

Sound mit dem ATmega16

Christian Arenz

SS 2007

Inhaltsverzeichnis

1	Grundlegendes zur Tonerzeugung	3
1.1	Parameter eines Tons	3
1.2	Wellenform	4
1.3	Notenlänge	4
1.4	Hüllkurve	5
2	Künstliche Klangerzeugung	6
2.1	Additive Synthese	6
2.2	Subtraktive Synthese	6
2.3	Direkte Digitale Synthese	6
2.4	Wavetable Synthese	8
3	Realisierung	10
3.1	Konzept A - Direkte Tonerzeugung	10
3.2	Konzept B - Pulsweitenmodulation	12
4	Fazit	15
A	PWM-Modi des ATmega16	17

Einleitung

Um Töne zu erzeugen bedarf es genauerer Kenntnisse über deren Eigenschaften und den Methoden ihrer Synthetisierung. Dieses Dokument soll einen kurzen Einblick in die Akustik, die Verfahren der künstlichen Tonerzeugung und deren Umsetzung mit dem ATmega16 geben. Es wird gezeigt welche Herangehensweisen prinzipiell möglich sind, wo ihre Grenzen liegen und welche einfach zu realisieren sind.

1 Grundlegendes zur Tonerzeugung

1.1 Parameter eines Tons

Wenn wir von Tönen sprechen meinen wir damit die Ausbreitung von Schall. Schall ist physikalisch gesehen eine Welle, die sich meist longitudinal durch ein Medium, mit einer für dieses charakteristischen Geschwindigkeit, ausbreitet.

Töne verstehen sich somit als Wellen. Ein Ton wird in erster Linie durch Tonhöhe und Lautstärke beschrieben, was der Frequenz und der Amplitude der erzeugten Welle entspricht.

Ton	Frequenz	Ton	Frequenz	Ton	Frequenz
C4	262 Hz	F4	349 Hz	A4	440 Hz
C4#	277 Hz	F4#	370 Hz	A4#	466 Hz
D4	294 Hz	G4	392 Hz	B4	494 Hz
E4	330 Hz	G4#	415 Hz	C5	523 Hz

Tabelle 1: Töne und deren Frequenzen

Das Verhältnis zweier benachbarter Halbtöne beträgt beim 12-Tonsystem $\sqrt[12]{2}$.

1.2 Wellenform

Ein weiteres wichtiges Kriterium ist die Wellenform, welche maßgeblich die Klangfarbe eines Tons bestimmt. Hiermit ist nicht die Form einer Schwingung als solche gemeint, wie z.B. die Rechteck, Dreieck, oder Sinusschwingung, vielmehr setzen sich Töne in der Regel aus mehreren Einzelschwingungen, den sogenannten Obertönen, zusammen. Dadurch entsteht die für ein bestimmtes Instrument charakteristische Wellenform. Bei Saiteninstrumenten zum Beispiel besteht die Obertonreihe (Gesamtheit der Obertöne) aus, zumindest näherungsweise, ganzzahligen Vielfachen der Grundschwingung.

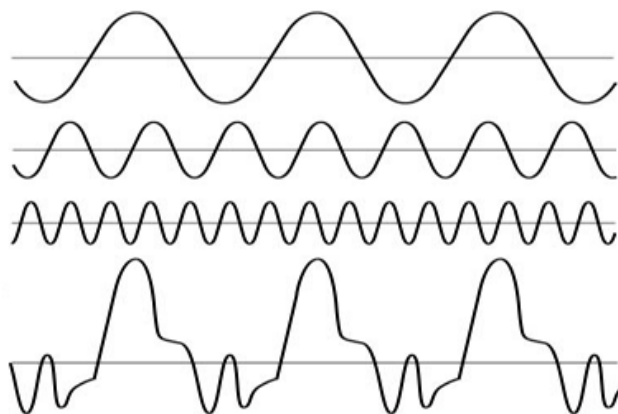


Abbildung 1: Grundschwingung + 2 Obertöne

1.3 Notendlänge

Weiterhin ist es notwendig sich mit den Notendängen auseinander zu setzen. In der elektronischen Musik hat sich als Einheit für die Gewwindigkeit eines Stücks die sogenannten „beats per minute“ eingebürgert.

Beats per minute ist ein Maß für die Anzahl der Viertelnoten pro Minute.

Somit lässt sich die Notendlänge leicht anhand folgender Formel errechnen.

$$t = \frac{1}{bpm}$$

Für 120 bpm ergibt dies:

- $\frac{1}{1}$ Note = 2000 ms

- $\frac{1}{2}$ Note = 1000 ms
- $\frac{1}{4}$ Note = 500 ms
- $\frac{1}{8}$ Note = 250 ms
- ...

1.4 Hüllkurve

Eine Aneinanderreihung gleich hoher Noten erscheint dem Hörer als eine, lange gespielte, Note. Um dies zu vermeiden hat man die Möglichkeit nach jeder gespielten Note eine kurze Pause einzulegen, was allerdings die gespielte Note, wenn auch nicht hörbar, verlängert. Man verwendet daher besser die sogenannte Hüllkurve oder auch Envelope.

Die Hüllkurve wird eingesetzt um die Lautstärke über die Dauer eines Tons zu verändern. Dabei wird der Ton in vier Phasen eingeteilt.

- Attack - Zeit in der die Lautstärke auf ihr Maximum ansteigt
- Decay - Zeit in der die Lautstärke auf den Sustain-Level zurückfällt
- Sustain - Wert für die Lautstärke auf dem der Ton gehalten wird
- Release - Zeit die der Ton zum Ausklingen benötigt

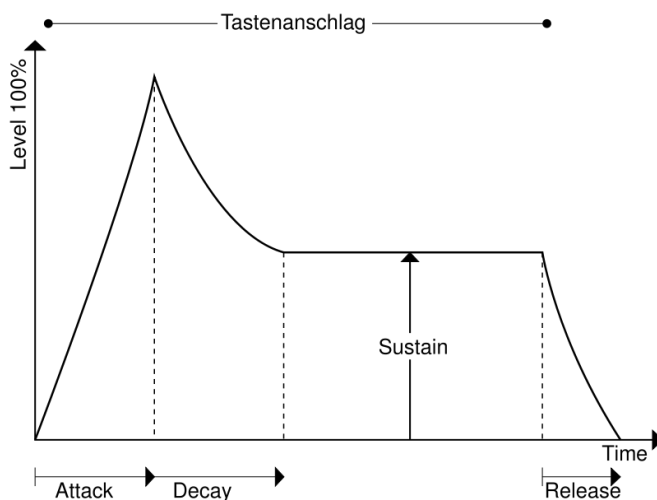


Abbildung 2: ADSR-Envelope

Aufgrund dieser vier typischen Parameter wird die Hüllkurve auch ADSR-Envelope genannt. Diese bringt weiterhin den Vorteil mit sich wesentlich lebendigere Klänge zu erzeugen.

2 Künstliche Klangerzeugung

Im Folgenden sollen kurz mehrere Verfahren zur künstlichen Klangerzeugung vorgestellt werden. Das Augenmerk fällt dabei auf diejenigen Verfahren, deren Umsetzung mit einem ATmega16 möglich sind. Komplexere Methoden wie z.B. das „physical modelling“ werden dabei aussen vor gelassen.

2.1 Additive Synthese

Die Additive Synthese basiert auf den Überlegungen des französischen Mathematikers Fourier. Nach diesem lässt sich jeder denkbare Klang aus einer Mischung von verschiedenen Sinusschwingungen darstellen, wie schon bei den Obertonreihen gesehen. Da komplexere Klänge aber zum Teil aus sehr vielen verschiedenen Frequenzen bestehen, ist dieses Syntheseverfahren für unsere Zwecke nur bedingt einzusetzen.

Die Additive Synthese findet allerdings bei der Erzeugung von Akkorden bzw. bei mehrstimmigen Liedern Verwendung.

2.2 Subtraktive Synthese

Der Additiven Synthese gegenüber steht die Subtraktive Synthese. Der ursprüngliche Klang wird durch einen Oszillator erzeugt und ist reich an Obertönen. Das „klangliche Rohmaterial“ wird durch Verwendung von Filtern oder Hüllkurven um bestimmte Frequenzanteile vermindert.

Dieses Verfahren soll nur der Vollständigkeit halber vorgestellt werden, und findet im weiteren keine Anwendung.

2.3 Direkte Digitale Synthese

Die Direkte Digitale Synthese (DDS) ist ein Verfahren welches häufig in Funktionsgeneratoren Anwendung findet. Sie ist einfach zu realisieren und hat nur geringe Ansprüche an die verwendete Hardware.

Bei der Direkten Digitalen Synthese wird eine Periode eines vorher aufgezeichneten Signals im Speicher gehalten. Um eine hohe Qualität zu erreichen, werden hierbei möglichst viele Werte mit einer möglichst hohen Amplitudenauflösung gesampelt. In der Praxis sind das üblicherweise 2000 bis 8000 Werte, bei einer Auflösung von 8-12 Bit.

Die Werte dieses Referenz-Signals werden nun in gleichen Zeitabständen wieder ausgegeben. Um verschiedene Frequenzen zu erzeugen überspringt man einzelne Stützstellen des Signals. Um z.B. eine Verdopplung der Grundfrequenz zu erreichen wird bei der Ausgabe

2 Künstliche Klangerzeugung

jede zweite Stützstelle ausgelassen. Durch Auslassen vereinzelter Stützwerte erreicht man eine nur geringfügig höhere Frequenz.

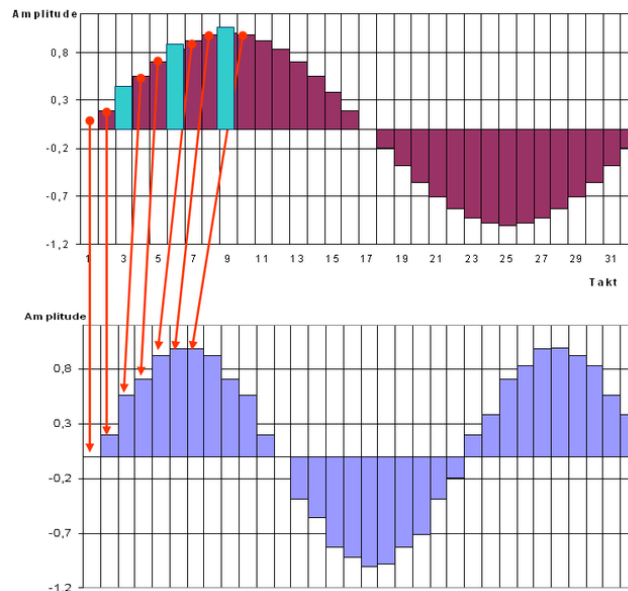


Abbildung 3: Direkte Digitale Synthese

Ein Nachteil der DDS ist die Tatsache, dass bei zu grossem Abstand zwischen Ausgabe-Frequenz und der des Referenz-Signals, die Wellenform sehr stark verfälscht wird. Oder anders, um einen im Vergleich zum Ausgangssignal hohen Ton zu erzeugen müssen sehr viele Samplewerte ausgelassen werden, welche aber unter Umständen wichtig für die Klangcharakteristik des Tons sind.

Eine einfache Implementierung der DDS sieht wie folgt aus:

...

```
uint8_t SINE[256] = {128,130,...}; // Tabelle mit 256 Werten
```

```
int main(){
    //PORTD als Output konfigurieren
    DDRD = 0xff;

    uint8_t cnt;
```

```
while(1){  
    cnt += 2; // 2: Verdoppelte Grundfrequenz  
    PORTD = SINE[cnt];  
}  
}  
  
...
```

Bei jedem Schleifendurchlauf wird der Counter erhöht und der neue Wert ausgegeben. Dabei errechnet sich die Ausgabe-Frequenz aus der CPU-Frequenz geteilt durch die Anzahl der Takte von Ausgabe zu Ausgabe. Die genauen Werte hierzu lassen sich natürlich in Assembler leichter ermitteln, da dort vom Compiler keine Optimierungen mehr vorgenommen werden. Die Grundfrequenz der erzeugten Welle errechnet sich wiederum aus der Ausgabe-frequenz geteilt durch die Anzahl der Samples pro volle Periode. Da ja genau eine Periode aufgezeichnet wurde entspricht diese, bei Ausgabe von jedem Wert genau der Anzahl der gesampelten Werte, in diesem Fall 256, bei Ausgabe jedes zweiten Wertes 128, usw.

Dieses zugegebenermaßen einfache Beispiel beinhaltet keine Logik zum Verändern des ausgegeben Tons und soll nur die prinzipielle Vorgehensweise demonstrieren.

2.4 Wavetable Synthese

Die Wavetable-Synthese funktioniert nach dem gleichen Prinzip wie die DDS. Auch hierbei wird ein Referenz-Signal im Speicher abgelegt, im Unterschied zur DDS wird im Regelfall nicht nur eine Periode, sondern der gesamte Ton aufgezeichnet, und dieser mit variabler Wiedergabegeschwindigkeit unter Berücksichtigung jeden Stützwertes ausgegeben. Ein Sample beinhaltet somit schon einen Lautstärkeverlauf und es muss dementsprechend keine Hüllkurve mehr generiert werden. Zu diesem Zweck wird das Sample direkt in die 4 Bereiche der ADSR-Envelope aufgeteilt.

Dadurch dass bei der Wavetable-Synthese jeder gespeicherte Wert ausgegeben wird, bleibt die Klangcharakteristik auch bei höheren Frequenzen weitestgehend erhalten. Die Veränderung der Frequenz lässt sich somit allerdings nur durch ein zusätzliches Delay nach der Ausgabe eines Wertes erreichen. Da durch dieses Delay die Frequenz nur verringert werden kann, wird die maximale Ausgabefrequenz durch den Systemtakt, oder besser, durch die Zeit zwischen zwei Ausgaben bestimmt.

Durch die einfache Realisierung wird die Wavetable-Synthese oft bei preiswerten Soundkarten verwendet. Diese haben dazu einen eigenen Wavetable-Speicher in dem die Samples abgelegt werden.

Qualitativ hochwertige Soundkarten verwenden mehrere Samples pro Instrument, um eine

genauere Darstellung der Wellenform in höheren Tonlagen zu erreichen, dies geht hin bis zu einem Sample pro Tonhöhe, was bereits mit einem Sampler zu vergleichen ist.

Ein einfaches Beispiel:

```
...  
  
uint8_t SINE[256] = {128,130,.....}; // Tabelle mit 256 Werten  
  
int main(){  
    uint8_t cnt;  
    while(1){  
        PORTA = SINE[cnt++];  
        _delay_ms(value);  
    }  
}  
  
...
```

3 Realisierung

3.1 Konzept A - Direkte Tonerzeugung

Eine Möglichkeit zur Erzeugung von Tönen bieten die Timer des ATmega. Es werden dabei keine der oben vorgestellten Methoden verwendet. Der Ton bzw. dessen Frequenz wird direkt im Timer eingestellt. Da möglichst das gesamte hörbare Spektrum abgedeckt werden soll, sollten die minimale Timerfrequenz ≤ 50 Hz und die maximale Timerfrequenz ≥ 20 kHz betragen.

gesucht: $F_{Ovf} \approx 20 - 50$ Hz

Prescaler	F_{Ovf}			
	8 bit	9 bit	10 bit	16 bit
1	78,1 kHz	39 kHz	19,5 kHz	305 Hz
8	9,8 kHz	4,8 kHz	2,4 kHz	38 Hz
64	1221 Hz	610 Hz	305 Hz	4,7 Hz
256	305 Hz	153 Hz	76,3 Hz	1,1 Hz
1024	76,3 Hz	38,1 Hz	19 Hz	0,3 Hz

\Rightarrow 16 bit / Prescaler 8

Mit diesen Parametern ergeben sich folgende Werte für die Frequenzen:

- $F_{Timer} = \frac{F_{CPU}}{8} = 2.5$ MHz
- $F_{min} = F_{OVF} = \frac{F_{Timer}}{2^{16}+1} = 38$ Hz
- $F_{max} = \frac{F_{Timer}}{2+1} = 833$ kHz

Auf diese Weise lässt sich zwar nur ein Rechtecksignal erzeugen, die Umsetzung ist dafür aber denkbar einfach und es werden keine zusätzlichen Anforderungen an die Hardware gestellt. Da die Timerfrequenz der Frequenz des Tones entsprechen soll bedeutet dies dass für jeden Ton ein eigenes Timer-TOP definiert werden muss.

Der Timer wird auf Fast PWM (WGM 14) und Compare Output Mode konfiguriert. Damit ergibt sich als Timer-TOP das ICR1 Register. Der Wert hierfür lässt sich durch folgende Formel errechnen.

$$ICR1 = \frac{F_{Timer}}{F_{Ton}}$$

Als Vergleichswert ergibt sich das OCR1A Register. Zu Beginn einer Timer-Periode (BOTTOM) wird der Ausgabepin OC1A auf HIGH-Pegel und bei Erreichen des OCR1A Wertes auf LOW-Pegel gesetzt. Das OCR1A Register kann somit genutzt werden um die Hüllkurve zu realisieren.

Eine einfache Implementierung könnte z.B. so aussehen:

```
...

#define C 4778
#define D 4257
#define E 3792

...

uint8_t SONG[] = {C,D,E...,STOP};           // Array mit Noten
uint8_t LENGTH[] = {500,520,540...,STOP}; // Array mit Notenlängen

ISR(TIMER1_OVF_vect){
    static uint8_t laenge;
    static uint8_t count;

    // wenn Note zu ende
    if(laenge++ > notenLaenge){

        // wenn Song zu ende
        if(SONG[count] == STOP) {
            TCCR1B &= ~(1<<CS11); // Timer aus
            return;
        }

        // sonst neue Note
        ICR1 = SONG[cnt]
        OCR1A = 500; // zu beginn der note irgend eine Lautstärke
        count++;
        laenge = 0;
    }
    // sonst Hüllkurve
    else {
        // tue irgendwas mit OCR1A
        ...
    }
}
```

3.2 Konzept B - Pulsweitenmodulation

Im Unterschied zu Konzept A wird bei der Pulsweitenmodulation (PWM) das Tastverhältnis bei konstanter Frequenz moduliert. Anders gesagt, es wird mit einem konstanten Timer-TOP gearbeitet. Das zu erzeugende analoge Signal wird als Pseudo-Sinus mit Hilfe der Timer ausgegeben. Die Amplitude des Ausgangs-Signals entspricht dabei der Pulslänge, also dem Verhältnis zwischen „Anzeit“ und „Auszeit“.

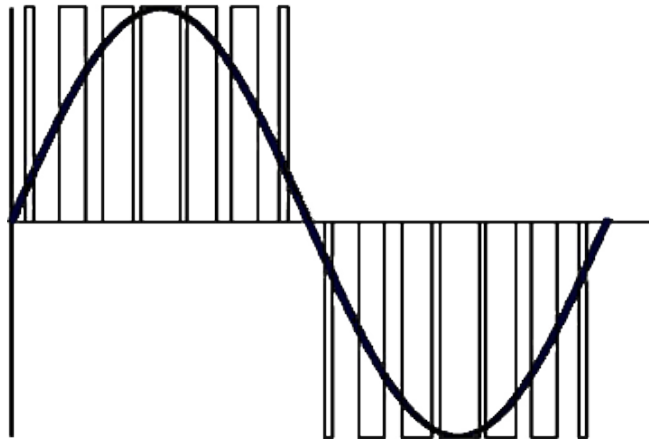


Abbildung 4: Pulsweitenmodulation

Der ATmega16 bietet die Möglichkeit mit jedem der drei Timer ein PWM-Signal zu erzeugen. Dabei sollte eine möglichst hohe Auflösung und eine PWM-Frequenz oberhalb des hörbaren Bereichs gewählt werden.

gesucht: $F_{Ovf} \geq 20 \text{ kHz}$

Prescaler	F_{Ovf}			
	8 bit	9 bit	10 bit	16 bit
1	78,1 kHz	39 kHz	19,5 kHz	305 Hz
8	9,8 kHz	4,8 kHz	2,4 kHz	38 Hz
64	1221 Hz	610 Hz	305 Hz	4,7 Hz
256	305 Hz	153 Hz	76,3 Hz	1,1 Hz
1024	76,3 Hz	38,1 Hz	19 Hz	0,3 Hz

⇒ 9 bit / Prescaler 1

Somit sind die meisten Eckdaten bereits bestimmt. Wegen der Auflösung von 9 Bit kommt nur Timer1 mit einer maximalen Auflösung von 16 Bit in Frage. Timer1 hat als Output-Pins OC1A / OC1B. Zu beachten ist der Modus in dem der Timer läuft. Gewählt wurde hier der Fast PWM Mode (WGM 6) mit einem Timer-TOP von 0x01FF. Die Pulslänge wird durch das OCR1A Register bestimmt.

Der Vorteil dieser Vorgehensweise liegt in der einfachen Erzeugung von Stereo-Signalen, da hierzu nur ein zweiter Vergleichswert (OCR1B) verwendet werden muss. Darüber hinaus ist es möglich durch einfaches Addieren der Signalwerte auch polyphone Klänge zu erzeugen. Die Vielstimmigkeit wird allerdings durch die PWM-Auflösung beschränkt. Realistisch sind 3 Töne pro Kanal bei einer Auflösung des Ausgangssignals von 7 Bit.

```
...

const uint8_t wave[256] PROGMEM = {0,0,0,0,1,1,1,2,...};

// phase pointers
uint16_t phase0;
uint16_t phase1;

register const uint8_t * wave_ptr asm("r10"); // points to wave

// phase accumulators
uint16_t accu0 = 0;
uint16_t accu1 = 0;

...

ISR(TIMER1_OVF_vect){

asm volatile(

    "push r30"           "\n\t"
    "push r31"           "\n\t"

    "lds r30,(accu0)"    "\n\t" // load 1st accu byte
    "lds r31,(accu0+1)" "\n\t" // load 2nd accu byte
    "add %A[phase0],r30" "\n\t" // add accu
    "adc %B[phase0],r31" "\n\t" // to phase
    "movw r30,r10"      "\n\t" // let Z point to wave_start_address
```

```
"add r30,%B[phase0]"      "\n\t" // add upper phase byte to lower wave_ptr
"adc r31,__zero_reg__"   "\n\t" // add carry to upper wave_ptr
"lpm r12,Z"              "\n\t" // load wave_value

"lds r30,(accu1)"        "\n\t"
"lds r31,(accu1+1)"      "\n\t"
"add %A[phase1],r30"     "\n\t"
"adc %B[phase1],r31"     "\n\t"
"movw r30,r10"           "\n\t"
"add r30,%B[phase1]"     "\n\t"
"adc r31,__zero_reg__"   "\n\t"
"lpm r13,Z"              "\n\t"

"add r12,r13"            "\n\t" // add
"clr r13"                "\n\t" // both
"adc r13,__zero_reg__"   "\n\t" // values
"out 40,r12"             "\n\t" // write new value
"out 41,r13"             "\n\t" // to OCR1A

: [phase0] "=r" (phase0),[phase1] "=r" (phase1),
: "0" (phase0),"1" (phase1),"2" (phase2)
: "r12","r13"
);
}
```

Bei dieser Methode existiert ein 2 Byte langer „Pointer“ für jeden Ton wobei nur das höhere Byte auf den aktuellen Wert innerhalb der Sinus-Tabelle zeigt. Dieser wird mit jedem Aufruf der Interrupt-Routine um den zugehörigen Phaseaccumulator erhöht, wodurch die jeweilige Tonhöhe bestimmt wird.

Wählt man hierfür z.B. einen Wert < 256 wird der Ausgabewert erst nach mehreren Durchläufen verändert, wählt man einen Wert > 256 wird der Ausgabewert bei jedem Durchlauf verändert.

Hierzu definiert man am besten für jede gewünschte Frequenz oder Tonhöhe eine Variable und übergibt diese später nur noch dem Inline-Assembler.

Nachdem die Ausgabewerte für jeden der zwei Töne bestimmt wurden, werden diese addiert und in das OCR1A-Register ausgegeben.

4 Fazit

Wie an den gezeigten Beispielen zu erkennen ist, kann mit nur geringem Aufwand ein recht akzeptables Ergebnis erzielt werden. Der ATmega16 bietet mit seinen Timern ein einfaches Werkzeug zur Tonerzeugung und es besteht die Möglichkeit, je nach verfügbarer Rechenzeit und Speicherauslastung, zwischen verschiedenen Varianten zu wählen.

Eine wenig rechenintensive und speicheraufwändige Lösung lässt sich durch direkte Tonerzeugung umsetzen. Hierbei müssen zwar, aufgrund der Einschränkung der Wellenform, Abstriche bei der Soundqualität gemacht werden, dafür ist eine Umsetzung aber einfach zu realisieren.

Ein wenig mehr Aufwand muss betrieben werden um der Klangcharakteristik eines akustischen Instruments näher zu kommen. Soll eine höhere Qualität erreicht werden steigt allerdings auch der Speicherbedarf durch mehrere Wave-Tables stark an.

Im Hinblick auf eine universelle Einchip-Lösung stellt sich die Umsetzung zwar schwierig aber nicht unmöglich dar. Es muss zwischen Qualität und Leistung vermittelt werden, einfache Melodien oder beeps sind aber ohne Probleme umzusetzen.

Literatur

- [1] www.wikipedia.org
- [2] www.roboternetz.de
- [3] www.sengpielaudio.com
- [4] www.mikrocontroller.net

A PWM-Modi des ATmega16

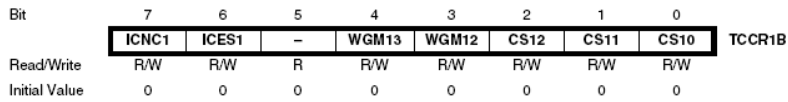
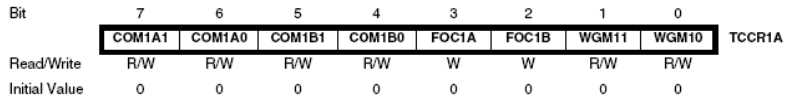


Abbildung 5: Timer/Counter Control Register

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Abbildung 6: Waveform Generation Modes

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OCnA/OCnB disconnected.
1	0	Clear OC1A/OC1B on compare match, set OC1A/OC1B at BOTTOM, (non-inverting mode)
1	1	Set OC1A/OC1B on compare match, clear OC1A/OC1B at BOTTOM, (inverting mode)

Abbildung 7: Compare Output Mode

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{clk}/1$ (No prescaling)
0	1	0	$clk_{clk}/8$ (From prescaler)
0	1	1	$clk_{clk}/64$ (From prescaler)
1	0	0	$clk_{clk}/256$ (From prescaler)

Abbildung 8: Prescaler

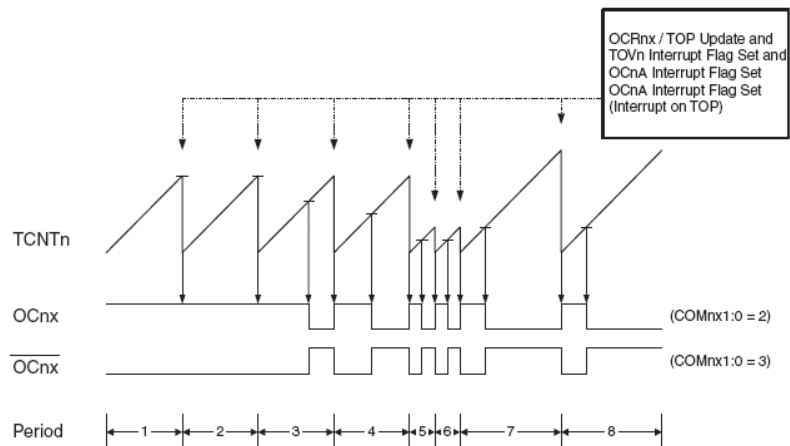


Abbildung 9: Timing Diagramm (Fast PWM)