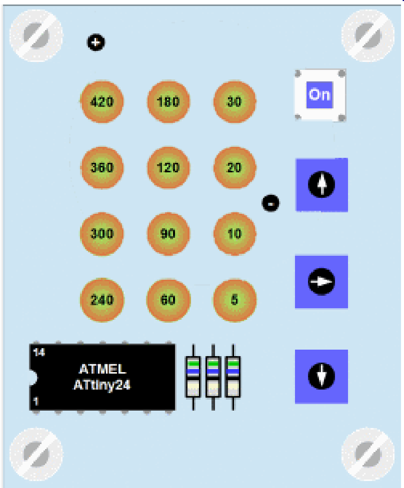


AVR applications

A multitimer with ATtiny24

Multitimer with ATtiny24 and 12 LEDs

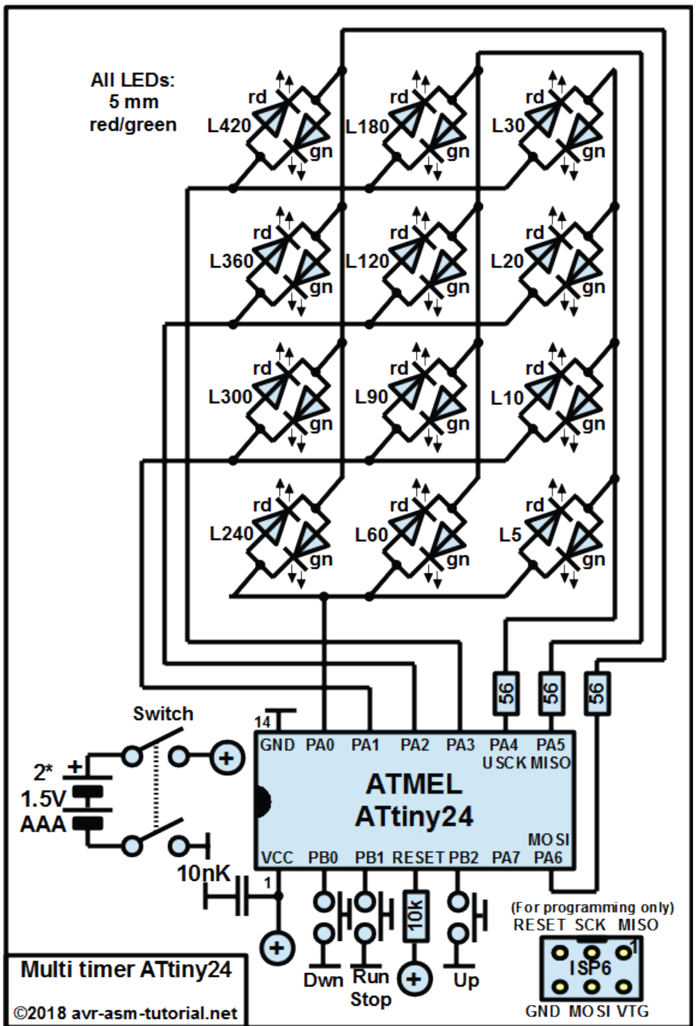
This describes a timer for between 5 seconds and seven minutes in 12 intervals. Selected time and run time is displayed with 12 red (running) or green (selection) LEDs. Selection is controlled by three buttons: up, down and run/stop.

1 Hardware

This is it:

- An ATtiny24 does the timing, reads the switches and does the LED control.
- 12 red/green duo LEDs do the display and are connected in a 4-by-3 matrix with port A of the controller.
- Three buttons are attached to port B of the controller and driven high by the internal pull up resistors.
- The whole device is operated with two AAA batteries at 3 V.

LED currents in this setting were measured and are between 10 mA for a green LED and 12 mA for a red LED. This is sufficient under normal operation. ATtiny24 operating current is smaller and is optimized by interrupt operation and sleep mode idle. In time selection mode LEDs are switched off after 60 seconds of inactivity, in run mode the LEDs are switched on and off for blinking so that the overall power consumption is further reduced.

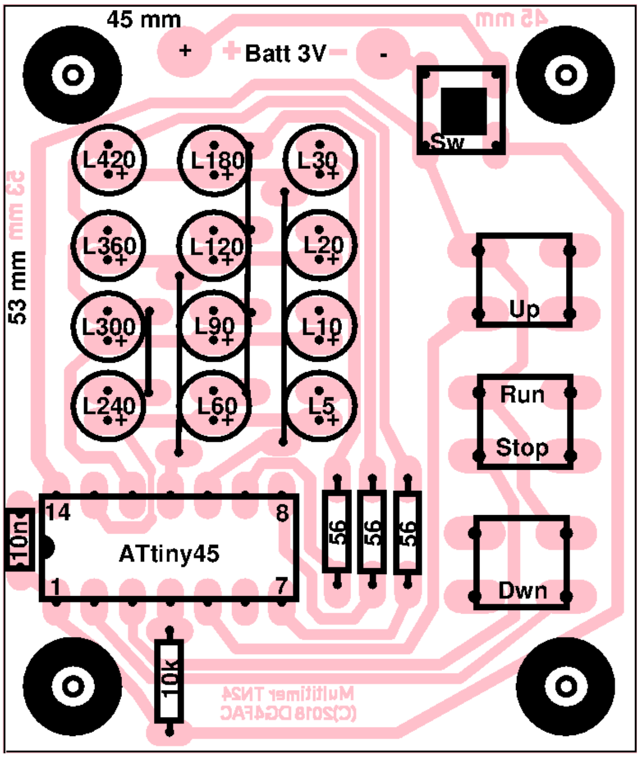


Page top | Hardware | Mounting | Software

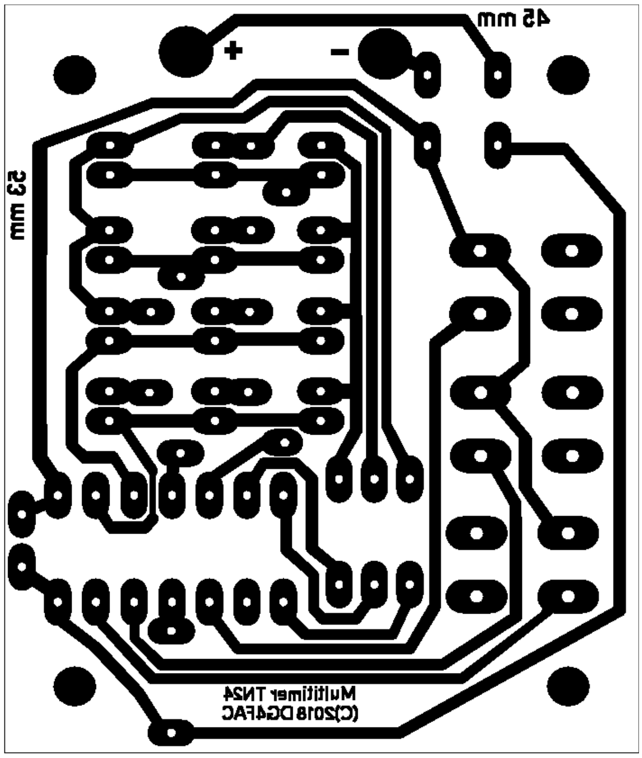
2 Mounting

2.1 PCB

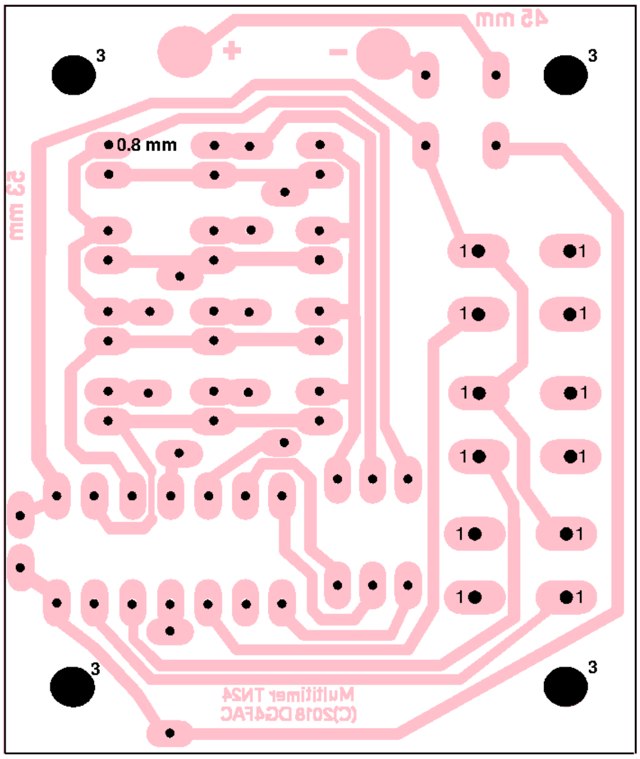
This is the 45-by-53 mm PCB layout. The dimensions were chosen to fit into a Euro-PCB of 100-by-160 mm.



This is the drill plan for the PCB, all unmarked holes are 0.8 mm.

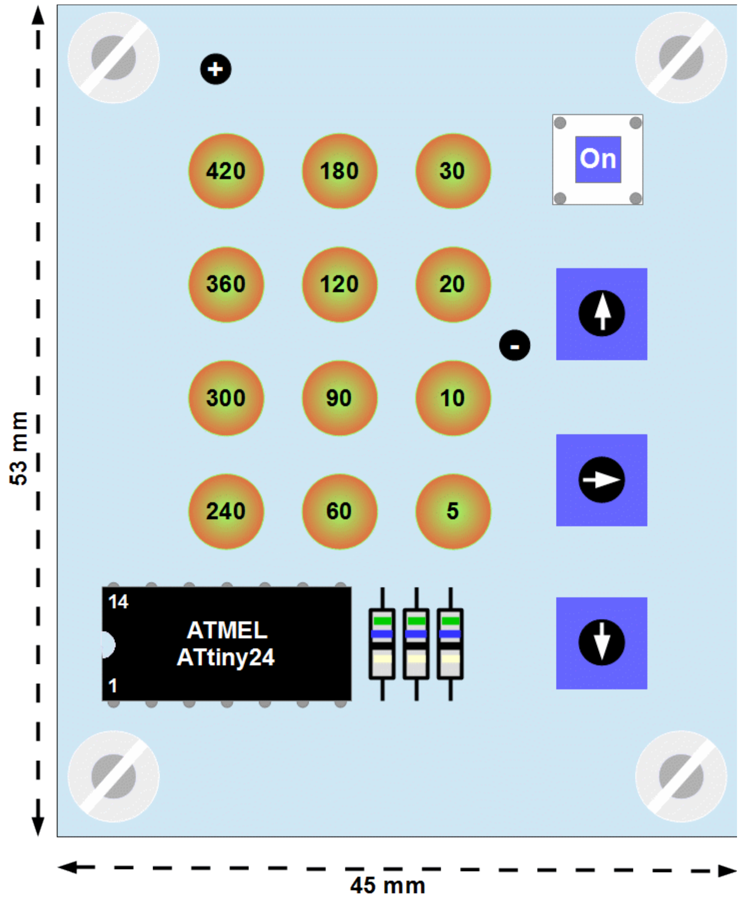


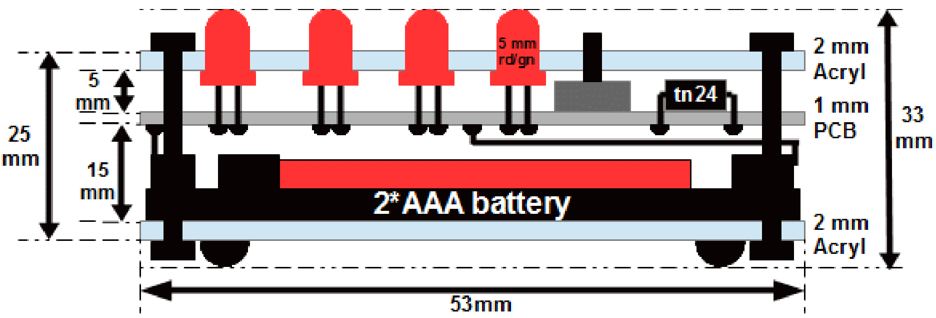
Left is the component placement on the PCB.



2.2 Mounting

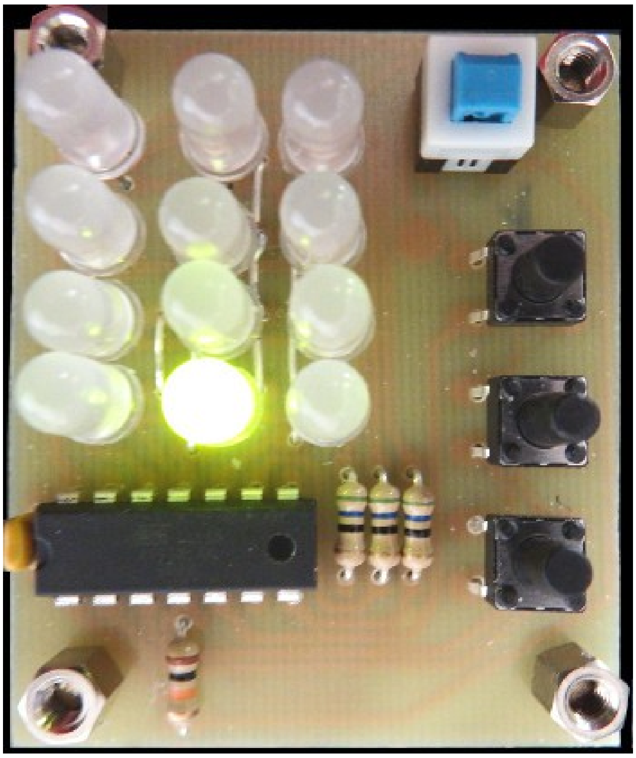
This is the top view of the device as it results from the component placement on the PCB. The buttons and the switch are designed for right-hand operation (sorry, left-handers).



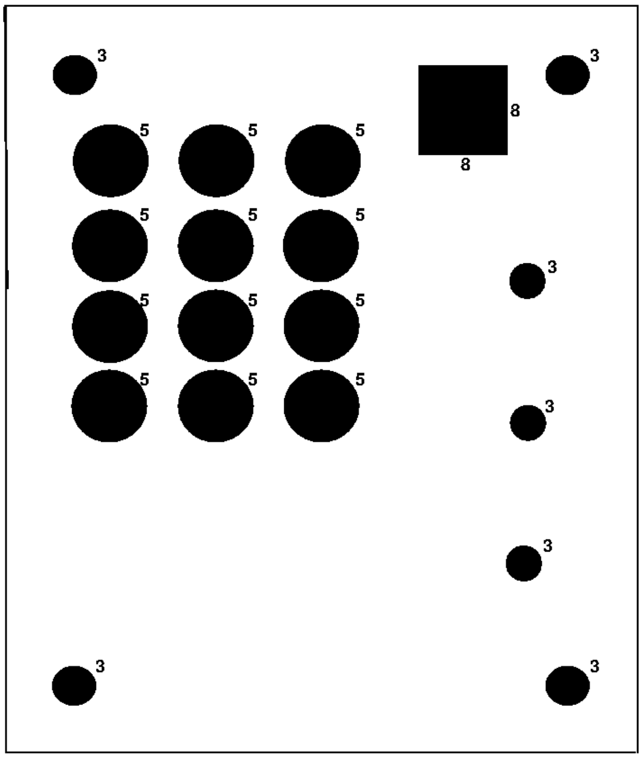


This is the side-view on the mounted device. It shows the acrylic glass casing on top and on the bottom and how the controller, the LEDs, the buttons and the AAA batteries fit into that. The battery is mounted to the PCB with two short cables and can be easily replaced by removing the lower acrylic glass layer.

This is the PCB with all components mounted.



This is the drill plan for the acrylic glass cov-
er.



Page top	Hardware	Mounting	Software

3 Software

The software is written in assembler, of course, to have strict control over the whole timing. The controller is in sleep mode to lower the power consumption as far as possible down to the active LED current. Sleep mode is only interrupted by the timer (0.1 second rhythm) and external key events (PCINT).

The following chapters demonstrate the structure of the software and how it works in detail.

1. 3.1 provides links to the source code,
2. 3.2 provides hardware debugging options,
3. 3.3 show the key's PCINT interrupt,
4. 3.4 show the timer's interrupt generation at each 0.1 seconds,
5. 3.5 demonstrates the 0.1 s interrupt processing, and
6. 3.6 shows the blinking rhythm's characteristic and how to achieve it.

One special design decision of the software is that all functions are performed within the two interrupt service routines. No code is executed outside this except the sleep and jump instruction back to the loop. This feature, while a little bit exotic, allows to produce compact code.

3.1 Source code

The assembler source code is available here for download and can be viewed in the attachment.

3.2 Hardware debug options

The source code starts with two hardware debug options: Debug_Leds = 1 switches the LEDs on with the first round in green and the second round in red and repeating forever, Debug_Switches = 1 lights one specific LED if one of the three buttons are pressed. You can use these options to test your hardware.

3.3 Key PCINT interrupt

This interrupt occurs whenever a key input pin changes its state. The routine has to

- detect the pressed key (Down, Start/Stop, Up), and
- take the selected actions, and
- debounce the key inputs by setting a register (rTgl) to its start value whenever a key has been pressed, and to
- not to accept any key action whenever rTgl is not zero.

First of all the input port, to which the keys are attached to, is read. It then masks all bits in the port to which no key is attached, by a one and compares the result with 0xFF. If that is the case no key is pressed and the routine returns from interrupt.

If at least one key is pressed it is tested if the toggle register rTgl is at zero. This register is

1. set by any detected key event to an initial value (cTgl),
2. is down-counted in each tenth of a second by the deci-second routine, and
3. when zero allows key events.

If rTgl is larger than zero, the initial value is restarted again and the service routine is finalized.

If, with rTgl at zero, the Run/Stop key is pressed (input pin is low) the bRun flag is inverted. If the flag is zero, the number of the current LED is set to the initially selected LED number and the LED is updated. The LED's color is green when bRun is cleared.

If bRun is set after inversion, the start procedure is absolved:

1. the 16 bit seconds counter rSecH:rSecL is set to the number of seconds of the selected LED,
2. the registers rPort, rDdr and rLedOff are set according to the number of seconds.

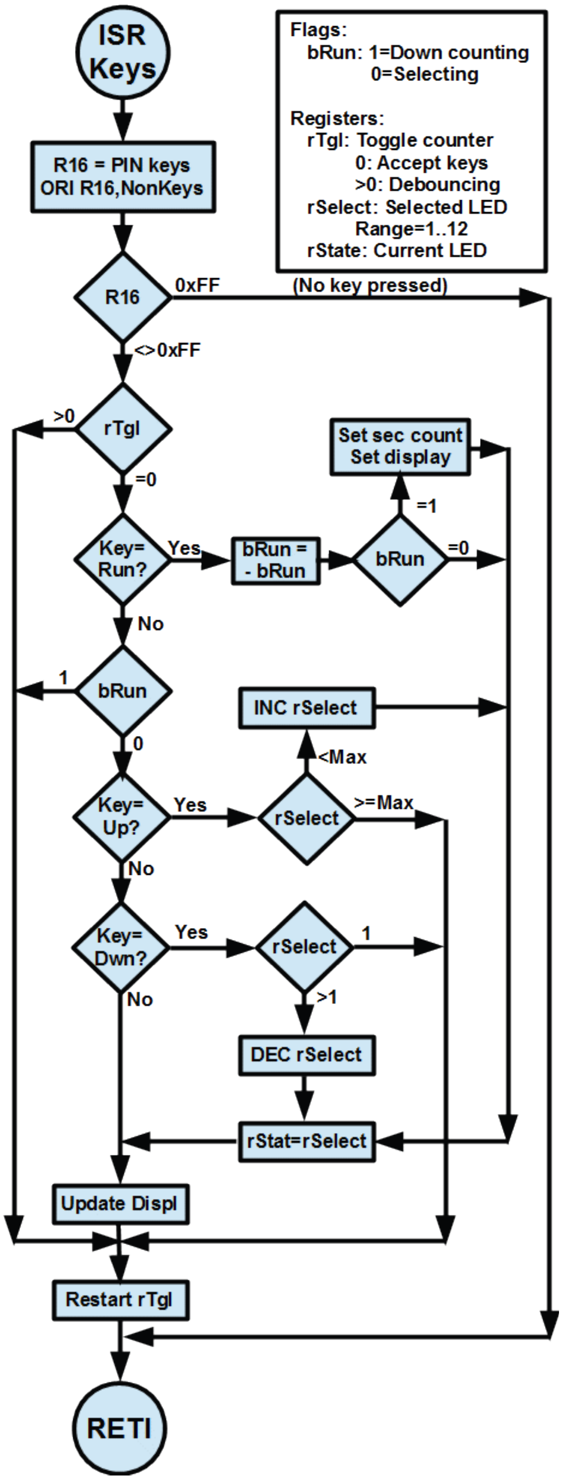
The selected LED is copied to the rState register and is displayed. The further processing of the LED display (down counting, PWM type of blinking) is performed by the Decisecond interrupt service routine.

If the Run/Stop key is not pressed it is first checked if the bRun flag is clear (no reaction on keys when down counting is active).

If the Up key is pressed, it is checked if the selected key number is already at its maximum (12). If yes, no action follows. If no, the selected LED is increased, copied to rState and the new LED is displayed.

If the Down key is pressed, it is checked if the selected key number is at its minumum (1). If yes, no action follows. If not the selected LED is decreased, copied to rState and the LED is displayed.

The source code for this flow is not listed here, it can be seen from the source code.

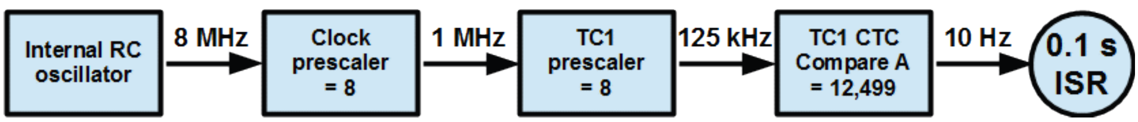


3.4 Decisecond timer generation

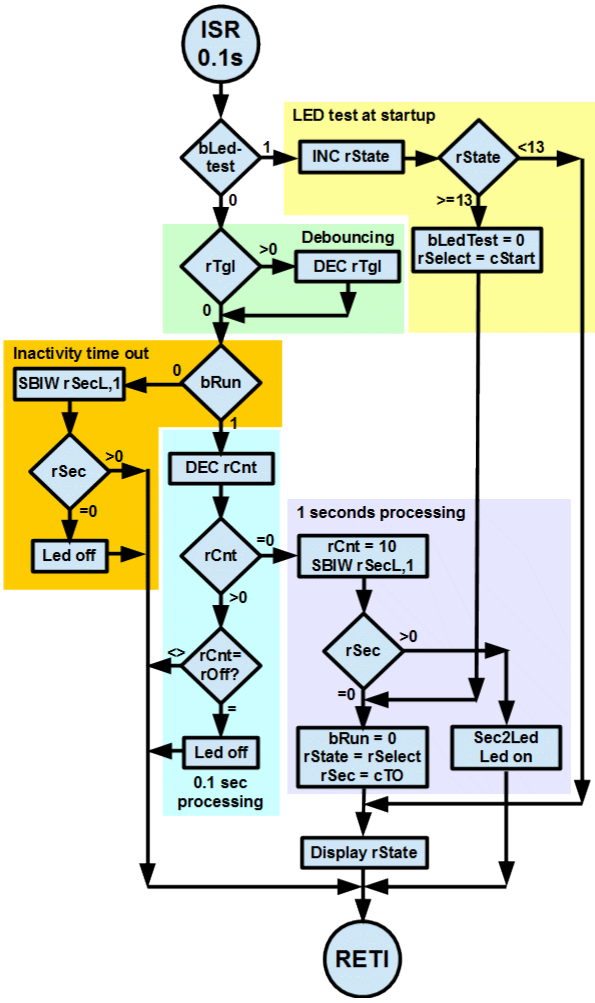
By default the ATtiny24 runs with a clock rate of 1 MHz by dividing the internal RC oscillator's frequency of 8 MHz by the clock prescaler of 8. For the blinking of the red LEDs (see 3.4) a 0.1 s timing is necessary. This is achieved by

- prescaling the 16 bit timer/counter TC1 by 8, and
- dividing the prescaled clock rate of 125 kHz by 12,500 in CTC mode.

This provides with a compare match interrupt every 0.1 s. Of course, for the second the 0.1 s pulse has to be divided by 10 before the time advances.



3.5 Decisecond timer interrupt



This is the complete timer interrupt flow chart. The different sections are marked to demonstrate their structure.

On start-up the flag bLedtest is set and the LEDs run up from 1 to 12. During this phase the register rState is increased and the LED in rState is displayed in red color. If rState reaches 13 the flags bLedtest is cleared and rSelect is set to the desired start value (by default 5). Further execution is the same as if the timer has counted down to zero: the flag bRun is cleared, the LED number rState is set to its pre-selected value in rSelect and the seconds counter is set to its default time out value (cTO = 600 for 60 seconds). The LED in rState is displayed (now in green because bRun is clear).

If outside the ledtest the debouncer register rTgl is decreased if it is not zero. This provides for the default of 0.3 s debouncing time.

If the down-counter is not running the inactivity time in 16 bit counter rSecH:rSecL is decreased. If that reaches zero, the LED is switched off by clearing the direction port of the LEDs.

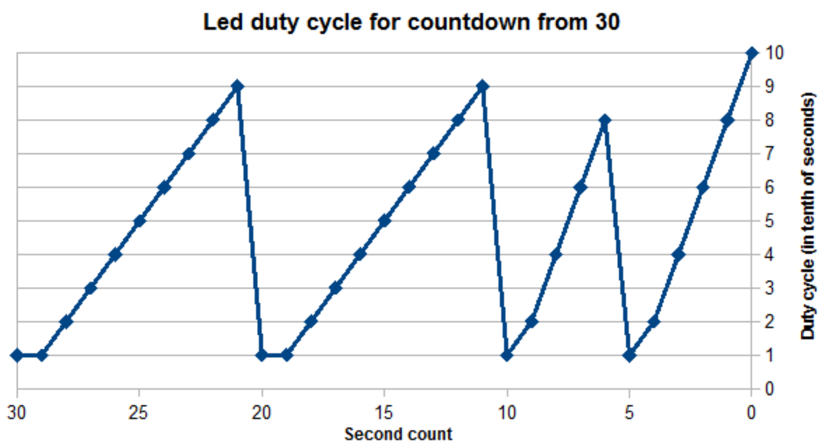
If running the rCnt register, which counts the tenth of seconds, is decreased. If that does not reach zero the rCnt value is compared with the rOff value. If equal the LED is switched off by writing 0 to the DDRA port (LED blinking in PWM mode).

If the 10-divider reaches zero, it is reloaded with 10 and the seconds counter in rSecH:rSecL is decreased. If this 16 bit register does not reach zero, the second count is converted to

1. the LED number in rState, and
2. the rOff value of this second.

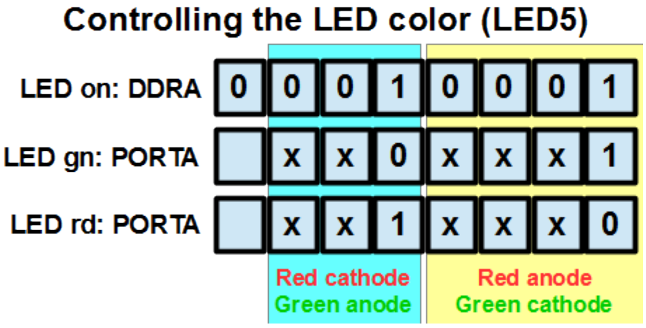
The LED is switched on in red (bRun flag = 1).

If the 16 bit counter reaches zero, the bRun flag is cleared, the rState is set to the rSelected register, the time-out value for inactivity is set to its default (600 = 60 s) and the LED in rState is displayed.



3.6 Blinking rhythm

This is the LED control setting. As an example the LED5 is displayed. This LED has its green anode on portpin PA0, its red anode on PA4. To be on, both direction bits have to be high. If PA0 is high and PA4 is low current flows in the green direction. If both bits are reversed the current flows in the opposite direction and the LED is red. All other outputs have their direction bits clear so their polarity does not matter.



To switch the LED off for blinking it is sufficient to write zero to the direction port, to turn it on again to write the direction byte to the direction port again.

This is the assembler code to control the color of LED5, depending from the current state of the bRun flag. An exclusive or (eor) with 0x7F inverts the bit polarity for all LED bits.

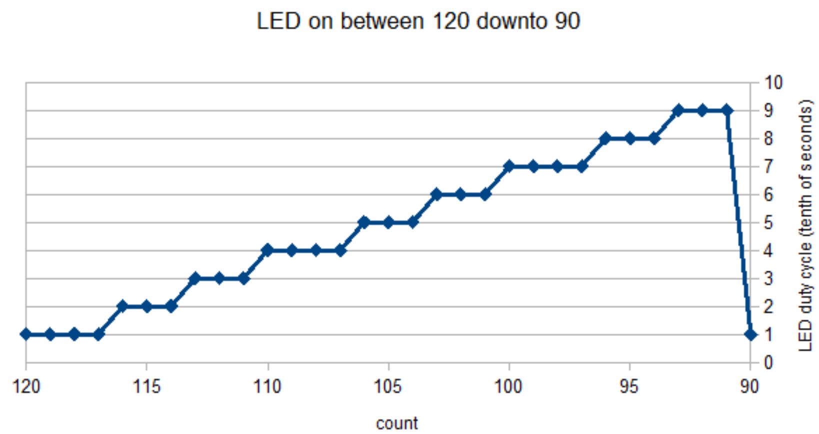
Assembler example:

```
ldi R16,0x01 ; Load R16 for PORTA with green LED
ldi R17,0x7F ; Load R17 with inverter
sbrc rFlag,bRun ; Is time running? Skip next if not
eor R16,R17 ; Yes, invert to red color
out PORTA,R16 ; To the output driver
ldi R16,0x11 ; Turn LED5 on
out DDRA,R16 ; Set direction port for LED5
```

If the 12 LED combinations of the direction and output ports are written to a table in the source code, even falsely mounted LEDs can be adopted: just reverse the output bit combinations.

In run mode the red LED blinking is working like that (here described for the 30 second selection):

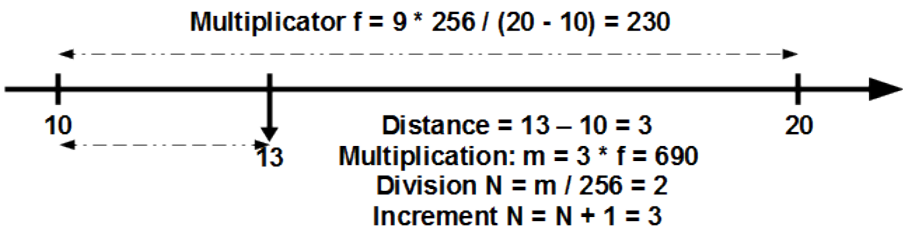
- The 30 second LED is blinking, duty cycle for on is 0.1 second with 0.9 seconds off.
- The nearer the time approaches 20 seconds left, the higher is the duty cycle (see diagram). With 21 seconds left the duty cycle is 90% (0.9 seconds on, 0.1 seconds off).
- When 20 seconds are reached, LED20 is pulsing red with 0.1 seconds on and 0.9 seconds off. The nearer the remaining time comes to 10 seconds the longer the duty cycle gets (see diagram).

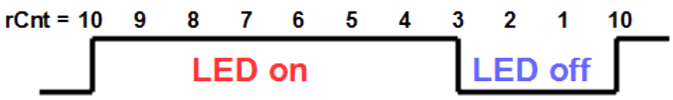


- The same happens when 5 seconds are reached.
- Finally, at zero rest time, the run mode is switched off and the LED30 is permanently green.

Shown above is the cycle between 120 and 90 and the resulting LED duty over time.

The calculation of the LED's duty cycle goes as follows (here with the example of times between 20 and down to 11 seconds and time at 13 seconds.





In software the LED's duty cycle is achieved by multiplying the difference between the time in seconds to the next lower limit (10) with a factor f that represents the difference between the upper (20) and lower (10) limit, divided into 9 stages and multiplied by 256 to avoid floating point math (just because it is simpler, less time consuming and less memory extensive that floating point math). f is calculated with the formula

$$f = 9 * 256 / (N_{upper} - N_{lower})$$

f for the different time periods is

- between 420 and 121: 38,
- between 120 and 31: 77,
- between 30 and 11: 230,

in any case smaller than 256.

In the example's case the multiplication factor f is 230. The multiplication of the difference of 3 with 230 leads to a 16 bit wide result of decimal 690 or 0x02B2. Dividing this result by 256 (simply ignoring the LSB of the result) leads to 2. Adding 1 to it yields the switch off cycle in register rOff:

$$N = (T - N_{lower}) * f / 256 + 1$$

Note that the division result is rounded down by ignoring the LSB of the result.

In the example: with the time in seconds at 13 the rOff value is 3. The LED will be on for the 0.1 s pulses in rCnt between the tenth and the third cycle, then off for three cycles.

In assembler the following procedures are followed. The LED to be blinking at a certain time is calculated by stepping through a table of durations and counting at which table entry the time is smaller or equalling the table entry. The count is the LED to be blinked.

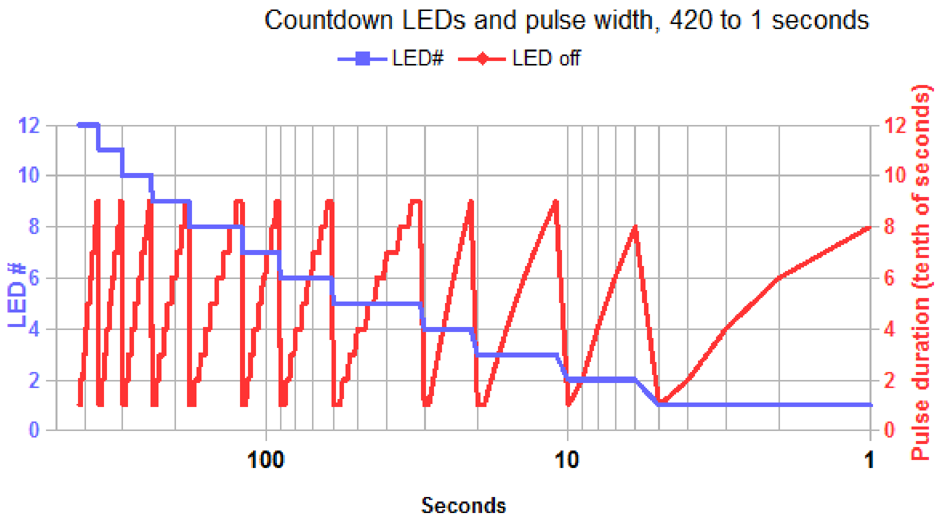
```
; Calculate LED from second counter
;   Converts the counter value in rCntH:rCntL
;     to the LED to be driven in rState
Sec2Led:
  ldi ZH,High(2*LedDur)
  ldi ZL,Low(2*LedDur)
  clr rmp ; rmp is counter
Sec2Led1:
  inc rmp ; Increase counter
  lpm XL,Z+ ; Read LSB from table to X
  lpm XH,Z+
  sec
  cpc rSecL,XL ; Compare LSB
  cpc rSecH,XH ; Compare MSB
  brcc Sec2Led1 ; Repeat
  mov rState,rmp ; Set LED number
  ret
;
; Duration table
LedNull:
.dw 0 ; Value is needed as lower limit for LED5
LedDur:
```

```
.dw 5,10,20,30
.dw 60,90,120,180
.dw 240,300,360,420
.dw 65535 ; End of table
;
```

Between 420 and 11 this is an 8-by-8 bit multiplication with a 16 bit result (of which the LSB is calculated but ignored). As the ATtiny24 has no hardware multiplicator multiplication this is done via software:

```
; Calculate LED duty cycle
;   R16 has LED number between 0 and 11
;   R17 is the difference between the next lower time limit
;     and the current time
;   Result is in rDuty
Duty:
  clr ZH ; Result MSB to zero
  tst R17 ; Is the difference zero?
  breq DutyZero
  ldi ZH,High(2*MultList) ; Multiplicator list
  ldi ZL,Low(2*MultList)
  add ZL,R16 ; Add LED number
  ldi R16,0
  adc ZH,R16 ; Add carry
  lpm R16,Z ; Read multiplicator
  tst R16 ; Zero or one?
  breq DutyLow ; Yes, treat different
  clr ZH ; Z for multiplication result
  clr ZL
  push R0 ; Save, use as MSB for multiplicator
  clr R0 ; Clear MSB
DutyMult:
  lsr R16 ; Shift lowest bit to carry
  brcc DutyMult1 ; If carry clear do not add to result
  add ZL,R17 ; Add multiplicator LSB
  adc ZH,R0 ; Add multiplicator MSB and carry
DutyMult1:
  lsl R17 ; Shift Multiplicator left
  rol R0 ; And highest bit to MSB
  tst R16 ; Already done?
  brne DutyMult ; Go on multiplying
  pop R0 ; Restore R0
DutyZero:
  inc ZH ; Add one to MSB
  mov rDuty,ZH ; Set rDuty from MSB result
  ret
; Smaller or equal 10
DutyLow:
  ldi ZH,High(2*TenTable) ; Point to ten table
  ldi ZL,Low(2*TenTable)
  add ZL,R17 ; Add to list
  ldi R16,0
  adc ZH,R16
  lpm rDuty,Z ; Read from list
  ret
;
; Multiplicator list for LED5 to LED420
MultList:
  .db 0,0 ; LED5 and LED10 are extra
  .db 230,230 ; LED20 and LED30
  .db 77,77 ; LED60 and LED90
  .db 77,38 ; LED120 and LED180
  .db 38,38 ; LED240 and LED300
```

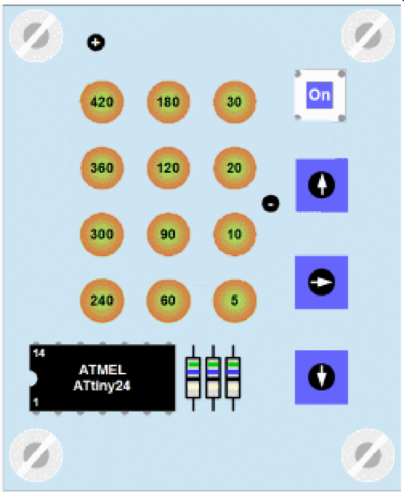
```
  .db 38,38 ; LED360 and LED420
;
; For the seconds from 10 to 1 it is easier to derive the
; tenth of seconds from a short table instead. The content
; of the table would be:
; Table with tenth of second duty cycle beween seconds
;   10 and 0
; Note: the higher the number the shorter the LED-on time
TenTable:
.db 9,8,6,4,2,10 ; Note: last value for even number of bytes
;
```



The results of these calculations over the whole counting range from 420 down to 1 is shown in the diagram.

Shown are the LED's numbers that are blinking during the different time phases and the durations over which theses LEDs are on: one tenth of a second for a very short pulse, five tens of a second for a half on/half off pulse and nine tenths of a second for a very long pulse.

Page top	Hardware	Mounting	Software



AVR applications

A multitimer with ATtiny24

Assembler source code

Source code for the multitimer tn24

HTML formatted assembler code, the original source code in asm format can be downloaded here.

```
;
; ****************************************
; * Multitimer with ATtiny24 and 12 LEDs *
; * Version 1.0 August 2018              *
; * (C)2018 avr-asm-tutorial.net         *
; ****************************************
;
.nolist
.include "tn24def.inc"
.list
;
; **********************************
;       H A R D W A R E
; **********************************
;
; Device: ATtiny24, Package: 14-pin-PDIP_SOIC
;
;
;              1 /‾‾‾‾‾‾|14
;        +3V o--|VCC GND|--o GND
; Taste Dwn o--|PB0 PA0|--o Led An 0
;  Taste Go o--|PB1 PA1|--o Led An 1
;     RESET o--|RES PA2|--o Led An 2
;  Taste Up o--|PB2 PA3|--o Led An 3
;        NC o--|PA7 PA4|--o Led Cat 0
; Led Cat 2 o--|PA6 PA5|--o Led Cat 1
;              |_______|
;
; *************************************
;  H A R D W A R E   D E B U G G I N G
; *************************************
;
; All hardware debugging code starting
;   at 000000 and ending in an indefinite
;   loop
;
; Debug the LEDs
;   LEDs are turned on from LED5 to LED420
;   First round: all in green
```

```
;   Second round: all in red
.equ Debug_Leds = 0 ; 1 = Debug, 0=Normal
;
.if Debug_Leds == 1
.equ cDelay=50000
  ldi R16,0 ; LEDs off
  clr R18
Debug_Led1:
  ldi ZH,High(2*LedTable)
  ldi ZL,Low(2*LedTable)
  mov R17,R16
  lsl R17
  add ZL,R17
  ldi R17,0
  adc ZH,R17
  lpm R17,Z+
  eor R17,R18
  out PORTA,R17
  lpm R17,Z
  out DDRA,R17
  ldi ZH,High(cDelay)
  ldi ZL,Low(cDelay)
Debug_Led2:
  sbiw ZL,1
  brne Debug_Led2
  inc R16
  cpi R16,13
  brcs Debug_Led1
  clr R16
  ldi R17,0x7F
  eor R18,R17
  rjmp Debug_Led1
  .endif
;
; Debug the switches
;   Checks the three switches
;     State of switch is displayed on
;       LED5 (Down), LED60 (Run/Stop) and LED240 (Up)
;     As long as pressed the
.equ Debug_Switches = 0 ; 1=Debug, 0=Normal
;
.if Debug_Switches == 1
Debug_Sw:
  ldi R16,0x07
  out PORTB,R16
  in R17,PINB
  ldi R16,1
  sbrs R17,0
  rjmp Debug_Sw_Nmbr
  ldi R16,5
  sbrs R17,1
  rjmp Debug_Sw_Nmbr
  ldi R16,9
  sbrs R17,2
  rjmp Debug_Sw_Nmbr
  clr R16
Debug_Sw_Nmbr:
  ldi ZH,High(2*LedTable)
  ldi ZL,Low(2*LedTable)
  lsl R16
  add ZL,R16
  ldi R16,0
  adc ZH,R18
  lpm R16,Z+
```

```
  out PORTA,R16
  lpm R16,Z+
  out DDRA,R16
  clr R16
Debug_Sw_Nmbr1:
  dec R16
  brne Debug_Sw_Nmbr1
  rjmp Debug_Sw
  .endif
;
; **********************************
;   A D J U S T A B L E   C O N S T
; **********************************
;
.equ clock=1000000 ; Define clock frequency
;
; Led state at start-up
.equ cStart = 1 ; Can be between 1 and 12
;
; Debouncing counter, in 0.1 seconds
.equ cDebounce = 2 ; Number of periods
;
; Auto blank when inactive
;   in tenth of seconds
.equ cAutoOff = 100 ; Automatic off
;
; **********************************
;  F I X  &  D E R I V.  C O N S T
; **********************************
;
; TC1 produces 0.1 Hz signal
.equ cTc1Presc = 8 ; TC1 prescaler
.equ cTc1Div = clock / cTc1Presc / 10 ; TC1 CTC divider
.equ cTc1CmpA = cTc1Div-1 ;
;
; Key debouncing counter
.equ cTgl = cDebounce + 1
;
; **********************************
;       R E G I S T E R S
; **********************************
;
.def rOff = R10 ; Switch LED off
.def rSelect = R11 ; Current led selected
.def rState = R12 ; Current led state
.def rPort = R13 ; Current Port
.def rDdr = R15 ; Current DDR
.def rmp = R16 ; Define multipurpose register
.def rFlag = R17 ; Flag register
  .equ bRun = 0 ; Down count running
  .equ bTimeOut = 1 ; End of LED display
  .equ bLedTest = 7 ; Led testing at startup
.def rCnt = R18 ; Count down 0.1 seconds
.def rTgl = R19 ; Debounce toggle register
; free: R20 to R23
.def rSecL = R24 ; Half seconds counter, LSB
.def rSecH = R25 ; dto., MSB
; used: R27:R26 = X for multiplication
; free: R29:R28 = Y
; used: R31:R30 = Z for diverse purposes
;
; **********************************
;         C O D E
; **********************************
```

```
;
.cseg
;
; **********************************
; R E S E T  &  I N T - V E C T O R S
; **********************************
  rjmp Main ; Reset vector
  reti ; EXT_INT0, unused
  reti ; PCI0, unused
  rjmp Pcint1Isr ; PCI1
  reti ; WATCHDOG, unused
  reti ; ICP1, unused
  rjmp Tc1CmpAIsr ; OC1A
  reti ; OC1B, unused
  reti ; OVF1, unused
  reti ; OC0A, unused
  reti ; OC0B, unused
  reti ; OVF0, unused
  reti ; ACI, unused
  reti ; ADCC, unused
  reti ; ERDY, unused
  reti ; USI_STR, unused
  reti ; USI_OVF, unused
;
; **********************************
;  I N T - S E R V I C E  R O U T .
; **********************************
;
; PCINT1 external int
;   executed on every ppin change of key inputs
;   identifies pressed key and starts respective
;     actions
Pcint1Isr:
  tst rTgl ; Check toggle counter
  brne PcInt1Isr9 ; Toggle period has not ended yet
  in rmp,PINB ; Read keys
  ori rmp,0b11111000 ; Set all unused pins
  cpi rmp,0xFF ; No key pressed
  breq Pcint1Isr9
  ldi ZL,cTgl
  mov rTgl,ZL
  sbrs rmp,1 ; Run/Stop key?
  rjmp KeyRun
  sbrc rFlag,bRun
  reti
  ldi rSecH,High(cAutoOff)
  ldi rSecL,Low(cAutoOff)
  sbrs rmp,0 ; Down key?
  rjmp KeyDown
  sbrs rmp,2 ; Up key?
  rjmp KeyUp
PcInt1Isr9:
  reti
;
KeyDown:
  mov rmp,rSelect
  cpi rmp,1 ; Already at lowest end?
  breq KeyDown1 ; Yes, ignore pulse
  dec rSelect
  mov rState,rSelect
  rcall SetLed
KeyDown1:
  reti
;
```

```
KeyUp:
  mov rmp,rSelect
  cpi rmp,12
  brcs KeyUp1
  ldi rmp,11
  mov rSelect,rmp
KeyUp1:
  inc rSelect
  mov rState,rSelect
  rcall SetLed
  reti
;
KeyRun:
  ldi rmp,1<<bRun ; Invert run flag
  eor rFlag,rmp
  sbrc rFlag,bRun ; Skip next if bRun is clear
  rjmp KeyStop
  ldi ZH,High(2*LedDur)
  ldi ZL,Low(2*LedDur)
  mov rmp,rSelect
  lsl rmp
  add ZL,rmp
  ldi rmp,0
  adc ZH,rmp
  lpm rSecL,Z+
  lpm rSecH,Z
  mov rState,rSelect
  ldi rmp,9
  mov rOff,rmp
  ldi rmp,10
  mov rCnt,rmp
  rcall SetLed
  reti
;
KeyStop:
  ldi rSecH,High(cAutoOff) ; Load automatic off counter
  ldi rSecL,Low(cAutoOff)
  mov rState,rSelect
  rcall SetLed
  reti

;
; TC1 Compare A int
;   executed any 0.1 seconds when counting
;   decreases count
;     when zero: switches off counting
;       and goes back to display selected time
;     when odd: switches current LED off
;     when not odd: switches current LED on
;     when smaller than lower limit:
;       decreases state and switches to lower
;         LED
Tc1CmpAIsr:
  sbrs rFlag,bLedTest ; Test LED?
  rjmp Tc1CmpAIsrRun
  inc rState ; Next Led
  ldi rmp,13 ; Last LED?
  cp rState,rmp
  brcs Tc1CmpAIsrLed ; No, display LED
  rjmp Tc1CmpAIsrStart
Tc1CmpAIsrRun:
  tst rTgl ; Check toggle register
  breq Tc1CmpAIsrRun1
  dec rTgl
```

```
Tc1CmpAIsrRun1:
  sbrs rFlag,bRun ; Counting active?
  rjmp Tc1CmpAIsrAuto ; No, to Auto off
  dec rCnt ; Count 0.1 s counter down
  breq Tc1CmpAIsrSec
  cp rCnt,rOff ; End of PWM cycle?
  brne Tc1CmpAIsrReti
  clr rmp ; Clear LED
  out DDRA,rmp
  reti
Tc1CmpAIsrSec:
  ldi rCnt,10 ; Restart 0.1 s counter
  sbiw rSecL,1 ; Count seconds down
  breq Tc1CmpAIsrStart ; End of count
  rcall Sec2Led
  rjmp Tc1CmpAIsrLed
Tc1CmpAIsrAuto:
  sbrc rFlag,bTimeOut ; Timed out?
  reti ; Yes
  sbiw rSecL,1 ; Auto off counter
  brne Tc1CmpAIsrBlink ; Not at zero
  clr rmp ; Switch LED off
  out DDRA,rmp
  sbr rFlag,1<<bTimeOut ; Set time out flag
  reti
Tc1CmpAIsrBlink:
  ldi rmp,0
  sbrs rSecL,0
  out DDRA,rmp
  sbrc rSecL,0
  out DDRA,rDdr
  reti
Tc1CmpAIsrStart:
  ; Switch to start
  cbr rFlag,(1<<bRun)|(1<<bLedtest) ; Switch run and bLedtest off
  mov rState,rSelect ; switch to selected state
  ldi rSecH,High(cAutoOff) ; Set auto off value
  ldi rSecL,Low(cAutoOff)
Tc1CmpAIsrLed:
  rcall SetLed
Tc1CmpAIsrReti:
  reti

; ******************************
;  I S R   S U B R O U T I N E S
; ******************************
;
;
; Calculate LED from second counter
;   Converts the seconds in rSecH:rSecL
;     to the LED to be driven in rState
;     and the pulse duration in rOff
Sec2Led:
  ; Get LED for seconds time
  ldi ZH,High(2*LedDur+2)
  ldi ZL,Low(2*LedDur+2)
  clr rState ; rState is LED # counter
Sec2Led1:
  inc rState ; Increase counter
  lpm XL,Z+ ; Read LSB from table to X
  lpm XH,Z+
  sec
  cpc rSecL,XL ; Compare LSB
  cpc rSecH,XH ; Compare MSB
```

```
  brcc Sec2Led1 ; Repeat
  ; Read lower limit to X
  sbiw ZL,4 ; Point to pre last entry in table
  lpm XL,Z+ ; Read LSB lower limit to X
  lpm XH,Z ; dto., MSB
  ; Difference time - lower count
  mov ZH,rSecH ; Copy time
  mov ZL,rSecL
  sub ZL,XL ; Subtract from time
  sbc ZH,XH
  mov XH,ZH ; Copy to X
  mov XL,ZL
  ; Get multiplicator for LED number
  ldi ZH,High(2*MultTab)
  ldi ZL,Low(2*MultTab)
  add ZL,rState
  ldi rmp,0
  adc ZH,rmp
  lpm rmp,Z
  ; Test multiplicator = 0
  tst rmp
  brne Sec2Led2 ; Not zero, multiply
  ; LED5 or LED10
  ldi ZH,High(2*TenTable)
  ldi ZL,Low(2*TenTable)
  add ZL,XL ; Add time LSB
  adc ZH,XH ; dto., MSB
  lpm rOff,Z ; Read PWM value from table
  ret
Sec2Led2:
  ; >LED10, multiplicator not zero, multiply
  clr ZL ; Z is result
  clr ZH
Sec2Led3:
  tst rmp ; Ready multiplying?
  breq Sec2Led5 ; Yes, end of multiplication
  lsr rmp ; Divide multiplicator by 2, lowest bit to carry
  brcc Sec2Led4 ; Carry clear, do not add to result
  add ZL,XL ; Add multiplicator
  adc ZH,XH
Sec2Led4:
  lsl XL ; Multiply by 2
  rol XH
  rjmp Sec2Led3 ; Go on multiplying
Sec2Led5:
  inc ZH ; Plus one
  mov rOff,ZH ; To off store
  ret
;
; Duration table
LedDur:
.dw 0,5,10,20,30
.dw 60,90,120,180
.dw 240,300,360,420
.dw 65535 ; End of table
;
; Multiplicator table
MultTab:
.db 0,0,0,230,230,77,77,77,38,38,38,38,38,1
;
; For the seconds from 10 to 1 it is easier to derive the
; tenth of seconds from a short table instead. The content
; of the table would be:
; Table with tenth of second duty cycle beween seconds
```

```
;   10 and 0
; Note: the higher the number the shorter the LED-on time
TenTable:
.db 0,2,4,6,8,9
;
; Switch to LED in rState
SetLed:
  mov rmp,rState
  lsl rmp ; Multiply by 2
  ldi ZH,High(2*LedTable) ; Point to led table
  ldi ZL,Low(2*LedTable)
  add ZL,rmp
  ldi rmp,0
  adc ZH,rmp
  lpm rPort,Z+
  lpm rDdr,Z
  ldi rmp,0x7F ; Invert to red?
  sbrc rFlag,bRun ; Skip next if not running
  eor rPort,rmp
  out PORTA,rPort
  out DDRA,rDdr
  cbr rFlag,1<<bTimeOut
  ret
;
; Led table of the ports
;   1. Byte: PORT, 2. Byte: DDR
LedTable:
.db 0b00000000,0b00000000 ; off
.db 0b00010000,0b00010001 ; LED 5 green
.db 0b00010000,0b00010010 ; LED 10 green
.db 0b00010000,0b00010100 ; LED 20 green
.db 0b00010000,0b00011000 ; LED 30 green
.db 0b00100000,0b00100001 ; LED 60 green
.db 0b00100000,0b00100010 ; LED 90 green
.db 0b00100000,0b00100100 ; LED 120 green
.db 0b00100000,0b00101000 ; LED 180 green
.db 0b01000000,0b01000001 ; LED 240 green
.db 0b01000000,0b01000010 ; LED 300 green
.db 0b01000000,0b01000100 ; LED 360 green
.db 0b01000000,0b01001000 ; LED 420 green
;
; **********************************
;  M A I N   P R O G R A M   I N I T
; **********************************
;
Main:
  ; Stack init
  .ifdef SPH
    ldi rmp,High(RAMEND) ; Set SPH
    out SPH,rmp
    .endif
  ldi rmp,Low(RAMEND)
  out SPL,rmp ; Init LSB stack pointer
  ; Clear LED
  clr rState
  rcall SetLed
  ; Set default start values
  ldi rmp,cStart
  mov rSelect,rmp ; Start with preselected value
  ldi rSecH,High(cAutoOff) ; Set auto off value
  ldi rSecL,Low(cAutoOff)
  ; Init PCINT for keys
  ldi rmp,(1<<PORTB0)|(1<<PORTB1)|(1<<PORTB2) ; Pull ups
  out PORTB,rmp ; on
```

```
  clr rmp ; Configure as inputs
  out DDRB,rmp
  ldi rmp,(1<<PCINT8)|(1<<PCINT9)|(1<<PCINT10) ; Mask key pins
  out PCMSK1,rmp
  ldi rmp,1<<PCIE1 ; Enable PCINT1
  out GIMSK,rmp
  ; Init TC1
  ldi rFlag,(1<<bLedTest)|(1<<bRun) ; Led test phase on
  ldi rmp,High(cTc1CmpA) ; Set compare value
  out OCR1AH,rmp ; MSB
  ldi rmp,Low(cTc1CmpA)
  out OCR1AL,rmp ; LSB
  clr rmp ; Mode port A
  out TCCR1A,rmp ; Control port TC1 A
  ldi rmp,(1<<WGM12)|(1<<CS11) ; CTC on compare A, presc=8
  out TCCR1B,rmp ; Control port TC1 B
  ldi rmp,1<<OCIE1A ; TC1 Compare A interrupt enable
  out TIMSK1,rmp ; in TC1 int mask
  ; Enable sleep
  ldi rmp,1<<SE ; Sleep mode idle
  out MCUCR,rmp
  ;
  ; Enable interrupts
  sei ; Enable interrupts
;
; **********************************
;    P R O G R A M   L O O P
; **********************************
;
Loop:
  sleep
  rjmp loop
;
; End of source code
;
```

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

To the top of that page