www.mikrocontroller.net

AVR-Tutorial: PWM

PWM - Dieses Kürzel steht für **P**uls **W**eiten **M**odulation.

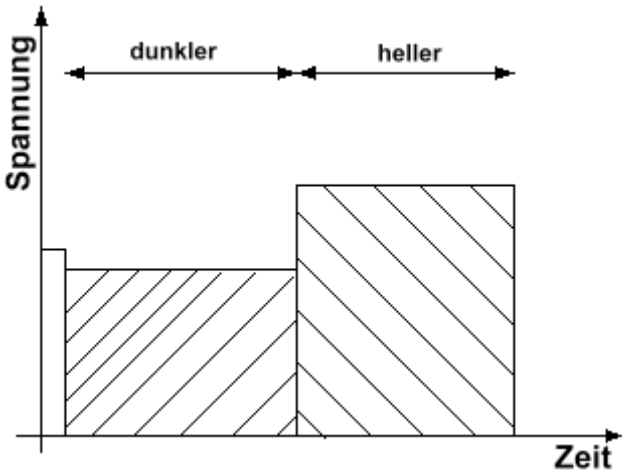
Inhaltsverzeichnis

[Verbergen]

Was bedeutet PWM?

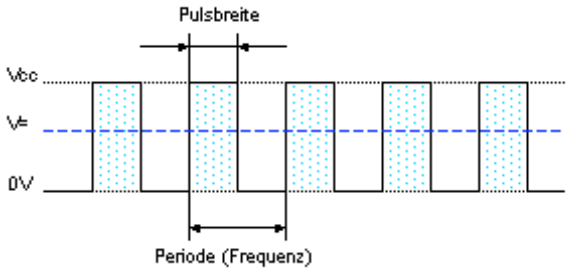
Viele elektrische Verbraucher können dadurch in ihrer Leistung reguliert werden, indem die Versorgungsspannung in weiten Bereichen verändert wird. Ein normaler Gleichstrommotor wird z.B. langsamer laufen, wenn er mit einer geringeren Spannung versorgt wird, bzw. schneller laufen, wenn er mit einer höheren Spannung versorgt wird. LEDs werden zwar nicht mit einer Spannung gedimmt, sondern mit dem Versorgungsstrom. Da dieser Stromfluss aber im Normalfall mit einem Vorwiderstand eingestellt wird, ist durch das Ohmsche Gesetz dieser Stromfluss bei konstantem Widerstand wieder direkt proportional zur Höhe der Versorgungsspannung.

Im wesentlichen geht es also immer um diese Kennlinie, trägt man die Versorgungsspannung entlang der Zeitachse auf:



Die Fläche unter der Kurve ist dabei ein direktes Mass für die Energie die dem System zugeführt wird. Bei geringerer Energie ist die Helligkeit geringer, bei höherer Energie entsprechend heller.

Jedoch gibt es noch einen zweiten Weg, die dem System zugeführte Energie zu verringern. Anstatt die Spannung abzusenken, ist es auch möglich die volle Versorgungsspannung über einen geringeren Zeitraum anzulegen. Man muß nur dafür Sorge tragen, daß im Endeffekt die einzelnen Pulse nicht mehr wahrnehmbar sind.



Die Fläche unter den Rechtecken hat in diesem Fall dieselbe Größe wie die Fläche unter der Spannung V=, glättet man die Spannung also mit einem Kondensator, ergibt sich eine niedrigere konstante Spannung. Die Rechtecke sind zwar höher, aber dafür schmäler. Die Flächen sind aber dieselben. Diese Lösung hat den Vorteil, dass keine Spannung geregelt werden muss, sondern der Verbraucher immer mit derselben Spannung versorgt wird.

Und genau das ist das Prinzip einer PWM. Durch die Abgabe von Pulsen wird die abgegebene Energiemenge gesteuert. Es ist auf einem μC wesentlich einfacher Pulse mit einem definiertem Puls/Pausen Verhältnis zu erzeugen als eine Spannung zu variieren.

PWM und der Timer

Der Timer1 des Mega8 unterstützt direkt das Erzeugen von PWM. Beginnt der Timer beispielsweise bei 0 zu zählen, so schaltet er gleichzeitig einen Ausgangspin ein. Erreicht der Zähler einen bestimmten Wert X, so schaltet er den Ausgangspin wieder aus und zählt weiter bis zu seiner Obergrenze. Danach wiederholt sich das Spielchen, der Timer beginnt wieder bei 0 und schaltet gleichzeitig den Ausgangspin ein, etc., etc. Durch verändern von X kann man daher steuern, wie lange der Ausgangspin, im Verhältnis zur kompletten Zeit die der Timer benötigt um seine Obergrenze zu erreichen, eingeschaltet ist.

Dabei gibt es aber verwirrenderweise verschiedene Arten der PWM:

- Fast PWM
- Phasen-korrekte PWM
- Phasen- und Frequenzkorrekte PWM

Für die Details zu jedem PWM-Modus sei auf das Datenblatt verwiesen.

Fast PWM

Die Fast PWM gibt es beim Mega8 mit mehreren unterschiedlichen Bit-Zahlen. Bei den Bit-Zahlen geht es immer darum, wie weit der Timer zählt, bevor ein Rücksetzen des Timers auf 0 erfolgt

- Modus 5: 8 Bit Fast PWM - Der Timer zählt bis 255
- Modus 6: 9 Bit Fast PWM - Der Timer zählt bis 511

- Modus 7: 10 Bit Fast PWM - Der Timer zählt bis 1023
- Modus 14: Fast PWM mit beliebiger Schrittzahl (festgelegt durch **ICR1**)
- Modus 15: Fast PWM mit beliebiger Schrittzahl (festgelegt durch **OCR1A**)

Grundsätzlich funktioniert der Fast-PWM Modus so, dass der Timer bei 0 anfängt zu zählen, wobei natürlich der eingestellte Vorteiler des Timers berücksichtigt wird. Erreicht der Timer einen bestimmten Zählerstand (festgelegt durch die Register **OCR1A** und **OCR1B**) so wird eine Aktion ausgelöst. Je nach Festlegung kann der entsprechende μC Pin (OC1A und OC1B) entweder

- umgeschaltet
- auf 1 gesetzt
- auf 0 gesetzt

werden. Wird der OC1A/OC1B Pin so konfiguriert, dass er auf 1 oder 0 gesetzt wird, so wird automatisch der entsprechende Pin beim Timerstand 0 auf den jeweils gegenteiligen Wert gesetzt.

Der OC1A Pin befindet sich beim Mega8 am Port B, konkret am Pin **PB1**. Dieser Pin muss über das zugehörige Datenrichtungsregister **DDRB** auf Ausgang gestellt werden. Anders als beim UART geschieht dies nicht automatisch.

Das Beispiel zeigt den Modus 14. Dabei wird der Timer-Endstand durch das Register **ICR1** festgelegt. Weiters wird die Funktion des OC1A Pins so festgelegt, dass der Pin bei einem Timer Wert von 0 auf 1 gesetzt wird und bei Erreichen des im **OCR1A** Registers festgelegten Wertes auf 0 gesetzt wird. Der Vorteiler des Timers, bzw. der ICR-Wert wird zunächst so eingestellt, dass eine an **PB1** angeschlossene LED noch blinkt, die Auswirkungen unterschiedlicher Register Werte gut beobachtet werden können. Den Vorteiler zu verringern ist kein Problem, hier geht es aber darum, zu demonstrieren wie PWM funktioniert.

Hinweis: Wie überall im ATMega8 ist darauf zu achten, dass beim Beschreiben eines 16-Bit Registers zuerst das High-Byte und dann das Low-Byte geschrieben wird.

```
.include "m8def.inc"

.def temp1          = r17

.equ XTAL = 4000000

    rjmp    init

;.include "keys.asm"
;
;

init:
    ldi     temp1, LOW(RAMEND)      ; Stackpointer initialisieren
    out     SPL, temp1
    ldi     temp1, HIGH(RAMEND)
    out     SPH, temp1

    ;
    ; Timer 1 einstellen
    ;
    ; Modus 14:
    ;    Fast PWM, Top von ICR1
    ;
    ;     WGM13    WGM12   WGM11    WGM10
    ;      1        1       1        0
    ;
    ;    Timer Vorteiler: 256
```

```
    ;     CS12     CS11    CS10
    ;      1        0       0
    ;
    ; Steuerung des Ausgangsport: Set at BOTTOM, Clear at match
    ;     COM1A1   COM1A0
    ;      1        0
    ;

    ldi      temp1, 1<<COM1A1 | 1<<WGM11
    out      TCCR1A, temp1

    ldi      temp1, 1<<WGM13 | 1<<WGM12 | 1<<CS12
    out      TCCR1B, temp1

    ;
    ; den Endwert (TOP) für den Zähler setzen
    ; der Zähler zählt bis zu diesem Wert
    ;
    ldi      temp1, 0x6F
    out      ICR1H, temp1
    ldi      temp1, 0xFF
    out      ICR1L, temp1

    ;
    ; der Compare Wert
    ; Wenn der Zähler diesen Wert erreicht, wird mit
    ; obiger Konfiguration der OC1A Ausgang abgeschaltet
    ; Sobald der Zähler wieder bei 0 startet, wird der
    ; Ausgang wieder auf 1 gesetzt
    ;
    ldi      temp1, 0x3F
    out      OCR1AH, temp1
    ldi      temp1, 0xFF
    out      OCR1AL, temp1

    ; Den Pin OC1A zu guter letzt noch auf Ausgang schalten
    ldi      temp1, 0x02
    out      DDRB, temp1

main:
    rjmp     main
```

Wird dieses Programm laufen gelassen, dann ergibt sich eine blinkende LED. Die LED ist die Hälfte der Blinkzeit an und in der anderen Hälfte des Blinkzyklus aus. Wird der Compare Wert in **OCR1A** verändert, so lässt sich das Verhältnis von LED Einzeit zu Auszeit verändern. Ist die LED wie im I/O Kapitel angeschlossen, so führen höhere **OCR1A** Werte dazu, dass die LED nur kurz aufblitzt und in der restlichen Zeit dunkel bleibt.

```
    ldi      temp1, 0x6D
    out      OCR1AH, temp1
    ldi      temp1, 0xFF
    out      OCR1AL, temp1
```

Sinngemäß führen kleinere **OCR1A** Werte dazu, daß die LED länger leuchtet und die Dunkelphasen kürzer werden.

```
    ldi      temp1, 0x10
    out      OCR1AH, temp1
    ldi      temp1, 0xFF
    out      OCR1AL, temp1
```

Nachdem die Funktion und das Zusammenspiel der einzelnen Register jetzt klar ist, ist es Zeit aus dem Blinken ein echtes Dimmen zu machen. Dazu genügt es den Vorteiler des Timers auf 1 zu setzen:

```
ldi     temp1, 1<<WGM13 | 1<<WGM12 | 1<<CS10
out     TCCR1B, temp1
```

Werden wieder die beiden **OCR1A** Werte 0x6DFF und 0x10FF ausprobiert, so ist deutlich zu sehen, dass die LED scheinbar unterschiedlich hell leuchtet. Dies ist allerdings eine optische Täuschung. Die LED blinkt nach wie vor, nur blinkt sie so schnell, daß dies für uns nicht mehr wahrnehmbar ist. Durch Variation der Einschalt- zu Ausschaltzeit kann die LED auf viele verschiedene Helligkeitswerte eingestellt werden.

Theoretisch wäre es möglich die LED auf 0x6FFF verschiedene Helligkeitswerte einzustellen. Dies deshalb, weil in **ICR1** genau dieser Wert als Endwert für den Timer festgelegt worden ist. Dieser Wert könnte genauso gut kleiner oder größer eingestellt werden. Um eine LED zu dimmen ist der Maximalwert aber hoffnungslos zu hoch. Für diese Aufgabe reicht eine Abstufung von 256 oder 512 Stufen normalerweise völlig aus. Genau für diese Fälle gibt es die anderen Modi. Anstatt den Timer Endstand mittels **ICR1** festzulegen, genügt es den Timer einfach nur in den 8, 9 oder 10 Bit Modus zu konfigurieren und damit eine PWM mit 256 (8 Bit), 512 (9 Bit) oder 1024 (10 Bit) Stufen zu erzeugen.

Phasen-korrekte PWM

Phasen- und Frequenz-korrekte PWM

PWM in Software

Die Realisierung einer PWM mit einem Timer, wobei der Timer die ganze Arbeit macht, ist zwar einfach, hat aber einen Nachteil. Für jede einzelne PWM ist ein eigener Timer notwendig. Und davon gibt es in einem Mega8 nicht all zu viele.

Es geht auch anders: Es ist durchaus möglich viele PWM Stufen mit nur einem Timer zu realisieren. Der Timer wird nur noch dazu benötigt, eine stabile und konstante Zeitbasis zu erhalten. Von dieser Zeitbasis wird alles weitere abgeleitet.

Prinzip

Das Grundprinzip ist dabei sehr einfach: Eine PWM ist ja im Grunde nichts anderes als eine Blinkschleife, bei der das Verhältnis von Ein- zu Auszeit variabel eingestellt werden kann. Die Blinkfrequenz selbst ist konstant und ist so schnell, dass das eigentliche Blinken nicht mehr wahrgenommen werden kann. Das lässt sich aber auch alles in einer ISR realisieren:

- Ein Timer (Timer0) wird so aufgesetzt, dass er eine Overflow-Interruptfunktion (ISR) mit dem 256-fache der gewünschten Blinkfrequenz aufruft.
- In der ISR wird ein weiterer Zähler betrieben (*PWMCounter*), der ständig von 0 bis 255 zählt.
- Für jede zu realisierende PWM Stufe gibt es einen Grenzwert. Liegt der Wert des PWMCounters unter diesem Wert, so wird der entsprechende Port Pin eingeschaltet. Liegt er darüber, so wird der entsprechende Port Pin ausgeschaltet

Damit wird im Grunde nichts anderes gemacht, als die Funktionalität der Fast-PWM in Software nachzubilden. Da man dabei aber nicht auf ein einziges OCR Register angewiesen ist, sondern in gewissen Umfang beliebig viele davon implementieren kann, kann man auch beliebig viele PWM Stufen erzeugen.

Programm

Am **Port B** werden an den Pins **PB0** bis **PB5** insgesamt 6 LEDs gemäß der Verschaltung aus dem I/O Artikel angeschlossen. Jede einzelne LED kann durch setzen eines Wertes von 0 bis 127 in die zugehörigen Register *ocr_1* bis *ocr_6* auf einen anderen Helligkeitswert eingestellt werden. Die PWM-Frequenz (Blinkfrequenz) jeder LED beträgt: (4000000 / 256) / 127 = 123Hz. Dies reicht aus um das Blinken unter die Wahrnehmungsschwelle zu drücken und die LEDs gleichmässig erleuchtet erscheinen zu lassen.

```
.include "m8def.inc"

.def temp  = r16

.def PWMCount = r17

.def ocr_1 = r18                        ; Helligkeitswert Led1: 0 .. 127
.def ocr_2 = r19                        ; Helligkeitswert Led2: 0 .. 127
.def ocr_3 = r20                        ; Helligkeitswert Led3: 0 .. 127
.def ocr_4 = r21                        ; Helligkeitswert Led4: 0 .. 127
.def ocr_5 = r22                        ; Helligkeitswert Led5: 0 .. 127
.def ocr_6 = r23                        ; Helligkeitswert Led6: 0 .. 127

.org 0x0000
        rjmp    main                    ; Reset Handler
.org OVF0addr
        rjmp    timer0_overflow         ; Timer Overflow Handler

main:
        ldi     temp, LOW(RAMEND)       ; Stackpointer initialisieren
        out     SPL, temp
        ldi     temp, HIGH(RAMEND)
        out     SPH, temp

        ldi     temp, 0xFF              ; Port B auf Ausgang
        out     DDRB, temp

        ldi     ocr_1, 0
        ldi     ocr_2, 1
        ldi     ocr_3, 10
        ldi     ocr_4, 20
        ldi     ocr_5, 80
        ldi     ocr_6, 127

        ldi     temp, 0b00000001        ; CS00 setzen: Teiler 1
        out     TCCR0, temp

        ldi     temp, 0b00000001        ; TOIE0: Interrupt bei Timer Overflow
        out     TIMSK, temp

        sei

loop:   rjmp    loop

timer0_overflow:                        ; Timer 0 Overflow Handler
        inc     PWMCount                ; den PWM Zähler von 0 bis
        cpi     PWMCount, 128           ; 127 zählen lassen
        brne    WorkPWM
        clr     PWMCount

WorkPWM:
        ldi     temp, 0b11000000        ; 0 .. Led an, 1 .. Led aus
```

```
        cp      PWMCount, ocr_1         ; Ist der Grenzwert für Led 1 erreicht
        brlt    OneOn
        ori     temp, $01

OneOn:  cp      PWMCount, ocr_2         ; Ist der Grenzwert für Led 2 erreicht
        brlt    TwoOn
        ori     temp, $02

TwoOn:  cp      PWMCount, ocr_3         ; Ist der Grenzwert für Led 3 erreicht
        brlt    ThreeOn
        ori     temp, $04

ThreeOn:cp      PWMCount, ocr_4         ; Ist der Grenzwert für Led 4 erreicht
        brlt    FourOn
        ori     temp, $08

FourOn: cp      PWMCount, ocr_5         ; Ist der Grenzwert für Led 5 erreicht
        brlt    FiveOn
        ori     temp, $10

FiveOn: cp      PWMCount, ocr_6         ; Ist der Grenzwert für Led 6 erreicht
        brlt    SetBits
        ori     temp, $20

SetBits:                                ; Die neue Bitbelegung am Port ausgeben
        out     PORTB, temp

        reti
```

Würde man die LEDs anstatt direkt an ein Port anzuschliessen, über ein oder mehrere Schieberegister anschließen, so kann auf diese Art eine relativ große Anzahl an LEDs gedimmt werden. Natürlich müsste man die softwareseitige LED Ansteuerung gegenüber der hier gezeigten verändern, aber das PWM Prinzip könnte so übernommen werden.

Siehe auch

- PWM
- AVR-GCC-Tutorial: PWM
- Soft-PWM - optimierte Software-PWM in C
- LED-Fading - LED dimmen mit PWM

Zurück zu Tasten | Hoch zu Inhaltsverzeichnis | Vor zu Schieberegister

Von „http://www.mikrocontroller.net/articles/AVR-Tutorial:_PWM“

Kategorien: AVR | AVR-Tutorial