

avrmusicbox: Babys Spieluhr mit 5 Melodien

V0.39 (c) 2006 Jörg Wolfram

1 Allgemeines

Das Programm unterliegt der GPL (GNU General Public Licence) Version 2 oder höher, jede Nutzung der Software/Informationen nonkonform zur GPL oder ausserhalb des Geltungsbereiches der GPL ist untersagt! Die Veröffentlichung dieses Programms erfolgt in der Hoffnung, daß es Ihnen von Nutzen sein wird, aber OHNE IRGENDNEINE GARANTIE, auch ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK.

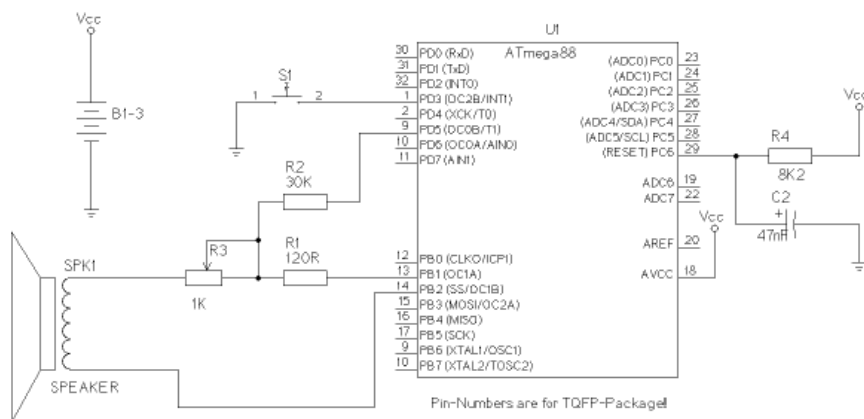
2 Geschichte

Als unser Sohn im Mai geboren wurde, haben wir ihm auch eine mechanische Spieluhr gekauft, ein niedliches Schaf. Leider war die Musik etwas zu laut und zu kurz zum Einschlafen, und spätestens beim Aufziehen war der Kleine wieder hellwach.

Zu dieser Zeit habe ich mich etwas intensiver mit den AVR-Mikrocontrollern beschäftigt und nun gab es eine Idee und einen guten Grund für ein kleines Projekt.

3 Konzept und Realisierung

Kernstück des Ganzen ist ein ATmega8 oder 88, wobei letzterer eine geringere Stromaufnahme im Sleep-Modus hat. Gerade bei Geräten mit Batteriebetrieb ein nicht zu unterschätzender Vorteil. Allerdings ist die Registererweiterung gegenüber dem ATmega8 mehr als haarsträubend realisiert, und so kommt relativ viel bedingte Assemblierung zum Einsatz.

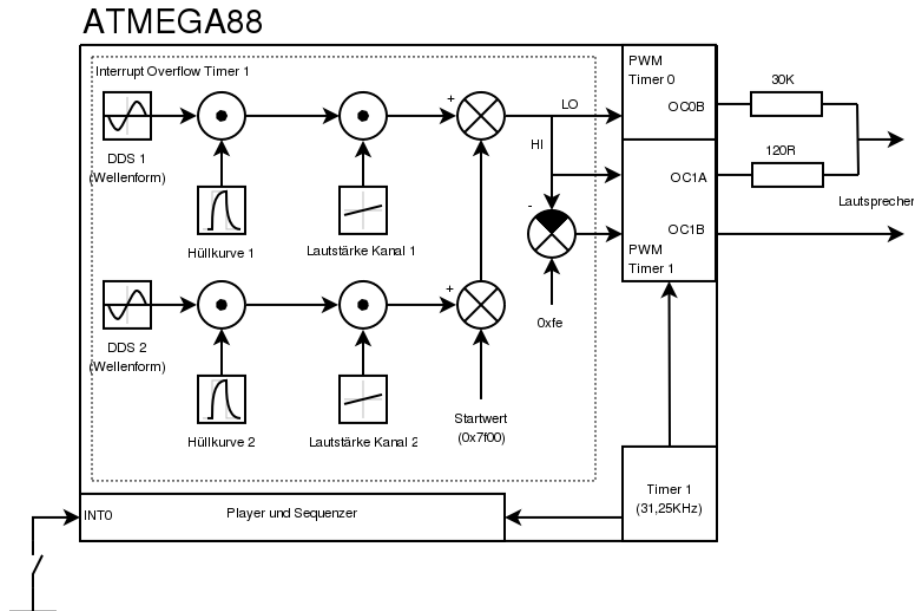


Zur Tonausgabe soll ein Lautsprecher mittels PWM an den AVR angeschlossen werden. Die meisten Konzepte, die ich im Internet gefunden habe, arbeiten mit einem Auskoppelkondensator zur gleichstrommäßigen Entkopplung des Lautsprechers. Diese Vorgehensweise hat aber einen großen Nachteil bei Batteriebetrieb, sie braucht Strom um den Kondensator umzuladen, unabhängig von der Signalamplitude. Mein Konzept braucht zwar mehr Ausgänge, der Lautsprecher wird direkt angeschlossen und der Stromverbrauch der Ausgangsstufe ist in etwa proportional zur Amplitude des Ausgangssignals.

- Ohne Signal wechseln beide PWM-Ausgänge in der Mitte der Periode ihre Polarität.
- Bei positiver Signalamplitude wechselt der Ausgang OCR1A vor Mitte der Periode auf LOW, Ausgang OCR1B nach Mitte der Periode.
- Bei negativer Signalamplitude wechselt der Ausgang OCR1B vor Mitte der Periode auf LOW, Ausgang OCR1A nach Mitte der Periode.

Das wird dadurch erreicht, daß das PWM-Register für Kanal A mit 127+Signal und das PWM für Kanal B mit 127-Signal geladen wird.

Im praktischen Betrieb reichen aber nach ersten Tests 8 Bits für die Signalamplitude nicht aus, da besonders bei kleinen Amplituden relativ starke Verzerrungen auftreten. Um die Auflösung zu erhöhen, kann leider nicht die PWM-Auflösung erhöht werden, da ansonsten die PWM-Frequenz in den hörbaren Bereich gerät. Also gibt es einen weiteren PWM-Ausgang, der einen Strom von ungefähr 1/256 des Hauptausganges auf die Seite von OCR1A einspeist. Rein theoretisch könnte somit die Auflösung auf 16 Bits hochgeschraubt werden, praktisch liegt die erreichbare Auflösung irgendwo bei 10-12 Bits. Durch Verändern von R2 können die Verzerrungen weitestgehend minimiert werden.



3.1 Die Interruptroutine

Ein wichtiger Teil der Programms ist die Interruptroutine für die Signalerzeugung. Sie läuft synchron mit der PWM-Frequenz, um Interferenzen und Jitter-Effekte zu vermeiden. Dadurch fällt natürlich die Möglichkeit aus, die Abtastrate mit der Tonfrequenz zu ändern. Aber es gibt eine einfache Lösung für das Problem, indem die Amplitude einer virtuellen Schwingung zum Samplezeitpunkt berechnet wird. Diese Technik ist auch als DDS (Direct digitally Syntheseis) bekannt. Bei jedem Aufruf der Interruptroutine wird ein Zeiger um einen Wert erhöht, der aus einer Notentabelle ausgelesen wird. Da sich die Wellenform nicht ändert, folgt jeder Schwingung eine identische weitere Schwingung, und somit wird ein etwaiger Überlauf des Zeigers ignoriert. Mathematisch lässt sich das so ausdrücken:

$$\text{Zeiger}(t) = (\text{Zeiger}(t-1) + \text{offset}) \text{ modulo } 65536$$

Der so erhaltene Zeiger wird gespeichert und als Ausgangszeiger für den nächsten Interrupt verwendet. Im aktuellen Interrupt wird der Zeiger durch 32 geteilt (eigentlich durch 64, aber die Wellenformwerte sind 2 Bytes groß), um dann mittels einer Wellenformtabelle mit 1024 Einträgen den aktuellen Signalwert zu bestimmen. Dieser wird aber nicht ausgegeben, sondern bedarf noch etwas Weiterbearbeitung.

Mit jedem Aufruf der Interruptroutine wird ein Prescaler hochgezählt und bei jedem Überlauf wird das Envelope-Bit gesetzt. Wird dies im weiteren Verlauf der Interruptroutine erkannt, wird der Envelope-Zeiger um einen Offset erhöht und das Envelope-Flag wieder zurückgesetzt. Ist sich nun der Envelopezeiger größer als die Anzahl der Einträge in der Envelope-Tabelle, wird der Kanal abgeschaltet. Andernfalls wird nun der aktuelle Lautstärkewert aus der Envelopetabelle ausgelesen und mit dem Signalwert aus der Wellenform multipliziert. Und weil wir gerade so schön beim Multiplizieren sind, wird der so erhaltene Wert mit einem 8-Bit Wert für die Kanallautstärke multipliziert. Zuguterletzt werden die Signalamplituden der beiden Kanäle addiert und die Steuerwerte für die PWM-Register berechnet.

3.2 Das Hauptprogramm

Das Haptprogramm besteht aus zwei Programmteilen, den Player mit Melodieauswahl und Power-Down und den Sequenzer. Letzterer interpretiert die Songdaten und erzeugt daraus die Parameter für die Interruptroutine.

Die Songdateien bestehen nur aus **.db Statements**, die einfach beim Assemblieren mitübersetzt werden und ein Definitionsfile sorgt dafür, daß für Noten etc. relativ plausible Symboliken verwendet werden können. Alle diese Symboliken sind genau 6 Zeichen lang, so lassen sich Fehler schneller finden.

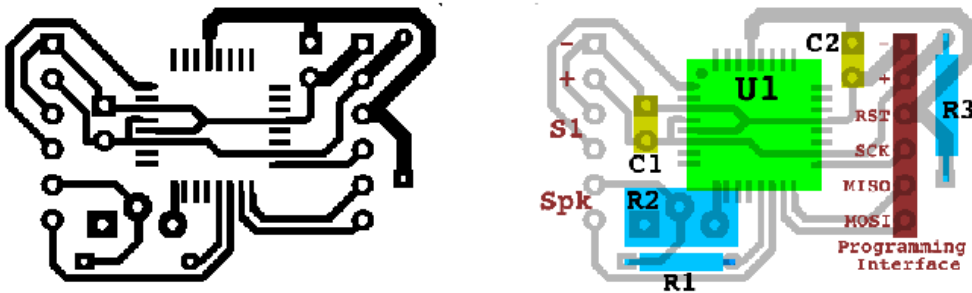
3.3 Melodien programmieren

Um eine neue Melodie zu programmieren ist es am einfachsten, eine vorhandene Songdatei als Basis zu verwenden. Am Anfang jeder Songdateien sind die verwendeten Symboliken kurz als Kommentar erklärt. Hier dazu noch ein paar Beispiele:

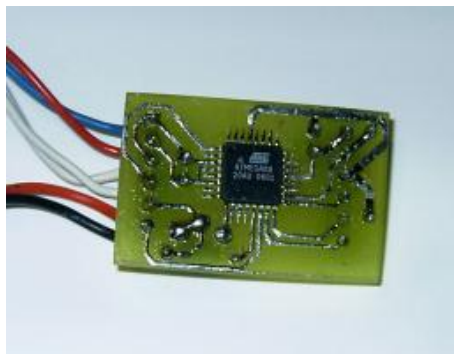
<code>_dummy</code>	Füllbyte, da jede .db-Zeile eine gerade Anzahl von Bytes enthalten muß
<code>__ende</code>	Ende der Noten
<code>_endvs</code>	Ende des Anspielens im Vorschaumodus
<code>takt16</code>	wartet 1/16 Note, bis weitere Daten gelesen werden
<code>c2__1</code>	spielt die Note C-2 im Kanal 1
<code>n108_1 set</code>	setzt die Hüllkurvengeschwindigkeit für Kanal 1 auf 1/8 Note
<code>vol__1,xxx</code>	Lautstärkewert für Kanal 1 setzen
<code>ver__2,xxx</code>	setzt den Halbtonversatz für Kanal 2 auf xxx (0...255), der wirklich gespielte Notenwert ist dann $Note=(Note+Versatz) \text{ modulo } 64$
<code>wad__1,lll,hhh</code>	setzt den Beginn der Wellenformtabelle für Kanal 1 auf die folgende Adresse Beispiel: <code>wad_1,LOW(wtab1*2),HIGH(wtab1*2)</code>
<code>ead__2,lll,hhh</code>	setzt den Beginn der Hüllkurventabelle für Kanal 2 auf die folgende Adresse Beispiel: <code>ead_2,LOW(etab1*2),HIGH(etab1*2)</code>

4 Aufbau

Das Ganze ist auf einer kleinen Platine (hier mit der SMD-Version) untergebracht:



Die folgenden zwei Bilder zeigen den Aufbau der Platine mit den externen Komponenten.



5 Bedienung

Die Bedienung der Spieluhr ist sehr einfach, als Bedienelemente gibt es nur einen Druckknopf und ein Potentiometer für die Lautstärke.

- Um die zuletzt gewählte Melodie erneut abzuspielen, muß der Knopf nur kurz gedrückt werden
- Um eine der programmierten Melodien zu wählen, muß der Knopf beim Einschalten länger als 1 Sekunde gedrückt gehalten werden. Danach werden alle Melodien nacheinander in einer Art „Schnelldurchlauf“ angespielt. Bei der gewünschten Melodie wird der Knopf einfach losgelassen, kurz darauf schaltet die Spieluhr in den normalen Abspielmodus.
- Bei den zwei letzten Strophen wird die Lautstärke verringert (soweit bei der Programmierung aktiviert). Nach dem Spielen schaltet sich die Spieluhr automatisch ab.
- Während des Spielens kann die Spieluhr durch kurzes Drücken des Knopfes abgeschaltet werden.

6 Projektstatus

V0.38 erste öffentliche Version
8.10.2006 Rückübersetzung der Dokumentation ins Deutsche