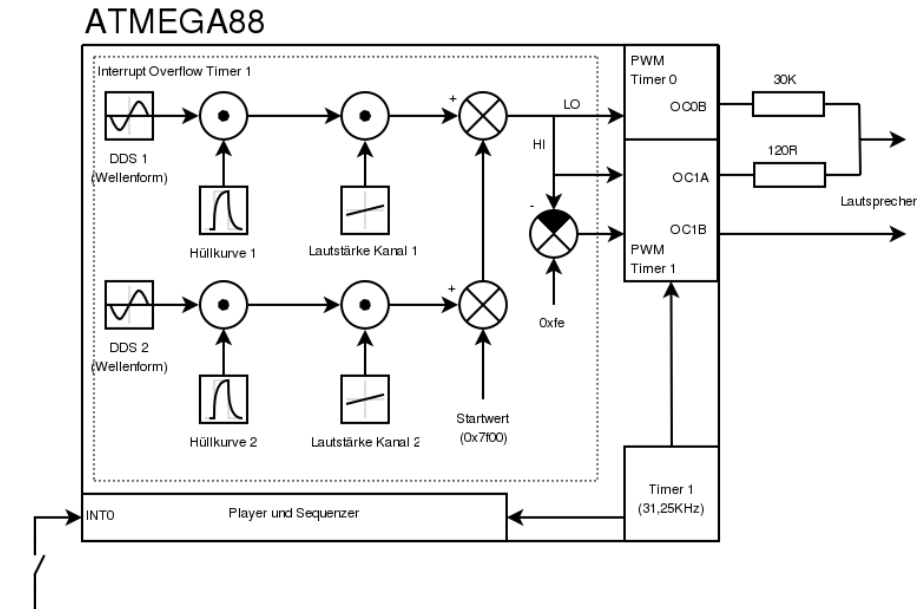


- Ohne Signal wechseln beide PWM-Ausgänge in der Mitte der Periode ihre Polarität.
- Bei positiver Signalamplitude wechselt der Ausgang OCR1A vor Mitte der Periode auf LOW, Ausgang OCR1B nach Mitte der Periode.
- Bei negativer Signalamplitude wechselt der Ausgang OCR1B vor Mitte der Periode auf LOW,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



3.1 Die Interruptroutine

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

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

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.

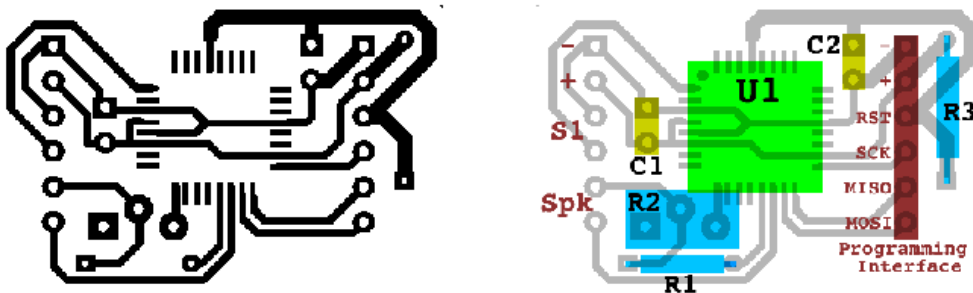
3.3 Melodien programmieren

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:

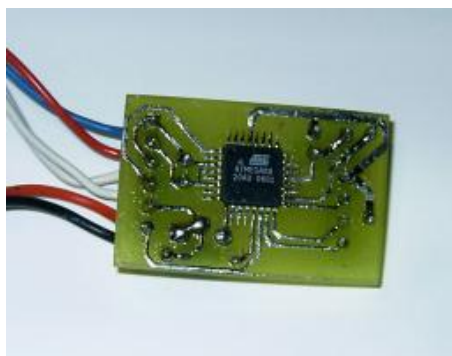
_dummy	Füllbyte, da jede .db-Zeile eine gerade Anzahl von Bytes enthalten muß
__ende	Ende der Noten
_endvs	Ende des Anspielens im Vorschaumodus
takt16	wartet 1/16 Note, bis weitere Daten gelesen werden
c2__1	spielt die Note C-2 im Kanal 1
n108_1 set	setzt die Hüllkurvengeschwindigkeit für Kanal 1 auf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

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

V0.40 etwas fehlerbereinigte Version