Dreiklanggong mit dem 8xC752

Daniel Rohner, TU-Wien, M&R-Systems DSK-407: GONG.C, PWMGEN.A51

In diesem Artikel:

- der Mikrocontroller 8xC752 (Philips)
- C- und Assemblerprogrammierung mit KEIL- Entwicklungsumgebung
- Compilereinstellungen für den 'C752

Wer kennt Ihn nicht, den Dreiklanggong-Baustein SAB600 von Siemens? Dieser Baustein verwendet drei 4bit-D/A-Wandler pro Ton für den Abklingvorgang. Die drei Töne werden nacheinander eingeschaltet und summiert. Der Ausgang kann einen 8 Ohm-Lautsprecher mit ca. 0,16W treiben. Will man größere Lautstärken erzielen, müßte ein Audioverstärker nachgeschaltet werden.

Der Mikrocontroller-Gong hat einen großen Vorteil: Er verwendet zur Amplitudenausgabe eine pulsbreitenmodulierte Spannung. Dadurch kann der Lautsprecher durch einen einfachen, geschalteten Verstärker betrieben werden - also keine analoge Endstufe, der Verstärker wird praktisch nicht warm.

Der 8xC752 von Philips

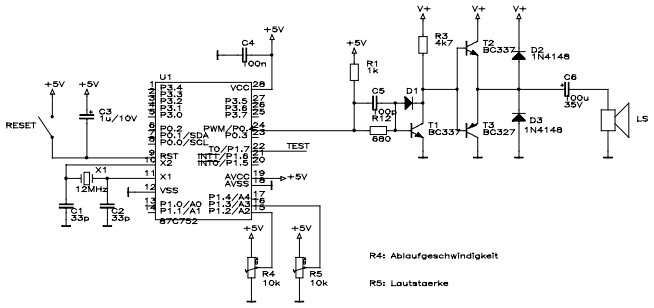
Der Mikrocontroller besitzt einen 80C51-Kern und hat folgende Eigenschaften:

- I^2C-bus interface
- 2k x 8 ROM
- 64 x 8 RAM
- 16bit autoreloadable counter/timer
- 5 channel 8bit A/D converter
- 8bit PWM output/timer
- 28pol. Gehäuse DIP oder PLCC

Hier wird nur der A/D-Wandler und der PWM-Ausgang verwendet, d.h. das Programm kann mit geringfügigen Änderungen auch auf jedem anderen Controller der 8051-Familie verwendet werden, der einen PWM-Ausgang und einen A/D-Wandler hat.

Gong- Schaltung

Über zwei Potentiometer wird die Lautstärke und die Wiedergabegeschwindigkeit eingelesen. Die Tonerzeugung erfolgt über den PWM-Ausgang. Die Ausgangsfrequenz wird mit 23,5kHz festgelegt, also deutlich über der Hörschwelle. Die Pulsbreite bestimmt die Amplitude der Ausgangsspannung (Lautstärke).



Versorgungsspannung V+ je nach Lautsprecherleistung 5....24V

Keil- C- Compiler und der 8xC752

Der 'C752 hat einen internen Code-Speicher von nur 2K und kann außerdem keine `LJMP`- und `LCALL`-Befehle ausführen. Der Keil C51-Compiler muß mit der Steueranweisung ROM (`SMALL`) aufgerufen werden, damit er keinen `LCALL`- bzw `LJMP`-Befehl kodiert. In die Eingabe-Modul-Liste des Linker/Locaters L51 muß die Speziallibrary `80C751.lib`, die diese Einschränkungen berücksichtigt, aufgenommen werden. Außerdem ist ein anderes Startup-Modul erforderlich.

Eine Batch-Datei zum Aufruf des Assemblers, C-Compilers und Linker/Locaters kann z.B. so aussehen:

```
a51 %2.asm
c51 %1.c CODE DEBUG OBJECTEXTEND  ROM (SMALL)
l51 %1.obj, %2.obj, start751.obj, 80C751.lib RAMSIZE (64)
oh51 %1
```

Durch die Anweisung `RAMSIZE(64)` wird dem Linker/Locater der kleinere RAM-Bereich des 'C752 bekanntgegeben. Nur so kann der Linker/Locater überprüfen, ob der RAM- Bereich für die Variablen und den Stack ausreicht.

Hungarian Notation

Windows- Programmierern ist sicher die in Charles Pezold's 'Programming Windows' vorgeschlagene Kennzeichnung der Datentypen von Variablen durch eine „Vorsilbe“ geläufig. Gerade bei der Programmierung von Mikrocontrollern ist es wichtig, über den Typ einer Variablen genau Bescheid zu wissen. Ich verwende deshalb auch bei der Mikrocontrollerprogrammierung die folgende Notation:

```
c     char
by    BYTE (= unsigned char)
i     int
w     WORD (= unsigned int)
b     BOOL (bit)
l     long
```

also z.B:

- eine Integervariable: `iVar`
- eine WORD- Variable: `wZeitgeber` u.s.w.

Diese Notation macht natürlich auch vor Zeigern nicht Halt, man schreibt z.B. für einen Pointer auf `int`: `piIntPointer`.

Das Programm

Der größte Teil ist in C geschrieben, die Interrupt-Prozedur für den PWM-Timer aus Geschwindigkeitsgründen in Assembler.

Im Programmkopf finden sich Typdefinitionen für die Typen `BYTE` und `WORD`, Variablendeklarationen und Prototypen für die verwendeten Funktionen.

Das Hauptprogramm führt die Initialisierung (Funktion `initialisieren()`) des PWM-Timers, die Interruptfreigabe und das erstmalige Einlesen der Lautstärken- und Ablaufgeschwindigkeits-Potentiometer durch.

In der für jedes Mikrocontrollerprogramm typischen Endlosschleife wird die Funktion `Gong()` aufgerufen, falls die Variable `byFlag` gesetzt wird. (`byFlag` könnte auch vom Typ `bit` sein.)

`byFlag` wird in der PWM-Timer-Interruptroutine abhängig von der Einstellung der Ablaufgeschwindigkeit gesetzt. Diese Art der Erzeugung eines Taktes wurde gewählt, damit kein zusätzlicher Timerinterrupt verwendet werden muß. Die Verwendung mehrerer Interruptquellen ist insofern etwas problematisch, weil der 'C752 nur eine Interrupt-Prioritätsebene unterstützt, d.h. ein gerade laufender Interrupt kann von einem zweiten (auch höherprioren) nicht unterbrochen werden.

Gong

Der Gongklang kann sehr einfach durch eine exponentiell abklingende Amplitude der Grundfrequenz nachgebildet werden. Der 'Anschlag' ergibt sich aus einem sprungförmigen Einsatz der Anfangs-(Maximal)amplitude.

In der Funktion `Gong()` werden die Gongschläge gestartet, die drei Gong-Grundfrequenzen festgelegt, der exponentielle Amplitudenabfall berechnet und die beiden Potentiometer eingelesen.

Die Gonggrundfrequenz wird durch die Variablen `byft` und `byfh` festgelegt. In der PWM-Timer-Interruptroutine (`pwm.asm`) wird nach `byfh` Durchläufen die PWM abgeschaltet, nach `byft` Durchläufen wieder eingeschaltet. Die PWM-Frequenz beträgt 23,5 kHz, d.h. der PWM-Timer-Interrupt erfolgt alle 42,5µs. Daraus ergibt sich für `byft` und `byfh`:

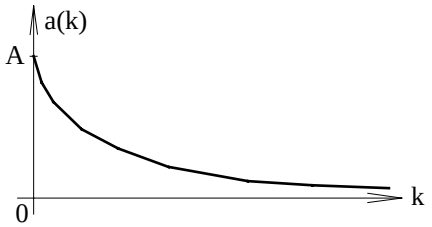
`byft` = 23,5kHz / Gongfrequenz

`byfh` = `byft` / 2

Die Grundfrequenzen wurden wie beim SAB600 gewählt:

1. Gongschlag	660Hz	`byft`= 36
2. Gongschlag	550Hz	`byft`= 43
3. Gongschlag	440Hz	`byft`= 54

Für die Berechnung des exponentiellen Abfalls wird die folgende Näherungsfomel verwendet:



a(k) = a(k-1) * λ ; a(0) = A ; k >= 1

Im Programm: a(k).....`byATmp`.....Amplitudenmerker

λ = 247/256 = 0,96...Abklingdauer

Nachdem die drei Gongschläge abgelaufen sind (`byGong` == 3), kann der Gong nur durch einen Reset wiedergestartet werden.

Der Testausgang TEST erlaubt die Funktionskontrolle des Mikrocontrollers mit Hilfe eines Oszilloskops. Der Testausgang wird je nach Stellung des Potentiometers für die Ablaufgeschwindigkeit alle 16ms.....5ms kurz HIGH.

Die PWM- Interruptprozedur

Die Erzeugung der Tonfrequenz (Grundfrequenz des Gongs) erfolgt durch Ein- und Ausschalten der PWM. Dazu wird - abhängig von den Variablen `byft` und `byfh` - das PWM-compare register PWCM mit minimaler Pulsbreite (PWCM=255) und dem momentanen Amplitudenwert `byAmplitude` geladen. Dabei muß `byAmplitude` von 255 abgezogen werden, weil einem Tastverhältnis Tv=1 ein Wert PWCM=0 entspricht.

Die Funktion wurde in Assembler implementiert, damit das compare register PWCM möglichst sofort nach Auftreten des Interrupts nachgeladen werden kann. (`MOV PWCM,byTmp`)

Erst dann wird der Akku und das Programm Status Wort auf den Stack gepusht. Anschließend wird der Variablen `byTmp` der Wert 255 bzw. 255-byAmplitude zugewiesen, d.h. PWCM wird erst beim darauffolgenden Funktionsaufruf (PWM-Interrupt) mit dem neuen Wert geladen.

Die Zählvariable `wCnt` wird bei jedem PWM- Interrupt um eins erhöht, und `byFlag` gesetzt, falls `wCnt > wZeitgeber`. So wird - wie schon erwähnt - die Ablaufgeschwindigkeit gesteuert. (Aufruf der Funktion `Gong()` in der Endlosschleife im Hauptprogramm.)

```
/* gong.c ------------------------------------------------------------------*/
/*                                                                           */
/* MC- Programm für Tonerzeugung mit dem '752  (Dreiklanggong)               */
/*---------------------------------------------------------------------------*/

#include <reg752.h>                 /* include file für den 87C752           */

typedef unsigned char  BYTE;
typedef unsigned int   WORD;
#define LOBYTE(int) int & 0xFF
#define HIBYTE(int) (int>>8) & 0xFF

#define PERIODE     0x00            /* PWM- Frequenz maximal (23.5kHz)       */
#define PWM_OFF     0xFF            /* kürzest möglicher Puls (PWM Low)      */

#define AMP_MAX     220             /* maximale Amplitude                    */

sbit    TEST=       P1^7;           /* ein Testausgang                       */

/* globale Variable:                */
WORD    wZeitgeber= 300;            /* Zeitgeber CompareWert                 */
BYTE    byFlag= 1;                  /* Flag -> Zeitgeber mit PWM- overflow   */
BYTE    byAmpPot= 0;                /* AmplitudenWert [0(Max)...255(Min)]    */
BYTE    byAmplitude= AMP_MAX;       /* momentaner Amplitudenwert             */
BYTE    byft= 36;                   /* Tonfrequenz- CompareWert              */
BYTE    byfh= 18;                   /* Tonfrequenz- CompareWert/2            */
WORD    wCnt= 0;                    /* Zählvariable, verwendet in pwm.asm    */
BYTE    byTmp;                      /* Variablen verwendet in pwm.asm        */
BYTE    byCnt;

/* function prototypes:             */
void Gong( void);
void ReadPots( BYTE);
void initialisieren( void);

/*****************************************************************************/
/*                                                                           */
/*                           HAUPTPROGRAMM                                   */
/*                                                                           */
/*****************************************************************************/
main()
{
    initialisieren();

    while (1)                       /* Endlos- Schleife                      */
    {
      if(  byFlag)                  /* bFlag wird in PWMGen() abhängig von
                                     * der Potstellung (wZeitgeber) gesetzt */
      {
       byFlag= 0;
       Gong();                      /* exp. Abfall berechnen, ADC einlesen   */
      }
    }
}
/* Ende Hauptprogramm ------------------------------------------------------*/


/* intitialisieren() -------------------------------------------------------*/
/*                                                                           */
/*---------------------------------------------------------------------------*/
void initialisieren( void)
{

        /* 23.529kHz- PWM- Generator: */
    PWMP= PERIODE;                  /* PWM prescaler                         */
    PWCM= PWM_OFF;                  /* PWM Ausgang LOW                       */

    IE= 0x88;
    PWENA= 1;

    ReadPots( 2);                   /* wZeitgeber einlesen (Pot an P1.2)     */
    ReadPots( 3);                   /* byAmpPot einlesen   (Pot an P1.3)     */

    byFlag= 1;
}
/* Ende initialisieren() ---------------------------------------------------*/


/* gong() ------------------------------------------------------------------*/
/*                                                                           */
/*       Berechnung der exponentiell abklingenden Amplituden,                */
/*       Auslösung der 3 Gongschläge und Einlesen der Potentiometer          */
/*                                                                           */
/*---------------------------------------------------------------------------*/
void Gong( void)
{
static BYTE   byATmp= 0;            /* Amplitudenmerker für letzten Ampwert  */
static BYTE   byAD= 2;              /* Analogwandler- Kanal [2,3]            */
static BYTE   byGong= 0;            /* Gonganschlag 0,1,2                    */
static BYTE   k = 0;                /* Stufennummer [0...128]                */
static BYTE   GongTab[3] = { 60, 60, 200};    /* Länge der Abklingvorgänge   */

       TEST =1;

       if( byGong < 3)
       {
        if( ++k < GongTab[byGong])       /* Länge k pro Gongschlag           */
        {
         if( byGong == 0)
           { byft= 36; byfh= 18; }       /* 36*42.5us => 654Hz  (statt 660) */
         else
           if( byGong == 1)
           { byft= 43; byfh= 22; }       /* 43*42.5us => 547Hz  (statt 550) */
           else
             if( byGong == 2)
             { byft= 54; byfh= 27; }     /* 54*42.5us => 434Hz  (statt 440) */

         if( k == 1) byATmp= byAmpPot;   /* Anfangsamplitude (Pot an P1.3)  */

         byATmp= ((WORD)(byATmp* 247))>>8;            /* 'log. Decrement'  */
         byAmplitude= ((WORD)(byATmp*AMP_MAX)) >> 8;  /* Norm. auf AMP_MAX */
        }
        else {
              k= 0;
              byGong++;
              byAmplitude= 0;
             }
       }

     ReadPots( byAD);                /* byAD = [2,3]                          */
     if( ++byAD > 3)
      byAD= 2;

     TEST= 0;
```

```
}
/* Ende Gong() ------------------------------------------------------------*/



/* ReadPots( byAD) -------------------------------------------------------*/
/*                                                                        */
/*   Liest den Analogkanal 2 (P1.2) als Wert für die Ablaufgeschwindigkeit  */
/*     und den Analogkanal 3 (P1.3) als Wert für die Lautstärke            */
/*                                                                        */
/*   byAD gibt den Analogkanal an; byAd= [2,3]                            */
/*------------------------------------------------------------------------*/
#define Start_ADC0  0x28
#define Start_ADC1  0x29
#define Start_ADC2  0x2A
#define Start_ADC3  0x2B
#define Start_ADC4  0x2C

void ReadPots( BYTE byAD)
{
   switch( byAD)
   {
     case 2:           /* AD2.....Grundfrequenz einlesen                  */
       ADCON= Start_ADC2;      /* start conversion  Ch2 an P1.2            */
       while( !(ADCON&0x10));  /* auf EOC warten                           */
       wZeitgeber= ADAT+ 120;
      break;

     case 3:           /* AD3.....Lautstärke einlesen                     */
       ADCON= Start_ADC3;      /* start conversion  Ch3 an P1.3            */
       while( !(ADCON&0x10));  /* auf EOC warten                           */
       byAmpPot= ADAT;
      break;

     default:
      break;
   }
}
/* Ende ReadPots() -------------------------------------------------------*/
```

```
;/*  PWMGEN  ---------------------------------------------------------------*/
;/*                                                                        */
;/*  PWM- interrupt routine  für den 8xC752                                */
;/*                                                                        */
;/*       Die PWM- interrupt routine liegt in einem absoluten CodeSegment  */
;/*       an der PWM- Interruptadresse 0x33. Dadurch muß kein Sprung aus-   */
;/*       geführt werden. Der erste Befehl nach einem Interrupt ist das    */
;/*       Nachladen des Pulsbreiteregisters (MOV PWCM, byTmp), d.h. nach    */
;/*       spätestens 5us ist das Register geladen (3us Interrupt, 2us MOV).*/
;/*                                                                        */
;/*       Wird die maximale Pulsbreite auf 42,5us-6us beschränkt (damit    */
;/*       keine Knackgeräusche auftreten), so ist die maximale Amplitude:  */
;/*       byAmplitude(Max) = 220.                                          */
;/*                                                                        */
;/*------------------------------------------------------------------------*/
```

```
$ NOMOD51
$ INCLUDE (REG752.INC)

NAME    PWMGEN

INTR    EQU     33H               ; interrupt Einsprungadresse
EXTRN   DATA    ( byTmp)          ; definiert im C- Programm
EXTRN   DATA    ( byCnt)          ;  -"-
EXTRN   DATA    ( byAmplitude)    ;  -"-
EXTRN   DATA    ( byfh)           ;  -"-
EXTRN   DATA    ( byft)           ;  -"-
EXTRN   DATA    ( wZeitgeber)     ;  -"-
EXTRN   DATA    ( byFlag)         ;  -"-
EXTRN   DATA    ( wCnt)           ;  -"-

PUBLIC  PWMGen                    ;

CSEG   AT INTR                    ; Einsprung Timer2 interrupt
                                  ; Absolutes Segment

PWMGen:
     MOV     PWCM,byTmp           ; CCL= byTmp;   Pulsbreite nachladen
     PUSH    ACC
     PUSH    PSW

     MOV     A,byCnt              ; if( byCnt == byfh)
     CJNE    A,byfh,?LBL1

     MOV     byTmp,#0FFH          ;   byTmp= 255;
     SJMP    ?LBL2
?LBL1:                            ; else
     MOV     A,byCnt
     CJNE    A,byft,?LBL2         ;  if( byCnt == byft)
     CLR     C                    ;  { byTmp= 255- byAmplitude;
     MOV     A,#0FFH
     SUBB    A,byAmplitude
     MOV     byTmp,A
     CLR     A                    ;    byCnt= 0;
     MOV     byCnt,A              ;  }
?LBL2:
     INC     byCnt                ; byCnt++;

     INC     wCnt+01H             ; wCnt++;
     MOV     A,wCnt+01H
     JNZ     ?LBL3
     INC     wCnt
?LBL3:
     CLR     C                    ; if( wCnt >= wZeitgeber)
     MOV     A,wCnt+01H           ; {
     SUBB    A,wZeitgeber+01H
     MOV     A,wCnt
     SUBB    A,wZeitgeber
     JC      ?LBL4
     MOV     byFlag,#01H          ;   byFlag= 1;
     CLR     A                    ;   wCnt= 0;
     MOV     wCnt,A
     MOV     wCnt+01H,A           ; }
?LBL4:
     POP     PSW
     POP     ACC
     RETI

END
```

□

Ergänzung zum folgenden Beitrag *Rund um das INTEL-HEX-Format*

Interrupts mit Dualboot

Anwendung für HEXPAT.EXE

Die Interruptvektoren des FSD51 enthalten einen Sprung auf die Adressen `A0xx`, sodaß auch eigene Interruptvektoren eingesetzt werden können, sofern sie entweder händisch oder durch das Anwenderprogramm auf die Adressen `A0xx` geschrieben werden.

Der C-Compiler von KEIL generiert selbsttätig Interruptvektoren auf die Adressen `00xx`. Die neuen Versionen ab Versionsnummer 3.4 können die Interruptvektoren mit der `#pragma`-Anweisung `IV (0xaaaa)` auch auf die Adresse `aaaaH` parallelverschieben.

Bei Verwendung älterer Versionen kann man sich mit einem Patch der HEX-Datei behelfen.

Annahme:

Man bearbeitet ein Programm `TESTI` und erhält durch

```
OHS51 TESTI.ABS
```

die HEX-Datei `TESTI.HEX`.

Durch Anwendung des HEX-Patchers `HEXPAT`, können alle Ladeadressen eines bestimmten Bereichs auf einen anderen Bereich verschoben werden:

```
HEXPAT TESTI A0 00
```

Das Programm `HEXPAT` erzeugt `TESTI.HEY`, die gepatchte HEX-Datei.

Man ruft den Debugger auf:

```
TS51 TESTI.ABS INIT (TESTI.INI)
```

Die Datei `TESTI.INI` enthält unter anderem die Zeile zum Nachladen der gepatchten HEX-Datei.

```
...
LOAD <pfad>TESTI.HEY
```

Die Datei TESTI wird dabei eigentlich zweimal geladen: zuerst beim Aufruf des Debugger in der Kommandozeile und dann in der INI-Datei. Beim ersten Laden über die Kommandozeile sind die Interruptvektoren noch auf 00xx ausgerichtet. Glücklicherweise meldet der TS51 dabei keinen Fehler. In diesem Ladevorgang werden auch alle Symbolinformationen im TS51 bekanntgegeben. Der zweite Ladevorgang ist eigentlich redundant mit Ausnahme eben der verschobenen Interruptvektoren. □

```
For those who believe in Winter Wonderland's, the Rockies present
excitement galore:


            *
             \
              \
               |\
              \ | \ (__)
              \\|| \(oo)
               \||\ \\/
                 ^^ \||
                   \\ ||
                    \\||
                     \||
                      ^^
                       \\_
                        \_
   Cow skiing a Black Diamond at Aspen


                    \ (__)
                    \\(oo)
                  /------\\\/
                 / |    (##)
                *  ||----||"
                   ^^    ^^
    This cow plays bagpipes.
```