Arbeitsprozessbericht

von Thorsten Kattanek / AEK3

Berlin, 13.01.2013

Projektname: **realSID**

- Einzelarbeit -

Thema:

Musik wie in der Homecomputer Ära ala Commodore64 in den 80igern, auf den heutigen Systemen zu produzieren, war mein Ziel für dieses Projekt. Dazu habe ich einen SID Chip „MOS-8580 R5“ Emulator programmiert (SIDClass), und einen Sequenzer (SequenzerClass) der diesen SID mit Daten füttern kann. Die SIDClass beinhaltet einen komplett emulierten „SID“, wobei ich dafür auch an realer Hardware versuche durchgeführt habe.
Diese SequenzerClass beinhaltet einen kompletten Sequenzer, der für sie Steuerung von 1-8 SID's (z.B realSID) konzipiert wurde. Die Klasse wird parallel zu den SID aufgerufen, also mit der selben „virtuellen“ Taktfrequenz wie die emulierten SID's.
Das ganze wird mit einer gut zu bedienenden Oberfläche verkleidet.

Ablauf:

Zum Anfang habe ich mich um die Programmierung der Emulation des MOS-8580 R5 (SID) gekümmert. Dazu habe ich viel im Internet recherchiert um Informationen über diesen Chip zu bekommen. Außerdem habe ich einen meiner alten C64er ans Oszilloskop gehangen und verschiedene Tests gemacht. Aus den ganzen Erkenntnissen entwickelte ich nun die SIDClass.

Nachdem die SID Emulation zu meiner Zufriedenheit lief, kam der nächste Schritt, welcher sich mit der Entwicklung der SequenzerClass beschäftigte. Dieser Sequenzer wurde genau für den SID Chip konzipiert. Man bedenke das der SID ein 8Bit Controller ist, somit ist auch ein gespeichertes Sequenzer File nur um die 60kb groß.

Nachdem eine Grundfunktionalität der SequenzerClass Gewährleistet war, fing ich an die Oberfläche des realSID zu entwickeln. Die GUI benutzt Qt als Framework (ähnlich .NET jedoch Plattform unabhängig !). Für ein besseren Workflow habe ich einige neue Steuerelemente von den Standardelementen abgeleitet, und meinen Bedürfnissen angepasst. zB. Das NotenEdit Steuerelement wurde von einem TextEdit abgeleitet, und nimmt nun nur Noten auf die gleich intern in NotenNummern gewandelt werden. Diese Nummer wird bei einer Änderung ans Hauptprogramm geschickt, ohne das sich das Hauptprogramm darum kümmern muss.

Nachdem die Oberfläche fertig war, habe ich noch weitere Funktionen zur SequenzerClass hinzugefügt, wie z.B. die Möglichkeit Effekte neben den Noten auszuführen. Effekte wie z.B. die Filtersteuerung des SID. Nach einigen Fehlerbeseitigungen, war das Projekt dann fertig, zumindest erst mal für den Abgabetermin. Das ganze möchte ich aber zukünftig, noch weiter ausbauen da ich noch sehr viele Ideen habe die ich aber aus Zeitgründen bis zu diesem Zeitpunkt nicht mehr mit einfließen lassen konnte.

Auftretende Probleme und dessen Lösung:

Wo ich mir an meisten den Kopf zerbrochen habe, war die Umsetzung des Filters des SID. Dieser ist obwohl es sich um ein IC handelt ein analoger Filter. Der SID ist quasi ein Zwitter aus Digitaler und Analoger Technik. Dank einer guten Seite im Netz bin ich an guten Pseudo-Code für Digitale Filter gekommen die genau das boten was ich suchte. Nun brauchte ich den Pseudo-Code nur in C++ Umsetzten, was dann wiederum kein Problem darstellte.

Meine Bewertung des Ergebnis:

Obwohl ich noch sehr, sehr viele Ideen habe was man noch einbauen könnte bin ich mit dem jetzigen Stand sehr zufrieden. Der SID klingt fast wie echt, und der Sequenzer läuft auch.
Ich habe das Programm unter Linux 32/64 Bit getestet, sowie unter WinXP und Win7. Unter allen Systemen lief es ohne Probleme und es gab keine Abstürze.
Einziger Kritikpunkt den ich mir gebe ist, das der Sequenzer ein Tick besser zu bedienen wäre, aber man braucht ja noch was für die zukünftigen Versionen.

Wie ist meine Bewertung des Prozesses ?

Da ich das ganze alleine ausgeheckt habe, gab es keine Probleme durch zuarbeite von anderen.
Ich habe von Anfang an versucht das ganze Sinnvoll abzuarbeiten, was mir am Ende auch gelungen ist.

Was kann ich für mich verbessern ?

Den Sourcecode während des Prozesses mehr zu Kommentieren, zu hat man am Ende mehr Zeit für andere Sachen :) .

Was würde ich das nächste mal anders machen ?

Nichts.

Würde ich das selbst gewählte Thema nochmals durchführen ?

Auf alle Fälle, hat mir viel Spaß bereitet. Des weiteren habe ich hier auch wieder eine Menge dazugelernt. Dieses Thema ist ja sehr vielseitig.

Quellen

Literatur:
C64 Handbuch :)
Das neue Commodore 64 Intern Buch
Das große Commodore 64 Buch

Internet:
http://www.dopeconnection.net/C64_SID.htm
http://www.sidmusic.org
http://www.schramm-software.de/tipps/zufallszahlen/

SID – „Sound Interface Device“ - Artikel
-- MOS 8580 R5 --

von Thorsten Kattanek
Berlin, 02.01.2013

Dies ist eine Zusammenfassung aller Erkenntnisse die ich durch Selbststudium oder aus dem Internet erworben habe. Dieser Artikel entstand parallel zur Entwicklung des realSID. Die Idee zu diesem Artikel kam dadurch, das ich selber als C64 Emulator Entwickler (Emu64) relativ wenig oder nur verstreute Information zu den SID gefunden habe. Ich habe versucht jedes Detail so genau wie möglich zu erklären.

Der Oszillator oder auch Wellengenerator

Diese Einheit erzeugt die verschiedenen Wellenformen des SID. Die Standard Wellen sind Dreieck, Sägezahn, Rechteck und Rauschen. Für die Erzeugung der Wellen werden noch 2 weitere Parameter benötigt. Zum einen die Frequenz die der Tonhöhe entspricht, und zum anderen wird noch für den Rechtecktyp die Pulsweite benötigt. Des weiteren gibt es noch das SYNC_BIT, RING_BIT und das TEST_BIT. Der Oszillator kümmert sich nicht um die Lautstärke, er gibt immer alles mit maximaler Amplitude aus. Schauen wir uns nun an wie im einzelnen der SID die verschiedenen Wellenformen erzeugt.

Das Herzstück des Oszillator ist ein 24 Bit Register welches pro Zyklus mit dem Inhalt des 16 Bit Frequenzregisters addiert wird. Wir nennen das Register ein mal FREQ_COUNTER. Die Addition wird wirklich in jedem Zyklus gemacht,wenn das TEST_BIT gelöscht ist.

Sägezahn ($10)

Die Sägezahnwelle erhält man, wenn man vom FREQ_COUNTER die oberen 12.Bit nimmt.
Also FREQ_COUNTER um 12.Bit nach rechts schiebt.

C++ Code: FREQ_COUNTER>>12;

Dreieck ($20)

Dreieck bekommt man wenn man bei gelöschten MSB des FREQ_COUNTER um 11.Bit nach rechts schiebt und mit 0xFFF AND verknüpft.
Bei gesetzten MSB um 11.Bit nach rechts schiebt invertiert und mit 0xFFF AND verknüpft.

```
C++ Code:   (FREQ_COUNTER & 0x800000)?(~FFREQ_COUNTER >>11) & 0xFFF:
                    (FREQ_COUNTER>>11) & 0xFFF;
```

Rechteck ($40)

Die Rechteckwelle wird erzeugt in den man die oberen 12.Bit von FREQ_COUNTER mit dem Wert aus dem Pulsweiten Register (12.Bit) vergleicht. Ist dieser größer-gleich dem Pulsweiten Register so wird 0xFFF ausgegeben ansonsten 0x000.

C++ Code: ((FREQ_COUNTER>>12) >= PULSE_REG)?0xFFF:0x000;

Rauschen ($80)

Rauschen besteht im Grunde aus Zufälligen Werten, da es aber in einem Computer keine „echten“

Zufallszahlen gibt, bedient man sich den sogenannten Pseudo-Zufallszahlen. Bewährt sind so genannte
linear rückgekoppelte Schieberegister, kurz LFSR. Das Prinzip des LFSR besteht darin, den Ausgang eines Schieberegisters auf den Eingang rückzukoppeln und mindestens eine EOR Operation durchzuführen.
Und so funktioniert das im SID (einfach mal angenommen :)).
Dazu wird ein 23 Bit SHIFT_REGISTER mit dem Startwert 0x7FFFF8 initialisiert. Auch bei einem Reset wird dieser Wert in das SHIFT_REGISTER geladen. Wenn das 19.Bit von FREQ_COUNTER von 0 nach 1 wechselt (Und nur dann!) dann wird das SHIFT_REGISTER neu berechnet. Dafür wird zuerst das Bit 0 für den neuen Inhalt des SHIFT_REGISTER berechnet. Dann wird das SHIFT_REGISTER um 1 nach links geschoben und das BIT0 rechts angehangen.

```
C++ Code:   BIT0 = ((SHIFT_REGISTER>>22) ^(SHIFT_REGISTER>>17)) & 0x01;
            SHIFT_REGISTER = (SHIFT_REGISTER<<1) & 0x7FFFFF | BIT0;
```

Den Ausgabewert erhält man nach folgenden Code:

```
C++ Code:   ((SHIFT_REGISTER&0x400000)>>11) | ((SHIFT_REGISTER&0x100000)>>10) |
            ((SHIFT_REGISTER&0x010000)>>7) | ((SHIFT_REGISTER&0x002000)>>5) |
            ((SHIFT_REGISTER&0x000800)>>4) | ((SHIFT_REGISTER&0x000080)>>1) |
            ((SHIFT_REGISTER&0x000010)<<1) |
            ((SHIFT_REGISTER&0x000004)<< 2);
```

Mischformen

Da die Mischformen nicht wie fälschlicher Weise AND verknüpft sind, sondern sich aus vielen digitalen und analogen Signalen sowie Seiteneffekten zusammensetzen, kann man diese so gut wie gar nicht berechnen. Deshalb geht man hier einen anderen Weg. Die Mischformen werden von einem Realen SID gesamplet. Da ich stolzer Besitzer eines Chameleons64 bin und diese eine REU beinhaltet bietet es
sich an diese für den Samplevorgang zu nutzen. Man kann mittels SID Register Nr.27 den aktuellen Wert des Oszillator 3 Ausgangs auslesen. Die REU hilft uns dabei mittels DMA die Daten auch für jeden Takt (Zyklus) zu bekommen. Wie dieser Vorgang nun im Einzelnen Funktioniert, werde ich an dieser Stelle nicht erläutern. Aber für interessierte gibt es im Anhang den Assemblerquellcode der für uns die Daten des SID in die REU schaufelt. Das Programm habe ich dafür selber geschrieben.

Hier ein Überblick wie die einzelnen Mischwellen aussehen:

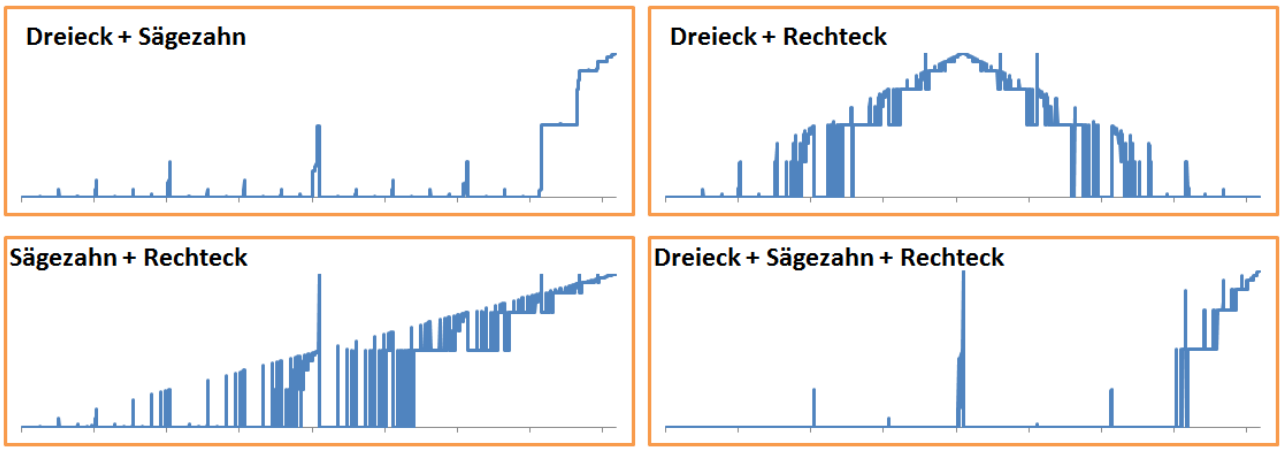


Abbildung 1: Mischwellen eines MOS-8580R5 (Die Pulsweite ist hier 0)

Mischung Rauschen mit anderen Wellenformen

Eine Mischung mit Rauschen führt zu einer Blockade des Oszillators und führt zu einer Ausgabe von Nullen.
Durch setzen und wieder löschen des TEST_BIT im Control Register kann diese Blockade aufgehoben werden.

Der Hüllkurvengenerator (Evelope)

Diese Einheit erzeugt den Lauststärkeverlauf einer Stimme. Es wird das Ausgangssignal des Oszillator mit dem Verlauft der Hüllkurve verknüpft. Das heißt bei 0 Werten erfolgt eine maximale Dämpfung und bei 0xFFF erfolgt keine Dämpfung. Um die Hüllkurve zu erzeugen benötigt man 5 Parameter, die da wären:
Attack (Anschlag), Decay (Abschwellen), Sustain (Haltepegel), Release (Abklingen) und das KEY_BIT.

! Die nun folgenden Schritte werden pro Zyklus einmal ausgeführt !

Das Herzstück hier ist ein 24 Bit Register. Dieses wir um 1 erhöht, und das wirklich in jedem Zyklus. Wir nennen das Register hier mal RATE_COUNTER.

Ist das höchste Bit von RATE_COUNTER nach einer Erhöhung gesetzt, so wird in diesem Zyklus RATE_COUNTER nochmals um eins erhöht und anschließen das höchste Bit wieder gelöscht.

Anschließend wird RATE_COUNTER mit einem weiteren Register verglichen dieses nennen wir mal RATE_PERIOD. Diese Register wird mit 16 verschieden Werten geladen, diese Werte werden aus einer Tabelle geladen. Es wird RATE_PERIOD mit jenem Tabellenwert geladen, mit dem Index aus einem der folgende SID Registern: Attack, Decay, Release. Welches Register als Indexgeber dient hängt davon ab ob das KeyBit gesetzt oder gelöscht wird oder ob ein Übergang von Attack nach Decay stattfand.

Hier die Werte in der Tabelle für RATE_PERIOD
PeriodeTable[16] = {9,32,63,95,149,220,267,313,392,977,1954,3126,3907,11720,19532,31251}

Ist nun RATE_COUNTER gleich RATE_PERIOD so wird RATE_COUNTER auf 0 gesetzt.

Anhang

Assemblerprogramm um die SID Wavedaten auf die REU zu bekommen

```
; ** SID-Wave-Capture **
; by Thorsten Kattanek
; Berlin, 29.12.2012

; Speichert OSC3 Ausgang auf REU für SID Emulation
; Es werden alle Waveformen gecaptured von 0-15
; $0000-0FFF WaveNr. $00
; $1000-1FFF WaveNr. $01
; $2000-2FFF WaveNr. $02
; $3000-3FFF WaveNr. $03 usw...
; Bei den Daten handelt es sich um die oberen 8Bit der Waveausgabe !
; Als Frequenzwert für das Register wird $1000 gesetzt

!to "sid-wave-capture.prg",cbm

sid = $d400
vic = $d000
reu = $df00

*=$0801

!byte $0c,$08,$dc,$07,$9e
!text "2064"
*=$0810

jmp $2000

        *= $2000      ;Assemble to $2000

       lda #$00       ;Bildschirm schwarz
       sta $d020
       sta $d021

       jsr $e544      ;Bildschirm löschen

       lda #<Ausgabe3  ;Startmeldung ausgeben
       sta $7c
       lda #>Ausgabe3
       sta $7d
       jsr txtout

       ;Testen ob eine REU vorhanden ist !!
       lda #$00        ;Null
       sta $df00       ;nach Register 0
       cmp $df00       ;noch drin?
       beq noreu       ;dann keine REU!

       lda reu
        and #16         ;check bit 4 for REU mem
        cmp #16         ;16 = 256Kbx1
        beq regcheck    ;yes, touch registers
        bne l1          ;no, 1700 ?
l1
       lda reu
        and #16         ;check bit 4 for REU mem
        cmp #0
        beq capture     ;reu 1700 found
        bne noreu       ;no ram-type, no reu, no fun ...
regcheck
       lda reu
        ldx #2
loop1
       txa
       sta $df00,x      ; write to registers 2-5
        inx
        cpx #5
        bne loop1
        ldx #02
loop2
       txa
        cmp $df00,x
        bne noreu
```

```
        inx
        cpx #5
        bne loop2
       jmp capture
noreu
       lda #<Ausgabe1 ;REU NOT FOUND ausgeben
       sta $7c
       lda #>Ausgabe1
       sta $7d
       jsr txtout
       rts

       ;Mit Aufzeichnung beginnen
capture
       lda #<Ausgabe4  ;Startmeldung ausgeben
       sta $7c
       lda #>Ausgabe4
       sta $7d
       jsr txtout
eingabe
       jsr $ffe4
       beq eingabe

       cmp #74
       beq ok
       rts
ok
       sei            ;Interrupt verhindern
       lda vic+$11    ;VIC Controlreg sichern
       sta d011

       lda #$00       ;VIC abschalten
       sta vic+$11

       lda #$00       ;Frequenz auf $1000 setzen
       ldx #$10
       sta sid+14
       stx sid+15

       sta sid+16     ;Pulsweite auf 0
       sta sid+17

       ldy #$00
loooop
       ; REU Installieren
       lda #$1b       ;SID Register OSC3 (Adresse im C64)
       ldx #$d4
       sta reu+2
       stx reu+3

       lda #$00       ;Startadresse in der REU
REU_ADD ldx #$00
       sta reu+4
       stx reu+5

       lda #$00
       sta reu+6      ;Bank 0 auswählen

       lda #$00       ;$1000 Byte übertragen
       ldx #$10
       sta reu+7
       stx reu+8

       lda #$00       ;Interrupts nicht erlauben
       sta reu+9

       lda #$80       ;Nur REU Adresse eröhen
       sta reu+10

       lda #$08       ;Reset Waveform
       sta sid+18

WAV_ADD lda #$00       ;Wird immer erhöht
       sta sid+18

       ;Warten auf Anfang nächte Periode
       ldx #$f0
```

```
wait
       nop
       nop
       nop
       nop
       nop
       nop
       dex
       bne wait

       beq wait1      ;Nur wegen 3 Zyklen
wait1  nop
       nop
       nop

       ldx #%10010000 ;Übertragung von C64 nach Reu

       stx reu+1      ;Übertragung starten

       lda REU_ADD+1
       clc
       adc #$10
       sta REU_ADD+1
       sta WAV_ADD+1

       iny
       cpy #$10
       bne loooop

       lda d011       ;VIC anschalten
       sta vic+$11

       lda #<Ausgabe2 ;REU NOT FOUND ausgeben
       sta $7c
       lda #>Ausgabe2
       sta $7d
       jsr txtout

       cli            ;Interrupt wieder freigeben
       rts

;/// Textausgabe: Text >> $7c/7d mit NUll am Ende !
txtout
       pha
       tya
       pha
       ldy #00
txtout_1
       lda ($7c),y
       beq txtout_end
       jsr $ffd2

       inc $7c
       bne txtout_2
       inc $7d
txtout_2
       jmp txtout_1
txtout_end
       pla
       tay
       pla
       rts

d011
!byte 0

Ausgabe1
!text  151,"REU NOT FOUND !",13,13,0
Ausgabe2
!text  151,13,"SAMPLE VORGANG IST BEENDET.",13,0
Ausgabe3
!text  152,">SID WAVE CAPTURE VON THORSTEN KATTANEK<",13,13,0
Ausgabe4
!text  151,"ACHTUNG! ES WERDEN ALLE DATEN AUF DER",13,"REU IN BANK 0 GELOESCHT !",13,13,"MOECHTEN
SIE FORTFAHREN J/N ?",13,0
```

Hauptklassen Übersicht

Das Projekt besteht noch aus weiteren als hier beschrieben Klassen, diese werden aber lediglich von den beiden hier aufgezählten Klassen benutzt und sind so zu sagen Unterklassen. Für Entwickler die auf dies SIDClass und SquenzerClass zurückgreifen möchten ist ein wissen über die Unterklassen nicht erforderlich !

Eine Interaktion mit den Klassen erfolgt ausschließlich über die hier angezeigten Funktionen, der Rest ist „private“ und so für die Außenwelt nicht zugänglich.

SIDClass

Alle public Funktionen im Überblick:

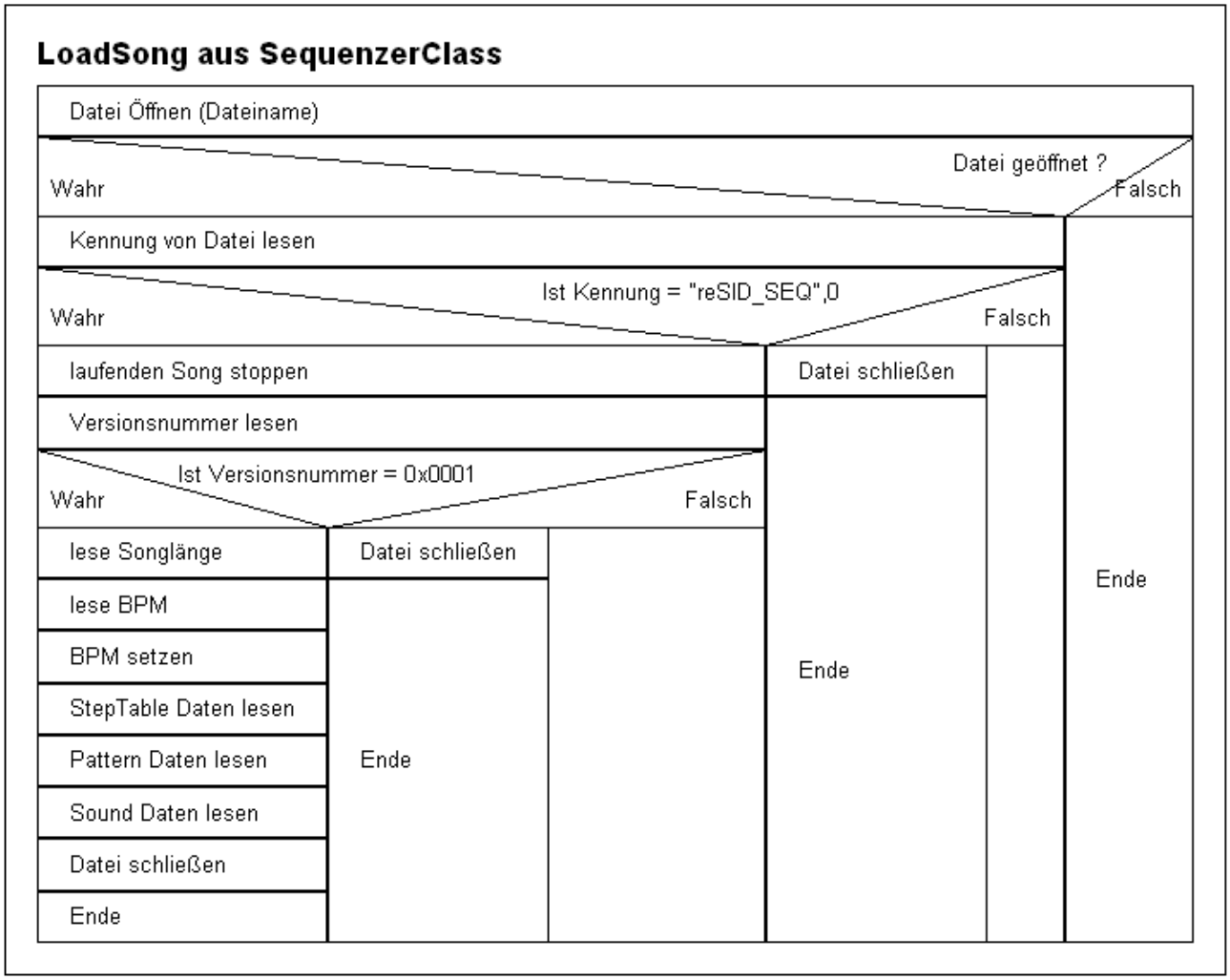
```
SIDClass()                              // Construktor
~SIDClass()                             // Destructor
void ResetSoundPufferPos(void)          // Zeiger innerhalb des Soundpuffers wird auf Null
                                           gesetzt
void OneCycle(void)                     // Wird pro Zyklus aufgerufen
float* GetSoundPuffer(void)             // Holt den Zeiger des Soundpuffers
int GetSoundPufferPos(void)             // Holt die Position innerhalb des Soundpuffers
void WriteIO(ushort adr, uchar wert)    // Schreibt einen Wert in eine SID Register (adr)
bool OpenSIDDump(char* filename)        // Es wird ein SIDDump File geladen (Emu64)
void PlaySIDDump(void)                  // SID Dump abspielen
void StopSIDDump(void)                  // SID Dump abspielen beenden
```

SequenzerClass

Alle public Funktionen im Überblick:

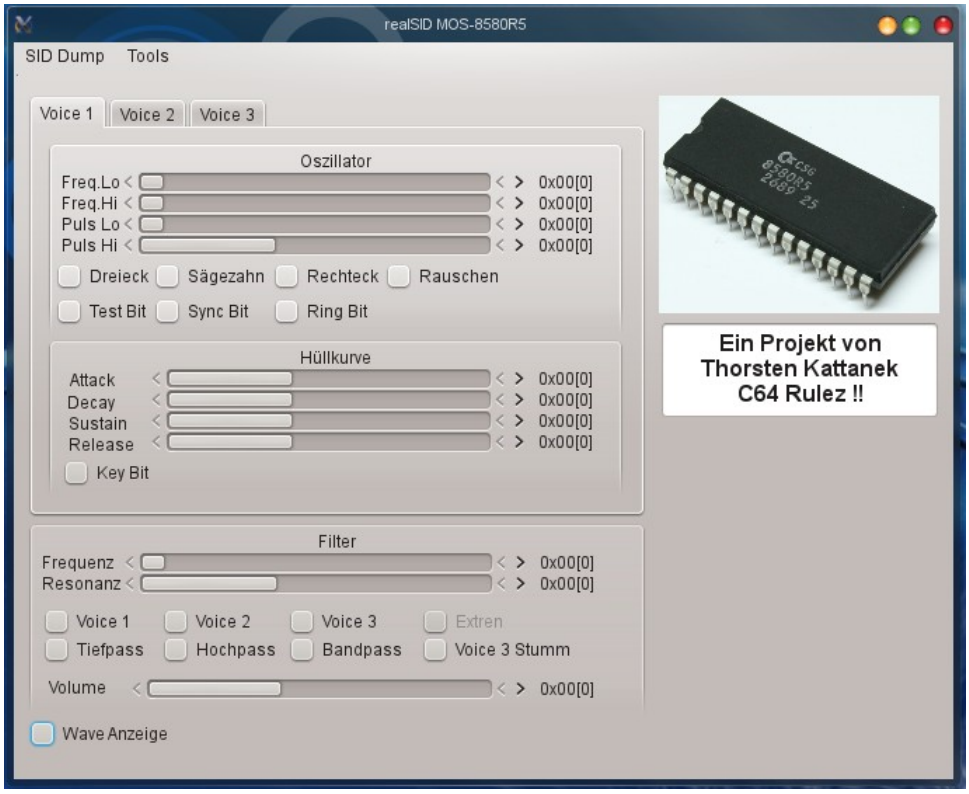
```
SequenzerClass()                        // Construktor
~SequenzerClass()                       // Destructor
unsigned short OneCycle(void)           // Wird parallel zum SID pro Zyklus aufgerufen
void SetBPM(int bpm)                    // Abspielgeschwindigkeit in BPM setzen
int GetBPM(void)                        // Abspielgeschwindigkeit auslesen
void SetSongLength(int length)          // Länge des Songs setzen
int GetSongLength(void)                 // Songlänge holen
bool LoadSong(char* filename)           // Ein Sequenzersong wird geladen
bool SaveSong(char* filename)           // Ein Sequenzersong wird gespeichert
int GetAktStepPos(void)                 // Aktuelle Step Position holen
int GetAktPatternPos(void)              // Aktuelle Position innerhalb des Pattern holen
PATTERN* GetPattenPointer(int nr)       // Zeiger von PATTERN[nr] holen
SOUND* GetSoundPointer(int nr)          // Zeiger von SOUND[nr] holen
STEP* GetStepTablePointer(void)         // Zeiger von der StepTable holen
void ClearSong(void)                    // Songspeicher wird gelöscht
void Play(void)                         // Song wird abgespielt
void Stop(void)                         // Song wird angehalten
```

Als Beispiel für ein Strukturgramm habe ich die Load Fuktion aus der SequenzerClass gewählt.



Anleitung für realSID

von Thorsten Kattanek
Berlin 13.1.2013



Inhaltsverzeichnis

1.0 Allgemeines

realSID emuliert zu 99% einen SID-MOS 8580 mit all seinen Funktionen. Da es sich um ein IC handelt lassen sich im Hauptfenster nur alle Register des SID verändern und man hört und sieht sofort die Auswirkung. Um auch Sinnvolle Eingaben zu machen ist es unabdingbar etwas über den SID zu wissen. Auch im miniSequenzer ist es vorteilhaft sich vorher etwas theoretisch mit dem SID zu beschäftigen. Dazu folgt im Anschluss ein kleiner Crashkurs in Sachen SID.

2.0 SID Crashkurs

Der SID besitzt insgesamt 3 Stimmen (voices). Diese können unabhängig von einander programmiert werden. Jede Stimme besitzt einen Wellengenerator (OSC) und eine Hüllkurvengenerator (ENV) sowie einige Steuerbits. Der OSC kann 4 verschiedene Wellenformen erzeugen die da wären: Dreieck, Sägezahn, Rechteck und Rauschen.
Alle Wellenformen können auch gemischt werden, jedoch keine mit Rauschen !
Damit der OSC eine Welle erzeugen kann benötigt er zusätzlich noch eine Frequenz. Diese wird als 16Bit Wert übergeben, jedoch ist dieser Wert nicht gleich die Frequenz. Diese wird mit folgender Formel berechnet: **FREQUENZ * 2^24 / 985248 Hz**

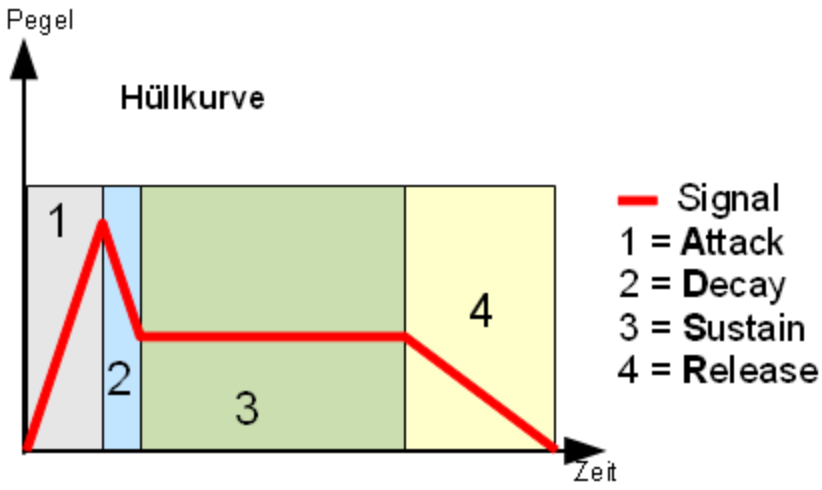
Um einen Ton auch zu hören muss man den ENV programmieren und starten. Die Hüllkurve beschreibt den zeitlichen verlauf des Laustärkepegels der Stimme.
Dazu besitzt der ENV vier Register: Attack, Decay, Sustain, Release.

Attack = Die Zeit vom Anschlag bis zu vollen Lautstärke (Das Key Bit wird gesetzt)
(Anschlag)

Decay = Die Zeit nachdem die volle Lautstärke erreicht wurde bis zum Sustainpegel
(Abschwellen)

Sustain = Pegel die die Decaypahse erreichen soll
(Halten)

Release = Die Zeit vom Löschen des Key Bit bis die Laustärke 0 ist
(Abklingen)



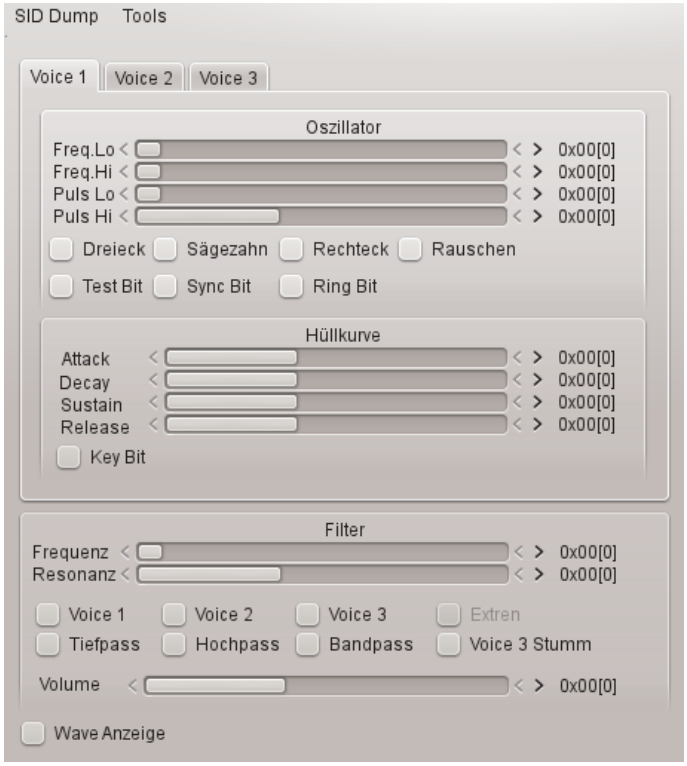
Des weiteren Gibt es noch einen programmierbaren Filter, worüber alle Stimmen geleitet werden

können wenn Sie möchten. Parameter für den Filter sind die Filterfrequenz, die Resonanz und wie gefiltert werden soll (Tiefpass, Hochpass und Bandpass).

Weiter gibt es noch ein Laustärkeregister welches die Gesamtlaustärke einstellt.

Viele weitere nützliche Informationen finden Sie unter Google → „c64 & sid“

3.0 realSID Hauptfenster



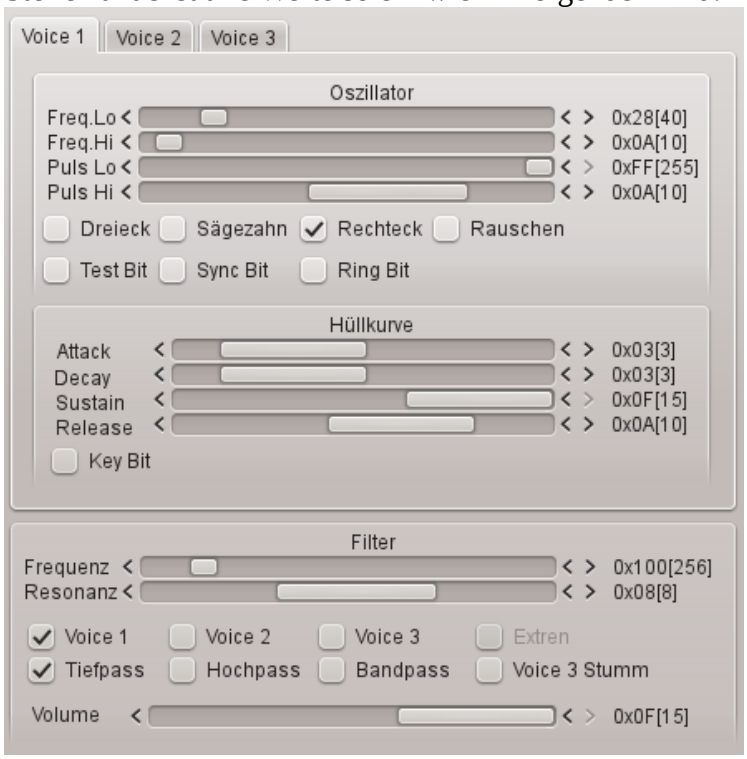
Über dem Menüpunkt SID Dump kann man mit dem Emu64 erzeugte SID Dumps laden und abspielen. Diese Funktion hatte ich eingebaut, um den Emulierten SID mit einem Originalen SID vergleichen zu können. Da diese Funktion ganz nett ist habe ich diese auch drinnen gelassen. SID Dump Files habe ich dem Projekt hinzugefügt, damit ihr auch diese Funktion Testen könnt.

Über Tools gelangen Sie zu einem sehr kleinen Sequenzer der es erlaubt den SID zu programmieren. Am unteren Ende des Fensters ist eine Chaeckbox die es erlaubt eine Wave Anzeige zuzuschalten. Somit kann man auch sehen was man hört ;)

Es werden alle SID Register mittels Schieberegler oder Checkboxen verändert.
Die Funktion wurde im Crashkurs erklärt.

3.1 Ein kleines Tutorial

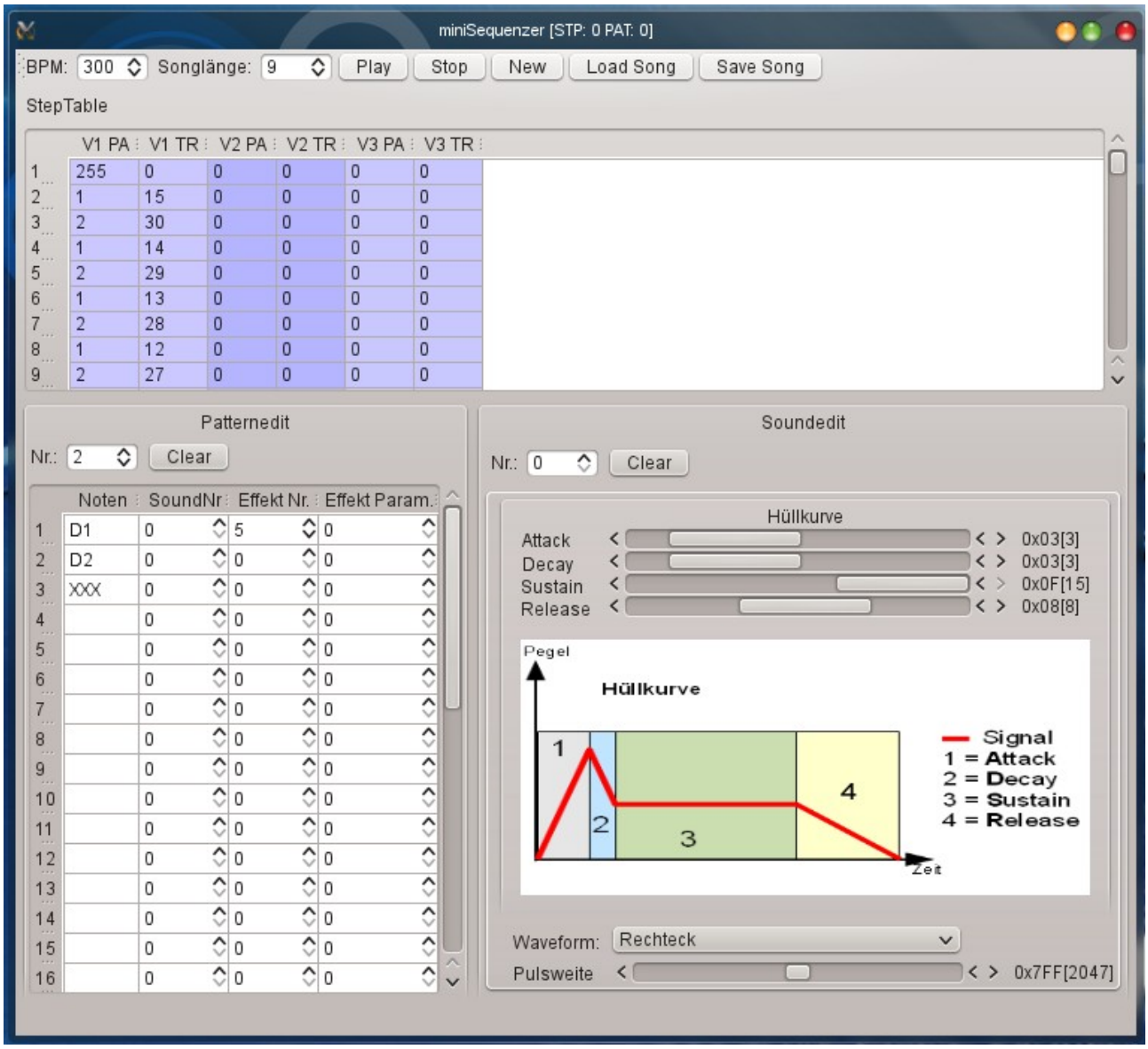
Stelle zunächst alle Werte so ein wie im folgenden Bild:



Haben Sie das getan? Na dann klicken sie nun auf die CheckBox Key Bit.
Der Ton wird nun solange zu hören sein bis Sie wieder die CheckBox Key Bit auf 0 setzen.

Viel Spaß nun mit dem rumprobieren :)

4.0 Der miniSequenzer



Mit dem miniSequenzer ist es möglich den emulierten SID zu programmieren. Es lässt sich immer nur ein Song pro Sitzung bearbeiten. Die Songs können gespeichert und zu einem späteren Zeitpunkt auch wieder geladen werden.

4.1 Fensteraufbau

Das Fenster ist in 4 Bereiche geteilt. Oben befindet sich die Toolbar die auch verschoben werden kann. Diese Toolbar enthält alle wichtigen Elemente zum Steuern des Songs.
Gleich darunter befindet sich die StepTable welche die Reihenfolge der abzuspielenden Pattern festlegt. Dann gibt es den Patterneditor mit diesen werden die Noten und Soundnummern innerhalb eines Pattern editiert. Und zu guter Letzt gibt es noch den Soundedit, dieser dient zum Festlegen einer Klangfarbe. Im folgenden werde ich alle Teile einzeln beschreiben.

4.2 Toolbar

BPM: Hier geben Sie die Geschwindigkeit, mit welcher der Song abgespielt werden soll, ein. BPM ist die Abkürzung für Beates per Minute.
Es können Werte von 0-999 eingegeben werden wobei 0, gleich Stop ist.

Songlänge: Hier geben Sie die Anzahl der Steps an die gespielt werden sollen. z.B. 1, so wird nur ein Step in der StepTable und dessen dazugehörigen Pattern abgespielt.
Die Maximale Länge beträgt 255.

Play: Fängt an den Song abzuspielen, es muss mindestens eine Songlänge von 1 sein.
Ist der Song am Ende angelangt, fängt er von vorne an zu spielen.

Stop: Hält das Abspielen des Songs an.

New: Alle Einträge werden gelöscht !

Load Song: Lädt ein zuvor gesicherten Song, und überschreibt den bisherigen im Speicher.
Achtung ! Es kommt keine Warnung.

Save Song: Speichert den aktuellen Song auf Festplatte oder ein anderes Medium.

4.3 StepTable

StepTable

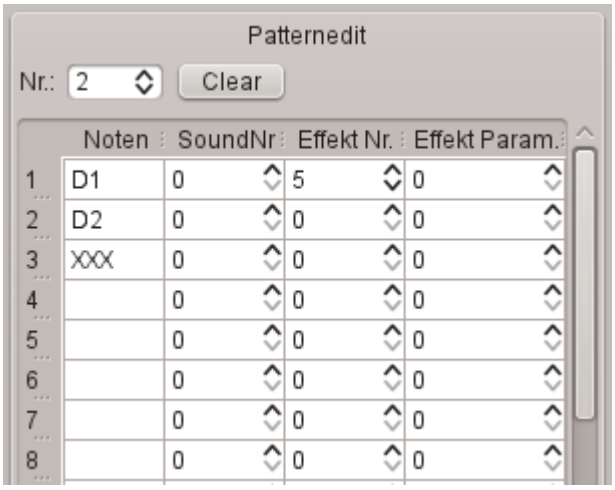
	V1 PA	V1 TR	V2 PA	V2 TR	V3 PA	V3 TR
1	255	0	0	0	0	0
2	1	15	0	0	0	0
3	2	30	0	0	0	0
4	1	14	0	0	0	0
5	2	29	0	0	0	0
6	1	13	0	0	0	0
7	2	28	0	0	0	0
8	1	12	0	0	0	0
9	2	27	0	0	0	0

Die StepTable ist eine Liste die den Ablauf des Songs steuert. Beim abspielen wird der Song von oben nach unten abgespielt. Die Tabelle wird in sechs Spalten geteilt.
Die Spalten geben die Stimmen (Voices) an und zwar 3 Stück. Diese sind Farblich etwas getrennt. Jede Voice ist nochmals in 2 Spalten geteilt, die erste ist die Pattern Nummer (PA) und die zweite der Transposewert (TR).

Als Patternwert (PA) können Werte von 0 bis 255 angegeben werden. Steht in V1 PA z.B. eine 2 so wird das Pattern mit der Nummer 2 auf der Stimme 1 abgespielt. Wird die Nummer 2 auch in V3 PA eingetragen so wird dieser Pattern auch auf Stimme 3 abgespielt.

Der Transposewert (TR) erlaubt es ein Pattern mit einer veränderten Tonhöhe wie im Pattern selbst abzuspielen. Dazu wird zu jeder Note im Pattern beim abspielen der Transposewert hinzu addiert. Der Transposewert kann von -63 bis 64 eingestellt werden.

4.4 Patternedit



Im Patternedot werden die Noten eingetragen die nacheinander abgespielt werden sollen.
Mit Nr.: wählen Sie den zu bearbeitenden Pattern aus.
Es können Pattern von 0-255 verwendet werden. Der Clear Button löscht den Kompletten Pattern.
Ein Pattern ist 32 Beates lang, es kann aber auch vorher beendet werden wenn nötig.

Ein Pattern ist in 4 Spalten unterteilt.

Noten: enthält die zu spielenden Noten von C0 – H6 (z.B C#1;D1;A6)
In diesem Feld können nur gültige Noten eingegeben werden, und ein Sonderzeichen „x“. Ein mit XXX gekennzeichneter Beat wird nicht mehr abgespielt und der Pattern wird dort beendet.
Ein leeres Notenfeld bedeutet das hier ein leer Takt ist.

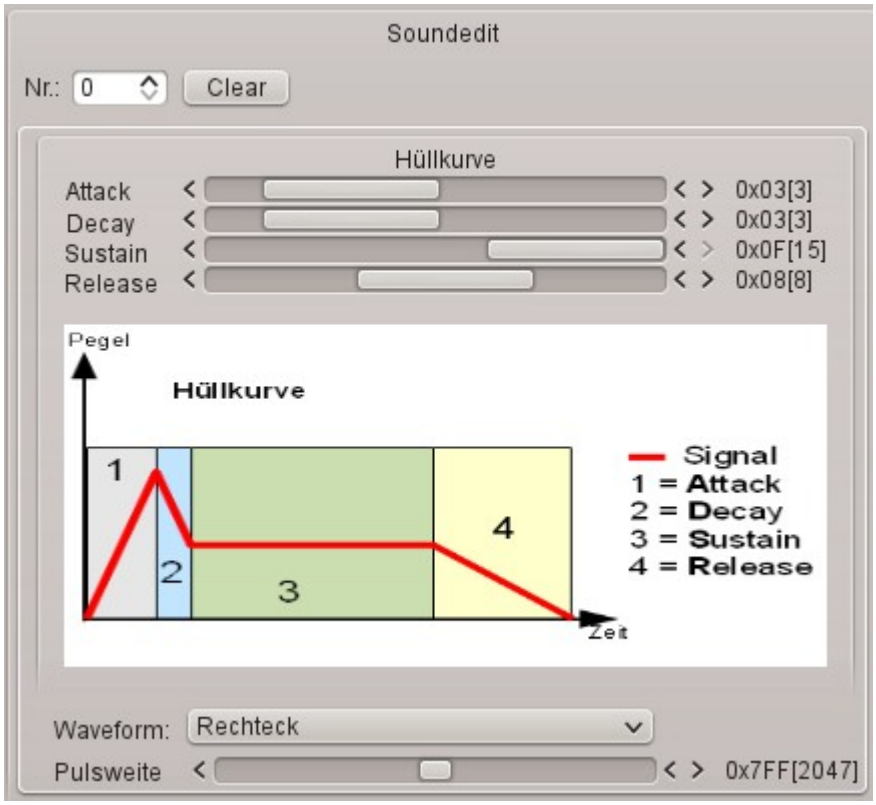
SoundNr: Hier wird die Nummer des Sounds eingegeben mit welchem diese Note abgespielt werden soll. Es können Sounds von 0-255 eingegeben werden.

Effekt Nr.: Hier kann eine Effektnummer eingegeben werden siehe Tabelle unten.

Effekt Param: Hier wird der Parameter für den Effekt angegeben wenn einer benötigt wird. Ansonsten wird der Wert ignoriert.

Nr.	Beschreibung	Parameter
00	Keine Auswirkung	
01	Setzt die Gesamtlautstärke	0 - 15
02	Setzen der Filterfrequenz	0 - 2047
03	Setzt die Resonanz	0 - 15
04	Schaltet die angegebene Stimmen über den Filter (0 = keine Stimme)	0 – 3 Bit/Stimme
05	Nimmt die angegebene Stimmen wieder vom Filter (0 = keine Stimme)	0 – 3 Bit/Stimme
06	Setzt den Filtertyp (0=Aus 1=Tiefpass 2=Hochpass 3=Bandpass)	0-3

4.5 Soundedit



Hier können Sie den Ton die richtige Farbe geben. Es wird der Lautstärkeverlauf oder auch die Hüllkurve eingestellt, sowie die Wellenform. Wird eine Wellenform gewählt die auch ein Rechteck enthält, so kann man mit der Pulsweite das Tastverhältnis dieser einstellen.

Es können 0 – 255 Sounds verwaltet werden. Mit Clear wird der auswählte Sound auf Standard Werte eingestellt. Diese Sound ist ein weicher Dreieckston.

Dies sollten nun alle Funktionen des realSID sein. Ich wünsche ihnen viel Spaß beim rum experimentieren mit dieser Software.