

Der PC-Lautsprecher

1. Theorie:

Wir werden jetzt zuerst ein bisschen Theorie machen und uns danach mit der Praxis beschäftigen. Der PC Lautsprecher wird über den *Timer-Baustein* angesteuert, der als Zeitgeber und Zähler fungiert. Der Timer-Baustein besteht intern aus 3 verschiedenen unabhängigen *Timerkanälen*, deren *I/O-Ports* die Adressen 40h (64d), 41h (65d) und 42 (66d) besitzen, sowie einem *Kommandoregister* mit der Adresse 43h (67d). Die I/O-Ports sind 8 Bit breit. Die ersten beiden Kanäle werden für die Systemzeit und für das *Refreshing* des RAMs verwendet. Der dritte Kanal ist mit dem *PC-Lautsprecher* verbunden. Veränderungen an den ersten beiden Kanälen könnten verhängnisvolle Folgen haben, weshalb im Code immer exakt geprüft werden sollte, ob die *Port-Adressen* stimmen.

Jeder dieser 3 Kanäle besitzt ein *Latch-Register* und ein *Counter-Register*, die beide 16 Bit breit sind, zwei *Signaleingänge* (CLOCK und GATE) sowie einen *Signalausgang* (OUT).

Jetzt zur Funktionsweise: Am Eingang CLOCK liegt eine konstante Frequenz von 1,19318 Mhz an. In das Latch-Register wird eine Zahl, der so genannte *Teiler*, geschrieben. Dieser Wert wird in das Counterregister übertragen und mit jedem Impuls am Eingang CLOCK um 1 dekrementiert. Wenn das Counter-Register den Wert 0 erreicht, wird ein Puls über Ausgang OUT geschickt. Nacheinander entsteht so bei Wiederholung eine Frequenz von $1193180/\text{Teiler}$.

Das Kommandoregister wird zum steuern dieses Bausteins mit dem *Kommandobyte* geladen, welches z.B. einstellt, welche Betriebsart gewählt wurde. Das Kommandobyte ist so aufgebaut:



Die Betriebsart bestimmt die Länge der Pulsfolge und, ob sie sich automatisch wiederholt usw. Der Code 011 bestimmt, dass eine Pulsfolge erzeugt wird, die zum Erzeugen von Tönen nützlich ist. Da die Ports ja nur 8 Bit breit sind, das Latch-Register aber 16 Bit, wird im Kommandobyte auch noch festgelegt, ob nur das High-Byte, nur das Low-Byte oder beide hintereinander (zuerst das Low-Byte, dann das High-Byte) bearbeitet werden sollen. Das 0.Bit legt fest, ob die Zahl im Kommandobyte binär- oder BCD-codiert lesen soll.

Hier ein Beispiel:

Das Kommandobyte 0xB6 = 10110110 fest, dass:

- Timer-Kanal 2 programmiert werden soll,
- erst das Low-und dann das High-Byte bearbeitet werden soll,
- eine symetrische Pulsfolge erzeugt werden soll,
- das Kommandobyte binär zui interpretieren ist.

Die Programmierung des Timer-Bausteins erfolgt in 2 Schritten:

1. Kommandobyte in den *Kommandoport* schreiben, und dadurch die Anweisungen verteilen
2. Teiler in den Port des ausgewählten Timer-Kanals schreiben

So jetzt könnten wir zwar Töne ausgeben, aber wir müssen dazu zuerst noch den Lautsprecher

anstellen (und später wieder ausstellen). Das ist ganz schlaue geregelt. Der Lautsprecher ist über ein UND-Gatter, das heißt, dass am Ende nur 1 raus kommt, wenn beide Eingänge 1 haben, mit dem OUT-Ausgang des 2. Timer-Kanals und dem Ausgang des 1. Bits des Port B (0x61) im PPI (=Programmable Peripheral Interface) verbunden. Bit 0 des Port B im PPI ist mit dem GATE-Eingang des PPI-Ports B verbunden. Das heißt, es wird nur etwas ausgegeben, wenn das 1. Bit des Port B im PPI auf 1 und das 0. Bit auf 0 steht. Wenn beide Bits gelöscht sind, wird der Lautsprecher wieder ausgeschaltet. Das ist eigentlich schon die gesamte Theorie, kommen wir jetzt zur Praxis!

2. Praxis:

Wir werden jetzt ein Paar Funktionen für die Programmiersprache C zusammen entwickeln. Nebenher schreiben wir die Funktionen dann auch noch in Assembler.

Also, fangen wir an, ich denke in C dürfte jeder die Funktionen „inp()“ und „outp()“ schon für sein OS implementiert haben, weshalb ich darauf baue.

Wir wissen, wir schalten den Lautsprecher an, indem wir die Zahl 10b an Port 0x61 schreiben.

Daraus machen wir schnell eine Funktion:

```
void switch_on()
{
    outp(0x61, inp(0x61) | 3);
}
```

Okay. In Assembler baue ich das schon alles mit in die _sound-Funktion ein.

So, an können wir ihn machen, nur aus noch net, aber wir wissen, wenn beide Bits in Port 0x61 auf 0 stehen, dann ist der Lautsprecher aus. Das geht also ganz leicht:

```
void switch_off()
{
    outp(0x61, inp(0x61) &~3);
}
```

Und in Assembler:

```
_switch_off:
    IN AL, 61h
    AND AL, 11111100b
    OUT 61h, AL
    RET
```

Tada. An und aus können wir ihn schon machen, aber das Herzstück kommt noch! Einen Ton ausgeben. Das geht mit folgendem, diesmal zuerst in Assembler ;-):

```
_sound:
    MOV AX, 34DDH
    MOV DX, 0012H
    CMP DX, BX
    JNC DONE1
    DIV BX
    MOV BX, AX
    IN AL, 61H
    TEST AL, 3
    JNZ A99
    OR AL, 3
    OUT 61H, AL
    MOV AL, 0B6H
    OUT 43H, AL
A99:
    MOV AL, BL
    OUT 42H, AL
    MOV AL, BH
    OUT 42H, AL
DONE1:
```

RET

So, jetzt in C:

```
void sound(unsigned frequenz)
{
    unsigned teiler;
    teiler = 1193180L/frequenz;
    outp(0x43,0xB6);
    outp(0x42,teiler&0xFF);
    outp(0x42,teiler >> 8);
    switch_on()
    //wenn der Sound aufhören soll, muss die Funktion
    //switch_off aufgerufen werden.
}
```

So, geschafft! Das waren die Geheimnisse des PC-Lautsprechers! Eins muss beim Nachmachen noch beachtet werden: Es muss zwischen den Tönen eine Funktion benutzt werden, die das Programm anhält, denn sonst werden die Töne viel zu schnell abgespielt, und man hört nix. Ich hoffe ich konnte alles so erklären, dass es verständlich war. Und die ein oder anderen Unklarheiten durch den letzten Praxisteil klären. Ich bedanke mich bei meinem C-Buch, aus dem ich die Informationen hab, bei ASHLEY4 aus dem FlatAssemblerForum (<http://board.flatassembler.net/>) für die Hilfe beim Code und bei allen anderen, die mir geholfen haben. Und ich wünsche allen noch viel Spaß beim OS-Coding.

Joachim Neu alias „joachim_neu“

Bei Anregungen, Vorschlägen und oder Fehlern bitte bei joachim_neu@web.de melden. Danke!