AVR Applications

Stopwatch with ATmega8 in Assembler

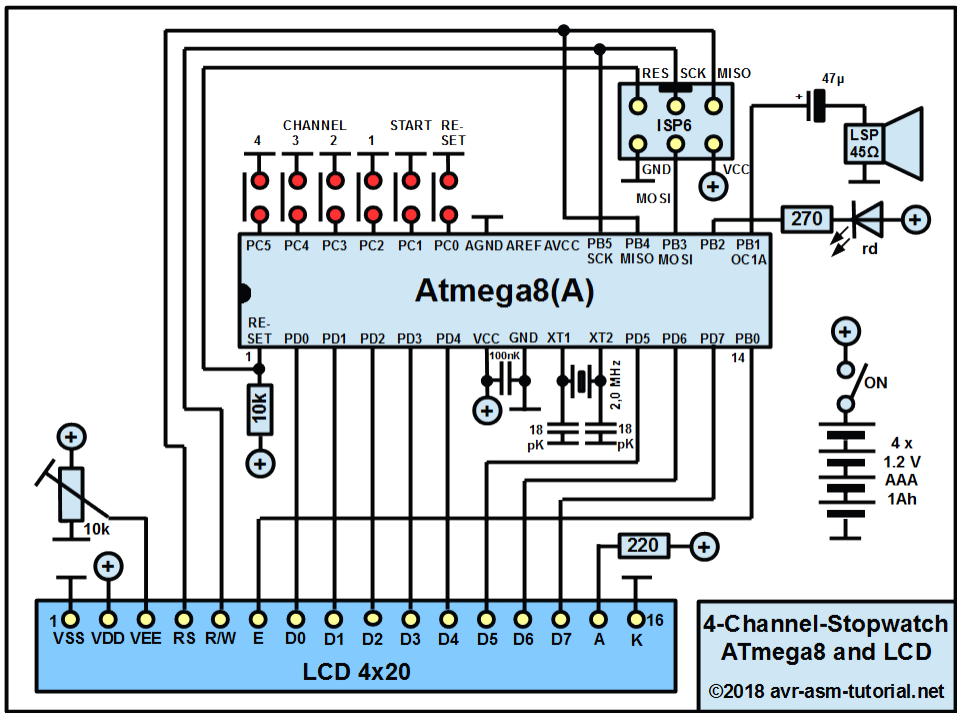
Stopwatch with an ATmega8

Described here is a stopwatch with the following properties:

- Four channels can be measured independently
- Time resolution at 1 ms
- Xtal clock with 50 ppm accuracy
- Start- and Stop control, inactive pauses possible
- Four line LCD for displaying four channels
- Specific tones for all events, select able by software
- Simple mounting
- Assembler software with source code
- Software compatible with 2.0 or 2.048 or 8.0 MHz Xtals

1 Hardware

This is the complete hardware.



The ATmega8 is clocked with a 2.048 MHz xtal. With small changes of the software, xtals with 2.0 or 8.0 MHz can also be used.

Power supply is with four 1.2 V recharge-able batteries so that standard LCDs with 5 V operating voltage can be used.

The four-by-twenty LCD is connected to an 8 bit bus and three control pins at the AT-mega8 (including Read/Write). By default the LCD is controlled in busy flag mode, but switching to wait mode is possible with a slight change of the source code.

One key clears the stopwatch, one key starts and stops the watch and four keys (or low-active sensors) stop the four channels.

The ISP interface allows programming of the controller within the running system.

2 Software structure

2.1 Display scheme

The display organization is shown here. It shows the display state

1. after reset,
2. during running time measurement, and
3. after having stopped two of the four channels.

Output format

Reset, not running

T	r	a	c	k	1			0	0	:	0	0	:	0	0	,	0		
T	r	a	c	k	2			0	0	:	0	0	:	0	0	,	0		
T	r	a	c	k	3			0	0	:	0	0	:	0	0	,	0		
T	r	a	c	k	4			0	0	:	0	0	:	0	0	,	0		

Running, no channel stopped

T	r	a	c	k	1			0	1	:	2	3	:	4	5	,	7		
T	r	a	c	k	2			0	1	:	2	3	:	4	5	,	7		
T	r	a	c	k	3			0	1	:	2	3	:	4	5	,	7		
T	r	a	c	k	4			0	1	:	2	3	:	4	5	,	7		

Running, channel 1 and 3 stopped

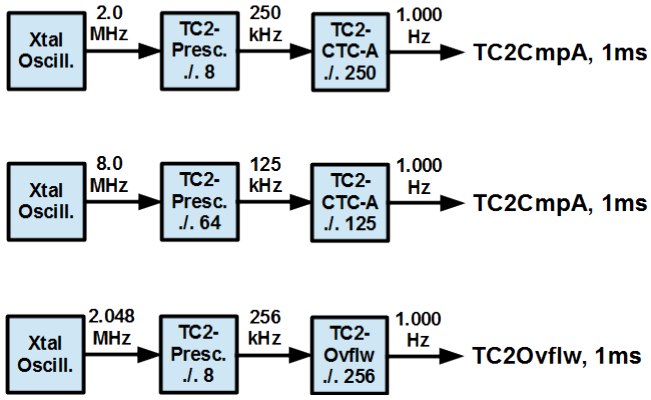
ZH																			
0	T	r	a	c	k	1	=	0	1	:	2	3	:	4	5	,	6	7	8
1	T	r	a	c	k	2		1	2	:	3	4	:	5	6	,	7		
2	T	r	a	c	k	3	=	0	1	:	2	3	:	4	5	,	6	7	8
3	T	r	a	c	k	4		1	2	:	3	4	:	5	6	,	7		
ZL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

2.2 Clock options

The millisecond clock is derived from the Xtal frequency as follows. When using a 2.0 or 8.0 MHz Xtal the 8 bit timer TC2 generates the ms clock via CTC mode and Compare Match A interrupt. If the default 2.048 MHz Xtal is used, this timer runs in free running mode and generates the 1 ms with the overflow interrupt.

At 2.0 and 2.048 MHz the 8 bit timer TC2 works with a prescaler of 8, if a 8.0 MHz Xtal is used the prescaler is at 64.

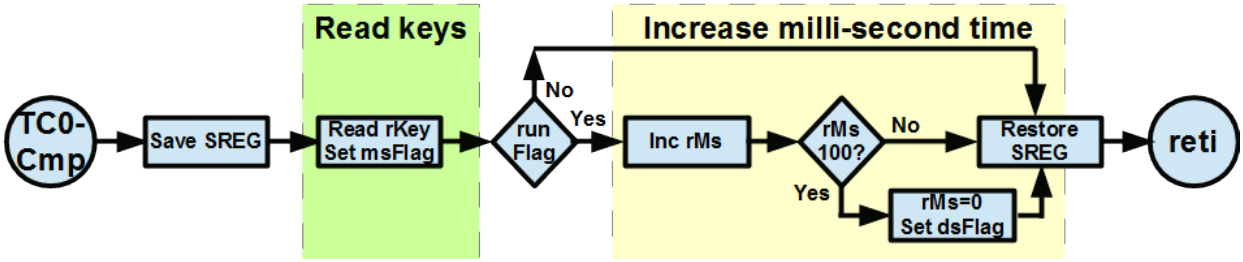
Milliseconds-timer



2.3 Interrupt Service Routine TC2

The Interrupt Service Routine of TC2, either triggered by the TC2OVF or by the TC2CMP vector, first reads in the current state of the keys. Handling of the keys is done outside the ISR, triggered by the bmS flag.

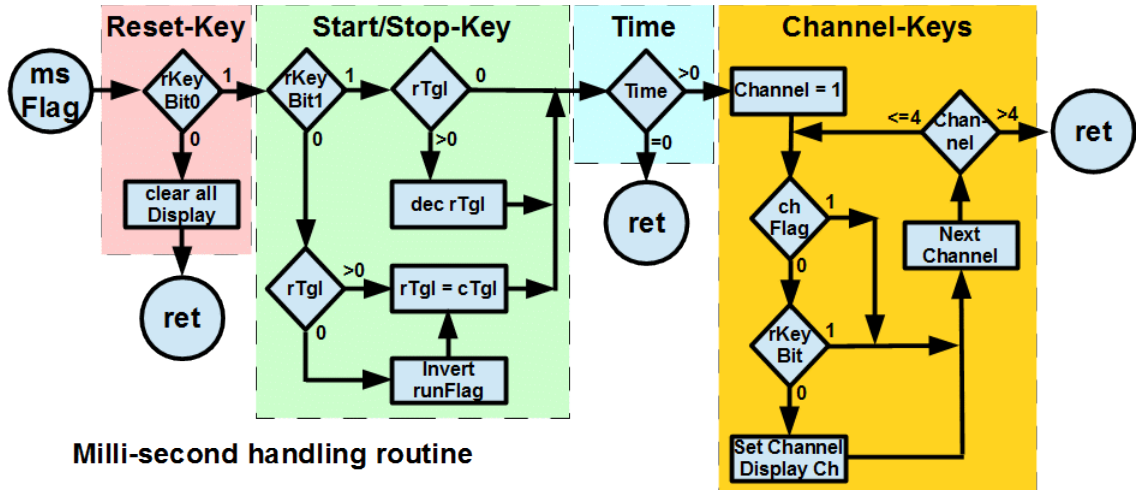
Milli-second timer interrupt



Then the bRun flag is checked if the time measurement is on. If yes the millisecond counter is increased. If that counter reaches 100, it is resetted and the bdS flag is set. Further time counting is done outside the ISR.

2.4 Handling of the millisecond flag

With the millisecond flag set, all keys are checked and, if active, the different actions are taken.



Milli-second handling routine

Firstly, the ms flag is cleared to enable her re-setting within the ISR's next cycle.

If the Reset key is pressed (input pin is low), the following happens:

1. the bRun flag is cleared, by that stopping the clock execution,
2. the time registers are cleared,
3. all channel flags are cleared,
4. all LCD lines display zero, and.
5. further key processing is skipped.

The Start/Stop key requires debouncing to avoid short pulses of the key. To do that a de-bouncer register called rTgl is set to a predefined number of counts. In case of an inactive Start/Stop key this register is counted down each millisecond. Only if that register is zero, an active pulse is accepted and the bRun flag is reversed. After each active pulse at the key input, be it successful or not, restarts this counter with the constant cTgl. cTgl should be large enough to cover usual bouncing times of keys (several tens of microseconds).

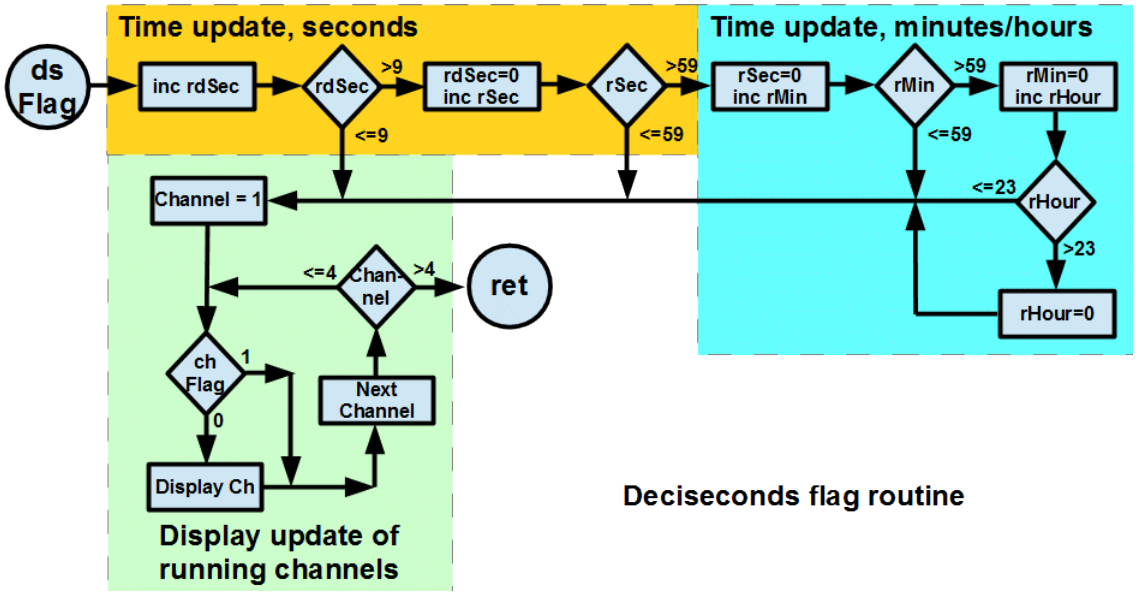
Following Start/Stop key processing the four channel inputs are checked. First it is checked if the time is zero, for which it makes no sense to stop channels. If larger than zero, all channels are checked whether the respective key is pressed and the channel is still running (channel bit in rFlag = 0). If that is the case, the channel is

1. stopped (by setting its flag bit),
2. the current time is written to the SRAM storage place of this channel,
3. the channel number is stored in the stop row in the SRAM, and
4. this time is displayed at the LCD line of that channel.

If all four channels have been processed it is checked whether all four channels are stopped. If that is the case, the stop row list in SRAM is used to display a sorted list of channels with increasing times.

2.5 Deciseconds handling

The bdS flag signals that 100 microseconds are over and that the deciseconds (and, if necessary, the rest of the clock registers) need an update.



The flag is cleared and the time is increased. In all channels that not yet have been stopped (channel flag = 0), the current time is displayed (hours, minutes, seconds and deciseconds, but not the milliseconds).

2.6 Tone generation

The tone generation is performed with the 16 bit timer TC1 in CTC mode and clocked with a prescaler of 8. At 2.0 MHz clock tones between 3.8 Hz and 250 kHz can be played, at 8.0 MHz clock between 15.3 Hz and 1 MHz are possible.

Tone output is on output pin OC1A, which is toggled on compare match if tones are on and cleared if tones are off. This unloads the capacitor on the OC1A output so that no current can flow into the pin when the operating voltage drops.

On each compare match an interrupt is triggered. The ISR counts down the 16 bit counter in R25:R24. If the counter reaches zero, the OC1A mode is changed to clear, switching tone output off in the next count cycle. Therefore R25:R24 controls the duration of the tones.

The tone frequencies and the duration of the tones is defined in the constants cTonexxx (in Hz) and cTonexxxDur (in ms). From these constants the Compare values cCmpXXX for the TC1-CTC and the values cCtrXXX for the tone duration counter are calculated and listed in a table called *ToneTable:* (duration word first, compare value word second).

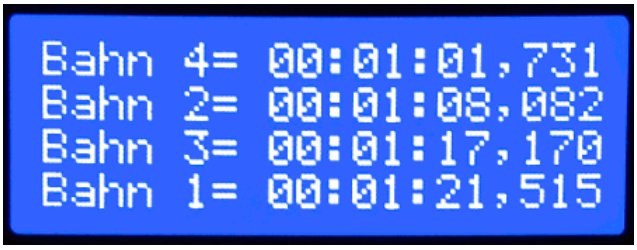
The routine *ToneStart:* starts the tone, for which its number (0..10) is in register rmp. First the counter value is read and set, then the compare value.

For the tones different octaves have been selected. During power-up and for the keys Reset and Start/Stop the fourth octave was selected, seconds/minutes/hours are one octave lower and the four channel keys are by two octaves lower. Other combinations and tones can be selected by changing the constants in the source code.

2.7 LED control

LED control is done with the portpin PB2. During init the LED is permanently on. When time measurement is active the LED is blinking in tenth of seconds for the four active channels (counting channels are on), followed by an off-period of four tenth of seconds. When all four channels are stopped the LED is permanently off.

2.8 Sorting by minimum times



If all four channels are stopped, the four times are displayed in a sorted list by minimum time. The row, in which the four channels were stopped, is stored in SRAM. The four times are either stored there.

If during assembling the switch *eep* was set to 1 all twenty data bytes for times and the four row bytes are written to EEPROM. The EEPROM content can be read.

3 Software

The assembler source code for the stopwatch with an ATmega8 can be downloaded from here and viewed in the attachment here For assembling the LCD include file lcd.inc is required which provides all LCD routines.

The software is by default configured as follows:

- Xtal frequency: 2.048 MHz, clock = 2048000
- LCD 4x20 8 bit data bus in busy mode
- EEPROM write off

When programming the chip its fuses have to be changed to a medium-speed crystal (2.0 or 2.048 MHz) or a high-speed crystal (8.0 MHz), otherwise the measured times and the tones played are not correct.

Praise, error reports, scolding and spam please via the comment page to me.

AVR Applications

Stopwatch with ATmega8
Assembler source code

Assembler source code for stopwatch ATmega8

This assembler source code can be downloaded in asm format here. Assembling requires the LCD include routines here.

```
;
; ************************************
; * Four channel stopwatch with ATmega8 *
; * Version 1.0 August 2018            *
; * (C)2018 avr-asm-tutorial.net       *
; ************************************
;
.nolist
.include "m8def.inc" ; Define device ATmega8
.list
;
; ****************************
;  D E B U G G I N G   C O D E
; ****************************
;
; All debugging code off (0) in final
.equ Eep = 0 ; 1 writes completed data to EEPROM
;
; ********************************
;        H A R D W A R E
; ********************************
;
; Device: ATmega8, Package: 28-pin-PDIP
;                 _________
;          1 /          |28
;  RESET o--|RES     PC5|--o Button Ch 4
; LCD-D0 o--|PD0     PC4|--o Button Ch 3
; LCD-D1 o--|PD1     PC3|--o Button Ch 2
; LCD-D2 o--|PD2     PC2|--o Button Ch 1
; LCD-D3 o--|PD3     PC1|--o Button Start/Stop
; LCD-D4 o--|PD4     PC0|--o Button Reset
;    VCC o--|VCC     GND|--o GND
;    GND o--|GND    AREF|--o NC
;   XTAL o--|XTAL1 AVCC|--o NC
;  8 MHz o--|XTAL2  PB5|--o SCK/LCD-RW
; LCD-D5 o--|PD5     PB4|--o MISO/LCD-RS
; LCD-D6 o--|PD6     PB3|--o MOSI
; LCD-D7 o--|PD7     PB2|--o LED-RD Cathode
;  LCD-E o--|PB0     PB1|--o LSP
;           |___________|
```

```
;

; *********************************
;  P O R T S   A N D   P I N S
; *********************************
;
; Red Led
.equ pLedRO = PORTB ; Red Led port output
.equ pLedRD = DDRB ; Red Led port direction
.equ bLedRO = PORTB2 ; Red Led output portpin
.equ bLedRD = DDB2 ; Red Led direction portpin
;
; Speaker
.equ pSpkO = PORTB ; Speaker port output
.equ pSpkD = DDRB ; Speaker port direction
.equ bSpkO = PORTB1 ; Speaker output portpin
.equ bSpkD = DDB1 ; Speaker direction portpin
;
; Buttons
.equ pKeyO = PORTC ; Button output port
.equ pKeyD = DDRC ; Button direction port
.equ pKeyI = PINC ; Button input port
;
; Clocking configuration
;   Clocks can be: 2.000 or 8.000 MHz or 2.048 MHz
;   If set to 2.048: TC2 normal counting, overflow int
;   If set to 2/8 MHz: TC2 as CTC, compare match int
.equ clock = 2048000 ; Clock frequency controller in Hz
;
; LCD configuration
; LCD size:
  .equ LcdLines = 4 ; Number of lines (1, 2, 4)
  .equ LcdCols = 20 ; Number of characters per line (8, 16, 20, 24)
; LCD databus:
  .equ LcdBits = 8 ; Bus width (4 or 8)
  ; If 4 bit bus:
    ;.equ Lcd4High = 1 ; Bus nibble (1=Upper, 0=Lower)
  .equ LcdWait = 0 ; Operation mode (0 with busy, 1 with delay loops)
; LCD databus ports
  .equ pLcdDO = PORTD ; Data output port
  .equ pLcdDD = DDRD ; Data direction port
; LCD control ports and pins
  .equ pLcdCEO = PORTB ; Control E output port
  .equ bLcdCEO = PORTB0 ; Control E output portpin
  .equ pLcdCED = DDRB ; Control E direction port
  .equ bLcdCED = DDB0 ; Control E direction portpin
  .equ pLcdCRSO = PORTB ; Control RS output port
  .equ bLcdCRSO = PORTB4 ; Controll RS output portpin
  .equ pLcdCRSD = DDRB ; Control RS direction port
  .equ bLcdCRSD = DDB4 ; Control RS direction portpin
; If LcdWait = 0:
  .equ pLcdDI = PIND ; Data input port
  .equ pLcdCRWO = PORTB ; Control RW output port
  .equ bLcdCRWO = PORTB5 ; Control RW output portpin
  .equ pLcdCRWD = DDRB ; Control RW direction port
  .equ bLcdCRWD = DDB5 ; Control RW direction portpin
; Include decimal conversion routines
  .equ LcdDecimal = 1 ; Decimal on
; Include hexadecimal conversion routines
  ;.equ LcdHex = 1 ; Hexadecimal off
; If only simulation only in SRAM:
  ;.equ avr_sim = 1 ; 1=Simulate, 0=No simulation
;
; *************************************
```

```
;   A D J U S T A B L E   C O N S T A N T S
; ****************************************
;
; English or German version
.equ LangEn = 1 ; English = 1, German = 0
;
.equ cToggle=60 ; Toggle suppression time in ms
;
; Speaker tones, frequencies
;   Englisch tone names (in brackets: german tone names)
.equ cToneInit = 3136 ; Tone at start-up, G7 (g4), in Hz
.equ cToneSec = 2794 ; Tone at seconds, F7 (f4), in Hz
.equ cToneMin = 2637 ; Tone at minutes, E7 (e4), in Hz
.equ cToneHour = 2349 ; Tone at hours, D7 (d4), in Hz
.equ cToneReset = 1319 ; Tone at reset, E6 (e3), in Hz
.equ cToneStart = 1175 ; Tone at start, D6 (d3), in Hz
.equ cToneStop = 1047 ; Tone at stop, C6 (c3), in Hz
.equ cToneCh1 = 440 ; Tone at channel 1, A4 (a1), in Hz
.equ cToneCh2 = 392 ; Tone at channel 2, G4 (g1), in Hz
.equ cToneCh3 = 349 ; Tone at channel 3, F4 (f1), in Hz
.equ cToneCh4 = 330 ; Tone at channel 4, E4 (e1), in Hz
;
; Speaker tones, durations
.equ cToneSecDur = 100 ; Tone duration at seconds, in ms
.equ cToneMinDur = 500 ; Tone duration at minutes, in ms
.equ cToneHourDur = 750 ; Tone duration at hours, in ms
.equ cToneResetDur = 500 ; Tone duration at reset, in ms
.equ cToneStartDur = 400 ; Tone duration at start, in ms
.equ cToneStopDur = 300 ; Tone duration at stop, in ms
.equ cToneCh1Dur = 200 ; Tone duration at channel 1, in ms
.equ cToneCh2Dur = 200 ; Tone duration at channel 2, in ms
.equ cToneCh3Dur = 200 ; Tone duration at channel 3, in ms
.equ cToneCh4Dur = 200 ; Tone duration at channel 4, in ms
;
; ****************************************
;  F I X E D  &  D E R I V E D  C O N S T
; ****************************************
;
.if clock == 8000000
  .equ cPresc2 = 64 ; Prescaler TC2 = 64
  .else
  .equ cPresc2 = 8 ; Prescaler TC2 = 8
  .endif
;
.if clock != 2048000
  .equ cCtcDiv2 = clock / cPresc2 / 1000 ; CTC Divider
  .equ cCmp2A = cCtcDiv2 - 1 ; Compare value TC2
  .endif
;
; Speaker tones to compare values
.equ cPresc1 = 8 ; Prescaler TC1
.equ cCmpInit = clock / cToneInit / cPresc1 / 2 - 1 ; Compare Init
.equ cCmpSec = clock / cToneSec / cPresc1 / 2 - 1 ; Compare second
.equ cCmpMin = clock / cToneMin / cPresc1 / 2 - 1 ; Compare minute
.equ cCmpHour = clock / cToneHour / cPresc1 / 2 - 1 ; Compare hour
.equ cCmpReset = clock / cToneReset / cPresc1 / 2 - 1 ; Compare Reset
.equ cCmpStart = clock / cToneStart / cPresc1 / 2 - 1 ; Compare Start
.equ cCmpStop = clock / cToneStop / cPresc1 / 2 - 1 ; Compare Stop
.equ cCmpCh1 = clock / cToneCh1 / cPresc1 / 2 - 1 ; Compare channel 1
.equ cCmpCh2 = clock / cToneCh2 / cPresc1 / 2 - 1 ; Compare channel 2
.equ cCmpCh3 = clock / cToneCh3 / cPresc1 / 2 - 1 ; Compare channel 3
.equ cCmpCh4 = clock / cToneCh4 / cPresc1 / 2 - 1 ; Compare channel 4
;
; Speaker tone durations to counter values
```

```
.equ cCtrInit = 1 ; Count until first interrupt execution
.equ cCtrSec = cToneSec * 2 * cToneSecDur / 1000 ; Counter second tone
.equ cCtrMin = cToneMin * 2 * cToneMinDur / 1000 ; Counter minute tone
.equ cCtrHour = cToneHour * 2 * cToneHourDur / 1000 ; Counter hour tone
.equ cCtrReset = cToneReset * 2 * cToneResetDur / 1000 ; Counter reset tone
.equ cCtrStart = cToneStart * 2 * cToneStartDur / 1000 ; Counter start tone
.equ cCtrStop = cToneStop * 2 * cToneStopDur / 1000 ; Counter stop tone
.equ cCtrCh1 = cToneCh1 * 2 * cToneCh1Dur / 1000 ; Counter channel 1 tone
.equ cCtrCh2 = cToneCh2 * 2 * cToneCh2Dur / 1000 ; Counter channel 2 tone
.equ cCtrCh3 = cToneCh3 * 2 * cToneCh3Dur / 1000 ; Counter channel 3 tone
.equ cCtrCh4 = cToneCh4 * 2 * cToneCh4Dur / 1000 ; Counter channel 4 tone
;
; *********************************
;       R E G I S T E R S
; *********************************
;
; free: R0 to R8
.def rmsBuf = R9 ; Buffer for milliseconds
.def rLedCtr = R10 ; LED counter register
.def rLed = R11 ; LED display register
.def rTgl = R12 ; Toggle counter
.def rKeyCmp = R13 ; Compare value buttons
.def rKey = R14 ; Key input port
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose inside ints
.def rFlag = R18 ; Flag register
  .equ bMs = 0 ; Millisecond flag
  .equ bRun = 1 ; Run/Stop flag
  .equ bCh1 = 2 ; Channel 1 flag
  .equ bCh2 = 3 ; Channel 2 flag
  .equ bCh3 = 4 ; Channel 3 flag
  .equ bCh4 = 5 ; Channel 4 flag
  .equ bdS = 6 ; Decisecond flag
  .equ bSort = 7 ; Sorted channels flag
.def rHour = R19 ; Hours
.def rMin = R20 ; Minutes
.def rSec = R21 ; Seconds
.def rdSec = R22 ; deziseconds
.def rmSec = R23 ; Milliseconds
.def rCtrL = R24 ; Counter for tone duration, LSB
.def rCtrH = R25 ; dto., MSB
; Used: XH:XL=R27:R26 for pointing to sData
; Used: YH:YL=R29:R28 for channel row addressing
; Used: ZH:ZL=R31:R30 for diverse purposes
;
; *********************************
;           S R A M
; *********************************
;
.dseg
.org SRAM_START
sData:
.byte 20 ; Reserve 4*5 bytes for time info
sChannelRow:
.byte 4 ; The row in that the channels were stopped
sDataEnd:
;
; *********************************
;       C O D E  S E G M E N T
; *********************************
;
.cseg
.org 000000
```

```
;
; ************************************
;  R E S E T  &  I N T - V E C T O R S
; ************************************
  rjmp Main ; Reset vector
  reti ; INT0
  reti ; INT1
  .if clock == 2048000
    reti ; TC2 Compare
        rjmp MilliSecIsr ; OVF2
        .else
        rjmp MilliSecIsr
        reti
        .endif
  reti ; ICP1
  rjmp OC1CmpIsr ; OC1A
  reti ; OC1B
  reti ; OVF1
  reti ; OVF0
  reti ; SPI
  reti ; URXC
  reti ; UDRE
  reti ; UTXC
  reti ; ADCC
  reti ; ERDY
  reti ; ACI
  reti ; TWI
  reti ; SPMR
;
; ***************************************
;  I N T - S E R V I C E  R O U T I N E S
; ***************************************
;
; TC2 Compare Match Interrupt
;   executed once every millisecond
;   reads input keys and sets ms flag
;   if running increases ms, if ms
;     reaches 100: restarts and sets ds flag
MillisecIsr:
  in rSreg,SREG ; Save SREG
  in rKey,pKeyI ; Read input keys
  sbr rFlag,1<<bmS ; Set millisecond flag
  sbrs rFlag,bRun ; Run-Flag?
  rjmp MillisecIsr1 ; No, skip
  inc rmSec ; Increase millisecond counter
  cpi rmSec,100 ; 100 ms reached?
  brcs MilliSecIsr1 ; No
  clr rmSec ; Restart milliseconds
  sbr rFlag,1<<bdS ; Set dezisecond flag
MillisecIsr1:
  out SREG,rSreg ; Restore SREG
  reti
;
; TC1 Compare Match Interrupt
;   executed on compare match A when one
;   half cycle of the tone is over
;   Decreases the duration counter, if
;     zero switches tone off by setting
;     speaker output to off after the
;     next cycle
OC1CmpIsr:
  in rSreg,SREG ; Save SREG
  sbiw rCtrL,1 ; Count down
  brne OC1CmpIsr1 ; Not at zero
```

```
  ldi rimp,1<<COM1A1 ; Clear speaker pin
  out TCCR1A,rimp
OC1CmpIsr1:
  out SREG,rSreg ; Restore SREG
  reti
;
; *********************************
;  M A I N   P R O G R A M   I N I T
; *********************************
;
Main:
.ifdef SPH
  ldi rmp,High(RAMEND)
  out SPH,rmp ; Init MSB stack pointer
  .endif
  ldi rmp,Low(RAMEND)
  out SPL,rmp ; Init LSB stack pointer
  ; Init button inputs
  ldi rmp,0x3F ; All six output pins high
  out pKeyO,rmp ; Switch pull-ups on
  clr rmp ; Direction low
  out pKeyD,rmp
  ; Init red led pin
  sbi pLedRD,bLedRD ; Direction = output
  cbi pLedRO,bLedRO ; Output low
  ldi rmp,0b11000011 ; LED register init
  mov rLed,rmp ; LED register
  ldi rmp,8 ; LED counter register init
  mov rLedCtr,rmp
  ; Init speaker pin
  cbi pSpkO,bSpkO ; Output low
  sbi pSpkD,bSpkD ; Direction = output
  ; Turn speaker init tone on
  ldi rCtrH,High(cCtrInit) ; Load duration counter
  ldi rCtrL,Low(cCtrInit)
  ldi rmp,High(cCmpInit) ; Load tone compare value to init tone
  out OCR1AH,rmp
  ldi rmp,Low(cCmpInit)
  out OCR1AL,rmp
  ldi rmp,1<<COM1A0 ; Toggle OC1A output on compare match
  out TCCR1A,rmp
  ldi rmp,(1<<WGM12)|(1<<CS11) ; CTC, Prescaler = 8
  out TCCR1B,rmp
  ; Start the LCD
  rcall LcdInit
  ldi ZH,High(2*LcdInit1)
  ldi ZL,Low(2*LcdInit1)
  rcall LcdText
  ldi rmp,40 ; Wait two seconds
Start1:
  rcall LcdWait50ms ; Wait 50 ms
  dec rmp ; Count down
  brne Start1 ; Wait on
  clr ZH ; Line 1 of the LCD
  clr ZL ; Column 1 of the LCD
  rcall LcdPos ; Set LCD position
  ldi ZH,High(2*LcdInit2) ; Mask to LCD
  ldi ZL,Low(2*LcdInit2)
  rcall LcdText
  ; Init the sram data
  clr rmSec ; Init milliseconds
  clr rdSec ; Init deciseconds
  clr rSec ; Init seconds
  clr rMin ; Init minutes
```

```
  clr rHour ; Init hours
  clr rFlag ; Clear all flags
  rcall ClearData ; Clear all SRAM data
  rcall Display ; Display all channels
  ; Init TC2 as millisecond timer
  .if clock == 2048000
    ; Set up TC2 as normal counter with precaler 8 and ovf int
    ldi rmp,(1<<CS21) ; Prescaler = 8
    out TCCR2,rmp
    ldi rmp,(1<<OCIE1A)|(1<<TOIE2) ; TC1 Comp Match, TC2 Overflow
    out TIMSK,rmp
    .else
    ldi rmp,cCmp2A ; Set CTC compare value
    out OCR2,rmp ; in compare register port
    .if clock == 8000000
      ldi rmp,(1<<WGM21)|(1<<CS22) ; CTC, Prescaler=64
      .else
      ldi rmp,(1<<WGM21)|(1<<CS21) ; CTC, Prescaler=8
      .endif
    out TCCR2,rmp ; to TC2 control port
    ldi rmp,(1<<OCIE1A)|(1<<OCIE2) ; TC1 Comp Match, TC2 Comp Match
    out TIMSK,rmp
    .endif
  ; Sleep enable
  ldi rmp,1<<SE ; Sleep enable in idle mode
  out MCUCR,rmp
  ; Interrupt enable
  sei ; Enable interrupts
;
; *********************************
;    P R O G R A M   L O O P
; *********************************
;
Loop:
  sleep ; go to sleep
  nop ; Dummy for wakeup
  sbrc rFlag,bmS ; Millisecond flag
  rcall Millisecond
  sbrc rFlag,bdS ; Decisecond flag
  rcall Dezisecond
  rjmp loop
;
; A millisecond is over, check the buttons
Millisecond:
  cbr rFlag,1<<bmS ; Clear flag
  mov rmsBuf,rmSec ; Milliseconds to buffer
  ldi rmp,0b00000001 ; Start with button 1
  mov rKeyCmp,rmp ; Move to compare register
  and rmp,rKey ; Check bit set
  brne ChkStartStop ; Bit is high, go on compare
  ; Reset-Button pressed
  clr rFlag
  clr rmSec ; Clear time
  clr rdSec
  clr rSec
  clr rMin
  clr rHour
  rcall ClearData ; Clear SRAM buffer
  ldi rmp,4 ; Tone #4
  rcall ToneStart
  clr ZH
  clr ZL
  rcall LcdPos
  ldi ZH,High(2*LcdInit2)
```

```
  ldi ZL,Low(2*LcdInit2)
  rcall LcdText
  rjmp Display ; Display all channels
ChkStartStop:
  lsl rKeyCmp ; Next bit
  mov rmp,rKey ; Copy keys
  and rmp,rKeyCmp ; Key bit set?
  brne ChkToggle ; One, check toggle empty
  tst rTgl ; Check toggle register
  brne ChkStartStopRestartToggle ; Not zero, restart
  ldi rmp,1<<bRun ; Toggle bRun
  eor rFlag,rmp ; Toggle bRun flag
ChkStartStopRestartToggle:
  ldi rmp,cToggle ; Load toggle value
  mov rTgl,rmp ; and write to toggle register
  rjmp ChkChannels ; Test the channel buttons
ChkToggle:
  tst rTgl ; Toggle counter zero?
  breq ChkChannels ; Yes, check channel buttons
  dec rTgl ; Decrease toggle register
;
; Check all channels if button pressed
;   rKeyCmp points to previous button position
ChkChannels:
  mov rmp,rmSec ; Time zero?
  or rmp,rdSec
  or rmp,rSec
  or rmp,rMin
  or rmp,rHour
  breq ChkChannels4 ; Yes, skip storage
  clr ZH ; Start in line 1
  ldi XH,High(sData) ; Point to data buffer
  ldi XL,Low(sData)
ChkChannels1:
  lsl rKeyCmp ; Next key
  mov rmp,rKeyCmp ; Read compare bit
  and rmp,rKey ; Is this key pressed?
  brne ChkChannels2 ; Key is not pressed
  mov rmp,rKeyCmp ; Read compare bit
  and rmp,rFlag ; This channel stopped?
  brne ChkChannels2 ; Yes, check next channel
  ; Channel has key down and is not stopped
  st X+,rHour ; store time stamp
  st X+,rMin
  st X+,rSec
  st X+,rdSec
  st X+,rmsBuf ; Milliseconds from buffer!
  sbiw XL,5 ; To start of data buffer
  or rFlag,rKeyCmp ; Set channel flag
  st Y+,ZH ; Store channel in row list
  push ZH
  ldi ZL,6
  rcall LcdPos
  rcall DisplayX
  ldi rmp,7 ; Tone select
  pop ZH ; Get ZH from stack
  push ZH ; and back to stack
  add rmp,ZH ; Add line counter
  rcall ToneStart ; Play tone
  pop ZH
  rjmp ChkChannels3
ChkChannels2:
  ; Channel was not set, advance pointer
  adiw XL,5 ; Point to next data set
```

```
ChkChannels3:
  inc ZH ; Increase line counter
  cpi ZH,4 ; End of lines?
  brcs ChkChannels1 ; Check next channel
  ldi rmp,0b00111100 ; Check all channels stopped
  and rmp,rFlag ; Channels
  cpi rmp,0b00111100 ; All channels set?
  breq DisplaySorted ; Display channels sorted
ChkChannels4:
  ret
;
; Display sorted channel list
DisplaySorted:
  .if Eep == 1
    rcall Write2Eep ; Write data area to EEPROM
    .endif
  sbr rFlag,1<<bSort ; To sort mode
  cbr rFlag,1<<bRun ; Stop watch
  clr ZH ; Line counter, start with line 1
  ldi YH,High(sChannelRow)
  ldi YL,Low(sChannelRow)
DisplaySorted1:
  ld rmp,Y ; Read channel number from sorted list
  cpi rmp,0xFF ; Channel not stopped?
  brne DisplaySorted3 ; Channel is stopped
  ; Clear that line
  ldi ZL,0 ; Column 1
  rcall LcdPos ; Set LCD position
  ldi ZL,20 ; 20 blanks
DisplaySorted2:
  ldi rmp,' ' ; Write blank
  rcall LcdChar ; to LCD
  dec ZL ; Further blanks?
  brne DisplaySorted2 ; Yes, repeat
  rjmp DisplaySorted6 ; Next entry
DisplaySorted3:
  ldi XH,High(sData) ; X to data area
  ldi XL,Low(sData)
  tst rmp ; channel = 0?
  breq DisplaySorted5 ; Yes, first channel
DisplaySorted4:
  adiw XL,5 ; Next channel data
  dec rmp ; Decrease channel number
  brne DisplaySorted4 ; Not zero, continue
DisplaySorted5:
  ldi ZL,5 ; To column 6
  rcall LcdPos ; Set LCD position
  ld rmp,Y ; Read channel number again
  subi rmp,-'0'-1 ; Add ASCII-1
  rcall LcdChar ; Display on LCD
  push ZH ; Save line
  rcall DisplayX ; Display this time
  pop ZH ; Restore line
DisplaySorted6:
  adiw YL,1 ; Next sort entry
  inc ZH ; Next line
  cpi ZH,4 ; Lines done?
  brcs DisplaySorted1 ; No, repeat
  ret
;
DisplayX:
  ldi rmp,'=' ; Display equal char
  rcall LcdChar
  ldi rmp,' ' ; and a blank
```

```
  rcall LcdChar
  ld rmp,X+ ; Load hour
  rcall LcdDec2
  ldi rmp,':'
  rcall LcdChar
  ld rmp,X+ ; Load minute
  rcall LcdDec2
  ldi rmp,':'
  rcall LcdChar
  ld rmp,X+ ; Load second
  rcall LcdDec2
  .if LangEn == 1
    ldi rmp,'.' ; Decimal point
    .else
    ldi rmp,',' ; Decimal comma
    .endif
  rcall LcdChar
  ld rmp,X+ ; Load decisecond
  subi rmp,-'0'
  rcall LcdChar
  ld rmp,X+ ; Load millisecond
  rjmp LcdDec2
;
; Debugging, write results to EEPROM
Write2Eep:
  ldi XH,High(sData) ; X points to SRAM
  ldi XL,Low(sData)
  clr ZH ; Z is EEPROM address pointer
  clr ZL
Write2Eep1:
  out EEARH,ZH ; Set address
  out EEARL,ZL
  ld rmp,X+ ; Read next byte
  out EEDR,rmp ; To data port
  ldi rmp,1<<EEMWE ; Enable master write
  cli
  out EECR,rmp ; Start EEPROM master write
  ldi rmp,1<<EEWE ; Enable write
  out EECR,rmp
  sei ; Enable interrupts
Write2Eep2:
  sbic EECR,EEWE ; Check write completed
  rjmp Write2Eep2
  adiw ZL,1 ; Next EEPROM location
  cpi XL,Low(sDataEnd+4) ; End of data?
  brne Write2Eep1 ; No continue
  ret
;
; A dezisecond is over, increase time
Dezisecond:
  cbr rFlag,1<<bdS ; Clear flag
  inc rdSec ; Increase deziseconds
  cpi rdSec,10 ; Already at 10?
  brcs Dezisecond2 ; No, continue
  clr rdSec ; Restart deciseconds
  inc rSec ; Increase seconds
  cpi rSec,60 ; 60 seconds?
  ldi rmp,1 ; Tone #1
  brcs Dezisecond1 ; No, start tone
  inc rmp ; Next tone
  clr rSec ; Restart seconds
  inc rMin ; Increase seconds
  cpi rMin,60 ; 60 minutes?
  brcs Dezisecond1 ; no, start minute tone
```

```
  inc rmp ; Next tone
  clr rMin ; Restart minutes
  inc rHour ; Increase hours
  cpi rHour,24 ; 24 hours?
  brcs Dezisecond1 ; No
  clr rHour ; Restart hours
Dezisecond1:
  rcall ToneStart ; Play tone
DeziSecond2:
  sbrs rFlag,bSort ; Display in sorted mode?
  rcall Display ; No, display
; LED action
  cbi pLedRO,bLedRO ; LED on
  sbrs rFlag,bRun ; If not running
  ret ; Skip if stopped
  lsr rLed ; Shift lowest bit to carry
  brcc DeziSecond3 ; Zero, skip LED off
  sbi pLedRO,bLedRO ; LED off
DeziSecond3:
  dec rLedCtr ; Decrease LED counter
  brne DeziSecond4 ; Not at zero
  mov rmp,rFlag ; Copy flags
  ori rmp,0b11000011 ; Set all non-channel bits
  mov rLed,rmp ; And copy to LED register
  ldi rmp,8 ; Restart LED counter
  mov rLedCtr,rmp ; in counter register
DeziSecond4:
  ret
;
; Display time on all active channels
Display:
  ldi rmp,0b00000100 ; Compare value for channel 1
  mov rKeyCmp,rmp ; Copy to compare register
  ldi ZH,0 ; Start with channel 1 on line 1
  ldi ZL,8 ; Column 8
Display1:
  push ZH ; Save Z
  push ZL
  mov rmp,rFlag ; Copy Flags to rmp
  and rmp,rKeyCmp ; Is flag of channel set?
  brne Display2 ; Bit is set, do not display
  rcall LcdPos ; Set LCD cursor
  rcall DisplayLine ; Display time on this line
Display2:
  lsl rKeyCmp ; Next channel compare
  pop ZL ; Restore Z
  pop ZH
  inc ZH ; Next line
  cpi ZH,4 ; End of display lines?
  brcs Display1 ; No, repeat
  ret
;
; Displays one line of current time
;   LCD address as set
;   X points to time data
DisplayLine:
  mov rmp,rHour ; Read hours
  rcall LcdDec2
  ldi rmp,':'
  rcall LcdChar
  mov rmp,rMin ; Read minutes
  rcall LcdDec2
  ldi rmp,':'
  rcall LcdChar
```

```
  mov rmp,rSec ; Read seconds
  rcall LcdDec2
  .if LangEn == 1
    ldi rmp,'.' ; Decimal point
    .else
    ldi rmp,',' ; Decimal komma
    .endif
  rcall LcdChar
  mov rmp,rdSec ; Read deziseconds
  subi rmp,-'0' ; Add ASCII-0
  rjmp LcdChar
;
; Clear data in sram
ClearData:
  ldi ZH,High(sData) ; Start of data buffer
  ldi ZL,Low(sData)
  clr rmp ; Write zeros to buffer
ClearData1:
  st Z+,rmp ; Write byte to buffer
  cpi ZL,Low(sDataEnd) ; Buffer end?
  brne ClearData1 ; No, go on
  ldi YH,High(sChannelRow+4) ; Set row pointer
  ldi YL,Low(sChannelRow+4)
  ldi rmp,0xFF ; Fill channel rows
  st -Y,rmp
  st -Y,rmp
  st -Y,rmp
  st -Y,rmp
  ret
;
; Start a tone
;   Tone number in rmp
ToneStart:
  lsl rmp ; Multiply by 2
  lsl rmp ; Multiply by 4
  ldi ZH,High(2*ToneTable) ; Tone table to Z
  ldi ZL,Low(2*ToneTable)
  add ZL,rmp ; Add tone 4*number to LSB
  clr rmp ; Add carry to MSB
  adc ZH,rmp
  lpm rCtrL,Z+ ; Read LSB duration
  lpm rCtrH,Z+ ; Read MSB duration
  lpm rmp,Z+ ; Read LSB CTC value
  lpm ZH,Z ; Read MSB CTC value
  out OCR1AH,ZH ; Write MSB to compare A
  out OCR1AL,rmp ; Write LSB to compare A
  ldi rmp,1<<COM1A0 ; Toggle speaker output
  out TCCR1A,rmp ; Start tone
  ret
;
; Tone table
ToneTable:
.dw cCtrInit,cCmpInit ; Tone 0
.dw cCtrSec,cCmpSec ; Tone 1
.dw cCtrMin,cCmpMin ; Tone 2
.dw cCtrHour,cCmpHour ; Tone 3
.dw cCtrReset,cCmpReset ; Tone 4
.dw cCtrStart,cCmpStart ; Tone 5
.dw cCtrStop,cCmpStop ; Tone 6
.dw cCtrCh1,cCmpCh1 ; Tone 7
.dw cCtrCh2,cCmpCh2 ; Tone 8
.dw cCtrCh3,cCmpCh3 ; Tone 9
.dw cCtrCh4,cCmpCh4 ; Tone 10
;
```

```
; Include the Lcd routines
.include "lcd.inc"
;
; Lcd init text
.if LangEn == 1
  ; English init text
  LcdInit1:
    .db " Stop watch ATmega8",0x0D
    .db "  Four channels at",0x0D,0xFF
    .db "   one millisecond",0x0D,0xFF
    .db "  (C)2018 by DG4FAC",0xFE
  ;
  ; English display mask
  LcdInit2:
    .db "Track1  00:00:00.0  ",0x0D,0xFF
    .db "Track2  00:00:00.0  ",0x0D,0xFF
    .db "Track3  00:00:00.0  ",0x0D,0xFF
    .db "Track4  00:00:00.0  ",0xFE,0xFF
;        01234567890123456789
  .else
  ; German init text
  LcdInit1:
    .db "  Stoppuhr ATmega8",0x0D,0xFF
    .db "  Vier Kanaele mit",0x0D,0xFF
    .db " einer Millisekunde",0x0D
    .db "  (C)2018 by DG4FAC",0xFE
  ;
  ; German display mask
  LcdInit2:
    .db "Bahn 1  00:00:00.0  ",0x0D,0xFF
    .db "Bahn 2  00:00:00.0  ",0x0D,0xFF
    .db "Bahn 3  00:00:00.0  ",0x0D,0xFF
    .db "Bahn 4  00:00:00.0  ",0xFE,0xFF
;        01234567890123456789
  .endif
;
; Copyright info
.db "(C)2018 by avr-asm-tutorial.net " ; Normal text
.db "C(2)10 8yba rva-mst-turoai.len t" ; Wordwise
;
; End of source code
;
```