Programmierung von ATMEL AVR Mikroprozessoren am Beispiel des ATtiny13

Eine Einführung in Aufbau, Funktionsweise, Programmierung
und Nutzen von Mikroprozessoren

Teil VI: Programmieren an weiteren Beispielen

Programmierbeispiel 03: LED an und ausschalten

bsp03_blink.asm - Editor

Datei Bearbeiten Format ?

```
; ***************************************************************
; * Mehr Action: LED-Blinker                                    *
; * (C)2005 by info@avr-asm-tutorial.net                        *
; ***************************************************************
;
.INCLUDE "tn13def.inc"
;
; Schaltbild:
;                     ATMEL ATtiny13
;                        ___   ____
;                 ___   1/   |_|    |8
;+5 Volt O--|___|----|Res      Vcc|----O + 5 Volt
;                     |            |
;                     |PB3      PB2|
;                     |            |6  ___
;                     |PB4      PB1|--|___|--|<|--O + 5 Volt
;                    4|            |   330    LED
;          0 Volt O---|Gnd      PB0|
;                     |____________|
;
        sbi DDRB,1 ; PB1 ist Ausgang
loop:
        cbi PORTB,1 ; Ausgang auf Null
        sbi PORTB,1 ; Ausgang auf Eins
        rjmp loop ; und von vorne
;
; End of source code
```

- „sbi DDRB,1“ schaltet Richtungsregister von Pin 6 auf Ausgang
- „cbi PORTB,1“ schaltet Ausgang auf Null, die LED leuchtet
- „sbi PORTB,1“ schaltet Ausgang auf Eins, die LED leuchtet nicht
- „rjmp loop“ springt wieder zurück du beginnt beim zweiten Befehl neu

- „cbi“ und „sbi“ brauchen je einen Takt, „rjmp“ braucht zwei Takte, macht zusammen vier Takte
- 1,2 MHz interner Takt durch vier ergibt eine Schaltfrequenz von

300 kHz

- Da ist beim besten Willen das An und Aus der LED nicht mehr zu erkennen.

Programmierbeispiel 04: LED an und aus mit Verzögerung

bsp04_blink_langsam.asm - Editor

Datei Bearbeiten Format ?

```
; ************************************************************
; * Mehr Action: LED-Blinker mit Verzoegerung                *
; * (C)2005 by info@avr-asm-tutorial.net                     *
; ************************************************************
;
.INCLUDE "tn13def.inc"
;
; Schaltbild:
;                    ATMEL ATtiny13
;                        ___   ___
;                    1/   |_|   |8
;+5 Volt O--|___|----|Res    Vcc|----O + 5 Volt
;           |___|    |           |
;                    |PB3    PB2|
;                    |           |6  ___
;                    |PB4    PB1|--|___|--|<|--O + 5 Volt
;                   4|           |   330   LED
;          0 Volt O----|Gnd    PB0|
;                    |___________|
;
; Benutzte Register
;
; ZH:ZL (R31:R30) ist Zaehlregister fuer Verzoegerung
;
; Programm
;
        sbi DDRB,1 ; PB1 ist Ausgang
loop:
        sbi PORTB,1 ; Ausgang auf Eins (LED ist aus)
loop1:
        sbiw ZL,1 ; Ziehe von ZH:ZL eine 1 ab
        brne loop1 ; wenn noch nicht Null, wiederhole
        cbi PORTB,1 ; Ausgang auf Null (LED ist an)
loop2:
        sbiw ZL,1 ; Ziehe von ZH:ZL eine 1 ab
        brne loop2 ; wenn noch nicht Null, wiederhole
        rjmp loop ; und das Ganze von vorne
;
; End of source code
;
```

- „sbi DDRB,1“ schaltet Richtungsregister von Pin 6 auf Ausgang
- „sbi PORTB,1“ schaltet Ausgang auf Eins, die LED ist aus
- „sbiw ZL,1“, „brne loop1“ zieht vom Inhalt des Registerpaars ZH:ZL so lange eins ab, bis es Null wird, das wiederholt sich 65535 mal
- cbi PORTB,1“ schaltet Ausgang auf Null, die LED ist an
- „sbiw ZL,1“, „brne loop1“ zieht vom Inhalt des Registerpaars ZH:ZL so lange eins ab, bis es Null wird, das wiederholt sich 65535 mal
- „rjmp loop“ springt wieder zurück und beginnt beim zweiten Befehl neu

- „cbi“ und „sbi“ brauchen je einen Takt, „rjmp“ braucht zwei Takte, die beiden Verzögerungs-schleifen brauchen zusammen 65535*4*2 = 524.280 Takte, macht zusammen 524.284 Takte
- 1,2 MHz interner Takt durch 524.284 ergibt eine LED-Schaltfrequenz von

2,29 Hz

Nachteile der Schleifenverzögerung

- Die CPU wird fast ausschließlich mit Unsinn beschäftigt (Abziehen, Prüfen, Rücksprung)
- Die CPU kann nichts anderes tun als die Schleife zu bearbeiten, sonst würde sich das Schleifen-Timing und die LED-Frequenz drastisch ändern
- Die CPU verbraucht unnötig viel Strom für das Lesen aus dem Programmspeicher und das Zählen in der Schleife

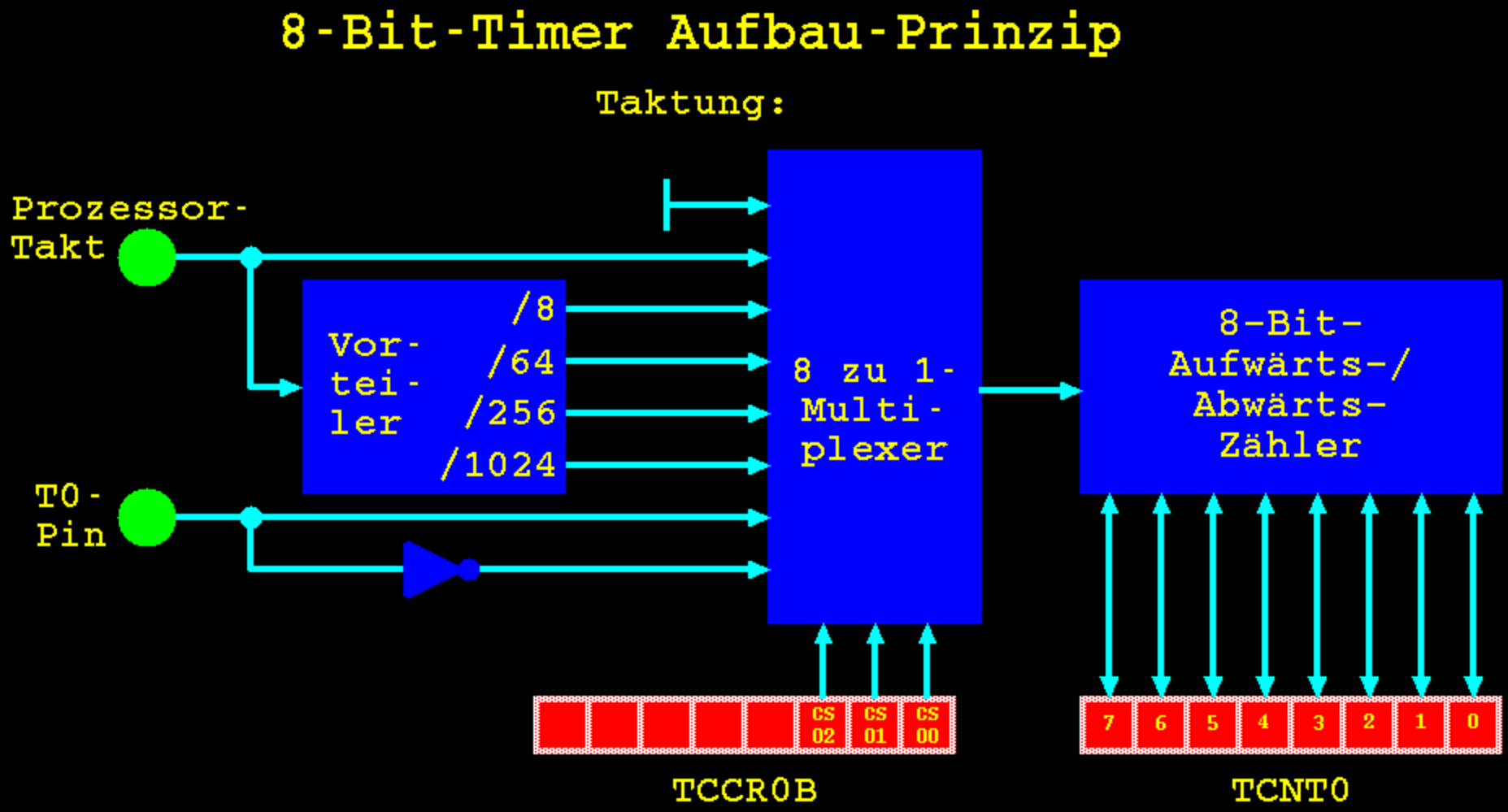
Daher:

- Verzögerungen besser mit dem eingebauten Timer.
- Möglichkeiten der Hardware soweit als möglich ausschöpfen.
- CPU schlafen legen, verbraucht dann nur noch wenig Strom.
- Timer läuft unabhängig von CPU, die kann währenddessen was anderes tun oder eben die meiste Zeit schlafen, bis es wieder Arbeit gibt.

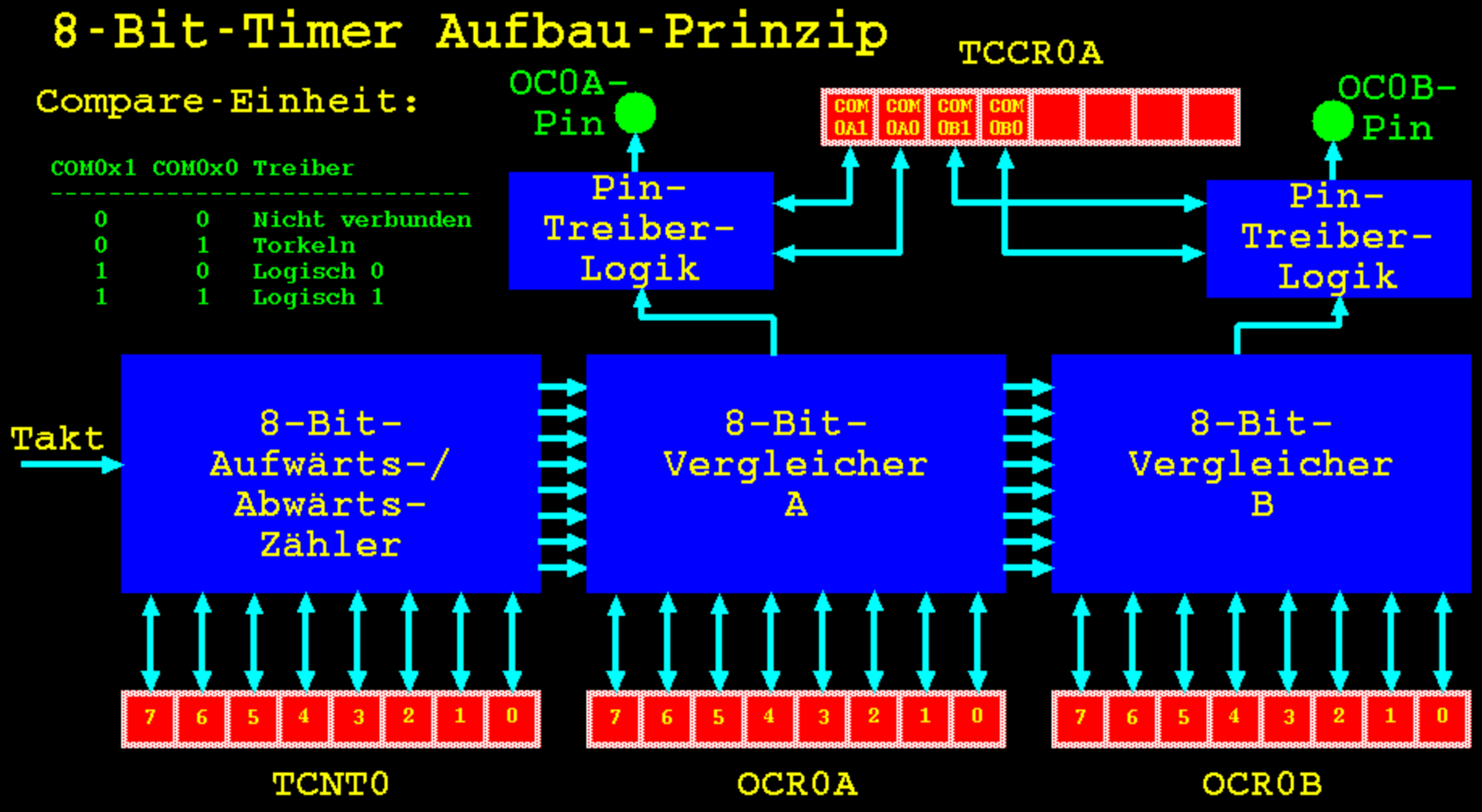
Funktionsweise von Timer/Countern

- Ein Timer ist ein Binärzähler, der von einem festen Zeittakt angetrieben wird.
- Der fester Zeittakt kann mittels eines programmierbaren Vorteilers (engl. Prescaler) vom Prozessortakt abgeleitet werden.
- Dient als Taktgeber das Signal an einem externen Pin, heißt der Timer Counter.
- Die Taktquelle, die den Timer/Counter antreibt, wird durch Schreiben eines Kontrollbytes an einen bestimmten I/O-Port ausgewählt. Der Zähler kann damit auch jederzeit von der CPU angehalten werden.
- Der Stand des Timers/Counters kann von der CPU aus jederzeit gelesen und beschrieben werden.
- Ein Überlaufen des Zählers kann für eine Unterbrechungsanforderung an die CPU verwendet werden (Timer Overflow Interrupt).
- Um die CPU von der Aufgabe zu entlasten, den Zählerstand laufend zu überwachen, sind an den Timer eine oder mehrere Vergleicher angegliedert. Sie vergleichen den Zählerstand mit einem programmierten Wert im Compare-Register und lösen bei Gleichheit (Compare Match) voreingestellte Ereignisse aus, wie z.B.:
  - Unterbrechungsanforderung an die CPU (Compare Match Interrupt),
  - Null-Setzen, Eins-Setzen oder Umkehr an einem externen Pin,
  - Rücksetzen des Timers auf Null.

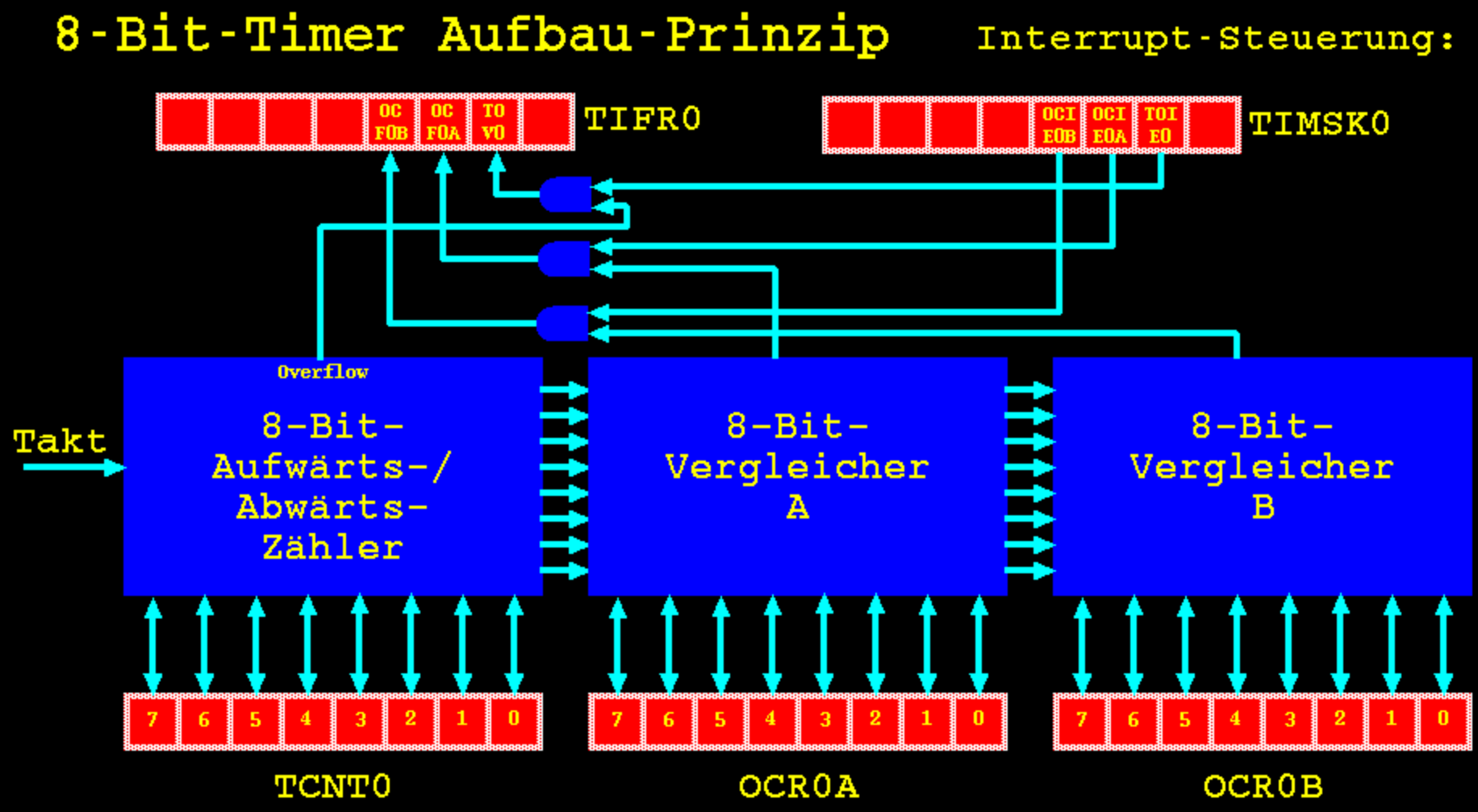
Aufbau des 8-Bit-Timers - Taktauswahl



Aufbau des 8-Bit-Timers - Vergleicher



Aufbau des 8-Bit-Timers - Interrupts



Beispiel 05: Timer steuert LED

bsp05_blink_Timer_kurz.asm.txt - Editor

Datei Bearbeiten Format ?

```
; ***************************************************************************
; * Der Timer blinkt alleine                                                *
; * (C)2005 by avr-asm-tutorial.net                                         *
; ***************************************************************************
;
.INCLUDE "tn13def.inc"
;
; Register Definitionen
;
.DEF rmp = R16 ; Multipurpose register
;
; Programmbeginn
;
; Internen Prozessortakt durch 32 teilen
; Takt = 9,6 MHz / 32 = 300 kHz
;
        ldi rmp,0b10000000 ; Vorteiler-Bit auf Eins setzen
        out CLKPR,rmp
        ldi rmp,0b00000101 ; Prozessorvorteiler auf 32 setzen
        out CLKPR,rmp
;
; PB1=OC0B als Ausgang setzen
;
        sbi DDRB,1
;
; 8-Bit-Timer mit 300 kHz Prozessortakt mit Vorteiler durch 1024
; 300 kHz / 1024 = 293 Hz, / 147 = 2 Hz, /2 = 1 Hz
;
        ldi rmp,147 ; Setze Compare A auf 147 (Ende Zaehler)
        out OCR0A,rmp
        ldi rmp,74 ; Setze Compare B auf halben Timer-Wert
        out OCR0B,rmp
        ldi rmp,0b00010010 ; CTC, toggle Ausgang B bei Compare Match
        out TCCR0A,rmp
        ldi rmp,0b00000101 ; Vorteiler durch 1024, Timer starten
        out TCCR0B,rmp
        ldi rmp,0b00100000 ; SLEEP Modus ermoeglichen
        out MCUCR,rmp
        sleep ; Prozessor schlafen legen
;
; End of source code
;
```

Zuerst wird der interne Vorteiler für den Prozessortakt auf 32 gesetzt. Dadurch läuft die CPU langsamer. Das wird durch zwei Ausgaben auf den Port CLKPR bewirkt.

Dann wird die Timerlänge auf 147 gesetzt (Schreiben in Compare-A-Port OCR0A). Der Timer setzt sich im CTC-Mode dann automatisch auf Null zurück und beginnt neu.

Durch Schreiben von 74 in den Compare-B-Port und durch das Kontrollwort in TCCR0A wird erreicht, dass bei halbem Timerwert der Ausgangspin OC0B seine Polarität wechselt (Toggle).

In Kontrollwort TCCR0B wird der Vorteiler des Zählers auf 1024 eingestellt, der Zähler zählt jetzt aufwärts.

Nach Setzen des SLEEP ENABLE Bits wird der Prozessor mit dem Befehl SLEEP schlafen gelegt, er wacht nicht wieder auf.

bsp06_lsp.asm - Editor

Datei Bearbeiten Format ?

```
;***************************************************************************
; * Ein Lautprecher macht Toene
; * (C)2005 by avr-asm-tutorial.net
;***************************************************************************
;
.INCLUDE "tn13def.inc"
;
; Schaltbild:
;                     ATMEL ATtiny13
;                      ____  ____
;                 1/   |_|     |8
;+5 Volt O--|___|----|Res    Vcc|----O + 5 Volt
;                    |         |
;                    |PB3    PB2|   ___
;                    |         |6 \_/
;                    |PB4    PB1|--|_|--O + 5 Volt
;                   4|         |   LSP
;          0 Volt O----|Gnd    PB0|
;                      |_________|
;
; Register definitions
;
.DEF rmp = R16 ; Multipurpose register
;
; Programmbeginn
;
;
; PB1=OC0B als Ausgang setzen
;
        sbi DDRB,1
        sbi DDRB,0
;
; 8-Bit-Timer mit 1,2 MHz Prozessortakt mit Vorteiler durch 8
; 1,2 MHz / 8 = 150 kHz, / 75 = 2000 Hz, /2 = 1000 Hz
;
        ldi rmp,75 ; Setze Compare A auf 147 (Ende Zaehler)
        out OCR0A,rmp
        ldi rmp,38 ; Setze Compare B auf halben Timer-Wert
        out OCR0B,rmp
        ldi rmp,0b00010010 ; CTC, toggle Ausgang B bei Compare
        out TCCR0A,rmp
        ldi rmp,0b00000010 ; Vorteiler durch 8, Timer starten
        out TCCR0B,rmp
        ldi rmp,0b00100000 ; SLEEP Modus ermoeglichen
        out MCUCR,rmp
        sleep ; Prozessor schlafen legen
;
; End of source code
;
```

Beispiel 06: NF-Erzeugung

Dasselbe Prinzip, aber anderes Timing: der Prozessortakt wird bei den voreingestellten 1,2 MHz belassen, statt der LED wird ein Piezo- oder 32-Ohm-Lautsprecher angeschlossen.

Die Timer-Werte werden so eingestellt, dass der Ausgang OC0B (PB1 an Pin 6) mit einer Frequenz von 1000 Hz seine Polarität wechselt (Vorteiler = 8, CTC bei 75, 1,2MHz / 8 / 75 = 2kHz, Polaritätswechsel am Ausgang bei Zählerstand 38, zwei Durchläufe ergeben eine volle NF-Schwingung, 2000 Hz / 2 = 1000 Hz).

Auch in diesem Beispiel wird die CPU schlafen gelegt und nicht wieder aufgeweckt.

Beispiel 07: Unterbrechung/Interrupt

- In diesem Beispiel wird demonstriert, wie ein Pegelwechsel an einem Eingangspin dazu verwendet werden kann, die schlafende CPU aufzuwecken und eine Aktion auszuführen (hier: Ein- bzw. Ausschalten einer LED).
- Der Eingangspin wird dazu mit einem internen Pull-Up-Widerstand auf +5V gezogen. Das entsprechende Portbit wird maskiert, so dass nur Pegelwechsel an diesem Pin Interrupts auslösen können.
- Wenn die Taste den Pin auf 0V zieht, wird ein Interrupt ausgelöst. Wird die Taste wieder losgelassen, löst der Pegelwechsel erneut einen Interrupt aus.
- Bei einem Interrupt legt die CPU die aktuelle Adresse auf dem Stapel im SRAM ab, verzweigt zu einer festen Adresse (Interrupt-Vektor) und bearbeitet die dortigen Befehle. Bei der Rückkehr wird die auf dem Stapel abgelegte Adresse wieder geholt und die Verarbeitung an der Adresse fortgesetzt, bei der die Unterbrechung auftrat.
- Jede Unterbrechungsmöglichkeit (beim ATtiny13 gibt es 10) hat ihren eigenen Platz in der Vektortabelle, so dass sehr schnell auf ein Ereignis reagiert werden kann.
- Weitere Unterbrechungen werden so lange unterbunden, bis die Service-Routine beendet ist. Danach kann dann die nächste Unterbrechung bearbeitet werden. Die Bearbeitung mehrer gleichzeitiger Interrupts erfolgt nach einer Prioritätsreihenfolge: je höher in der Tabelle de Vektor steht (niedrigere Adresse), desto vorrangiger.

Beispiel 07: Schaltbild

bsp07_keyint.asm - Editor

Datei Bearbeiten Format ?

```
;  ***************************************************************
;  * Angeschlossener Taster unterbricht und steuert LED an *
;  * (C)2005 by info@avr-asm-tutorial.net                        *
;  ***************************************************************
;
.INCLUDE "tn13def.inc"
;
;  Schaltbild:
;                       ATMEL ATtiny13
;                                 ____  ____
;                 ___           1/    |_|    |8
;+5 Volt o--|___|----|Res          Vcc|----o + 5 Volt
;                   |                  |
;                   |PB3          PB2  |
;                  /  3                |6  ___
;      0 Volt o--o o--|PB4          PB1  |--|___|--|<|--o + 5 Volt
;                   4|                  |   330   LED
;         0 Volt o----|Gnd          PB0  |
;                     |__________________|
;
```

Schalter an PB4 angeschlossen,

PB4 mit internem Pull-Up auf +5V

zieht Eingang auf Null Volt

LED an PB1 angeschlossen,

PB1 auf Null: LED ist an

PB1 auf Eins: LED ist aus

Beispiel 07: Interrupt Vektoren

bsp07_keyint.asm - Editor

Datei Bearbeiten Format ?

```
;
; Register Definitionen
;
.DEF rmp = R16 ; Multipurpose register
.DEF rimp = R17 ; Interrupt multipurpose register
;
; Constants
;
;
; Reset- und Interrupt-Vektoren
;
.CSEG ; Assembliere in den Programm-Flash-Speicher
.ORG $0000 ; beginne mit Adresse 0
;
; Sprungvektoren fuer Reset und Interrupts
;
        rjmp main ; Reset vector
        reti ; Int0 interrupt vector
        rjmp intpcint ; PCINT0 vector
        reti ; TC0 overflow vector
        reti ; Eeprom ready vector
        reti ; Analog comparator int vector
        reti ; TC0 CompA vector
        reti ; TC0 CompB vector
        reti ; WDT vector
        reti ; ADC conversion complete vector
;
; PCINT0 Service Routine
;   wird jedes Mal ausgefuehrt, wenn sich der Pegel am
;   Pin 3 (=PB4) aendert
;
intpcint:
        sbic PINB,4 ; Ueberspringe Befehl wenn PB4 Null
        rjmp intpcint1 ; springe weil PB4 = Eins
        cbi PORTB,1 ; schalte LED an
        reti ; Kehre vom Interrupt zurueck
intpcint1:
        sbi PORTB,1 ; schalte LED aus
        reti ; Kehre vom Interrupt zurueck
```

Register definieren: symbolische Namen für bestimmte Register festlegen.

Die Interrupt-Vektor-Tabelle des ATtiny13: Der Befehl an der Adresse 0000 wird beim Start ausgeführt (Sprung zum Label main:). Der Interrupt INT0 wird hier nicht benutzt, an Adresse 0001 erfolgt ein einfacher Rücksprung vom Interrupt (*reti*). Bei einem Pegelwechsel an Pin 3 wird der PCINT0 Interrupt an Adresse 0002 angesprungen, der nach Label intpcint: verzweigt. Alle anderen Vektoren werden nicht benutzt.

Die Interupt-Service-Routine intpcint: stellt als erstes fest, ob der Tasteneingang Null der eins ist. Bei Null wird der rjmp nicht ausgeführt.

Ist er Null, wird die LED angeschaltet und die Routine beendet (*reti*).

Ist er Eins, wird die LED ausgeschaltet und die Routine beendet (*reti*).

Beispiel 07: Anstoßen des Ganzen (Init)

bsp07_keyint.asm - Editor

Datei Bearbeiten Format ?

```
; Main program start
;
main:
;
; Stapelzeiger setzen fuer Rueckkehr-Adressen vom Interrupt
;
        ldi rmp,LOW(RAMEND) ; Stapelzeiger auf Ende SRAM
        out SPL,rmp
;
; Angeschlossene Hardware initiieren
;
        sbi DDRB,1 ; LED-Ausgang als Ausgang definieren
        sbi PORTB,1 ; LED ausschalten
        cbi DDRB,4 ; Taster-Eingang als Eingang definieren
        sbi PORTB,4 ; Internen Pull-Up-Widerstand einschalten
;
; Pin-Change-Interrupt fuer Taste aktivieren
;
        ldi rmp,0b00010000 ; Maskieren der aktiven Eingaenge
        out PCMSK,rmp
        ldi rmp,0b00100000 ; PCINT0-Interrupts ermoeglichen
        out GIMSK,rmp
;
; Interrupts generell einschalten
;
        sei ; Setze Interrupt Flagge
;
; Schlafmodus der CPU einstellen
;
        ldi rmp,0b00100000 ; Schlafen ermoeglichen, Modus Idle
        out MCUCR,rmp
;
; Loop mit Interrupt
;
loop:
        sleep ; Prozessor schlafen legen
        nop ; Tue nichts nach dem Aufwachen
        rjmp loop ; Prozessor wieder schlafen legen
;
; Ende Quellcode
;
```

Hauptprogramm, wird beim Reset ausgeführt (Adresse 0000: Sprungbefehl nach main:).

Der „Stapel“ im SRAM wird dazu benutzt, um beim Interrupt die Rücksprungadresse dort abzulegen und bei *reti* wieder dorthin zu springen.

Der LED-Ausgang wird initiiert.

Der Tasten-Eingang wird definiert und der Pull-Up-Widerstand eingeschaltet.

Die Maske in PCMSK wählt den Eingang aus, bei dem Interrupt erfolgen soll (Bit 4).

Im Int-Maskenregister GIMSK wird der PCINT0-Interrupt ermöglicht.

Im Statusregister der CPU wird das Interrupt-Bit gesetzt, die CPU lässt Ints zu.

Das SLEEP ENABLE Bit wird gesetzt.

Die CPU wird mit SLEEP schlafen gelegt, wacht beim Interrupt auf, führt ihn aus, und wird anschließend wieder zu Bett gebracht.

Beispiel 07: Testaufbau