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

von Thorsten Kattanek
Berlin, 29.12.2012

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. Dieser Artikel soll nun alles zusammenfassen, damit neue Entwickler es leichter haben an diese Informationen zu gelangen. Dieser Artikel kann frei weitergeben werden, jedoch Änderungen bitte nur mit meinem Einverständnis. Ich habe versucht jedes Detail so genau wie möglich zu erklären. Fehler oder Falschaussagen kann ich nicht ausschließen und übernehme dafür auch keine Haftung. Ich freue mich aber über jeden Hinweis auf Fehler oder Verbesserungsvorschläge.

Eine aktuelle Version dieses Artikels, sowie das realSID Projekt findest du öffentlich auf GitHub:
https://github.com/tkattanek/realSID

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. 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. Die Addition wird wirklich in jedem Zyklus gemacht. Wir nennen das Register ein mal FREQ_COUNTER.

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:

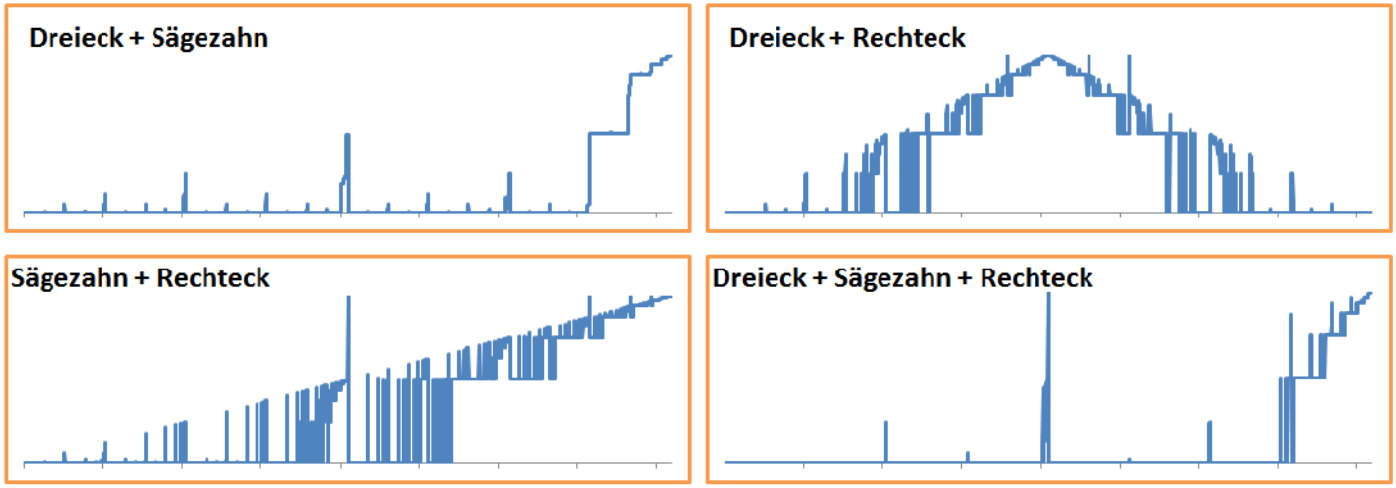

Abbildung 1: Mischwellen eines MOS-8580R5 (Die Pulsweite ist hier 0)

Mischung aus Dreieck und Sägezahn (0x30)

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 Bits im Control Register kann diese Blockade aufgehoben werden.

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/

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
```