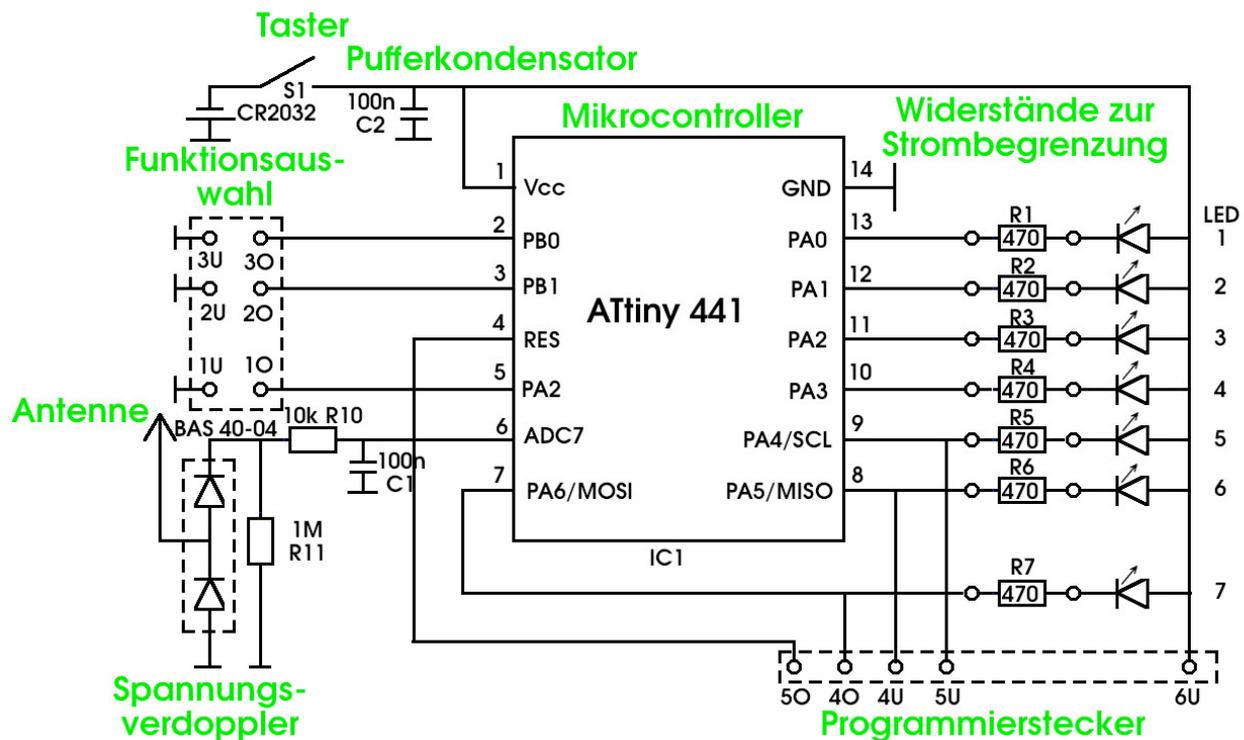


1. Das Schaltbild



1.1 Die Bestandteile unserer Schaltung

Damit sich in so einer Schaltung überhaupt was tut, braucht es natürlich eine Energiequelle. In unserem Fall ist das eine kleine Batterie.

So eine Batterie hält natürlich nur eine begrenzte Zeit. Die Zeit lässt sich aber kräftig verlängern, wenn man dafür sorgt, dass das Gerät nicht dauernd läuft. Es braucht also etwas, um das Gerät ein- und ausschalten zu können. Dafür haben wir einen kleinen Taster spendiert. Ist der gedrückt, ist das Platinchen in Betrieb, lässt man den Taster los, wird die Verbindung zum Mikrocontroller unterbrochen und die Batterie wird nicht weiter belastet.

Ist der Mikrocontroller über den Taster mit der Batterie verbunden, zieht er nicht gleichmäßig Strom, sondern für ganz kurze Zeiten mehr und danach für längere Zeiten deutlich weniger Strom. Solche Stromspitzen kann die Batterie aber nicht liefern. Um das Problem zu beseitigen, wurde ein Pufferkondensator eingebaut. Das ist ein Bauelement, das wie ein winziger Akku keine Mengen von Energie speichern und sehr schnell wieder abgeben kann. Braucht der Mikrocontroller also sehr schnell Strom, so kommt der aus dem Kondensator, die Batterie lädt den Kondensator danach langsam wieder auf.

Der Mikrocontroller ist das zentrale Bauelement in unserer Schaltung. Dieses kleine Wunderwerk kann heutzutage das, wofür man noch vor wenigen Jahren einen großen Rechner gebraucht hat. Der winzige Chip enthält

- einen Programmspeicher, in dem die Anweisungen stehen, was er zu tun hat,

- einen Stapel von Merzzellen für seine Rechenarbeit (einen sogenannten RAM),
- Merzzellen, die ihren Inhalt auch dann noch behalten, wenn der Strom abgeschaltet wird - also ein Langzeitgedächtnis, das als EEPROM bezeichnet wird,
- ein Rechenwerk, das Zahlen zusammenzählen, voneinander abziehen, mit einander multiplizieren und mit einer Menge anderer logischer Rechenoperationen umrechnen kann.
- Stoppuhren (auch Timer genannt),
- elektronische Schalter, um die Ergebnisse an den Anschlüssen sichtbar zu machen,
- ein Thermometer,
- Messgeräte für Spannungen (A/D-Wandler), mit dem an Eingängen angelegte Spannungen gemessen, also in Zahlenwerte umgewandelt werden und
- einen „Wachhund“ (Watchdog), der darüber wacht, dass das Programm nicht in einer logischen Endlosschleife hängen bleibt.

Mit den oben schon erwähnten elektronischen Schaltern können die Ausgänge des Mikrocontrollers und damit die angeschlossenen Leuchtdioden ein- und ausgeschaltet werden. Würde man die Leuchtdioden allerdings direkt an den Controller anschließen, würden sie nur einmal kurz aufblitzen, um dann durchzubrennen. Auch der Mikrocontroller würde damit sein Leben aushauchen. Um das zu vermeiden, braucht es etwas, was dafür sorgt, dass der Strom in den gewünschten Grenzen bleibt. Dafür sind sieben Widerstände da, die wir gemeinsam eingelötet haben.

Zusätzlich zu den oben beschriebenen Bauelementen wurde noch eine kleine Schaltung auf die Platine gesetzt, mit der man Radiosignale (also hochfrequente) Strahlung empfangen kann. Diese Schaltung besteht aus einer kleinen Antenne (einer Leiterbahn auf der spitzen Seite der Platine), zwei Dioden (das sind Bauelemente, die Strom nur in einer Richtung durchlassen), zwei Widerständen und einem Kondensator.

Mit der Antenne wird das Signal eingefangen, die beiden Dioden dienen als Spannungsverdoppler und machen die Schaltung damit empfindlicher. Mit dem Kondensator wird die Spannung für die Messung im Mikrocontroller lange genug gehalten und die Widerstände lassen die Ladung aus dem Kondensator wieder langsam abfließen.

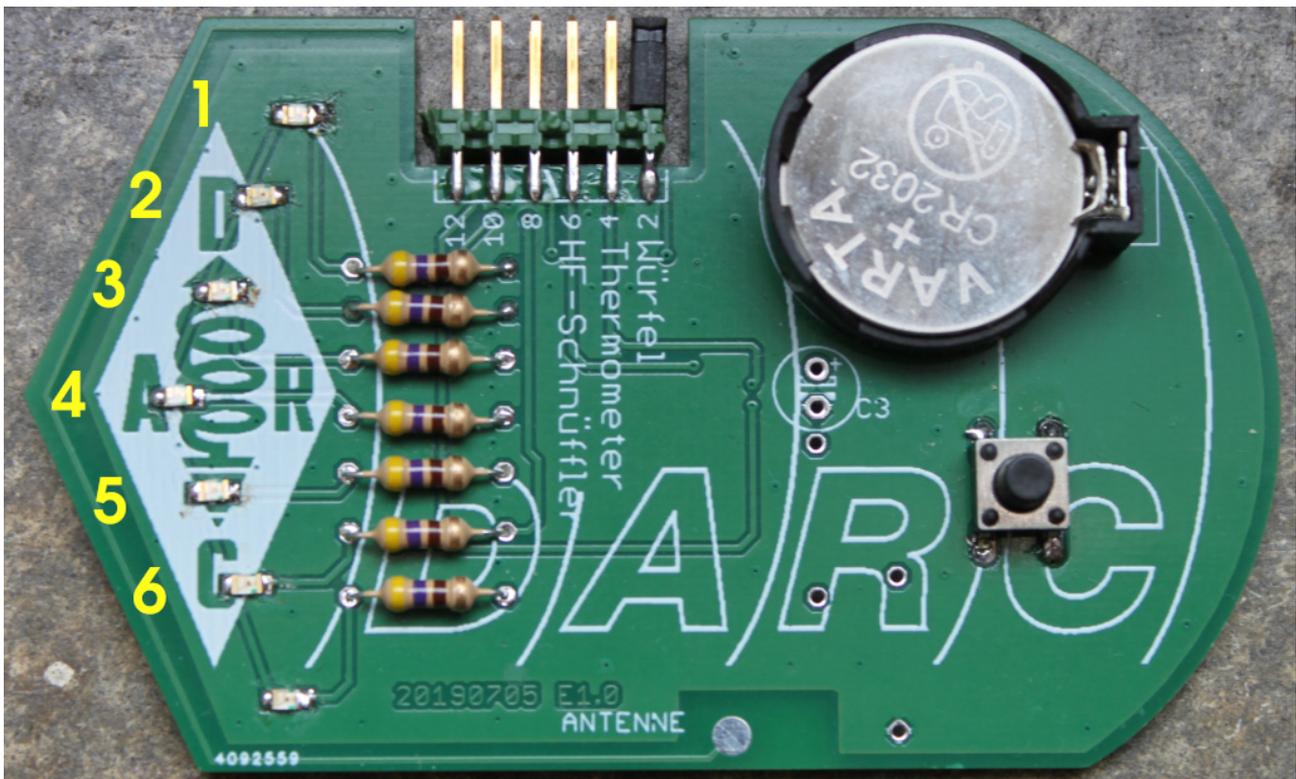
Schließlich fehlt noch etwas, um dem Mikrocontroller mitteilen zu können, welche seiner ausprogrammierten Funktionen er überhaupt ausführen soll. Dazu gibt es eine 12-polige Stiftleiste mit einer Steckbrücke. Wenn der Mikrocontroller anläuft, „sieht er zuerst nach“, auf welchen Stiften die Steckbrücke steckt und startet danach die Funktion, die der Position der Steckbrücke entspricht. Eigentlich bräuchte es für unsere vier unterschiedlichen Funktionen nur 6 Stifte, es sind aber zwölf. Die restlichen wurden dafür benutzt, um das Programm vor dem ersten Start in den Controller zu laden. Über diese Stifte könnte das Programm auch jederzeit gegen ein anderes ausgetauscht werden.

2. Die Funktionen

Für unsere Platine wurden drei verschiedene Funktionen ausprogrammiert:

- Würfel,
- Thermometer (in zwei Varianten) und
- HF-Schnüffler.

Auswählen kann man die jeweilige Funktion, wenn man die Brücke auf eine der drei bezeichneten Positionen steckt und danach die Taste drückt. Je nach Funktion haben die LEDs unterschiedliche Bedeutungen. Welche das sind, ist in den nächsten Punkten genauer beschrieben.

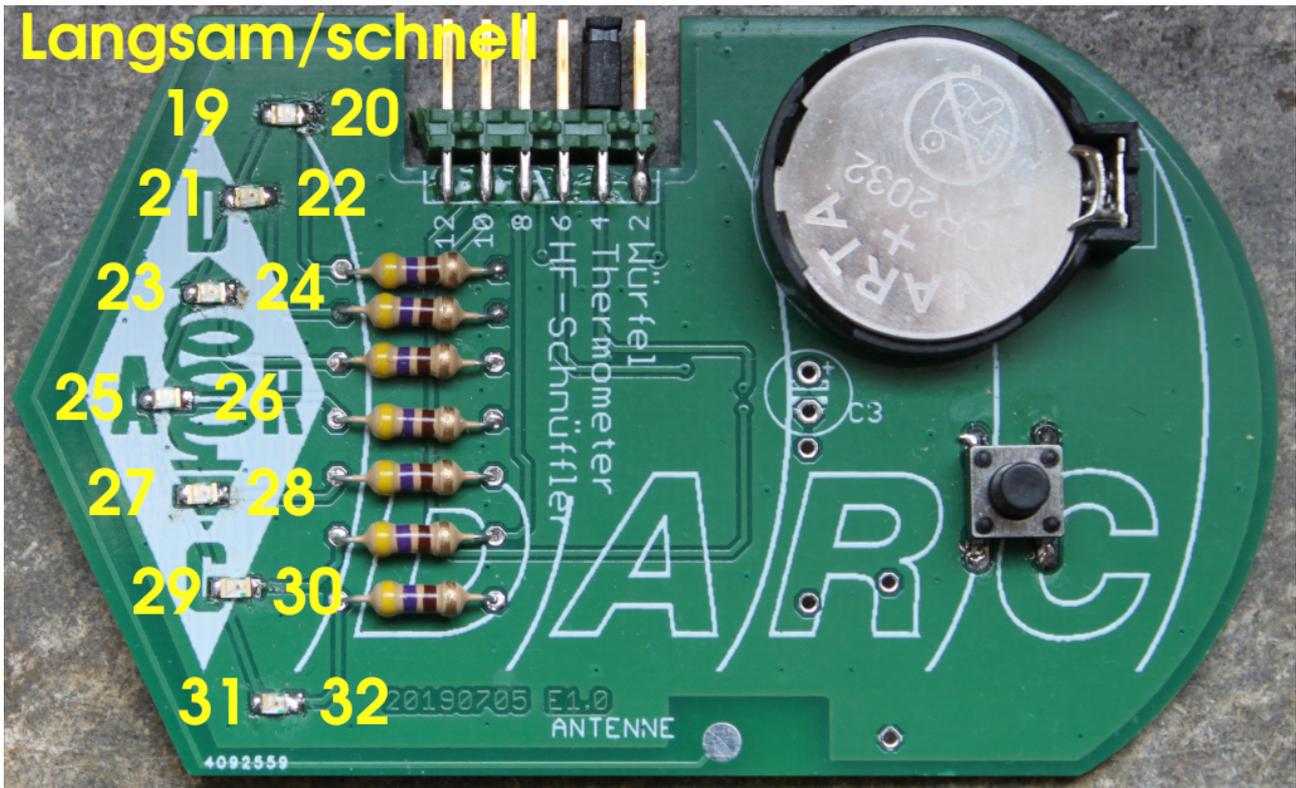


2.1 Der Würfel

Wird die Steckbrücke auf die Position 2 gesteckt und der Taster gedrückt, beginnt eine der oberen 6 Leuchtdioden zu blinken. Welche das sein wird, hängt vom Zufall ab und ist, wie beim klassischen Würfel auch, nicht vorhersagbar.

Technisch wurde das im Prinzip so realisiert, dass beim Drücken der Taste die Temperatur 100 mal hintereinander gemessen wird und die Ergebnisse aufaddiert werden. Die so entstandene Zahl wird dann durch 6 geteilt und der Divisionsrest ausgegeben. Dabei wird ausgenutzt, dass die Temperaturmessungen immer mit etwas Rauschen behaftet sind. Eben dieses Rauschen (also zufällig höhere oder kleinere Werte) wird durch das beschriebene Verfahren um den Faktor 100 verstärkt und kann dann so für den Würfel verwendet werden.

2.2 Das Thermometer



Wird die Steckbrücke auf die Position 4 „Thermometer“ gesteckt, dann wird beim nächsten Tastendruck die prozessorintern gemessene Temperatur ausgegeben.

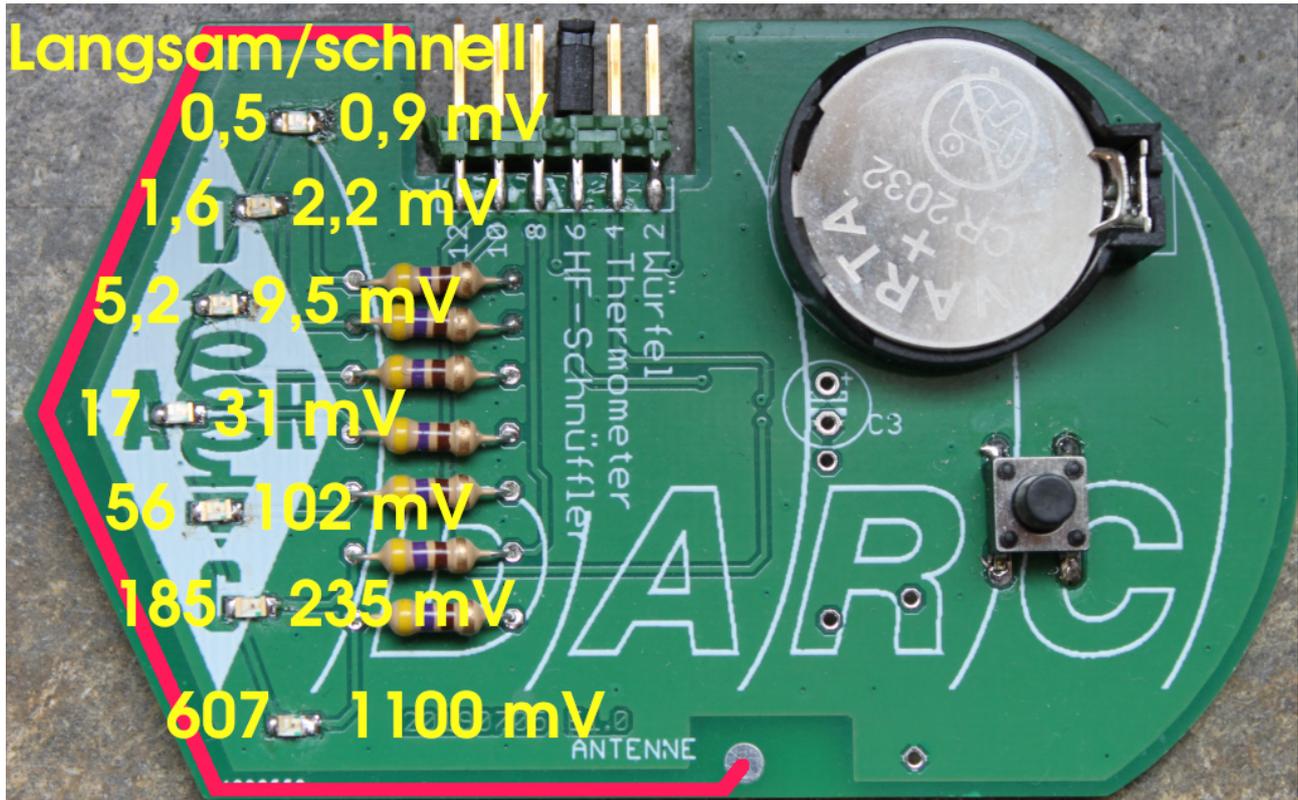
Um mit nur 7 Leuchtdioden mehr als 7 unterschiedliche Temperaturen anzeigen zu können, werden für die LEDs verschiedene Blinktakte benutzt. Damit können dann Werte zwischen 19 Grad und 32 Grad angezeigt werden. Welche Temperatur gemeint ist, wenn eine der LEDs langsam oder schnell blinkt, kann man aus dem Bild oben erkennen. Für Werte außerhalb des 19 Grad bis 32 Grad-Bereiches gibt es einige Besonderheiten:

- Bleiben alle LEDs aus, so liegt die Temperatur bei genau 18 Grad.
- Blinken die obere und die untere LED gleichzeitig, liegt die Temperatur unter 18 Grad.
- Blinken die obere, die mittlere und die untere, liegt die Temperatur über 32 Grad.

Will man die Temperaturwerte unter 18 Grad oder über 32 Grad genauer wissen, kann man eine Sonderfunktion der Thermometers nutzen, die weiter unten beschrieben wird.

Ein Hinweis vielleicht noch: Der Temperatursensor befindet sich im Mikrocontroller selbst. Das ist der kleine 14-beinige Schaltkreis auf der Unterseite der Platine. Wenn man eine sinnvolle Temperaturmessung machen möchte, sollte man drauf achten, dass man den Controllerchip nicht vorher mit den Fingern berührt und damit erwärmt hat.

2.3 Der HF-Schnüffler



Rücksichtslosigkeit ist etwas, was in den letzten Jahren immer mehr zugenommen hat. Leider betrifft das auch die Störsignale, die elektronische Geräte abstrahlen und damit andere Gerät aus dem Takt bringen. Ganz vorne dran sind dabei Netzteile z.B. für LED-Beleuchtungen, in denen man die eigentlich vorgeschriebenen Entstör-Maßnahmen geflissentlich vergisst. Ganz besonders tun sich dabei besonders billige, fernöstliche Produkte hervor. Dazu kommt noch eine fast schon kriminelle Idee: Für die Übertragung von Telefon, Radio oder Fernsehen keine eigenen Kabel zu verwenden, sondern stattdessen die Steckdose, also die Netzleitungen zu missbrauchen. Netzleitungen sind dafür nun mal nicht gemacht und für Hochfrequenz sowas wie Schläuche die fast nur aus Löchern bestehen. Die so missbrauchten Netzleitungen strahlen dann hochfrequente Störsignale ab, dass sich die Balken biegen.

Solche „HF-Strahlungsünder“ lassen sich mit unserem Hochfrequenzschnüffler anzeigen. Dazu stecken wir die Steckbrücke auf die mit „HF-Schnüffler“ bezeichnete Position 6. Bei nur schwacher HF-Strahlung zeigt das Gerät nichts an, was eigentlich der Normalzustand sein sollte. Kommt man mit der Antenne, die an der spitzen Platinenseite liegt (im Bild rot dargestellt), in die Nähe einer Strahlungsquelle und drückt die Taste, so beginnt die obere LED langsam zu blinken. Wird die Strahlung, die die Antenne empfängt, stärker, wird das Blinken schneller. Bei der nächst höheren Stärke beginnt die zweite LED langsam zu blinken und so weiter. Für die Ausleuchtung der LEDs misst der Mikrocontroller die Spannung an dem mit der Antenne verbundenen Eingang. Die oberste LED beginnt langsam zu blinken, wenn die Spannung 0,5mV (ein halbes tausendstel Volt) erreicht, die unterste LED blinkt schnell, wenn die Spannung über 1,2V liegt, also mehr als das 2000-fache des kleinsten Wertes erreicht.

Ausprobieren kann man den HF-Schnüffler, indem man ihn in die Nähe eines Netzkabels hält. Die meisten störenden Geräte strahlen ihre Störungen nicht direkt ab, sondern koppeln einen großen Anteil in das Stromnetz ein. Deshalb kann man in der Nähe von Netzkabeln (leider) fast immer Störungen, also Hochfrequenzabstrahlung feststellen.

Natürlich gibt es nicht nur Störer, sondern auch gewünschte Ausstrahlungen, wie z.B., die von Handys. Auch die kann man mit unseren HF-Schnüffler erkennen. Allerdings braucht es dafür etwas Geduld, denn man muss einen Zeitpunkt erwischen, zu dem das Handy wirklich strahlt.

Lötet man an dem mit „Antenne“ bezeichneten Punkt ein kurzes Stück Draht, wird die Platine noch einmal deutlich empfindlicher.

!!! Wichtiger Warnhinweis !!!

Mit der Platine oder einer eventuell angelöteten Zusatzantenne keine spannungsführenden Teile berühren. Das kann im harmlosesten Fall die Platine zerstören, wenn es böse ausgeht, aber auch zu einem elektrischen Schlag führen.

2.4 Thermometer mit binärer Anzeige



Eigentlich wurde diese spezielle Anzeige eher für Testzwecke eingebaut. Man kann sie aber für Temperaturen benutzen, die sich mit der „normalen“ Thermometerfunktion nicht anzeigen lassen: Temperaturen unter 18 Grad oder über 32 Grad.

Für die Binäranzeige „parkt“ man die Steckbrücke auf der Position 12. Man kann sie aber auch ganz weglassen. Parkt man sie stattdessen auf der angegebenen Position, ist die Gefahr kleiner, dass die kleine Steckbrücke verlorengeht.

Drückt man jetzt auf die Taste, werden eine oder mehrere LEDs gleichzeitig blinken. Um von der Anzeige zum Temperaturwert zu kommen, muss man zwei Fälle unterscheiden:

- Die unterste LED (Vorzeichen) ist aus:

In dem Fall ist die Temperatur über 0 Grad.

Die „Wertigkeit“ der LEDs ist von oben nach unten 1, 2, 4, 8, 16 und 32.

Man erhält die Temperatur, wenn man die Wertigkeiten aller ausgeleuchteten LEDs zusammenzählt.

Beispiel: Ist die oberste, dann die übernächste und ab da wieder die übernächste LED ausgeleuchtet.

Das sind dann also die LEDs mit den Wertigkeiten 1, 4 und 16. Die gemessene Temperatur ist damit $1 + 4 + 16 = 21$ Grad.

Ist überhaupt keine LED ausgeleuchtet, steht das für 0 Grad,

- Die unterste LED (Vorzeichen) ist an:

Die Temperatur ist in dem Fall unter 0 Grad.

Jetzt ist die Berechnung der gemessenen Temperatur etwas komplizierter. Die „Wertigkeit“ ist auch jetzt, wie oben schon beschrieben, von oben nach unten 1, 2, 4, 8, 16 und 32. Jetzt muss man aber die Wertigkeiten der **nicht** ausgeleuchteten LEDs zusammenzählen und die so erhaltene Zahl noch um „1“ erhöhen, um zum Temperaturwert zu kommen. Beispiel: Die LEDs mit den Wertigkeiten 1, 4, 16 und 32 sowie die Vorzeichen-LED sind ausgeleuchtet. **Nicht** ausgeleuchtet sind damit die LEDs 2 und 8. Für die Temperatur ergibt sich damit $-((2 + 8) + 1) = -11$ Grad.

Bei den Informatikern heißt diese etwas sonderbar anmutende Berechnung „Zweier-Komplement“.

2.5 Die Brückenpositionen 8 und 10


```

*          Temperatur oder einer Spannung in logarithmischem          *
*          Maßstab.                                                  *
* E/A:      PA0...PA6: LED_Ausgänge                                  *
*          PA7:      Analogeingang                                  *
*          PB0:      = 0: Spannungsausgabe                          *
*          PB1:      = 0: Thermometer                               *
*          PB2:      = 0: Würfel                                    *
* Historie: 15.09.17: Ausprogrammierung des HF-Schnüfflers.         *
*          16.09.17: Ausprogrammieren des Thermometers             *
*          17.09.17: Start der Würfelsoftware, Fehlerbehebung VGL_16 *
*          18.09.17: Fertigstellung Würfelsoftware                *
*          04.10.17: Fehlerbehebung mehrfacher Würfelausgaben     *
*          26.08.19: Zahl der Würfelmessungen von 40 auf 100 erhöht *
*          23.01.23: Erweiterung der Würfelfunktion um Vorgängerwerte *
* ----- *
*
* Zelleninhalte                                                    *
* 0070      Zustand der Digitaleingänge PB0...PB2 (Brückenbelegung)*
* 0071      Zähler für die Ermittlung eines Mittelwerts            *
* 0072      MSB des Rohwerts des Meßwerts                          *
* 0073      LSB des Rohwerts des Meßwerts                          *
* 0074      MSB des gemittelten Meßwerts                           *
* 0075      LSB des gemittelten Meßwerts                           *
* 0076      Ausleuchtmuster der LEDs                               *
* 0077      Kennung für den Blinktakt                              *
* 0078      MSB Temperaturkorrektur ($55: noch nicht kalibriert)  *
* 0079      LSB Temperaturkorrektur                                *
* 007A      Kalibrierkennung $55: Kalibrierung durchgeführt      *
* 007B      Würfelzähler (1: Ergebnis ermittelt)                   *
* 007C      Temperaturwert für Würfelzufallszahl LSB              *
* 007D      Temperaturwert für Würfelzufallszahl MSB              *
* ----- *

```

```
#pragma prozessor ATtiny441
```

```

#define      K_TEMP      20          ; Temperatur auf die kalibriert wird
#define      W_MESSUNGEN 100         ; Anzahl Messwerte für den Würfel

#define      LED_FREQ0_H  64         ; Wert für niedrige Blinkfrequenz MSB
#define      LED_FREQ0_L  0          ; Wert für niedrige Blinkfrequenz LSB
#define      LED_FREQ1_H  32         ; Wert für hohe Blinkfrequenz MSB
#define      LED_FREQ1_L  0          ; Wert für hohe Blinkfrequenz LSB
#define      LED_ZEIT_H   8          ; Wert für Leuchtzeit MSB
#define      LED_ZEIT_L   0          ; Wert für Leuchtzeit LSB

#define      M0_M         $01        ; MSB Wert 0 Grad
#define      M0_L         $13        ; LSB Wert 0 Grad      275 / 113
#define      M18_M        $01        ; MSB Wert 18 Grad
#define      M18_L        $25        ; LSB Wert 18 Grad

#define      BRUECKEN     $0070     ; Zelle für Brückenbelegung
#define      MW_ZHL       $0071     ; Mittelwertzähler
#define      ROHW_M       $0072     ; MSB des gemessenen Rohwertes
#define      ROHW_L       $0073     ; LSB des gemessenen Rohwertes

```

```

#define    KERG_M      $0074      ; MSB des gemittelten Rohwertes
#define    KERG_L      $0075      ; LSB des gemittelten Rohwertes
#define    MUSTER      $0076      ; Ausleuchtmuster der LEDs
#define    BLINKTAKT   $0077      ; Kennung Blinktakt schnell/langsam
#define    KORR_M      $0078      ; MSB des Korrekturwerts für die Temperatur
#define    KORR_L      $0079      ; LSB des Korrekturwerts für die Temperatur
#define    KALIB_K     $007A      ; Kalibrierkennung
#define    W_ZHL       $007B      ; Würfelzähler
#define    W_WERT_M    $007C      ; Zufallswert für Würfel MSB
#define    W_WERT_L    $007D      ; Zufallswert für Würfel LSB

```

```

#define    Z_MSB       $00        ; MSB Start Merzzellen
#define    Z_LSB       $70        ; LSB Start Merzzellen

```

```

* ===== Interruptliste =====
*

```

```

0000    rjmp RESET_INT          ; RESET Interrupt
000C    rjmp TIMER_INT1A       ; Timer 1 Interrupt COMP1A
000E    rjmp TIMER_INT1B       ; Timer 1 Interrupt COMP1B
001A    rjmp ADC_INT           ; ADC Interrupt

```

```

* ===== Versionsangabe ===== *

```

```

:VERSION

```

```

0028    data 'LW'              ; Programmversion
        data '23'
        data '01'
        data '23'
        data 0

```

```

* ===== RESET Interrupt ===== *

```

```

:RESET_INT          ; Reset Interrupt

```

```

    ldi r16, $01          ; Obergrenze STACK MSB
    out SPH, r16         ; laden
    ldi r16, $5F         ; Obergrenze Stack LSB laden
    out SPL, r16         ; laden
    ldi r16, $80         ; Änderung für zentralen Takt
    out CLKPR, r16       ; aktivieren
    ldi r16, $04         ; Zentralen Takt auf 8MHZ/16 = 500kHz
    out CLKPR, r16       ; einstellen

```

```

* ----- Einstellung der Ports ----- *

```

```

    ldi r16, $7F         ; Vorbesetzung für E/A-Ports A
    out PORTA, r16       ; Alle LEDs (PA0...PA6) aus
    out DDRA, r16        ; PA0...PA6 Ausgänge, PA7 Eingang
    ldi r16, $FF         ; Alle digitalen Eingangspuffer von
    out DIDR0, r16       ; PORTA ausschalten
    clr r16              ; Alle PB-Ports: Eingänge

```

```

    out  DDRB, r16          ; B-Ports auf "Eingang" setzen
    ldi  r16, $0F          ; Vorbesetzung für Pullup-Widerstände
    out  PORTB, r16        ; Pullup-Widerstände PBx setzen

* ----- Rücksetzen der Merzkellen ----- *

    clr  r16                ; Vorbesetzung der Merzkellen
    clr  r17                ; Zähler für Löschschleife
    ldi  r27, Z_MSB         ; MSB Bereich für Merzkellen
    ldi  r26, Z_LSB        ; LSB Bereich für Merzkellen

:RESET_INT:1
    st   X+, r16           ; Zelle löschen
    inc  r17               ; Durchlaufzähler +1
    cpi  r17, $20          ; Grenze Erreicht?
    brne RESET_INT:1      ; --> Nein

* ----- Abschalten nicht benötigter Funktionen ----- *

    ldi  r16, $06          ; Takte für Timer 0 und USI
    out  PRR, r16          ; abschalten
    in   r16, MCUCR        ; MCUCR laden
    ori  r16, $20          ; Sleep mode idle
    out  MCUCR, r16        ; MCUCR zurückschreiben

* ----- Kalibrierwert für das Thermometer einlesen ----- *

    ldi  r16, $55          ; Blockiert MSB der Korrektur solange
    sts  KORR_M, r16       ; noch keine korrekten Daten vorhanden
    ldi  r16, 2            ; EEPROM-Adresse für Kalibriererkennung
    call EEPROM_LESEN     ; Kalibriererkennung auslesen
    sts  KALIB_K, r17      ; und merken
    cpi  r17, $55          ; Kalibrierung durchgeführt?
    brne RESET_INT:2      ; ==> Nein
    ldi  r16, 0            ; Adresse LSB Kalibrierwert
    call EEPROM_LESEN     ; LSB des Kalibrierwertes auslesen
    sts  KORR_L, r17       ; und merken
    ldi  r16, 1            ; Adresse MSB Kalibrierwert
    call EEPROM_LESEN     ; MSB des Kalibrierwertes auslesen
    sts  KORR_M, r17       ; und merken

* ----- Brückenbelegung einlesen ----- *

:RESET_INT:2
    in   r16, PINB         ; Signalzustand Port B einlesen
    ori  r16, $08          ; Pullup für RESET setzen
    lds  r17, KORR_M       ; Korrekturwert für Kalibrierung lesen
    cpi  r17, $55          ; Hat eine Kalibrierung stattgefunden?
    breq RESET_INT:4      ; ==> Nein, Pullupwiderstände aktiv lassen
    out  PORTB, r16        ; Spart den Strom bei gesteckten Brücken

:RESET_INT:4
    com  r16                ; gesteckte Brücke: "1"
    andi r16, $07          ; Info über Brücken isolieren
    sts  BRUECKEN, r16     ; und sichern

* ----- Einstellen des Timer 1 ----- *

    ldi  r16, LED_FREQ0_H ; MSB des höchsten Zählerwerts
    out  OCR1AH, r16       ; laden

```

```

    ldi r16, LED_FREQ0_L      ; LSB des höchsten Zählerwerts
    out OCR1AL, r16          ; laden
    ldi r16, LED_ZEIT_H      ; MSB des AUS-Zeitpunktes der LEDs
    out OCR1BH, r16          ; laden
    ldi r16, LED_ZEIT_L      ; LSB des AUS-Zeitpunktes der LEDs
    out OCR1BL, r16          ; laden
    clr r16                   ; Startwert Zähler
    out TCNT1H, r16           ; Startwert Zähler MSB
    out TCNT1L, r16           ; Startwert Zähler MSB
    clr r16                   ; Keine direkten Verbindungen nach außen
    out TCCR1A, r16           ; Timer Controlregister laden
    ldi r16, $0A              ; CTC, Vorteiler 1:8 Gesamtzeit 1,048s
    out TCCR1B, r16           ; Timer Controlregister laden
    ldi r16, $06              ; Compare Interrupts COMPA und COMB
    out TIMSK1, r16           ; Interrupt für Zähler 1 aktivieren

* ----- Holen des Zufallswertes aus der letzten Würfelaktion ----- *

    ldi r16, $10              ; LSB-Adresse des alten Würfelwerts
    call EEPROM_LESEN         ; Alten Wert holen
    sts W_WERT_L, r17          ; und sichern
    ldi r16, $11              ; MSB-Adresse des alten Würfelwerts
    call EEPROM_LESEN         ; Alten Wert holen
    sts W_WERT_M, r17          ; und sichern

* ----- Modusabhängige Einstellungen ----- *

    lds r16, BRUECKEN         ; Status der Steckbrücken laden
    cpi r16, $01              ; HF-Schnüffler?
    brne RESET_INT:3         ; ==> Nein!

* ----- Einstellungen für den HF-Schnüffler ----- *

    ldi r16, $87              ; ADC7 (Temp) Vcc= 1,1V
    out ADMUX, r16            ;
    ldi r16, $8A              ; ADC EIN, Interrupt ein, Vorteiler = 4
    out ADCSRA, r16           ;
    rjmp RESET_INT:ENDE

* ----- Einstellung ADC für Würfel und Thermometer ----- *

:RESET_INT:3
    ldi r16, $A2              ; ADC8 (Temp) Vcc= 1,1V
    out ADMUX, r16            ;
    ldi r16, $8A              ; ADC EIN, Interrupt ein, Vorteiler = 4
    out ADCSRA, r16           ;
:RESET_INT:ENDE

* ----- Endlosschleife ----- *

    sei                       ; Interrupts aktivieren

:RESET_INT:SCHLEIFE
    sleep
    rjmp RESET_INT:SCHLEIFE

* ===== Interrupt COMPA des Timers 1 ===== *

```

```

:TIMER_INT1A                                ; Timer 1 Interrupt COMP1A
    push r16
    push r24
    in    r24, SREG                          ; SREG retten

* ----- Rücksetzen der Merzzellen ----- *

    clr  r16                                ; Löschen der Merzzellen
    sts  MW_ZHL, r16                         ; Löschen des Mittelwertzählers
    sts  ROHW_M, r16                         ; Löschen des Rohwertes MSB
    sts  ROHW_L, r16                         ; Löschen des Rohwertes LSB
    lds  r16, BRUECKEN                       ; Zustand der Steckbrücken holen
    cpi  r16, $04                            ; Würfel aktiv?

    brne TIMER_INT1A:2                       ; ==> Nein
    lds  r16, W_ZHL                          ; Würfelzähler holen
    cpi  r16, 1                              ; Würfelergebnis schon ermittelt?
    breq TIMER_INT1A:3                       ; ==> Ja! ADC Aktivierung überspringen
:TIMER_INT1A:2
    ldi  r16, $06                            ; Takte für Timer 0 und USI AUS, ADC EIN
    out  PRR, r16                            ; ADC-Takt einschalten
    in   r16, MCUCR                          ; MCUCR laden
    ori  r16, $01                            ; Sleep mode Noise Reduction
    out  MCUCR, r16                          ; MCUCR zurückschreiben + ADC Start
    rjmp TIMER_INT1A:4
:TIMER_INT1A:3
    ldi  r16, $07                            ; Takte für Timer 0 und USI AUS, ADC AUS
    out  PRR, r16                            ; ADC-Takt ausschalten
    in   r16, MCUCR                          ; MCUCR laden
    andi r16, $FE                            ; Sleep mode Idle
    out  MCUCR, r16                          ; MCUCR zurückschreiben
:TIMER_INT1A:4

* ----- Ausgabe des Messergebnisses ----- *

    lds  r16, MUSTER                         ; Bitmuster für HF-Ausleuchtung holen
    out  PORTA, r16                          ; Ergebnis anzeigen
    lds  r16, BLINKTAKT                      ; Kennung für den Blinktakt einlesen
    tst  r16                                  ; Schnell blinken?
    brne TIMER_INT1A:1                       ; ==> Ja!
    ldi  r16, LED_FREQ0_H                    ; MSB des höchsten Zählerwerts für
    out  OCR1AH, r16                         ; langsames Blinken laden
    ldi  r16, LED_FREQ0_L                    ; LSB des höchsten Zählerwerts für
    out  OCR1AL, r16                         ; langsames Blinken laden
    rjmp TIMER_INT1A:ENDE                    ; Bearbeitung abgeschlossen
:TIMER_INT1A:1
    ldi  r16, LED_FREQ1_H                    ; MSB des höchsten Zählerwerts für
    out  OCR1AH, r16                         ; schnelles Blinken laden
    ldi  r16, LED_FREQ1_L                    ; LSB des höchsten Zählerwerts für
    out  OCR1AL, r16                         ; schnelles Blinken laden
:TIMER_INT1A:ENDE
    out  SREG, r24                           ; SREG restaurieren
    pop  r24
    pop  r16
    reti                                     ; Rücksprung aus TIMER_INT1A

```

* ===== Interrupt COMPB des Timers 1 ===== *

```
:TIMER_INT1B                ; Timer 1 Interrupt COMP1B
    push r16
    push r24
    in   r24, SREG            ; SREG retten
    ldi  r16, $7F            ; LEDs abschalten um Strom
    out  PORTA, r16          ; zu sparen.
    out  SREG, r24           ; SREG restaurieren
    pop  r24
    pop  r16
    reti                     ; Rücksprung aus TIMER_INT1B
```

* ===== Interrupt des ADC ===== *

```
:ADC_INT                    ; ADC Interrupt: Konvertierung fertig
    push r16
    push r17
    push r18
    push r19
    push r24
    in   r24, SREG            ; SREG retten
    lds  r16, BRUECKEN        ; Status gesteckter Brücken
    cpi  r16, 0               ; Temperatur Binärausgabe?
    breq ADC_INT:2           ; ==> Ja
    cpi  r16, 1               ; HF-Schnüffler?
    breq ADC_INT:3           ; ==> Ja
    cpi  r16, 2               ; Temperatur Balkenausgabe?
    breq ADC_INT:2           ; ==> Ja
    cpi  r16, 4               ; Würfel
    breq ADC_INT:4a          ; ==> Ja
    out  SREG, r24           ; SREG restaurieren
    pop  r24
    pop  r19
    pop  r18
    pop  r17
    pop  r16
    reti                     ; Rücksprung aus ADC_INT
```

* ***** Code für den HF-Schnüffler ***** *

```
:ADC_INT:3
    lds  r16, ROHW_L          ; LSB der Merzkelle für den Rohwert
    lds  r17, ROHW_M          ; MSB der Merzkelle für den Rohwert
    in   r18, ADCL            ; LSB des Meßwerts
    in   r19, ADCH            ; MSB des Meßwerts
    add  r18, r16              ; LSBs der 16-Bit-Werte
    adc  r19, r17              ; LSBs der 16-Bit-Werte
    sts  ROHW_L, r18          ; Aktualisierten Meßwert LSB sichern
    sts  ROHW_M, r19          ; Aktualisierten Meßwert MSB sichern
    lds  r16, MW_ZHL          ; Zähler für Mittelwertbildung lesen
    inc  r16                  ; und weiterstellen
    cpi  r16, 4               ; Ende erreicht?
    breq ADC_INT:1           ; ==> Ja!
```

```

        sts MW_ZHL, r16          ; Neuen Zähler sichern
        in  r16, ADCSRA         ; ADC Steuerregister holen
        ori r16, $40           ; Startbit auffordern
        out ADCSRA, r16        ; Nächste Konvertierung starten
        rjmp ADC_INT:ENDE

:ADC_INT:1
        clr  r16                ; Zähler auf Anfang zurück
        sts MW_ZHL, r16        ; und sichern
        lsr  r19                ; Summe der Meßwerte durch
        ror  r18                ; 4 dividieren
        lsr  r19
        ror  r18
        sts  KERG_L, r18        ; Gemittelte Meßwerte für die
        sts  KERG_M, r19        ; Logarithmierung sichern
        call LOG                ; Meßwerte logarithmieren Ausgabe berechnen
        in  r16, MCUCR         ; MCUCR laden
        andi r16, $FE          ; Sleep mode Idle
        out  MCUCR, r16        ; MCUCR zurückschreiben
        rjmp ADC_INT:ENDE

:ADC_INT:4a
        rjmp ADC_INT:4

* ***** Code für das Thermometer ***** *

:ADC_INT:2
        lds  r16, MW_ZHL        ; Stand des Durchlaufzählers holen
        tst  r16                ; Erster Durchlauf?
        brne ADC_INT:7         ; ==> Nein: Löschvorgang überspringen
        clr  r16                ; Vorbesetzung für Rohwertpuffer
        sts  ROHW_L, r16        ; LSB Rohwert löschen
        sts  ROHW_M, r16        ; MSB Rohwert löschen

:ADC_INT:7
        lds  r16, ROHW_L        ; LSB der Merzkelle für den Rohwert
        lds  r17, ROHW_M        ; MSB der Merzkelle für den Rohwert
        in  r18, ADCL           ; LSB des Meßwerts
        in  r19, ADCH           ; MSB des Meßwerts
        add  r18, r16           ; LSBs der 16-Bit-Werte
        adc  r19, r17           ; MSBs der 16-Bit-Werte
        sts  ROHW_L, r18        ; Aktualisierten Meßwert LSB sichern
        sts  ROHW_M, r19        ; Aktualisierten Meßwert MSB sichern
        lds  r16, MW_ZHL        ; Zähler für Mittelwertbildung lesen
        inc  r16                ; und weiterstellen
        cpi  r16, 8             ; Ende erreicht?
        breq ADC_INT:5         ; ==> Ja!
        sts  MW_ZHL, r16        ; Neuen Zähler sichern
        in  r16, ADCSRA         ; ADC Steuerregister holen
        ori  r16, $40           ; Startbit auffordern
        out  ADCSRA, r16        ; Nächste Konvertierung starten
        rjmp ADC_INT:ENDE

:ADC_INT:5
        clr  r16                ; Zähler auf Anfang zurück
        sts  MW_ZHL, r16        ; und sichern
        lsr  r19                ; Summe der Meßwerte durch

```

```

ror r18 : 8 dividieren
lsr r19
ror r18
lsr r19
ror r18
sts KERG_L, r18 ; Gemittelten Meßwert für die
sts KERG_M, r19 ; Korrektur sichern
call KORR ; Meßwert korrigieren
in r16, PINB ; Aktuelle Brücken einlesen
com r16 ; logisch invertieren
andi r16, $07 ; und Brückeninformation lesen
lds r17, BRUECKEN ; Alte Brückeninfo lesen
cp r16, r17 ; Haben sich die Brücken geändert?
breq ADC_INT:6 ; ==> Nein, Kalibrierung überspringen
lds r17, KALIB_K ; Kalibrierkennung holen
cpi r17, $55 ; Wurde bereits kalibriert?
breq ADC_INT:6 ; ==> Ja, Kalibrierung überspringen
call KALIBRIERUNG ; Kalibrierwerte in EEPROM schreiben
:ADC_INT:6
lds r16, BRUECKEN ; Brückenbelegung lesen
cpi r16, 2 ; Balkenausgabe?
breq ADC_INT:8 ; ==> Ja

* ----- Binärausgabe relativ zu 0 Grad ----- *

ldi r16, M0_L ; LSB Basis 0 Grad für Binärausgabe
ldi r17, M0_M ; MSB Basis 0 Grad für Binärausgabe
call SUB ; Differenz zu 0 Grad ermitteln
lds r18, KERG_L ; Relevantes LSB laden
com r18 ; wegen invertierter LED Ausleuchtung
sts MUSTER, r18 ; In Ausgabezelle laden
clr r18 ; Langsamen Blinktakt einstellen und für
sts BLINKTAKT, r18 ; die Ausgabe merken
rjmp ADC_INT:9

* ----- Balkenausgabe relativ zu 18 Grad ----- *

:ADC_INT:8
ldi r16, M18_L ; LSB Basis 18 Grad für Binärausgabe
ldi r17, M18_M ; MSB Basis 18 Grad für Binärausgabe
call SUB ; Differenz zu 18 Grad ermitteln
lds r18, KERG_L ; Relativen Temperaturwert holen
tst r18 ; Temperaturwert auf Grenzen prüfen
brmi ADC_INT:10 ; ==> Temperatur zu niedrig
cpi r18, 15 ; Temperatur zu hoch?
brsh ADC_INT:11 ; ==> Ja
call LED_MU ; Muster für LED-Ausleuchtung ermitteln
rjmp ADC_INT:9

:ADC_INT:10 ; LED-Muster für Temp << setzen
ldi r18, $BE ; 1. und letzte LED an
sts MUSTER, r18
clr r18 ; Kennung für langsames Blinken
sts BLINKTAKT, r18
rjmp ADC_INT:9

:ADC_INT:11 ; LED-Muster für Temp << setzen

```

```

        ldi r18, $B6                ; 1., mittlere und letzte LED an
        sts MUSTER, r18
        clr r18                    ; Kennung für langsames Blinken
        sts BLINKTAKT, r18
:ADC_INT:9
        in r16, MCUCR              ; MCUCR laden
        andi r16, $FE              ; Sleep mode Idle
        out MCUCR, r16            ; MCUCR zurückschreiben
        rjmp ADC_INT:ENDE

* ***** Code für den Würfel ***** *

:ADC_INT:4
        lds r16, W_WERT_L          ; LSB der Merzkelle für den Rohwert
        lds r17, W_WERT_M          ; MSB der Merzkelle für den Rohwert
        in r18, ADCL               ; LSB des Meßwerts
        in r19, ADCH               ; MSB des Meßwerts
        add r18, r16               ; LSBs der 16-Bit-Werte
        adc r19, r17               ; MSBs der 16-Bit-Werte
        sts W_WERT_L, r18          ; Aktualisierten Meßwert LSB sichern
        sts W_WERT_M, r19          ; Aktualisierten Meßwert MSB sichern
        lds r16, MW_ZHL            ; Zähler für Würfelmessungen lesen
        inc r16                    ; und weiterstellen
        cpi r16, W_MESSUNGEN        ; Ende erreicht?
        breq ADC_INT:12            ; ==> Ja!
        sts MW_ZHL, r16            ; Neuen Zähler sichern
        in r16, ADCSRA             ; ADC Steuerregister holen
        ori r16, $40               ; Startbit auffordern
        out ADCSRA, r16            ; Nächste Konvertierung starten
        rjmp ADC_INT:ENDE

:ADC_INT:12
        clr r16                    ; Zähler auf Anfang zurück
        sts MW_ZHL, r16            ; und sichern
        lds r16, W_WERT_L          ; MOD_6 mit aufaddierten Temperaturwerten
        lds r17, W_WERT_M          ; versorgen
        call MOD_6                 ; Auf Würfelzahlen reduzieren
        ldi r17, $01              ; Startwert Ausleuchtmuster

:ADC_INT:13
        tst r16                    ; Schiebeoperation abgeschlossen?
        breq ADC_INT:14            ;
        lsl r17                    ; Bitmuster weiterschieben
        dec r16                    ; Schleifenzähler zurück
        rjmp ADC_INT:13

:ADC_INT:14
        lds r19, W_ZHL             ; Kennung ob Wert ermittelt holen
        tst r19                    ; Wert bereits ermittelt?
        brne ADC_INT:ENDE         ; ==> Ja! Ausgabemuster nicht mehr laden
        com r17                    ; Wegen der invertierten LED-Anzeige
        andi r17, $7F              ; auf die 7 LEDs beschränken
        sts MUSTER, r17           ; Für die Ausgabe abspeichern
        ldi r17, $01              ; Kennung: "Wert ermittelt" setzen
        sts W_ZHL, r17            ; und sichern
        call W_SICHERUNG           ; Letzte Würfelwerte --> EEPROM

:ADC_INT:ENDE

```

```

        out  SREG, r24                ; SREG restaurieren
        pop  r24
        pop  r19
        pop  r18
        pop  r17
        pop  r16
        reti                          ; Rücksprung aus ADC_INT

* ----- *
* Funktion: Sicherung der Werte W_WERT_L und W_WERT_M / 2 im EEPROM      *
* Versorgung: W_WERT_L, W_WERT_M                                         *
* Ergebnis: Aktualisierter EEPROM                                         *
* Veränderte Register: Keine                                             *
* ----- *

:W_SICHERUNG
        push r16
        push r17
        push r18
        push r19
        lds  r18, W_WERT_L          ; Würfelwert laden
        lds  r19, W_WERT_M
        lsr  r19                    ; MSB des Würfelwerts :2
        ror  r18                    ; LSB des Würfelwerts :2
        ldi  r16, $10               ; EEPROM Adresse für LSB des Wertes
        mov  r17, r18               ; LSB holen
        call EEPROM_SCHREIBEN      ; und sichern
        ldi  r16, $11               ; EEPROM Adresse für MSB des Wertes
        mov  r17, r19               ; MSB holen
        call EEPROM_SCHREIBEN      ; und sichern
        pop  r19
        pop  r18
        pop  r17
        pop  r16
        ret                          ; Rücksprung aus UP W_SICHERUNG

* ----- *
* Funktion: Logarithmus des Bereiches 0 ... 2261 auf den Bereich 0 ... 14 *
* Versorgung: KERG_L, KERG_M: Umzurechnender Wert                       *
* Ergebnis: Ergebnis der Umrechnung in MUSTER und BLINKTAKT            *
* Veränderte Register: Keine                                             *
* ----- *

:LOG
        push r16
        push r17
        push r18
        push r19
        push r20
        lds  r16, KERG_L            ; LSB des umzurechnenden Wertes
        lds  r17, KERG_M            ; MSB des umzurechnenden Wertes
        ldi  r18, 213
        ldi  r19, 8
        call VGL_16                 ; Wert r16/r17 mit r18/r19 vergleichen

```

```

        cpi r20, 0                ; Ergebnis auswerten
        brmi LOG:1                ; Wert ist <= 2261
        ldi r18, 14                ; Ergebnis für Wert > 2261
        rjmp LOG:ENDE
:LOG:1
        ldi r18, 224
        ldi r19, 4
        call VGL_16                ; Wert r16/r17 mit r18/r19 vergleichen
        cpi r20, 0
        brmi LOG:2
        ldi r18, 13                ; Ergebnis für Wert > 1248, <= 2261
        rjmp LOG:ENDE
:LOG:2
        ldi r18, 179
        ldi r19, 2
        call VGL_16                ; Wert r16/r17 mit r18/r19 vergleichen
        cpi r20, 0
        brmi LOG:3
        ldi r18, 12                ; Ergebnis für Wert > 691, <= 1248
        rjmp LOG:ENDE
:LOG:3
        ldi r18, 124
        ldi r19, 1
        call VGL_16                ; Wert r16/r17 mit r18/r19 vergleichen
        cpi r20, 0
        brmi LOG:4
        ldi r18, 11                ; Ergebnis für Wert > 380, <= 691
        rjmp LOG:ENDE
:LOG:4
        ldi r18, 210
        ldi r19, 0
        call VGL_16                ; Wert r16/r17 mit r18/r19 vergleichen
        cpi r20, 0
        brmi LOG:5
        ldi r18, 10                ; Ergebnis für Wert > 210, <= 380
        rjmp LOG:ENDE
:LOG:5
        cpi r16, 116
        brlo LOG:6
        ldi r18, 9                ; Ergebnis für Wert > 116, <= 210
        rjmp LOG:ENDE
:LOG:6
        cpi r16, 64
        brlo LOG:7
        ldi r18, 8                ; Ergebnis für Wert > 64, <= 116
        rjmp LOG:ENDE
:LOG:7
        cpi r16, 35
        brlo LOG:8
        ldi r18, 7                ; Ergebnis für Wert > 35, <= 64
        rjmp LOG:ENDE
:LOG:8
        cpi r16, 20

```

```

        brlo LOG:9
        ldi r18, 6 ; Ergebnis für Wert > 20, <= 35
        rjmp LOG:ENDE
:LOG:9
        cpi r16, 10
        brlo LOG:10
        ldi r18, 5 ; Ergebnis für Wert > 10, <= 20
        rjmp LOG:ENDE
:LOG:10
        cpi r16, 6
        brlo LOG:11
        ldi r18, 4 ; Ergebnis für Wert > 6, <= 10
        rjmp LOG:ENDE
:LOG:11
        cpi r16, 3
        brlo LOG:12
        ldi r18, 3 ; Ergebnis für Wert > 3, <= 6
        rjmp LOG:ENDE
:LOG:12
        mov r18, r16 ; Werte 0, 1, 2 direkt übernehmen
:LOG:ENDE
        call LED_MU ; Ausleuchtungsmuster erzeugen
        pop r20
        pop r19
        pop r18
        pop r17
        pop r16
        ret ; Rücksprung aus UP LOG

```

```

* ----- *
* Funktion: Vergleicht die Werte r16/r17 mit R18/r19 und legt das Ergebnis *
*           in r20 ab *
* Versorgung: r16...r19 Zu vergleichende Werte *
* Ergebnis: Vergleichsergebnis in r20 *
* Veränderte Register: r20 *
* ----- *

```

```

:VGL_16
        cp r17, r19
        brlo VGL_16:1 ; MSB kleiner --> gesamte Zahl kleiner
        brne VGL_16:2 ; MSB Größer --> gesamte Zahl größer
        cp r16, r18 ; LSBs vergleichen
        brlo VGL_16:1 ; MSB gleich, LSB kleiner
        brne VGL_16:2 ; MSB gleich, LSB größer 170917
        ldi r20, 0 ; Gleich-Kennung setzen
        rjmp VGL_16:ENDE
:VGL_16:1
        ldi r20, $FF ; Kennung für "kleiner" setzen
        rjmp VGL_16:ENDE
:VGL_16:2
        ldi r20, 1 ; Kennung für "größer" setzen
        rjmp VGL_16:ENDE

```

```

:VGL_16:ENDE
    ret                ; Rücksprung aus UP VGL_16

```

```

* ----- *
* Funktion:          Lesen eines Bytes aus dem unteren Teil des EEPROM      *
* Versorgung:        r16: LSB der EEPROM-Adresse (MSB ist auf 0 gesetzt)    *
* Ergebnis           r17: Aus dem EEPROM ausgelesener Wert                 *
* Veränderte Register: r17                                                *
* ----- *

```

```

:EEPROM_LESEN
    sbic EECR, 1        ; Läuft noch eine Schreiboperation?
    rjmp EEPROM_LESEN  ; --> Ja
    clr r17             ; MSB der EEPROM-Adresse
    out EEARH, r17     ; schreiben
    out EEARL, r16     ; LSB der EEPROM-Adresse schreiben
    sbi EECR, 0        ; Leseoperation starten
    in r17, EEDR       ; Wert auslesen
    ret                ; Rücksprung aus UP EEPROM_LESEN

```

```

* ----- *
* Funktion:          Schreiben eines Bytes in den unteren Teil des EEPROM   *
* Versorgung:        r16: LSB der EEPROM-Adresse (MSB ist auf 0 gesetzt)    *
* Ergebnis           r17: In den EEPROM zu schreibender Wert              *
* Veränderte Register: keine                                              *
* ----- *

```

```

:EEPROM_SCHREIBEN
    sbic EECR, 1        ; Läuft noch eine EEPROM Schreiboperation?
    rjmp EEPROM_SCHREIBEN ; --> Ja
    push r18

```

```

:EEPROM_SCHREIBEN:1
    in r18, SPMCSR      ; Nach Beschreibung ist der Name SPMCR
    sbrc r18, 0         ; Läuft noch eine Flash Schreiboperation?
    rjmp EEPROM_SCHREIBEN:1 ; --> Ja
    clr r18             ; MSB der EEPROM Adresse
    out EEARH, r18     ; schreiben
    out EEARL, r16     ; LSB der EEPROM-Adresse schreiben
    in r18, SREG       ; Wegen Interruptsperre retten
    cli                ; Interruptsperre während EEPROM Schreiben
    out EEDR, r17      ; EEPROM-Daten ablegen
    sbi EECR, 2        ; Schreiben einleiten
    sbi EECR, 1        ; Schreiben ausführen
    out SREG, r18      ; Interrupts wieder freigeben
    pop r18
    ret                ; Rücksprung aus UP EEPROM_SCHREIBEN

```

```

* ----- *
* Funktion:          Korrektur des Temperaturwertes um den Kalibrierwert    *
* Versorgung:        KORR_M / KORR_L die bei der Kalibrierung ermittelten  *
*                   Korrekturwerte. KERGM / KERGL der zu korrigierende    *
* ----- *

```

```

*                               gemittelte Meßroh wert.                               *
* Ergebnis                       KER G_M / KER G_L: Der korrigierte Meßwert          *
* Veränderte Register: keine                                             *
* -----*

```

```

:KORR

```

```

    push r16
    push r17
    push r18
    push r19
    lds r17, KORR_M                ; MSB Korrekturwert holen
    cpi r17, $55                  ; Kennung: keine Korrektur?
    breq KORR:1                   ; ==> Ja, Korrektur überspringen
    lds r16, KORR_L                ; LSB Korrekturwert holen
    lds r18, KER G_L              ; LSB Temperaturroh wert holen
    lds r19, KER G_M              ; MSB Temperaturroh wert holen
    add r18, r16                  ; Rohwert
    adc r19, r17                  ; Um Kalibrierung korrigieren
    sts KER G_L, r18              ; und
    sts KER G_M, r19              ; sichern

```

```

:KORR:1

```

```

    pop r19
    pop r18
    pop r17
    pop r16
    ret                            ; Rücksprung aus UP KORR

```

```

* -----*
* Funktion: Berechnet zu der mit K_TEMP angegebenen Temperatur den              *
*           Korrekturwert und schreibt ihn zusammen mit der Kalibrierkennung    *
*           in die Zellen 0 bis 2 in den EEPROM                                *
* Versorgung: K_TEMP: Temperatur auf die kalibriert wird                       *
* Ergebnis: Aktualisierter EEPROM                                              *
* Veränderte Register: keine                                                  *
* -----*

```

```

:KALIBRIERUNG

```

```

    push r16
    push r17
    push r24
    push r25
    clr r24                        ; Wegen Wortoperation
    lds r17, KER G_M                ; MSB des Rohwertes der Temperatur
    lds r16, KER G_L                ; LSB des Rohwertes der Temperatur
    com r16                          ; 1er Komplement LSB
    com r17                          ; 1er Komplement MSB
    ldi r25, 1                      ; für 2er Komplement
    add r16, r25                    ; r16, r17: 2er Komplement des
    adc r17, r24                    ; Temperaturrohwertes
    ldi r25, M0_M                   ; 0 Grad Sollwert MSB
    ldi r24, M0_L                   ; 0 Grad Sollwert LSB
    adiw r24, K_TEMP                 ; Sollwert für die Temperatur K_TEMP
    add r24, r16                    ; LSB des Korrekturwertes

```

```

adc r25, r17 ; MSB des Korrekturwertes
ldi r16, 0 ; EEPROM Adresse des Korrekturwertes LSB
mov r17, r24 ; Korrekturwert LSB
call EEPROM_SCHREIBEN ; sichern
ldi r16, 1 ; EEPROM Adresse des Korrekturwertes MSB
mov r17, r25 ; Korrekturwert MSB
call EEPROM_SCHREIBEN ; sichern
ldi r16, 2 ; EEPROM Adresse der Kalibrierkennung
ldi r17, $55 ; Kennung: Kalibrierung erfolgt
sts KALIB_K, r17 ; blockiert EEPROM Schreibschleife
call EEPROM_SCHREIBEN ; Kalibrierkennung abspeichern
pop r25
pop r24
pop r17
pop r16
ret ; Rücksprung aus UP KALIBRIERUNG

```

```

* ----- *
* Funktion: Berechnet aus dem Meßwert und der übergebenen Basiswert die *
*           Differenz *
* Versorgung: Temperaturwert in KERGM / KERGL *
*           r16: LSB des Basiswerts *
*           r17: MSB des Basiswerts *
* Ergebnis: Differenz in KERGM / KERGL *
* Veränderte Register: keine *
* ----- *

```

:SUB

```

push r16
push r17
push r18
push r19
ldi r18, 0 ; Wegen Wortoperation
ldi r19, 1 ; Für Komplementbildung
com r16 ; LSB 1er Komplement
com r17 ; MSB 1er Komplement
add r16, r19 ; LSB 2er Komplement
adc r17, r18 ; MSB 2er Komplement
lds r18, KERGL ; Rohwert LSB holen
lds r19, KERGM ; Rohwert MSB holen
add r18, r16 ; LSBs addieren
adc r19, r17 ; MSBs addieren
sts KERGM, r19 ; Temperaturwert MSB sichern
sts KERGL, r18 ; Temperaturwert LSB sichern
pop r19
pop r18
pop r17
pop r16
ret ; Rücksprung aus SUB

```

```

* ----- *
* Funktion: Setzt den in r18 übergebenen Wert (Bereich 0...14) in das *

```

```

*           Ausgabebitmuster 0...6 um und eine Blinkkennung (ungerade      *
*           schnell, gerade: langsam)                                     *
* Versorgung: r18, umzusetzender Wert                                   *
* Ergebnis   MUSTER / BLINKTAKT                                         *
* Veränderte Register: keine                                           *
* ----- *

```

```

:LED_MU
    push r16
    push r18
    push r20
    mov  r20, r18                ; Ergebnis retten
    inc  r18                    ; Wegen der Behandlung der "1"
    lsr  r18                    ; Nummer der auszuleuchtenden LED
    breq LED_MU:1              ; ==> Keine LED auszuleuchten
    ldi  r16, $01              ; Startmuster
:LED_MU:2
    dec  r18                    ; LED-Nummer als Schleifenzähler
    breq LED_MU:3              ; ==> Bit auf richtiger Stelle
    lsl  r16                    ; LED-Position weiterschieben
    rjmp LED_MU:2              ; ==> nächste Runde
:LED_MU:3
    com  r16                    ; Ausleuchtung auf 7 Stellen beschränken
    andi r16, $7F              ; Wegen invertierter LED Ausgabe
    rjmp LED_MU:4              ; Bit 8 (Analogeingang) ausblenden
:LED_MU:1
    ldi  r16, $7F              ; ==> Ausleuchtmuster fertig
:LED_MU:4
    ldi  r16, $7F              ; Keine LED an: Fixes Ausleuchtmuster
:LED_MU:4
    sts  MUSTER, r16           ; Alle LEDS aus
                                ; sichern

```

```

* ----- Ermittlung des Blinktakts ----- *
    com  r20                    ; Wegen Langsam/Schellwechsel invertieren
    andi r20, $01              ; LSB des 1...14 Ergebnisses isolieren
    sts  BLINKTAKT, r20        ; und sichern
    pop  r20
    pop  r18
    pop  r16
    ret                        ; Rücksprung aus LED_MU

```

```

* ----- *
* Funktion: 16 Bit Subtraktion                                         *
* Versorgung: r16, r17, r18, r18                                       *
* Ergebnis:  r17/r16 - r19/r18                                         *
* Veränderte Register: r16, r17                                         *
* ----- *

```

```

:SUB_16
    push r18
    push r19
    push r20
    push r21
    ldi  r20, 0                ; Wegen Wortoperation
    ldi  r21, 1                ; Für 2er Komplementbildung

```

```

com r18 ; LSB 1er Komplement
com r19 ; MSB 1er Komplement
add r18, r21 ; LSB 2er Komplement
adc r19, r20 ; MSB 2er Komplement
add r16, r18 ; LSBs addieren
adc r17, r19 ; MSBs addieren
pop r21
pop r20
pop r19
pop r18
ret ; Rücksprung aus SUB_16

```

```

* ----- *
* Funktion: 16 Bit Modulo 6 *
* Versorgung:r16, r17 *
* Ergebnis: r17/r16 modulo 6 --> r16 *
* Veränderte Register: r16 *
* ----- *

```

```

:MOD_6
    push r17
    push r18
    push r19
    push r20
    ldi r19, $C0 ; MSB Vergleichszahl für modulo 6
    ldi r18, $00 ; LSB Vergleichszahl für modulo 6
:MOD_6:1
    call VGL_16
    tst r20 ; Ist Zahl kleiner als die Vergleichszahl?
    brmi MOD_6:2 ; ==> Ja, Subtraktion überspringen
    call SUB_16 ; 16-Bit Subtraktion
:MOD_6:2
    lsr r19 ; 16-Bit Vergleichszahl
    ror r18 ; halbieren
    cpi r18, 3 ; Ende erreicht?
    breq MOD_6:3 ; ==> Ja, Berechnung abgeschlossen
    rjmp MOD_6:1 ; ==> Nächste Runde
:MOD_6:3
    pop r20
    pop r19
    pop r18
    pop r17
    ret ; Rücksprung aus MOD_6

```