktivieren. Dies ist relevant, wenn mehrere SPI-fähige ICs an einem gemeinsamen Datenbus angeschlossen werden, damit jeweils nur ein Baustein die Daten verarbeitet, da es beim SPI-Protokoll im Gegensatz zu beispielsweise I²C keine Adressierung von Empfängern gibt.

Zwar liefert der CCS-Compiler eine Funktion mit, welche ein SPI-Master-Interface bereitstellt, diese stellte sich jedoch als unflexibel heraus, da ihr die zu sendenden Daten in 8Bit-Blöcken übergeben werden müssen, und der Aufwand, das 24Bit Steuerwort für die PLL so zu formatieren, dass es zu dieser Routine passt, wäre grösser gewesen als eine eigene Routine zu schreiben, welche die Daten im einem Block ausgibt.

Da lediglich ein SPI-Empfänger angesteuert werden muss, wurde die Berechnung der PLL-Register und die SPI-Emulation in einer eigenen Unterfunktion gekapselt, welcher lediglich die Frequenz in Kilohertz übergeben wird, auf die die PLL rasten soll. Sämtliche Berechnungen werden innerhalb dieser Funktion durchgeführt.

Der Übersichtlichkeit wegen sei an dieser Stelle auf den entsprechenden Code-Abschnitt im Anhang verwiesen, da diese Funktion sehr umfangreich ist.

5.4 Pulsweitenmodulation

5.4.1 Was ist „PWM“ ?

Im Gegensatz zu einem analogen Signal, wleches jeden beliebegen Wert annehmen kann, gibt es bei einem PWM-Signal nur zwei Zustände: logisch „0“ oder logisch „1“; man kann den Ausgang einer PWM-Stufe also mit einem Schalter vergleichen, der entweder an oder aus ist.
Um dennoch verschiedene Werte darstellen zu können, wird bei der PWM das Tastverhältnis, d.h. die Einschaltdauer, bei konstanter Abtastfrequenz moduliert und anschließend über ein Tiefpassfilter integriert.
Beträgt diese Abtastfrequenz beispielsweise 100 Hz, so kann der PWM-Ausgang immer logisch „0“ liefern, was zu einer Ausgangsspannung von 0 V führt.
Liefert die PWM-Stufe dagegen über die gesamte Periode logisch „1“, so beträgt die Ausgangsspannung über die gesamte Zeit 5V, wenn man ein TTL-Digitalsystem zu Grunde legt.
Soll eine Amplitude, welche zwischen diesen beiden Werten liegt, erzeugt werden, so wird das PWM-Signal eine gewisse Zeit innerhalb dieser Periode eingeschaltet und anschließend bis zu deren Ende abgeschaltet.
Um eine Spannung von 2,5 Volt am Ausgang zu erreichen, muss die Einschaltdauer folglich 50 % betragen.

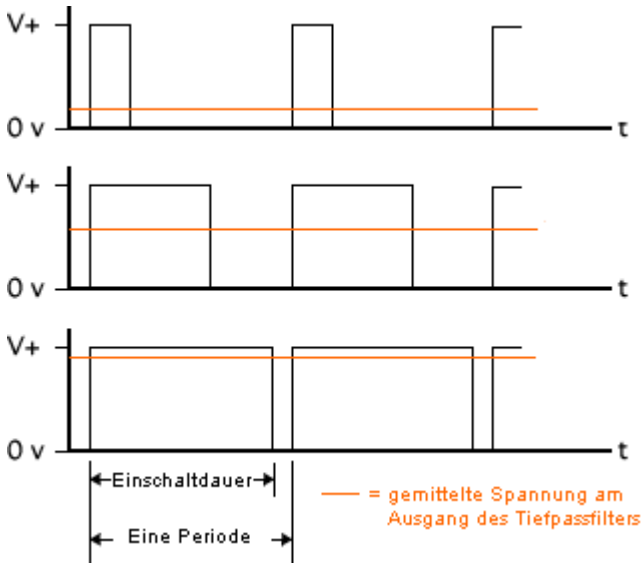


Abbildung 8 : Funktionsweise der PWM

5.4.2 Implementierung der PWM-Routine

Im Gegensatz zur SPI-Routine konnte die bereits im CCS-Compiler integrierte Routine zur Erzeugung der PWM-Signale übernommen werden.
Dieser Routine wird eine Zahl zwischen 0 und 1023 übergeben, wobei 0 einer Ausgangsspannung von 0 Volt und 1023 einer Ausgangsspannung von 5 Volt nach dem Tiefpass entspricht.
Jede Periode wird folglich in 1024 Abschnitte aufgeteilt, welche die Quantisierungsstufen bilden.

5.5 Umsetzung dieser Bedingungen in Software

Im Folgenden möchte ich einige Funktionen, die der internen Datenverarbeitung des Microcontrollers im HF-Teil dienen, näher vorstellen.
Es handelt sich hierbei um Funktionen, die nicht (oder nicht in gewünschter Form) zum Lieferumfang des CCS-Compilers gehören und für dieses Projekt benötigt wurden.
Der vollständige C-Code befindet sich im Anhang dieser Dokumentation.

5.5.1 Einlesen von Strings über die serielle Schnittstelle

Wie eingangs erwähnt hatte die dem CCS-Compiler beiliegende Funktion Probleme mit Strings länger als sechs Zeichen.
Folglich wurde auch an dieser Stelle eine neue Funktion geschrieben:

```
void get_string_neu(char *s)
  {
  int len;
  char c;
  len = 0;

  do
    {
    c = getc();
    s[len] = c;
    len++;
    } while(c != 13); //CR
  s[--len] = 0x00;
  }
```

Codeauszug 1 : Einlesen von Zeichenketten über die serielle Schnittstelle

Der Funktion wird ein Zeiger auf eine String-Variable übergeben, in die jedes empfangene Zeichen geschrieben wird.
Wird das „Carriage Return“-Zeichen (ASCII-Zeichen 13) empfangen, so ist das Ende der Zeichenkette erreicht und der String wird durch Anfügen der Nullterminierung abgeschlossen.

5.5.2 Umwandlung vom Datentyp String in den Datentyp Int32

Um das Steuerwort zum Setzen der Ausgangsfrequenz verarbeiten zu können, ist es erforderlich, eine ASCII-Zeichenkette in eine nicht vorzeichenbehaftete 32Bit Ganzzahlvariable umzuwandeln.
Folglich müssen Werte zwischen 40000 (kHz) und 500000 (kHz) konvertiert werden. Da dem CCS-Compiler lediglich eine solche Routine für 16Bit – Ganzzahlen beilag, wurde diese als Basis für die folgende neu entstandene 32Bit Routine genutzt, welche in *Codeauszug 2* : Umwandlung des ASCII-Strings in eine 32Bit Ganzzahlzu sehen ist.

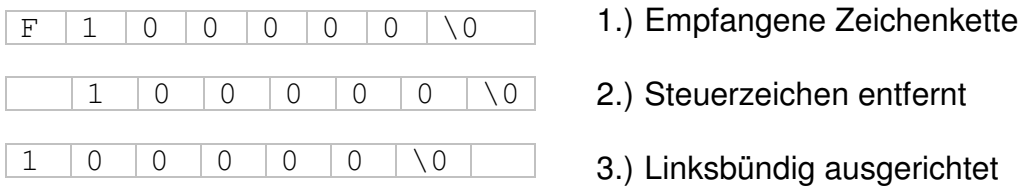
```
unsigned int32 atoui32_neu (char *s)
{
  unsigned int32 result;
  int sign, base, index;
  char c;

  index = 0;
  result = 0;

  if(s)
    c = s[index++];


  if (c >= '0' && c <= '9')
  {

     {
      while (c >= '0' && c <= '9')
      {
        result = (result << 1) + (result << 3);  // result *= 10;
        result += (c - '0');
        c = s[index++];
      }
    }

  }


  return(result);
}
```

Codeauszug 2 : Umwandlung des ASCII-Strings in eine 32Bit Ganzzahl

Der Funktion „atoi32“, was für „**A**scii **to** unsigned **i**nteger **32**“ steht, wird ein Zeiger auf die Stringvariable s übergeben.
Im Folgenden wird überprüft, ob das übergebene Zeichen eine Zahl im Bereich 0 bis 9 ist oder ob die Nullterminierung erreicht wurde, womit die Umwandlung abgeschlossen ist.
Wurde eine gültige Ziffer gefunden, wird diese von der ASCII-Darstellung in das entsprechende numerische Symbol übersetzt und die Bearbeitung bei dem nachfolgenden ASCII-Zeichen fortgesetzt, während die bearbeitete Stelle der umgewandelten Ganzzahl um eine Zehnerpotenz herabgesetz wird.

5.5.3 Formatierung der empfangenen Zeichenkette

Da die empfangenen Zeichenketten neben einem Zahlenwert immer ein Steuerzeichen (F für Frequenz, A für Amplitude) enthalten, können diese nicht direkt in Ganzzahlen umgewandelt werden.
Vor der Umwandlung muss dieses Steuerzeichen interpretiert und anschließend entfernt werden.
Die Auswertung erfolgt, indem das erste Zeichen des empfangenen Strings isoliert wird.
Je nachdem, um welchen Buchstaben es sich handelt, wird in das entsprechende Unterprogramm gewechselt.
Anschließend wird das erste Zeichen des Strings entfernt und der Rest eine Stelle nach links verschoben:



1.) Empfangene Zeichenkette

2.) Steuerzeichen entfernt

3.) Linksbündig ausgerichtet

Beispiel 2 : Formatierung empfangener Zeichenketten zur Weiterverarbeitung

Das Entfernen des Steuerzeichens wird durch folgende Routine erledigt:

```
void remove_first(char *s)
  {
  int l, i;
  i = 0;
  l = strlen(s);

  do {
     s[i] = s[i+1];
    i++;
     }while(i<l && s[i]!=0);
  }
```

Codeauszug 3 : Formatierung empfangener Zeichenketten

Anstatt das führende Element zu entfernen, wird es einfach durch „nach links rücken“ des zweiten Elements überschrieben.
Jedes Element wird nach links verschoben, einschließlich der Nullterminierung.
Wurde diese verschoben, ist die Routine beendet.

5.5.4 EEPROM-Zugriff

Der gewählte PIC-Microcontroller verfügt über 258 Bytes internes Flash-EEPROM, welches in 256 8Bit-Speicherzellen organisiert ist.
Um eine ausreichende Anzahl von Kalibrier-Stützstellen für die Amplitudenregelung im EEPROM ablegen zu können, musste eine Möglichkeit gefunden werden, die Festkommadarstellung, welche 32Bit Speicher belegt, in ein platzsparenderes Format zu konvertieren, da in dieser Anwendung nur eine Stelle hinter dem Komma relevant ist.
Nach verschiedenen Überlegungen hat sich nachfolgende Methode bewährt, um die 32Bit Float-Zahlen mit einer Genauigkeit von einer Nachkommastelle als 16Bit-Ganzzahlen darzustellen, wodurch sich der Speicherbedarf halbieren ließ

Um die Kalibrierdaten für die Amplitude im EEPROM abzulegen, werden diese nach folgendem Schema codiert:

1.) Multiplizieren der Kommavariable („float“) mit 10,0
2.) Ergebnis als 16Bit-Ganzzahl abspeichern; Nachkommastellen gehen dabei verloren
3.) Diese 16Bit-Ganzzahl in zwei 8Bit-Blöcke aufspalten, welche nacheinander im EEPROM unter der angebenenen Adresse abgelegt werden können

Das Auslesen der Pegelwerte aus dem EEPROM erfolgt in umgekehrter Reihenfolge:

1.) Ersten 8Bit-Block aus dem EEPROM auslesen
2.) Zweiten 8Bit-Block aus dem EEPROM auslesen und an den ersten anhängen, wodurch sich wieder eine 16Bit Ganzzahl ergibt
3.) Die 16Bit Ganzzahl in eine „float“-Variable umwandeln
4.) Diese „float“-Variable durch 10,0 dividieren

Ablegen eines Werts im EEPROM:

Abzuspeichernder Pegel: 15,4 (dBm)

1.) 15,4 * 10,0 = 154,0
2.) 154,0 ➔ 154
3.) Bilden der 8Bit-Blöcke und Abspeichern im EEPROM

Auslesen aus dem EEPROM:

1.) & 2.) Auslesen der 2 * 8Bit ➔ 154
3.) 154 ➔ 154,0
4.) 154,0 / 10,0 = 15,4

Beispiel 3 : Datenkonvertierung für das Ablegen im EEPROM

Der schreibenden Funktion wird die Kommazahl und die Adrese übergeben, in welcher diese abgelegt werden soll, während die lesende Funktion lediglich die Speicheradresse des auszulesenden Datums übergeben bekommt und das Resultat als Rückgabewert an die aufrufende Funktion übergibt.

```
void write_eeprom_float (int address, float data)
{
  signed int16 data_int16;
  data_int16 = (long)(data * 10.0);
  address = address * 2;
  WRITE_EEPROM(address, data_int16>>8);
  WRITE_EEPROM(address+1, data_int16);
}
```

Codeauszug 4 : Datenaufbereitung und Schreiben ins EEPROM

```
float read_eeprom_float(int address)
{
  signed int16 data = 0;

  address = address * 2;
  data=read_eeprom(address);
  delay_ms(5);
  data=((data<<8)|(read  eeprom(address+1)));
```

Codeauszug 5 : Lesen aus dem EEPROM und Datentyp-Konvertierung

Selbstverständlich ließe sich die Genauigkeit durch eine Multiplikation bzw. Division mit beispielsweise 100 auf zwei Nachkommastellen steigern, was in dieser Anwendung jedoch keine Rolle spielt und deshalb auch nicht weiter verfolgt wurde.

Um die Zieladresse für die EEPROM-Zugriffe zu berechnen, entstand darüber hinaus eine kleine Funktion, welche aus der momentan eingestellten Frequenz die zutreffende Adresse bestimmt und an die aufrufende Funktion zurückgibt.

```
int get_eeprom_address()
  {
  return ((fsoll / 1000) - 40) / 5 ); // ➔  40 MHz ≡ 0 ; 500 MHz ≡ 92
  }
```

Codeauszug 6 : Bestimmen der Zieladresse aus der momentanen Frequenz fsoll

Eine detaillierte Auflistung der EEPROM-Belegung findet sich im Anhang.

5.5.5 Ansteuerung des PIN-Abschwächers

Diese Funktion hat die Aufgabe, die Spannung am Ausgang des Tiefpasses am PWM-Port so einzustellen, damit die gewünschte Dämpfung des PIN-Dämpfungsglieds erzielt wird.

Als Basis für diese Funktion wurde zunächst die Dämpfung in Abhängigkeit vom PWM-Steuerwort (0 ..1023) ermittelt und in einem Diagramm aufgetragen.

Anschließend wurden verschiedene Polynomannäherungen zu dieser Kurve gesucht, über welche sich das PWM-Steuerwort zu einer bestimmten Solldämpfung berechnen lässt.
Bei Tests mit zwei- bis sechspoligen Polynomen stellte sich heraus, dass die Kurve bei einer Simulation am Computer mit steigender Polynomordnung genauer wurde; bei der Implementation im PIC war hingegen bei Polynomordnungen größer 4 eine deutliche Abweichung in den Messergebnissen sichtbar, was sich auf Genauigkeitsprobleme bei der Darstellung sehr kleiner Zahlenwerte im Microcontroller zurückführen lässt.
Beispielhaft hierfür sei der Auszug aus dem Excel-Dokument auf der nächsten Seite, mit welchem die Qualität der Polynom-Annäherung untersucht wurde:
Das Polynom 4. Ordnung liefert bis zu einer Solldämpfung von 15dB vernünftige PWM-Werte, während das Polynom 6. Ordnung bereits bei einer Solldämpfung von 10dB deutlich abzuweichen beginnt.
Zwar liegt das Polynom höherer Ordnung anfangs näher am Sollwert, jedoch machen sich die Abweichungen hin zu kleinen PWM-Werten (also hohen Einfügedämpfungen) immer stärker bemerkbar:
Während zwischen 0dB und 1dB Einfügungsdämpfung 109 PWM-Werte liegen, so sind es zwischen 14dB und 15dB gerade noch 30 PWM-Werte.
Die Annäherungsfunktion muss folglich für größere Dämpfungen eine höhere Genauigkeit aufweisen, da Fehler hier stärker ins Gewicht fallen als bei kleinen Einfügungsdämpfungen.

Annäherungsfunktionen für die Dämpfung

		Messkurve (Messwerte)		*Testergebnis der Polynome im PIC*	
Att (dB)	Pout (dBm)	**PWM (R1)**		**6pol (R2)**	**5pol (R3)**	**4pol (R4)**
0	15	**1023**		**1020**	**1015**	**1006**
1	14	**914**		**918**	**926**	**932**
2	13	**852**		**851**	**856**	**865**
3	12	**803**		**800**	**799**	**804**
4	11	**756**		**753**	**748**	**748**
5	10	**706**		**706**	**700**	**695**
6	9	**654**		**656**	**653**	**645**
7	8	**602**		**606**	**606**	**598**
8	7	**552**		**557**	**559**	**553**
9	6	**505**		**513**	**513**	**510**
10	5	**462**		**475**	**469**	**468**
11	4	**423**		**447**	**429**	**428**
12	3	**386**		**429**	**394**	**389**
13	2	**353**		**423**	**365**	**353**
14	1	**321**		**431**	**343**	**319**
15	0	**291**				**288**
16	-1	**263**				
17	-2	**237**				
18	-3	**212**				

R2:	y = 0,0004x6 - 0,0232x5 + 0,5331x4 - 5,8263x3 + 31,5325x2 - 128,3196x + 1020,9207
R3:	y = -0,0022x5 + 0,1071x4 - 1,8296x3 + 14,5002x2 - 101,7515x + 1015,2949
R4:	y = 0,0060x4 - 0,2304x3 + 4,0848x2 - 77,5562x + 1006,1267

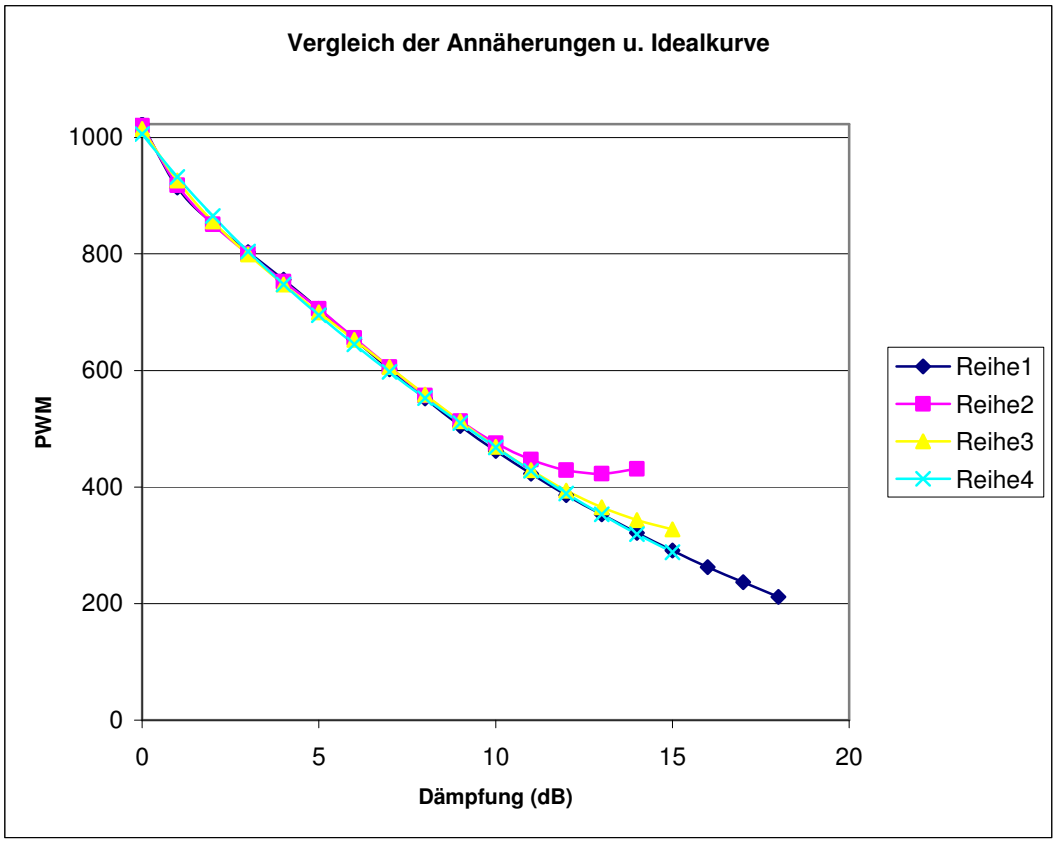


Abbildung 9 : Vergleich verschiedener Annäherungspolynome des PIN-Dämpfungsglieds

6. Entwicklung des Bedienteils

Bis zu diesem Punkt konnte der HF-Teil nur vom PC aus über die serielle Schnittstelle gesteuert werden.
Wie in der Einleitung bereits erwähnt, war es jedoch auch Bestandteil meiner Projektarbeit, ein Bedienteil zu entwickeln, mit welchem der HF-Teil gesteuert werden kann.

Ausgehend von der Forderung nach Display, Drehgeber und Tastern als Interaktionselemente wurden zunächst passende Bauelemente bestellt.
Als Microcontroller fiel auch im Bedienteil die Entscheidung zugunsten des 16F876 von Microchip, da im Zuge der Programmierung des Synthesizer-Teils viele Erfahrungen gesammelt wurden.
Als Display dient, der besseren Ablesbarkeit wegen, ein OLED-Display mit 2 Zeilen und 16 Zeichen je Zeile.
Um zwischen Frequenz- und Amplitudeneinstellung umzuschalten dienen 2 Taster; weitere 2 Taster dienen der Auswahl der zu ändernden Stelle (10er, 100er, …).
Ein weiterer Taster, ein Modell mit eingebauter Leuchtdiode, dient dem An- bzw. Abschalten des HF-Ausgangs; leuchtet die LED, ist der Ausgang aktiv.

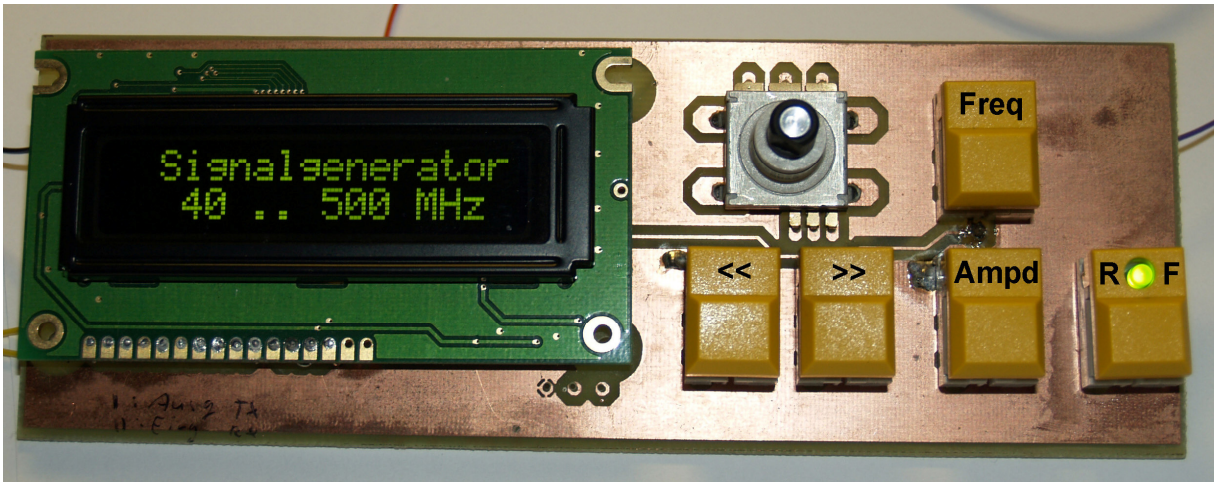


Abbildung 10 : Ansicht des fertig bestückten Bedienteils

Um das Bedienteil so kompakt als möglich zu machen, entschied ich mich zur ausschließlichen Verwendung oberflächenmontierter Bauelemente.
Als problematisch erweis sich dieses Vorhaben beim Microcontroller:
Dieser befindet sich in einem SOIC-Gehäuse (SOIC = **S**mall **O**utline **I**ntegrated **C**ircuit), welches direkt auf der Platine aufgelötet wird.
Im Schadensfall muss der PIC folglich ausgelötet werden, doch leider erwies sich die Klebeschicht unter den dünnen Leiterbahnen, welche das IC kontaktieren, als nicht sehr thermisch belastbar, was beim Tausch des Controllers zum Ablösen dieser führte.
Im HF-Teil ist dagegen ein Stecksockel vorhanden, was sich in der Entwicklungsphase diesbezüglich als nützlich erwies.

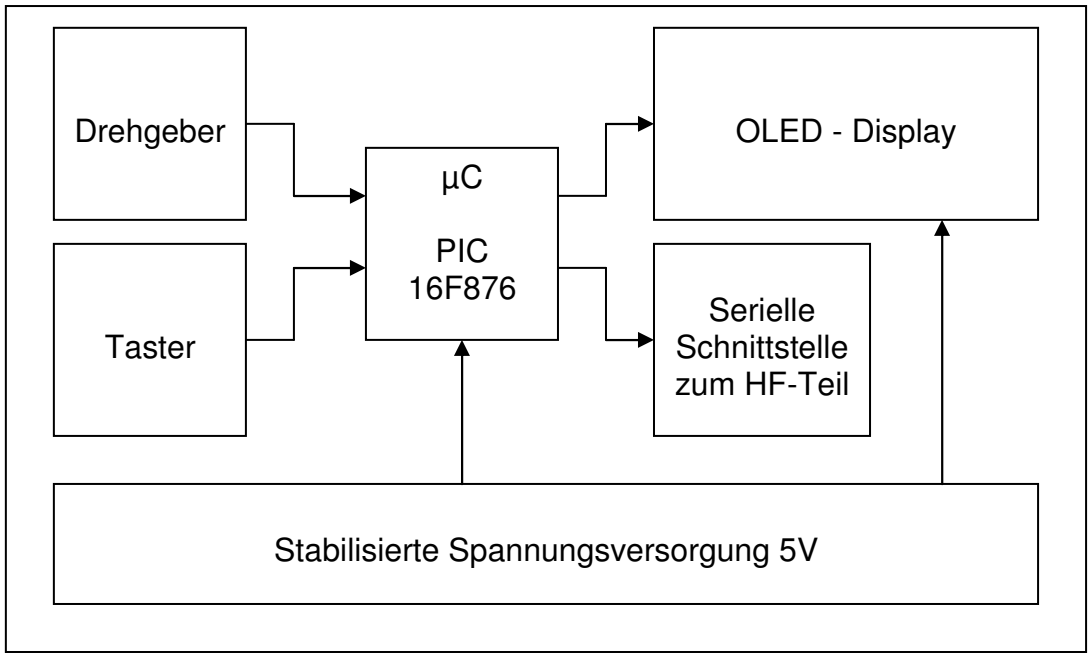


Abbildung 11 : Blockschaltbild des Bedienteils

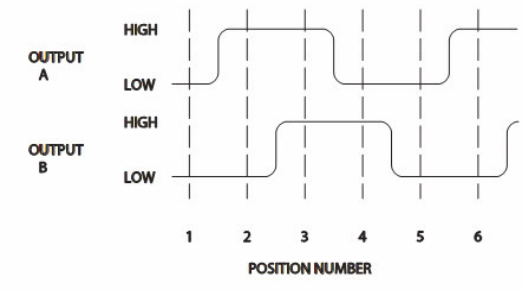
6.1 Hardware

6.1.1 Drehgeber

Beim Drehgeber handelt es sich um einen optischen Encoder vom Hersteller Grayhill.

Abbildung 12 : Drehgeber

Da die Schaltstellungen optisch erkannt werden, muss beim Verarbeiten der Daten im Microcontroller nicht per Software entprellt werden, wie dies bei mechanischen Kontakten nötig wäre.
Das Ausgangssignal des Drehencoders ist ein 2Bit-Quadratursignal (siehe Abbildung 13):



CLOCKWISE ROTATION		
POSITION	OUTPUT A	OUTPUT B
1		
2	•	
3	•	•
4		•

• INDICATES LOGIC HIGH; BLANK INDICATES LOGIC LOW. CODE REPEATED EVERY 4 POSITIONS

Abbildung 13 : Ausgangssignal des Drehgebers

Je nach Drehrichtung eilt die Phase von Signal B der von Signal A um 90° voraus oder hinterher; die Drehrichtung kann folglich ermittelt werden, indem man beispielsweise während der steigenden Flanke von Signal A überprüft, ob Signal B logisch „1“ oder „0“ ist.

6.1.2 Display

Als Display wurde ein OLED – Display des Herstellers „Electronic Assembly“ verwendet, welches in 2 Zeilen mit jeweils 16 Spalten den vollständigen ASCII-Zeichensatz beherrscht; zusätzlich können zehn benutzerdefinierte Zeichen angelegt werden.
Dieses Feature wurde beispielsweise bei der Einblendung einer Amplitude < 0 dBm genutzt:

Abbildung 14 : Anzeige des Displays

Um einen Abstand zwischen Frequenz und Amplitude darstellen zu können, musste anstatt des normalen Minuszeichens ein „Gekürztes“ verwendet werden, welches als benutzerdefiniertes Zeichen angelegt wurde, da ansonsten nicht genügend Stellen zur Verfügung gestanden wären.
Das Zeichen kann dazu als Bitmuster eingegeben werden und wird beim Programmstart in das Zeichen-RAM des Displays kopiert.

In der zweiten Zeile sieht man das “^“ – Symbol, welches momentan auf die 0,1er-Stelle der Frequenz zeigt. Somit wird beim Drehen des Encoders diese Stelle verändert.

Das OLED-Display ist Softwarekompatibel zu standard LC-Dot-Matrix-Displays mit HD44780 – Controllern.

6.1.3 Spannungsversorgung

Zur Bereitstellung der stabilisierten 5V-Betriebsspannung wurde ein Low-Drop-Spannungsregler in SMD-Ausführung vom Typ TS2940 verwendet.
Ein Low Drop – Regler wäre hier zwar nicht nötig gewesen, war aber noch übrig und wurde daher eingebaut.

6.1.4 In System Programming - Schnittstelle

Zum Programmieren des Microcontrollers wurde auch am Bedienteil eine In System-Programmierschnittstelle eingebaut.
Somit kann der PIC im Zielsystem umprogrammiert werden.

6.1.5 Entwicklung der Platine (Schaltplan + Layout)

Der Schaltplan des Bedienteils wurde mittels der Software „Eagle“ von CadSoft erstellt und anschließend entflochten.

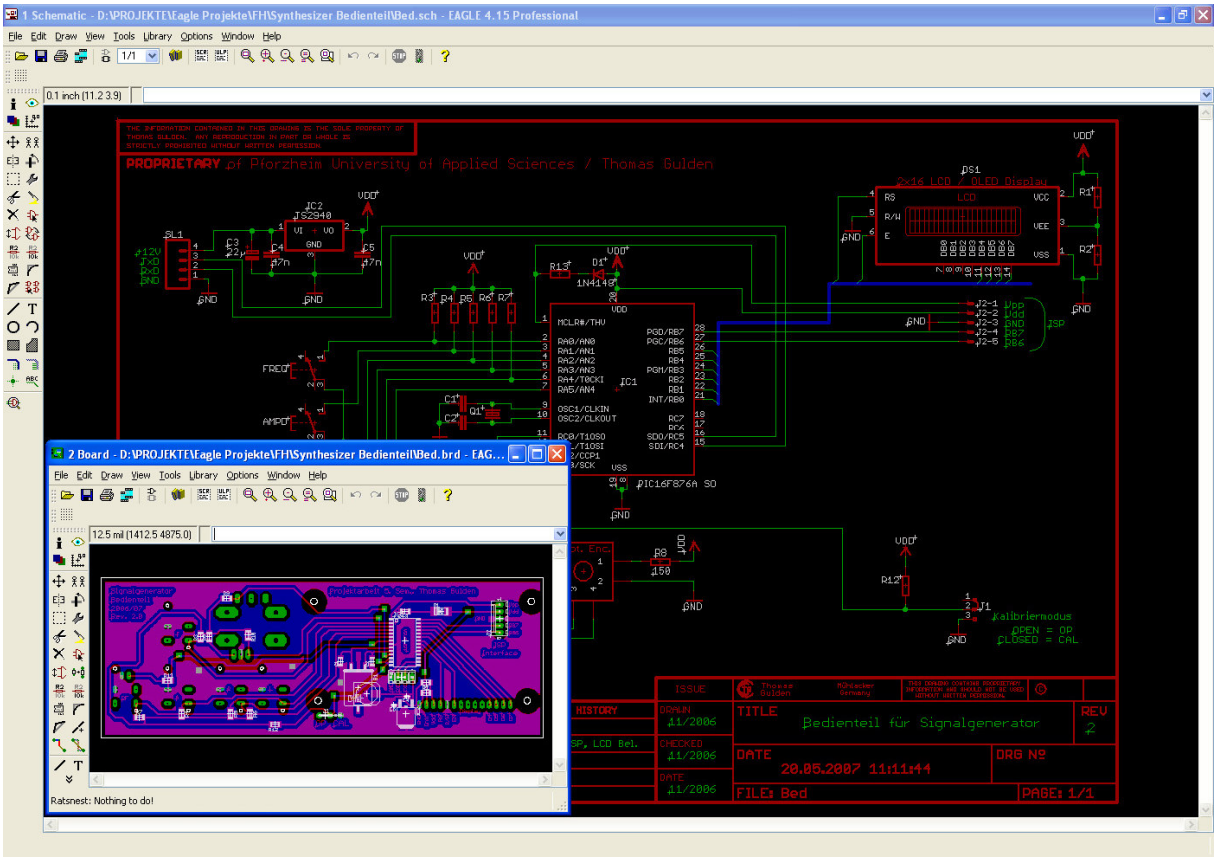


Abbildung 15 : Software zur Erzeugung des Schaltplans und Layouts

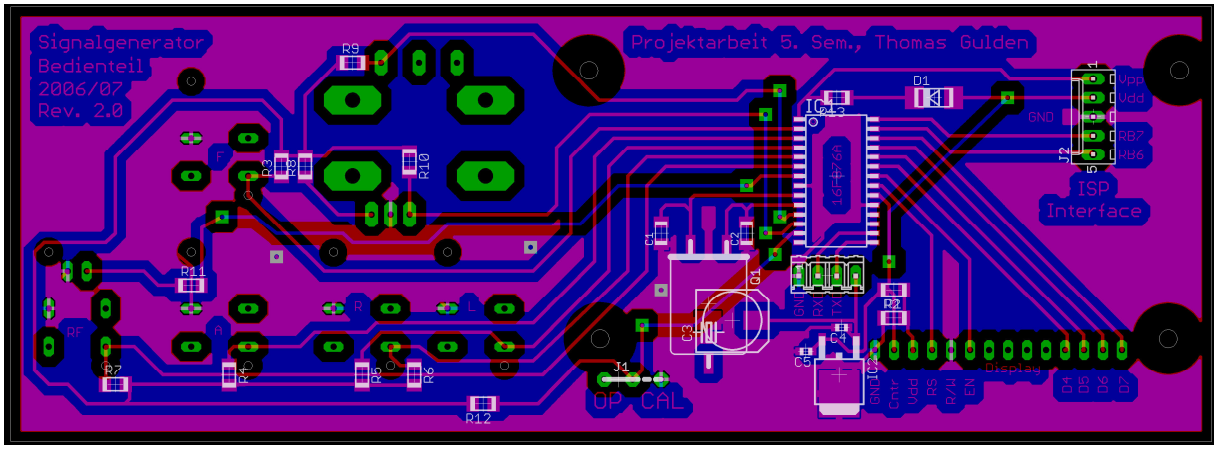


Abbildung 16 : Layout der Platine des Bedienteils

Schaltplan sowie Layout befinden sich in Originalgröße im Anhang dieser Dokumentation

6.2 Software

6.2.1 Programmablauf

Um die Eingabeelemente, welche der Interaktion des Benutzers mit dem Gerät dienen, ohne Verzögerungen einlesen zu können, gibt es zwei Möglichkeiten.

Zum Einen kann man durch einen Tastendruck oder eine Drehung am Encoder einen Interrupt auslösen, durch welchen das laufende Programm unterbrochen wird, um der Benutzereingabe zu folgen.
Dies bietet zwar den Vorteil einer unmittelbaren Reaktion, stößt bei der Verwendung des benutzten Microcontroller allerdings auf ein Problem:
Die PIC16 – Serie verfügt lediglich über einen einzigen externen Interrupt.
In der Praxis müsste man folglich die Taster und den Drehgeber „verODERn“ und dieses Signal dem Interrupteingang zuführen.
Würde nun ein interrupt ausgelöst, müsste per Software ermittelt werden, was die Quelle dessen ist.

Da die Anwendung eines externen ODER-Gatters, sei es ein TTL-Baustein oder diskret mit Dioden aufgebaut, eher unelegant erschien, wurde hier jedoch einer anderen Methode der Vorrang gegeben:
Anstatt das ganze Programm mitsamt den Ausgaberoutinen für das Display und die serielle Schnittstelle und der verschiedenen Berechnungen abzuarbeiten, werden nur bestimmte Programmabschnitte bei jedem Durchlauf ausgeführt.
Diese sind

- Einlesen der Tasten
- Einlesen des Drehgebers
- Überprüfen der „changes“ – Variable
  - ➔ Nur bei Bedarf neue Frequenz / Amplitude anzeigen und an HF-Teil senden

Wird bei einer dieser Operationen eine Änderung festgestellt, so wird zunächst die zur gewünschten Funktion gehörende Hilfsvariable (z.B. Frequenz 100Hz höher, RF off, …) verändert und anschließend eine „changes“ – Variable gesetzt, über welche das Programm erkennt, ob sich etwas geändert hat.
Ist der Zustand dieser Variable auf „1“, so werden beim nächsten Programmdurchlauf die Hilfsvariablen abgearbeitet und die neuen Daten werden ausgegeben.
Abschließend wird die „changes“ – Variable wieder auf „0“ zurückgesetzt und bis zur nächsten Änderung läuft lediglich das Einleseprogramm.

Der Vorteil besteht darin, dass es dem Controller nun möglich ist, sehr schnell auf Eingaben zu reagieren, da keine rechenintensiven Codeabschnitte im Leerlauf abgearbeitet werden müssen; als Beispiel sei hier die Display-Ausgabe genannt, welche ca. 4 ms benötigt, um eine Seite (2 x 16 Zeichen) an das Display zu schreiben, um dessen Timing-Spezifikation nicht zu verletzen. Während dieser Zeit würde auf eine Eingabe keine Reaktion erfolgen.
Zudem ist die Darstellung auf dem Display klarer, wenn die gleichen Daten lediglich einmalig hineingeschrieben werden, als wenn alle 5 ms eine Aktualisierung durchgeführt wird.

Es ergibt sich daraus folgender Programmablaufplan:

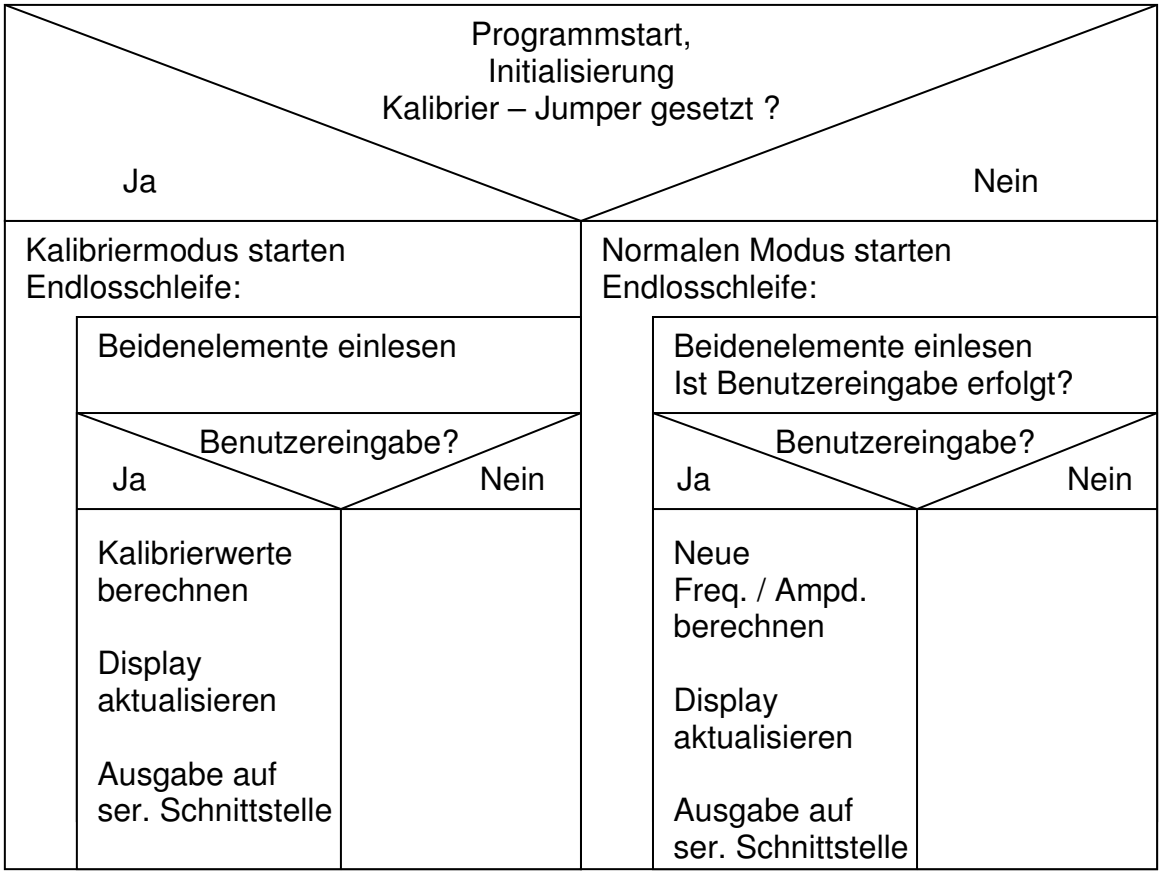


Abbildung 17 : Programmablauf der Bedienteil - Software

6.2.2 Ansteuerung des Displays

Zur Ansteuerung von Displays mit HD44780-kompatiblen Controllern ist dem CCS-Compiler zwar eine Routine beigelegt, leider unterstützte diese jedoch weder eine Pinbelegung anders als vorgeschlagen, noch verfügte diese über eine Möglichkeit, benutzerdefinierte Zeichen zu erstellen.
Also musste die bestehende Funktion so modifiziert werden, dass sie den Ansprüchen gerecht wurde.
Im Zuge dieser Modifikation wurde die Funktion aus Speicherplatzgründen (die abgespeicherten Texte einschließlich deren Ausgaberoutine belegen relativ viel Speicher im PIC) deutlich entschlackt:
Die Standard-Displayroutine besteht aus einem Teil, welcher auf das Display schreibt und einem weiteren Teil, mit welchem vom Display gelesen werden kann.
Dieser Programmabschnitt wurde, da unnötig, vollständig entfernt, was die Displaysteuerung auf etwa 40 % verkleinerte.

Durch die modifizierte Ausgaberoutine ist es möglich, auf das Display als auch auf die serielle Schnittstelle mit dem “printf“ – Befehl zu schreiben:

```
Printf (“Test“);          Ausgabe des Worts “Test“ auf der seriellen Schnittstelle
Printf (lcd_putc, “Test“); Ausgabe des Worts “Test“ auf dem Display
```

Durch Einfügen von “lcd_putc“, dem Namen der Schreibroutine des Displays, wird der Datenstrom zu diesem umgeleitet.

6.2.3 Einlesen der Taster-Zustände

Um zu erkennen, ob ein Taster gedrückt wurde, ist folgender Code entstanden:

```
void Tastenabfrage()
  {
   int1 Taster_status;

   Taster_status  = input(Taster);



    If (Taster_status == 0)          // Abfrage
        {

        // Auszuführender Code wenn Taster gedrückt wurde

         while ( (input(Taster) == 0)  )
                {
                        // Falls der Taster immer noch gedrückt wird,
                           warten, bis dieser solgelassen wurde
                }
```

Codeauszug 7 : Tasterabfrage

Die Taster sind low-aktiv, d.h. sie besitzen einen Pull-Up-Widerstand.
Die Eingänge werden bei jedem Programmdurchlauf eingelesen (polling).
Wird festgestellt, dass ein Eingang low-Potential führt (Taster_status = 0), wird der entsprechende Code ausgeführt; anschließend wird geprüft, ob der Taster immer noch gedrückt wird (while – Schleife) um zu verhindern, dass einmaliges Drücken eine mehrfache Ausführung des Codes zur Folge hat.
Nachteil dieser Methode ist jedoch, dass ein festgehaltener Taster den Programmablauf blockiert, was in dieser Anwendung jedoch nicht stört, da keine zeitkritischen Komponenten enthalten sind.
Abhilfe wäre möglich, indem man wichtige Aufgaben an einen Timer-Interrupt koppelt, sodass diese in festen Zeitabständen immer ausgeführt werden.

6.2.4 Einlesen des Drehgebers

Das Auswerten der Drehrichtung erfolgt, indem die aktuellen Zustände der Signalleitungen A und B in Variablen abgespeichert werden und kurz darauf mit den aktuellen Werten verglichen werden.

Unter der Bedingung, dass sich der Zustand lediglich eines Signals zwischen den zwei Abtastzeitpunkten geändert hat, kann nun anhand der Änderung die Drehrichtung ermittelt werden.
In der Praxis erfolgt das Einlesen der Signale so schnell, dass beim Test keine falsche Auswertung des Drehgebers provoziert werden konnte.

```
float RotEncCal_float (float Zahl)
  {
  static int1 Aalt, Balt;
  Aalt = input (A);    // Hilfsvariablen (Vergleichspegel)
  Balt = input (B);
  delay_ms(5);

  if ( (Aalt != input (A)) || (Balt != input(B) ) )   // es wurde gedreht
    {
    if ( ((Aalt == 0) && (input(A)==1)) && (Balt==0) && (input(B) == 0)  )
                              // Drehung im Uhrzeigersinn ➔ Inkrementierung
      {
      Zahl = Zahl + 0.1;     // „Zahl“ wird inkrementiert um 0,1
      Aalt = input (A);      // Jetzige Signalpegel als neue Vergleichspegel setzen
      Balt = input (B);
      }
    if ( ((Aalt == 1) && (input(A)==1)) && (Balt==0) && (input(B) == 1)  )
                              // Drehung im Uhrzeigersinn ➔ Inkrementierung
      {
      Zahl = Zahl + 0.1;     // „Zahl“ wird inkrementiert um 0,1
      Aalt = input (A);       // Jetzige Signalpegel als neue Vergleichspegel setzen
      Balt = input (B);
      }
    … … … …          // andere mögliche Zustandsänderungen der Signalpegel
    }
  }
```

Codeauszug 8 : Einlesen des Drehgebers

6.2.5 Auswahl der zu verändernden Stelle

Je nach Auswahl werden vom Drehgeber die 0.1er, 1er, 10er oder 100er - Stelle manipuliert.
Die Auswahl erfolgt nach folgendem Prinzip:

Zunächst wird über zwei Taster eine Stelle ausgewählt, wobei diese durch eine Indexzahl repräsentiert wird.

- -1 entspricht 0,1er - Stelle
- 1 entspricht 1er - Stelle
- 2 entspricht 10er - Stelle
- 3 entspricht 100er – Stelle

Der eine Taster dekrementiert den Index, während der andere diesen inkrementiert, somit kann man eine Grob- und Feineinstellung der Frequenz und Amplitude vornehmen.
Parallel dazu wird das Pfeilsymbol “^“ unter dem entsprechenden Zeichen im Display eingeblendet.

Wird der Drehgeber im Uhrzeigersinn (➔ Inkrementierung) betätigt und es ist die Frequenzeinstellung aktiv, so wird geprüft, welchen Wert der Stellenindex zur Zeit beträgt. Angenommen dieser zeigt auf die 10er-Stelle, so wird die Zahl 10 zur aktuellen Frequenz addiert. Anschließend wird überprüft, ob sich das Resultat im gültigen Wertebereich für Frequenzen (40 MHz ≤ f ≤ 500 MHz) befindet und, falls nötig, zurückgesetzt.

Die wesentlichen Funktionen hierfür sind

- Stelle_shift : Ändern der Indexzahl mittels der Auswahltaster (siehe Codeauszug 9)
- Stelle_DeExp : Indexzahl in Dezimalzahl umwandeln (siehe Codeauszug 9)
- Verarbeitung : Ändern der ausgewählten Stelle durch Addition / Subtraktion der aus Stelle_DeExp erhaltenen Dezimalzahl (siehe Codeauszug 10)

```
void Stelle_shift(int1 dir)   // 0 => down 1 => up
  {
  if (dir == 0)           // dec. Index
    {
    if (Stelle <= -1 ) Stelle = -1;
    if (Stelle == 1)   Stelle = -1;
    if (Stelle == 2)   Stelle = 1;
    if (Stelle == 3)   Stelle = 2;

    }
  else                    // inc. Index
    {
    if (F_A == 0)      // Frequenz hat auch 100er Stelle
      {
      if (Stelle >= 3)  Stelle = 3;
      if (Stelle == 2)  Stelle = 3;
      }
    else                  // Amplitude hat nur 10er Stelle ==> 100er Stelle sperren
      {
      if (Stelle >= 2)  Stelle = 2;
      }
    if (Stelle == 1)  Stelle = 2;
    if (Stelle == -1) Stelle = 1;


    }
  }


float Stelle_DeExp()
                          // Wandelt die Indexzahl in eine Dezimalzahl um;
  {
  switch (Stelle)
    {
    case -1  :  return 0.1;
              break;
    case  1  :  return 1.0;
              break;
    case  2  :  return 10.0;
              break;
    case  3  :  return 100.0;
              break;
    default  :  break;
    }
  }
```

Codeauszug 9 : Auswahl der zu verändernden Stelle

```
void Verarbeitung (int1 dec_inc)    // dec_inc=0 => decr; =1 => incr
  {
  changes = 1;

  if (dec_inc == 0)       // DEKREMENTIEREN
    {
    if (F_A == 0)              // F_A = 0 ➔ Frequenz dekrementieren
      {
      Frequenz  = Frequenz  - Stelle_DeExp();
      }
    else                       // F_A = 1 ➔ Amplitude inkrementieren
      {
      Amplitude = Amplitude - Stelle_DeExp();
      }
    }
  else                // INKREMENTIEREN
    {
    if (F_A == 0)              // F_A = 0 ➔ Frequenz inkrementieren
      {
      Frequenz  = Frequenz  + Stelle_DeExp();
      }
    else                       // F_A = 1 ➔ Amplitude inkrementieren
      {
      Amplitude = Amplitude + Stelle_DeExp();
      }
    }
                               // Setzen von 40 MHz als untere und 500 MHz als
                                  obere Grenze:
  if (Frequenz >= 500.0)
    Frequenz = 500.0;
  if (Frequenz <= 40)
    Frequenz = 40.0;
  if (Amplitude >= 15.0)
    Amplitude = 15.0;
  }
```

Codeauszug 10 : Bearbeiten der ausgewählten Stelle, Bereichsgrenzen - Überprüfung

6.2.6 Steuerung der Kalibrierung

Die Kalibrierung erfolgt, indem der Jumper auf der Platinenrückseite vor dem Einschalten gesetzt wird.
Im Kalibriermenü kann nun gewählt werden, was kalibriert werden soll:

- “1: Pwr vs. Freq.“ Amplitudenverlauf ohne Dämpfung über der Frequenz in 5 MHz-Schritten mit einem Power-Meter messen und eingeben
- “2: 15 dB Att.“ Ausgangsleistung mit dem Power-Meter messen und eingeben; es ist nur das 15dB Dämpfungsglied zugeschaltet. Die Dämpfungsbestimmung erfolgt in Bandmitte bei 280 MHz
- “2: 25 dB Att.“ Ausgangsleistung mit dem Power-Meter messen und eingeben; es ist nur das 25dB Dämpfungsglied zugeschaltet. Die Dämpfungsbestimmung erfolgt in Bandmitte bei 280 MHz
- “4: PIN Att.“ Optional; hier kann eine Funktion aufgerufen werden, über welche Streuungen zwischen den PIN-Abschwächern der einzelnen Signalgeneratoren ausgeglichen werden können. Da es Ziel dieser Arbeit war, einen funktionsfähigen Prototyp aufzubauen, wurde diese Option zwar vorgesehen aber nicht implementiert

Die eingegebenen Daten werden an den HF-Teil gesendet und dort weiter verarbeitet.
Die „Intelligenz“ steckt somit fast ausschließlich im HF-Teil, wodurch auch andere Steuergeräte als dieses Bedienteil einfach realisierbar sind.

7 Molex – Anschlussplatine

Um die Baugruppen HF-Teil, Bedienteil und Netzteil miteinander zu verbinden, entstand eine kleine Adapterplatine, auf welcher drei Molex Steckverbinder aufgelötet sind.
Zusätzlich kann ein weiterer Molex-Verbinder montiert werden, über welchen die seriellen Datenleitungen Bedienteil ➔ HF – Teil und HF-Teil ➔ Bedienteil abgegriffen werden können.

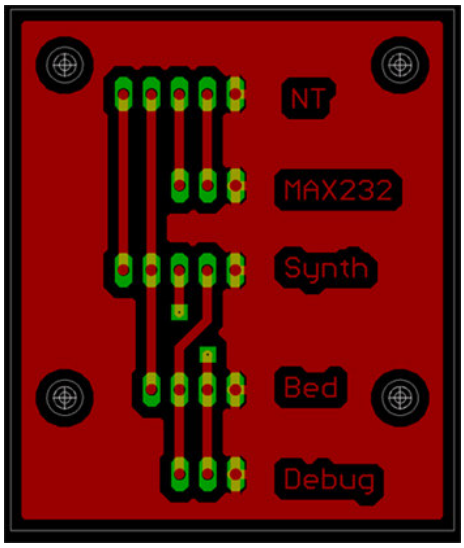


Abbildung 18 : Anschlussplatine

Somit können die einzelnen Baugruppen ohne Lötkolben gewechselt werden.

8 Optimierungsansätze

8.1 Durchgeführte Optimierungen

8.1.1 Separate Spannungsversorgung

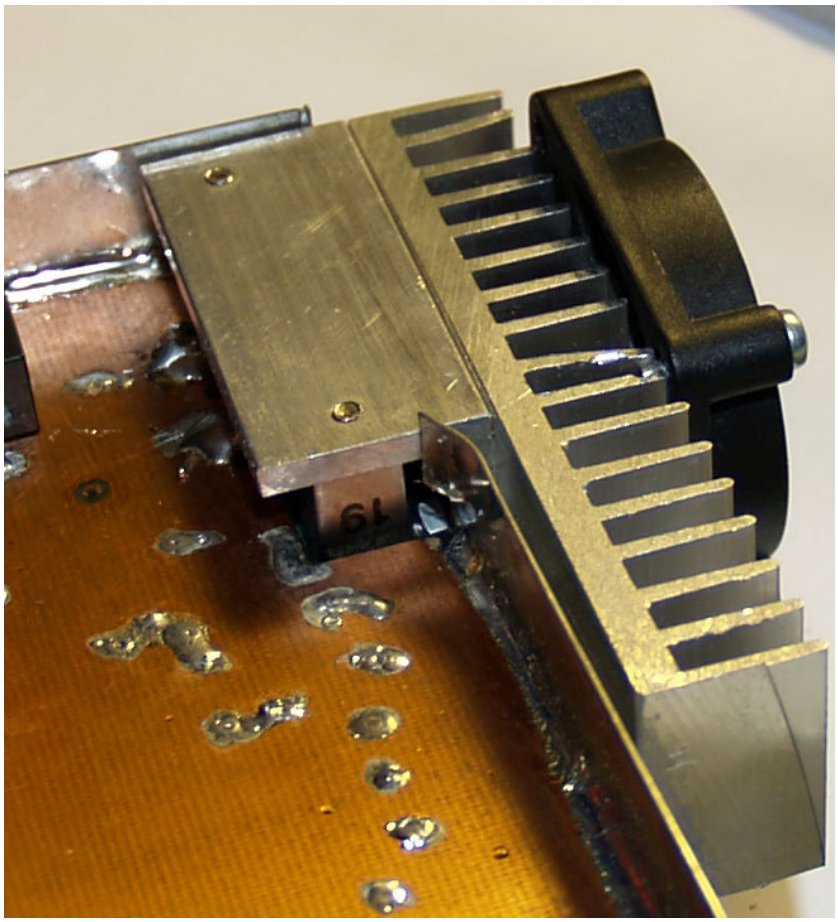
Im Originalzustand sind der HF-Teil und die Spannungsversorgung zwei getrennte Baugruppen, welche über ein Flachbandkabel miteinander verbunden sind.
Auf der Spannungsversorgungsplatine sorgt ein 24V – Längsregler (7824) für die Bereitstellung der stabilisierten Versorgungsspannung des Endstufenmoduls.
Die restliche Schaltung benötigt eine Versorgungsspannung von 5 Volt.
Um diese bereitzustellen, wurde ein Schaltregel-IC LM2574N-5 eingebaut, welches eine Ausgangsspannung von 5V bei maximal 0,5 A liefern kann.

Bei den Tests fiel jedoch auf, dass beim Zuschalten der Festwert-Dämpfungsglieder die 5V-Spannung kurzzeitig einbrach, wodurch die Oszillatoren in der Frequenz schwankten und die Ausgangsleistung einbrach, da die Versorgungsspannung als Referenz für den PWM-Ausgang des Microcontrollers dient.

Zur Problemlösung wurde die Spannungsversorgung folgendermaßen umstrukturiert:
Auf der Netzteilplatine wurde der 5V Schaltregler durch einen 12V Längsregler (7812) ersetzt.
Die Stabilisierung auf 5V erfolgt im HF-Teil durch zwei separate Spannungsregler (TS2940); ein Regler für die Relais und ein weiterer für die Versorgung des Microcontrollers, der Oszillatoren und der Treiberverstärker.
Dadurch konnte die Rückwirkung durch das Zuschalten der Relais auf ein Minimum reduziert werden.

8.1.2 Verlustleistung des Endstufenmoduls

Ein weiteres Problem war die Verlustleistung des Endstufenmoduls.
Dieses hat bei einer Versorgungsspannung von 24 Volt eine Stromaufnahme von etwa 300 mA was eine Eingangsleistung >7 Watt ergibt.
Die maximale Ausgangsleistung des Signalgenerators liegt bei höchstens +20 dBm, also 100 mW.
Folglich werden etwa 7 Watt in Wärme umgesetzt.
Ursprünglich war das Endstufenmodul lediglich auf eine 5 mm starke Aluminiumschiene aufgeflanscht, welches eine Fläche von etwa 15 cm x 3 cm aufwies und sich innerhalb des Weißblechgehäuses des HF-Teils befand.
Bereits nach kurzer Betriebszeit erreichte diese Schiene selbst bei offenem Gehäuse eine Temperatur von etwa 70 Grad Celsius und die Ausgangsleistung ging zurück.
Um längere Betriebszeiten zu ermöglichen, war die Montage des Hybridmoduls auf einem nach außen geführten Kühlkörper unumgänglich.
Zusätzlich mit einem Lüfter versehen bleibt die Temperatur des Moduls unter 40 Grad Celsius.



8.1.3 „In System Programming“ – Schnittstelle

„In System Programming“, kurz ISP, ist die Bezeichnung für eine Schnittstelle, die es ermöglicht, den Microcontroller zu programmieren, ohne diesen aus dem Zielsystem zu entfernen.
Dazu müssen lediglich die Programmierleitungen auf eine Steckerleiste geführt werden, an die ein Programmieradapter angeschlossen werden kann.
Dies bietet den Vorteil, dass das IC nicht ein- und ausgesteckt werden muss, was sich äußerst nachteilig auf dessen Lebensdauer auswirkt, da hierbei leicht die Anschlusspins abbrechen. Außerdem bietet ISP die Möglichkeit, Programmänderungen zügig zu erproben, da nichts umgesteckt werden muss.

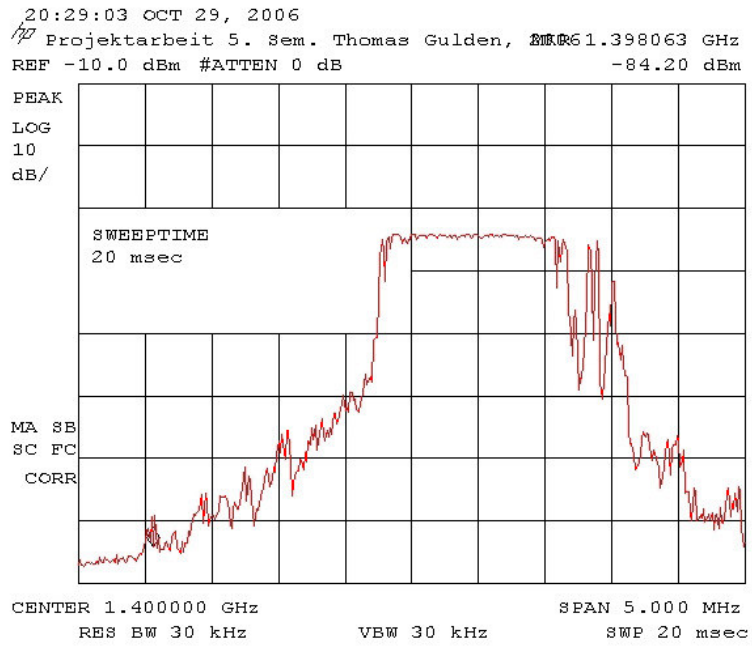
Der Prototyp verfügte leider über keine ISP-Schnittstelle, weshalb diese nachgerüstet wurde, um die Entwicklung zu vereinfachen.

8.2 Zukünftige Optimierungsansätze

Die hier aufgeführten Möglichkeiten wurden im Zuge dieser Projektarbeit zwar in Betracht gezogen, aber nicht realisiert, da sie nicht Thema der Projektarbeit waren.

8.2.1 Loop – Filter

Um mit dem Signalgenerator arbeiten zu können sollte eine Optimierung des Loop Filters im Mittelpunkt weitergehender Optimierungen stehen.



Das Ausgangssignal besitzt momentan eine spektrale Breite von rund 1,5 MHz.

Zudem rastet der variable Oszillator im oberen Abstimmbereich nicht.

Abbildung 19 : Ausgangsspektrum des variablen Oszillators vor dem Heruntermischen

8.2.2 Streuung der PIN-Dämpfglieder

Um die Firmware des HF-Teils auf andere Geräte übertragen zu können, muss in der aktuellen Version eine Messkurve der Dämpfung über der PWM-Spannung aufgenommen werden und eine Polynomannäherung durchgeführt werden, welche in den Quelltext übertragen werden muss.

Denkbar wäre hier, eine solche Kurve an mehreren HF-Teilen aufzunehmen und zu vergleichen, ob durch einfache Offset – Bildung beim PWM-Steuerwort eine Kalibrierung durchführbar wäre, da der Verlauf der Dämpfungskurve bei allen Geräten gleich sein müsste.

9 Schlussbetrachtung

Trotz erheblicher Mängel der Ausgangshardware konnte das Gerät aus meiner Sicht im Rahmen meiner Projektarbeit betriebsfähig gemacht werden und die Software nebst Bedienteil gemäß den geforderten Gesichtspunkten entwickelt werden.