Arbiträrer Funktionsgenerator

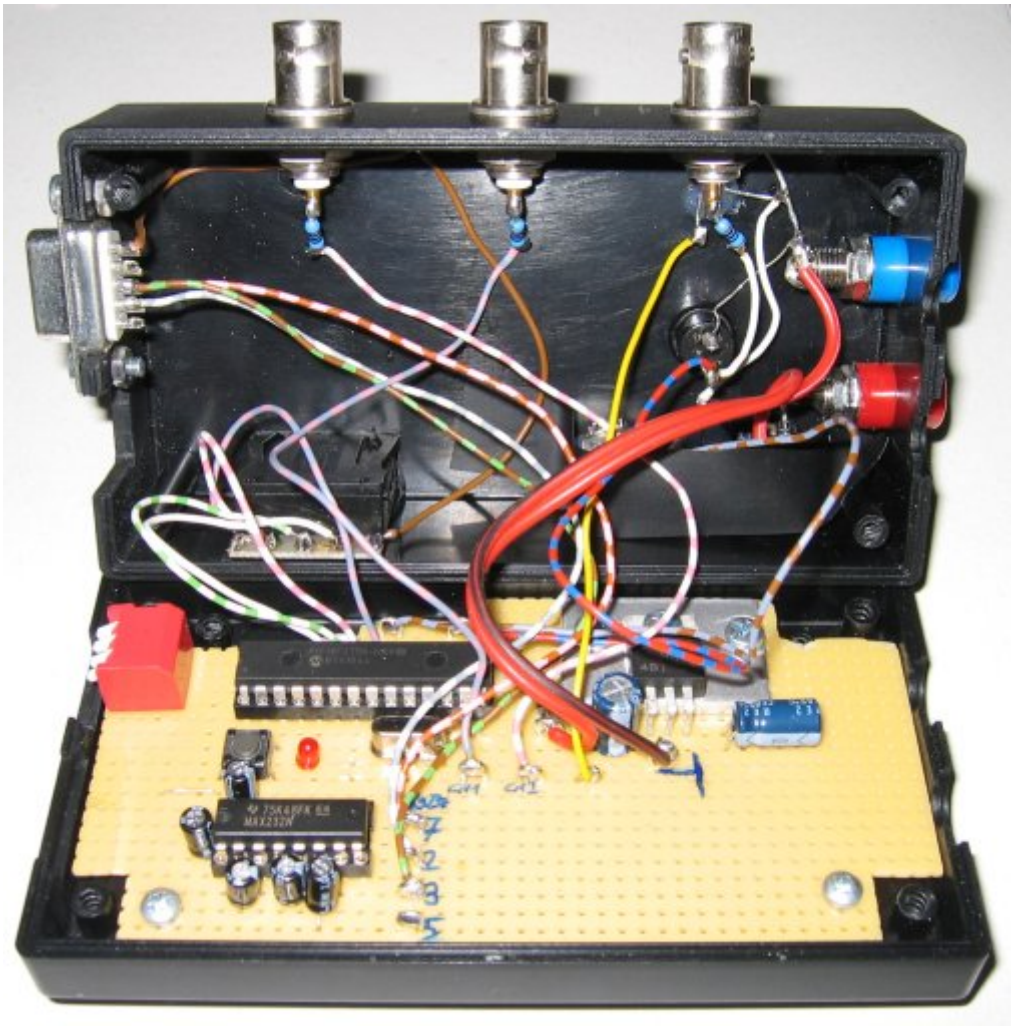
Zum testen von speziellen Schaltungen wie z.B. Frequenz-Spannungswandlern benötigte ich einen 2 kanaligen Frequenzgenerator der arbiträr arbeitet, also über der Zeit vorher einprogrammierte Frequenzverläufe ausgibt. Sicher gibts es deutlich einfachere Möglichkeiten .... Aber das ganze war auch ein Anlass sich mal wieder einen größeren PIC näher anzuschauen und die Assembler-Kenntnisse aufzufrischen !

ARB.Gen in Kürze :

Rechteck-Frequenzausgabe auf 2 getrennten Kanälen, je 0 bis 4kHz, TTL Single oder Repetitive Modus, Triggerbar von aussen per Taster oder TTL 16 verschiedene Patterns für beide Kanäle Kurvenformen per BCD-Schalter wählbar Pro Kanal und Pattern sind je max. 64 Zeitpunkte mit Angabe der Frequenzen möglich Max. Zeitangabe 32Sekunden in einem Raster von 1ms per RS232 von aussen mit neuen Kurvenverläufen programmierbar (im Source per Bootloader) Versorgung 8..16V (je nach Kühlkörper des Spannungsreglers 🙂

Aussenansicht von ARB.GEN :

Was steckt in ARB.GEN ?



Im wesentlichen nur ein PIC18F2550, 5V Spannungsregler, RS232 Pegelwandler sowie ein paar Tasten, Buchsen und Schalter. Aufgebaut wie immer auf Lochraster. Der freie Platz unten rechts auf der Platine war nur Vorhalt für Erweiterungen.

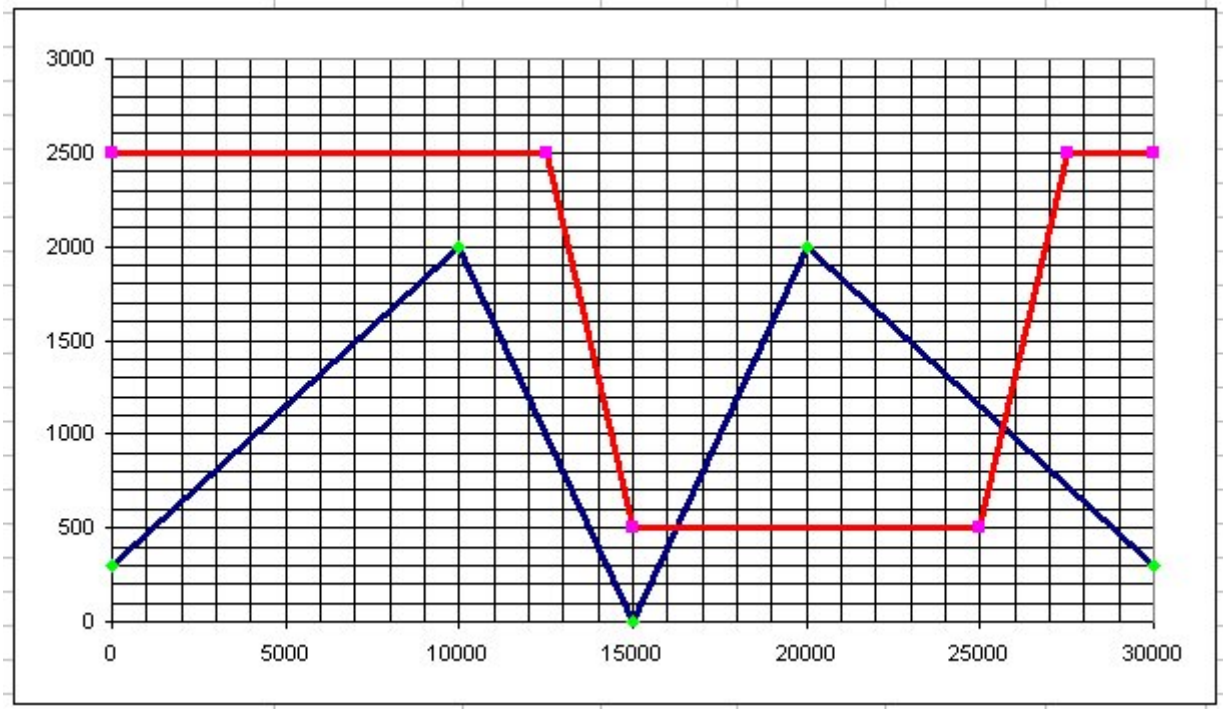
Bedienung :

Zuerst muss an sich überlegen, welche Kurvenformen man abfahren möchte auf beiden Kanälen. Bei ständiger wiederholung (Repetitive) sollte der letzte Datenpunkt auf der selben Frequenz sein wie der erste. Dazu gilt das für bei Frequenztabellen der Endpunkt gleich ist, d.h. eine Tabelle darf nicht bei 10sec. aufhören und die andere erst bei 25sec.

Am besten eine kleine Tabelle erstellen, sich das Ergebnis schon mal optisch anschauen und die Werte auch gleich in Hex-Werte umrechnen lassen :

Kanal 1 Blau			
Zeit [ms]	Freq. [Hz]	Zeit [hex]	Freq. [hex]
0	300	0000	012C
10000	2000	2710	07D0
15000	0	3A98	0000
20000	2000	4E20	07D0
30000	300	7530	012C

Kanal 2 Rot			
Zeit [ms]	Freq. [Hz]	Zeit [hex]	Freq. [hex]
0	2500	0000	09C4
12500	2500	30D4	09C4
15000	500	3A98	01F4
25000	500	61A8	01F4
27500	2500	6B6C	09C4
30000	2500	7530	09C4

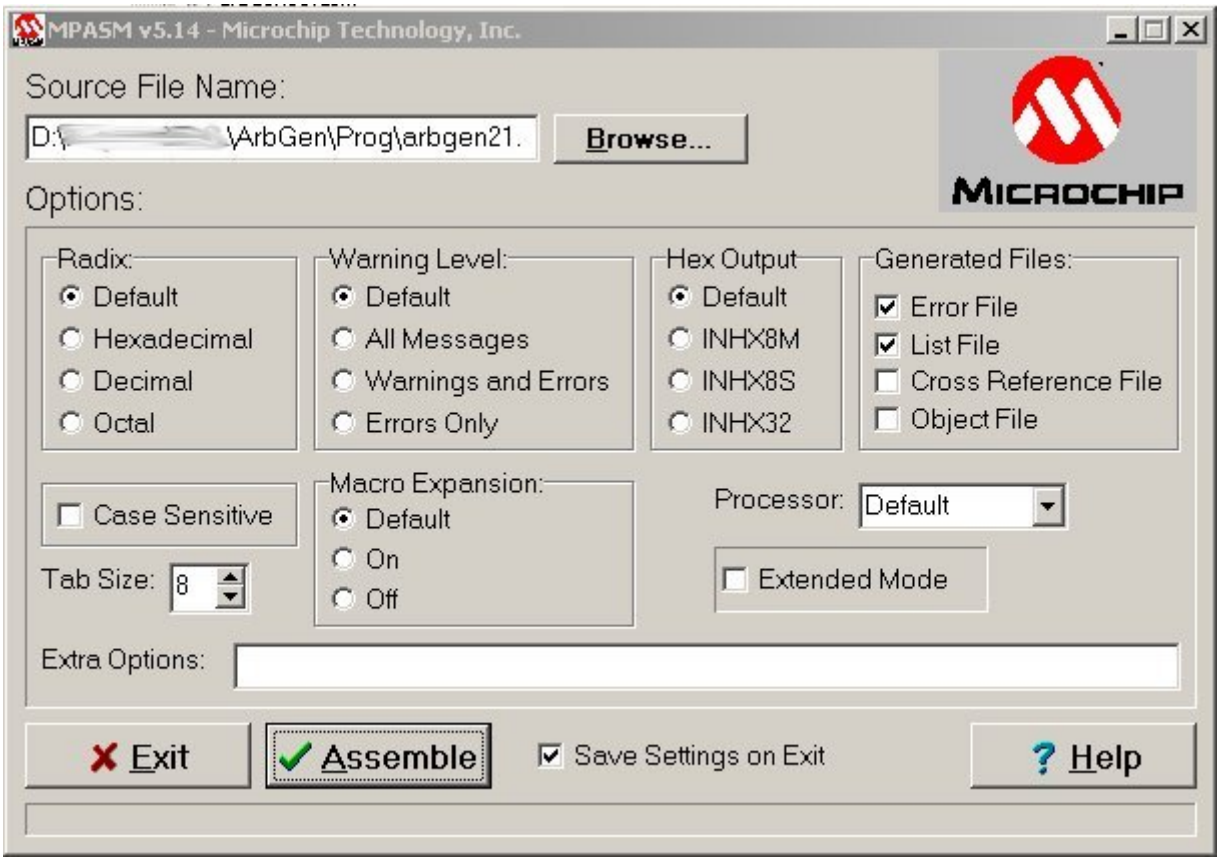


Diese Tabelle muss nun in den Source-Code übernommen werden an die gewünschte Nummer des Speichers (hier Waveform WF00). Alle unbenutzten Speicher sind auf 4x$FF gesetzt :

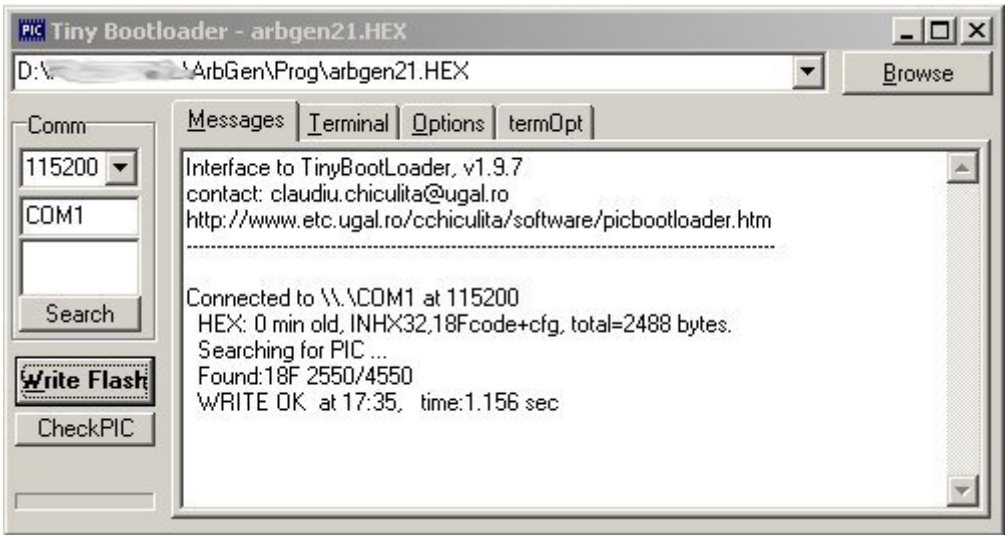
```
    Im Datenteil des Source-Codes :

    ORG    1000H        ; WF00, CH1
    db    0x00,0x00,0x01,0x2C
    db    0x27,0x10,0x07,0xD0
    db    0x3A,0x98,0x00,0x00
    db    0x4E,0x20,0x07,0xD0
    db    0x75,0x30,0x01,0x2C
    db    0xFF,0xFF,0xFF,0xFF
    ORG    1100H        ; WF00, CH2
    db    0x00,0x00,0x09,0xC4
    db    0x30,0xD4,0x09,0xC4
    db    0x3A,0x98,0x01,0xF4
    db    0x61,0xA8,0x01,0xF4
```

```
    db    0x6B,0x6C,0x09,0xC4
    db    0x75,0x30,0x09,0xC4
    db    0xFF,0xFF,0xFF,0xFF
    ORG    1200H        ; WF01, CH1
    db    0xFF,0xFF,0xFF,0xFF
    ORG    1300H        ; WF01, CH2
    db    0xFF,0xFF,0xFF,0xFF
    ...
```

Dann den Source-Code neu kompilieren mit den MPASMWIN.EXE :



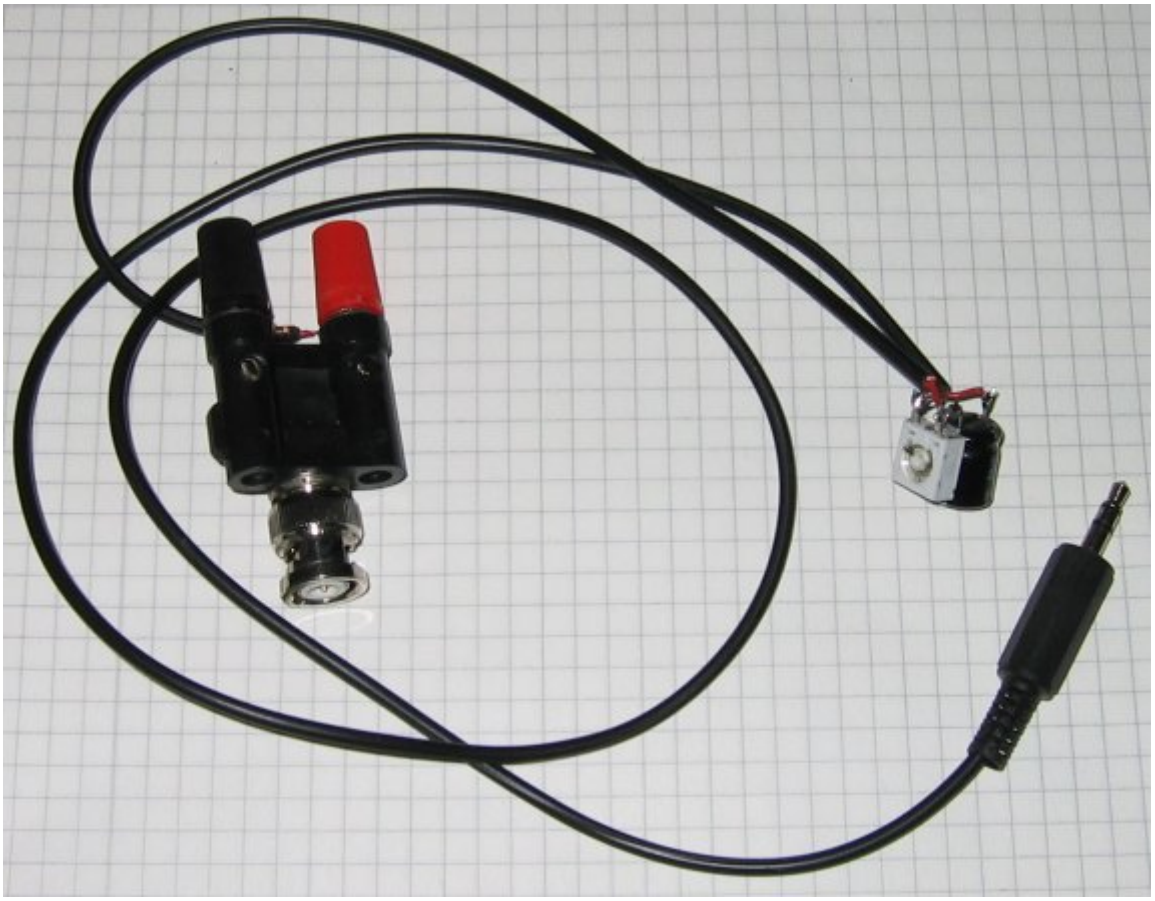
Damit bekommen wir das HEX File arbgen21.HEX welches wir nur per Bootloader in den PIC hineinladen:



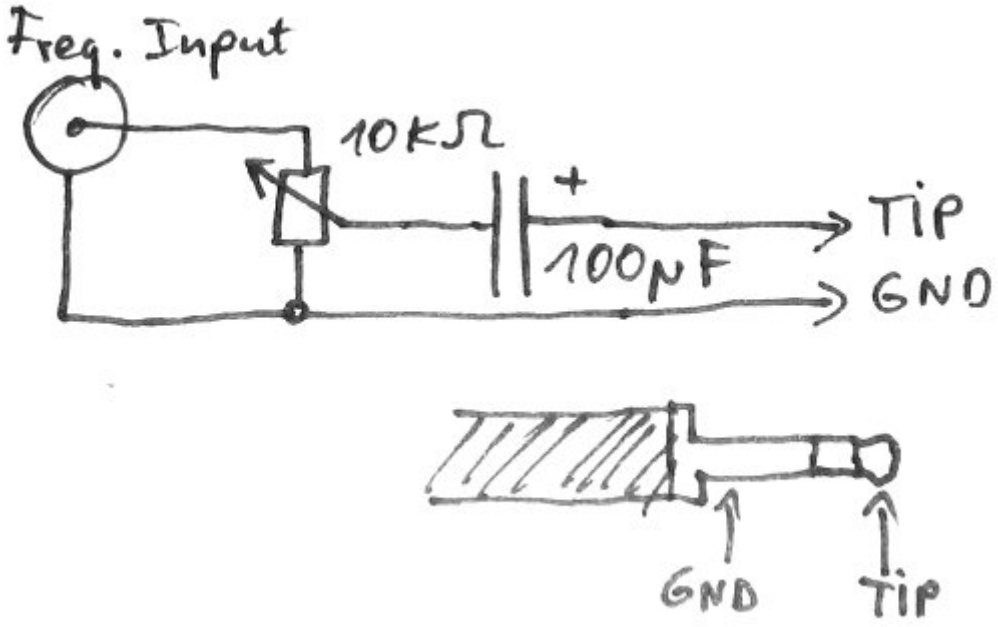
Das wärs auch schon. Ab jetzt liegt unter Waveform „0“ die Kurvenform von Kanal 1 und Kanal 2.

Wie überprüft man nun die Ausgänge ? Ein normaler Frequenzzähler kommt bei schnellen Frequenzänderungen nicht mehr mit. Mit Hilfe einer Soundkarte und einem FFT-Programm Spectrum Lab : http://freenet-homepage.de/dl4yhf/spectra1.html Spectrogram 16 : http://www.visualizationsoftware.com/gram.html

Dann baut man sich einen kleinen Adapter und los gehts :



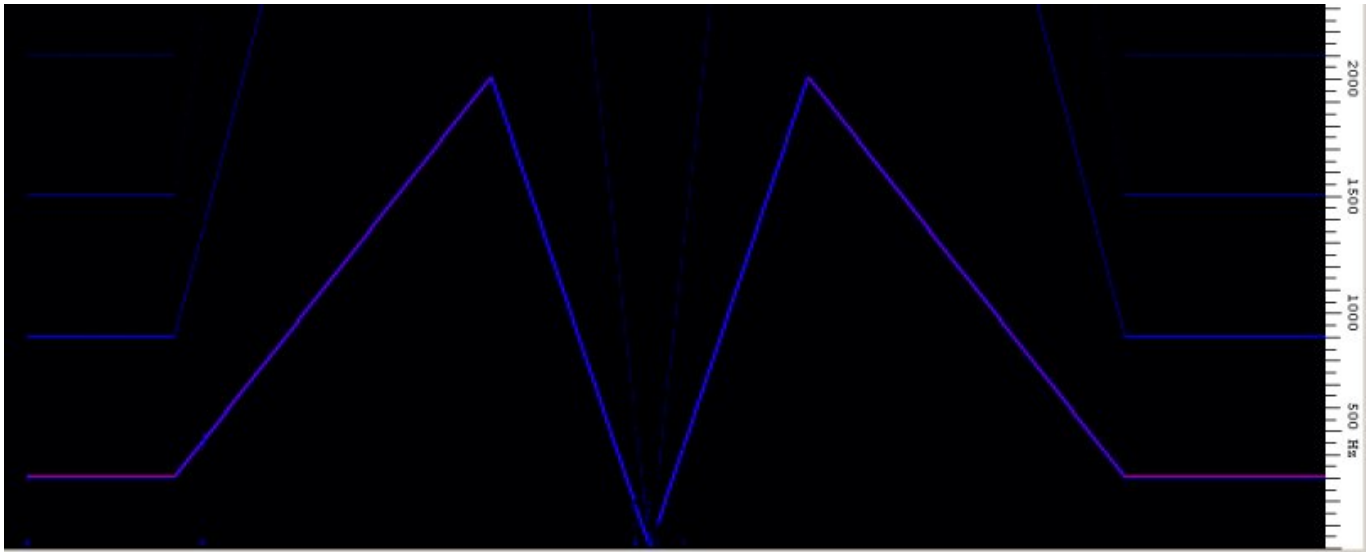
Schaltung :



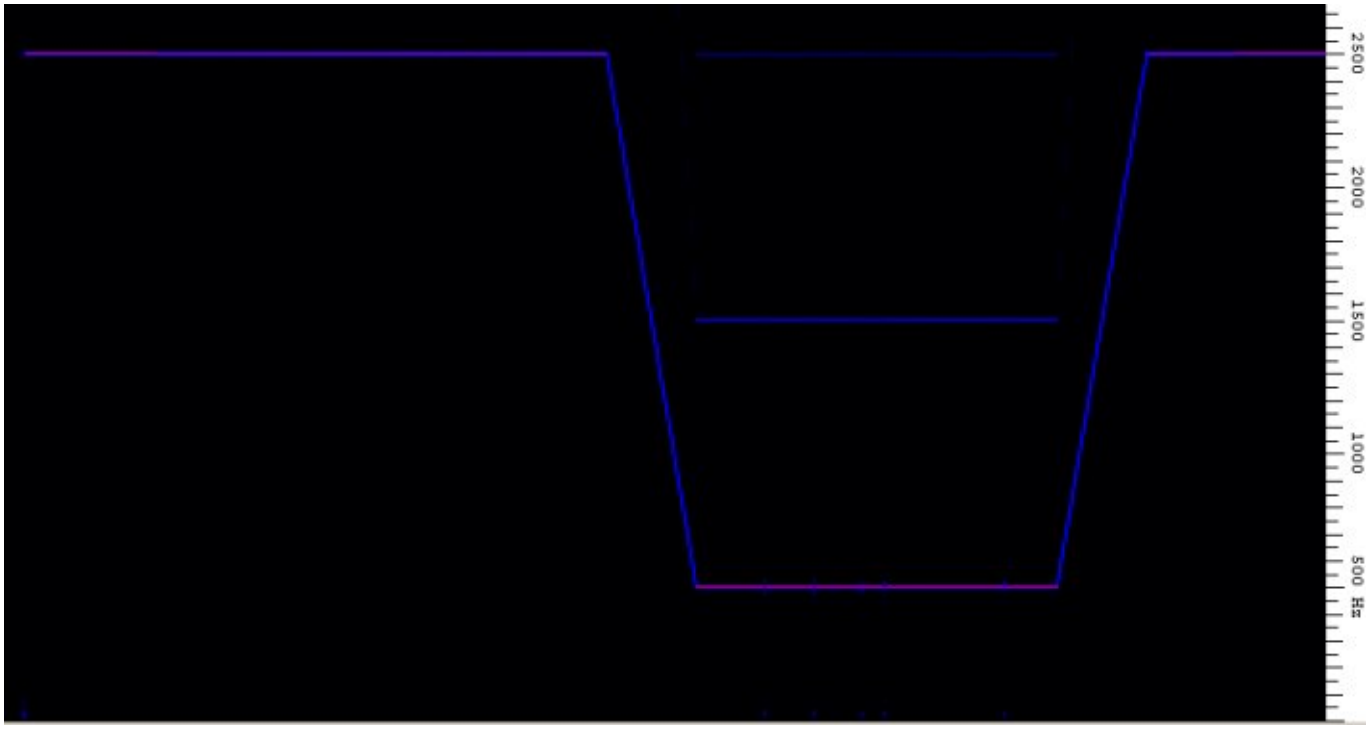
Nun kann man bequem die Kanäle betrachten. Da der PIC ein Rechteck ausgibt, sind damit natürlich

auch vielfache der Frequenzen vorhanden, welche dann als dunkle Linien zu sehen sind :

Kanal 1 (Single, am Anfang einige Sekunden kein Trigger!):

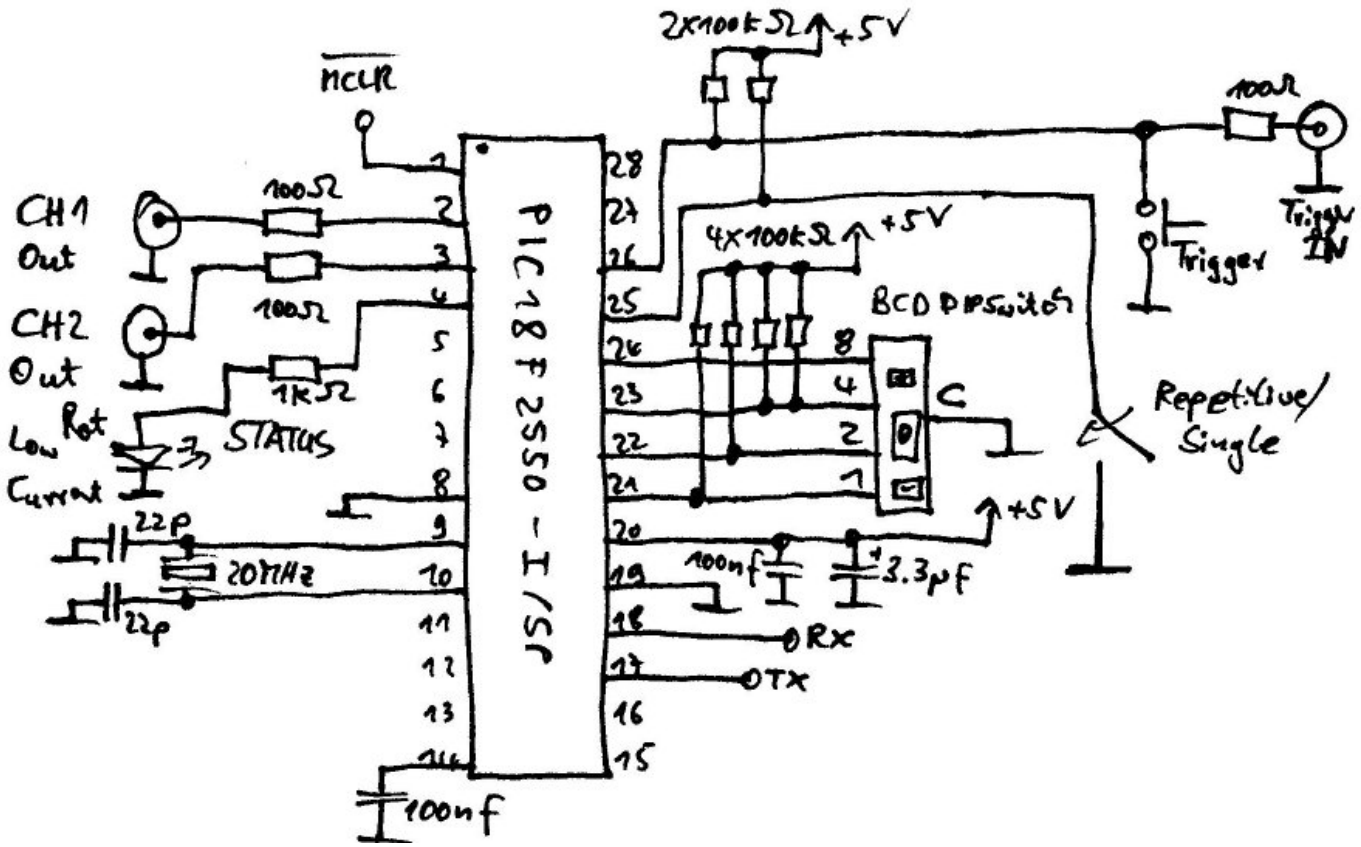


Kanal 2 (Single, am Anfang einige Sekunden kein Trigger!):

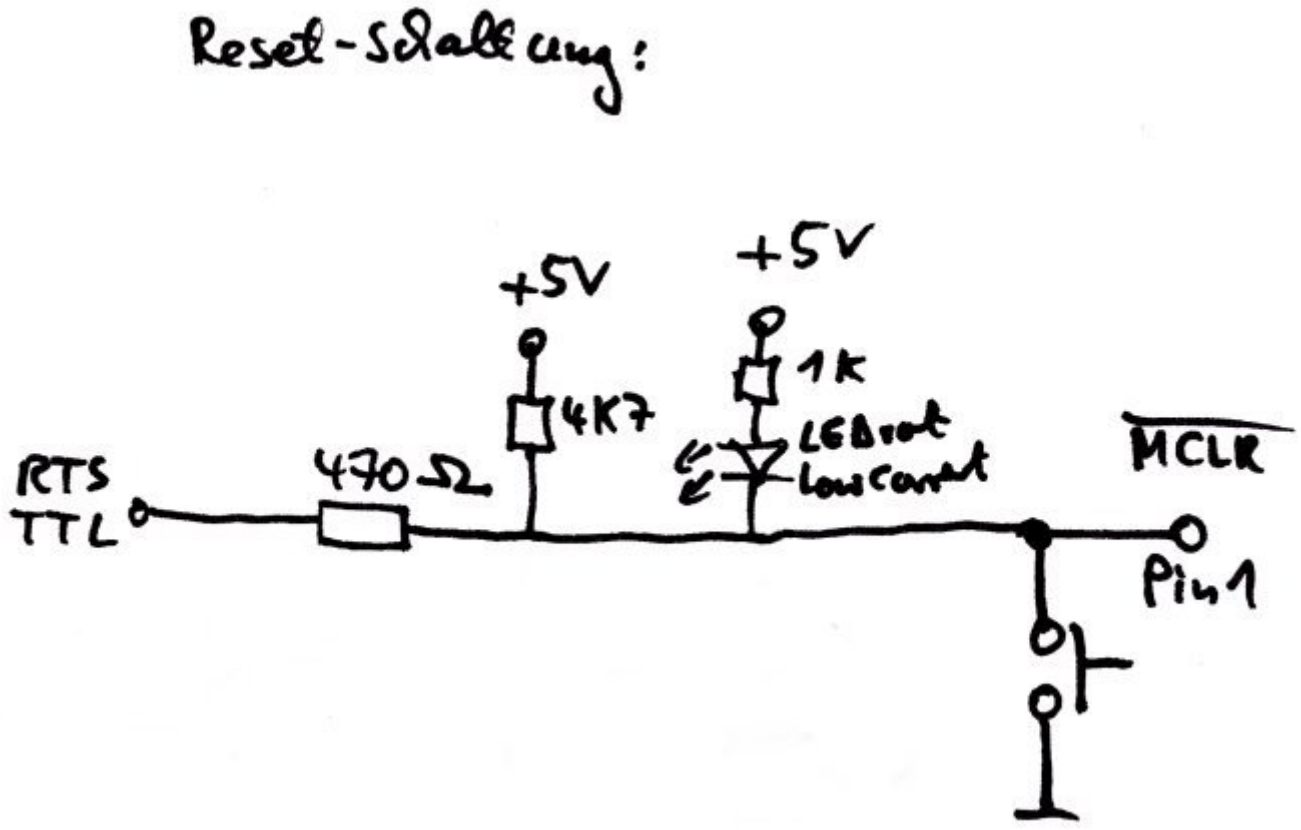


Schaltungsbeschreibung :

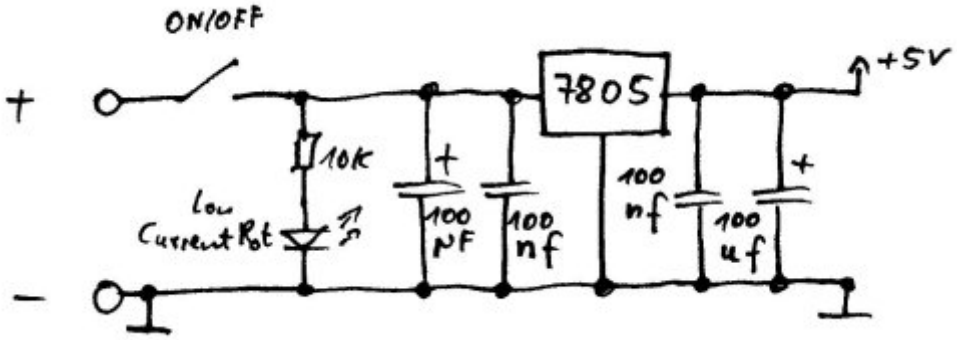
Der PIC mit den Bedienelementen sowie den Ein/Ausgängen:



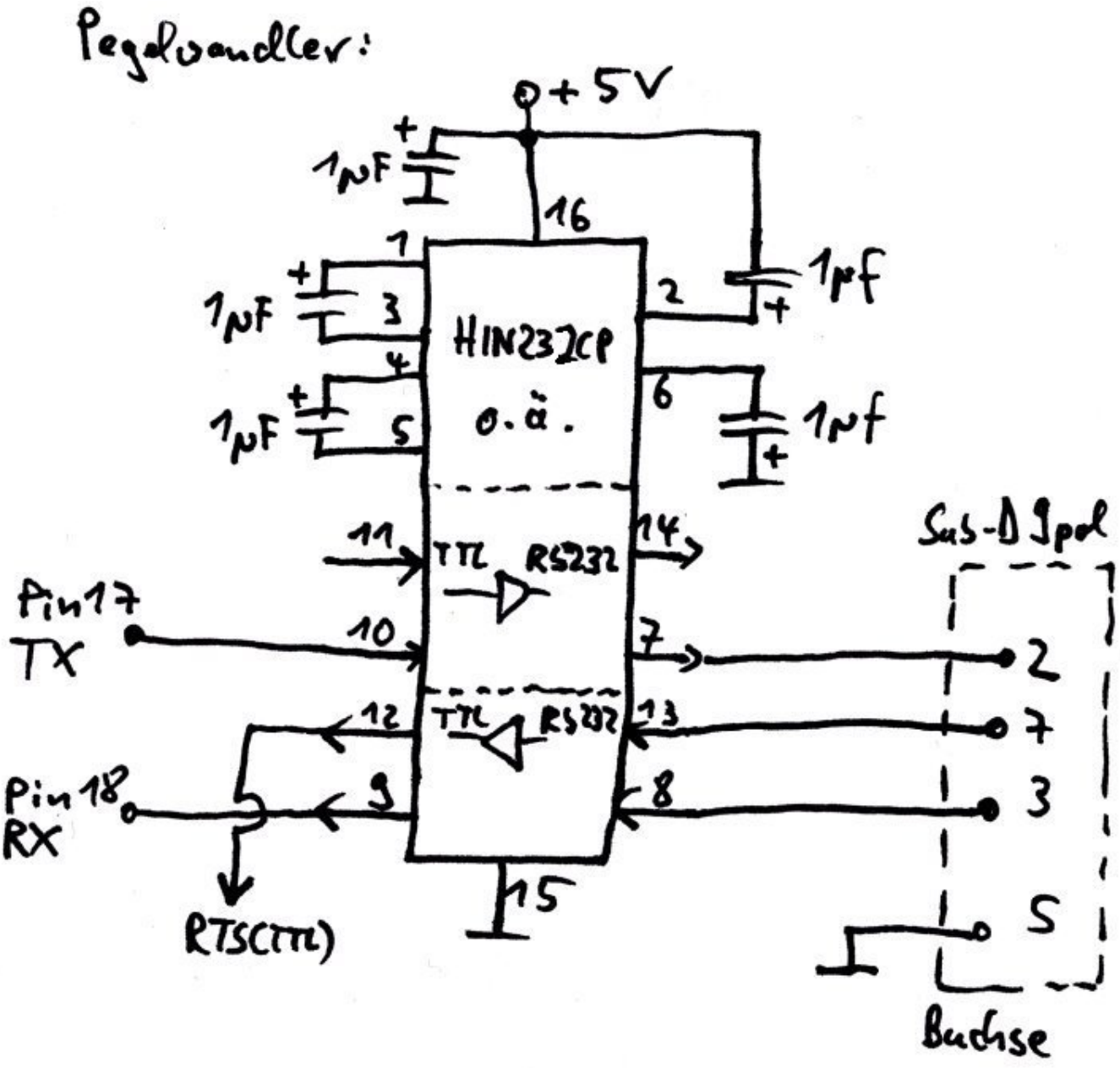
Die Reset-Schaltung:



Die Spannungsversorgung:



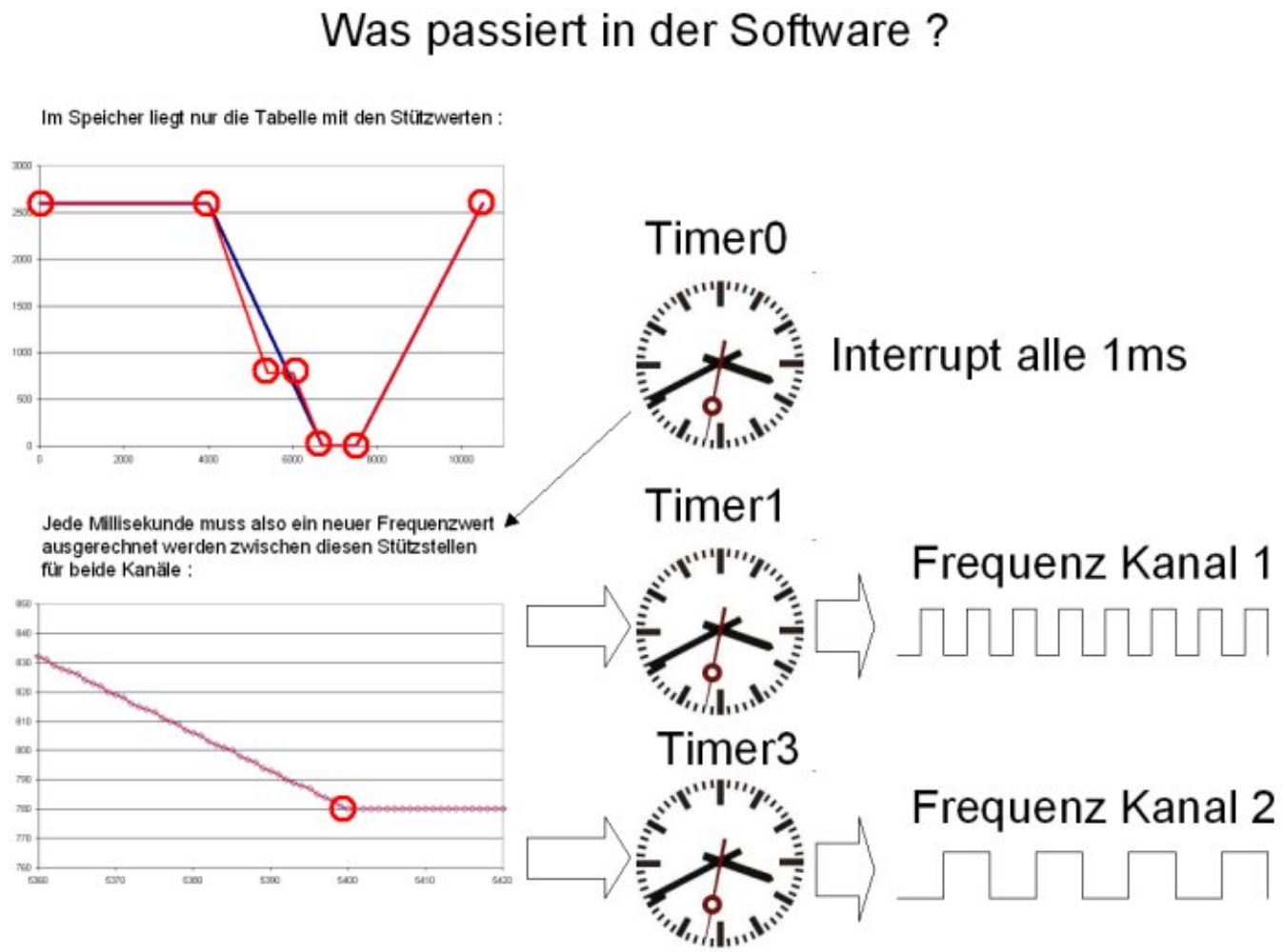
Der Pegelwandler (z.B. HIN232 oder MAX232 o.ä.):



Softwarebeschreibung :

Da ich gerne in Assembler programmiere ist das Projekt doch etwas aufwändiger aber auch anspruchsvoller geworden. Insgesamt ca. 1700 Zeilen Code. Von der 32k Speicher sind ca. 3k für das Programm und der Rest wird für die großen Tabellen verwendet. Alle mathematischen Routinen wollte

ich nur mit 8 Bit Registern umsetzen, da einige Routinen sicher mal wieder bei den nächsten 16Fxxx Projekten zum Einsatz kommen. Ansonsten wäre es möglich gewesen die Spezial-Features des 18F2550 zu nutzen. Insgesamt sind rund 100 Arbeitsstunden in das Projekt geflossen was sicher auch daran lag das es die ersten Gehversuche auf dem 18Fxxx sind.



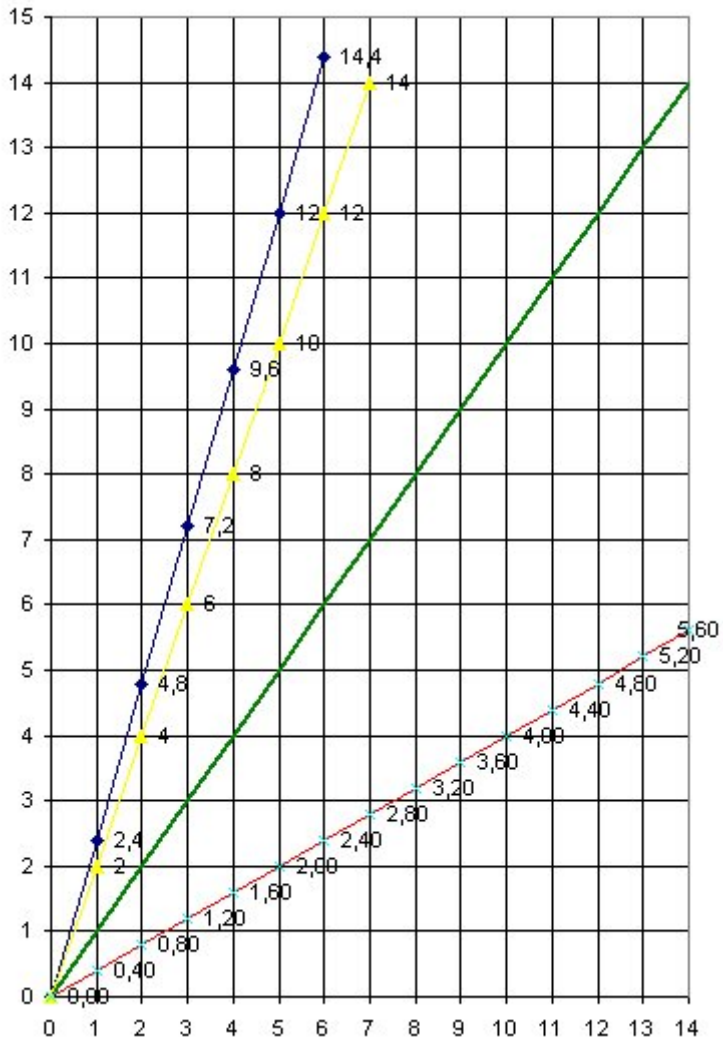
Es werden 3 von 4 Timern des PIC benutzt. Die Timer 1 und 3 werden benötigt für die Generierung der Frequenzen. Timer 0 ist der 1 Millisekunden-Timer. Alle Millisekunde wird also nachgeschaut ob ein neuer Stützpunkt vorhanden ist und falls nicht wird ein neuer Zwischenwert der Frequenz für beide Kanäle ausgerechnet. Da im Speicher nur Stützstellen gespeichert sind, müssen alle Zwischenwerte auf den Kurvenverläufen im PIC berechnet werden.

Und an dieser Stelle sieht man schon die größte Schwierigkeit : Wie berechnet man die Zwischenwerte einer Kurve, wenn nur die Anfangs und Endpunkte bekannt sind ? Auf dem Papier ein einfaches Problem, allerdings wollte ich ja auch keine Float-Berechnungen mit Division verwenden und das ganze noch in 16 Bit Register pressen !

Eine Lehrbuch-Lösung für solch ein Problem ist z.B. der Bresenham-Algorithmus wie er zum Zeichnen von Geraden auf einem Bildschirm verwendet wird
http://de.wikipedia.org/wiki/Bresenham-Algorithmus

Klingt alles sehr schön, lernt auch jeder Informatiker, taug hier in Praxis mal wieder nur bedingt ... Für alle Winkel größer 45 Grad schlug Bresenham und alle späteren Lehrbücher das Drehen und Spiegeln vor. Bedeutet z.B. das dann die Linie statt von unten links nach oben rechts umgekehrt gemalt wird. Das geht bei einem Plotter wie Bresenham ihn verwendete und auch auf einem Bildschirm

problemlos. Würde hier allerdings bedeuten das ich von der nächsten Stützstelle alle Schritte zurückrechnen müsste bis zum aktuellen Zeitwert. Unmöglich da zu Rechenintensiv. Also musste eine Idee her :



Beispiel für eine Korrektur der Daten :

Grün ist die erste Winkelhalbierende bis zu der der Bresenham funktioniet.

Blau ist die gewünschte Gerade mit einer Steigung von z.B. m=2,4.

Idee ist nun, das mit man dy durch dx teilt und das ganzzahlige Ergebnis notiert. Dies ist später die Schritthöhe p der Korrekturgerade.

Die neue Gerade die Bresenham berechnen muss ist also dy=dy-n*p.

Durch diesen Trick liegt die Gerade immer unter 45 Grad (hier die rote Linie)

Nach dem berechnen eines neuen Schrittes wird dann bei jedem Schritt von x wieder x*p dazuaddiert.

Diese additive Gerade ist hier gelb dargestellt.

Vorteil ist nun, das sich der Bresenham trotzdem verwenden läßt. Denn dieser kommt ohne Fließkomma aus und braucht auch keine Division - also Ideal für Assembler.

Das zweite große Problem ist, das die Timer welche die Frequenz erzeugen sollen nur mit einer Zeit geladen werden können. Da aber alle Millisekunde ein Frequenzschritt berechnet wird, muss dieser noch umgerechnet werden in Zeit. Dafür ist eine Division nötig. Durch diverse Tricks wird der Wertebereich der 16 Bit Register stark erhöht (siehe Code).

Die Teilung selber beruht auf einem sehr effizienten CORDIC-Algorithmus, der nur Addition, Subtraktion und Verschiebeoperationen braucht. Also auch ideal für Assembler.

Wenn immer Routinen aus den weiten des Internet verwendet wurden, ist dies mit Angabe der Quelle geschehen, so das hier nochmal nachgelesen werden kann.

Hier also die Software für den PIC und das Hexfile :

PIC Inhalt ArbiträrGenerator

arbgen_source.zip

Kompilieren ist möglich mit dem MPASMWIN.EXE (v5.14) aus dem kostenlosen Komplettpaket MPLAB von Microchip.

Der Bootloader :

Zum Einsatz kommt in diesem Projekt der sehr gute und kleine Bootloader von Claudiu Chiculita (v1.9.7) : http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm

Dieser musste leicht angepasst werden, damit bei 20 MHz Quarz und 48 MHz Ausgangstakt der PLL die serielle Kommunikation wieder bei 115200 Baud steht. Download :

Bootloader angepasst

tinybld_18f252_48mhz_intern.zip

Assembler-Tips für den 18F2550 :

Interessanterweise wird der 18F2550 von den meisten Leuten nur noch in C programmiert so das bei einigen Problemen keine Assembler-Beispiele im Internet zu finden waren. Die meisten Routinen hatte ich auf dem 16F876 entwickelt und die liefen erst mal nur noch sehr merkwürdig auf dem 18F2550....

1. Es wird kein Bank-Switching mehr benötigt. Einfach das Register mit Namen ansprechen und fertig.

2. Vorsicht bei der Verwendung von RLCF/RRCF sowie RETLW. Hier das Datenblatt genau durchlesen. Diese Befehle sind anders als bei PIC 16Fxxx.

3. Der Bootloader hatte zunächst nicht funktionert. Grund war einfach das ich die PLL Teiler anders belegen wollte. Deswegen musste der Bootloader geändert und neu kompiliert werden damit auch die serielle Kommunikation bei anderem Quarz/Teiler-Verhältnis wieder stimmt.

4. Als Beispiel was beim umsetzen schiefgehen kann : DECF hat nicht nur auf das Zero-Flag Einfluss wie bei 16Fxxx :

```
Original-Code vom 16F876:
Lb1     addwf   Ones,f
        decf    Tens,f
```

```
        btfss   STATUS,CARRY
        goto    Lb1
musste beim 18F2550 dann lauten :
Lb1     decf    Tens,f
        addwf   Ones,f
        btfss   STATUS,CARRY
        goto    Lb1
```

Warum ? Im Original-Code macht der uC eine Addition, dort wird das Carry gesetzt. Das folgende DECF ändert nur (falls nötig) das ZERO-Flag, überschreibt aber nicht das CARRY-Flag. Der 18F2550 ändert aber beim DECF das ZERO UND das CARRY, und schon ist eine Info überschrieben... Deswegen muss beim übernehmen von 16Fxxx Code peinlich genau darauf geachtet werden, welche Befehle wie benutzt werden und vor allem in welcher Reihenfolge !

5. In einer Interrupt-Routine nie direkt auf PORTx zugreifen um Ausgänge zu setzen oder einzulesen. Dies darf nur über die Latch Register LATx geschehen. Andernfalls gibt es sehr merkwürdige Effekte von „vergessener“ Portänderung.

6. Falls wie hier die Interrupt-Prioritäten vergeben werden muss vorher überlegt werden was wirklich High und was Low Prio hat. High unterbricht Low-Prio was bei Zeitkritischen Interrupt-Routinen wie hier die Frequenzausgabe fatal enden kann.

7. Unbedingt alle Errata-Sheets von Microchip lesen zu diesem Prozessor. Dort sind auch einige Hinweise zu den Timern zu finden.