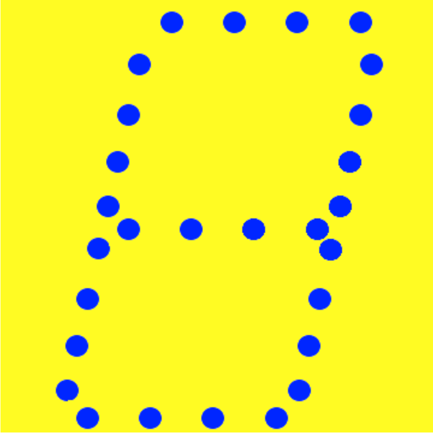
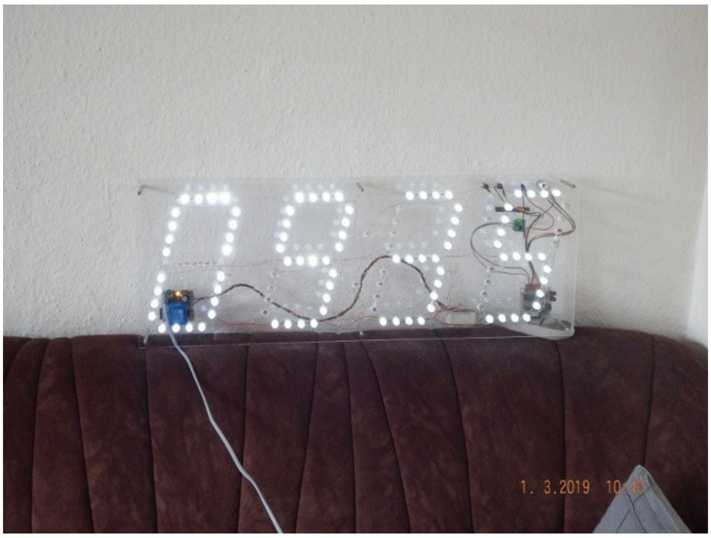
Path: Home => AVR-EN => Applications => Large watch m48



AVR applications

Large watch with ATmega48
Hardware, Mounting, Use and Software for the large watch

Large watch with ATmega48



1 Properties

The hardware properties are:

- Four digits with 7 segments each display hour and minute tens and ones,
- The seven segments consist of four 10mm LEDs, 28 LEDs per digit, very bright,
- two LEDs in the middle blink in seconds,
- Giant: 680-by-240 mm display front, can be read from a large distance,
- Low power consumption: by stacking the LEDs of a segment only 25 mA per segment or maximum 200 mA per digit (all segments on), multiplexing by four to reduce hardware parts,
- Manyfold adjustment of the watch:
    1. manually with two keys and a potentiometer,
    2. automatically by attaching a DCF77 receiver module,
- Brightness control either via a backlight sensor or via the potentiometer (configurable),
- Exact time: xtal controlled clock signal with 4.096 MHz xtal,
- Low overall power consumption: ca. 5 kWh for a whole year,
- Multiple diagnose tools integrated in the source code to identify wiring errors in the hardware,
- Well-documented free assembler source code, easily modifiable for different configurations,
- Free documentation with all source code and all calculation and all design graphics available,
- Simple re-programming within the running system by integrated ISP6 interface, therefore fast changes to the configuration possible.

2 Hardware

2.0 Preliminary remarks

When developing this hardware I collected the following experiences, that might be useful for others:

1. My first approach was to clock the controller directly with a 32.768 kHz crystal. That turned out to be impracticable for the following reasons. First: it is possible to clock the controller with that low frequency. My first try was erroneous: I fused the controller with an external crystal oscillator attached, not with an external crystal attached (error #1). ATMEL's Studio 4.19 did not recognize the error and returned a correctly verified setting. I don't know with which clock signal (as there should have been none) this verification was made, but anyway: a second try to access the fuses via ISP failed (of course and as to be expected).
   After repairing the error in HV/Parallel mode in an STK500, it didn't work either. I programmed at the planned 3.3V operating voltage and had set the brown-out-detection to 2.7V (error #2). As a test program I had burned a short sequence, blinking the attached green LED. But: nothing was blinking. Only if I rose the operating voltage to 5V (of cause without the display LEDs attached), the blinking started. The reason was that the brown-out detection already reacted on much higher voltages the programmed (up to 4V when set to 2.7V). Only after turning brown-out-detection off the blinking worked at whatever operating voltage. Therefore: do not trust ATMEL's brown-out-properties!
   For the third error I don't have such an easy work-around: the Studio refused to program the AVR with clock frequencies smaller than 5 kHz. It was necessary to select a 4 kHz ISP frequency because the next higher entry was already 57.6 kHz (which would certainly exceed one quarter of the 32.768 kHz clock = 8.192 kHz). So 4 kHz was the only alternative but the Studio does not allow to program flash or EEPROM at below 5 kHz. Even though the handbook on the ATmega48 does not have any programming frequency limits in ISP mode. So it must be an artificially set limit of the Studio software. Anyway: producing a rather complex source code without a working ISP interface is a mess, and I gave up the 32.768 kHz clock idea at that point. I have been wondering if the project would really work at that clock, and unfortunately had already took some efforts to optimize execution times at that time.
2. I had planned to avoid the 16 diodes for the constant-current transistors, that usually are necessary to limit the base voltage of the transistors, by reducing the operating voltage of the controller down to 3.3 V, so that the emitter voltages would go down to (3.3 - 0.6) = 2.7 V. With a 100Ω resistor the constant current would be 27 mA, ideal for the LEDs in multiplex mode.
   Usually one uses an Integrated Circuit regulator. Unfortunately 3.3 V regulators come only in SMD. Anyway, I tried a BA033FB, and soldered the three pins with some short wires. Unfortunately my operating voltage was not at 3.3 V but at more than 20 V. Fortunately I had the supply part not attached to the controller at that time, my ATmega48 and some of the transistors would have died that way.
   In those parts the ground pin is not attached to the middle pin (like their datasheet says - and lies about that) but only to the cooling pad. But connecting the cooling pad with minus did reveal error #4: the voltage on the output pin of the BA033FB was not around 3.3 V, but at 4.04 V instead. By that even exceeding the upper limit

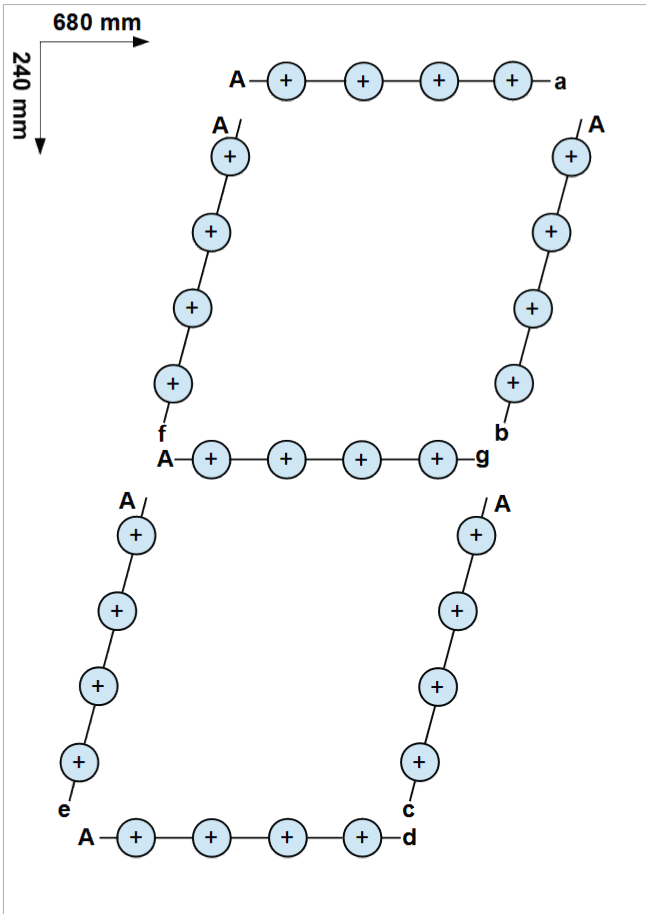
listed in the data sheet by far. That would have blown my LEDs with a by far too large current. At how many mA's will the specified voltage be reached? I don't know but rather fast decided to not use an Integrated regulator, because the next lower regulator, a 78L02, has a specified lowest voltage below the lower limit where an ATmega48 is reliably running (2.7 V). And constructed and build my own regulator with a zener diode and a transistor.

3. The transformer that I had calculated in advance would be a 2.8VA 2*12V type and would have fitted exactly my needs. But the one that was delivered (with 2*12V written on top) was a 2*15V type instead (error #5). The extra voltage nearly blew up my electrolytic capacitor as it produced 27 V at zero load (without consuming current). So I had to build in a small yellow LED to bring the no load voltage down to 25 V.
   The power of the transformer was really 2.8VA, but the higher voltage caused a smaller current than calculated (error #6). An attached 82Ω as load, usually consuming 200 mA (well below the specified 2*117 = 234 mA), brought the voltage to well below my LED consumption voltage of 13 V: a 2.8VA transformer with 2*15V only delivers 187 mA. So I had to reduce the LED's current consumption by reducing the operating voltage of the controller down to 2.7 V instead of the planned 3.3 V. Just to work around the false transformer. Fortunately the brightness of the LEDs is not very different between 27 and 21 mA.
4. Normally, when multiplexing smaller LED devices, a 50 or 60 Hz multiplexing frequency is sufficient and no flickering occurs. Not so with the very large digits here: even a 75 Hz multiplexing showed some slight flickering (error #7). The concept had to be changed to well above 100 Hz.
5. The DCF77 receiver module used showed two additional errors. Firstly, at 2.7 V operating voltage well within the specified range, switching the internal pull-up resistor of the portpin on lead to a voltage range exceeding the high-low transition voltage of the input pin (error #8). The internal pull-up of 47k already exceeds the capability of the receiver module to drive an input pin. The error was corrected by switching the pull-up off (do not use that when the DCF77 input pin is not attached, spurious signals here can block the software from doing other purposes!

   The second error is that this module is not producing clear DCF77 signals (error #9). It's HF amplification is so small that already a slight maladjustment of the antenna direction produces garbage at the input. And it is very prone to any small signals that are caught from switched power supplies in the near. And: I am only 80 km away from DCF77's antenna here in South Hesse. I would never again use that module for any other application.

Nine errors, of which eight were not caused by me, are a little too much to have fun.

2.1 The display part



This is one of the four display digits of the watch. It consists of 28 10-mm LEDs, that are connected four-by-four to yield seven segments. All anodes of the seven segments are connected together and are connected to one of the four anodes A1 to A4 on the 16-pin flat cable and female connector.

The cathodes a to g are interconnected with all four digits and are also connected to the flat cable and female connector.

Two LEDs in the middle form an additional cathode connection called h. The anode of the two can be attached to one of the four anode lines. To which of the anode lines those two are connected can be configured in the source code.

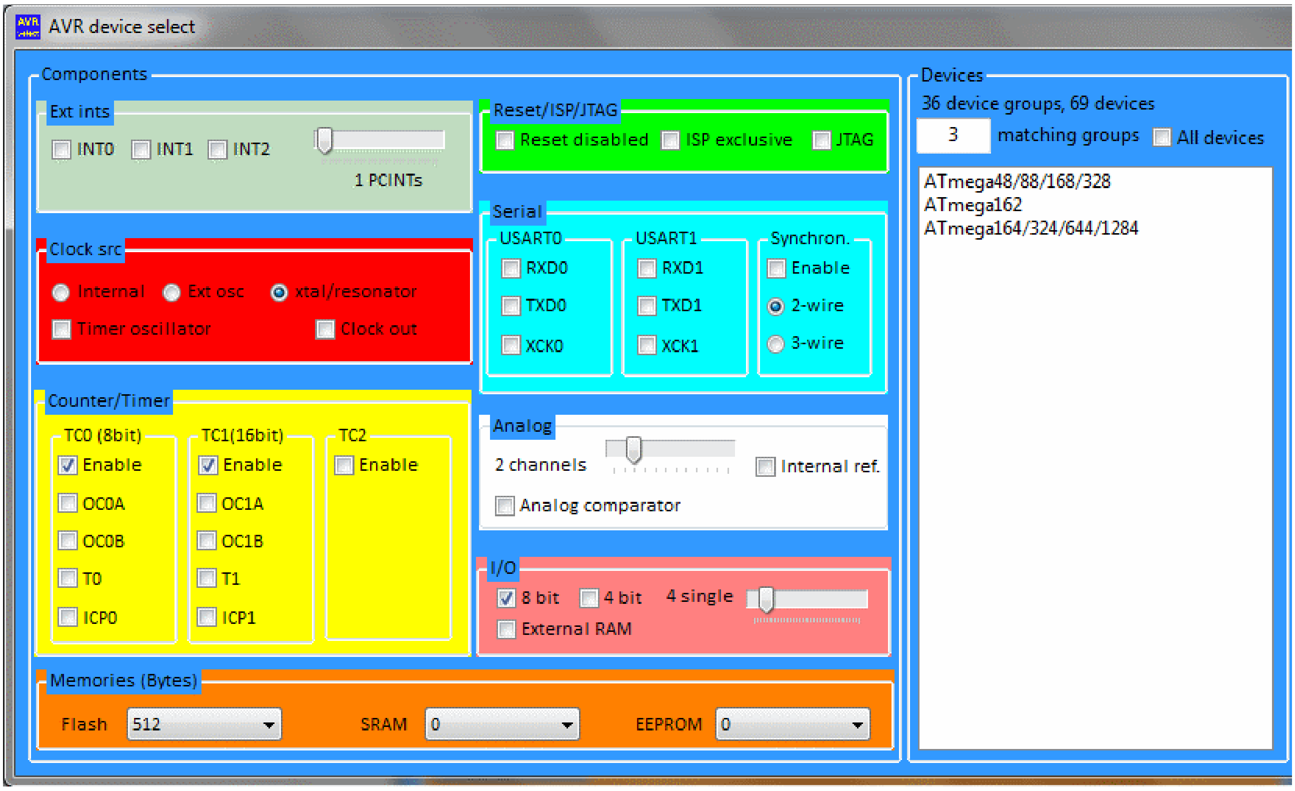
The 16-pin connection is documented in the schematic of the controller and driver.

2.2 Controller part

2.2.1 Selecting the AVR type

The AVR type results from the following hardware requirements:

1. The watch shall run without any DCF77 synchronization over a longer time without having to be re-adjusted. Therefore either the controller or the timer/counter TC2 have to use the crystal pins.
2. In order to not have to consult the DCF77 and the two key inputs these have to initiate a PCINT when their logical level changes. So at least one PCINT is required.
3. The eight constant current switches driving the cathodes shall be in one port, the four anode switches in another port to be easily accessible.
4. Measuring of the potentiometer and of the foto transistor voltage requires two ADC channels.
5. For multiplexing the display an 8-bit-timer, for measuring the duration of DCF77 signals a 16-bit-timer are required.



AVR device select

Components

Ext ints

□ INT0 □ INT1 □ INT2

1 PCINTs

Reset/ISP/JTAG

□ Reset disabled □ ISP exclusive □ JTAG

Clock src

○ Internal ○ Ext osc ● xtal/resonator

□ Timer oscillator □ Clock out

Serial

USART0: □ RXD0 □ TXD0 □ XCK0

USART1: □ RXD1 □ TXD1 □ XCK1

Synchron.: □ Enable ● 2-wire ○ 3-wire

Counter/Timer

TC0 (8bit): ☑ Enable □ OC0A □ OC0B □ T0 □ ICP0

TC1(16bit): ☑ Enable □ OC1A □ OC1B □ T1 □ ICP1

TC2: □ Enable

Analog

2 channels □ Internal ref.

□ Analog comparator

I/O

☑ 8 bit □ 4 bit 4 single

□ External RAM

Memories (Bytes)

Flash 512 SRAM 0 EEPROM 0

Devices

36 device groups, 69 devices

3 matching groups □ All devices

ATmega48/88/168/328
ATmega162
ATmega164/324/644/1284

The software here delivers a small selection of AVR type groups. The ATmega48 and its larger companions fulfill those requirements.

2.2.2 Selecting the clock frequency

Clocking of the ATmega48 is done with a crystal attached, by default of 4.096 MHz. Two ceramic capacitors of 22 pF assist in starting the internal oscillator. The external crystal fuse of the ATmega48 has to be activated to change from the internal RC oscillator to the external crystal (see the fuse section), because otherwise the watch is by a factor of 1,024 too slow (35 seconds per day).

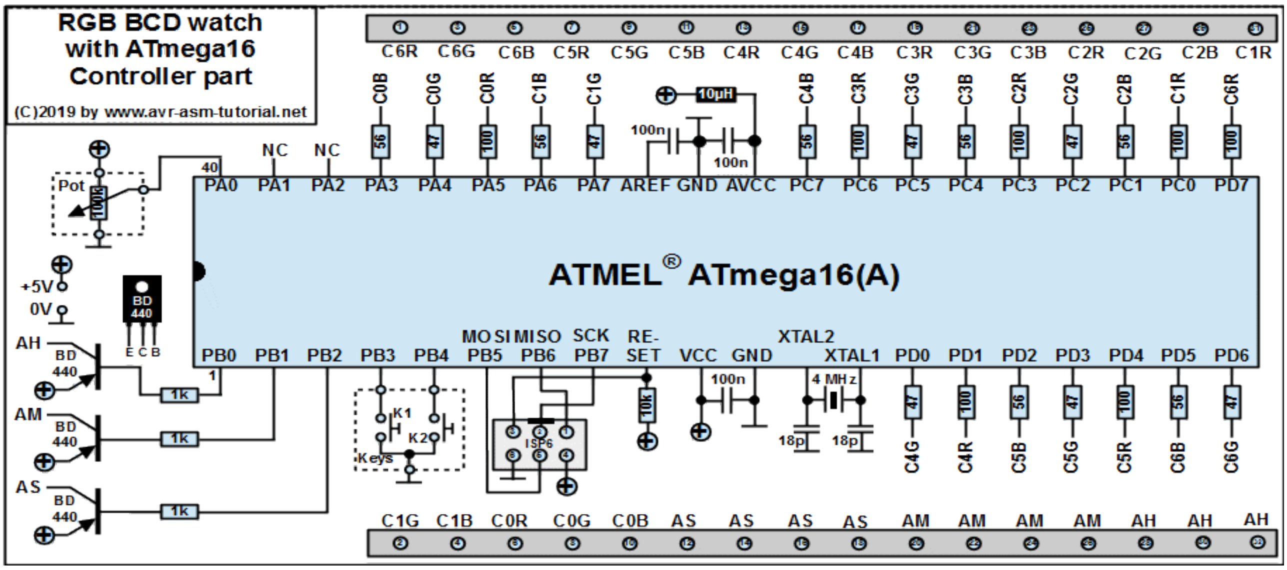
The reasons for selecting this xtal frequency are as follows (see the calculation sheet "Timer" in the OpenOffice spreadsheet here):

1. The xtal shall be available in electronic shops.
2. The xtal frequency shall be dividable by 8 or 64 and by 256 without a decimal remainder.
3. Division by 256 is required to allow the multiplex timing by an 8-bit timer and to allow preliminary switching the active period off by use of a compare register, that writes zeros to the anode driver outputs (dimming function switch-off values between 0 and 255). If you use a different xtal frequency that requires CTC mode to achieve a zero remainder (e.g. 2.0 or 4.0 MHz), has to move the switching function to compare B and has to convert the compare values to fit to the CTC value by use of multiplication.
4. If you select a prescaler value of 8, the smallest xtal frequency of 2.048 MHz yields much too high multiplex frequencies (1,000 Hz), which increases power consumption and HF noise. When selecting 64 as prescaler the MUX frequency would be too low (31.25 Hz), producing strong flickering of the display. Dividing the clock frequency by use of the CLKPR division avoids that.

4.096 MHz is a good selection, the MUX frequency is 125 Hz and optimized.

2.2.4 Schematic of the watch

This is the schematic of the controller and of all connected components. The spreadsheet "Partslist" in the OpenOffice document here lists all necessary components and their current prices.


RGB BCD watch
with ATmega16
Controller part
(C)2019 by www.avr-asm-tutorial.net
C6R C6G C6B C5R C5G C5B C4R C4G C4B C3R C3G C3B C2R C2G C2B C1R
C0B C0G C0R C1B C1G C4B C3R C3G C3B C2R C2G C2B C1R C6R
56 47 100 56 47 56 100 47 56 100 47 56 100 100
10µH
100n
100n
NC NC
40
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 AREF GND AVCC PC7 PC6 PC5 PC4 PC3 PC2 PC1 PC0 PD7
Pot
100k
+5V
0V
BD 440
E C B
ATMEL® ATmega16(A)
AH
AM
AS
MOSI MISO SCK RE-SET
XTAL2
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7 SET VCC GND XTAL1 PD0 PD1 PD2 PD3 PD4 PD5 PD6
1
1k
1k
1k
K1
K2
Keys
ISP6
10k
100n
4 MHz
18p 18p
47 100 56 47 100 56 47
C4G C4R C5B C5G C5R C6B C6G
C1G C1B C0R C0G C0B AS AS AS AS AM AM AM AM AH AH AH

2.2.5 Cathode drivers as constant current sources

Port D of the controller controls the eight cathodes of the display. The port pins drive, with resistors of 1k to limit port pin current in case that the display is not attached, the base of transistors BC547 (any NPN small signal can be used instead). The emitters are connected to 100Ω resistors to ground. That produces constant currents of

$$I\ (mA) = (2.7 - 0.6) / 100 * 1000 = 21\ mA$$

The port pin outputs are active high.

This is the reason for selecting 2.7 V as operating voltage of the controller. 5 V instead would have produced emitter voltages of 4.4 V and the voltages of the four LEDs (4 * 3.18 V = 12.72 V and at least 0.2 V CE(saturated) for the anode driver transistor BD440 would have resulted in 17.3 V. The consequence would have been that a transformer of 2*15V would have been required (which is not available with the necessary power rating). An alternative would have been to reduce the emitter voltage with two diodes down to 1.4 V. Because 16 of those diodes would have been necessary, the reduction of the operating voltage down to 2.7 V needed less components.

2.2.6 Anode driver

The four anodes are controlled by the lower half of port B. The three lower port bits are coupled directly to the bases of BC547 NPN transistors, those are driven via the internal pull-up resistors. The current from the pull-ups of 47k is enough to drive those transistors via a hFE(min) of 150:

$$I_C = (2{,}7V - 0{,}7V) / 47k\Omega * hFE_{min} * 1000 = 6{,}4\ mA$$

The NPN transistors, via resistors of 2k2, drive the base of the PNP anode driver transistors BD440 (any small power transistor can be used). Already a hFE of 30 drives the anode current of up to 200 mA.

Because the fourth anode bit is also needed for serving the ISP6 interface (MOSI), this NPN transistor is attached with a 10k resistor, and the port bit is configured as output.

2.2.7 ISP6 programming interface

The port pins PB3 to PB5 serve the ISP6 interface, by which the flash memory can be programmed within the running system. If no power supply is attached during programming the programming device has to supply 3.3 v. If the display is not connected, 5 V can also be used to program the flash. Don't attach the LEDs when at 5 V, that would ruin some of the LEDs.

2.2.8 Other peripheral components

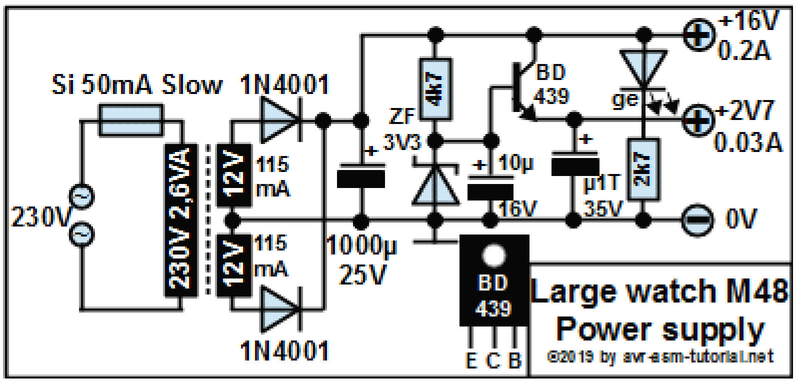
Port C is connected to the other peripherals:

1. The potentiometer is connected to the PC0/ADC0 input. The voltage there is converted to digital values, which in case the potentiometer is configured as dimming source dims the LEDs. If the foto transistor is used the potentiometer input is only used in case the time is re-adjusted.
2. On PC1/ADC1 a foto transistor for measuring background light is attached. This controls the brightness of the LEDs if so configured.
3. A small green LED is attached to PC2. This can be used for prototyping (see here for debugging options) or for any other use (not used by default).
4. On PC3 and PC4 resp. the following three pins of the 10-pin connector the two keys are attached. Those are held high with the internal pull-ups and active low when a key is closed (active low).
5. On PC5 a DCF77 receiver module can be attached. It is irrelevant if its signal is active high or low, both work correct. By default, its pull-up is active, but can be deactivated by a configuration setting. The module shall be able to work with 2.7 V operating voltage.

2.3 Power supply

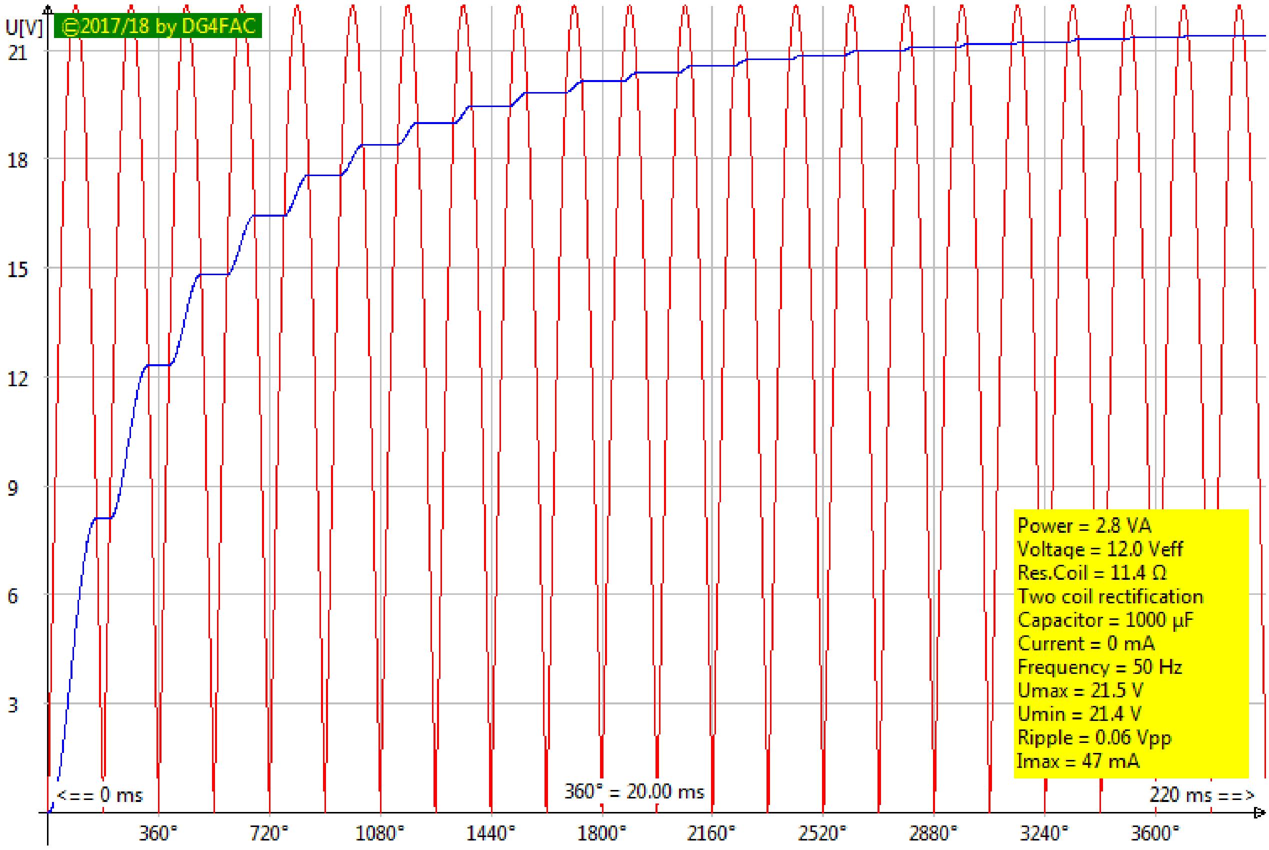
Th power supply has to have the following properties:

1. It has to supply 16 V and 0.2 A, required for the LEDs.
2. The controller needs 2.7 V with a maximum current of 30 mA. The regulator has to be compatible with the 25 V max voltage on the electrolytical capacitor (without LED load).



The schematic shows such a standard power supply with a 2*12V transformer and 2.8VA. The two diodes are 1N4001 or equivalent.

This simulates the displayed power supply without any load attached, see Power-Supply-Software here). The voltage on the cap stays well below 25 V. In practice the transformer had 2*15V instead and increased coil resistances. When no load was attached, the voltage rose above 25 V and had to be reduced by attaching a small yellow LED to below 25 V. Do not trust delivered products, those might be different from what is written on them.

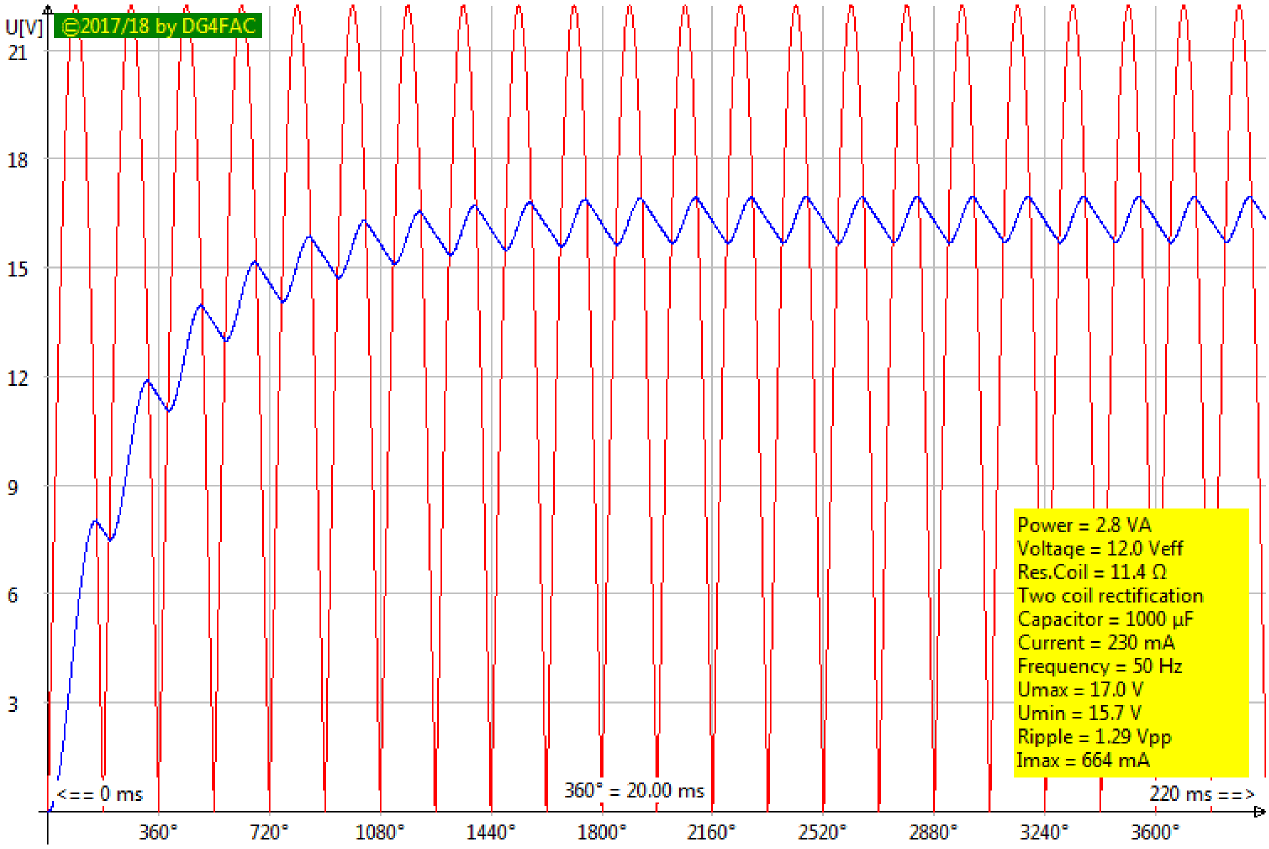


The elevated no-load voltage of the real transformer also had the effect that this voltage, when a 200 mA load was attached, broke completely down to 12.4 V only. By far too low to drive four LEDs. Because the 200 mA only occur if all eight segments are on (and only for a short mux cycle), it was sufficient to reduce the operating voltage of the controller down to 2.7 V and so to reduce the resulting segment current. If your transformer safely delivers 200 mA you can increase the operating voltage to 3.3 V, and to operate the watch with currents of up to 27 mA per segment.

Normally one uses an integrated regulator for the purpose of supplying the controller. But there are no integrated 2.7V regulators available that have a certified voltage. As the 3.3V regulator had 4.04V instead, I decided to construct my own. I used a 3.3V zener diode and an NPN to get rid of all problems.

This is the calculated voltage of the supply at 230 mA load, as designed. The 230 mA would be the maximum if 3.3V operating voltage, 27 mA LED segment current, eight segments and the maximum consumption of the controller would occur.

The voltage drops down to slightly below 16 V, with a ripple of 1.3 V, which is off-regulated by the constant current transistors.



So far the theory. The real transformer did not follow this calculation at all (see above).

Page top | Properties | Hardware | Mounting | Software

3 Mounting the device

3.1 LED front plate

The 28 10mm LEDs per displayed digit are placed on a width of 170 mm onto a 680-by-240mm plate of acrylic glass. Four LEDs each are interconnected to form one segment (see drawing above). Each digit has seven segments.

For drilling the 112 holes, the four DINA4 prints of the digits (see the OpenOffice document here) were cutted out and fixed on the acrylic plate. Drilling started with a 1.5mm drill, enlarging the holes with 1 or 1.5mm larger drills. Drilling was done with a low speed to avoid melting of the acrylic glass.

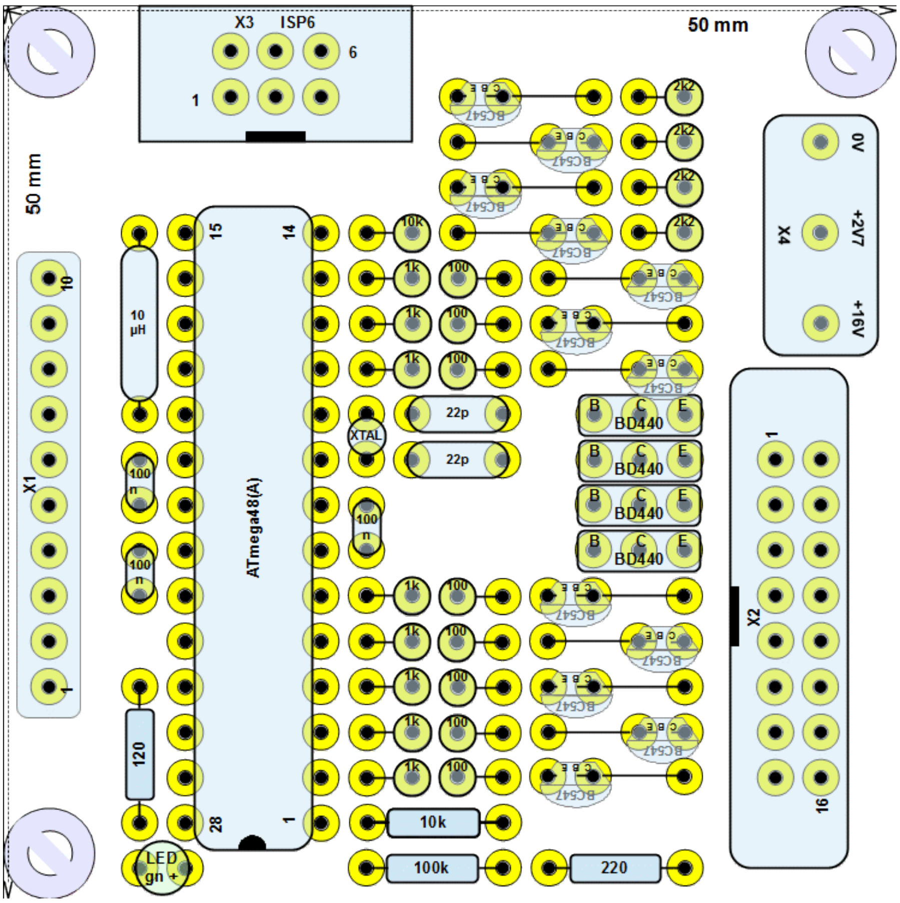
From 6mm on I used a trimming tool at fast speed, from the front and from behind.

When the holes were accessible with my drill press, a 10mm drill was used at high drill speed and slow sink speed. The melting of the acrylic glass helped to avoid cracking.

Those holes that were not accessible, were widened with 7, 8 and 9mm drills. The last mm was taken with a sharp knife to avoid cracking.

The pages 2 and 3 of the print also include the two LEDs in the middle. Those are drilled similarly, and wiring the anode goes with one of the four other digits. The placement of the anode has to be configured in the source code.

For protection and for upright standing the watch a second acrylic glass plate of the same size is mounted on the backside with six 50mm spacers. This plate also holds all PCBs and attached components.

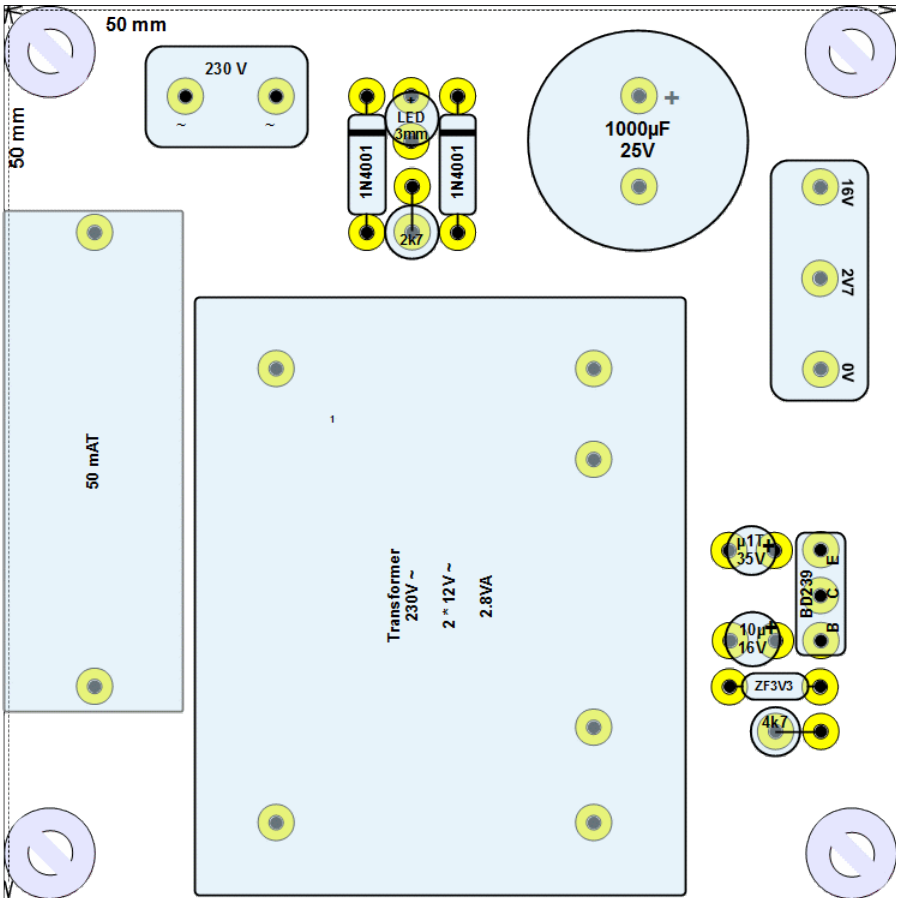


3.2 Controller part

These are the components of the controller part on a 50-by-50mm PCB. All external components are connected with plugs, so one can plug-off the controller part in the finally mounted device.

3.3 Power supply part

The power supply is also built on a 50-by-50mm PCB.



4 Software

4.1 Downloads

The source code in assembler can be downloaded here and can be viewed in the attachment.

The following documents are additionally available in OpenOffice format:

1. spreadsheets with all calculations (Timer, led currents, measured led voltages, etc., are in this document,
2. the eelectrical schematics of the controller and the power supply are in this document,
3. the layouts of the controller and the power supply PCB are in this document (no, I don't have made a printed layout design),
4. the four pages of the 7-segment layout of the front plate are in this document.

4.2 Assembling the source code

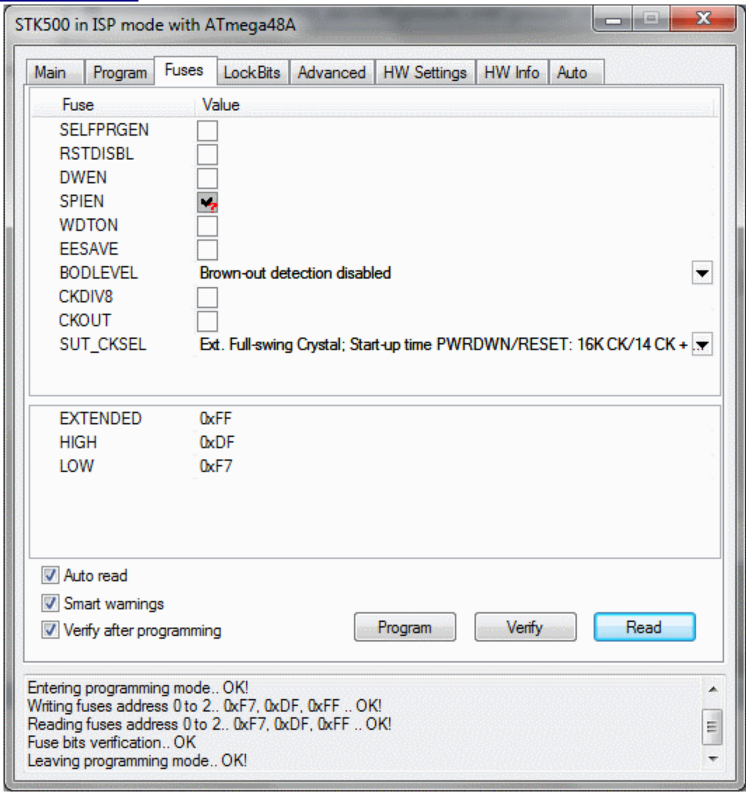
To assemble use the above link to the asm source to download the asm file. Please check, by opening the asm code with a simple editor (NOT a word processing program, please, this will destroy the simple text and add various formatting information, which makes it unreadable for the assembler), that no debugging switches are activated (see below).

For assembling you'll need an assembler that is capable to understand .IF directives. ATMEL's assembler 2 is able to understand. For those who do not want to download 900 MB monster software (the Studio) and don't want to install that, or for those who do not run one of the Windows operating systems, my own assembler gavrasm is simpler and better. A How-To for Windows here and for Linux here demonstrates how to do that. Those who run a different operating system (32 bit Win, Mac-OS, etc.) can download the source code of gavrasm and compile it with the Free Pascal compiler for that operating system.

The assembled machine code as Intel-Hex-File .hex should be found in the folder where the source code resides.

4.3 Flashing, fuses

The hex code has to be written to the flash memory of the ATmega48. For that you need a hardware burner and the software for that. Programming can be done via the ISP6 interface plug on the controller PCB.



STK500 in ISP mode with ATmega48A

Fuse	Value
SELFPRGEN	□
RSTDISBL	□
DWEN	□
SPIEN	☑
WDTON	□
EESAVE	□
BODLEVEL	Brown-out detection disabled
CKDIV8	□
CKOUT	□
SUT_CKSEL	Ext. Full-swing Crystal; Start-up time PWRDWN/RESET: 16K CK/14 CK +

EXTENDED	0xFF
HIGH	0xDF
LOW	0xF7

☑ Auto read
☑ Smart warnings
☑ Verify after programming

```
Entering programming mode.. OK!
Writing fuses address 0 to 2.. 0xF7, 0xDF, 0xFF .. OK!
Reading fuses address 0 to 2.. 0xF7, 0xDF, 0xFF .. OK!
Fuse bits verification.. OK
Leaving programming mode.. OK!
```

Prior or after flashing the fuses of the ATmega48 have to be changed to use the external crystal as clock source. Also clear the CLKDIV8 and the CKOUT fuses and switch off the brown-out-detection BOD. In practice, allowing brown-out-detection at 1.7V blocked all activities of the controller at 2.7V operating voltage, so do not use this useless feature.

By use of a debugging function in the source code you can check if the controller works correct (let the small green LED blink), see the debugging options here).

4.4 Hardware diagnosis

The source code includes several functions that can assist in getting the hardware to function correct. Those are activated by changing the "No"s to &quote;Yes&quote; in the source code, by re-assembling and burning the hex code to the ATmega48's flash. It does not make much sense to set more than one debug option at a time as those excude each other in most cases.

1. "Debug_ledgreen": Following power on the green LED should blink if the controller works correct.
   If the LED is off, the controller does not run correct and you'll have to search the wiring error in the clock section. If the blink rhythm is too fast, your controller is on an elevated clock frequency (check the crystal). If it runs too slow, also check your connections.
   If the green LED is permanently on: your ATmega48 is dead. That happened in my prototype, and I still do not know why (no, I did not reverse the operating voltage, and: yes: I tried High Voltage/ Parallel Programming in an STK500 to recover the chip - even that did not work). A matter for the dust-bin and final disposal in the controller-heaven ...
2. "Debug_current": This option activates all eight cathode drivers, so you can connect the cathode pins on X2 with an ampere meter to +16 or +20 Volt to measure the constant current. That should show roughly 21 mA. If the LED display is attached to X2 all seven segments of the ten-hour digit should be on.
3. "Debug_segments": That switches the seven (resp. eight) segments of all four digits on (one by one, from a to h). The speed can be varied with the constant "cDebug_segDelay". With 1 the change goes very fast, with 10 it is slower.
4. "Debug_mux8" This simulates a multiplex cycle, with all seven resp. eight segments on and switches from digit to digit. The mux frequency can be varied with changing the constant "cDebug_muxfreq". At 62 Hz nearly no flickering can be seen, beyond 100 Hz the muxing cannot be seen any more.
5. "Debug_adc": The measuring results from the ADC, the MSB of the sum (between 0 and 255), is displayed in decimal format on the LEDs.
6. "Debug_keys": The input signals on the two key input pins are displayed on the ten-hour (key 1) and on the one-hour (key 2) position. Displayed is a small i (segment c of the display) for a high input pin or a small o for a low input pin.
7. "Debug_dcferr": All error signals on the DCF77 input are displayed. The ten-hours digit shows a large E, the one-hours digit the error number. The following error numbers mean:
   0. or blank: No error
   1. Signal shorter than required for a received 0
   2. Signal shorter than required for a received 1
   3. Signal shorter than required for a pause
   4. Signal shorter than required for a missing 60th second
   5. Signal longer than required for a missing 60th second
   6. Following minute change not 59 bits received
   7. Minute parity uneven
   8. Hour parity uneven
   9. No signal on the DCF77 input pin

   Correct signal durations overwrite the error number with a blank, so that only error

numbers can be seen.
If diverse different error numbers occur, the tolerance parameter in the constant "cDcfTol" can be increased.

8. "Debug_dcfany": All correct signals are displayed (from right to left): received zeros are displayed as small o, ones with a small i (segment c), pauses with a capital P.
9. "Debug_dcfdur": All signal durations are displayed in hexadecimal format on the display (with the hexadecimal digits 0 to 9 and A, b, C, d, E, F.).

4.5 Constants to be adjusted in the software

Within the section "Adjustable constants" of the source code diverse adjustments can be made that change properties of the watch. All constants in this section can be changed without risking malfunctions.

1. **.equ xtal = 4096000 ; in Hz**
   If a different crystal is used, you can adjust its frequency here. The frequency has to be dividable by 4096 or 8192 without a decimal remainder, otherwise clocking of the watch will be inaccurate. Use the OpenOffice spreadsheet **Timer** in the document here to perform those calculations.
2. **.equ cClkpr = 4 ; Either two or four**
   With that the clock prescaler is re-adjusted. It can be either 2, 4 or 8. Use the spreadsheet to study the effect.
3. **.equ cStartHours = 0x09 ; Start at 20:00 h**
   .equ cStartMinutes = 0x59
   This adjusts the time that the watch starts with during power-up. Values have to be in packed BCD format.
4. **.equ cDcfOnly = Yes ; Display/Clear unsynced time**
   Switches the leds off, as long as no DCF synchronization took place yet.
5. **.equ tDcf0 = 100 ; 100 ms for a 0**
   .equ tDcf1 = 200 ; 200 ms for a 1
   .equ tDcfP = 850 ; Pause for 0 and 1 to next second
   .equ tDcfM = 1850 ; Pause for 59th second pulse
   .equ tDcfT = 3000 ; Time-out of DCF signal
   These parameters adjust the duration of DCF77 signals in Milliseconds. If your module has different durations, you can adjust those in the personalized section.
6. **.equ cDcfTol = 10 ; Tolerance in %**
   The unavoidable tolerances of the durations of DCF77 signals can be adjusted here. If your signal durations have a larger variance, increase that constant to 15 or 20. If overlapping occurs, you'll be notified by an error message.
7. **.equ cAnDp = 3 ; Should be between 1 and 4**
   This adjusts to which anode the double point in the middle is connected. If false, the blinking does not occur.
8. **.equ tBounce = 50 ; Bouncing time, in ms**
   This adjusts the time that the keys have to be inactive until negative pulses on those inputs are leading to an active effect. If your keys bounce longer than this, increase that value.
9. **.equ cDimOpto = No ; Select the source**

   This parameter selects whether the foto transistor (Yes) or the potentiometer (No) adjusts the dimming of the display.

Any changes come into effect following re-assembling and the transfer of the hex code to the flash.

4.6 How the software works

The following chapters elaborate on the basic functioning of the software.

4.6.1 Timing control

The whole timing control is made with the timer/counter TC0. The following relationships play a role:

1. Frequency of the oscillator with external crystal = 4,096 MHz,
2. Clock prescaler by 4 with CLKPR, controller clock = 1,024 MHz,
3. TC0 prescaler by 8, TC0 clock = 128 kHz,
4. Fast PWM mode mit TOP=255, TC0-Overflow-Int = 500 Hz.

Within the overflow interrupt the following tasks are performed:

1. output of the next left digit of the watch and activation of the next left anode driver, readjustment for the next mux stage,
2. down counting of the half-second divider by one, if zero: setting the half-second flag and down counting of the minute divider (from 120 down to zero), if that reaches zero: setting the minute flag,
3. if the toggle counter of the keys is not at zero: if one or both keys are pressed restart the toggle period, if none is pressed down counting of the toggle counter.

Additionally TC0 is used to dim the display. This is done when the compare value A is reached and, in its interrupt service routine, writes zero to the anode driver output port.

Because the compare value A changes rather often (every time the ADC reaches 64 conversions) it had to be taken care to not miss compare matches due to setting a smaller value while already at a higher count. Therefore the TC0 works in Fast PWM mode and not as normal counter. In PWM mode the update of the compare value is delayed until the current PWM cycle has ended. This works fine, but not with extremely small compare A values of less than 2. In that case flickering of single digits occurs, and I don't understand why. As this only occurs at extremely low dimming, I can live with that.

4.6.2 AD conversion as additional clock source

The AD converter works with a clock prescaler of 128 and converts either the analog voltage on the potentiometer (on ADC0) or on the collector of the foto transistor (ADC1). If time setting is active (key 1 has been pressed and time setting is still active), in any case ADC0 is measured.

Within the interrupt service routine of the ADc the results of 64 consecutive measurements are summed up and the conversion is restarted. If those 64 measurements are completed, restart is omitted, the MSB of this sum is copied and the bAdc flag is set. Further processing is performed outside the service routine.

Outside the service routine the flag is cleared, the channel is set depending from the current mode, the sum is cleared, the counter restarts at 64 and the first conversion is started. If time-setting mode is inactive, the MSB of the sum is either directly written to the compare A port of TC0 (potentiometer dimming) or is inversed (foto transistor dimming) and then written to compare A.

If time-setting is active then see chapter 4.6.4 below.

The following frequency scheme of the AD conversion applies:

1. Controller clock: 1,024 MHz
2. AD clock prescaler: 128
3. Clocks per conversion: 13

4. Number of conversions summed up: 64
5. Conversion frequency: 9.62 Hz
6. Conversion time: 104 ms

4.6.3 Multiplexing

The four bytes with the cathodes to be activated for the four digits are located in SRAM. Those are output with a frequency of 500 Hz per digit during the TC0 overflow int. The output direction is reversed (from A4 to A1) because the end of the cycle can be detected in a simpler way (carry flag set after right-shifting the anode register).

Before the next byte combination is brought to the cathode output port, the anodes are switched off. This was necessary to avoid flickering of the display.

4.6.4 Adjusting the time with the keys and the potentiometer

Adjusting the time starts when key a is closed for the first time. This is recognized via a PCINT, where changes on the key input pins and on the DCF77 signal input pin are masked to lead to an interrupt. The flag bKey1 is only activated if the input pin is low and if the toggle counter is at zero.

The flag bKey1 activates the bKeyA flag that signals an active time setting phase. The ADC is set to measure potentiometer voltages on ADC0. Incoming ADC result sums, if complete, now are multiplied by 24, the result is converted from binary to packed BCD format and the two digits are converted to 7-segment and written to the SRAM storage for display.

If the half-seconds divider is below 25% of its time, the two digits are blanked instead. That results in a blinking of the two hour digits.

If the hours are set, another key pressing of key 1 results in setting the bKeym flag additionally. From now on multiplication is done with 60 and the 7-segment result is written to the minute position in SRAM.

The third pressing of key 1 converts the hours and minutes that were adjusted to set the hours and minutes of the watch. Additionally the half-seconds and the minute dividers are restarted and eventual flags, that have been set in the meantime, are cleared. The bKeyM and bKeyA flags are also cleared and the current time is displayed.

If the key 2 is pressed, while bKeyA is active, the setting of minutes returns to setting the hours. If currently setting the hours is active while key 2 is pressed, the time-setting mode is skipped and the original time (which continued to run during time-setting mode, but was not displayed) is displayed.

If the time setting needs longer than the selected period of 10 minutes, the mode is also cleared and returns back to the original time.

Each key pressing event restarts the toggle counter to prevent from any reactions to spurious signals from the key inputs.

4.6.5 Adjusting the time with DCF77 signals

Each level change on the DCF77 input pin leads to a PCINT and a respective flag is set.

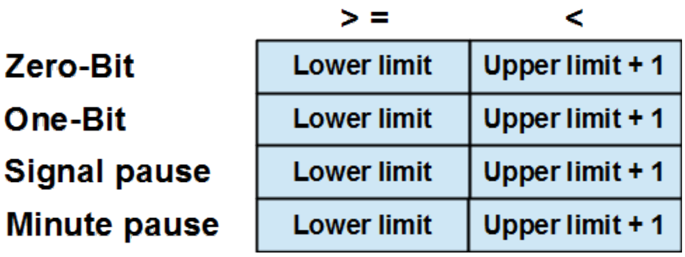
The flag leads to reading the current count from the 16-bit timer TC1. This timer advances with a prescaler value of 1024 at 1 kHz (1 ms per count). Each input signal restarts the TC1, so that each signal's duration can be assessed to find out what time information has been received from DCF77.

A zero bit of DCF77 should be around 100 ms long. If the tolerance is set to 10% it can be between 90 and 110 ms long. With the TC1 clock the count shall be in that range for a zero.

Each of the received signals has its specific range. A minute change has a duration of between 1800 and 1900 ms, depending if the last bit was a one or a zero. With 10% tolerance the counter value should be between 1850 - 185 = 1665 and 1850 + 185 = 2015. As each level change causes an interrupt, the normal pause between the end of a bit and the beginning of the next second's pulse should be between 800 and 900 ms long, but does not require any further action.

From that the following algorithm results:

1. If the signal duration is shorter than the table's first value, an error has occurred.
2. If the signal duration is longer or equal the first value and shorter than the second value, a zero, a one, a pause or a minute change is recognized and has to be handled.
3. If the signal duration is equal or above the second value, the next value pair has to be tried. If the last duration, a minute change, is exceeded an error happened.



	> =	<
Zero-Bit	Lower limit	Upper limit + 1
One-Bit	Lower limit	Upper limit + 1
Signal pause	Lower limit	Upper limit + 1
Minute pause	Lower limit	Upper limit + 1

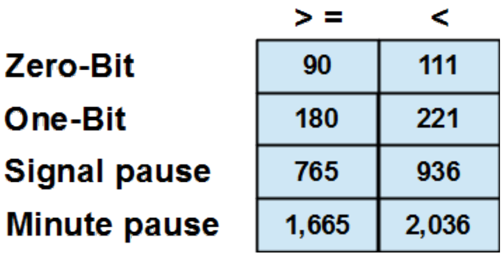
Received zeros and ones are shifted with ROR into a bit buffer that is at least 40 bits long. When shifted those signals have to be counted (to check if exactly 59 have been received when the minute change signal occurs).

Second pauses are ignored.

If a minute change occurs,

1. it is checked whether exactly 59 data bits have been received,
2. the minutes are to be extracted from the bit stream, the respective parity has to be checked (must be even) and has to be written to the minute storage,
3. the hours are extracted, their parity bit checked and written to the hour storage,
4. if both were correct: the time is set, the half-second and minute divider restarted and pending flags are cleared.

These are the values of the table by default with a tolerance of 10%. No overlapping occurs. As each TC1 count is associated with 1,024 controller instructions, short delays in reading counter values are irrelevant.

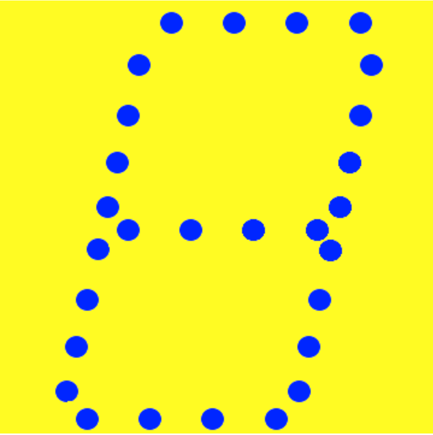


	> =	<
Zero-Bit	90	111
One-Bit	180	221
Signal pause	765	936
Minute pause	1,665	2,036

Page top | Properties | Hardware | Mounting | Software

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

Source code



AVR applications

Large watch with ATmega48
Assembler software for the large watch

Assembler source code for the large ATmega48 watch

The original assembler source code is here.

```
;
; *********************************
; * Large watch with ATmega48     *
; * (C)2019 avr-asm-tutorial.net  *
; *********************************
;
.nolist
.include "m48adef.inc" ; Define device ATmega48A
.list
;
; **********************************
;    D E B U G   S W I T C H E S
; **********************************
;
; Switches debug options on/off
;   Make sure that all switches are off in the final version
;
.equ Yes = 1 ; For debug switches
.equ No = 0
;
; Debug the green led
.equ Debug_ledgreen = No ; Debug the green led only
;
; Debug the leds on start-up
.equ Debug_leds = No ; Debug the leds on start-up
;
; Debug the currents and blink the green led
.equ Debug_current = No ; Switch the current drivers on
;
; Debug the segments of the display
.equ Debug_segments = No ; Switch the segments on
  ; 1 = 9 seconds for all four digits
  ; 10 = slow, 80 seconds for all four digits
  ; 100 = extremely slow, 25 seconds per digit
  .equ cDebug_segDelay = 100 ; Delay the active segment
```

```
;
; Debug the muxing
.equ Debug_mux8 = No ; Multiplex the four displays
  ; The mux frequency for all four digits once
  ; Must be between 4 and 10,000 Hz
  .equ cDebug_muxfreq = 150 ; MUX frequency in Hz
;
; Debug the ADC results
.equ Debug_adc = No ; Displays the ADC results
;
; Debug the two keys
.equ Debug_keys = No ; Displays the key status
;
; Debug errors in DCF signal reception
;   Displays E and error number instead of hours
.equ Debug_dcferr = No ; Yes/No
.equ Debug_dcfany = No ; Displays any signals
.equ Debug_dcfdur = No ; Displays signal durations
;
; **********************************
;        H A R D W A R E
; **********************************
;
; Device: ATmega48A, Package: 28-pin-PDIP-S
;
;           __________
;        1 /          |28
;  Res o--|RESET   PC5|--o DCF77-In
;    a o--|PD0     PC4|--o Key1-In
;    b o--|PD1     PC3|--o Key2-In
;    c o--|PD2     PC2|--o Led green cathode
;    d o--|PD3     PC1|--o Optosensor
;    e o--|PD4     PC0|--o Pot
;  +5V o--|VCC     GND|--o 0V
;   0V o--|GND    AREF|--o 100nF
;   X1 o--|PB6    AVCC|--o +5V
;   X2 o--|PB7     PB5|--o SCK
;    f o--|PD5     PB4|--o MISO
;    g o--|PD6     PB3|--o MOSI+A4
;    h o--|PD7     PB2|--o A3
;   A1 o--|PB0     PB1|--o A2
;       14|___________|15
;

; **********************************
;  P O R T S   A N D   P I N S
; **********************************
;
.equ p7SegO = PORTD ; Seven-segment output port
.equ p7SegD = DDRD ; Seven-segment direction port
.equ pAnodeO = PORTB ; Anode driver output port
.equ pAnodeD = DDRB ; Anode driver direction port
.equ pLedGO = PORTC ; Green led output port
.equ pLedGD = DDRC ; Green led direction port
.equ pLedGI = PINC ; Green led input port
.equ bLedGO = PORTC2 ; Green led portbit output
.equ pDcfKeyO = PORTC ; DCF and key port output port
.equ pDcfKeyD = DDRC ; DCF and key direction port
.equ pDcfKeyI = PINC ; DCF&Key input port
.equ bDcfI = PINC5 ; DCF77 input pin
.equ bKey1I = PINC4 ; Key1 input portpin
```

```
.equ bKey2I = PINC3 ; Key2 input portpin
;
; **********************************
;   A D J U S T A B L E   C O N S T
; **********************************
;
; Frequency of the external xtal
;   has to be dividable by 2/4, 8 and 256 (=4096/8192)
;   without any fractional remainder
.equ xtal = 4096000 ; in Hz
;
; Clock prescaler applied
.equ cClkDiv = 4 ; Either one, two, four or eight
;
; Start time setting of the clock
;   Packed BCD format
.equ cStartHours = 0x20 ; Start at 20:00 h
.equ cStartMinutes = 0x00
;
; DCF time only (clear display as long as not synced with DCF77)
.equ cDcfOnly = No ; Display/Clear unsynced time
;
; DCF signal durations in ms
.equ dcfdur_personally = No ; No = set default durations
;
.if dcfdur_personally != Yes
  .equ tDcf0 = 100 ; 100 ms for a 0
  .equ tDcf1 = 200 ; 200 ms for a 1
  .equ tDcfP = 850 ; Pause for 0 and 1 to next second
  .equ tDcfM = 1850 ; Pause for 59th second pulse
  .equ tDcfT = 3000 ; Time-out of DCF signal
   ; DCF77 signal duration tolerance
  .equ cDcfTol = 15; Tolerance in %
  .else
  .equ tDcf0 = 80 ; for too short signals
  .equ tDcf1 = 190 ; 200 ms for a 1
  .equ tDcfP = 850 ; Pause for 0 and 1 to next second
  .equ tDcfM = 1850 ; Shorter period for 59th second pulse
  .equ tDcfT = 3000 ; Time-out of DCF signal
  ; DCF77 signal duration tolerance
  .equ cDcfTol = 25; Tolerance in %
  .endif
;
;
; DCF77 signal input pull-up resistor
.equ cDcfPullUp = No ; Yes or no
;
; The anode line to which the double point in the
;   middle is attached to
.equ cAnDp = 3 ; Should be between 1 and 4
;
; Key bouncing period
.equ tBounce = 50 ; Bouncing time, in ms
;
; Skip input mode after inactive time
.equ cSkipInpMinutes = 10 ; Minutes until input has to be finished
;
; Select the dimming source
;   0: Selects the potentiometer
;   1: Selects the opto sensor
.equ cDimOpto = No ; Select the source
```

```
;
; *********************************
;  F I X  &  D E R I V.  C O N S T
; *********************************
;
; Clock divider conversion
.if cClkDiv == 1
  .equ cClkPr = 0
  .else
  .if cClkDiv == 2
    .equ cClkPr = 1
    .else
    .if cClkDiv == 4
      .equ cClkPr = 2
      .else
      .if cClkDiv == 8
        .equ cClkPr = 3
        .else
        .error "cClkDiv has illegal value!"
        .endif
      .endif
    .endif
  .endif
;
; Clock frequency
.equ clock=xtal/cClkDiv ; Define clock frequency
;
; Half seconds and minute dividers
.equ cTc0Prsc = 8 ; TC0 prescaler
.equ cTc0Frq = clock / cTc0Prsc / 256 ; TC0 int frequency
.equ cSec2 = (cTc0Frq+1) / 2 ; Half second counter
.equ c75pcon = cSec2 / 4 ; Period over which the
   ; selected digits are displayed when input is
   ; active
   .if c75pcon>255
     .error "Off period too long, reduce c75pcon!"
         .endif
;
; DCF signal counts, with rounding
.equ cTc1Tick = (clock+512) / 1024 ; Timer TC1 tick in Hz, @4.096 MHz = 4000 Hz
.equ cDcf0Min = ((tDcf0-tDcf0*cDcfTol/100)*cTc1Tick+500)/1000 ; Count 0 minimum
.equ cDcf0Max = ((tDcf0+tDcf0*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count 0
maximum
.equ cDcf1Min = ((tDcf1-tDcf1*cDcfTol/100)*cTc1Tick+500)/1000 ; Count 1 minimum
.equ cDcf1Max = ((tDcf1+tDcf1*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count 1
minimum
.equ cDcfPMin = ((tDcfP-tDcfP*cDcfTol/100)*cTc1Tick+500)/1000 ; Count pause
minimum
.equ cDcfPMax = ((tDcfP+tDcfP*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count pause
maximum
.equ cDcfMMin = ((tDcfM-tDcfM*cDcfTol/100)*cTc1Tick+500)/1000 ; Count minute
minimum
.equ cDcfMMax = ((tDcfM+tDcfM*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count minute
maximum
.equ cDcfT = (tDcfT*cTc1Tick+500)/1000+1 ; Counter time out
.if cDcfT>65535
  .error "Clock frequency too high for DCF duration counting"
  .endif
.if (cDcf0Max>=cDcf1Min)||(cDcf1Max>=cDcfPMin)||(cDcfPMax>=cDcfMMin)
  .error "Overlapping DCF duration(s), reduce cDcfTol!"
  .endif
```

```
;
.if cDcfPullUp == Yes
 .equ mDcfKeyO = (1<<PORTC5)|(1<<PORTC4)|(1<<PORTC3)
 .else
 .equ mDcfKeyO = (1<<PORTC4)|(1<<PORTC3)
 .endif
;
; Key bouncing constant
.equ cTc0Presc = 64 ; TC0 prescaler value
.equ cTc0Mux = clock / cTc0Presc / 256 ; MUX interrupt frequency
.equ cBounce = (tBounce*cTc0Mux+500)/1000 ; Bounce constant
;
; **********************************
;             C L O C K S
; **********************************
;
; Default xtal frequency 4.096 MHz, CLKPR=4, effective clock=1.024 MHz
;
; TC0:
;   Clocked with a prescaler of 8, clock tick = 128 kHz @ 1.024 MHz
;   Fast PWM counting, TOP = 0xFF, overflow int = 500 Hz (2 ms) @ 1.024 MHz
;     MUX-frequency = 500 / 4 = 125 Hz
;   16-Bit-Register downcount from 1,000, yields 2 Hz signal for blinking the
double dot
;     If zero: Set bSec flag, register downcount from 120 to yield minute for
clock,
;       If zero: Set bMin flag
;   If rBounce not at zero:
;     Both key inputs high: downcount rBounce, otherwise rBounce = cBounce
;   Timer interrupt on overflow
;   Compare match A: OCR0A interrupt (clears anode driver for dimming the LEDs)
; TC1:
;   Counts the duration of DCF77 signals
;     Clocked with a prescaler of 1,024 = 1 kHz (1.0 ms) @ 1.024 MHz
;   Normal counting (cleared by PCINT on DCF signal input)
;   Timer interrupt on compare match A: sets the bDcfTO flag
; ADC:
;   Converts measurements on ADC0/ADC1 inputs
;   Clock prescaler = 128
;   N clock cycles per conversion = 13
;   Conversion frequency = 615.38 Hz (1.625 ms) @ 1.024 MHz
;   Sums up 64 ADC results = 104 ms @ 1.024 MHz
;   If 64 adders complete: Set bAdc flag
;     ca. 1 MUX cycle per update
;     As the TC0 compare match update is done at the beginning
;     of the next mux event (each 2 ms), no missing compare matches
;     occur (no flickering)
;
; **********************************
;       R E G I S T E R S
; **********************************
;
; used: R1:R0 for DCF signal duration and
      ; for hardw multiplication
.def rAdcCtr = R2 ; ADC sum counter
.def rAdcSumL = R3 ; ADC result sum, LSB
.def rAdcSumH = R4 ; dtp., MSB
.def rAdc = R5 ; ADC result MSB
.def rInput = R6 ; Input pins
.def rDcfBits = R7 ; Counter for DCF bits
.def rDcf3 = R8 ; DCF bits, byte 4
```

```
.def rDcf4 = R9 ; DCF bits, byte 5
.def rDcf5 = R10 ; DCF bits, byte 6
.def rDcf6 = R11 ; DCF bits, byte 7
.def rDcf7 = R12 ; DCF bits, byte 8
.def rDcfErr = R13 ; DCF77 signal error
.def rMux = R14 ; Mux channel
.def rSreg = R15 ; Save status register
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose inside ints
.def rFlag = R18 ; Flag register
  .equ bMin = 0 ; A minute is over
  .equ bSec2 = 1 ; A half second is over
  .equ bDcf = 2 ; Level change on DCF input
  .equ bDcfTO = 3 ; Time out Dcf input signal
  .equ bKey1 = 4 ; Key1 pressed
  .equ bKey2 = 5 ; Key2 pressed
  .equ bKeyA = 6 ; Key input active
  .equ bKeyM = 7 ; Minute key input active
.def rFlag2 = R19 ; Second flag register
  .equ bAdc = 0 ; An ADC result is available
.def rHours = R20 ; Hours time, packed BCD
.def rMinutes = R21 ; Minutes time, packed BCD
.def rBounce = R22 ; Debouncing key counter
.def rMin = R23 ; Minute counter
.def rSec2L = R24 ; 1/2 seconds counter, LSB
.def rSec2H = R25 ; dto., MSB
; used: R27:R26 = X as pointer
; used: R29:R28 = Y for MUX
; used: R31:R30 = Z for multiple purposes outside int
;
; **********************************
;           S R A M
; **********************************
;
.dseg
.org SRAM_START
sMux:
.byte 4 ; 4 bytes for muxing the display
sMuxEnd:
;
sInpTime:
.byte 2 ; 2 bytes input time buffer
;
sSkipInp:
.byte 1 ; Skip input after inactive time
;
; If debug any DCF signals
sDcfPos:
.byte 1 ; Position of the next display (+1)
;
; **********************************
;         C O D E
; **********************************
;
.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
        reti ; PCI0
        rjmp Pci1Isr ; PCI1 for DCF and key input changes
        reti ; PCI2
        reti ; WDT
        reti ; OC2A
        reti ; OC2B
        reti ; OVF2
        reti ; ICP1
        rjmp Tc1CmpAIsr ; OC1A, Dcf77 time-out
        reti ; OC1B
        reti ; OVF1
        rjmp Tc0CmpAIsr ; OC0A: clear anode driver
        reti ; OC0B
        rjmp Tc0OvfIsr ; MUX and time
        reti ; SPI
        reti ; URXC
        reti ; UDRE
        reti ; UTXC
        rjmp AdcIsr ; ADCC, Conversion complete
        reti ; ERDY
        reti ; ACI
        reti ; TWI
        reti ; SPMR
;
; **********************************
;  I N T - S E R V I C E   R O U T .
; **********************************
;
; PCI1 Interrupt service routine
Pci1Isr:
  in rSreg,SREG ; Save SREG
  in rimp,pDcfKeyI ; Read DCF signal and keys
  eor rimp,rInput ; Compare with last input
  sbrc rimp,bDcfI ; DCF77 bit set?
  sbr rFlag,1<<bDcf ; Set DCF flag
Pci1Isr1:
  tst rBounce ; Check bouncing counter
  brne Pci1Isr3 ; Still bouncing, ignore
  ; rInput key bit was 1/0, now is 0/1: EOR bit is one
  sbrs rimp,bKey1I ; Key 1 input changed?
  rjmp Pci1Isr2 ; No, skip
  sbis pDcfKeyI,bKey1I ; Input is one?
  sbr rFlag,1<<bKey1 ; Set key 1 flag
Pci1Isr2:
  sbrs rimp,bKey2I ; Key 2 input changed?
  rjmp Pci1Isr3 ; No, skip
  sbis pDcfKeyI,bKey2I ; Input is one?
  sbr rFlag,1<<bKey2 ; Set key 2 flag
Pci1Isr3:
  in rInput,pDcfKeyI ; Read DCF signal and keys again
  out SREG,rSreg ; Restore SREG
  reti
;
; TC0 overflow interrupt service routine
Tc0OvfIsr:
  in rSreg,SREG ; Save SREG
  clr rimp ; Anodes off
  out pAnodeO,rimp
  ld rimp,-Y ; Read next mux byte
  out p7SegO,rimp ; Cathodes out
  out pAnodeO,rMux ; Set anodes
  lsr rMux ; Next lower anode
  brcc Tc0OvfIsr1 ; Not at the end
  ldi rimp,0b00001000 ; Start with anode 4
  mov rMux,rimp ; in rMux
  ldi YH,High(sMuxEnd) ; Restart from the end
  ldi YL,Low(sMuxEnd)
Tc0OvfIsr1:
  tst rBounce ; Check if bouncing active
  breq Tc0OvfIsr4 ; No, skip
  sbis pDcfKeyI,bKey1I ; Key 1 not pressed?
  rjmp Tc0OvfIsr2 ; No, restart bouncing
  sbic pDcfKeyI,bKey2I ; Key 2 pressed?
  rjmp Tc0OvfIsr3 ; No, decrease bounce count
Tc0OvfIsr2:
  ldi rBounce,cBounce ; Restart bouncing
  rjmp Tc0OvfIsr4 ; Continue ISR
Tc0OvfIsr3:
  dec rBounce ; Down count rBounce
Tc0OvfIsr4:
  sbiw rSec2L,1 ; Decrease seconds divider
  brne Tc0OvfIsr5 ; Not zero, skip
  ldi rSec2H,High(cSec2) ; Restart seconds divider
  ldi rSec2L,Low(cSec2)
  sbr rFlag,1<<bSec2 ; Set half second flag
  dec rMin ; Decrease minute divider
  brne Tc0OvfIsr5 ; Not zero, skip
  ldi rMin,120 ; Restart minute counter divider
  sbr rFlag,1<<bMin ; Set minute flag
Tc0OvfIsr5:
  out SREG,rSreg ; Restore SREG
  reti
;
Tc1CmpAIsr:
  in rSreg,SREG ; Save SREG
  sbr rFlag,1<<bDcfTO ; Set DCF time-out flag
  out SREG,rSreg ; Restore SREG
  reti
;
; TC0 Compare A Interrupt service routine
Tc0CmpAIsr:
  ldi rimp,0 ; Clear anode driver
  out pAnodeO,rimp ; in anode port
  reti
;
; ADC conversion complete interrupt service routine
AdcIsr:
  in rSreg,SREG ; Save SREG
  lds rimp,ADCL ; Read LSB result
  add rAdcSumL,rimp ; Add to LSB sum
  lds rimp,ADCH ; Read MSB
  adc rAdcSumH,rimp ; Add this and carry to MSB sum
  dec rAdcCtr ; Decrease counter
  brne AdcIsr1 ; Not zero
  sbr rFlag2,1<<bAdc ; Set flag
  out SREG,rSreg ; Restore SREG
  reti
AdcIsr1:
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  sts ADCSRA,rimp ; Restart ADC
  out SREG,rSreg ; Restore SREG
  reti
;
; **********************************
;  M A I N   P R O G R A M   I N I T
; **********************************
;
Main:
  ; Init stack
        ldi rmp,High(RAMEND)
        out SPH,rmp ; Init MSB stack pointer
        ldi rmp,Low(RAMEND)
        out SPL,rmp ; Init LSB stack pointer
  ; Init clock prescaler
  ldi rmp,1<<CLKPCE ; Activate clock prescaler
  sts CLKPR,rmp ; in clock prescaler port
  ldi rmp,cClkPr ; Set new prescaler
  sts CLKPR,rmp ; in clock prescaler port
;
; ******************************
;  H A R D W A R E   D E B U G
; ******************************
;
; Debug hardware options
;
; Debug the current drivers
.if Debug_current == Yes
  ldi rmp,0xFF ; All driver pins as output
  out p7SegD,rmp ; in direction port
  out p7SegO,rmp ; and activated
  .endif
;
; Blink the green LED
.if (Debug_ledgreen == Yes)||(Debug_current == Yes)
    sbi pLedGD,bLedGO ; enable output
  Debug_ledgreen1:
    rcall ToggleGreen
  Debug_ledgreen2:
    sbiw ZL,1
    brne Debug_ledgreen2
    rjmp Debug_ledgreen1
  .endif
;
; Debug the segments
.if Debug_segments == Yes
  ; Hint: uses rHours as anode driver and
  ; rMinutes as cathode driver
    ldi rmp,1<<bLedGO ; Green led as output
        out pLedGD,rmp ; Set portpin direction
    rcall ToggleGreen ; Toggle the green led
    ldi rmp,0xFF ; All cathode pins as outputs
    out p7SegD,rmp ; in direction port
    clr rmp ; All cathodes off
    ldi rmp,0b00001000 ; Anode driver 4 as output
    out pAnodeD,rmp ; in direction port
    clr rmp ; Anode drivers off
    out pAnodeO,rmp ; in anode port
  Debug_seg1:
    ; Start with digit 1
    ldi rHours,0x01 ; Start with digit 1 anode
  Debug_seg2:
    ldi rmp,0
        out pAnodeO,rmp
    ldi rMinutes,0x01 ; and with the first segment
  Debug_seg3:
    out p7SegO,rMinutes ; Activate the cathodes
        out pAnodeO,rHours ; and the anode
    ldi rmp,cDebug_segDelay ; Load segment delay counter
  Debug_seg4:
    sbiw ZL,1 ; down-count delay
    brne Debug_seg4 ; until zero
    dec rmp ; Repeat counter
    brne Debug_seg4 ; Additional delay
    lsl rMinutes ; next segment
    brcc Debug_seg3
    lsl rHours
    sbrs rHours,4 ; Bit 4 zero?
    rjmp Debug_seg2 ; No, next digit
    rjmp Debug_seg1 ; Restart with digit 1
  .endif
;
; Debug muxing
.if Debug_mux8 == Yes ; Multiplex the four displays
  ; Wait time for MUX frequency
  ;   Delay of loop is N = 2 + 4 * (c-1) + 3
  ;   c = (N-5) / 4 + 1
  ;   c= (clock/fMux/4-5)/4+1
  .equ cDebug_muxdelay =(clock/cDebug_muxfreq/4-5)/4+1
  ; Hint: uses rHours as anode driver
    ldi rmp,1<<bLedGO ; Green led as output
        out pLedGD,rmp ; Set portpin direction
    rcall ToggleGreen ; Toggle the green led
    ldi rmp,0xFF ; All cathode pins as outputs
    out p7SegD,rmp ; in direction port
    ldi rmp,0xFF ; All cathodes on
        out p7SegO,rmp
    ldi rmp,0b00001000 ; Anode driver 4 as output
    out pAnodeD,rmp ; in direction port
  Debug_mux8a:
        ldi rHours,0x01 ; Anode driver 1 on
  Debug_mux8b:
        out pAnodeO,rHours
    ldi ZH,High(cDebug_muxdelay)
        ldi ZL,Low(cDebug_muxdelay)
  Debug_mux8c:
    sbiw ZL,1
        brne Debug_mux8c
        lsl rHours
        sbrs rHours,4
        rjmp Debug_mux8b
        rjmp Debug_mux8a
  .endif
;
; ********************************
;     N O R M A L   I N I T
; ********************************
;
  ; Init ports
  clr rmp ; Anodes off
  out pAnodeO,rmp ; in Anode port
  ldi rmp,0b00001000 ; Anode 4 as output pin
  out pAnodeD,rmp ; in anode port
  clr rmp ; Outputs cathodes low
  out p7SegO,rmp ; to portpins
  ldi rmp,0xFF ; Port as output
  out p7SegD,rmp ; to direction port
  clr rmp ; DCF and key inputs off
  out pDcfKeyD,rmp ; in DCF and key inputs
  ldi rmp,mDcfKeyO ; Set pull-ups
  out pDcfKeyO,rmp ; in DCF and key pins
  ; Init TC0 for MUX
  ldi rmp,0x01 ; Start with very short dim period
  out OCR0A,rmp
  ldi rmp,(1<<WGM00)|(1<<WGM01) ; Fast PWM mode
  out TCCR0A,rmp
  ldi rmp,(1<<CS01) ; Prescaler=8
  out TCCR0B,rmp
  ldi rmp,(1<<TOIE0)|(1<<OCIE0A) ; Interrupt on overflow and compare A
  sts TIMSK0,rmp
  ; Init TC1
  ldi rmp,High(cDcfT) ;  DCF time-out, MSB
  sts OCR1AH,rmp
  ldi rmp,Low(cDcfT) ; dto, LSB
  sts OCR1AL,rmp
  clr rmp ; CTC mode on compare A
  sts TCCR1A,rmp
  ldi rmp,(1<<CS10)|(1<<CS12)|(1<<WGM12) ; CTC on compare A, presc=1024
  sts TCCR1B,rmp
  ldi rmp,1<<OCIE1A ; Timer int mask for compare A
  sts TIMSK1,rmp ; to int mask port TC1
  ; Init ADC
  ldi rmp,64 ; Start ADC counter
  mov rAdcCtr,rmp ; ... with 64
  ldi rmp,(1<<REFS0) ; MUX to channel ADC0
  sts ADMUX,rmp ; Set channel selection
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  sts ADCSRA,rmp ; Restart ADC
  ; Init flags and other parameters
  clr rFlag
  ldi rSec2H,High(cSec2) ; Init second counter, MSB
  ldi rSec2L,Low(cSec2) ; dto., LSB
  ldi rMin,120 ; Init minute counter
  ldi rHours,cStartHours ; Set initial time, hours
  ldi rMinutes,cStartMinutes ; dto., minutes
  rcall SetTime ; Convert time to display mux
  ldi YH,High(sMuxEnd)
  ldi YL,Low(sMuxEnd)
  ldi rmp,0b00010000
  mov rMux,rmp
  ldi rmp,Low(sMuxEnd) ; DCF output position
  sts sDcfPos,rmp
  ; Init PCINT
  in rInput,pDcfKeyI ; Read inputs
  ldi rmp,(1<<PCINT13)|(1<<PCINT12)|(1<<PCINT11) ; The interrupt generators
  sts PCMSK1,rmp
  ldi rmp,1<<PCIE1 ; PCINT1 enable
  sts PCICR,rmp
  ; Sleep mode
  ldi rmp,1<<SE ; Sleep mode idle
  out SMCR,rmp ; in SMCR
  ; Enable interrupts
        sei ; Enable interrupts
;
```

```
; **********************************
;    P R O G R A M   L O O P
; **********************************
;
Loop:
  sleep ; Go to sleep
  nop ; after wake-up
  sbrc rFlag,bDcfTo ; DCF77 time-out clear?
  rcall DcfTimeOut ; Yes, time out
  sbrc rFlag,bDcf ; DCF input level change clear?
  rcall Dcf ; Yes, analyze
  sbrc rFlag,bKey1 ; Key 1 pressed?
  rcall Key1 ; Yes, react
  sbrc rFlag,bKey2 ; Key 2 pressed?
  rcall Key2 ; Yes, react
  sbrc rFlag,bMin ; Minute flag clear?
  rcall Minute ; Set, go to minutes
  sbrc rFlag,bSec2 ; Second flag clear?
  rcall Second ; Set, go to seconds
  sbrc rFlag2,bAdc ; ADC flag clear?
  rcall AdcFlag ; Set, go to ADC conversion
  .if Debug_keys == Yes
    rcall KeyDisplay
        .endif
  rjmp loop ; Restart loop from the beginning
;
; **********************************
;    F L A G   R E A C T I O N S
; **********************************
;
; **********************************
;  H A L F   S E C O N D   O V E R
; **********************************
;
; Half second over, blink double point
Second:
  cbr rFlag,1<<bSec2 ; Clear flag
  lds rmp,sMux+cAnDp-1 ; Read mux byte where double point is attached to
  sbrc rmp,7 ; Seventh bit clear?
  rjmp Second1
  ori rmp,0x80 ; Set seventh bit
  sts sMux+cAnDp-1,rmp ; Bit 7 high to mux
  ret
Second1:
  andi rmp,0x7F ; Clear seventh bit
  sts sMux+cAnDp-1,rmp ; Bit 7 low to mux
  ret
;
; **********************************
;      M I N U T E   O V E R
; **********************************
;
; A minute is over, increase time
Minute:
  cbr rFlag,1<<bMin ; Clear flag
  lds rmp,sSkipInp ; Read skip input time
  tst rmp ; At zero?
  breq Minute0
  dec rmp
  sts sSkipInp,rmp
  brne Minute0
  cbr rFlag,(1<<bKeyA)|(1<<bKeyM) ; Clear input flags
Minute0:
  ldi rmp,0x07 ; Add 7 to BCD
  add rMinutes,rmp ; to minutes
  brhs Minute1 ; Half overflow
  ldi rmp,0x06 ; Subtract 6
  sub rMinutes,rmp ; from minutes
Minute1:
  cpi rMinutes,0x60 ; 60 minutes over?
  brcs SetTime ; Set the time
  clr rMinutes ; Restart at zero
  ldi rmp,0x07 ; Add 7 to BCD
  add rHours,rmp ; to hours
  brhs Minute2 ; Half overflow
  ldi rmp,0x06 ; Subtract 6
  sub rHours,rmp ; from hours
Minute2:
  cpi rHours,0x24 ; Next day?
  brcs SetTime ; No
  clr rHours ; Restart day
;
; **********************************
;      D I S P L A Y   T I M E
; **********************************
;
; Convert the hhmm time and display
SetTime:
  .if cDcfOnly == Yes
    ; Do not update time
    ret
    .endif
SetTime1:
  sbrc rFlag,bKeyA ; Is a key input active?
  ret ; Yes, skip time output
  ldi XH,High(sMux)
  ldi XL,Low(sMux)
  mov rmp,rHours ; Read hours
  rcall Convert2Seven
  mov rmp,rMinutes
  rjmp Convert2Seven
;
; *******************************
;    D C F 7 7   S I G N A L S
; *******************************
;
; DCF77 Time-Out signal input
DcfTimeOut:
  cbr rFlag,1<<bDcfTo ; Clear flag
  .if cDcfOnly == Yes
    ldi rmp,0 ; Clear the MUX area
    sts sMux,rmp
    sts sMux+1,rmp
    sts sMux+2,rmp
    sts sMux+3,rmp
    .endif
  ret
;
; Active DCF signal
Dcf:
  cbr rFlag,1<<bDcf ; Clear flag
  lds XL,TCNT1L ; Read TC1 count, LSB first
  lds XH,TCNT1H ; dto., MSB next
  tst XH ; Check MSB
  brne Dcf1 ; Larger than zero, fine
  cpi XL,4 ; Minimum is 1 ms
  brcc Dcf1 ; Ok
  ret ; Ignore pulse, too short!
Dcf1:
  .if Debug_dcfdur == Yes
    ; Display signal duration in hex
        mov R1,XH ; Copy duration to R1:R0
        mov R0,XL
        ldi XH,High(sMux) ; X to sMux
        ldi XL,Low(sMux)
        mov rmp,R1 ; MSB first
        rcall Convert2Seven ; Write first two nibbles
        mov rmp,R0 ; LSB next
        rcall Convert2Seven ; Write second two nibbles
        mov XL,R0 ; Copy duration to X again
        mov XH,R1
        .endif
  ldi rmp,0xFF ; Counter for compares
  clr rDcfErr ; Error counter
  ldi ZH,High(2*DcfDur) ; Point Z to value table
  ldi ZL,Low(2*DcfDur)
Dcf2:
  inc rDcfErr ; Next DCF error
  inc rmp ; Next count
  cpi rmp,5 ; Maximum correct count = 4
  brcc DcfErr9 ; Error 9 (Signal too long)
  lpm R0,Z+ ; Read LSB min from table
  cp XL,R0 ; Compare with LSB min
  lpm R0,Z+ ; Read MSB min from table
  cpc XH,R0 ; Compare with MSB min
  brcs DcfError ; Error, signal too short
  lpm R0,Z+ ; Read LSB max from table
  cp XL,R0 ; Compare with LSB min
  lpm R0,Z+ ; Read MSB max from table
  brcc Dcf2 ; Larger than max table value
  cpi rmp,2 ; Zero or one?
  brcc Dcf4 ; No
  ; Received a correct bit
  .if Debug_dcfAny == Yes
    push rmp
    lsr rmp
        ldi rmp,0b01011100
        brcc Dcf2a
        ldi rmp,0b00000100
  Dcf2a:
    rcall DcfReport
        pop rmp
        .endif
  lsr rmp ; Shift counter bit 0 to carry
  ror rDcf7 ; Roll carry into DCF bit buffer
  ror rDcf6
  ror rDcf5
  ror rDcf4
  ror rDcf3
  inc rDcfBits
  rjmp DcfErrClear
Dcf3:
  .if Debug_dcfAny == Yes
```

```
    ldi rmp,0b01110011
    rcall DcfReport
    .endif
  rjmp DcfErrClear
Dcf4:
  cpi rmp,4
  brcs Dcf3 ; Pause, ignore
  ; Received a correct minute signal
  inc rDcfErr ; Next error
  ldi rmp,59 ; 59 bits received?
  cp rmp,rDcfBits ; Number of bits
  ldi rmp,0 ; Clear number of bits
  mov rDcfBits,rmp ; in counter register
  brne DcfError ; Next error
  inc rDcfErr ; DCF error 7
  lsr rDcf5 ; Shift Parity2 to carry
  ror rDcf4 ; and into Byte 5
  ror rDcf3 ; Minute 40s to byte 4
  lsr rDcf4 ; Shift hours right
  ror rDcf3 ; Shift parity1 to minutes
  mov rmp,rDcf4 ; Minutes to rmp
  rcall Parity ; Check parity in rmp
  brne DcfError ; Parity odd
  inc rDcfErr ; Next error
  mov rmp,rDcf3 ; Check parity minutes
  rcall Parity ; Check parity in rmp
  brne DcfError ; Parity odd
  ; All checks performed and errorfree
  mov rHours,rDcf4 ; Read hours
  andi rHours,0x3F ; Remove upper two bits
  mov rMinutes,rDcf3
  andi rMinutes,0x7F ; Isolate minutes
  rcall DcfErrClear
  rjmp SetTime1 ; Display time
;
; DCF signal errors
;   0: No error
;   1: Signal shorter than 0
;   2: Signal shorter than 1
;   3: Signal shorter than pause
;   4: Signal shorter than missing second
;   5: Signal longer than missing second
;   6: Not 59 seconds received
;   7: Minute parity is odd
;   8: Hour parity is odd
;   9: Time out of signal input
;
DcfErr9:
  ; Error 9: signal too long
  ldi rmp,9
  mov rDcfErr,rmp
DcfError:
.if Debug_dcferr==Yes
    ldi XH,High(sMux)
    ldi XL,Low(sMux)
    mov rmp,rDcfErr
    ori rmp,0xE0 ; Error sign
    rcall Convert2Seven
    clr rmp ; Clear the last two digits
    st X+,rmp
    st X+,rmp
  .endif
  ret
;
; Clear the DCF error number
DcfErrClear:
  .if Debug_dcferr == Yes
    ldi rmp,0
    sts sMux+1,rmp
        .endif
  ret
;
; Display DCF report in rmp
.if Debug_dcfAny == Yes
  DcfReport:
    ldi XH,High(sMux)
    lds XL,sDcfPos
    st -X,rmp
    cpi XL,Low(sMux)
    brne DcfReport1
    ldi XL,Low(sMuxEnd)
  DcfReport1:
    sts sDcfPos,XL
    ret
  .endif
;
; Check parity of rmp
Parity:
  clr ZL ; ZL is bit counter
Parity1:
  lsr rmp
  brcc Parity2
  inc ZL ; Count
Parity2:
  brne Parity1
  andi ZL,1
  ret
;
; DCF77 signal durations
DcfDur:
.dw cDcf0Min,cDcf0Max
.dw cDcf1Min,cDcf1Max
.dw cDcfPMin,cDcfPMax
.dw cDcfMMin,cDcfMMax
DcfDurEnd:
;
; *******************************
;   K E Y   P R O C E S S I N G
; *******************************
;
; Key1 is pressed
Key1:
  cbr rFlag,1<<bKey1 ; Clear flag
  ldi rmp,0b10101010
  sts sMux,rmp
  ldi rBounce,cBounce ; Start bouncing period
  sbrc rFlag,bKeyA ; Key input inactive?
  rjmp Key1Active ; Yes, key flag is active
  sbr rFlag,1<<bKeyA ; Set key flag active
  ret
Key1Active:
  sbrc rFlag,bKeyM ; Minute flag clear?
  rjmp Key1Minute ; No, go to minute
  sbr rFlag,1<<bKeyM ; Set M flag
  ldi rmp,24 ; Convert ADC to hours
  mul rmp,rAdc
  ldi XH,High(sInpTime) ; Point to hours input
  ldi XL,Low(sInpTime)
  rcall ToBcd ; Convert binary in rmp to packed BCD
  mov rmp,ZL ; Result to rmp
  ldi XH,High(sMux) ; Hours display
  ldi XL,Low(sMux)
  rjmp Convert2Seven ; Display the selected hours
;
Key1Minute:
  ldi rmp,60 ; Convert ADC to minutes
  mul rmp,rAdc
  ldi XH,High(sInpTime+1) ; Point X to minutes
  ldi XL,Low(sInpTime+1)
  rcall ToBcd ; Convert to BCD
  adiw XL,1 ; Point to behind minutes
  ld rMinutes,-X ; Result to minutes
  ld rHours,-X ; and hours
  ldi rSec2L,Low(cSec2) ; Restart half second divider
  ldi rSec2H,High(cSec2)
  ldi rMin,120 ; Restart minute counter
  cbr rFlag,(1<<bMin)|(1<<bSec2)|(1<<bKeyA)|(1<<bKeyM)
  rjmp SetTime ; Set the current time
;
; Key2 is pressed
Key2:
  cbr rFlag,1<<bKey2 ; Clear flag
  ldi rBounce,cBounce ; Start bouncing period
  sbrs rFlag,bKeyA ; Key input active?
  ret ; No, ignore key
  sbrc rFlag,bKeyM ; Minute active?
  rjmp Key2Minute ; Yes
  cbr rFlag,1<<bKeyA ; Stop input
  rjmp SetTime
;
Key2Minute:
  cbr rFlag,1<<bKeyM ; Return to hour input
  rjmp SetTime
;
; Convert the binary number in R1 to packed BCD in ZL
;   and write result to the X location
ToBcd:
  ldi ZL,-0x10
  ldi rmp,10
ToBcd1:
  subi ZL,-0x10
  sub R1,rmp
  brcc ToBcd1
  add R1,rmp
  add ZL,R1 ; Add the ones
  st X,ZL ; Store packed BCD in SRAM
  ret
;
; Displays the keys
KeyDisplay:
  ldi rmp,0b01011100 ; Small 0
  sbic pDcfKeyI,bKey1I ; Key 1
  ldi rmp,0b00000100 ; Small 1
  sts sMux,rmp
  ldi rmp,0b01011100 ; Small 0
  sbic pDcfKeyI,bKey2I ; Key 1
  ldi rmp,0b00000100 ; Small 1
  sts sMux+1,rmp
  ret
;
; **************************************
;    A D C   R E S U L T   R E A D Y
; **************************************
;
AdcFlag:
  cbr rFlag2,1<<bAdc ; Clear ADC flag
  mov rAdc,rAdcSumH ; Copy MSB sum
  clr rAdcSumL ; Restart sum
  clr rAdcSumH
  ldi rmp,64 ; Restart counter
  mov rAdcCtr,rmp ; into rAdcCtr
.if cDimOpto == Yes
  ldi rmp,(1<<REFS0)|(1<<MUX0) ; Mux to channel ADC1
  sbrc rFlag,bKeyA ; Clock setting active?
  ldi rmp,(1<<REFS0) ; Yes, MUX to channel ADC0
  .else
  ldi rmp,(1<<REFS0) ; MUX to channel ADC0
  .endif
  sts ADMUX,rmp ; Set new channel selection
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  sts ADCSRA,rmp ; Restart ADC
.if Debug_Adc == Yes
  rjmp AdcOut ; Display the ADC result
  .endif
  sbrc rFlag,bKeyA ; Key input active?
  rjmp AdcFlag2 ; Update the current input digit
  .if cDimOpto == Yes
    com rAdc ; Invert result
        .endif
AdcFlag1:
  out OCR0A,rAdc ; Write result to compare port TC0
    ; for dimming
  ret
AdcFlag2:
  tst rSec2H ; MSB larger than 0?
  brne AdcFlag3 ; Yes, display number
  cpi rSec2L,c75pcon ; LSB smaller than 75%
  brcc AdcFlag3
  ; Switch the two digits off
  ldi XH,High(sMux) ; Point X to hours
  ldi XL,Low(sMux)
  sbrc rFlag,bKeyM ; Minute key clear?
  ldi XL,Low(sMux+2) ; No, point to minutes
  clr rmp ; Write zeroes to digit
  st X+,rmp
  st X,rmp
  ret
AdcFlag3:
  ; Key input is active, calculate digit
  ldi rmp,24 ; Multiply ADC result by 24
  sbrc rFlag,bKeyM ; Minute flag clear?
  ldi rmp,60 ; Multiply ADC result by 60
  mul rmp,rAdc ; Hardware multiplication
  ; Convert MSB result to packed BCD
  ldi ZL,-0x10 ; Start with minus 10
  ldi rmp,10
AdcFlag4:
  subi ZL,-0x10 ; Add ten to result
  sub R1,rmp ; Subtract 10 from ADC MSB
  brcc AdcFlag4 ; If not carry continue subtracting
  add R1,rmp ; Undo last subtraction
  add ZL,R1 ; Add rest to result
  mov rmp,ZL
  ldi XH,High(sMux) ; Point X to hours
  ldi XL,Low(sMux)
  sbrc rFlag,bKeyM ; Minute key clear?
  ldi XL,Low(sMux+2) ; No, point to minutes
  rjmp Convert2Seven ; Convert to display
;
; Display the ADC results
AdcOut:
  ldi XH,High(sMux) ; Point X to sMux
  ldi XL,Low(sMux)
  ldi rmp,0b01110111 ; A sign
  st X+,rmp ; to hour tens
  mov R1,rAdc ; Move the rAdc to R1
  ldi rmp,100 ; Hundreds
  rcall AdcOutDec ; Count hundreds
  ldi rmp,10
  rcall AdcOutDec
  mov R0,R1 ; Display the rest
  rjmp AdcOutDec2
;
; Convert R1 to a decimal and display
;   rmp is the decimal (100, 10)
;   Uses R0
AdcOutDec:
  clr R0
  dec R0
AdcOutDec1:
  inc R0
  sub R1,rmp
  brcc AdcOutDec1
  add R1,rmp
; Convert R0 to 7segment and display
AdcOutDec2:
  ldi ZH,High(2*SevenSeg)
  ldi ZL,Low(2*SevenSeg)
  add ZL,R0
  ldi rmp,0
  adc ZH,rmp
  lpm rmp,Z
  st X+,rmp
  ret
;
; ***************************************
;    B A S I C   S U B R O U T I N E S
; ***************************************
;
; Toggles the green led
;   by outputting on the in port
ToggleGreen:
  ldi rmp,1<<bLedGO ; The green led
  out pLedGI,rmp ; Toggle the led
  ret
```

```
;
; Convert rmp to 7segment and write result to X
Convert2Seven:
  push rmp ; Save for LSB
  swap rmp
  rcall Convert2SevenDigit
  pop rmp
Convert2SevenDigit:
  andi rmp,0x0F
  ldi ZH,High(2*SevenSeg) ; Load table, MSB
  ldi ZL,Low(2*SevenSeg) ; dto., LSB
  add ZL,rmp ; Add number, LSB
  ldi rmp,0 ; Zero
  adc ZH,rmp ; Add carry
  lpm rmp,Z ; Read from flash
  st X+,rmp
  ret
;
; Seven-segment table
;     ---- a     hgfedcba
;  f |    | b
;     -g--
;  e |    | c
;     ----  d
;
SevenSeg:
.db 0b00111111,0b00000110 ; 0+1
.db 0b01011011,0b01001111 ; 2+3
.db 0b01100110,0b01101101 ; 4+5
.db 0b01111101,0b00000111 ; 6+7
.db 0b01111111,0b01101111 ; 8+9
.db 0b01110111,0b01111100 ; 10(A)+11(b)
.db 0b00111001,0b01011110 ; 12(C)+13(d)
.db 0b01111001,0b01110001 ; 14(E)+15(F)
;
; End of source code
; Copyright
.db "(C)2019 by avr-asm-tutorial.net "
.db "C(2)10 9yba rva-mst-turoai.lrn t"
;
```

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