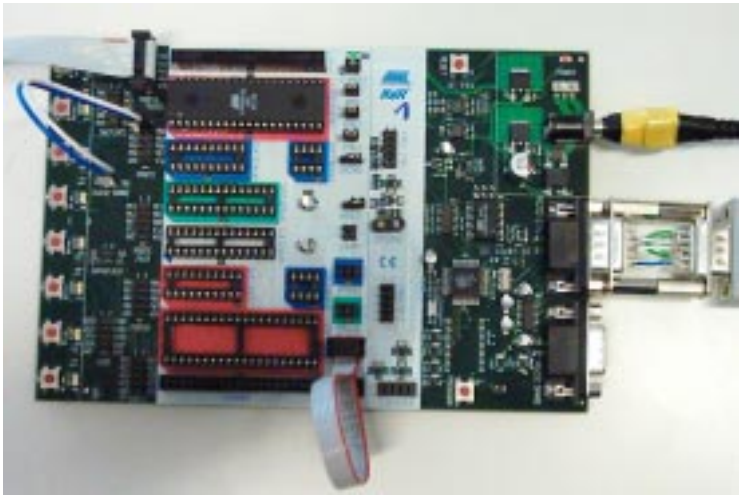
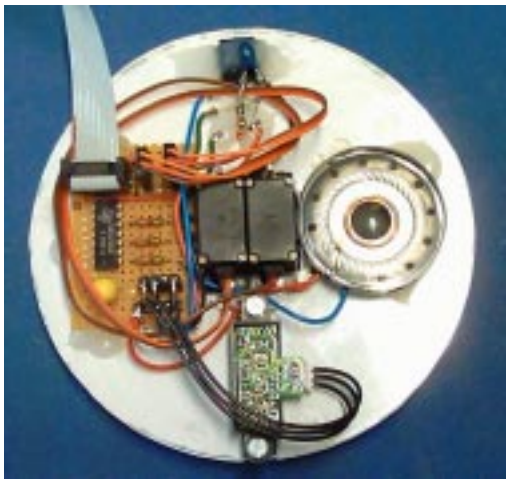
Exercise for "Embedded Systems"

AtmUhr

In this exercise the software for an alarm clock is developed step by step. The program is executed by an Atmel processor (ATMEGA32) located on the development board STK500. Port A of the ATMEGA32 is used to control the alarm clock.



The clock uses two servos to display the time (servo 0 for hours, servo 1 for minutes), a green LED (on: AM, off: PM), a red LED (on: alarm enabled, off: alarm disabled), an IR diode (for programming the clock with a usual IR remote control), a distance sensor (to switch off the alarm by moving the hand in front of the sensor) and a speaker for the alarm.



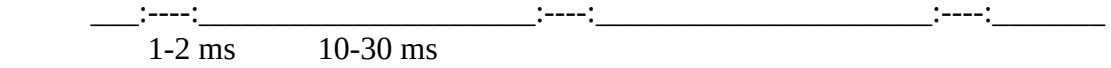
1. Hardware

1.1 Connection:

Port A, Pin0 (output)	servo 0 (hour)
Port A, Pin1 (output)	servo 1 (minute)
Port A, Pin2 (output)	speaker (1: current on, 0: current off)
Port A, Pin3 (output)	red LED (1: LED on, 0: LED off)
Port A, Pin4 (output)	green LED (1: LED on, 0: LED off)
Port A, Pin5 (input)	unused
Port A, Pin6 (input)	IR diode
Port A, Pin7 (input)	distance sensor

1.2 Controlling the servos:

The servo gets periodically (10 - 30 milliseconds) a short 1 impulse. The duration of the pulses determines the position of the servo. For a middle position the pulse must be about 1.5 ms and for the both end positions about 1 respectively 2 ms.



1.3 Controlling the speaker:

The alarm signal is generated by switching the current through the speaker periodically on/off. If a 1 is written to Port A, Pin 2 the current is on, if a 0 is written to Port A, Pin 2 the current is off. To save power, the current should be off if the alarm is off.

1.4 Controlling the LED's:

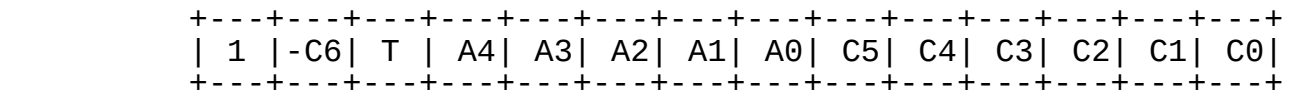
If a 1 is written to Port A, Pin 3(4), the red (green) LED is on, if a 0 is written, the LED is off.

1.5 Reading the distance sensor:

The distance sensor generates an analog voltage on Port A, Pin 7 which is proportional to the distance of an object from the sensor. The values are in the range 0.4 V (for 30 cm) and 3 V (for 4 cm)

1.6 Reading the IR diode:

The IR diode outputs a 0 if it receives a 36 kHz infrared input signal, otherwise a 1. A remote control using RC5 mode transmits for every key press 14 data bits:



The transfer time for a data bit is 2 * 889 µs. For a "1" bit the IR signal is off for the first 889 µs and on for the second 889 µs. For a "0" bit it is converse. This means, if a "1" bit is

transferred, you get on Port A Pin 6 for 889 µs 5V and for the next 889 µs 0V, for a "0" bit it is converse.

A4-A0:	Device Address (TV=0, SAT=8, ...)
C6-C0:	Command (e.g. 12=off, 16=volume +, ...) Originally there were only 64 codes (C5-C0), later the second start bit was used as negated C6 to extend the number of codes to 128.
T	Toggle Bit: For any key press of the same key this bit is toggled. This allows to distinguish between a repeatedly key press and a long key press (auto repeat).

2. Using the alarm clock

2.1 Display

Normally the clock displays the current time (hour + minute, no seconds). In the range 0:00-11:59 the green LED (AM) is on. If there is an object in front of the distance sensor, the alarm time is displayed instead of the current time.

2.2 Setting the time

A new time is set with the remote control by entering the time with the keys 0-9 and then pressing the Blue Key. If the Blue Key is not pressed within about 10-20 seconds after the last entered digit, the clock returns to normal display mode without changing the time.

2.3 Setting the alarm time

The alarm time is set with the remote control by entering the time with the keys 0-9 and then pressing the Red Key. If the Red Key is not pressed within about 10-20 seconds after the last entered digit, the clock returns to normal display mode without changing the alarm time.

2.4 Activating the alarm function

The alarm function is activated/deactivated by pressing the red OFF key on the remote control. If the alarm function is activated, the red LED is on. The alarm signal is on, if the current time is equal the set alarm time and the alarm function is activated (i.e. for no longer than one minute). The alarm signal is witched off by pressing the MUTE key on the remote control or waving the hand in front of the distance sensor.

3. Software

The included program provides the following functions:

3.1 Initializing routine

Initializes the stack, Port A, V24, remote control receiver, clock, A/D converter, servos, Timer0 and Timer2.

3.2 Interrupt routine for Timer 2

This code is able to control up to 8 servos (in this application only the servos 0 and 1 are used). The main program only has to write values in the range -50 to +50 (equivalent to an angel of -90° and +90°) into the variables *servo0* and *servo1*, the rest is done by the interrupt routine. If it is necessary to adjust the zero point, then this can be done in the above mentioned initializing routine.

3.3 Interrupt routine for Timer 0

This interrupt routine is executed every 64 µs and is responsible for :

- Increment the 24 bit counter *ticks*
- Generate the alarm signal
- Interpret the IR signal from the remote control

The code for the remote control is completely implemented and writes the key code into the key buffer. The repeat function of the remote control is disabled (i.e. even if a key is pressed for a longer time, only one key code is inserted into the buffer)

3.4 Reading the key buffer

The main program uses the following two (already implemented) functions to read the key buffer:

test_key: Z=0 if a key code is available in the buffer
Z=1 if no key code is available in the buffer
No registers are modified.

get_key: reads the next key code from the buffer and returns it to the calling program in register r16. No other register are modified. If there is no key code in the buffer, this function waits until a key on the remote control is pressed.

3.5 Reading the distance sensor

get_abst: uses the AD converter to convert the analog voltage from the distance sensor into a value between 0-255. This value is returned to the calling program in register r16. No other register are modified.

3.6 Main program

The main program uses the following variables:

ticks: 3 byte, is incremented in the interrupt routine every 64 µs.
time_h: actual time: hour (0-23)
time_m: actual time: minute (0-59)
weck_h: alarm time: hour (0-23)
weck_m: alarm time: minute (0-59*)*
weck_on:
- 0: alarm disabled
- 1: alarm enabled
- -1: alarm enabled, but alarm signal was already switched off by the remote control or distance sensor

speak:
- 0 alarm signal off
- >0 alarm signal on

4. Exercises

Exercise 1:

To support at least simple debugging, the V24 port of the Atmel processor is used to display text on the PC monitor. After programming the ATMega32, move the V24 cabel from the RS232 CTRL to the RS232 SPARE connector on the STK500 board and start a terminal program on the PC (9600 baud, 8 data bit, no parity bit, 1 stop bit). To display characters, the (already implemented) subroutine *out_r16* can be used. This subroutine sends the byte in r16 to the V24 port (no register are modified by the subroutine).

a) Implement a minimal main program (starting at the label *main:*), which continuously writes the characters "abababab..." to the PC screen.

b) Implement a subroutine *out_hex_r16* which displays the content of r16 as a hex value on the PC screen. Use *out_r16* for the actual output.

c) Write a main program which uses *get_key* to read input from the remote control and display the key code in hex form on the PC screen. Generate a code table for all keys on the remote control.

d) Write a main program which uses *get_abst* to read the distance sensor and display the value in hex form on the PC screen. Select a value for a reasonable distance to switch off the alarm.

Exercise 2

Write a main program which uses two keys of the remote control to switch on/off the green LED.

Exercise 3

a) Write a main program, which first initializes the variables *time_h*, *time_m*, *weck_h*, *weck_m*, *weck_on* and *speak* with 0. Then use two keys of the remote control to increment/decrement the variable *time_h* (within the limit 0-23).

b) Write a subroutine *display*, which displays the content of *time_h* on the hour hand (servo0). If the value is in the range 0-11, the green LED should be on (AM), otherwise *time_h*-12 (PM) should be displayed on the hour hand. To do this, read the corresponding value from the table *tab_h* and write it into the variable *servo0*. If you use a different type of servos, you maybe have to do some minor corrections to values in the table. Call this subroutine after each modification of *time_h* by the remote control

Exercise 4

a) Extend the main program of Exercise 3, so that also the variable *time_m* can be increase/decreased by two different keys on the remote control. If there is an overflow (<0 or >59), the variable *time_h* also has to be modified.

b) Extend the subroutine *display* from Exercise 3, so that the minutes are displayed with the second servo (servo1). Extract the corresponding value for the servo position from the table *tab_m*.

c) Modify the code to display the hour, so that the hour hand is positioned between two hour markers, dependent of the minute value (by interpolation).

Exercise 5

Replace the code for incrementing /decrementing the hour/minute by a code which allows to directly enter the time using the 0-9 keys on the remote control. If the Blue Key is pressed, the entered time should be stored as the current time (*time_h, time_m*) and if the Red Key is pressed, the entered time should be stored as the alarm time (*weck_h, weck_m*).

Exercise 6

a) Extend the interrupt routine for timer 0, so it increments the 3 byte variable *ticks* with every interrupt. If a value is reached which corresponds to 1 minute, the variable has to be reset and the variable *time_m* (and maybe *time_h*) adopted.

b) The main program now should not only wait for input from the remote control but also continuously call the subroutine *display*, so the actual time (which is modified by the interrupt routine) is displayed (therefore use *tst_key* to test if a key code is available before calling *get_key* to read the key code).

Exercise 7

Extend the main program, so that the alarm time is displayed if something is held in front of the distance sensor (also here: green LED on for AM).

Exercise 8

a) Modify the main program, so that the variable *weck_on* changes from 0 to 1 or from <>0 to 0 if the OFF key is pressed on the remote control.

b) Modify the subroutine *display*, so that the red LED is on if the variable *weck_on* is <>0.

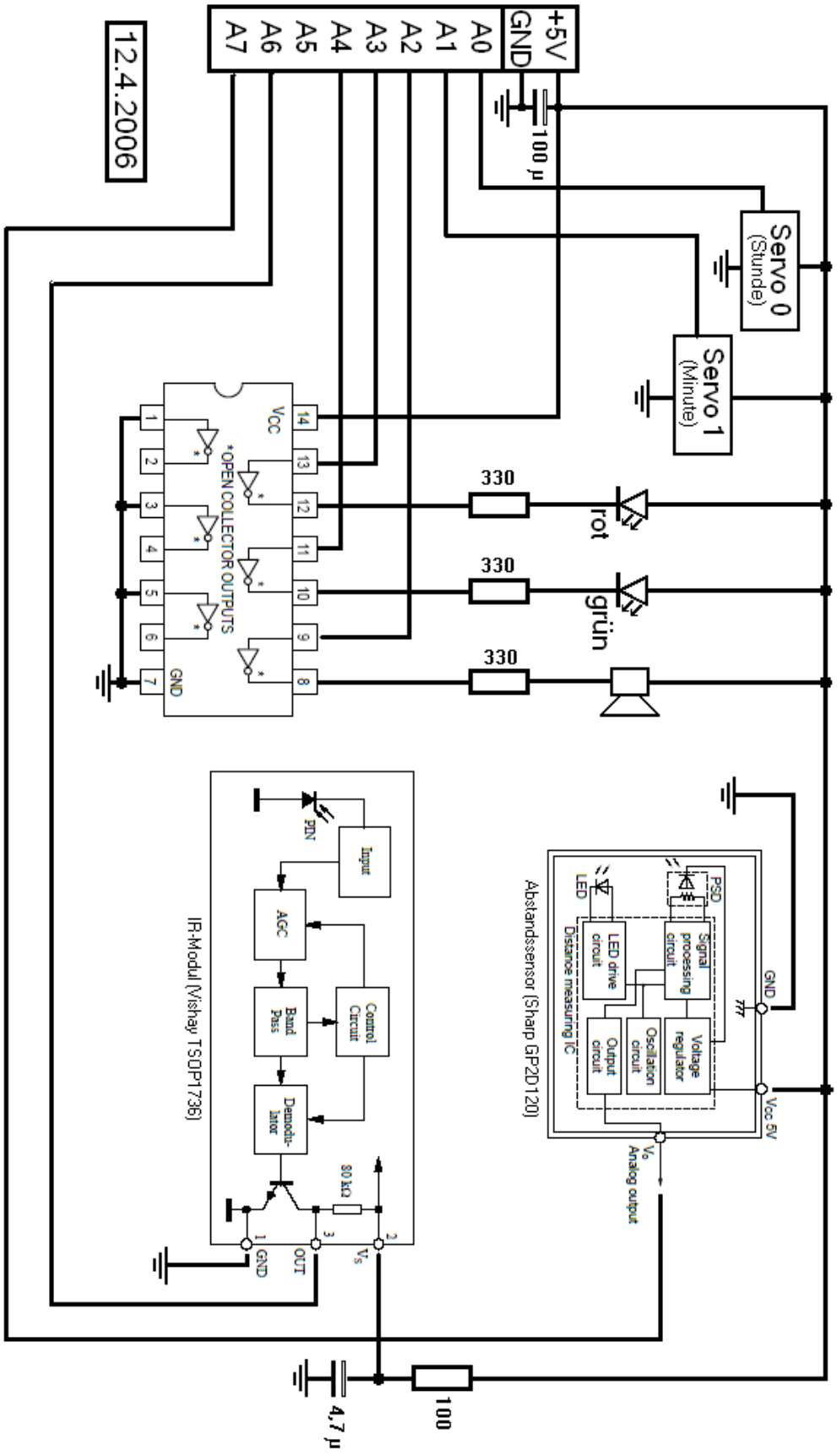
Exercise 9

a) Modify the main program, so that the variable *speak* changes from 0 to 1 or from <>0 to 0 if the MUTE key is pressed on the remote control.

b) Modify the interrupt routine for timer 0, so that an alarm signal is generated by the speaker if the variable *speak* is not 0.

c) Now the variable *speak* should no longer be set by the MUTE key on the remote control, but then, when the actual time is equal to the set alarm time and *weck_on* not equal 0 (this should be done in the main program and not in the interrupt routine). The variable *speak* should be reset when either the time is no longer equal to the set alarm time (i.e. after maximal one minute), the MUTE key is pressed on the remote control or something is held in front of the distance sensor.

5. Schematic


12.4.2006
+5V
GND
A0
A1
A2
A3
A4
A5
A6
A7
100 µ
Servo 0
(Stunde)
Servo 1
(Minute)
330
330
330
rot
grün
Vcc
GND
*OPEN COLLECTOR OUTPUTS
IR-Modul (Vishay TSOP1736)
PIN
Input
AGC
Band Pass
Demodu-lator
Control Circuit
80 kΩ
OUT
Vs
Abstandssensor (Sharp GP2D120)
PSD
LED
LED drive circuit
Signal processing circuit
Output circuit
Oscillation circuit
Voltage regulator
Distance measuring IC
Vcc 5V
Vo
Analog output
4,7 µ
100

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) gruene 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_timer1event
        jmp.l   reset           ; irq07_timer1cmpA
        jmp.l   reset           ; irq08_timer1cmpB
        jmp.l   reset           ; irq09_timer1ovf
        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,servo1+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  #%11010110,r16    ; adc enable, start conv., no auto trig.,
        move.b  r16,?adc_ctr      ; clear IF, int disable, prescale 64
_10:    skipeq.bit #1,?adc_ctr[4] ; conversion ready
        br.w    _10               ; no
        move.b  ?adc_hi,r16
        rts.w
; ***********************************************************


IF debug
; ***************** Ausgabe V24 ****************************
out_r16:
   _10: skipeq.bit #1,?v24_ctr1[5]      ; data register empty
        br.w    _10
        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    _1sig           ; fb_mode=0

;---+________/////|/////--\\\\\\|\\\\\\
;   33332222222222211111111110000000000
;   43210987654321098765432109876543210

_nosig: move.b  fb_count,r16
        skipeq.bit #1,?port_a_pin[6]
        br.w    _10             ; 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    _end1

   _10: 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    _end1

_0sig:  move.b  r17,(sp)-
        move.b  r30,(sp)-
        move.b  r31,(sp)-
        move.b  fb_count,r16
        skipeq.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 y 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


_1sig:  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 UBRRL
v24_ctr2:       blk.b   1       ; 0a UCSRB
v24_ctr1:       blk.b   1       ; 0b UCSRA
v24_dat:        blk.b   1       ; 0c UDR
spcr:           blk.b   1       ; 0d
spsr:           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) gruene LED (1:an)
; a6: (in)  IR Diode
; a7: (in)  Abstandssensor

.ORG 0
                                ; Interrupt Tabelle
        jmp     irq00_reset
        jmp     reset           ; irq01_ext0
        jmp     reset           ; irq02_ext1
        jmp     reset           ; irq03_ext2
        jmp     irq04_timer2cmp
        jmp     irq05_timer2ovf
        jmp     reset           ; irq06_timer1event
        jmp     reset           ; irq07_timer1cmpA
        jmp     reset           ; irq08_timer1cmpB
        jmp     reset           ; irq09_timer1ovf
        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
;    ENDIF
; ***********************************************************

; ***************** 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 servo1
        sts     servo1+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:
   d10: sbis    v24_ctr1,5        ; 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    _1sig           ; fb_mode=0

;---+________/////|/////--\\\\\\|\\\\\\
;   3333222222222221111111110000000000
;   4321098765432109876543210987654321

_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    _end1

   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    _end1

_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


_1sig:  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
;                 0   1   2   3   4   5   6   7   8   9   10  11  12
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 UBRRL
v24_ctr2:       .BYTE   1       ; 0a UCSRB
v24_ctr1:       .BYTE   1       ; 0b UCSRA
v24_dat:        .BYTE   1       ; 0c UDR
spcr:           .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
eearh:          .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
```