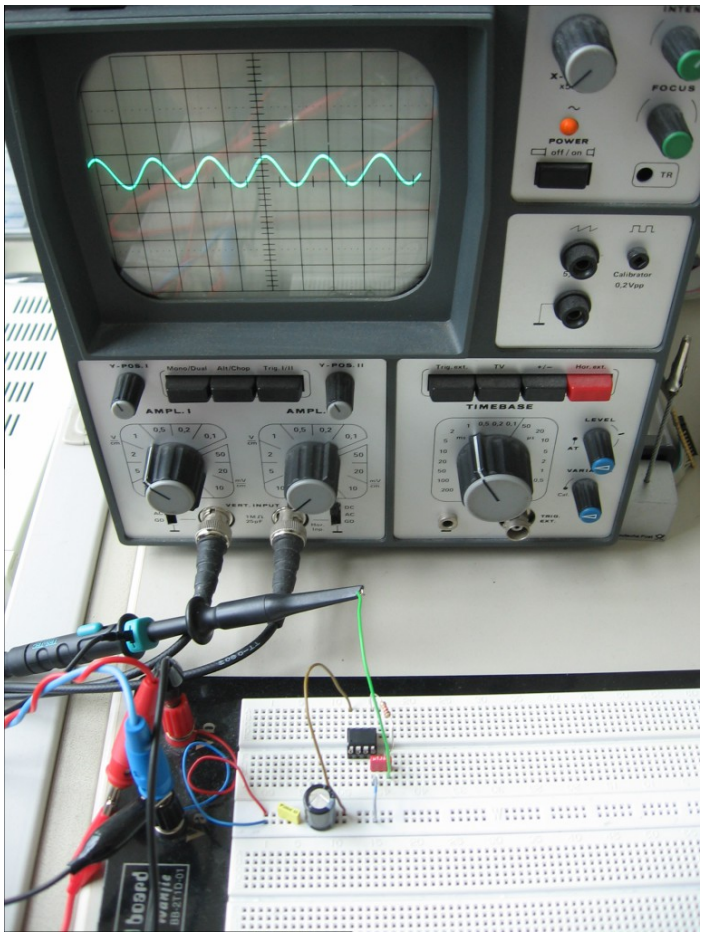
AtTiny13- NF- Sinusoszillatoren

Ralf Beesner

6.9.2011



1 Einleitung

Ein ellenlanger Thread im QRP-Forum, in dem sich Funkamateure treffen, die an Selbstbau und Funkbetrieb mit kleiner Sendeleistung interessiert sind, gab mir den Anstoß, auch einmal mit PWM- Sinusgeneratoren herumzuspielen.

Url des Threads: `http://www.qrpforum.de/index.php?page=Thread&threadID=6394`

Herausgekommen sind einige kleine Bascom- Programme für den ATtiny 13:

- die ersten drei Programme erzeugen Sinustöne mit unterschiedlicher Auflösung (64, 128 und 256 Schritte)
- beim vierten Programm lässt sich der Tongenerator tasten
- das fünfte unterdrückt Schaltknackse beim Ein- und Ausschwingen
- beim sechsten Programm kann man zusätzlich die Tonhöhe in einigen Stufen verändern

2 „fast PWM“ und „phase correct PWM“

Pulsweitenmodulation beruht auf dem Zusammenspiel eines schnellen Schalters und eines Tiefpasses. Die Schaltfrequenz ist viel höher als die resultierende Ausgangsfunktion; der Tiefpass „bügelt“ die schnellen Schaltvorgänge glatt. Je nach Tastverhältnis (also dem Verhältnis der Zeiten, die der Schalter geschlossen ist, zu den Zeiten, die er offen ist), stellt sich hinter dem Tiefpass eine mittlere Spannung ein. Ändert sich das Tastverhältnis, ändert sich auch der Mittelwert.

Der Timer im AtTiny 13 macht 8 bit- PWM; man hat also je nach Tastverhältnis eine Ausgangsspannung in 256 Stufen von 0 bis 1 (Betriebsspannung).

Bei konstantem Tastverhältnis ergibt sich hinter dem Tiefpass eine Gleichspannung; ändert sich das Tastverhältnis periodisch nach einer vorgegebenen Funktion, ergibt sich eine Ausgangskurve, die dem Änderungen des Tastverhältnisses folgt.

Dieses Verfahren nennt Atmel „fast PWM“.

Es hat einen kleinen Nachteil, der aber für die meisten Anwendungen irrelevant ist: Ändert sich das Tastverhältnis, treten „Phasensprünge“ auf. Beispiel:

- bei 256 Schritten und 10% Tastverhältnis liegt die Mitte des „1“- Signals bei 5%; die Mitte des „0“- Signals bei 55%
- bei 90% Tastverhältnis liegt die Mitte des „1“- Signals bei 55%; die Mitte des „0“- Signals bei 95%.

„Phase correct PWM“ umgeht dies, indem der Timer bei 10% Tastverhältnis von 0 bis 255 läuft, dann jedoch rückwärts zählt. Er schaltet bei 245 ein und beim rückwärts-Zählen bei 245 wieder aus. Er braucht für den Zyklus 512 Schritte, ist also nur halb so schnell, aber die Mitte des „1“- Signals liegt dann immer bei 255 Schritten und die Mitte des „0“- Signals bei 0 Schritten.

Für die meisten Anwendungen ist das nicht nötig; es hat den Nachteil, dass die PWM-Frequenz sich halbiert.

Erwähnenswert ist „phase correct PWM“ dennoch, weil der BASCOM- Befehl „Config Timer0“ nur diesen Modus kennt und man an den Registerbits herumfummeln muss, um „fast PWM“ zu erzwingen.

3 Es ist alles schon erfunden....

Der Satiriker Karl Kraus prägte mal die treffende Beschreibung vieler Besprechungen und Diskussionsrunden: „Es ist alles schon mal gesagt worden; nur noch nicht von jedem!“

So ist es auch hier; das Prinzip der Pulsweitenmodulation ist bei Wikipedia ausführlich erklärt: `http://de.wikipedia.org/wiki/Pulsweitenmodulation`

Ein Programmbeispiel im Franzis- „Lernpaket Bascom“ behandelt eine „weiche“ Blinkschaltung, in der der Mikrocontroller eine LED nicht hart schaltet, sondern mit einer PWM- Sinusfunktion ansteuert, und eines im Franzis- Buch „Mikrocontroller in C programmieren“ beschäftigt sich mit einem Funktionsoszillator, der zwar nicht mit PWM arbeitet, aber ebenfalls eine Sinusfunktion nutzt.

Aus der zweiten Quelle habe ich mir die DATA- Zeilen für die Sinusfunktionen „geborgt“, denn der Autor (Dr. Günter Spanner) hat eine Excel- Tabelle zur Berechnung der Sinusschritte erstellt, die zwar für 256 Schritte ausgelegt ist, sich aber leicht auf 128 und 64 Schritte abändern lässt.

Ich habe die resultierenden DATA- Zeilen allerdings etwas umgestellt; die beiden Autoren lassen die Sinusschwingung jeweils bei 128 beginnen. Meine beginnen nicht bei 128, sondern bei 0 (also im negativen Maximum der Sinuskurve). Das vermindert den Einschaltknacks beim Starten des Generators, der durch den abrupten Sprung des Ausgangssignales auf 128 verursacht würde.

Während der PWM- Timer bei der weichen Blinkschaltung gemächlich zu Werke gehen darf, muss er sich für NF schon richtig anstrengen: die PWM- Schaltfrequenz sollte im Ultraschallbereich liegen, damit man PWM- Reste nicht als überlagertes störendes Pfeifen hört. Der AtTiny 13 muss daher mit 9,6 MHz Takt arbeiten (bei 8 bit Auflösung und „phase correct PWM“ beträgt der PWM- Takt 1/512 des Controller- Takts, also 18,75 kHz).

Da Kinder, Jugendliche und Haustiere eventuelle PWM- Reste bei 18,75 kHz noch hören könnten, sollte man zur Sicherheit „fast PWM“ verwenden.

4 Hardware

Der Schaltplan zeigt die Maximalbeschaltung.

Als PWM- Ausgang wird OC0A / PB0 genutzt. Ein Tiefpass sorgt dafür, dass die PWM- Schaltfrequenz gedämpft und das Signal auf Kopfhörerlautstärke abgeschwächt wird.

Ein Piezoschwinger (Buzzer) ist für die Wiedergabe der relativ niedrigen Sinustöne nicht geeignet.

Taster S1 dient zum Ein- und Ausschalten des Generatorsignals; die optionalen Taster S2 und S3 dienen zur Umschaltung der Tonhöhe.

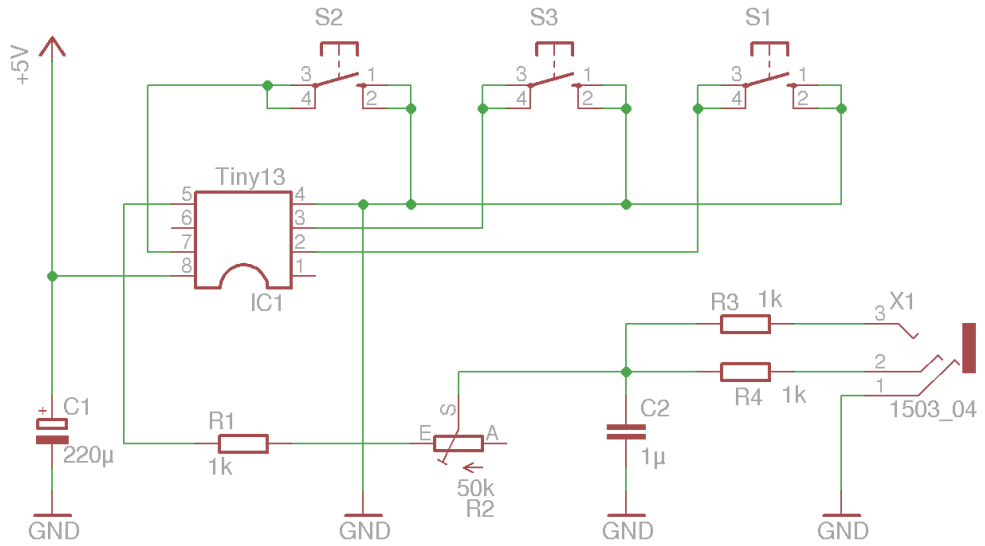


Abbildung 1: Schaltbild

Der Tiefpass ist „pi mal Daumen“ berechnet; R1 bis R4 plus Kopfhörer bilden einen zusammengefassten Widerstand von ca 500 Ohm, und mit 1 μF ergibt sich eine 3-dB-Grenzfrequenz von ca. 2 kHz. Ist die Last hochohmiger, kann man die Widerstände vergrößern und C2 proportional dazu verkleinern.

5 die drei Sinusgeneratoren

An PB0 des AtTiny liegt der Tiefpass für das Ausgangssignal, ansonsten ist keine weitere Hardwarebeschaltung erforderlich (außer dem Stützkondensator für die Betriebsspannung).

Die Software liest in einer Dauerschleife die DATA- Bytes und schreibt sie in das Pulsweitenregister des Timer0. Die Tonhöhe des Sinustons wird davon bestimmt, wie oft pro Sekunde die Do - Loop- Schleife durchlaufen wird. Durch Einfügen einiger μs Wartezeit pro Durchlauf läßt sich die Tonhöhe verringern.

Die Variante mit 256 Schritten erzeugt zwar den reinsten Sinus, schafft aber nur relativ wenige Durchläufe und ergibt daher eine recht niedrige Maximalfrequenz.

Bei der Variante mit 64 Schritten hört sich der Sinus unsauberer an, aber die Schleife wird sehr häufig durchlaufen. Ohne Verzögerungszeiten hört man einen schrillen Piepston.

Die Variante mit 128 Schritten erschien mir daher ein guter Kompromiss, mit dem ich weitergemacht habe.

6 getasteter Sinusgenerator

Das Programm „tiny-sinusoszi-switched-1.bas“ fragt zunächst den Zustand des Portpin PB3 ab. Er liegt normalerweise auf „1“, weil die internen Pullup- Widerstände der Eingänge aktiviert wurden. Drückt man die Taste S1, wird er auf „0“ gezogen, und mit erfüllter Bedingung startet die Erzeugung des Sinustons.

Wie bereits geschrieben, startet der Sinus bei “0“, also dem negativen Maximum und endet bei geöffneten Schalter auch bei „0“. Zwar hört man noch einen Knacks beim Ein- und Ausschalten des Generators, er wird aber dadurch abgemildert.

Das unvermeidliche Schalter-Prellen würde sich in „kratzig“ klingenden Ein- und Ausschwingvorgängen äussern; daher wird der Taster nicht bei jedem Durchlauf der inneren Sinus- Schleife abgefragt, sondern erst, nachdem er sich beruhigt hat (dazu dient die Schleife mit der Variablen „L“. Bei offenem Taster wird jeweils eine kleine Wartezeit (15 ms) eingelegt, bevor der Schalter erneut abgefragt wird.

7 Sinusgenerator mit weicherer Hüllkurve

Das Programm „tiny-sinusoszi-switched-2.bas“ erzeugt die ersten und letzten acht Sinus- Vollwellen mit ansteigender bzw. absteigender Amplitude, also einer weichen Hüllkurve.

Der gewählte Verlauf der Hüllkurve ist nicht optimal, sondern möglichst simpel. Ein Exponentialverlauf lässt sich am einfachsten realisieren: Shiftet man ein Byte um 1 Bit nach rechts, ist der Wert halbiert, shiftet man ihn um 7 Bit, ist der Wert auf 1/128 geteilt.

Damit kann man den Sinus sehr einfach bei den ersten 7 Schwingungen anschwellen lassen (von 1/128 auf volle Amplitude) und bei den letzten 7 Durchläufen abschwellen lassen. Da die Hüllkurve nicht optimal ist, sind die Ein- und Ausschwingvorgänge nicht völlig weich, aber das Knacksen ist noch stärker gedämpft.

8 Sinusgenerator mit wählbarer Tonhöhe

Da noch Platz im Flash war (und drei Ports des AtTiny ungenutzt), habe ich im Programm „tiny-sinusoszi-switched-3.bas“ die Möglichkeit eingebaut, die Tonhöhe in wenigen groben Stufen umzuschalten. Wenn die Tasten S2 oder S3 gedrückt sind, wird bei jedem zusätzlichen Drücken der Taste S1 die Frequenz erhöht oder vermindert.

Die umständliche Bedienung habe ich gewählt, weil dadurch die Abfrage der Tasten nicht in der Hauptschleife erfolgen muss (die zusätzlichen Befehle würde die Schleifendurchläufe verlangsamen und dadurch die maximal erreichbare Tonhöhe vermindern).

Die Tonhöhe- Stufen sind relativ grob, weil ich die einzuschiebenden Wartezeiten nicht einfach mit dem Befehl „Waitus B “ (analog zu „Waitms B“) erschlagen konnte. Es klappte einfach nicht, und im „Kleingedruckten“ der Bascom- Hilfe habe ich dann gelesen, dass „Waitus“ nicht mit Variablen als Argument aufgerufen werden darf, sondern nur mit Konstanten. Grund ist wohl, dass der Befehl möglichst wenige Prozessorzyklen verbrauchen soll, damit die übergebene Zeit (z.B. „Waitus 3“) halbwegs eingehalten wird (hier bekommt man die Grenzen der Hardware zu spüren, denn 3 μs sind bei 9,6 MHz Prozessortakt nur 29 Takte ....)

So musste ich das dann mit einer recht lahmen Do - Incr B - Loop Until B = A - Schleife lösen, die zu recht groben Abstufungen führt.

9 Fazit

Man kann die beschriebenen Sinus- Generatoren „stand alone“ verwenden oder als Unterbaugruppe. Oftmals ist es einfacher, die gewünschte Funktionalität auf zwei Mikrocontroller (und zwei Programme) aufzuteilen, statt sie mit einem Controller zu „erschlagen“. So kann man auch Interrupt- Probleme entschärfen, denn kleine Zeitverzögerungen durch Interrupt- Routinen können leicht die Tonqualität verschlechtern.