

Gymnasium Alexandrinum Coburg

Kollegstufenjahrgang 2002/2003

Facharbeit im Fach Physik

Bau eines digitalen Funktionsgenerators

Verfasser: Andreas Schwarz
Leistungskurs: Physik
Kursleiter: Ottmar Lemke
Abgabetermin: 3. Februar 2003

Inhaltsverzeichnis

1. Einleitung	3
2. Analoge Funktionsgeneratoren	4
2.1. Aufbau	4
2.1.1. Rechteck- und Dreieckschwingungen	4
2.1.2. Sinusschwingungen	5
2.2. Nachteile	5
3. Digitale Funktionsgeneratoren	6
3.1. Aufbau	6
3.1.1. Funktionsberechnung	6
3.1.2. Digital-Analog-Wandler	8
3.1.3. Filterung	10
4. Bau eines digitalen Funktionsgenerators	14
4.1. Spannungsversorgung	14
4.2. Prozessor	15
4.3. Digital-Analog-Wandler	15
4.4. Amplituden- und Offseteinstellung und Filterung	15
4.5. Ergebnis	17
A. Literatur	18
B. Schaltplan	20
C. Programm	23

1. Einleitung

Ein Funktionsgenerator ist ein elektronisches Gerät, das verschiedene Signalformen mit einstellbarer Amplitude und Frequenz erzeugen kann.

Die einstellbaren Signalformen sind meistens Sinus, Rechteck, Dreieck und Sägezahn (auch Rampe genannt). Seltener vorhanden sind Pulsweitenmodulation¹ oder Exponentialfunktionen.

Funktionsgeneratoren werden verwendet, um das Verhalten von Schaltungen zu testen. Ein typischer Einsatzzweck ist die Untersuchung von Verstärkern. Dabei schließt man den Signalausgang des Funktionsgenerators an den Eingang des Verstärkers an und vergleicht das Oszilloskopbild des Signals mit dem Signal, das am Ausgang des Verstärkers anliegt. Auf diese Weise lassen sich Daten wie Verstärkung, Übersteuerungsverhalten, Frequenzgang und Verzerrung untersuchen.

In dieser Facharbeit werde ich zunächst den allgemeinen Aufbau von analogen und digitalen Funktionsgeneratoren beschreiben und anschließend ein funktionierendes „Modell“ eines digitalen Funktionsgenerators vorstellen.

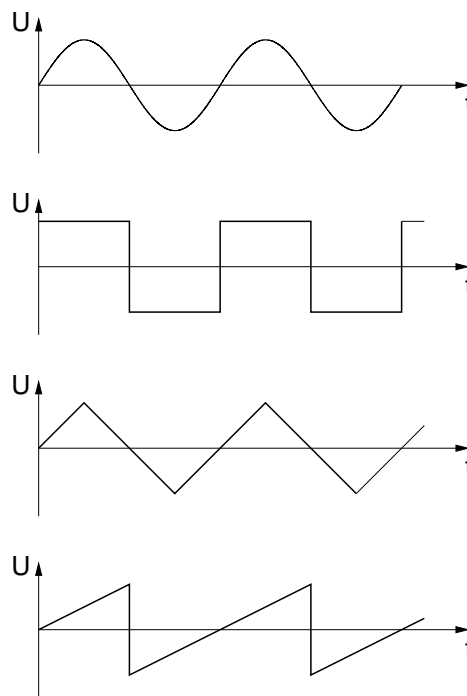


Abbildung 1: Typische Signalformen eines Funktionsgenerators: Sinus, Rechteck, Dreieck, Sägezahn

¹wie Rechteck, aber mit einstellbarem An/Aus-Verhältnis

2. Analoge Funktionsgeneratoren

2.1. Aufbau

In analogen Funktionsgeneratoren werden die Signale erzeugt, indem Schaltungen mit Spulen oder Kondensatoren zum Schwingen gebracht werden. Um aus dieser Schwingung verschiedene Signalformen zu erzeugen, wird meist der folgende Aufbau verwendet:

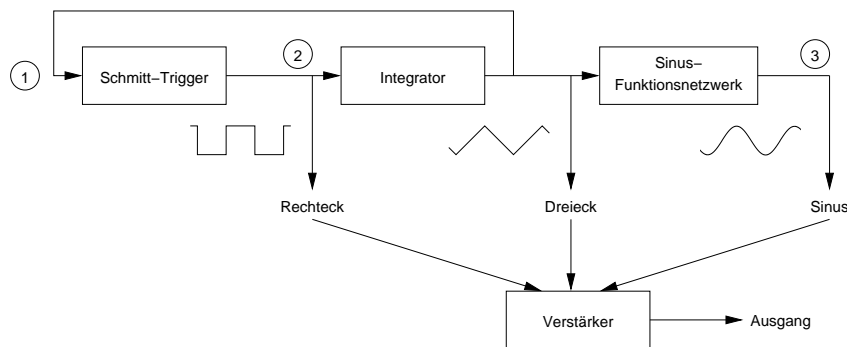


Abbildung 2: Schematischer Aufbau eines analogen Funktionsgenerators

2.1.1. Rechteck- und Dreieckschwingungen

Die Spannung an Punkt 2 wird von dem Integrator² integriert, wobei eine positive Spannung zu einer fallenden Spannungsrampe am Ausgang führt, eine negative Spannung zu einer steigenden. Diese wird gleichzeitig an den Eingang des *Schmitt-Triggers* zurückgeführt. Sobald die Eingangsspannung des Schmitt-Triggers eine der Umschaltsschwellen $-\frac{2}{3}U_{max}$ oder $+\frac{2}{3}U_{max}$ erreicht hat, springt die Ausgangsspannung an Punkt 2 auf $-U_{max}$ bzw. $+U_{max}$. Dadurch wird die Integrationsrichtung umgekehrt, die Steigung der Spannungsrampe ändert also ihr Vorzeichen. Die Spannung steigt oder sinkt nun wieder so lange, bis die andere Umschaltsschwelle des Schmitt-Triggers erreicht ist und ändert dann erneut die Richtung. Dieser Vorgang wiederholt sich immer wieder, so dass an Punkt 2 eine Rechteck- und an Punkt 3 eine Dreieck-Schwingung gemessen werden kann. Die Frequenz hängt von der Geschwindigkeit der Integration, also der Steigung der Dreiecksspannung, ab.

²Die Integrationschaltung besteht im wesentlichen aus einem Operationsverstärker, der einen Kondensator mit einer konstanten, von der Eingangsspannung abhängigen Geschwindigkeit lädt.

2.1.2. Sinusschwingungen

Um aus dem erzeugten Dreiecksignal eine angenäherte Sinusschwingung zu gewinnen, wird ein *Sinusfunktionsnetzwerk* verwendet. Damit wird die Eingangsspannung mit Hilfe von Dioden in mehrere Abschnitte „zerlegt“, die jeweils unterschiedliche, durch Spannungsteiler angepasste Steigungen besitzen, wobei die Signalform so verändert wird dass sie einem Sinus ähnelt. Eine ausführliche Erklärung dieses Verfahrens findet sich in [1, S. 213ff].

2.2. Nachteile

Diese Art der Signalerzeugung hat jedoch einige Nachteile. Da alleine der Integrator für die Frequenz des Ausgangssignals verantwortlich ist, ist dessen Aufbau relativ kritisch wenn man eine genaue und stabile Frequenz erhalten möchte. Besonders bei hohen Frequenzen über 100 kHz stößt diese Methode an ihre Grenzen.

Ein weiteres Problem ist, dass eine genaue Frequenzeinstellung oder -anzeige schwierig zu realisieren ist. Fertigungstoleranzen von 5-20% sind bei Kondensatoren keine Seltenheit, deshalb kann man die Ausgangsfrequenz nicht einfach aus den in der Integrationschaltung verwendeten Bauteilwerten berechnen und als Skala am Einstellknopf antragen, sondern müsste die Frequenz in Abhängigkeit von der Einstellung messen und die Abweichung mit einem Potentiometer kompensieren. Aber da die Bauteile durch Alterung und Temperaturunterschiede ihre Werte verändern, wäre auch dann eine regelmäßige Nachkalibrierung notwendig.

3. Digitale Funktionsgeneratoren

3.1. Aufbau

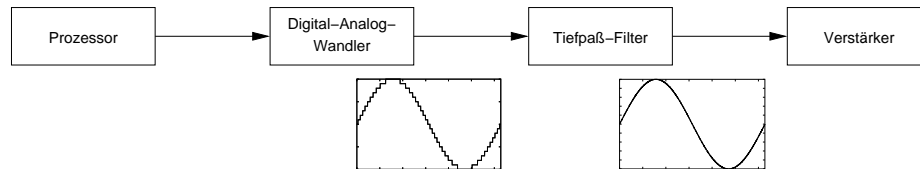


Abbildung 3: Schematischer Aufbau eines digitalen Funktionsgenerators

Der grundlegende Unterschied zur analogen Signalerzeugung besteht darin, dass die verschiedenen Signale nicht direkt durch „echte“ Schwingungen erzeugt werden. Stattdessen übernimmt ein Prozessor die Berechnung der Funktionswerte und wandelt sie mit einem *Digital-Analog-Wandler* in die entsprechende Ausgangsspannung um. Der Prozessor läuft dabei mit einer festen Taktfrequenz, unterschiedliche Ausgangsfrequenzen des Funktionsgenerators werden durch die Software realisiert.

Dieser Ansatz hat einige deutliche Vorteile. Die feste Frequenz, die zum Betrieb des Prozessors benötigt wird, lässt sich viel stabiler und genauer herstellen, als es mit der Integrator/Schmitt-Trigger-Kombination möglich wäre. Mit einem handelsüblichen integrierten Quarzoszillator erreicht man mit sehr geringem Aufwand und ohne Kalibrierung eine Genauigkeit von 100ppm³. Außerdem ist die Anzeige der Frequenz kein Problem, da der Prozessor den exakten Wert „kennt“, und ihn einfach auf einer angeschlossenen LED- oder LCD-Anzeige darstellen kann.

3.1.1. Funktionsberechnung

Bei einem digitalen Sinusgenerator werden die Funktionswerte nie unmittelbar berechnet, sondern sie werden schon beim Entwurf des Gerätes vorausberechnet und fest in den Prozessor eingespeichert, da die Berechnung einer Sinusfunktion zu viel Rechenzeit in Anspruch nehmen würde um in Echtzeit durchgeführt zu werden.

Die Verarbeitung dieser gespeicherten Funktionswerte wird von einem Programm übernommen, das in einem Prozessor ausgeführt wird. Der zentrale Teil dieses Berechnungsprogramms ist der *Phasenakkumulator*, eine Schleife, die mit einer festen Frequenz wiederholt wird. Mit jedem Durchlauf dieser Schleife wird zu einer Variablen c ein zur gewünschten Ausgangsfrequenz des Funktionsgenerators proportionaler Wert a addiert. Die Variable c stellt die Phasenposition in der Schwingung dar.

Eine anschauliche Darstellung dieses Funktionsprinzips zeigt Abbildung 4:

³ „parts per million“, 1ppm = 0,0001%

Die schwarzen Punkte sind die Werte, die die Variable c annehmen kann, die grauen Kreissektoren stellen die abgespeicherten Funktionswerte für die angegebenen Phasenwinkel dar. Von einer Schwingungsperiode der Sinusfunktion sind in diesem Beispiel 16 Werte vorhanden; jeder dieser Werte wird von einem der grauen Kreissektoren repräsentiert.

Das Programm erhöht nun die Variable c bei jedem Schleifendurchgang mit einer bestimmten Schrittweite a , liest danach den im danebenliegenden Kreissektor „gespeicherten“ Funktionswert aus, und erzeugt eine entsprechende Spannung am Ausgang.

Wenn die in Abbildung 4 als „Zeiger“ dargestellte Position der Variablen c den Kreis einmal umrundet hat, ist eine Schwingungsperiode beendet.

Die Umrundung des Kreises geht um so schneller, je größer a ist. Verdoppelt man a , dann wird die Umlaufdauer (Dauer einer Periode) halbiert und die Frequenz verdoppelt. a ist also direkt proportional zur Frequenz.

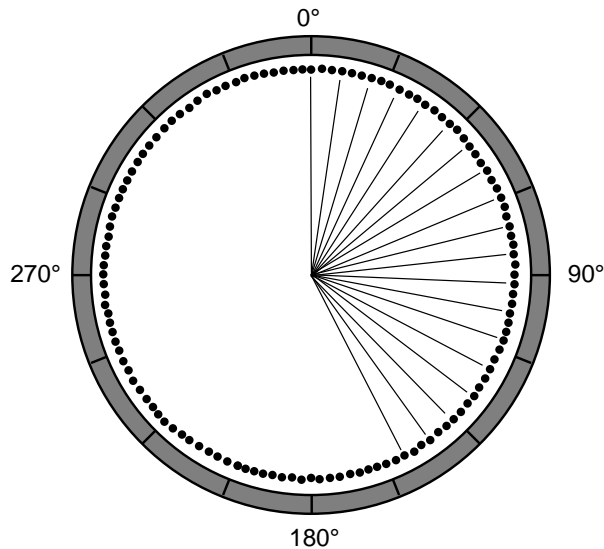


Abbildung 4: Prinzip des Phasenakkumulators (vgl. [4])

Abbildung 5 stellt ein paar Ergebnisse dar, wie sie durch dieses Verfahren erzielt werden. Ausgegangen wurde dabei von Abbildung 4. Als Schrittweite a des Zeigers wurden die Werte 3, 4 und 12 eingesetzt, für die Schrittweite 3 ist in Abbildung 4 außerdem die um den Kreis laufende Variable c als „Uhrzeiger“ dargestellt.

Wie man leicht erkennen kann, verhält sich die resultierende Frequenz proportional zur Schrittweite. Allerdings wird auch deutlich, dass die Signalform bei hohen Werten von a (hier z.B. 12), also bei hohen Frequenzen, ungleichmäßig und unsauber wird, und dass nicht jede Schwingungsperiode exakt die gleiche Form hat. Der

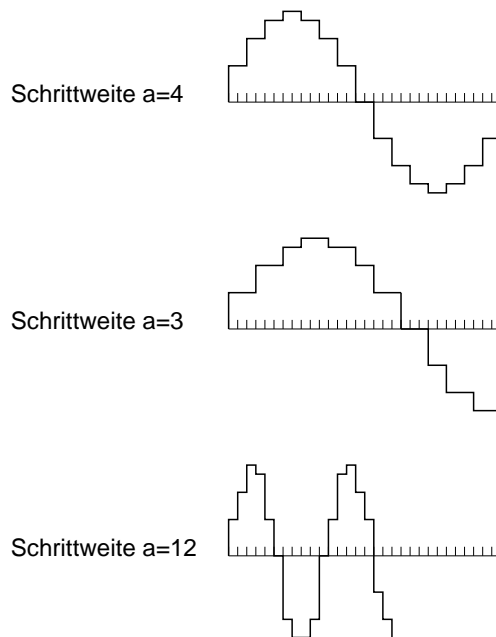


Abbildung 5: Beispielergebnisse einer DDS-Synthese

Grund dafür ist, dass durch die große Schrittweite manche Speicherstellen ausgelassen werden müssen. Deshalb ist man in der Praxis bestrebt, nicht nur eine möglichst große Anzahl von Funktionswerten zu speichern, sondern auch die Wiederholungsrate des Additionsvorgangs möglichst hoch zu wählen, damit es bei hohen Frequenzen nicht zu Störungen kommt.

Diese Art der Signalerzeugung mittels eines Phasenakkumulators nennt man *Direkte Digitale Synthese* (DDS). Meistens wird sie zur Erzeugung von Sinus-Schwingungen verwendet, aber es ist genauso möglich statt den Sinus-Funktionswerten die Werte einer beliebigen anderen Funktion einzusetzen.

In der Praxis trifft die Darstellung des Berechnungsvorgangs als ein „Programm“ nicht immer zu. Häufig ist die Funktionalität des beschriebenen Programms in einem integrierten Schaltkreis „fest verdrahtet“; der Additionsvorgang wird dann nicht durch Programmbefehle durchgeführt, sondern direkt durch eine digitale Schaltung. Dadurch sind weitaus höhere Frequenzen bis in den GHz-Bereich hinein möglich.

3.1.2. Digital-Analog-Wandler

Der Prozessor, der die Funktionswerte berechnet (bzw. aus dem Speicher liest), kann die Werte nicht direkt in eine Spannung umwandeln, da die digitalen Ausgän-

ge nur zwei verschiedene Zustände⁴ annehmen können. Um eine feinere Abstufung zu erhalten, müssen deshalb mehrere dieser Ausgänge kombiniert werden. Die Zusammenschaltung erfolgt über ein Widerstandsnetzwerk (Abbildung 6).

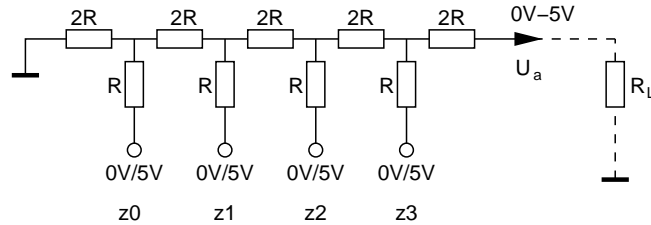


Abbildung 6: Digital-Analog-Wandler mit 4 Bit Auflösung

Nach [1, S. 636f] errechnet sich die Ausgangsspannung wie folgt:

$$U_a = \frac{U_{ref}}{16} \cdot \frac{R_L}{R + R_L} \cdot Z$$

R_L ist der Lastwiderstand am Ausgang des DA-Wandlers, also der Eingangswiderstand des am Ausgang angeschlossenen Gerätes. Dieser sollte im Vergleich zu R möglichst hoch sein, da sonst die Spannung durch die hohe Belastung des Widerstandsnetzwerks reduziert wird.

Z erhält man, indem man die Zustände (1/0) der digitalen Eingänge als die einzelnen Bits einer Binärzahl betrachtet. Der linke Eingang stellt dabei das niederwertigste Bit dar, der rechte das höchstwertigste. Wenn als Beispiel die beiden linken Eingänge aktiv, also auf 5 V, und die beiden rechten inaktiv, also auf 0 V sind, dann ist Z die Dualzahl 0011, was 3 im dezimalen System entspricht. Nimmt man den Idealfall $R_L = \infty$ an, so erhält man, wenn man für U_{ref} die Spannung 5 V einsetzt, eine Ausgangsspannung von 0,9375 V.

Je mehr Prozessorausgänge zusammenschaltet werden, um so feiner ist die Abstufung der Spannung. Der in Abbildung 6 dargestellten Wandler besitzt eine Auflösung von 4 Bit und kann durch die Zusammenschaltung von 4 Digitalausgängen 2^4 verschiedene Spannungswerte zwischen 0 V und 5 V erzeugen. Die Auflösung des DA-Wandlers kann durch die Verwendung von mehr Digitalausgängen theoretisch beliebig erhöht werden. Um eine feine Abstufung nutzen zu können, müssen die Funktionswerte allerdings auch in der entsprechenden Genauigkeit verfügbar sein. Oft verwendete Längen für DA-Wandler sind 8 und 16 Bit, da die Funktionswerte dann genau ein oder zwei *Byte* lang sind, was den üblichen Speichereinheiten eines Prozessors entspricht.

⁴in der Regel 0 V und 5 V

3.1.3. Filterung

Abbildung 7 zeigt beispielhaft ein ungefiltertes Sinus-Signal, das aus einem DA-Wandler kommt. Wie man sofort sieht, hat das Signal im Gegensatz zu dem erwünschten reinen Sinus eine treppenähnliche Form. Der Grund dafür ist die begrenzte Auflösung des Wandlers. Der hier verwendete 6 Bit-Wandler kann nur $2^6 = 64$ unterschiedliche Spannungswerte erzeugen, so dass zwischen diesen Werten Sprünge im Ausgangssignal entstehen. Um die unerwünschten Störungen zu entfernen, muss man zuerst einmal betrachten, was diese „Treppenstufen“ überhaupt ausmacht.

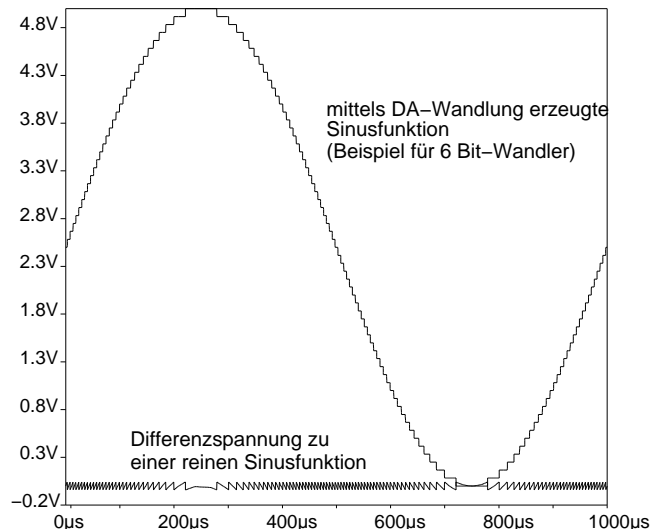


Abbildung 7: Ungefiltertes Sinus-Signal aus einem 6 Bit-DA-Wandler

Jedes periodische Signal setzt sich aus unendlich vielen Sinusschwingungen unterschiedlicher Frequenz und Amplitude zusammen (darstellbar durch *Fourier-Analyse*, vgl. [3, S. 189ff]). Bei dem abgebildeten „treppenförmigen“ Sinus mit 1000 Hz besteht dieses Frequenzspektrum neben der Grundfrequenz aus vielen weiteren höheren Frequenzen. Um die reine Grundschwingung zu erhalten, müssen diese störenden Oberschwingungen deshalb mit einem *Tiefpassfilter* herausgefiltert werden.

Wie der Name schon andeutet, filtert ein Tiefpassfilter die hohen Frequenzanteile eines Signals heraus und lässt die tiefen Frequenzen passieren. Die Kurve des Frequenzgangs fällt dabei natürlich nicht schlagartig mit einer bestimmten Frequenz auf 0, sondern sinkt mit einem weichen Knick ab (siehe Abbildung 8).

Die Frequenz, ab der das Signal um 3 dB⁵ gedämpft wird, nennt man die *Grenzfrequenz* des Filters.

⁵ $1 \text{ dB} = 20 \cdot \log_{10} \frac{A_2}{A_1}$; eine Dämpfung um 3 dB bedeutet also, dass die Amplitude A_2 um 29% kleiner ist als die Amplitude A_1 (vgl. [2, S. 33]).

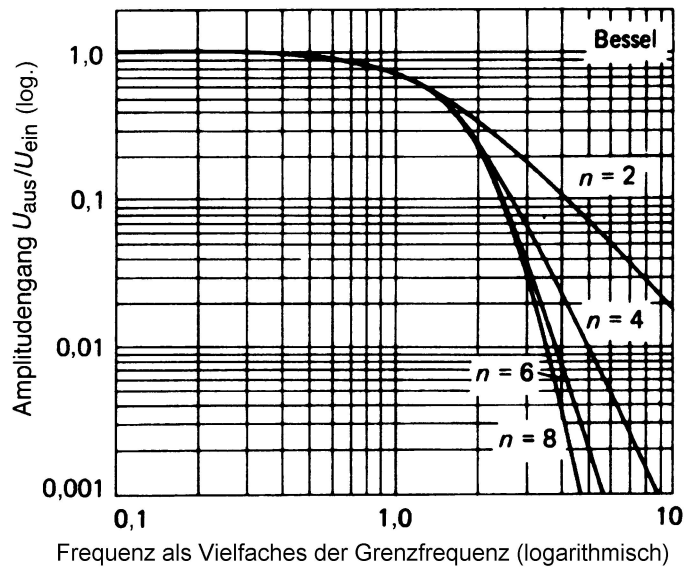


Abbildung 8: Frequenzgang eines Bessel-Filters [2, S. 315]

Aufgebaut wird so ein Filter meistens aus einem Operationsverstärker, Widerständen und Kondensatoren. Im Prinzip macht man sich dabei die Tatsache zunutze, dass der Wechselstromwiderstand eines Kondensators mit steigender Frequenz sinkt, wodurch es möglich ist die hohen Frequenzanteile nach Masse „kurzzuschließen“. Um die genauen Bauteilwerte für die erforderlichen Kondensatoren und Widerstände zu berechnen, benötigt man drei Angaben: den *Typ* des Filters, die *Grenzfrequenz*, und die *Ordnung*.

Der *Filtertyp* gibt an, nach welchen Gesichtspunkten der Frequenzgang optimiert ist (vgl. [1, S. 267ff]):

- *Butterworth*: möglichst lang horizontal verlaufender Frequenzgang, spätes Abknicken
- *Tschebyscheff*: steilerer Abfall, aber stärkere Welligkeit
- *Bessel*: möglichst frequenzunabhängige Signallaufzeit

Wie wir später noch sehen werden, ist ein Bessel-Filter für die Glättung des DA-Wandler-Signals am besten geeignet.

Eine höhere *Ordnung* bedeutet praktisch, dass man mehrere Filterstufen hintereinander schaltet, um so einen steileren Abfall des Frequenzgangs bei der Grenzfrequenz zu erzielen⁶, was natürlich auch auf einen höheren Schaltungsaufwand

⁶siehe Abbildung 8; die Ordnung wird hier durch den Parameter n angegeben

hinausläuft.

Wenn man die ebenfalls in Abbildung 7 gezeigte Differenzspannung zu einem reinen Sinus betrachtet, kann man grob abschätzen, dass der Hauptteil dieser Störschwingungen im Bereich ab 20 kHz liegt. Ein Tiefpass-Filter mit einer fest eingestellten Grenzfrequenz von z.B. 10 kHz könnte also diese durch die DA-Wandlung entstandenen Störungen größtenteils beseitigen und würde eine am Funktionsgenerator eingestellte Ausgangsfrequenz von über 5 kHz trotzdem relativ ungedämpft durchlassen.

Anders sieht es dagegen aus, wenn der Funktionsgenerator statt eines Sinus-Signals z.B. ein dreieck- oder rechteckförmiges Signal erzeugen soll. Da dieses bereits höherfrequente Schwingungen als die Grundfrequenz enthält, werden bei zu starker Filterung die Ecken „abgerundet“ und die Signalform verfälscht.

Zur Verdeutlichung der in einer Dreieckschwingung enthaltenen Frequenzen kann man die Funktion durch Kombination unendlich vieler Sinus-Funktionen annähern (*Fourier-Reihe*, vgl. [3, S. 192]):

$$f(t) = \frac{8A}{\pi^2} \left[\cos(\omega t) + \frac{1}{3^2} \cos(3\omega t) + \frac{1}{5^2} \cos(5\omega t) + \dots \right]$$

Um ein einigermaßen genaues Dreieck zu erhalten, benötigt man, wie anhand von Abbildung 9 nachvollziehbar, mindestens die fünffache Frequenz der Grundschwingung. Wenn man also wie oben einen Tiefpassfilter mit einer Grenzfrequenz von 10 kHz einsetzen würde, könnte der Funktionsgenerator nur Dreieckssignale bis ungefähr 2 kHz in vernünftiger Qualität erzeugen. Bei höheren Frequenzen würden die oberen Sinus-Komponenten des Signals so stark gedämpft, dass die Dreieckspitzen abgeflacht würden und die Signalform sich immer weiter einem einfachen Sinus annähern würde. Diesem Problem begegnet man, indem man die Abstufungen des DA-Wandlers möglichst fein wählt, so dass die Frequenz der Störschwingungen immer um ein Vielfaches höher ist als die des Nutzsignals.

In diesem Zusammenhang tritt auch ein anderes Problem der Filterung zutage: das Ausgangssignal ist im Vergleich zum Eingangssignal immer phasenverschoben, weil es beim Durchgang durch den Filter um eine gewisse Zeit verzögert wird. Das Problem daran ist, dass diese *Laufzeit* von der Frequenz abhängig ist. Bei einer Sinusfunktion ist das noch kein Problem, da die Phasenlage des Ausgangssignal ja keine Rolle spielt. Bei komplexeren Signalen (Rechteck, Dreieck, ...), die mehrere Frequenzanteile enthalten, führt jedoch eine ungleichmäßige Verzögerung der verschiedenen Bestandteile dazu, dass das Signal verzerrt wird und seine ursprüngliche Form verliert.

Verhindern kann man das, indem man einen Filtertyp verwendet, bei dem die Phasenverschiebung möglichst proportional zur Frequenz und damit die Laufzeit konstant ist: einen *Bessel-Filter* (vgl. [1, S. 277]).

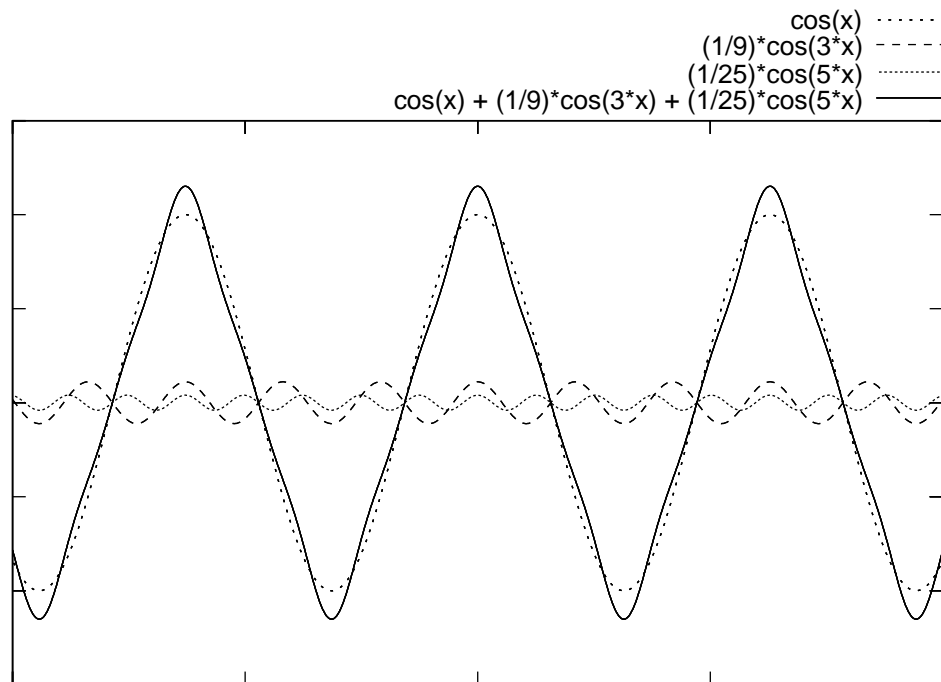


Abbildung 9: Zusammensetzung eines Dreiecksignals aus drei Sinuskomponenten

4. Bau eines digitalen Funktionsgenerators

In diesem Kapitel wird nun der Aufbau eines digitalen Funktionsgenerators beschrieben. Dabei werde ich jeweils auf die einzelnen Baugruppen (Spannungsversorgung, Prozessor, DA-Wandler, Amplituden- und Offseteinstellung und Filter) näher eingehen. Die kompletten Schaltpläne befinden sich in Anhang B.

Vor der Entwicklung des Funktionsgenerator hatte ich mir die folgenden Ziele gesetzt:

- einstellbare Signalformen: Sinus, Rechteck, Dreieck, Sägezahn
- Frequenz: ungefähr 1 Hz bis 10 kHz
- Ausgangsamplitude: bis ca. 10 V
- einstellbare Offsetspannung: bis ca. ± 6 V
- Anzeige von Frequenz und Signalform auf einem LCD

4.1. Spannungsversorgung

Die Schaltung des Funktionsgenerators benötigt mehrere verschiedene Spannungen. Zur Versorgung des Prozessors ist eine feste Gleichspannung von 5 V notwendig. Die Spannung, die zum Betrieb des Filters und des Ausgangsverstärkers benötigt wird, muss symmetrisch⁷ und vom Betrag her etwas größer als die maximal gewünschte Ausgangsamplitude des Funktionsgenerators sein, da die meisten Operationsverstärker nur Spannungen verarbeiten können, die ca. 1 V unter der Versorgungsspannung liegen. Um die angestrebte Amplitude von ca. 10 V zu erreichen, wurde für den Funktionsgenerator deshalb eine Spannung von 11,3 V gewählt.

Um diese Spannungen zu erhalten, wird ein Transformator verwendet, der eine Wechselspannung mit einem Effektivwert von 10,5 V erzeugt.

Diese Spannung wird mit Dioden gleichgerichtet und dabei in die positiven und negativen Halbwellen aufgeteilt, wobei zwei entgegengesetzt gepolte, zwischen 0 V und $\pm\sqrt{2} \cdot U_{\text{eff}}$ pulsierende Spannungen entstehen. Um aus den Sinus-Halbwellen eine echte Gleichspannung zu erhalten, müssen diese zuerst mit Kondensatoren geglättet werden. Danach ist die Spannung aber noch nicht völlig glatt, sondern schwankt immer noch um einige Prozent, und auch wegen den Unregelmäßigkeiten im Stromnetz ist eine auf diese Weise geglättete Spannung nicht besonders gut als Grundlage für ein genaues Ausgangssignal geeignet. Um eine nahezu ideale Gleichspannung zu erzeugen, werden deshalb integrierte Spannungsregler (LM317 [14],

⁷Eine symmetrische Spannungsversorgung besteht aus zwei Spannungen mit dem gleichen Betrag, aber unterschiedlichen Vorzeichen.

LM337 [15]) eingesetzt, die die Eingangsspannung auf einen niedrigeren, aber konstanten Wert begrenzen. Dieser Wert wird durch die Größe der angeschlossenen Widerstände auf $\pm 11,3\text{ V}$ festgelegt.

Um die für den Betrieb des Prozessors notwendige Spannung zu erhalten, wird ein weiterer Spannungsregler verwendet, der aus den $+11,3\text{ V}$ eine Spannung von 5 V erzeugt.

4.2. Prozessor

Die Berechnung der Funktionswerte übernimmt in dieser Schaltung ein frei programmierbarer Mikrocontroller vom Typ *ATmega8* [12].

Das Programm arbeitet nach dem in Punkt 3.1.1 beschriebenen Prinzip, und ist für die Erzeugung von Sinus-, Rechteck-, Dreieck- und Sägezahnsignalen ausgelegt⁸.

Von der Sinus- und Dreiecksfunktion wurden 256 Funktionswerte mit einer Genauigkeit von 8 Bit im Prozessor abgespeichert, die Funktionswerte für Rechteck und Sägezahn berechnet das Programm direkt. Der Additionsvorgang des Phasenakkumulators wird exakt 256.000 Mal pro Sekunde durchgeführt.

Neben der Ausgabe der Funktionswerte an den DA-Wandler ist der Prozessor für die Abfrage der Bedientasten und für die Anzeige von Signalform und Frequenz auf einem LCD zuständig.

Das Programm besteht aus 814 Zeilen Assembler-Sourcecode und ist bis auf einen kleinen Ausschnitt von 60 Zeilen meine Eigenentwicklung. Für die Kompilierung wurde eine leicht modifizierte Version des Assemblers *tavrasm* [10] verwendet; der kompilierte Maschinencode wurde mit dem Programm *uisp* [11] über einen PC-Adapter in den Prozessor geladen.

Der Programmcode ist in Anhang C sowie auf der beiliegenden CD enthalten.

4.3. Digital-Analog-Wandler

Zur Umwandlung der gespeicherten Funktionswerte in eine Spannung wird ein Widerstandsnetzwerk verwendet. Um eine Auflösung von 8 Bit zu erzielen, werden 8 digitale Ausgänge des Prozessors zusammengeschaltet, es sind also $2^8 = 256$ verschiedene Spannungen zwischen 0 V und 5 V möglich.

4.4. Amplituden- und Offseteinstellung und Filterung

Das Ausgangssignal des Digital-Analog-Wandlers bewegt sich zwischen 0 V und 5 V , es hat also eine Amplitude von $2,5\text{ V}$ und einen Gleichspannungsanteil von $2,5\text{ V}$. Da der Funktionsgenerator grundsätzlich ein Signal ohne Gleichspannungsanteil

⁸Theoretisch wäre auch jede beliebige andere Signalform darstellbar, aber andere als die genannten haben in der Praxis kaum eine Bedeutung.

erzeugen soll, wird diese *Offsetspannung* von 2,5 V vor der Weiterverarbeitung subtrahiert (*Offsetkompensation*). Das erfolgt mit der Schaltung aus [7, Bild 2].

Zusätzlich erfüllt dieser Schaltungsteil den Zweck, den DA-Wandler vom Filter zu entkoppeln. Würde man darauf verzichten, dann würde der Kondensator am Eingang des Filters auch das ungefilterte Ausgangssignal des DA-Wandlers (vgl. 3.1.2) beeinflussen. Dadurch wäre ein Umschalten vom gefilterten auf das ungefilterte Signal nicht möglich, weil auch das eigentlich ungefilterte Signal durch den danach geschalteten Filter verändert wäre.

Zur Filterung wird hier ein aktiver Bessel-Tiefpaß zweiter Ordnung verwendet, da dieser Filtertyp eine weitgehend frequenzunabhängige Laufzeit besitzt und das Signal deshalb kaum verzerrt (vgl. 3.1.3). Als Grenzfrequenz wurde der Wert 20kHz gewählt, welcher sich in Versuchen als optimaler Kompromiss zwischen guter Sinus-Glättung und möglichst unverfälschter Wiedergabe von Dreieck- Sägezahn- und Rechteckschwingungen herausgestellt hat.

Der Aufbau des Filters erfolgt nach dem Grundprinzip aus [1, S. 295]. Der Filtertyp und die Grenzfrequenz werden dabei alleine durch die Werte der verwendeten Kondensatoren und Widerstände bestimmt, die Berechnung dieser Werte wurde mit dem Programm ELEC2000 [8] durchgeführt.

An dieser Stelle wurde ein Umschalter eingebaut, der es zu Demonstrationzwecken erlaubt, vom gefilterten auf das ungefilterte Signal umzuschalten. Dass die Amplitude bei diesen Signalen nicht gleich groß ist, liegt daran, dass das Signal beim Durchgang durch den Filter um 1,3 dB verstärkt wird.

Die nächste Verarbeitungsstufe ist die Einstellung der Amplitude. Dazu wird ein als invertierender Verstärker beschalteter Operationsverstärker verwendet (vgl. [6]). Die Widerstände, die den Verstärkungsfaktor V festlegen, sind durch ein Potentiometer ersetzt, so dass die Verstärkung regulierbar ist. Der Verstärkungsfaktor wird dabei zusätzlich durch einen Vorwiderstand begrenzt, da ein Operationsverstärker eine Spannung, die größer ist als seine Versorgungsspannung, sowieso nicht erzeugen kann, und eine zu hohe Verstärkung deshalb keinen Sinn machen würde.

Eine weitere gewünschte Funktion ist eine einstellbare Offsetspannung⁹. Dazu wird ein weiterer Operationsverstärker eingesetzt, der als *Subtrahierverstärker* angeschlossen ist. Das bedeutet, dass die beiden Eingangsspannungen voneinander subtrahiert werden. Legt man an den einen Eingang das Signal, und an den anderen die gewünschte Offsetspannung, so ist das Signal am Ausgang entsprechend entlang der U -Achse verschoben. Die Offsetspannung wird durch ein Potentiometer erzeugt und kann ca. -6 V bis +6 V betragen.

Bei der Einstellung der Amplitude kann es dazu kommen, dass sich die Spannungsverschiebung zusammen mit der Amplitude ändert, obwohl die Offsetspannung auf 0 eingestellt ist. Das liegt daran, dass die zu Beginn beschriebene Off-

⁹Gleichspannungsanteil, entspricht Verschiebung der Funktion an der U -Achse

setkompensation die Offsetspannung von 2,5 V nicht völlig entfernt hat, so dass dieser minimale Unterschied durch die Amplitudenregelung mitverstärkt wird. Verhindern könnte man dies, indem man das bei der Offsetkompensation eingebaute Trimpotentiometer exakt justiert.

Bei der Einstellung von Amplitude und Offsetspannung ist darauf zu achten, dass keine höheren Spannungen als etwa 10 V entstehen (z.B. bei maximaler Amplitude und hohem Gleichspannungsanteil), weil diese Signalspitzen ansonsten abgeschnitten würden¹⁰. Eine Gefahr für die Schaltung entsteht hierdurch jedoch nicht.

4.5. Ergebnis

Schon während des Baus des Funktionsgenerators zeigte sich, dass die gesetzten Ziele relativ problemlos erfüllt werden konnten. Die Dreieck-, Rechteck- und Sägezahnoscillierungen behalten trotz Filterung bis ca. 10 kHz gut ihre Form, Sinus ist auch bei viel höheren Frequenzen (z.B. 25 kHz) noch verwendbar (auch wenn dann die Amplitude durch die Filterung stark reduziert ist). Da bis kurz vor der Fertigstellung eine so hohe Frequenz überhaupt nicht vorgesehen war, funktioniert die Frequenzanzeige bei Frequenzen größer als ca. 26 kHz nicht; auf eine Begrenzung des einstellbaren Wertes nach oben hin wurde dennoch verzichtet, damit die hohen Frequenzen trotzdem genutzt werden können.

Der Unterschied zwischen dem gefilterten und dem ungefilterten Signal wird besonders deutlich, wenn man einen Sinus von 25 kHz auf dem Oszilloskop betrachtet und zwischen den beiden Signalen umschaltet. Während das ungefilterte Signal durch die hohe Schrittweite des Akkumulators extrem gestört ist, und man deutlich die treppenartigen Abstufungen erkennt, ist das Signal bei Zuschaltung des Filters praktisch störungsfrei. Die anderen Schwingungsformen sind bei dieser Frequenz kaum noch erkennbar, da sie sich durch die fast vollständige Wegfilterung der oberen Frequenzkomponenten schon sehr stark einem Sinus nähern.

Der Ausgang des Funktionsgenerators kann maximal ca. 40 mA liefern. Obwohl die verwendeten Operationsverstärker laut [13] kurzschlußgeschützt sind, sollte man es unbedingt vermeiden einen Lautsprecher oder andere induktive Lasten an den Funktionsgenerator anzuschließen, da das Gerät durch die hohen Selbstinduktionsspannungen zerstört werden würde. Um das Signal trotzdem hörbar zu machen, kann man es an den Line-In-Eingang eines Audioverstärkers anschließen; die Amplitude sollte dabei auf 1 V eingestellt sein.

¹⁰Ein Operationsverstärker kann normalerweise nur Spannungen verarbeiten und erzeugen, die mindestens ungefähr 1 V unter der Versorgungsspannung liegen. Die höchstmögliche Spannung liegt in dieser Schaltung daher ungefähr bei 10 V.

Anhang

A. Literatur

- [1] U. Tietze, Ch. Schenk: **Halbleiter-Schaltungs-Technik**, 5. Auflage, Springer Verlag, 1980
- [2] P. Horowitz, W. Hill: **Die Hohe Schule der Elektronik Bd. 1**, 3. Auflage, Elektor Verlag, 1996
- [3] Helmut Lindner, Harry Brauer, Constans Lehmann: **Taschenbuch der Elektrotechnik und Elektronik**, 7. Auflage, Fachbuchverlag Leipzig, 1998
- [4] Dr. Papay Zsolt: **DDS technology**
<http://www.hit.bme.hu/people/papay/sci/DDS/start.htm>
- [5] Manfred Winterhoff u.a.: **de.sci.electronics FAQ**
<http://dse-faq.e-online.de/dse-faq.html>
- [6] Gunter Konig: **Operationsverstarker**
<http://www.mikrocontroller.net/articles/op.htm>
- [7] Thomas Schaerer: **Operationsverstarker I**
<http://www.e-online.de/public/schaerer/opa1.htm>
- [8] Patrick Schnabel: **ELEC2000, Praxisnahe Rechenprogramme fur die Elektronik**
<http://www.e-online.de/public/schaerer/elec2000.htm>
- [9] Andre Birua: **Some Useful Assembler Multibyte Maths Subroutines & Macrocalls**
<http://mirror01.users.i.com.ua/~birua/math32.html>
- [10] Tom Mortensen: **tavrasm**
<http://www.tavrasm.org>
- [11] Marek Michalkiewicz, Theodore A. Roth: **uisp**
<http://savannah.nongnu.org/projects/uisp/>
- [12] Atmel Corporation: **ATmega8 Datasheet**
<http://www.eu.atmel.com/atmel/acrobat/doc2486.pdf>
- [13] ST Microelectronics: **TL072 Datasheet**
<http://www.st.com/stonline/books/pdf/docs/2298.pdf>

- [14] National Semiconductor: **LM117/317 Datasheet**
<http://www.national.com/ds/LM/LM117.pdf>
- [15] National Semiconductor: **LM137/337 Datasheet**
<http://www.national.com/ds/LM/LM137.pdf>

B. Schaltplan

Die folgenden Seiten enthalten die kompletten Schaltpläne des Funktionsgenerators.

B. Schaltplan

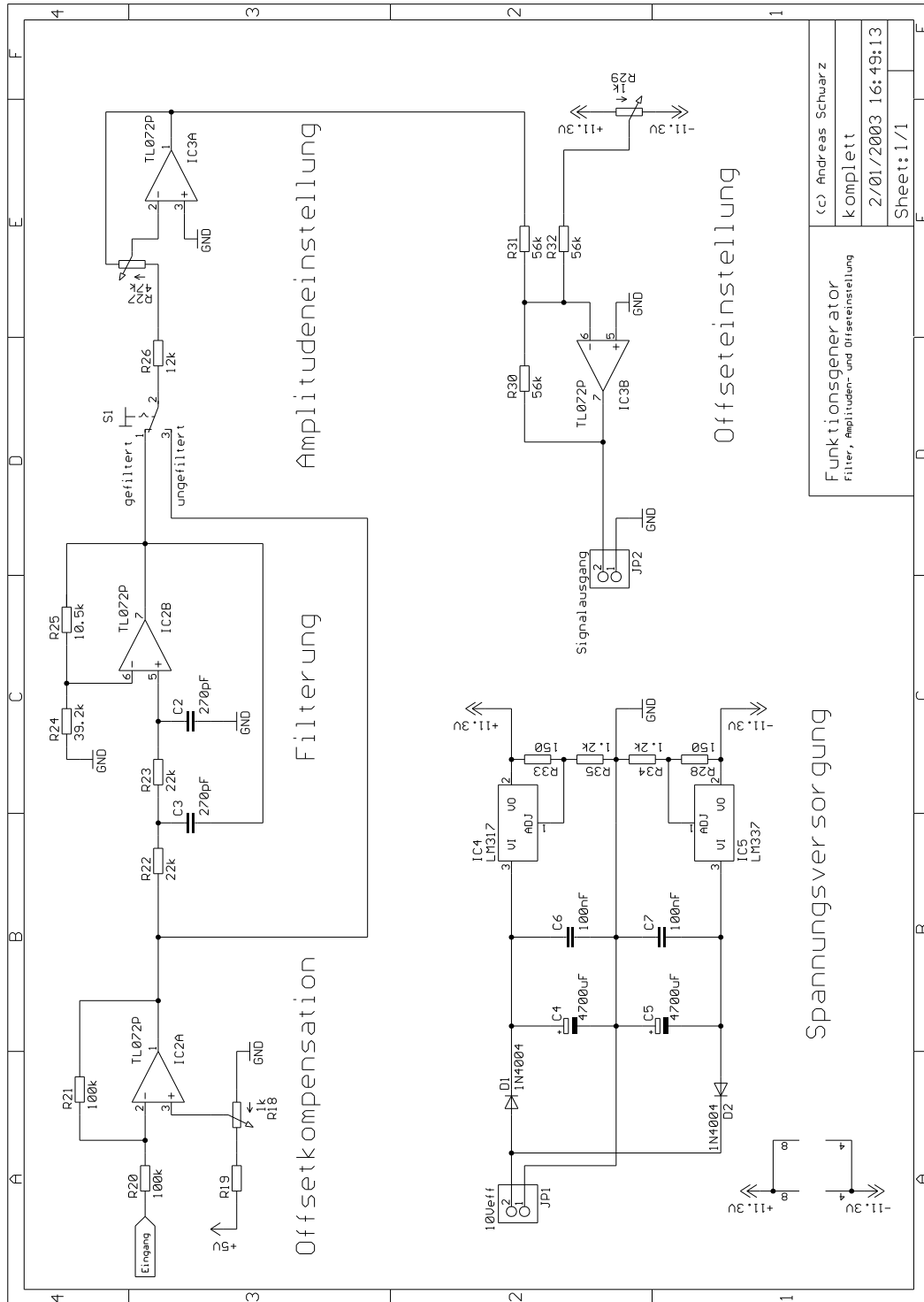


Abbildung 11: Schaltplan, Teil 2

C. Programm

```
; Digitaler Funktionsgenerator für Sinus, Rechteck, Dreieck und Sägezahn
;
; Copyright (C) 2003 Andreas Schwarz <andreas-s@web.de>
;
; This program is free software; you can redistribute it and/or
; modify it under the terms of the GNU General Public License
; as published by the Free Software Foundation; either version 2
; of the License, or (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with this program; if not, write to the Free Software
; Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
;
;
; Änderungen:
;
; ???.05.02 - Anfang: einfache Ausgabe von Sinus-Funktionswerten mit fester Frequenz
; ???.???.02 - Phasenakkumulator
;           - Einstellung der Frequenz über Taster
;           - Anzeige von Frequenz und Signalform auf LCD
; 18.10.02 - Frequenzanzeige funktioniert korrekt
; 19.10.02 - Auswahl der Signalformen
;           - Signalformen Dreieck und Sägezahn hinzugefügt
; 15.12.02 - Delay-Funktionen verbessert (jetzt teilweise mit Timer statt Warteschleife)
; 29.01.03 - in der Frequenzanzeige wird nun eine Nachkommastelle angezeigt
;           - "Überlauf" der Frequenzeinstellungs-Variable nach unten hin
;           - tritt nicht mehr auf; nach oben hin immer noch (ist aber unkritisch)
;           - Umstellung von AT90S2333 auf ATmega8 (theoretisch bessere Sicherheit für
;           EEPROM-Inhalt wegen funktionierender Brown-Out-Protection)
; 31.01.03 - feinere Frequenzeinstellung (in 0.5 Hz-Schritten)

.include "m8def.inc"

.listmac

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                               Konstanten                               ;;
;;                               =====                               ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.equ    COMPAREVALUE = 24          ; Zählerstand bei dem der Interrupt
                                   ; ausgelöst wird

.equ    KEY_UPF      = 3
.equ    KEY_UP       = 2
.equ    KEY_DOWN     = 1
.equ    KEY_DOWNNF   = 0
.equ    KEY_WAVE     = 4

.equ    STEPF        = 256        ; starke Änderung der Phasenakk.-Schrittweite
.equ    STEP         = 32         ; schwache Änderung der Phasenakk.-Schrittweite
```

C. Programm

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          Registerdefinitionen          ;;
;;          =====                      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.def      null      = R1          ; immer 0
.def      full      = R2          ; immer "voll" = 0xFF = 255 = 0b11111111

.def      sregsave  = R9          ; Zwischenspeicher für das Statusregister
                                   ; während der Interruptroutine

.def      temp1     = R16         ; Zwischenspeicher 1
.def      temp2     = R17         ; Zwischenspeicher 2
.def      temp3     = R18         ; Zwischenspeicher 3

.def      change1   = R3          ; Schrittweite, die durch den Phasenakkumulator
                                   ; zur Phasenposition addiert wird
                                   ; (24 Bit, auf 3 Register a 8 Bit aufgeteilt)

.def      change2   = R4
.def      change3   = R5

.def      z1        = R28         ; Dieses Registertripel enthält die Phasenposition
.def      z2        = R29         ; zu der die Register changeX bei der Phasenakk.
.def      z3        = R30         ; addiert werden

.def      waveform  = R20         ; eingestellte Wellenform
                                   ; (Sinus, Rechteck, Dreieck oder Sägezahn)

; Register für die Binär nach Dezimal-Wandlung
.def      fbin0     = R6          ; binary value byte 0 (LSB)
.def      fbin1     = R7          ; binary value byte 1
.def      fbin2     = R21         ; binary value byte 2
.def      fbin3     = R22         ; binary value byte 3 (MSB)
.def      tBCD0     = R21         ; BCD value digits 1 and 0 (same as fbin2)
.def      tBCD1     = R22         ; BCD value digits 3 and 2 (same as fbin3)
.def      tBCD2     = R23         ; BCD value digits 4 and 5
.def      tBCD3     = R24         ; BCD value digits 6 and 7
.def      tBCD4     = R25         ; BCD value digits 8 and 9
.def      fbinL     = R26         ; binary value Low byte
.def      fbinH     = R27         ; binary value High byte

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          Makros          ;;
;;          =====          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Z-Pointer auf bestimmte Adresse setzen
.MACRO SetZPtr ;(Adresse)
    ldi ZL, LOW(@0)
    ldi ZH, HIGH(@0)
.ENDMACRO

;String aus dem EEPROM auf dem LCD anzeigen
.MACRO print ;(String-Adresse)
    push temp1
    ldi      temp1,@0
    rcall printstring

```

C. Programm

```
        pop temp1
.ENDMACRO

;LCD-Cursor an bestimmte Position setzen
.MACRO locate ;(Zeile,Spalte)
        push temp1
        ldi temp1, 0b10000000|(((@0)-1)<<6)|((@1)-1)
        rcall lcd_command
        pop temp1
.ENDMACRO

;bestimmte Anzahl ms warten
.MACRO waitms
        push temp1
        ldi temp1, @0*6
        rcall wms
        pop temp1
.ENDMACRO

.MACRO wait10ms
        push temp1
        ldi temp1, @0*6
        rcall w10ms
        pop temp1
.ENDMACRO

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          Interruptvektoren          ;;
;;          =====                   ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.org 0x000                                ; kommt ganz an den Anfang des Speichers
        rjmp RESET                        ; (hier fängt die Programmausführung an!)
                                           ; der Timer-Interrupt und die Tabellen werden
                                           ; übersprungen -> zum Hauptprogramm

.org 0C1Aaddr                             ; Int.vektor für Timer 1 Compare Match

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          Timer-Interrupt          ;;
;;          =====                   ;;
;;          ;;                         ;;
;; Dieser Programmteil wird exakt 256000 Mal pro Sekunde      ;;
;; durch den Timer aufgerufen. Das Hauptprogramm wird dazu    ;;
;; unterbrochen und anschließend fortgesetzt.                ;;
;;          ;;                         ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

timer:
        in sregsave, SREG                 ; Statusregister sichern

; Sägezahn und Rechteck bekommen eine Sonderbehandlung:
        cpi waveform, t_square            ; ist Rechteck eingestellt?
        breq square                       ; wenn ja, springe zu square
        cpi waveform, t_sawtooth         ; ist Sägezahn eingestellt?
        breq sawtooth                    ; wenn ja, springe zu sawtooth

;Bei Sinus oder Dreieck wird einfach der Wert aus dem Speicher gelesen
```

C. Programm

```
lpm                                ; Funktionswert aus dem Speicher lesen
add z1, change1                    ; Phasenakkumulator
adc z2, change2
adc z3, change3
out PORTD, R0                       ; Wert an den Analog-Digital-Wandler ausgeben
out SREG, sregsave                  ; Statusregister wiederherstellen
reti                                ; Interrupt verlassen, zurück zum Hauptprogramm

square:                             ; falls Rechteck:
add z1, change1                    ; Phasenakkumulator
adc z2, change2
adc z3, change3
cpi z3, 128                          ; Wenn die Mitte einer Schwingungsperiode
                                        ; überschritten ist...

brlo square_1
out PORTD, null                      ; ...dann minimale Ausgangsspannung (0)
out SREG, sregsave                  ; Statusregister wiederherstellen
reti                                ; Interrupt verlassen, zurück zum Hauptprogramm

square_1:
out PORTD, full                      ; ...andernfalls maximale Ausgangsspannung
out SREG, sregsave                  ; (siehe oben)
reti                                ;

sawtooth:
add z1, change1                    ; Phasenakkumulator
adc z2, change2
adc z3, change3
out PORTD, z3                        ; Beim Sägezahn ist die Spannung proportional zur
                                        ; Position in der Phase, das höchstwertige Byte
                                        ; der Position kann also direkt an den AD-Wandler
                                        ; gegeben werden

out SREG, sregsave
reti

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Tabellen mit den Funktionswerten                                         ;;
;; =====                                                                ;;
;;                                                                           ;;
;; Für Sinus und Dreieck sind hier die vorausberechneten                 ;;
;; Funktionswerte abgespeichert                                           ;;
;;                                                                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.org 0x100
SINTABLE:
.db 0x80,0x82,0x85,0x88,0x8C,0x8F,0x92,0x95
.db 0x98,0x9B,0x9E,0xA1,0xA4,0xA7,0xAA,0xAD
.db 0xB0,0xB3,0xB6,0xB9,0xBB,0xBE,0xC1,0xC3
.db 0xC6,0xC9,0xCB,0xCE,0xD0,0xD3,0xD5,0xD7
.db 0xDA,0xDC,0xDE,0xE0,0xE2,0xE4,0xE6,0xE8
.db 0xE9,0xEB,0xED,0xEE,0xF0,0xF1,0xF3,0xF4
.db 0xF5,0xF6,0xF8,0xF9,0xFA,0xFB,0xFC
.db 0xFD,0xFD,0xFE,0xFE,0xFE,0xFF,0xFF,0xFF
.db 0xFF,0xFF,0xFF,0xFF,0xFE,0xFE,0xFE,0xFD
.db 0xFD,0xFC,0xFB,0xFA,0xF9,0xF9,0xF8,0xF6
.db 0xF5,0xF4,0xF3,0xF1,0xF0,0xEE,0xED,0xEB
```

C. Programm

```
.db 0xE9,0xE8,0xE6,0xE4,0xE2,0xE0,0xDE,0xDC
.db 0xDA,0xD7,0xD5,0xD3,0xD0,0xCE,0xCB,0xC9
.db 0xC6,0xC3,0xC1,0xBE,0xBB,0xB9,0xB6,0xB3
.db 0xB0,0xAD,0xAA,0xA7,0xA4,0xA1,0x9E,0x9B
.db 0x98,0x95,0x92,0x8F,0x8C,0x88,0x85,0x82
.db 0x80,0x7D,0x7A,0x77,0x73,0x70,0x6D,0x6A
.db 0x67,0x64,0x61,0x5E,0x5B,0x58,0x55,0x52
.db 0x4F,0x4C,0x49,0x46,0x44,0x41,0x3E,0x3C
.db 0x39,0x36,0x34,0x31,0x2F,0x2C,0x2A,0x28
.db 0x25,0x23,0x21,0x1F,0x1D,0x1B,0x19,0x17
.db 0x16,0x14,0x12,0x11,0xF,0xE,0xC,0xB
.db 0xA,0x9,0x7,0x6,0x5,0x4,0x3
.db 0x2,0x2,0x1,0x1,0x1,0x0,0x0,0x0
.db 0x0,0x0,0x0,0x0,0x1,0x1,0x1,0x2
.db 0x2,0x3,0x4,0x5,0x6,0x6,0x7,0x9
.db 0xA,0xB,0xC,0xE,0xF,0x11,0x12,0x14
.db 0x16,0x17,0x19,0x1B,0x1D,0x1F,0x21,0x23
.db 0x25,0x28,0x2A,0x2C,0x2F,0x31,0x34,0x36
.db 0x39,0x3C,0x3E,0x41,0x44,0x46,0x49,0x4C
.db 0x4F,0x52,0x55,0x58,0x5B,0x5E,0x61,0x64
.db 0x67,0x6A,0x6D,0x70,0x73,0x77,0x7A,0x7D
```

TRIANGLETABLE:

```
.db 0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e
.db 0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e
.db 0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e
.db 0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e
.db 0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e
.db 0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e
.db 0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e
.db 0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e
.db 0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e
.db 0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e
.db 0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae
.db 0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe
.db 0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce
.db 0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde
.db 0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee
.db 0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe
.db 0xff,0xfd,0xfb,0xf9,0xf7,0xf5,0xf3,0xf1
.db 0xef,0xed,0xeb,0xe9,0xe7,0xe5,0xe3,0xe1
.db 0xdf,0xdd,0xdb,0xd9,0xd7,0xd5,0xd3,0xd1
.db 0xcf,0xcd,0xcb,0xc9,0xc7,0xc5,0xc3,0xc1
.db 0xbf,0xbd,0xbb,0xb9,0xb7,0xb5,0xb3,0xb1
.db 0xaf,0xad,0xab,0xa9,0xa7,0xa5,0xa3,0xa1
.db 0x9f,0x9d,0x9b,0x99,0x97,0x95,0x93,0x91
.db 0x8f,0x8d,0x8b,0x89,0x87,0x85,0x83,0x81
.db 0x7f,0x7d,0x7b,0x79,0x77,0x75,0x73,0x71
.db 0x6f,0x6d,0x6b,0x69,0x67,0x65,0x63,0x61
.db 0x5f,0x5d,0x5b,0x59,0x57,0x55,0x53,0x51
.db 0x4f,0x4d,0x4b,0x49,0x47,0x45,0x43,0x41
.db 0x3f,0x3d,0x3b,0x39,0x37,0x35,0x33,0x31
.db 0x2f,0x2d,0x2b,0x29,0x27,0x25,0x23,0x21
.db 0x1f,0x1d,0x1b,0x19,0x17,0x15,0x13,0x11
.db 0x0f,0x0d,0x0b,0x09,0x07,0x05,0x03,0x01
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          Stringkonstanten (im EEPROM)                               ;;
```

C. Programm

```
;;          =====          ;;
;;          ;;
;; Hier sind die Zeichenketten gespeichert, die auf dem LCD ;;
;; angezeigt werden sollen; die Speicheradressen der ;;
;; Strings dienen gleichzeitig als Konstanten zur ;;
;; Festlegung der eingestellten Signalform. ;;
;;          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.eseg

.org 0x01
t_sinus:
.db "Sinus ",0
t_square:
.db "Rechteck ",0
t_triangle:
.db "Dreieck ",0
t_sawtooth:
.db "Saegezahn",0
t_hz:
.db " Hz",0

.cseg

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          Initialisierung          ;;
;;          =====          ;;
;;          ;;
;; Hier wird der Prozessor initialisiert: ;;
;; Stackpointer, Ein-Ausgänge, Timerfrequenzen, ... ;;
;;          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RESET:
    ldi temp1, low(RAMEND)
    out SPL, temp1                ; Stackpointer initialisieren
    ldi temp1, high(RAMEND)
    out SPH, temp1

    ldi temp1, 0x00
    out EEARH, temp1

    ldi temp1, 0xFF
    out DDRD, temp1                ; Port D = Ausgang

    ldi temp1, 0x00
    out DDRC, temp1                ; Port C = Eingang
    ldi temp1, 0xFF
    out PORTC, temp1                ; interne Pullups für Port C aktivieren

    ldi temp1, 0xFF
    out DDRB, temp1                ; Port B = Ausgang
    ldi temp1, 0x00
    out PORTB, temp1

    ldi temp1, 0x00                ; Additionsregister mit 0x00FA00 vorbelegen
    mov change1, temp1            ; (entspricht einer Ausgangsfrequenz von 1000 Hz)
```

C. Programm

```
ldi temp1, 0xFA
mov change2, temp1
ldi temp1, 0x00
mov change3, temp1

clr z1 ; Phasenposition auf 0 setzen
clr z2
clr z3

ldi waveform, t_sinus ; Voreinstellung für Ausgangssignal: Sinus

clr null
clr full
com full

SetZPtr(SINTABLE*2) ; Adresse der Sinus-Tabelle in den Z-Pointer laden

ldi temp1, low(COMPAREVALUE) ; Compare-Wert laden; bei einem Zählerstand von
; COMPAREVALUE wird der Timer-Interrupt aktiv
out OCR1AL, temp1
ldi temp1, high(COMPAREVALUE)
out OCR1AH, temp1
ldi temp1, (1<<WGM12)|(1<<CS10) ; Timer 1 starten, Frequenz 6553600/1 Hz
out TCCR1B, temp1

ldi temp1, (1<<CS02)|(1<<CS00) ; Timer 0 starten, Frequenz 6553600/1024 Hz
; (dient als Zeitgeber für die Verzögerungen
; bei der Tastenauswertung und LCD-Ansteuerung)
out TCCR0, temp1

ldi temp1, 1<<OCIE1A ; Timer 1 Compare-Interrupt aktivieren
out TIMSK, temp1

rcall lcd_init ; LCD initialisieren
rcall lcd_clear ; LCD löschen

rcall update ; Frequenzanzeige aktualisieren

sei ; Interrupts global aktivieren

; Übergang in die ...

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Hauptschleife ;
;; ===== ;
;; ;
;; Endlosschleife, wird immer wieder wiederholt; wertet ;
;; Tastendrucke aus und aktualisiert die Anzeige; wird ;
;; automatisch 256000 Mal in der Sekunde durch den ;
;; Timerinterrupt unterbrochen ;
;; ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

mainloop:
sbic PINC, KEY_WAVE ; Taste für Wellenform-Änderung gedrückt?

rjmp skip1 ; wenn nicht, dann zu skip1 springen
; andernfalls weiter
```

C. Programm

```
    cpi waveform, t_sinus           ; Sinus eingestellt?
    breq make_square               ; wenn ja, dann zu make_square springen
    cpi waveform, t_square         ; Rechteck eingestellt?
    breq make_triangle             ; wenn ja, dann zu make_square springen
    cpi waveform, t_triangle       ; Dreieck eingestellt?
    breq make_sawtooth             ; wenn ja, dann zu make_saw springen
    cpi waveform, t_sawtooth       ; Sägezahn eingestellt?
    breq make_sinus                ; wenn ja, dann zu make_sinus springen

make_sinus:
    ldi waveform, t_sinus          ; Sinus auswählen
    SetZPtr(SINTABLE*2)
    rjmp ml_wtest_end             ; zu ml_wtest_end springen
make_square:
    ldi waveform, t_square         ; Rechteck auswählen
    rjmp ml_wtest_end             ; zu ml_wtest_end springen
make_triangle:
    ldi waveform, t_triangle       ; Dreieck auswählen
    SetZPtr(TRIANGLETABLE*2)
    rjmp ml_wtest_end             ; zu ml_wtest_end springen
make_sawtooth:
    ldi waveform, t_sawtooth       ; Sägezahn auswählen
    rjmp ml_wtest_end             ; zu ml_wtest_end springen

ml_wtest_end:

    rcall update                   ; Anzeige aktualisieren
    wait10ms 30                    ; ca. 300 ms Pause

skip1: sbic PINC, KEY_DOWNF        ; Taste -- gedrückt?
    rjmp skip2
    ldi temp1, low(STEPF)
    ldi temp2, high(STEPF)
    sub change1, temp1             ; Frequenz stark verringern
    sbc change2, temp2
    sbc change3, null
    brcc PC+2                       ; bei Überlauf...
    rcall set0                       ; ... Frequenz auf 0 setzten.
    rcall update                     ; Anzeige aktualisieren
    waitms 1                          ; ca. 1 ms Pause

skip2: sbic PINC, KEY_UPF          ; Taste ++ gedrückt?
    rjmp skip3
    ldi temp1, low(STEPF)
    ldi temp2, high(STEPF)
    add change1, temp1             ; Frequenz stark erhöhen
    adc change2, temp2
    adc change3, null
    rcall update                     ; Anzeige aktualisieren
    waitms 1                          ; ca. 1 ms Pause

skip3: sbic PINC, KEY_DOWN        ; Taste - gedrückt?
    rjmp skip4
    ldi temp1, STEP
    sub change1, temp1             ; Frequenz schwach verringern
    sbc change2, null
    sbc change3, null
```

C. Programm

```
        brcc PC+2                ; bei Überlauf...
        rcall set0               ; Frequenz auf 0 setzen.
        rcall update            ; Anzeige aktualisieren
        wait10ms 22             ; ca. 220 ms Pause

skip4:  sbic PINC, KEY_UP        ; Taste + gedrückt?
        rjmp mainloop
        ldi temp1, STEP
        add change1, temp1      ; Frequenz schwach erhöhen
        adc change2, null
        adc change3, null
        rcall update            ; Anzeige aktualisieren
        wait10ms 22             ; ca. 220 ms Pause

        rjmp mainloop          ; zurück zum Anfang der Hauptschleife springen

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;           Anzeige aktualisieren                                     ;;
;;           =====                                               ;;
;;                                                                 ;;
;;                                                                 ;;
;; (Signalform anzeigen, Frequenz berechnen und anzeigen)          ;;
;;                                                                 ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

update:
        locate(1,1)             ; Signalform an Zeile 1, Pos. 1 ausgeben
        mov temp1, waveform
        rcall printstring
        locate(2,1)

        clr fbin0
        clr fbin1
        clr fbin2
        clr fbin3

                                           ; Additionswert des Phasenakk.
                                           ; mit 10 multiplizieren...

        ldi temp3, 10
update_multiplyloop:
        add fbin0, change1
        adc fbin1, change2
        adc fbin2, change3
        dec temp3
        cpi temp3, 0
        brne update_multiplyloop

                                           ; ...durch 64 teilen...

        ldi temp3, 6
update_divideloop:
        clc
        ror fbin2
        ror fbin1
        ror fbin0
        dec temp3
        cpi temp3, 0
        brne update_divideloop

                                           ; ...in Dezimalziffern umwandeln...
```

C. Programm

```
rcall bin4BCD

; ...und auf dem LCD anzeigen:
push tBCD0 ; Ziffernpaare auf den Stack PUSHen
push tBCD1
push tBCD2

; und in der umgekehrten Reihenfolge wieder runter
; POPen (wäre auch eleganter gegangen, aber es
; funktioniert)

pop temp1
push temp1
swap temp1
andi temp1, 0x0F
subi temp1, -0x30 ; Ziffer in ASCII-Code umwandeln
rcall lcd_data
pop temp1
andi temp1, 0x0F
subi temp1, -0x30 ; dito
rcall lcd_data

pop temp1
push temp1
swap temp1
andi temp1, 0x0F
subi temp1, -0x30
rcall lcd_data
pop temp1
andi temp1, 0x0F
subi temp1, -0x30
rcall lcd_data

pop temp1
push temp1
swap temp1
andi temp1, 0x0F
subi temp1, -0x30
rcall lcd_data

ldi temp1, ',' ; das Komma vor der letzten Ziffer
rcall lcd_data

pop temp1
andi temp1, 0x0F
subi temp1, -0x30
rcall lcd_data

print(t_hz) ; nach dem Wert die Einheit " Hz" anzeigen

ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Umrechnung binär nach dezimal ; ;
;; ===== ; ;
;; ; ;
;; (Quelle: http://mirror01.users.i.com.ua/~birua/math32.html) ; ;
;; ; ;
;; Funktionsprinzip: ; ;
;; - umzurechnende Zahl durch 10000 teilen ; ;
```

C. Programm

```

;; - Ergebnis = höchste Stelle der Dezimalzahl      ;;
;; - Rest durch 1000 teilen                          ;;
;; - Ergebnis = zweithöchste Stelle der Dezimalzahl ;;
;; - Rest durch 100 teilen                          ;;
;;   ...                                             ;;
;;   ..                                             ;;
;;   ...                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Bin2BCD20:    mov     fbinL,fbin2      ; for compatibility with Math32
              mov     fbinH,fbin3
Bin2BCD16:    ldi     tBCD2,0xff      ; initialize digit 4
binBCD_4:     inc     tBCD2
              subi    fbinL,low(10000) ; subiw fbin,10000
              sbci    fbinH,high(10000)
              brcc    binBCD_4
              ldi     tBCD1,0x9f      ; initialize digits 3 and 2
binBCD_3:     subi    tBCD1,0x10
              subi    fbinL,low(-1000) ; subiw fbin,1000
              sbci    fbinH,high(-1000)
              brcs    binBCD_3
binBCD_2:     inc     tBCD1
              subi    fbinL,low(100)   ; subiw fbin,100
              sbci    fbinH,high(100)
              brcc    binBCD_2
              ldi     tBCD0,0xa0      ; initialize digits 1 and 0
binBCD_1:     subi    tBCD0,0x10
              subi    fbinL,-10       ; subiw fbin,10
              brcs    binBCD_1
              add     tBCD0,fbinL
binBCD_ret:   ret

Bin4BCD:     rcall   Bin2BCD20
              clr     tBCD3           ; initial highest bytes of result
              ldi     tBCD4,0xfe
binBCD_loop:  subi    tBCD0,-0x33      ; add 0x33 to digit 1 and 0
              sbrs    tBCD0,3         ; if bit 3 clear
              subi    tBCD0,0x03      ; sub 3
              sbrs    tBCD0,7         ; if bit 7 clear
              subi    tBCD0,0x30      ; sub $30
              subi    tBCD1,-0x33     ; add 0x33 to digit 3 and 2
              sbrs    tBCD1,3         ; if bit 3 clear
              subi    tBCD1,0x03      ; sub 3
              sbrs    tBCD1,7         ; if bit 7 clear
              subi    tBCD1,0x30      ; sub $30
              subi    tBCD2,-0x33     ; add 0x33 to digit 5 and 4
              sbrs    tBCD2,3         ; if bit 3 clear
              subi    tBCD2,0x03      ; sub 3
              sbrs    tBCD2,7         ; if bit 7 clear
              subi    tBCD2,0x30      ; sub $30
              lsl     fbin0
              rol     fbin1           ; shift lower word
              rol     tBCD0           ; through all bytes
              rol     tBCD1
              rol     tBCD2
              rol     tBCD3
              rol     tBCD4
              brmi    binBCD_loop     ; 7 shifts w/o correction of MSD

```

C. Programm

```
rol    fbinH           ; since Bin2BCD fbinH = 0xff
brcc   binBCD_ret     ; so as to do 16 shifts in total
subi   tBCD3,-0x33    ; add 0x33 to digit 7 and 6
sbrs   tBCD3,3        ; if bit 3 clear
subi   tBCD3,0x03     ; sub 3
sbrs   tBCD3,7        ; if bit 7 clear
subi   tBCD3,0x30     ; sub $30
subi   tBCD4,-0x03    ; add 0x03 to digit 8 only
sbrs   tBCD4,3        ; if bit 3 clear
subi   tBCD4,0x03     ; sub 3
rjmp   binBCD_loop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          LCD-Ansteuerung                                     ;;
;;          =====                                           ;;
;;                                                                 ;;
;; Ein Standard-LCD mit HD44780-Controller ist an Port B      ;;
;; angeschlossen; die Ansteuerung beruht auf den              ;;
;; LCD-Routinen die ich Anfang 2002 anhand des Datenblatts    ;;
;; geschrieben habe                                          ;;
;;                                                                 ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Sendet einen im EEPROM gespeicherten Text an das LCD
printstring:
    mov temp2, temp1
    out EEARL, temp2           ; Adresse setzen
    sbi EECR, 0
nextchar:
    in temp1, EEDR
    push temp1
    push temp2
    rcall lcd_data
    pop temp2
    pop temp1
    inc temp2
    out EEARL, temp2
    sbi EECR, 0
    in temp1, EEDR
    tst temp1                   ; ist r0 gleich 0?
    brne nextchar              ; wenn nicht 0, dann zurück zu nextchar
    ret

; sendet ein Datenbyte an das LCD
lcd_data:
    mov temp2, temp1           ; "Sicherungskopie" für
                                ; die Übertragung des 2.Nibbles
    swap temp1                 ; Vertauschen
    andi temp1, 0b00001111     ; oberes Nibble auf Null setzen
    sbr temp1, 1<<4            ; entspricht 0b00010000
    out PORTB, temp1           ; ausgeben
    rcall lcd_enable           ; Enable-Routine aufrufen
                                ; 2. Nibble, kein swap da es schon
                                ; an der richtigen stelle ist
    andi temp2, 0b00001111     ; obere Hälfte auf Null setzen
    sbr temp2, 1<<4            ; entspricht 0b00010000
```

C. Programm

```
        out PORTB, temp2           ; ausgeben
        rcall lcd_enable           ; Enable-Routine aufrufen
        rcall delay50us           ; Delay-Routine aufrufen
        ret                       ; zurück zum Hauptprogramm

; sendet einen Befehl an das LCD
lcd_command:                          ; wie lcd_data, nur ohne RS zu setzen
        mov temp2, temp1
        swap temp1
        andi temp1, 0b00001111
        out PORTB, temp1
        rcall lcd_enable
        andi temp2, 0b00001111
        out PORTB, temp2
        rcall lcd_enable
        rcall delay50us
        ret

; erzeugt den Enable-Puls
lcd_enable:
        sbi PORTB, 5              ; Enable high
        nop                       ; 4 Taktzyklen warten
        nop
        nop
        nop
        cbi PORTB, 5              ; Enable wieder low
        ret

; Pause nach jeder Übertragung
delay50us:                             ; ungefähr (SEHR ungefähr) 50us Pause
        ldi temp1, $62
delay50us_:
        dec temp1
        brne delay50us_
        ret

; Längere Pause für manche Befehle
delay5ms:                               ; ~5ms Pause
        ldi temp1, $31
WLOOP0:ldi temp2, $C9
WLOOP1:dec temp2
        brne WLOOP1
        dec temp1
        brne WLOOP0
        ret                       ; wieder zurück

; Initialisierung des LCDs
lcd_init:
        ldi temp3,50
powerupwait:
        waitms 5
        dec temp3
        brne powerupwait
        ldi temp1, 0b00000011     ; muss 3mal hintereinander gesendet
        out PORTB, temp1         ; werden zur Initialisierung
        rcall lcd_enable         ; 1
        waitms 5
        rcall lcd_enable         ; 2
```

C. Programm

```
waitms 5
rcall lcd_enable           ; und 3
waitms 5
ldi temp1, 0b00000010     ; 4bit-Modus einstellen
out PORTB, temp1
rcall lcd_enable
waitms 5
ldi temp1, 0b00101000     ; noch was einstellen (Cursor, Shift, usw.)...
rcall lcd_command
ldi temp1, 0b00001100     ; ...
rcall lcd_command
ldi temp1, 0b00000100     ; fertig
rcall lcd_command
ret

; Display löschen
lcd_clear:
    ldi temp1, 0b00000001
    rcall lcd_command
    waitms 5
    ret

; Cursor zum Anfang
lcd_home:
    ldi temp1, 0b00000010
    rcall lcd_command
    waitms 5
    ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      sonstige Funktionen      ;
;;      =====      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Frequenz auf 0 setzen
set0: clr change1
      clr change2
      clr change3
      ret

; x ms warten
wms:  push temp2
      out TCNT0, null

wms_1: in temp2, TCNT0
       cp temp2, temp1
       brlo wms_1
       pop temp2
       ret

; x*10 ms warten
w10ms: push temp2
        ldi temp2, 10
w10ms_1: dec temp2
         rcall wms
         cpi temp2, 0
```

C. Programm

```
brne w10ms_1  
pop temp2  
ret
```

Ich erkläre hiermit, dass ich die Facharbeit ohne fremde Hilfe
angefertigt und nur die im Literaturverzeichnis angeführten
Quellen und Hilfsmittel benützt habe.

Coburg, den 2. Januar 2003

.....