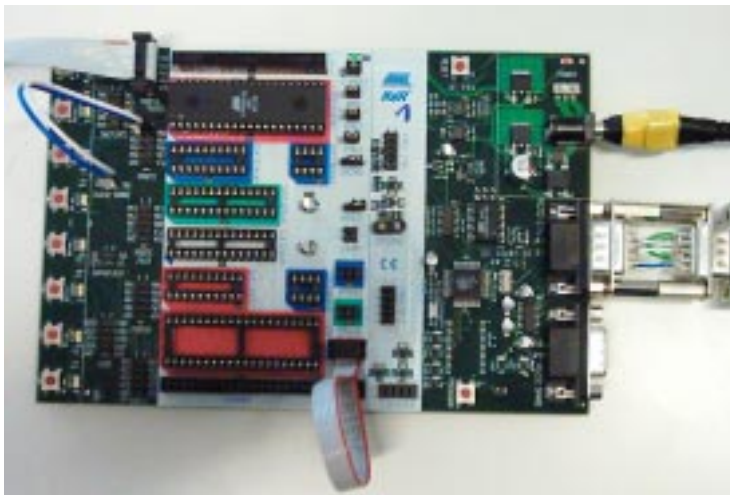


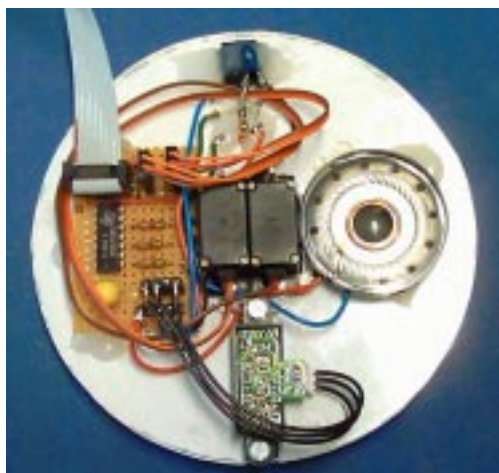
Übungsaufgabe zur Vorlesung "Eingebettete Systeme"

AtmUhr

Zu entwickeln ist die Software für einen Wecker. Das Programm läuft auf einem Atmel Prozessor (ATMEGA32) der sich auf dem Entwicklungsboard STK500 befindet und der über die serielle Schnittstelle vom PC programmiert wird. Die Steuerung der Uhr erfolgt über 8 Leitungen die den Port A des ATMEGA32 mit der Uhr-Hardware verbinden.



Die Uhr besitzt zwei Servos zur Anzeige der Zeit (Servo 0 für die Stunden, Servo 1 für die Minuten), eine grüne Leuchtdioden (an: AM, aus: PM), eine rote Leuchtdiode (an: Weckalarm an, aus: Weckalarm gesperrt), eine Infrarotempfangsdiode (zur Programmierung des Weckers mittels einer handelsüblichen Fernsehfernbedienung), einen Abstandssensor (zum Ausschalten des Wecksignals durch einer Handbewegung vor dem Sensor) sowie einen Lautsprecher für das Wecksignal.



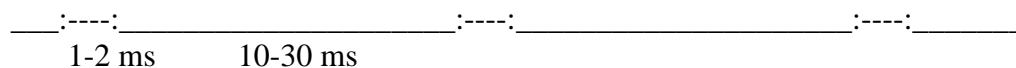
1. Hardware

1.1 Anschlußbelegung:

Port A, Pin0 (Ausgang)	Servo 0 (Stunde)
Port A, Pin1 (Ausgang)	Servo 1 (Minute)
Port A, Pin2 (Ausgang)	Lautsprecher (1: Strom an, 0: Strom aus)
Port A, Pin3 (Ausgang)	rote LED (1: LED an, 0: LED aus)
Port A, Pin4 (Ausgang)	grüne LED (1: LED an, 0: LED aus)
Port A, Pin5 (Eingang)	unbenutzt
Port A, Pin6 (Eingang)	IR-Diode
Port A, Pin7 (Eingang)	Abstandssensor

1.2 Ansteuerung eines Servos:

Der Servo erhält über die Steuerleitung periodisch (ca. alle 10 - 30 Millisekunden) einen kurzen 1 Impuls. Die Länge diese Impulses bestimmt die Position des Servos. Die Mittelstellung liegt bei ungefähr 1,5 ms, die beiden Endanschläge bei ca. 1 bzw. 2 ms.



1.3 Ansteuerung des Lautsprechers:

Das Alarmsignal wird durch ein periodisches Ein-/Ausschalten des Stroms durch den Lautsprecher erzeugt. Wird an Port A, Pin2 eine 1 ausgegeben, fließt Strom, bei einer 0 fließt kein Strom. Die Frequenz ist so zu wählen, daß der Ton gut hörbar ist. Falls das Alarmsignal aus ist, soll dieses Steuersignal aus Energiespargründen auf 0 gesetzt werden.

1.4 Ansteuerung der Leuchtdioden:

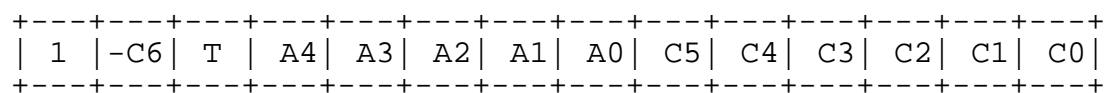
Eine 1 auf Pin 3(4) des Ports A schaltet die rote (grüne) LED ein, eine 0 schaltet die LED aus.

1.5 Lesen des Abstandssensors:

Der Abstandssensor liefert an Port A, Pin 7 eine analoge Spannung entsprechend dem Abstand des Sensors von einem Hindernis. Die Spannungswerte liegen zwischen 0,4 V (bei 30 cm) und 3 V (bei 4 cm)

1.6 Lesen des IR-Diode:

Die IR-Diode liefert eine 0 wenn ein 36 kHz Infrarotsignal empfangen wird, ansonsten eine 1. Eine Fernbedienung im RC5 Mode überträgt bei einem Tastendruck 14 Datenbits:



Die Übertragungszeit für ein Datenbit beträgt $2 * 889 \mu\text{s}$. Bei einem "1" Bit ist die ersten 889 μs das IR-Signal aus und in den zweiten 889 μs das 36 kHz IR-Signal an. Bei einem "0" Bit ist es umgekehrt. D.h. an Port A Pin 6 liegen bei der Übertragung einer "1" für 889 μs 5V und für die nächsten 889 μs 0V an, bei einer "0" umgekehrt.

A4-A0: Geräteadresse (TV=0, SAT=8, ...)

C6-C0: Kommando (z.B. 12=Aus, 16=Lautstärke+,)

Ursprünglich existierten nur 64 Codes (C5-C0), später wurde dann das zweite Startbit als negiertes C6 umdefiniert um 128 Codes übertragen zu können.

T Toggel-Bit: Bei jedem erneuten drücken einer Taste ändert sich dieses Bit. Dadurch kann das mehrfache Drücken einer Taste vom dauerhaften Drücken (repeat Funktion) unterschieden werden.

2. Bedienung der Uhr

2.1 Anzeige

Im Normalbetrieb zeigt die Uhr die augenblickliche Zeit (Stunde+Minute, keine Sekunde) an, wobei im Zeitraum 0:00-11:59 die grüne Leuchtdiode (AM) an ist. Wird die Hand vor den Abstandsensor gehalten, wird die eingestellte Weckzeit angezeigt.

2.2 Setzen der Uhrzeit

Eine neue Uhrzeit wird mittels der Fernbedienung (Tasten 0-9) eingegeben und durch drücken der Blauen Taste übernommen. Wird vor dem Drücken der Blauen Taste längere Zeit keine weitere Ziffer eingegeben, kehrt die Uhr zur Anzeige der alten Zeit zurück.

2.3 Setzen der Weckzeit

Die Weckzeit wird mittels der Fernbedienung (Tasten 0-9) eingegeben und durch drücken der Roten Taste übernommen. Wird vor dem Drücken der roten Taste längere Zeit keine weitere Ziffer eingegeben, kehrt die Uhr zur Anzeige der Zeit zurück.

2.4 Aktivieren der Weckfunktion

Die Weckfunktion wird durch Drücken der roten AUS Taste auf der Fernbedienung ein- bzw. ausgeschaltet. Bei aktivierter Weckfunktion leuchtet die rote LED. Der Alarmton ist an, wenn die augenblickliche Uhrzeit identisch mit der eingestellten Weckzeit ist und die Weckfunktion aktiviert ist (d.h. für maximal eine Minute). Das Ausschalten des Alarmtons erfolgt durch Drücken der STUMM Taste auf der Fernbedienung oder durch das Bewegen der Hand vor dem Abstandssensor.

3. Software

Das beigefügte Programm enthält bereits folgende Funktionen:

3.1 Initialisierungsroutinen

Initialisiert Stack, Port A, V24, Fernbedienung, Zeitzähler, A/D-Wandler, Servos, Timer0 und Timer2.

3.2 Interruptroutine für Timer 2

Diese Routine ist in der Lage bis zu 8 Servos anzusteuern (in dieser Anwendung werden nur die beiden Servos 0 und 1 benutzt). Das Hauptprogramm hat dabei nur Werte zwischen -50 und +50 (entspricht ca. -90° und $+90^\circ$ Auslenkung) in die Variablen *servo0* und *servo1* zu schreiben, den Rest übernimmt die Interruptroutine. Falls aufgrund von Fertigungstoleranzen eine Nullpunktskorrektur nötig ist, kann dies in der oben erwähnten Initialisierungsroutine erfolgen.

3.3 Interruptroutine für Timer 0

Diese Interruptroutine wird alle 64 μ s aufgerufen und hat mehrere Aufgaben:

- Inkrementieren des 24 Bit Zählers *ticks*
- Ausgeben des Alarmsignals
- Auswerten des IR-Signals von der Fernbedienung

Der Fernbedienungsteil ist bereits komplett implementiert und schreibt die gelesenen Tastencodes in einen Tastaturpuffer. Die repeat Funktion der Fernbedienung ist dabei ausgeschaltet (d.h. auch längeres Drücken einer Taste liefert nur einmal den Tastencode).

3.4 Eingabe über die IR-Fernbedienung

Die Anwendersoftware greift über folgende zwei (bereits implementierten) Routinen auf den Tastaturpuffer zu:

test_key: Z=0 falls Zeichen im Tastaturpuffer vorhanden sind
Z=1 falls kein Zeichen im Tastaturpuffer vorhanden ist
Register werden nicht verändert.

get_key: liest das nächste Zeichen aus dem Tastaturpuffer und übergibt es im Register r16 an das aufrufende Programm. Weitere Register werden nicht verändert. Falls kein Zeichen im Tastaturpuffer vorhanden ist, wartet die Routine bis ein Zeichen eingegeben wird.

3.5 Lesen Abstandssensors

get_abst: liest den vom Abstandssensor gelieferten Wert (0-255) ein und übergibt ihn im Register r16 an das aufrufende Programm. Weitere Register werden nicht verändert.

3.6 Hauptprogramm

Das Hauptprogramm benützt folgende Variablen:

ticks: 3 Byte, wird in der Interruptroutine alle 64 µs hochgezählt
time_h: aktuelle Zeit Stunde (0-23)
time_m: aktuelle Zeit Minute (0-59)
weck_h: Weckzeit Stunde (0-23)
weck_m: Weckzeit Minute (0-59)
weck_on: 0: Weckfunktion aus
 1: Weckfunktion an
 -1: Weckfunktion an, aber Wecksignal wurde bereits durch Fernbedienung oder Abstandssensor abgeschaltet

speak: 0 Signalton aus
 >0 Signalton ein

4. Aufgaben

Aufgabe 1:

Um zumindest eine minimale Debuggingmöglichkeit zu haben, wird die V24 Schnittstelle des Atmel Prozessors benutzt um Zeichen auf dem PC Bildschirm auszugeben. Stecken Sie dazu (nach dem Programmieren des Atmel) das V24-Kabel vom RS232 CTRL auf den RS232 SPARE Stecker des Atmelboards um und starten Sie auf dem PC ein beliebiges Terminalprogramm (9600 Baud, 8 Datenbit, kein Paritybit, 1 Stoppbit). Zur Ausgabe von Zeichen dient das (bereits vorhandene) Unterprogramm *out_r16*, welches das im Register r16 übergebene Byte über die V24 Schnittstelle ausgibt (dieses Unterprogramm verändert keine Registerinhalte).

- Schreiben Sie ein kleines Hauptprogramm (ab dem Label *main*:), das abwechselnd die Zeichen "a" und "b" auf dem Bildschirm des PC's ausgibt.
- Schreiben Sie ein Unterprogramm *out_hex_r16* das den Inhalt des Registers r16 als Hexadezimalzahl auf dem Bildschirm des PC's ausgibt Benutzen Sie dabei zur aktuellen Ausgabe *out_r16*.
- Schreiben Sie ein Hauptprogramm, das mittels *get_key* Zeichen von der Fernbedienung einliest und hexadezimal auf dem PC ausgibt. Erstellen Sie sich ein Tabelle über die Tastencodes der Fernbedienung.

- d) Schreiben Sie ein Hauptprogramm, das mittels *get_abst* den Abstandssensor ausliest und geben Sie diesen Wert hexadezimal auf dem PC aus. Notieren Sie sich den Wert für einen sinnvollen Abstand zum Ausschalten des Wecktons.

Aufgabe 2

Schreiben Sie ein Hauptprogramm das mittels von zwei frei wählbaren Tasten auf der Fernbedienung die grüne LED ein- bzw. ausschaltet.

Aufgabe 3

- a) Schreiben Sie ein Hauptprogramm das zunächst die Variablen *time_h*, *time_m*, *weck_h*, *weck_m*, *weck_on* und *speak* mit 0 initialisiert. Anschließend soll man in einer Endlosschleife mittels zwei frei wählbarer Tasten der Fernbedienung die Variable *time_h* (im gültigen Bereich 0-23) um Eins erhöhen bzw. erniedrigen werden können.
- b) Schreiben Sie ein Unterprogramm *display*, das den in der Variablen *time_h* gespeicherten Wert auf dem Stundenzeiger der Uhr ausgibt. Falls der Wert im Bereich 0-11 liegt, soll die grüne LED leuchten (AM), ansonsten soll *time_h*-12 (PM) ausgegeben werden. Dazu entnehmen Sie den der Stunde entsprechenden Wert aus der Tabelle *tab_h* und schreiben ihn in die Variable *servo0*. Da die einzelnen Servos voneinander abweichen können, müssen Sie eventuell die Werte in der Tabelle leicht anpassen. Rufen Sie dieses Unterprogramm aus dem Hauptprogramm nach jeder Veränderung von *time_h* auf.

Aufgabe 4

- a) Erweitern Sie das Hauptprogramm von Aufgabe 3 so, daß durch zwei andere Tasten auch die Variable *time_m* erhöht bzw. erniedrigt werden kann. Bei einem Überlauf der Minuten (<0 oder >59) soll die Stunde jeweils angepaßt werden.
- b) Erweitern Sie das Unterprogramm *display* von Aufgabe 3 so, daß es auch die Minuten auf dem zweiten servo anzeigt. Entnehmen Sie dazu die Werte für die Servoposition der Tabelle *tab_m*.
- c) Verändern Sie das Programm zum Anzeigen der Stunde so, daß abhängig von dem Minutenwert der Stundenzeiger auch zwischen den ganzen Stundenzahlen positioniert wird (durch Interpolation).

Aufgabe 5

Ersetzen Sie den Programmteil zum Hoch-/Herunterzählen der Stunden und Minuten durch eine Eingabeprogramm mit dem die Zeit durch das Ziffernfeld (0-9) direkt eingegeben werden kann. Durch anschließendes Drücken der Blauen Taste soll die eingegebene Zeit als Uhrzeit (*time_h*, *time_m*), durch Drücken der Roten Taste als Weckzeit (*weck_h*, *weck_m*) übernommen werden.

Aufgabe 6

- a) Erweitern Sie die Interruptroutine für den Timer 0 so, daß die 3 Byte Variable *ticks* bei jedem Interrupt um 1 erhöht wird. Falls ein Wert erreicht ist, der einer Minute entspricht, ist der Zähler auf Null zurück zu setzen und die Variable *time_m* (und gegebenenfalls *time_h*) anzupassen.
- b) Das Hauptprogramm soll in der Endlosschleife nun nicht nur auf Tastatureingaben warten, sondern auch kontinuierlich das Unterprogramm *display* aufrufen, damit die durch die Interruptroutine hochgezählte Zeit dargestellt wird (deshalb vor Aufruf von *get_key* mit *tst_key* testen ob ein Zeichen vorhanden ist).

Aufgabe 7

Erweitern Sie das Hauptprogramm, so daß die Uhr von der Darstellung der Uhrzeit zur Darstellung der Weckzeit wechselt, wenn die Hand vor den Abstandssensor gehalten wird (auch hier: grüne LED für AM).

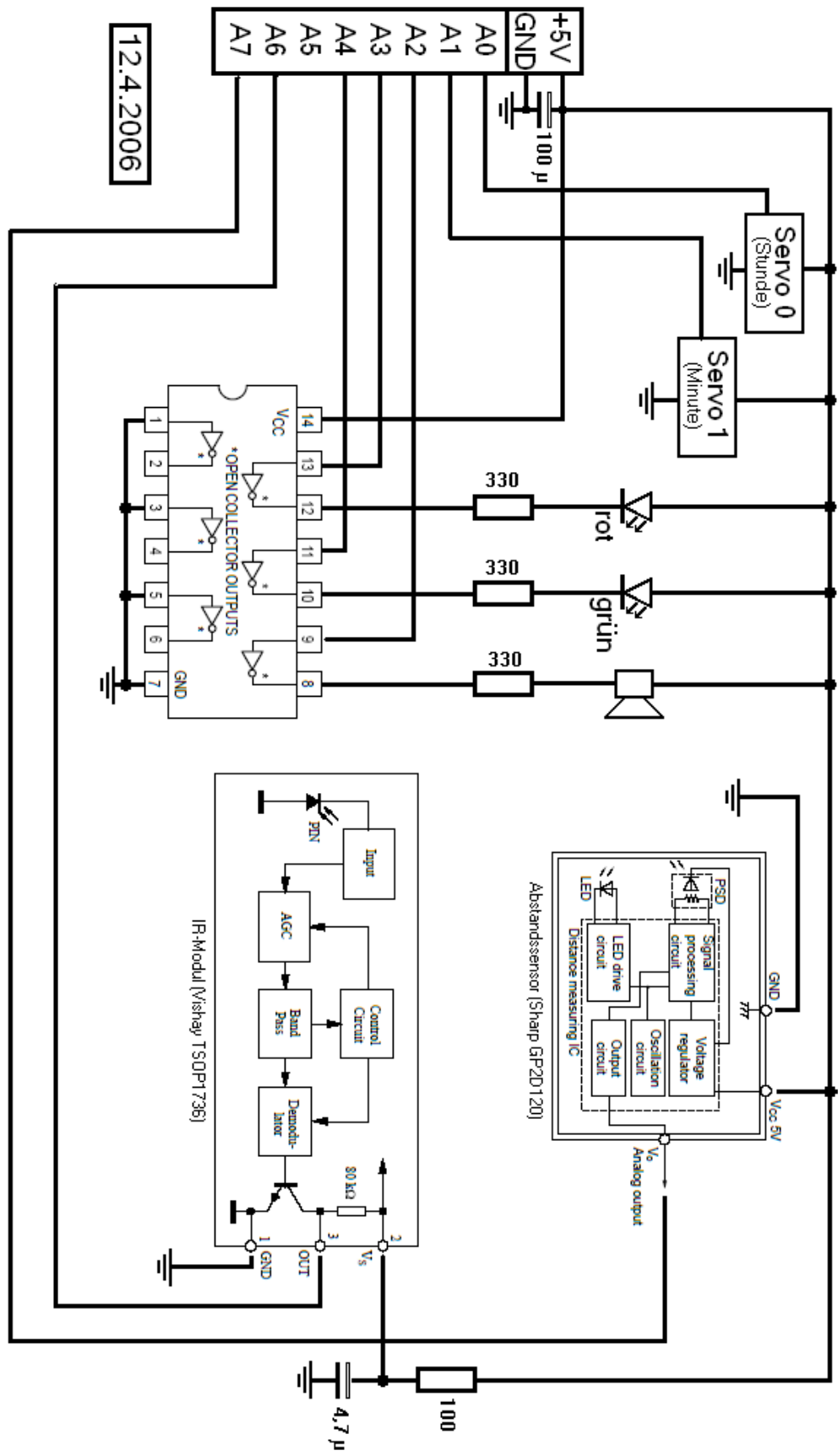
Aufgabe 8

- a) Verändern Sie das Hauptprogramm so, daß beim Drücken der AUS Taste die Variable *weck_on* von 0 nach 1 oder von ungleich 0 nach 0 wechselt.
- b) Verändern Sie das Unterprogramm *display* so, daß die rote LED leuchtet wenn die Variable *weck_on* ungleich 0 ist.

Aufgabe 9

- a) Ändern Sie das Hauptprogramm so, daß bei Drücken der STUMM Taste die Variable *speak* von 0 nach 1 oder von ungleich 0 nach 0 wechselt.
- b) Erweitern Sie die Interruptroutine für den Timer 0 so, daß ein gut hörbarer Ton auf dem Lautsprecher ausgegeben wird, wenn die Variable *speak* ungleich 0 ist.
- c) Die Variable *speak* soll nun nicht mehr durch die STUMM Taste gesetzt werden, sondern wenn die aktuelle Uhrzeit mit der eingestellten Weckzeit übereinstimmt und *weck_on* ungleich 0 ist (im Hauptprogramm, nicht in der Interruptroutine). Die Variable *speak* soll wieder zurück gesetzt werden, wenn entweder Zeit und Weckzeit nicht mehr gleich sind (d.h. nach einer Minute hört der Alarm von selbst auf), die STUMM Taste gedrückt wird oder wenn die Hand vor den Abstandssensor gehalten wird.

5. Schaltplan



5. Sourcecode (ADELA-Syntax)

```
debug=1
; a0: (out) servo stunde
; a1: (out) servo minute
; a2: (out) Lautsprecher (1:Strom an)
; a3: (out) rote LED (1:an)
; a4: (out) grüne LED (1:an)
; a6: (in) IR Diode
; a7: (in) Abstandssensor

; Interrupt Tabelle
jmp.l irq00_reset
jmp.l reset ; irq01_ext0
jmp.l reset ; irq02_ext1
jmp.l reset ; irq03_ext2
jmp.l irq04_timer2cmp
jmp.l irq05_timer2ovf
jmp.l reset ; irq06_timerlevent
jmp.l reset ; irq07_timer1cmpA
jmp.l reset ; irq08_timer1cmpB
jmp.l reset ; irq09_timerlovf
jmp.l irq0a_timer0cmp
jmp.l irq0b_timer0ovf
jmp.l reset ; irq0c_spi
jmp.l reset ; irq0d_V24Rx
jmp.l reset ; irq0e_V24dat
jmp.l reset ; irq0f_V24Tx
jmp.l reset ; irq10_ADC
jmp.l reset ; irq11_EEPROM
jmp.l reset ; irq12_analog
jmp.l reset ; irq13_2wire
jmp.l reset ; irq14_ProgMem

reset:
irq00_reset:
; ***** init stack *****
move.b #(ramend >> 8),r16
move.b r16,?sp_hi
move.b #(ramend & $ff),r16
move.b r16,?sp_lo
; *****

; ***** init ports *****
move.b #$1f,r21 ; a0-a4 output
move.b r21,?port_a_dir
; *****

; ***** init V24 *****
IF debug
move.b #51,r16
move.b r16,?v24_ctr4 ; 9600 baud, 8 MHz CPU

move.b #%10000110,r16 ; 1, asyn, no parity, 1 stop, 8 bit, 0
move.b r16,?v24_ctr3

move.b #%00001000,r16 ; no RX int, no TX int, no empty int, RX disable
move.b r16,?v24_ctr2 ; TX enable, 8 data bits, -, 9 bit
ENDIF
; *****

; ***** init Fernbedienung *****
move.b #(buf&$ff),r16
move.b r16,buf_ptr
move.b #(key&$ff),r16
move.b r16,key_read
move.b r16,key_write
move.b #0,r16
move.b r16,fb_mode
dec.b r16
move.b r16,last_key
; *****

; ***** init uhr *****
move.b #0,r16
move.b r16,ticks
move.b r16,ticks+1
move.b r16,ticks+2
; *****

; ***** init a/d wandler *****
move.b #%00100111,r16 ; ext. aref, left adj, pin a7
move.b r16,?adc_mux
; *****

; ***** init servos *****
move.b #(servo0 >> 8),r31
```

```

        move.b    #(servo0 & $ff),r30
        move.b    r30,servo_ptr
        move.b    r31,servo_ptr+1

        move.b    #8,r16
        move.b    #0,r17            ; pos=0
        move.b    #95,r18          ; nullpunkt = 95*16us = 1,5 ms
        move.b    #1,r19          ; port pin 0
_10:    move.b    r17,(r31|r30)+    ; pos
        move.b    r18,(r31|r30)+    ; nullpunkt
        move.b    r19,(r31|r30)+    ; pin
        add.b     r19,r19
        dec.b     r16
        bne.b     _10

        cmp.b     #255,r18
        beq.b     _20

        move.b    #4,r16            ; 4 x
        move.b    #255,r18          ; 2 ms pause
        br.w      _10
_20:    move.b    #92,r16            ; Nullpunktkorrektur fuer servo1
        move.b    r16,servol+1
; *****

; ***** init Timer 0 *****
; fuer Uhr, Sound und Fernbedienung
        move.b    #%00001011,r16    ; CTC mode, 8 MHz/64 =125 kHz
        move.b    r16,?timer0_ctr    ; 8 us Takt
        move.b    #8,r16            ; 8*8us=64us
        move.b    r16,?timer0_cmp
        move.b    ?timer_ienable,r16
        or.b      #2,r16            ; enable Timer0 compare match int
        move.b    r16,?timer_ienable
; *****

; ***** init Timer 2 *****
; fuer Servos
        move.b    #%00001101,r16    ; CTC mode, 8 MHz/128 =62,5 kHz
        move.b    r16,?timer2_ctr    ; 16 us Takt -> 255*16us = 4 ms
        move.b    ?timer_ienable,r16
        or.b      #$80,r16          ; enable Timer0 compare match int
        move.b    r16,?timer_ienable
; *****

; ***** enable Interrupts *****
        move.bit  #1,sr[7]          ; enable Interrupts
; *****

; ***** Hauptprogramm *****
main:   br.w      main
; *****

; ***** Uhrzeit ausgeben *****
display:                                ; r25: stunden    r24: minuten
        rts
; *****

; ***** Lesen Fernbedienung *****
get_key: ; gelesenes Zeichen -> r16
        move.b    r31,(sp)-
        move.b    r30,(sp)-
        move.b    #(key>>8),r31
        move.b    key_read,r30
_10:    move.b    key_write,r16
        cmp.b     r30,r16
        bne.b     _20
        sleep
        br.w      _10

_20:    move.b    (r31|r30)+,r16
        move.b    r30,r31
        and.b     #$1f,r31
        bne.b     _30
        move.b    #(key&$ff),r30
_30:    move.b    r30,key_read
        move.b    +(sp),r30
        move.b    +(sp),r31
        rts.w

tst_key: ; Z=1 fall kein Zeichen vorhanden
        move.b    r0,(sp)-
        move.b    r1,(sp)-
        move.b    key_read,r0
        move.b    key_write,r1
        cmp.b     r0,r1
        move.b    +(sp),r1
        move.b    +(sp),r0
        rts.w

```

```
; ***** Lesen Abstandssensor *****  
get_abst: ; Abstandssensor -> r16  
        move.b   #0x10101010,r16    ; adc enable, start conv., no auto trig.  
        move.b   r16,%adc_ctr       ; clear IF, int disable, prescale 64  
_l0:     skipcq.bit #1,%adc_ctr[4]   ; conversion ready  
        br.w     _l0                ; no  
        move.b   %adc_hi,r16  
        rts.w  
  
; *****  
  
IF debug  
; ***** Ausgabe V24 *****  
out_r16:  
_l0:     skipcq.bit #1,%v24_ctrl[5]   ; data register empty  
        br.w     _l0  
        move.b   r16,%v24_dat  
        rts.w  
  
out_hex_r16:  
        rts.w  
  
; *****  
ENDIF  
  
; ***** Timer0 overflow interrupt *****  
irq0b_timer0ovf:  
        rte  
;  
  
; ***** Timer0 compare interrupt *****  
; Aufruf alle 64 us  
; Zaehlt Uhr (ticks 32 bit) hoch  
;  
  
irq0a_timer0cmp:  
        move.b   r16,(sp)-  
        move.b   %sreg,r16  
        move.b   r16,(sp)-  
  
clock:  ; uncrement ticks, time_m und time_h  
  
ton:    ; Alarmton ausgeben  
  
fernbed:move.b   fb_mode,r16  
        sub.b    #1,r16  
        beq.b    _0sig             ; fb_mode=1  
        bpl.b    _nosig            ; fb_mode=2  
        br.w     _lsig             ; fb_mode=0  
  
;---+-----/|_|_|_|-\\|||\\\|\\\\\\|\n\n;\n33332222222222222222111111111111000000000000\n;\n43210987654321098765432109876543210
```

```
_nosig: move.b   fb_count,r16  
        skipcq.bit #1,%port_a_pin[6]  
        br.w     _l0              ; Eingangssignal=0  
  
        dec.b    r16              ; Wartezeit--  
        bpl.b    _20              ; nicht abgelaufen  
        eor.b    r16,r16          ; nicht negativ  
_20:     move.b   r16,fb_count  
        br.w     _endl  
  
_l0:     or.b     r16,r16           ; Wartezeit abgelaufen?  
        bne.b    _err1            ; nein  
        move.b   #1,r16  
        move.b   r16,fb_mode      ; 0 Signal  
        move.b   #34,r16  
        move.b   r16,fb_count     ; max. 34*64us = 2.18 ms "0"  
        br.w     _endl  
  
_0sig:   move.b   r17,(sp)-  
        move.b   r30,(sp)-  
        move.b   r31,(sp)-  
        move.b   fb_count,r16  
        skipcq.bit #0,%port_a_pin[6]  
        br.w     _30              ; Eingangssignal=1  
        dec.b    r16              ; max. Laenge ueberschritten  
        bmi.b    _err2            ; ja  
        move.b   r16,fb_count  
        br.w     _end2  
  
_30:     move.b   #0,r17  
_50:     move.b   #(buf>>8),r31  
        move.b   buf_ptr,r30  
        cmp.b    #(buf+27)&$ff,r30 ; schon 27 0/1 empfangen
```

```

bhs.b _err2 ; ja, dann Fehler
cmp.b #26,r16 ; weniger als (34-26)*64us = 512 us
bhs.b _err2 ; ja, dann Fehler
cmp.b #15,r16 ; 512us < t < 1152us
bhs.b _40 ; ja, dann eine 0/1
cmp.b #13,r16 ; 1152us < t < 1408 us
bhs.b _err2 ; ja, dann Fehler
move.b r17,(r31|r30)+ ; 1408us < t < 2176us : zwei 0/1
_40: move.b r17,(r31|r30)+
move.b r30,buf_ptr
move.b r17,fb_mode ; 1/0 Signal
move.b #34,r16
move.b r16,fb_count ; max. 34*64us = 2.18 ms "1"
; br.w _end2

_end2: move.b +(sp),r31
move.b +(sp),r30
move.b +(sp),r17
_end1: move.b +(sp),r16
move.b r16,?sreg
move.b +(sp),r16
rte

_err2: move.b +(sp),r31
move.b +(sp),r30
move.b +(sp),r17
_err1: move.b #2,r16
move.b r16,fb_mode ; kein Signal
move.b #56,r16
move.b r16,fb_count ; mindestens 3.5 ms inaktiv
move.b #buf+$ff,r16
move.b r16,buf_ptr ; buf_ptr zurueck setzen
br.w _end1

_lsig: move.b r17,(sp)-
move.b r30,(sp)-
move.b r31,(sp)-

move.b fb_count,r16
move.b #1,r17
skipeq.bit #1,?port_a_pin[6]
br.w _50 ; Eingangssignal=0
dec.b r16 ; max. Laenge ueberschritten
move.b r16,fb_count
bpl.b _end2 ; nein, dann weiter warten
move.b buf_ptr,r30
cmp.b #(buf+26)&$ff,r30 ; mindestens 26 0/1 empfangen
blo.b _err2 ; nein, dann Fehler
move.b buf+3,r17 ; Toggle bit
add.b r17,r17 ; soll MSB werden
move.b #(buf>>8),r31
move.b #(buf+15)&$ff,r30 ; Zeiger auf 1. Kommando Bit

_60: move.b (r31|r30)+,r16 ; naechstes Kommando Bit
add.b r17,r17 ; alter Wert nach links
or.b r16,r17 ; + neues Bit
inc.b r30
cmp.b #(buf+26)&$ff,r30 ; alle 6 Kommando Bits bearbeitet
blo.b _60 ; nein, dann weiter

move.b last_key,r16 ; kein Tastenrepeat
cmp.b r16,r17 ; gleiche Taste wie letzte?
beq.b _err2 ; ja, dann ignorieren
move.b r17,last_key
and.b #$3f,r17 ; nur 6 bit

move.b #key>>8,r31
move.b key_write,r30 ; Schreibzeiger
move.b r17,(r31|r30)+ ; empfangenes Kommando abspeichern
move.b r30,r17
and.b #$1f,r17 ; naechstes zu schr. Byte (0-31)
bne.b _70 ; falls nicht 0, kein wrap around
move.b #(key&$ff),r30 ; sonst Pufferanfang eintragen
_70: move.b key_read,r16
skipeq.b r30,r16 ; Lesepointer gleich neuem Schreibpointer
move.b r30,key_write ; nein, dann Schreibpointer aktualisieren
br.w _err2

; *****

; ***** Timer2 overflow interrupt *****
irq05_timer2ovf:
rte.w
; *****

; ***** Timer2 compare interrupt *****
mask=%00000011 ; nur servo0 und servo1

```

```

irq04_timer2cmp:
    move.b    r16,(sp)-
    move.b    r17,(sp)-
    move.b    r18,(sp)-
    move.b    r30,(sp)-
    move.b    r31,(sp)-
    move.b    ?sreg,r16
    move.b    r16,(sp)-

    move.b    servo_ptr,r30
    move.b    servo_ptr+1,r31

    move.b    (r31|r30)+,r16
    move.b    (r31|r30)+,r17
    add.b     r16,r17
    move.b    (r31|r30)+,r16
    and.b     #mask,r16
    move.b    ?port_a_dat,r18
    and.b     #-mask,r18
    or.b      r18,r16
    move.b    r16,?port_a_dat
    move.b    r17,?timer2_cmp

    cmp.b     #(servo_end & $ff),r30
    bne.b     _10
    move.b    #(servo0 >> 8),r31
    move.b    #(servo0 & $ff),r30
_10: move.b    r30,servo_ptr
    move.b    r31,servo_ptr+1

    move.b    +(sp),r16
    move.b    r16,?sreg
    move.b    +(sp),r31
    move.b    +(sp),r30
    move.b    +(sp),r18
    move.b    +(sp),r17
    move.b    +(sp),r16
    rte.w

tab_h:  dc.b    -48,-40,-32,-24,-16, -8,  0,  8 ,16 ,24, 32,  40, 48, 0
;        0   1   2   3   4   5   6   7   8   9  10  11  12
tab_m:  dc.b    48, 46, 45, 43, 42, 40, 38, 37 ,35 ,34 ; 0x
    dc.b    32, 30, 29, 27, 26, 24, 22, 21 ,19 ,18 ; 1x
    dc.b    16, 14, 13, 11, 10,  8,  6,  5 , 3 , 2 ; 2x
    dc.b     0, -2, -3, -5, -6, -8,-10,-11 ,-13,-14; 3x
    dc.b   -16,-18,-19,-21,-22,-24,-26,-27 ,-29,-30; 4x
    dc.b   -32,-34,-35,-37,-38,-40,-42,-43 ,-45,-46; 5x

even

; *****

; *****
; IO Adressraum
; *****

@=0

twbr:    blk.b    1      ; 00
twsr:    blk.b    1      ; 01
twar:    blk.b    1      ; 02
twdr:    blk.b    1      ; 03
adc_lo:   blk.b    1      ; 04 ADCL
adc_hi:   blk.b    1      ; 05 ADCH
adc_ctr:  blk.b    1      ; 06 ADCSRA
adc_mux:  blk.b    1      ; 07 ADMUX
acsr:     blk.b    1      ; 08
v24_ctr4: blk.b    1      ; 09 UBRR
v24_ctr2: blk.b    1      ; 0a UCSRB
v24_ctr1: blk.b    1      ; 0b UCSRA
v24_dat:  blk.b    1      ; 0c UDR
spsr:     blk.b    1      ; 0d
spdr:     blk.b    1      ; 0e
spdr:     blk.b    1      ; 0f
port_d_pin: blk.b    1      ; 10 PIND
port_d_dir: blk.b    1      ; 11 DDRD
port_d_dat: blk.b    1      ; 12 PORTD
port_c_pin: blk.b    1      ; 13 PINC
port_c_dir: blk.b    1      ; 14 DDRC
port_c_dat: blk.b    1      ; 15 PORTC
port_b_pin: blk.b    1      ; 16 PINB
port_b_dir: blk.b    1      ; 17 DDRB
port_b_dat: blk.b    1      ; 18 PORTB
port_a_pin: blk.b    1      ; 19 PINA
port_a_dir: blk.b    1      ; 1a DDRA

```

```

port_a_dat:      blk.b  1      ; 1b PORTA
eecr:            blk.b  1      ; 1c
eedr:            blk.b  1      ; 1d
eearl:           blk.b  1      ; 1e
eearh:           blk.b  1      ; 1f
v24_ctr3:        blk.b  1      ; 20 UCSRC
wdtcr:           blk.b  1      ; 21
assr:            blk.b  1      ; 22
timer2_cmp:       blk.b  1      ; 23 OCR2
timer2_dat:       blk.b  1      ; 24 TCNT2
timer2_ctr:       blk.b  1      ; 25 TCCR2
icr1l:           blk.b  1      ; 26
icr1h:           blk.b  1      ; 27
ocr1bl:          blk.b  1      ; 28
ocr1bh:          blk.b  1      ; 29
ocr1al:          blk.b  1      ; 2a
ocr1ah:          blk.b  1      ; 2b
tcnt1l:          blk.b  1      ; 2c
tcnt1h:          blk.b  1      ; 2d
tccr1b:          blk.b  1      ; 2e
tccr1a:          blk.b  1      ; 3f
sfior:           blk.b  1      ; 30
ocdr:            blk.b  1      ; 31
timer0_dat:       blk.b  1      ; 32 TCNT0
timer0_ctr:       blk.b  1      ; 33 TCCR0
mcucsr:          blk.b  1      ; 34
mcucr:           blk.b  1      ; 35
twcr:            blk.b  1      ; 36
spmcr:           blk.b  1      ; 37
timer_iflag:      blk.b  1      ; 38 TIFR
timer_ienable:    blk.b  1      ; 39 TIMSK
gifr:            blk.b  1      ; 3a
gicr:            blk.b  1      ; 3b
timer0_cmp:       blk.b  1      ; 3c OCR0
sp_lo:           blk.b  1      ; 3d SPL
sp_hi:           blk.b  1      ; 3e SPH
sreg:            blk.b  1      ; 3f

; *****
; SRAM Adressraum
; *****

@=0
      blk.b  32      ; register
      blk.b  64      ; IO register

ticks:      blk.b  4

speak:      blk.b  1
time_h:     blk.b  1
time_m:     blk.b  1
weck_on:    blk.b  1
weck_h:     blk.b  1
weck_m:     blk.b  1

fb_mode:    blk.b  1
fb_count:   blk.b  1
buf_ptr:    blk.b  1
key_read:   blk.b  1
key_write:  blk.b  1
last_key:   blk.b  1

servo_ptr:  blk.b  2
servo0:     blk.b  3      ; wert (-128 .. +127); nullpunkt (~175)
servo1:     blk.b  3
servo2:     blk.b  3
servo3:     blk.b  3
servo4:     blk.b  3
servo5:     blk.b  3
servo6:     blk.b  3
servo7:     blk.b  3
pause0:     blk.b  3
pause1:     blk.b  3
pause2:     blk.b  3
pause3:     blk.b  3
servo_end:

      blk.b  ((@+31)>>5)<<5)-@
buf:        blk.b  32
key:        blk.b  32
ramend=$85f

```

6. Sourcecode (ATMEL-Syntax)

```
.EQU debug=1
; a0: (out) servo stunde
; a1: (out) servo minute
; a2: (out) Lautsprecher (1:Strom an)
; a3: (out) rote LED (1:an)
; a4: (out) grueene LED (1:an)
; a6: (in) IR Diode
; a7: (in) Abstandssensor

.ORG 0

; Interrupt Tabelle
jmp irq00_reset ; irq00_reset
jmp reset ; irq01_ext0
jmp reset ; irq02_ext1
jmp reset ; irq03_ext2
jmp irq04_timer2cmp
jmp irq05_timer2ovf
jmp reset ; irq06_timerlevent
jmp reset ; irq07_timer1cmpA
jmp reset ; irq08_timer1cmpB
jmp reset ; irq09_timerlovf
jmp irq0a_timer0cmp
jmp irq0b_timer0ovf
jmp reset ; irq0c_spi
jmp reset ; irq0d_V24Rx
jmp reset ; irq0e_V24dat
jmp reset ; irq0f_V24Tx
jmp reset ; irq10_ADC
jmp reset ; irq11_EEPROM
jmp reset ; irq12_analog
jmp reset ; irq13_2wire
jmp reset ; irq14_ProgMem

reset:
irq00_reset:
; ***** init stack *****
ldi r16,(ramend >> 8)
out sp_hi,r16
ldi r16,(ramend & $ff)
out sp_lo,r16
; *****
; ***** init ports *****
ldi r21,$1f ; a0-a4 output
out port_a_dir,r21
; *****
; ***** init V24 *****
; IF debug
ldi r16,51
out v24_ctr4,r16 ; 9600 baud, 8 MHz CPU

ldi r16,0b10000110 ; 1, asyn, no parity, 1 stop, 8 bit, 0
out v24_ctr3,r16

ldi r16,0b00001000 ; no RX int, no TX int, no empty int, RX disable
out v24_ctr2,r16 ; TX enable, 8 data bits, -, 9 bit
; ENDDIF
; *****
; ***** init Fernbedienung *****
ldi r16,(buf&$ff)
sts buf_ptr,r16
ldi r16,(key&$ff)
sts key_read,r16
sts key_write,r16
ldi r16,0
sts fb_mode,r16
dec r16
sts last_key,r16
; *****
; ***** init uhr *****
ldi r16,0
sts ticks,r16
sts ticks+1,r16
sts ticks+2,r16
; *****
; ***** init a/d wandler *****
ldi r16,0b00100111 ; ext. aref, left adj, pin a7
out adc_mux,r16
; *****
; ***** init servos *****
```

```

        ldi        r31,(servo0 >> 8)
        ldi        r30,(servo0 & $ff)
        sts        servo_ptr,r30
        sts        servo_ptr+1,r31

        ldi        r16,8
        ldi        r17,0                ; pos=0
        ldi        r18,95                ; nullpunkt = 95*16us = 1,5 ms
        ldi        r19,1                ; port pin 0
a10:    st         Z+,r17                ; pos
        st         Z+,r18                ; nullpunkt
        st         Z+,r19                ; pin
        add        r19,r19
        dec        r16
        brne       a10

        cpi        r18,255
        breq       a20

        ldi        r16,4                ; 4 x
        ldi        r18,255                ; 2 ms pause
        rjmp       a10
a20:    ldi        r16,92                ; Nullpunktkorrektur fuer servol
        sts        servol+1,r16
; *****

; ***** init Timer 0 *****
; fuer Uhr, Sound und Fernbedienung
        ldi        r16,0b00001011        ; CTC mode, 8 MHz/64 =125 kHz
        out        timer0_ctr,r16        ; 8 us Takt
        ldi        r16,8                ; 8*8us=64us
        out        timer0_cmp,r16
        in         r16,timer_ienable
        ori        r16,2                ; enable Timer0 compare match int
        out        timer_ienable,r16
; *****

; ***** init Timer 2 *****
; fuer Servos
        ldi        r16,0b00001101        ; CTC mode, 8 MHz/128 =62,5 kHz
        out        timer2_ctr,r16        ; 16 us Takt -> 255*16us = 4 ms
        in         r16,timer_ienable
        ori        r16,$80                ; enable Timer0 compare match int
        out        timer_ienable,r16
; *****

; ***** enable Interrupts *****
        sei                ; enable Interrupts
; *****

; ***** Hauptprogramm *****
main:    rjmp       main
; *****

; ***** Uhrzeit ausgeben *****
display:                ; r25: stunden    r24: minuten
        ret
; *****

; ***** Lesen Fernbedienung *****
get_key: ; gelesenes Zeichen -> r16
        push       r31
        push       r30
        ldi        r31,(key>>8)
        lds        r30,key_read
b10:    lds        r16,key_write
        cp         r16,r30
        brne       b20
        sleep
        rjmp       b10

b20:    ld         r16,Z+
        mov        r31,r30
        andi       r31,$1f
        brne       b30
        ldi        r30,(key&$ff)
b30:    sts        key_read,r30
        pop        r30
        pop        r31
        ret

tst_key: ; Z=1 fall kein Zeichen vorhanden
        push       r0
        push       r1
        lds        r0,key_read
        lds        r1,key_write
        cp         r1,r0
        pop        r1
        pop        r0

```



```
ret
; *****
; ***** Lesen Abstandssensor *****
get_abst: ; Abstandssensor -> r16
        ldi    r16,0b11010110      ; adc enable, start conv., no auto trig.,
        out    adc_ctr,r16          ; clear IF, int disable, prescale 64
c10:     sbis   adc_ctr,4             ; conversion ready
        rjmp   c10                  ; no
        in     r16,adc_hi
        ret
; *****

;IF debug
; ***** Ausgabe V24 *****
out_r16:
        db10:  sbis   v24_ctrl,5      ; data register empty
        rjmp   d10
        out    v24_dat,r16
        ret

out_hex_r16:
        ret
; *****
;ENDIF

; ***** Timer0 overflow interrupt *****
irq0b_timer0ovf:
        reti
; *****

; ***** Timer0 compare interrupt *****
; Aufruf alle 64 us
; Zaehlt Uhr (ticks 32 bit) hoch
;

irq0a_timer0cmp:
        push   r16
        in     r16,sreg
        push   r16

clock:   ; uncrement ticks, time_m und time_h

ton:     ; Alarmton ausgeben

fernbed:lds    r16,fb_mode
        subi   r16,1
        breq   _0sig           ; fb_mode=1
        brpl   _nosig          ; fb_mode=2
        rjmp   _lsig           ; fb_mode=0

;---+-----/|||||/////--\\\\\\\\|\\\\\\\\
; 33332222222222221111111111100000000000
; 43210987654321098765432109876543210

_nosig: lds     r16,fb_count
        sbis   port_a_pin,6
        rjmp   e10            ; Eingangssignal=0

        dec    r16              ; Wartezeit--
        brpl   e20              ; nicht abgelaufen
        eor     r16,r16         ; nicht negativ
e20:     sts     fb_count,r16
        rjmp   _endl

e10:     or      r16,r16         ; Wartezeit abgelaufen?
        brne   _err1           ; nein
        ldi     r16,1
        sts     fb_mode,r16     ; 0 Signal
        ldi     r16,34
        sts     fb_count,r16    ; max. 34*64us = 2.18 ms "0"
        rjmp   _endl

_0sig:   push    r17
        push    r30
        push    r31
        lds     r16,fb_count
        sbic    port_a_pin,6
        rjmp   e30            ; Eingangssignal=1
        dec     r16            ; max. Laenge ueberschritten
        brmi    _err2         ; ja
        sts     fb_count,r16
        rjmp   _end2

e30:     ldi     r17,0
e50:     ldi     r31,(buf>>8)
        lds     r30,buf_ptr
```

```

        cpi        r30,(buf+27)&$ff ; schon 27 0/1 empfangen
        brcc       _err2           ; ja, dann Fehler
        cpi        r16,26         ; weniger als (34-26)*64us = 512 us
        brcc       _err2           ; ja, dann Fehler
        cpi        r16,15         ; 512us < t < 1152us
        brcc       e40            ; ja, dann eine 0/1
        cpi        r16,13         ; 1152us < t y 1408 us
        brcc       _err2           ; ja, dann Fehler
        st         Z+,r17         ; 1408us < t < 2176us : zwei 0/1
e40:    st         Z+,r17
        sts        buf_ptr,r30
        sts        fb_mode,r17    ; 1/0 Signal
        ldi        r16,34
        sts        fb_count,r16   ; max. 34*64us = 2.18 ms "1"
;
        rjmp       _end2

_end2:   pop       r31
        pop       r30
        pop       r17
_end1:   pop       r16
        out       sreg,r16
        pop       r16
        reti

_err2:   pop       r31
        pop       r30
        pop       r17
_err1:   ldi       r16,2
        sts        fb_mode,r16    ; kein Signal
        ldi       r16,56
        sts        fb_count,r16   ; mindestens 3.5 ms inaktiv
        ldi       r16,buf&$ff
        sts        buf_ptr,r16    ; buf_ptr zurueck setzen
        rjmp       _end1

_lsig:   push      r17
        push      r30
        push      r31

        lds       r16,fb_count
        ldi       r17,1
        sbis      port_a_pin,6
        rjmp      e50            ; Eingangssignal=0
        dec       r16            ; max. Laenge ueberschritten
        sts        fb_count,r16
        brpl      _end2          ; nein, dann weiter warten
        lds       r30,buf_ptr
        cpi       r30,(buf+26)&$ff ; mindestens 26 0/1 empfangen
        brcs      _err2          ; nein, dann Fehler
        lds       r17,buf+3       ; Toggle bit
        add       r17,r17         ; soll MSB werden
        ldi       r31,(buf>>8)
        ldi       r30,(buf+15)&$ff ; Zeiger auf 1. Kommando Bit

e60:    ld         r16,Z+          ; naechstes Kommando Bit
        add       r17,r17         ; alter Wert nach links
        or        r17,r16        ; + neues Bit
        inc       r30
        cpi       r30,(buf+26)&$ff ; alle 6 Kommando Bits bearbeitet
        brcs      e60            ; nein, dann weiter

        lds       r16,last_key    ; kein Tastenrepeat
        cp        r17,r16         ; gleiche Taste wie letzte?
        breq      _err2           ; ja, dann ignorieren
        sts        last_key,r17
        andi      r17,$3f         ; nur 6 bit

        ldi       r31,key>>8
        lds       r30,key_write   ; Schreibzeiger
        st        Z+,r17          ; empfangenes Kommando abspeichern
        mov       r17,r30
        andi      r17,$1f         ; naechstes zu schr. Byte (0-31)
        brne      e70            ; falls nicht 0, kein wrap around
        ldi       r30,(key&$ff)   ; sonst Pufferanfang eintragen
e70:    lds       r16,key_read
        cpse      r16,r30         ; Lesepointer gleich neuem Schreibpointer
        sts        key_write,r30   ; nein, dann Schreibpointer aktualisieren
        rjmp      _err2

; *****

; ***** Timer2 overflow interrupt *****
irq05_timer2ovf:
        reti
; *****

; ***** Timer2 compare interrupt *****

```

```

.EQU mask=0b00000011 ; nur servo0 und servo1

irq04_timer2cmp:
    push    r16
    push    r17
    push    r18
    push    r30
    push    r31
    in      r16,sreg
    push    r16

    lds     r30,servo_ptr
    lds     r31,servo_ptr+1

    ld      r16,Z+
    ld      r17,Z+
    add     r17,r16
    ld      r16,Z+
    andi    r16,mask
    in      r18,port_a_dat
    andi    r18,~mask
    or      r16,r18
    out     port_a_dat,r16
    out     timer2_cmp,r17

    cpi     r30,(servo_end & $ff)
    brne    f10
    ldi     r31,(servo0 >> 8)
    ldi     r30,(servo0 & $ff)
f10: sts    servo_ptr,r30
    sts     servo_ptr+1,r31

    pop     r16
    out     sreg,r16
    pop     r31
    pop     r30
    pop     r18
    pop     r17
    pop     r16
    reti

tab_h: .DB    -48,-40,-32,-24,-16, -8,  0,  8 ,16 ,24, 32,  40, 48, 0
;
tab_m: .DB    48, 46, 45, 43, 42, 40, 38, 37 ,35 ,34 ; 0x
        .DB    32, 30, 29, 27, 26, 24, 22, 21 ,19 ,18 ; 1x
        .DB    16, 14, 13, 11, 10,  8,  6,  5 , 3 , 2 ; 2x
        .DB    0, -2, -3, -5, -6, -8,-10,-11 ,-13,-14; 3x
        .DB    -16,-18,-19,-21,-22,-24,-26,-27 ,-29,-30; 4x
        .DB    -32,-34,-35,-37,-38,-40,-42,-43 ,-45,-46; 5x

; *****

;*****
; IO Adressraum
;*****

.DSEG
.ORG 0

twbr:      .BYTE 1      ; 00
twsr:      .BYTE 1      ; 01
twar:      .BYTE 1      ; 02
twdr:      .BYTE 1      ; 03
adc_lo:    .BYTE 1      ; 04 ADCL
adc_hi:    .BYTE 1      ; 05 ADCH
adc_ctr:   .BYTE 1      ; 06 ADCSRA
adc_mux:   .BYTE 1      ; 07 ADMUX
acsr:      .BYTE 1      ; 08
v24_ctr4:  .BYTE 1      ; 09 UBRR1
v24_ctr2:  .BYTE 1      ; 0a UCSRB
v24_ctr1:  .BYTE 1      ; 0b UCSRA
v24_dat:   .BYTE 1      ; 0c UDR
spsr:      .BYTE 1      ; 0d
spsr:      .BYTE 1      ; 0e
spdr:      .BYTE 1      ; 0f
port_d_pin: .BYTE 1      ; 10 PIND
port_d_dir: .BYTE 1      ; 11 DDRD
port_d_dat: .BYTE 1      ; 12 PORTD
port_c_pin: .BYTE 1      ; 13 PINC
port_c_dir: .BYTE 1      ; 14 DDRC
port_c_dat: .BYTE 1      ; 15 PORTC
port_b_pin: .BYTE 1      ; 16 PINB
port_b_dir: .BYTE 1      ; 17 DDRB
port_b_dat: .BYTE 1      ; 18 PORTB

```

```

port_a_pin:      .BYTE 1      ; 19 PINA
port_a_dir:      .BYTE 1      ; 1a DDRA
port_a_dat:      .BYTE 1      ; 1b PORTA
eecr:            .BYTE 1      ; 1c
eedr:            .BYTE 1      ; 1d
eearl:           .BYTE 1      ; 1e
eeahr:           .BYTE 1      ; 1f
v24_ctr3:        .BYTE 1      ; 20 UCSRC
wdtcr:           .BYTE 1      ; 21
assr:            .BYTE 1      ; 22
timer2_cmp:       .BYTE 1      ; 23 OCR2
timer2_dat:       .BYTE 1      ; 24 TCNT2
timer2_ctr:       .BYTE 1      ; 25 TCCR2
icr1l:           .BYTE 1      ; 26
icr1h:           .BYTE 1      ; 27
ocr1bl:          .BYTE 1      ; 28
ocr1bh:          .BYTE 1      ; 29
ocr1al:          .BYTE 1      ; 2a
ocr1ah:          .BYTE 1      ; 2b
tcnt1l:          .BYTE 1      ; 2c
tcnt1h:          .BYTE 1      ; 2d
tccr1b:          .BYTE 1      ; 2e
tccr1a:          .BYTE 1      ; 3f
sfior:           .BYTE 1      ; 30
ocdr:            .BYTE 1      ; 31
timer0_dat:       .BYTE 1      ; 32 TCNT0
timer0_ctr:       .BYTE 1      ; 33 TCCR0
mcucsr:          .BYTE 1      ; 34
mcucr:           .BYTE 1      ; 35
twcr:            .BYTE 1      ; 36
spmcr:           .BYTE 1      ; 37
timer_iflag:      .BYTE 1      ; 38 TIFR
timer_ienable:    .BYTE 1      ; 39 TIMSK
gifr:            .BYTE 1      ; 3a
gicr:            .BYTE 1      ; 3b
timer0_cmp:       .BYTE 1      ; 3c OCR0
sp_lo:           .BYTE 1      ; 3d SPL
sp_hi:           .BYTE 1      ; 3e SPH
sreg:            .BYTE 1      ; 3f

; *****
;   SRAM Adressraum
; *****

;   .DSEG
;   .ORG 0
;   .BYTE 32      ; register
;   .BYTE 64      ; IO register

.DSEG
.ORG 96

ticks:           .BYTE 4

speak:           .BYTE 1
time_h:          .BYTE 1
time_m:          .BYTE 1
weck_on:         .BYTE 1
weck_h:          .BYTE 1
weck_m:          .BYTE 1

fb_mode:         .BYTE 1
fb_count:        .BYTE 1
buf_ptr:         .BYTE 1
key_read:        .BYTE 1
key_write:       .BYTE 1
last_key:        .BYTE 1

servo_ptr:       .BYTE 2
servo0:          .BYTE 3      ; wert (-128 .. +127); nullpunkt (~175)
servo1:          .BYTE 3
servo2:          .BYTE 3
servo3:          .BYTE 3
servo4:          .BYTE 3
servo5:          .BYTE 3
servo6:          .BYTE 3
servo7:          .BYTE 3
pause0:          .BYTE 3
pause1:          .BYTE 3
pause2:          .BYTE 3
pause3:          .BYTE 3
servo_end:

x:               .BYTE ((x+31)>>5)<<5)-x
buf:             .BYTE 32
key:             .BYTE 32
.EQU ramend=$85f

```