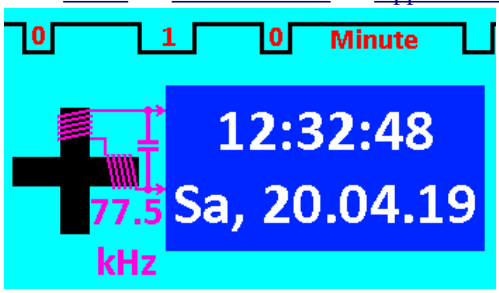


Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 receivers

Note that parts of these pages have not been tested yet and provide pre-liminary information only! Some links do not yet work because this webpage is still under construction.

Table of content

DCF77 receivers

Of course: there are cheap (but also very expensive) DCF77 receivers available (at least in Germany) and there is no need for home-brewing. So why built your own? Now, just because it is fun, and because you'll learn some RF basics, and as it is fine to handle RF by yourself (and not to end as a RF lay person while hanging on your mobile all the time). And if you live in some distance to Frankfurt/Germany: the commercially available receivers are so dump that you need some more amplification to get this signal and to build your own atomic watch for it. And what about those that operate their commercial receiver in an environment that produces lots of very-long wave signals, such as Chinese switching power supplies or energy saving lamps? The commercial receiver then is overwhelmed by those signals, and does not find date and time in the air. Here you'll find receivers that are small enough to work correct even under these adverse circumstances.

DCF77 how-to

DCF77 is a transmitter that "officially" (yes, there is a law on that) sends time and date information continuously. It transmits in the Very-Long-Wave band at 77.5 kHz. The time and date information is encoded into 59 bits that are send within one minute, with the 60th bit missing (signaling that the minute is over). These bits are sent by temporarily reducing the RF power of the transmitter down to 20% of its peak power for either 100 ms (which is a zero bit) or 200 ms duration (which is a one bit). So, all you need to do is to

- detect amplitude drops and risings of the received RF signal,
- to measure the duration of those, decide whether they encode a zero or a one, and to collect those bits,
- to measure the duration of pauses in the signal, where RF power of DCF77 is high, and to detect pauses of 1.800 to 1.900 ms duration, when a minute change occurs,
- to re-arrange the 59 bits to extract BCD encoded
    - minutes,
    - hours,
    - weekday (1 for Monday, etc., to 7),
    - day,
    - month, and
    - year.

  all for the minute that started after this long pause, and: yes, that can all be done with standard CMOS gates, with two hands full of integrated circuits filling a Euro

size board. Thanks to modern micro controllers that all fits into an 8-pin DIP IC nowadays.
- displaying all this on a 1-, 2- or 4-line LCD.

That is all it needs. If you want to do that with a PC or laptop running a modern operating system: forget it, you won't be able to get this modern operating system with its time-sharing and window reporting system to count 100 or 200 ms long pulse durations. Better use an ATtiny to do all this and transmit the date and time via a RS232 or whatever serial interface to the PC or laptop. An ATtiny works with less than one percent of the clock rate of a PC but is fast enough to react on even shorter pulses. Modern operating systems are neither designed nor able to react fast enough on those events.

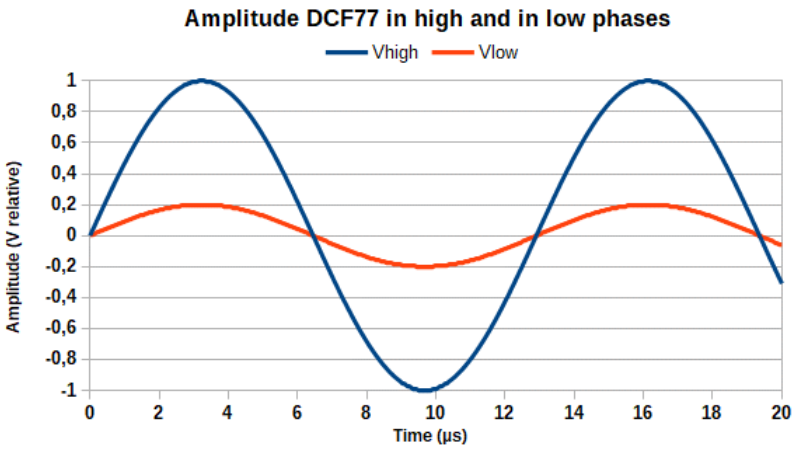
Unable to program AVRs? Not willing to learn assembler programming? Yes, you can do that all with two hands full of CMOS-ICs. The time and date bits of DCF77 are 59 bits long, for that you need eight 8-bit shift registers. If you skip the first 20 bits, you need ”only” five of those. To check the parity bits of the minutes and hours, you need two parity generators. If you want to check the parity of the 23 date bits as well, you need additionally three of those. If you want to display that on 7-segment digits, you'll need at least ten 4-bit-latches-and-decoders, if you want to see the seconds and the weekdays as well: additionally three of those and an 8-bit counter. If you want to be alarmed at a certain time, you'll need additional counters and comparers. And if you want your watch to function correct even when DCF77 is not transmitting due to local lightning, you'll need at least ten additional CMOS-ICs. And, due to the current-hungry 7-segment displays, at least 500 mA power supply. In contrast to that: an ATtiny25 in an 8-pin-package and an ATtiny24 in a 14-pin-package, together with a four-line LCD do that all with less than 10 mA (mainly for the LCD's backlight). With a small rechargeable battery of 1,200 mAh you can operate this for days. No: do not fall back to the Eighties, it is not worth it. Just learn how to program AVRs and to write assembler programs rather than investing your time and brain in boring CMOS wiring.

On this web page you'll find all you need to receive, detect and decode the time and date.

DCF77 receiver basics

77.5 kHz is a bit faster than audio signals, but still is in the same range (ok, bats do not hear that any more). So RF of this frequency is less sensitive and does not need special RF transistors or high-speed opamps. So you can just amplify it with any transistor or opamp type, such as a 741. Ideal for a beginner in RF.

Special is only that the signals come in with a rather small voltage. Well below a standard dynamic microphone with its 5 mV. Here, at a distance of 28 km to Mainflingen near Frankfurt, a ferrite antenna tuned with a capacitor to 77.5 kHz produces a sine wave with roughly 5 mV, which already can be seen on an analog oscilloscope. But at



larger distances, only a few micro-volt come from the ferrite, associated with lots of (random or systematic) noise.

That is how the two amplitudes, in a high and a low phase, look like over time during one-and-a-half sine waves. The information is encoded in that amplitude (amplitude-modulated, AM), so we have to detect the amplitude's height to decode the information therein.

That is why we cannot use amplifiers that have no gain regulation: either we have a too small gain and our amplitude rectifier does not see a signal at all, or we have a too large gain: then we get a rectangle, which has nearly the same peak voltage on the rectifier, no matter if the amplitude is high or low. And our information is lost in both cases.

So, we have to regulate the gain of the amplifier (automatic gain control, AGC), so that the amplified signal produces enough rectified voltage in high phases, but not too much so that the rectifier starts clipping the amplitude in that high phase. Just enough, so that we detect the loss of amplitude in the low phase. And: the gain regulation has to be slowed down, so that it doesn't increase the gain during the low phase, which can last either 0.1 or 0.2 seconds. The delay in regulation shall be longer than one second or longer.

Fortunately a simple ferrite rod, with some tens of windings of copper wire, and a capacitor of a few nF capacity are a very good RF filter. At resonance, its resistance is extremely high (approx. some 100 kΩ) and its related bandwidth is rather small (a few kHz). So a ferrite rod is

1. a good receiver for that kind of RF,
2. a good collector, as it collects RF over its complete length,
3. a very good amplifier as it increases RF voltages if in resonance (not so much the overall power due to the high resistance but only the voltage), and
4. also a good selector, suppressing 50/60Hz stray voltages as well as your local short wave transmitter signal with a few Megawatt power.

Do not try ferrite-free air coils, they do not have enough inductivity (or otherwise are extremely large for that low frequency).

Unfortunately ferrite rods are sensitive to directions: if your ferrite points to the wrong direction, you'll get nothing but noise and nothing to derive time and date from. This web site also has a solution for that, see below.

As the distance to the transmitter and the direction of the rod towards DCF77 play a role, and also propagation issues of the VLW band might play a role, an amplifier with a fixed gain, e.g. in my case 1,000 would be enough, is not a good idea. It is either too high, by that overloading the AM rectifier stage and no amplitude drop can be detected or it is too low and does not produce a DC signal, if its peak voltage is below the diode's forward voltage of 0.2 or 0.3 V. So, a good DCF77 receiver has to have a gain regulation. That makes a 741 opamp or a simple transistor amplifier a very bad choice.

Gain regulation should be able to regulate the amplifier gain by at least a factor of 10 (in the near-field) or 100 (in larger distances). And it should be automatically follow the changing signal strength, making it an AGC (automatic gain control). It should not be fast enough for the 100 ms or 200 ms long amplitude drops, so that gain regulation would mask the incoming bits, but should rather be able to average the signal over a few seconds.

If you are in a distance of several 100 km or even beyond 1,000 km, you need much more gain than 1,000 to get the DCF77 signal. If your necessary gain is in the 10,000 to 100,000 range, an issue plays a role that any amplifier of that gain has: stray signals can oscillate the amplifier. This is especially the case if each of your amplifier stages reverses the signal by 180 degrees (as usual transistor amplifiers do) and your third stage strays its signal back into the first stage's input: a perfect oscillator is working then. Self-oscillation is an issue, even at only 77.5 kHz and even if you regulate the gain to down below the oscillation point. So, the direct receiver always has a limited gain.

As, unfortunately, your ferrite rod is a perfect receiver for those stray signals, it helps to do the amplification of the signal on a different frequency than that the ferrite rod is tuned to. Here, the superhet principle comes into play: it mixes the input frequency (77.5 kHz) with an oscillator frequency (in that case e. g. 110.268 kHz), filters the subtracted product (in that case 32.768 kHz) and amplifies this. As 32.768 kHz is far away from the ferrite rod's 77.5 kHz, it does not interfere with that. And: this intermediate frequency (IF) can be filtered by using easily available xtals (for watches), so that even 10 or 20 Hz below or above signals are perfectly suppressed. This also disables noise and disables interfering signals from power supplies and energy saving lamps. This web site shows how to do that, see below.

For the beginner in RF, a short intro to resonance might be useful. A coil is a resistor for AC: its resistance is depending from the AC's frequency and can be calculated by the following equation:

$$Z_L = 2 * \Pi * f * L$$

with Z being in Ohms (Ω), Π is 3.141592654, f in Hz and L in Henry (H). The resistance increases if the frequency or the inductivity increases.

The same for capacitors, but in that case it is reversed:

$$Z_C = 1 / 2 / \Pi / f / C$$

Z again in Ω, f again in Hz, C here in Farad (F). The resistance decreases if f or C increases.

The term **2 * Π * f** is called circular frequency and abbreviated as small omega (ω). With that the above formulas are as follows:

$$Z_L = \omega * L$$
$$Z_C = 1 / \omega / C$$

At resonance both the inductive and capacitive resistance is equal, $Z_L = Z_C$, making

$$\omega * L = 1 / \omega / C$$

In case of resonance, the inductive and capacitive resistance increases with a **quality factor**, depending mainly from the normal (Ohm's) resistance of the coil. This factor is around 100 for a normal coil or 40 for a high-Ohm coil. That is why the ferrite resonant circuit has such a high resistance at 77.5 kHz.

Arithmetic says that you can calculate

inductivity by $L = 1 / \omega^2 / C$
capacity by $C = 1 / \omega^2 / L$ and
frequency by $f = 1 / 2 / \Pi / \sqrt{(L * C)}$

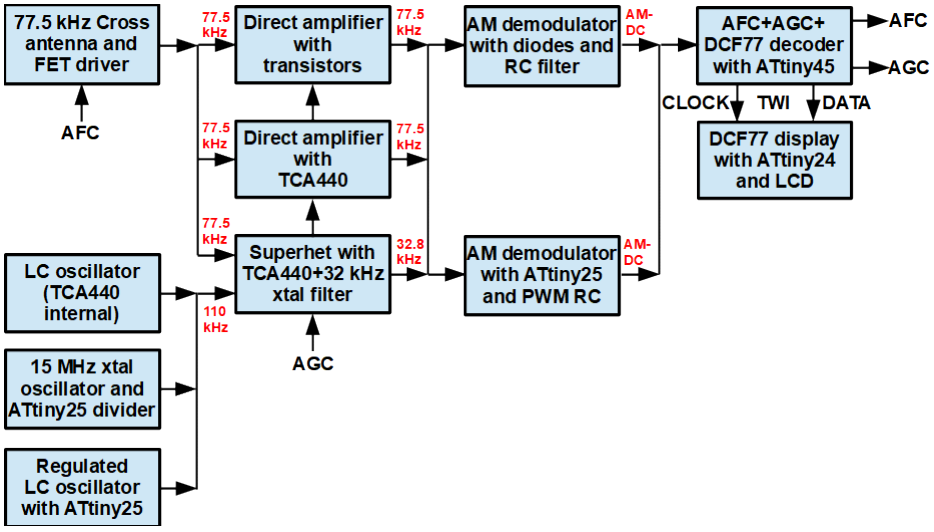
That is all that you need in the math section.

What you get here

Overview on what is described here

Here are some descriptions of home-brew-able receivers for DCF77. Lots of different tastes are covered here.

A **cross antenna** for DCF77 (described here), that makes reception of DCF77 independent from directions towards Mainflingen near Frankfurt, where the transmitter is located. A 90° and a 45° version has been designed, built and tested. The antenna includes a FET stage that serves as a buffer between the high-impedance ferrite antenna and the capacitor(s) that form a resonant circuit and the lower impedance of the following amplifier stages. To adjust the frequency of the resonant circuit exactly to DCF77's transmit signal on 77.5 kHz an automatic frequency control (AFC) has been added, consisting of a variable capacity diode (varactor), a capacitor and a resistor. Adjusting the AFC voltage allows to vary resonance frequencies between 77 and 78.5 kHz (for the 90° cross antenna and over a larger bandwidth for the 45° cross antenna. This brings an elevated noise immunity and a higher RF sensitivity.



A **direct amplifier for DCF77 RF with transistors** (described here): amplifies the 77.5 kHz RF by several thousand-fold to allow reception in the far distance to the transmitter. Works with standard electronic parts and does not use special parts. Two stages amplify the weak signal. Of course, the gain of the amplifier can be adjusted. This is done with diode attenuators, so that the working conditions of the transistor amplifiers remain unchanged. The diode's currents can be manually adjusted or via a PWM plus a PNP buffer stage.

As an alternative to transistorized stages a **TCA440 amplifier** can be used (described here). This provides even more gain. The oscillator and mixer, also integrated in a TCA440, are not used, only the IF amplifier stages. The gain can be easily adjusted by applying increased voltages on the respective input pin. The TCA440 has an auxiliary output to drive a mechanical meter for the gain (leave that open if you don't need it). While the

number of necessary parts is smaller than of a transistorized version, the accessibility of TCA440s is also smaller as the circuit is not in production any more.

A **superhet receiver** with a TCA440 (described here): Reception and pre-amplification are on a frequency of 77.5 kHz, then mixed with an oscillator frequency to form a 32.768 Hz mixer product. Either 77.5+32.768 or 77.5-32.768 kHz can be used for that. The internal oscillator is working with an external coil and capacitor and works on 110.268 kHz. The mixer signal is then filtered with an LC circuit and 32.768 kHz crystal(s). That is fed into the IF amplifier. Its output, again filtered with a LC resonant circuit, is rectified and produces an amplitude dependent DC signal. The AGC of the IF amplifier works as described above.
In a subversion, the TCA440's oscillator input is driven with a crystal-derived sine wave signal of 110.294 kHz, produced by an AVR ATtiny25, which is clocked with a 15 MHz xtal oscillator and divides this clock by 68 and by 2. Rectangle to sine wave conversion uses a 3-stage RC filter for the positive and negative output, which are fed into the TCA440's oscillator input. The box on the bottom shows this concept.

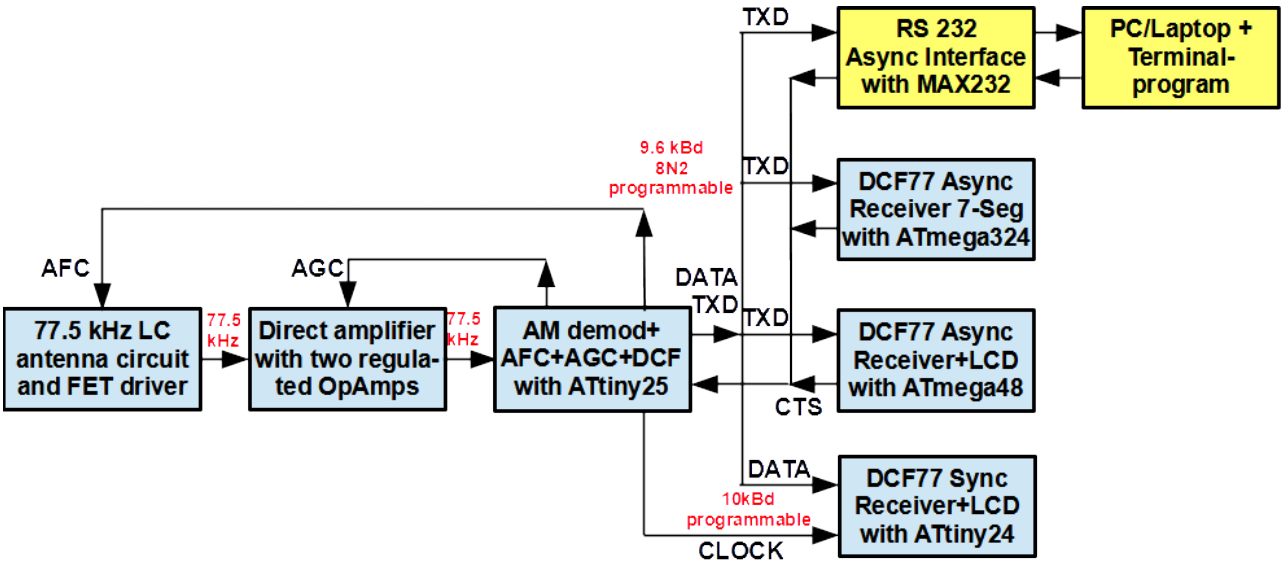
In another subversion the oscillator signal for the TCA440 is generated by a regulated LC oscillator. A crystal driven ATtiny25 measures the frequency of the LC oscillator and regulates the frequency by use of a PWM and two varicap diodes.

The **rectification** of the DCF77 signal from the direct receivers as well as of the 32 kHz IF from the superhet can be made with diodes. An alternative solution would be an ATtiny25 rectifier (described here).

An ATtiny25 has anything under **control** (described here): it measures the DC coming from the rectifier, derives the AFC and AGC voltages with two PWM channels and detects amplitude losses (zeroes and ones from DCF77. It decodes those zeroes and ones, derives the time information from that, detects errors in the DCF77 signals and sends all information over a tw-wire interface to an ATtiny24.

An ATtiny24 **displays** all the information received from the ATtiny25 on an attached 4-line LCD (described here).

As a bonus application I added a direct receiver, described here, that does it all in a small Tiny25: to detect the amplitude of DCF77, to generate voltages that control the frequency of the input LC ferrite circuit as well as the gain of two operational amplifiers, to decode the DCF77 amplitude drops, to convert it into time and date information with extensive error detection and to send the decoded information either over a synchronized serial interface or to transmit these over an asynchronous serial interface.



For displaying the information four different opportunities are presented, that span from displaying the received time and/or date information

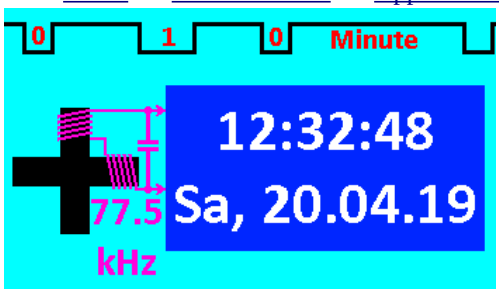
1. with an ordinary terminal program over a standard RS232 interface, or
2. to receive it in async mode and synchronize a 7-segment display, e. g. with a large LED display, with that, or
3. to receive it asynchronously and display it on an LCD of programmable size (single, double or four lines, 8/16/20/24 characters per line), or
4. to receive it synchronously and display it on an LCD with programmable size.

Links to other documents

The following additional documents can be downloaded from the website:

- All calculation sheets in one Libre-Office spreadsheet document here (24 sheets, 1,36 MB).
- All drawings in one Libre-Office draw document here (25 drawings, 108 kB).

Note that in sub-pages of the website additional Libre-Office documents are available for download.



Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 cross antenna

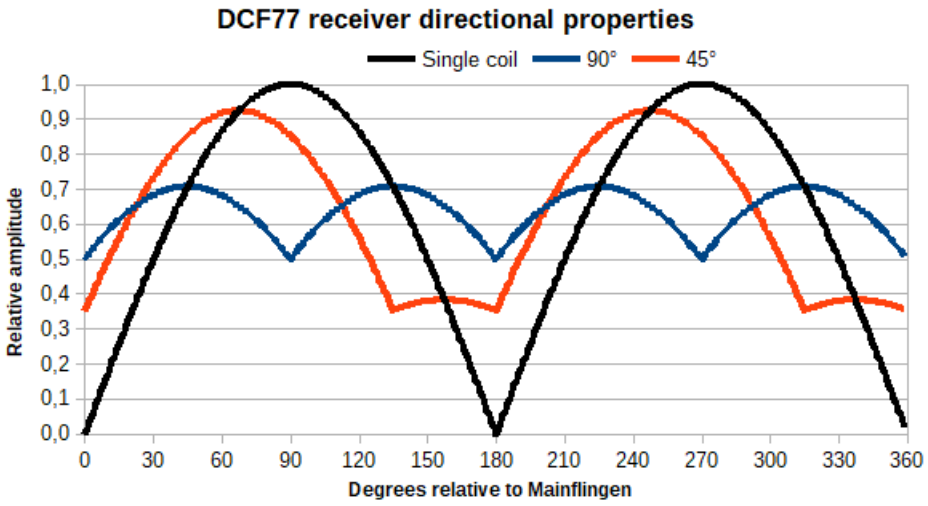
1 DCF77 cross antenna

The German date and time standard transmitter signal of DCF77 can be received all over Europe, but signal strength is strongly depending from the antenna direction. To get independent from the direction this cross antenna has been developed and tested.

I tested two versions of the antenna. Both consist of two coils on two different ferrite rods. In the first version, those two ferrite rods are mounted in an angle of 90°. In the second version those are mounted in an angle of 45°. By angling, one of the two rods always has reception, no matter what angle the DCF77 transmitter has towards the antenna rods, even if the other rod is completely misaligned. The signal of the sum of both coils is never zero, the misaligned coil just does not add to the total signal.

The second version, with 45°, has been developed because two horizontal 10 cm rods do not fit into many plastic casings. In small casings (e. g. with 5 cm) the first rod can be placed on the bottom while the second rod can be placed on the top of the casing, both having 45° offset to each other. The nearer the angle of both towards 90° they come, the smaller is the angled amplitude difference.

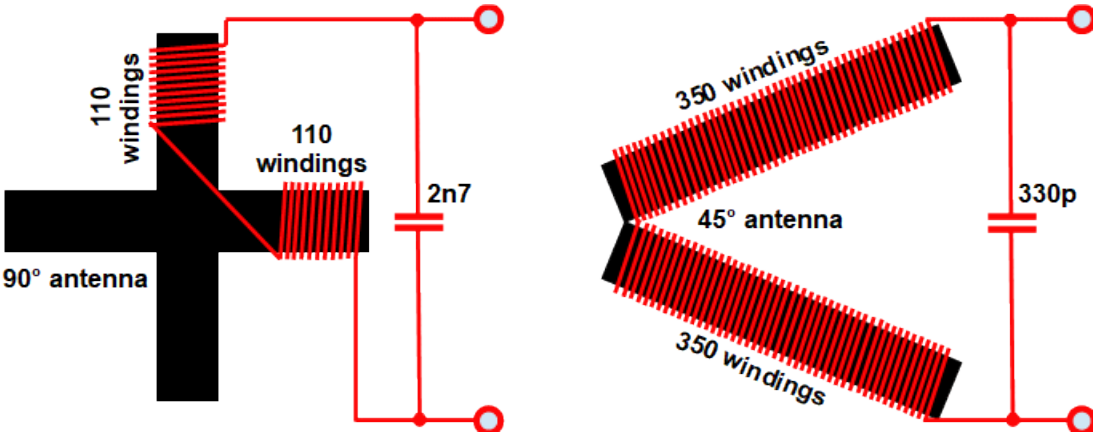
That is how the signal strength varies in different angles, for a single rod/coil and for two rods/coils in 90 and 45° direction. In the cases with two rods the reception strength is never zero but varies between 0.5 and 0.7 (90°) resp. 0.92 and 0.35 (45°).



To test the two rods a little further, I have given them different windings. For the 90° version I covered 45% of the 10 cm ferrite rod with copper enameled wire (0.255 mm) in single layer fashion (just because the two rods then can be tied together in a 90° angle without too much interference between the two coils). That meant 110 windings for each coil. In the second version I covered the complete rods with single-layer wire, leaving 0.5 cm on both ends uncovered. That meant approximately 350 windings for each coil.

Tying the two coils together and angling them has interesting effects on the inductivity of the single coils and on their sum. In the 90° case the inductivity of the crossed coils is smaller than the sum of the inductivity of both, while in the 45° case their inductivity increases. If, in the second case, their mounting in a larger distance (e.g. 10 cm between both) is chosen, this effect will be much smaller than with both ends tied together.

1.1 Mounting



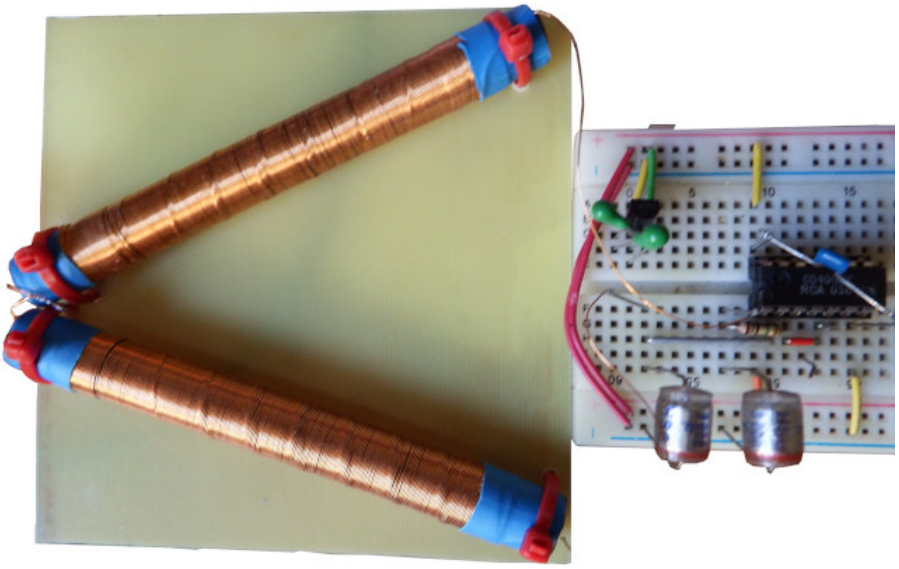
The 90° antenna (left picture) is mounted like shown. First the rod is shrinked with a piece of plastic cover to get some distance between the rod and the wire. The 10 cm rod is then covered with 110 windings of 0.255 mm copper wire over 45% of its length and both wire ends are fixed with plastic tape. The two rods are then tied together with two crossed cable ties so that the angle is approximately 90°. The inner ends of the two coils are soldered together.

The 45° antenna is mounted similarly but the whole rods are covered with wire (except 5 mm on both ends). For each coil one needs approximately 13 m of copper wire. Both ends are fixed with a piece of shrink sleeve. Both ferrite rods/coils are mounted on a blank 100-by-100 mm epoxy plate and fixed with cable ties. The near ends of both coils are soldered together.

1.2 Measuring the coils

The coils have been measured with two different methods:

1. with a FET and a variable capacitor equipped grid dip meter,
2. with a CMOS oscillator.



1.2.1 Measuring results with a grid dip meter

With my grid dip meter I installed the coil and varied the FET oscillator with the 2*365 pF variable capacitor. From

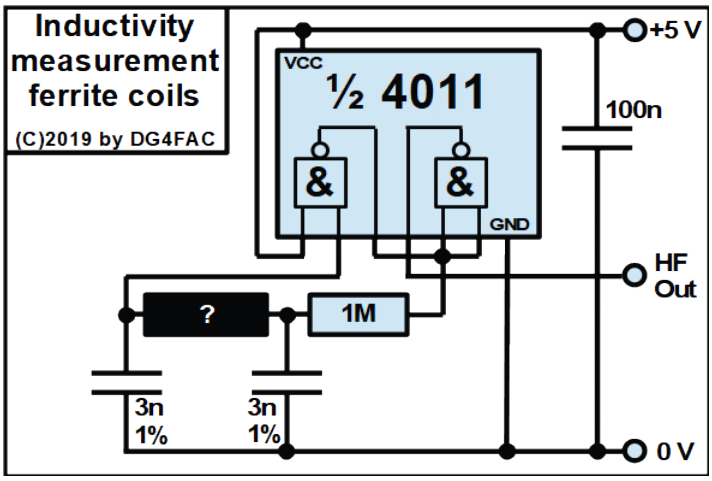
previous experiments with fixed inductivities a capacity of 200 pF (in full capacity) resp. 23 pF as smallest capacity has been determined.

The two 45° coils oscillated with 217.64 resp. with 221.75 kHz under full capacity, resulting in inductivities of 2.67 resp. 2.58 mH. The sum of both would then be 5.25 mH.

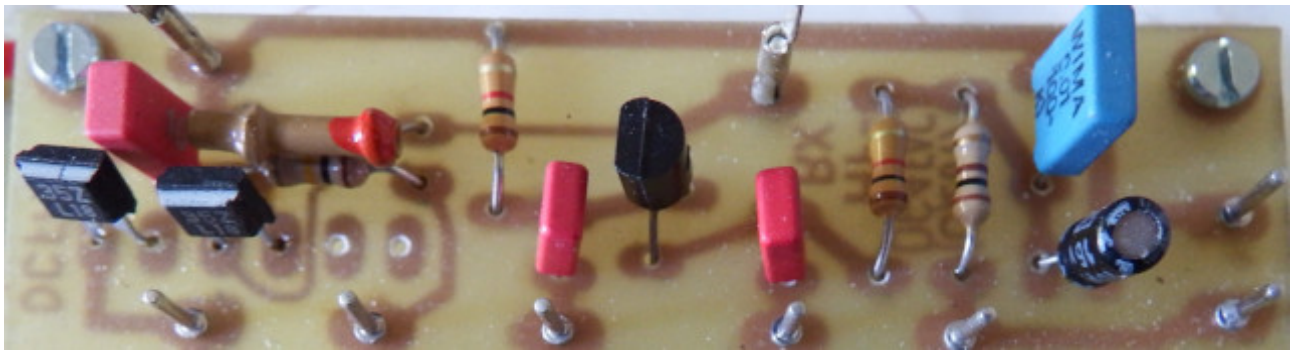
1.2.2 Measuring results with a CMOS oscillator

This schematic was used to determine the inductivity in a different method. Measuring with this resulted in significantly larger inductivities of 3.87 resp. 3.79 mH, which would result in a sum of 7,66 mH.

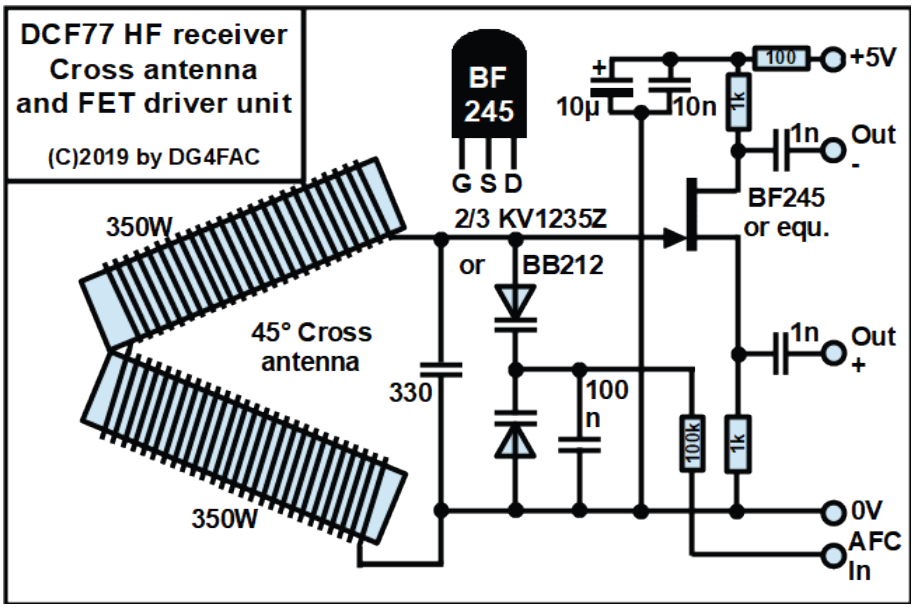
Measuring the 45° coils tied together resulted in a significantly higher inductivity: 9.58 mH. That's what you get from nearing the coils in an angle.



1.3 Buffer stage



This is the schematic for the buffer stage (the 45° version). The antenna circuit is formed with the stacked coils and a capacitor of 330 pF. The signal goes to the gate of a N-FET (any N-FET type can be used). The drain and the source of the N-FET are connected to two resistors of 1k (to the filtered operating voltage and to ground, the HF is coupled with two 1nF capacitors (Z_C = 2.5 kΩ) to the amplifier stages (symmetric output).



Note that the 90° version needs a larger capacitor of 2.7 nF due to the smaller inductivity of the coils.

The buffer stage with the FET is necessary to protect the sensitive properties of the LC resonance circuit. The large coil above has an inductivity of 9.58 mH. That means that the coil has, at 77.5 kHz, an inductive reactance of $Z_L = 2 * \Pi * f * L$ of 4,66 kΩ. If the coil is in resonance with the capacitor, the reactance of the LC circuit is by a factor of Quality larger than this, the circuit has more than 466 kΩ. That means that the resonance curve is very narrow, suppresses nearby noise sources and the sensitivity is very high.

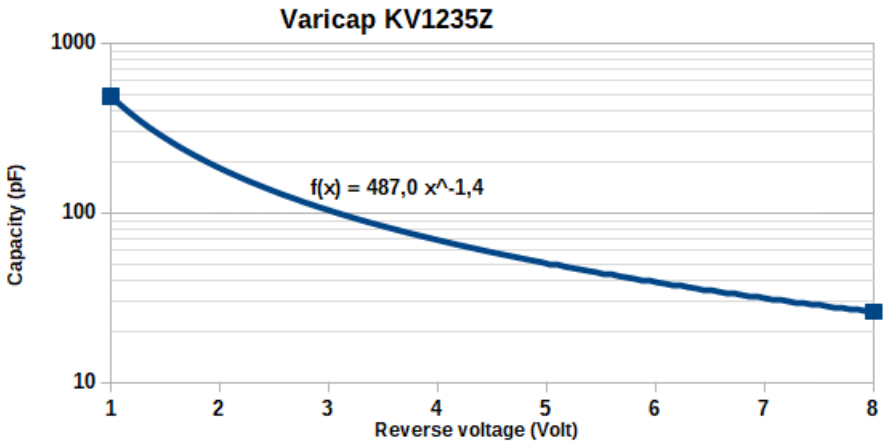
Hence, it would be not a good idea to attach a stage with a lower resistance to it. This would seriously drop the LC circuit's high sensitivity and would broaden the resonance curve. The FET stage does not amplify, but only keeps the high entry resistance and provides a reduction of the resistance at its output. The high quality of the LC circuit on its input is protected and kept.

1.4 AFC Frequency adjustment

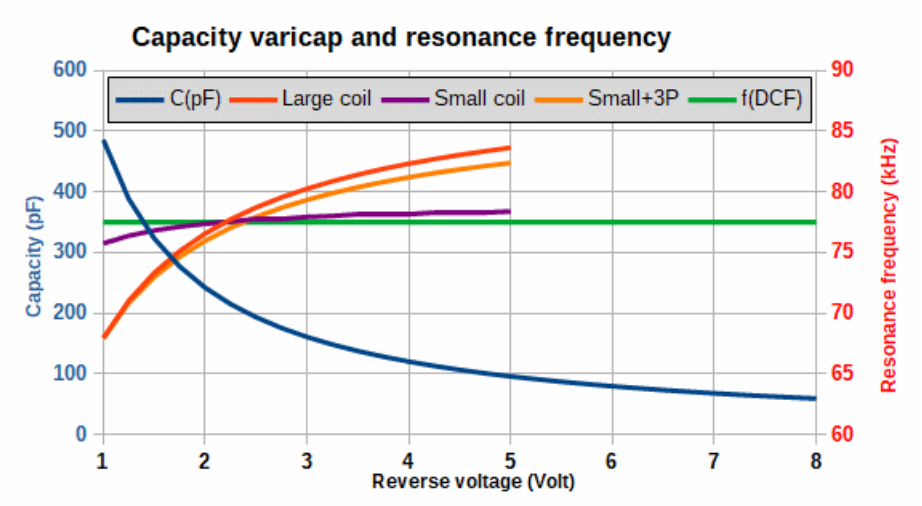
The resonance frequency of the cross antenna and the 330pF capacitor can differ slightly (temperature, iron in the near field, etc.). Therefore two varactor diodes are attached to the antenna circuit, both in reverse direction (anti parallel). I have used two of the three diodes in a TOKO KV1235Z, use of other types such as BB112 (double diode) is possible. The diodes should at least have 100 pF at 0.7V (medium wave types).

Depending from the AFC voltage (0 to 5 V) half of the capacity of the regulating varactor lies parallel to the antenna circuit. This allows for a sensitive regulation of the resonance frequency and adjustment to 77.5 kHz. You can use a potentiometer, a trim resistor or a digital PWM to adjust that. Because the varactor diodes are operated in reverse direction, no current is drawn from the diodes.

This is approximately the capacity of the KV1235Z diode versus the reverse voltage applied. As the original curve in the available datasheets looks a little bit weird, I have interpolated it with a polynome (see the calculation sheet "FET-RX" in the LibreOffice Calc file. So do not expect this to be correct.



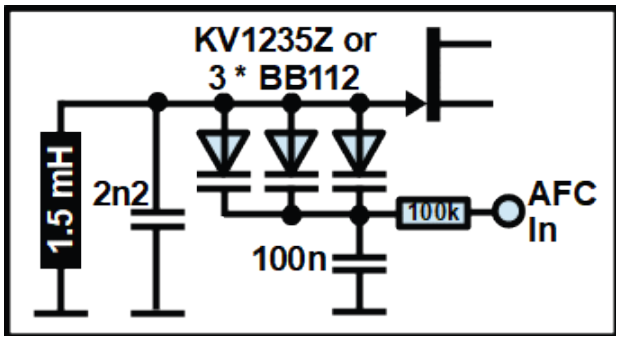
This is the capacity of the varactor diode and the resonance frequency of the cross antenna versus the AFC voltage applied. Two combinations are considered here:



- a large coil of 9.58 mH and a small fixed cap of 330 pF (red curve),
- a small coil of 1.5 mH and a large fixed cap of 2.7 nF (violet curve).

The range that the combination of the large coil and the small fixed cap allow is fine, but the small coil with the large fixed cap covers only a very small range. Note that two of those varicap diodes are anti-parallel, so their capacity is halved.

In this case we can apply the varicaps a little different to enlarge the range: we put three of them in parallel and reduce the fixed cap to 2.2 nF. The orange curve in the diagram shows that the range is now comparable to that with the large coil.



1.5 Properties of the cross antenna

By adding two coils with only one capacitor in the antenna circuit a phenomenon occurs that has to be accounted for in frequency adjustment: both coils have a combined inductivity as if they were one but each coil has its own additionally. This is approximately half of the combined inductivity and produces its own resonance. If the capacitor is larger, this second resonance can be reached. In the 90° case this second resonance cannot be reached because the varactor diodes do not have enough capacity. But in the 45° case, with a large inductivity and a smaller capacitor, the varactor diodes can well reach this second resonance points. In order to not stick to this second resonance point (with its single direction property) the voltage of the varactors should always start from +5V downwards, even if the signal strength is larger with the larger capacitor (e. g. when one of the coils is in perfect direction towards the transmitter).

This second resonance could have been avoided if both coils get their own (larger) capacitor. But that would make frequency adjustment via AFC more complicated because one needs two PWMs for AFC or a small fixed capacitor exactly compensating the difference of the inductivity of each coil.

Practice has shown that this antenna is very selective. While my energy saving lamp transmits at roughly 80 kHz, and with that strong signal confusing commercially available

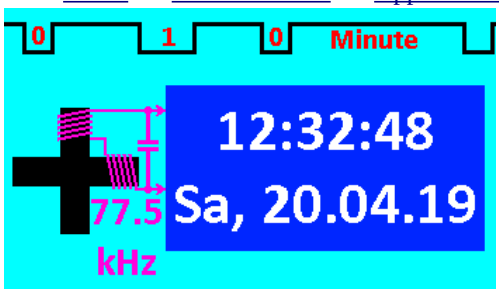
DCF77 receivers so that they do not work in less than 50 cm distance to the lamp, the cross antenna is not sensible for that. Whether

- this has to do with the varactor diodes that allow exact resonance to 77.5 kHz (maladjustment to the lamp's frequency shows a much stronger signal there), or whether
- the bandwidth of the LC circuit is indeed that narrow (has to be, otherwise the 80 kHz would still come in even if adjusted to 77.5 kHz), or whether
- commercially available DCF77 receivers have no N-FET buffer stage but couple the signal to a transistor, by this reducing the high resonance resistance of the LC circuit and increasing its bandwidth, or whether
- those do not have an AFC to exactly adjust the frequency, and can well be far away from 77.5 kHz,

can not be determined exactly, but this effect alone is a good argument for having a home-brewed receiver instead of the cheap mass ware.

The cross antenna is very insensitive to direction changes. The amplitude drop down to 0.35, when in maximum misalignment, is simply compensated by a small change in the AGC voltage that regulates the gain of the receiver.

Because I do not own a mechanical compass (and the one in my Android mobile is a useless equipment here because the term "North" is a very wide field for that equipment) I am not able to provide exact directional data on the cross two antenna versions. Sorry for that.



Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 transistor receiver

2 Transistorized DCF77 receiver amplifier

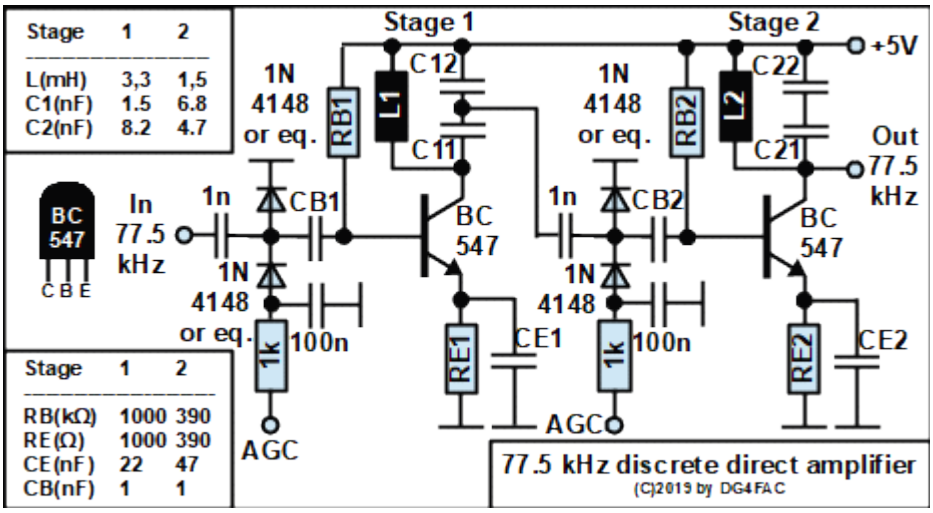
An RF amplifier for DCF77, transmitting on a frequency of 77.5 kHz, has to

- amplify the antenna signal by at least 10,000 fold,
- avoid self-oscillation of the amplifier by regulating its gain (AGC), and
- has to drive the final rectifier stage with enough RF power.

2.1 Amplifier and driver for DCF77 RF

The amplifier has two stages, each equipped with a usual NPN small signal transistor (you can use any available type):

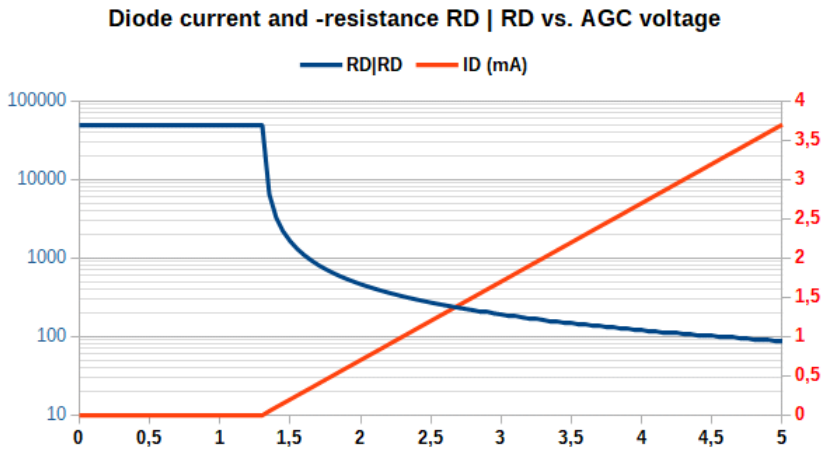
1. The first stage is a voltage amplifier with a resonant LC circuit for 77.5 kHz in its collector. To reduce load influ-



ences from the next stage on the resonant circuit, the capacitor of the LC is divided into a large and a small capacitor and so divided by 4.5 with that capacitive divider.

2. The second stage is a similar voltage amplifier but with a slightly smaller inductivity and larger capacitors. As the next stage is not interfering the LC resonant circuit, no voltage division is made here.

The gain of the two stages is extremely high, due to the very high resistance of the LC circuit in the collector at resonance (approx. 161 kΩ in stage 1 and 70 kΩ in stage 2). And this high gain is frequency-specific. Each stage amplifies the signal by roughly 1,000-fold.



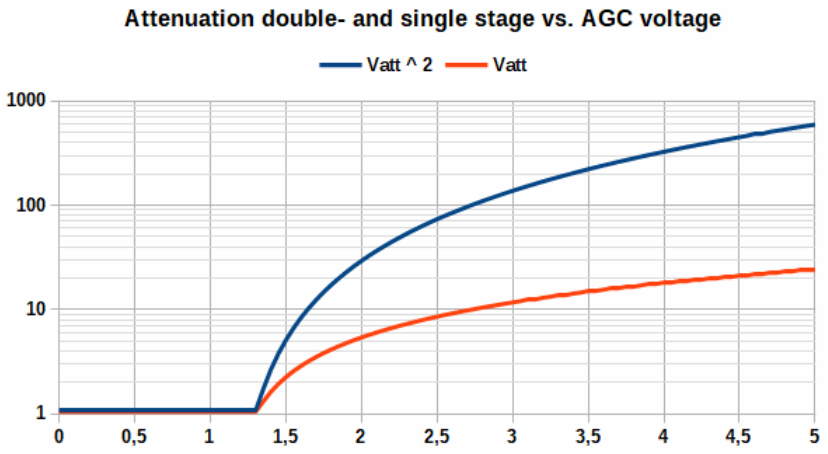
I also tried an additional stage of a similar design. With that stage I had to reduce the gain because of self-oscillation. I tried the diode attenuator as well as reducing the emitter capacitors and increasing the emitter resistors: it is all the same, the applicable gain of stage 3 is of no use, so it does not make any sense to add it.

The diode attenuator on the first and second stage input works as follows. Increasing the current through the diodes (red curve) reduces their resistance, which is

$$R = V_{Diode} / I_{Diode}.$$

At the highest current here, I_{Diode} is (5 - 2 * 0.65) / 1 k = 3,7 mA, so the diode resistance is R_D = 0.65 / 3.7 = 176 Ω. As both diodes are parallel to ground (the upper one direct, the lower one via the 100nF-capacitor) the diodes are parallel and the resistance of the two parallel diodes is 88 Ω. With a capacitor of 1 nF, its capacitive reactance Z_C is Z_C = 1 / 2 / Π / 77500 / 1E-9 = 2,053 Ω.

The capacitor and the two diodes make up a resistor divider that attenuates the signal to the 0.041-fold, or with a factor of 24.4. Both attenuators reduce the gain of the amplifier by the 595-fold.



The diagram shows the attenuation of a single stage (red line) and of the second stage (blue line) of the two-stage amplifier versus the AGC voltage applied. The AGC

voltage reduces the gain of the two stages smoothly (note that the PNP drive stage reaches saturation at 2.8 V, so the curves are theoretical above that!).

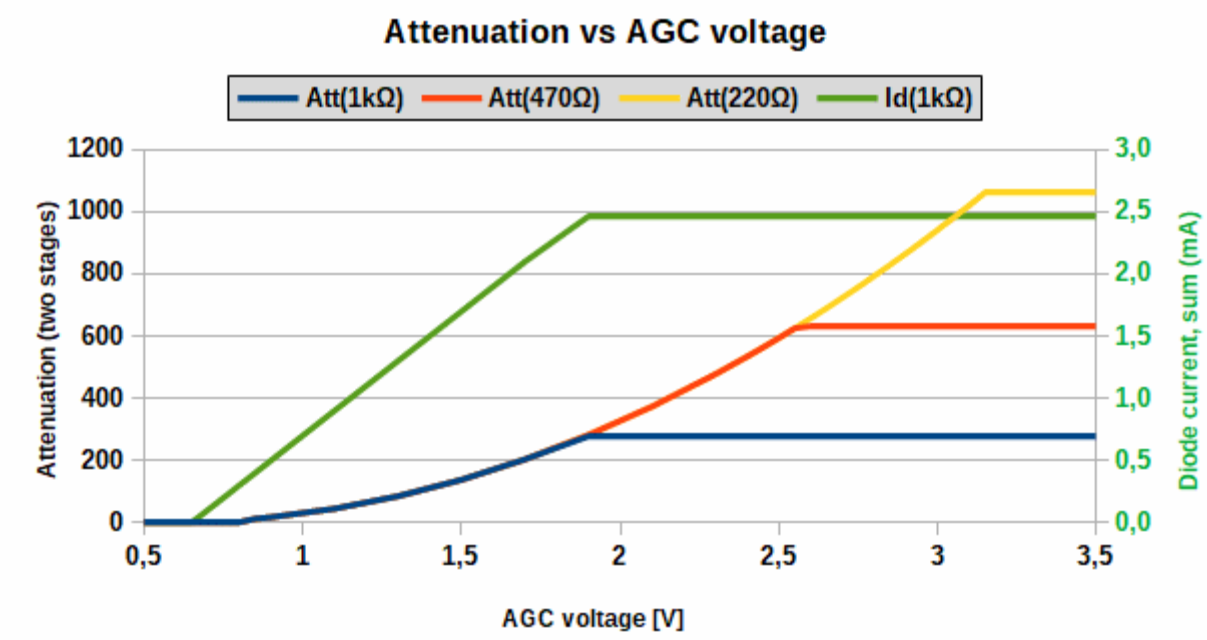
In stage 1 the diode attenuator following adds 1 nF to the C12 capacitor. That reduces the resonance frequency of the LC combination slightly. This effect is calculated in the Libre-Office spreadsheet here. The resonance frequency shifts from 77.8 down to 77.15 kHz, still is within the bandwidth of DCF77 and has no negative consequences.

The same calculation sheet calculates the influence of the emitter-base capacitor of the transistor, in this case the stage 2 capacity. This has positive consequences as it reduces the resonance frequency down from 77.95 to 77.82 kHz. Because the tolerances of the parts used have a much higher influence, this is rather of an academic nature.

The signal of DCF77 is highly amplified, the gain can be reduced by applying current through the diode attenuators. I use a trim resistor which drives the base of a PNP transistor with 1 kΩ on its emitter (with the collector on ground) to get enough diode current. This stage is also required when driving the AGC with a pulse-width modulated signal from an AVR. This allows a fine tuning of the gain.

In a second test configuration I attached the base of the PNP driver to a potentiometer resistor of 100 kΩ, with both ends attached to the operating voltage. With the potentiometer the gain can be adjusted very sensitive. If you use manual adjusting, it can be recommended to reduce the variability of the potentiometer with additional fixed resistors to reduce the sensitivity.

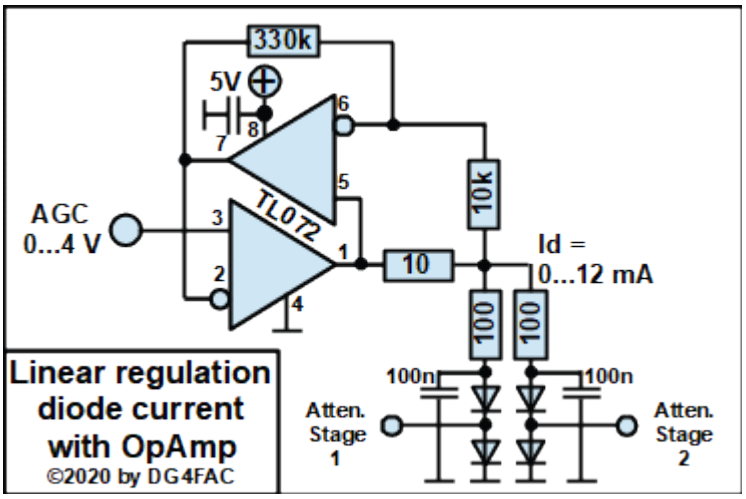
The design of the PNP driver stage with 1 kΩ to plus and two 1 kΩ before the two diodes of the attenuators, reaches saturation when the AGC exceeds 2.5 V: the diode current is not rising any more because the two diode pair currents exceed the driver current through the 1kΩ to plus. The diode current (in the diagram in green, right side scale) is then constant, limiting the possible attenuation.



Because I live in 28 km distance to the DCF77 antenna and so have strong reception signals, the attenuation with the 1kΩ resistors was large enough to reduce the field strength.

If you live even closer to Mainflingen, you might need more attenuation, reduced resistors of 470Ω and 220Ω are shown in the diagram.

Or, you can decide to use this Mercedes-Benz-type driver with two opamps, which drives the diode currents in a super-linear manner (from 0.0 V AGC voltage on), with max. 12 mA diode current and to achieve an attenuation of 1,514-fold with that. But this is the version for the electronics lover only.

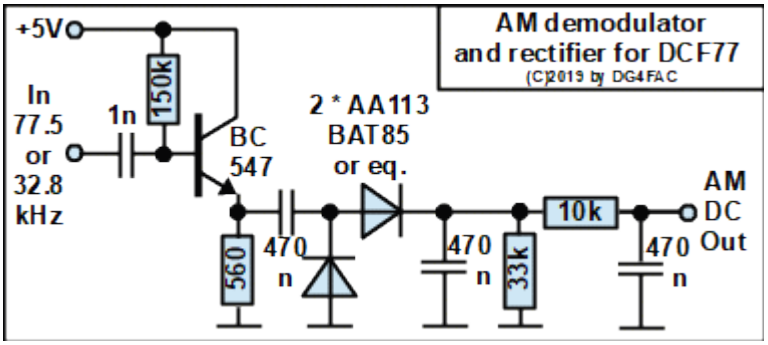


2.2 Rectification

The rectification can be made with a diode rectifier or an ATtiny25 controller (see chapter 2.2.2 for more and chapter 7 for a detailed description).

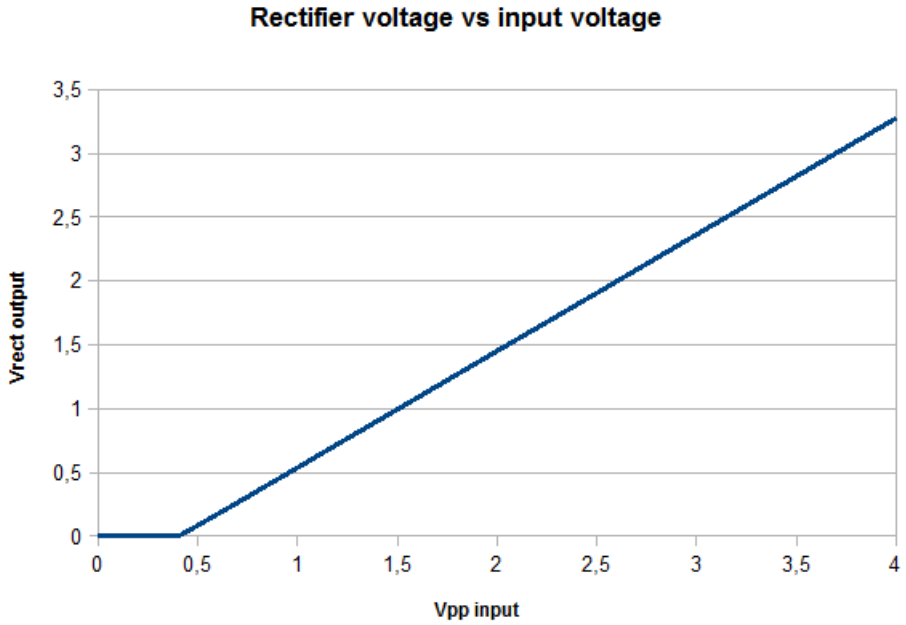
2.2.1 Diode Rectifier

This here is the diode rectifier for the amplitude-modulated RF signal. Two Germanium or Schottky diodes rectify and double the DC made from the RF, with two capacitors of 470 nF. The resistor of 33 kΩ unloads the capacitors during the amplitude drop of the DCF77 signal, with a half-life time of approximately 10 ms. The following RC with R=10kΩ and C=470nF reduces humming of the 77.5kHz signal, and a clean signal, to be fed into an ADC stage of an AVR.



The first stage decouples the diode rectifier from the second stage of the amplifier, so that the resonance circuit of the second stage is not overloaded by the low diode resistances. This stage has no amplification, it just reduces the source impedance.

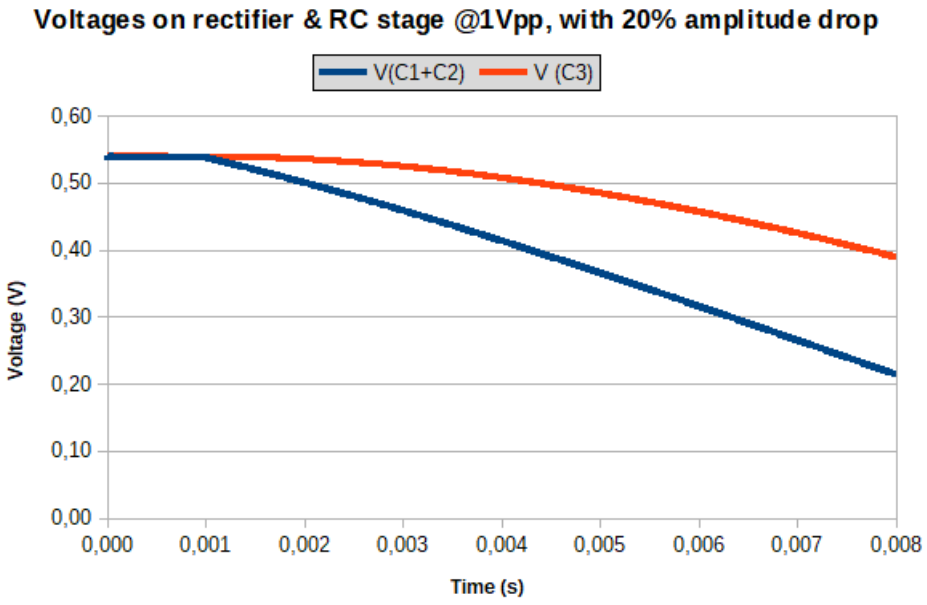
This is the produced DC voltage for different AC voltages. The rectifier does not work below 0.4 Vpp input voltage due to the diodes. But it provides enough DC voltage for normal RF or IF signals.



This is the unload curve of the RC combination with C=470nF and R=33kΩ. With t = 0.69*R*C = 0.01 s it is steep enough to detect the 100 resp. 200 ms long amplitude drops when DCF77 transmits a zero or a one.

In red the delayed drop on the R=10kΩ/C=470nF filter can be seen. It is slightly delayed, but drops down with a similar speed like the voltage on the input.

Calculation of those curves was performed with the OpenOffice spreadsheet here. The sheet **SimRectifier** simulates for a frequency of 77.5 Hz and for the diverse parts of the rectifier and for a selectable resolution. Fields that require an input are with a green background color. It simulates



- the voltages on the two rectifier capacitors (in columns C and D and their sum in column E,
- the drop in amplitude with a selectable level, starting at a selectable time, and
- the voltage on the RC filter output.

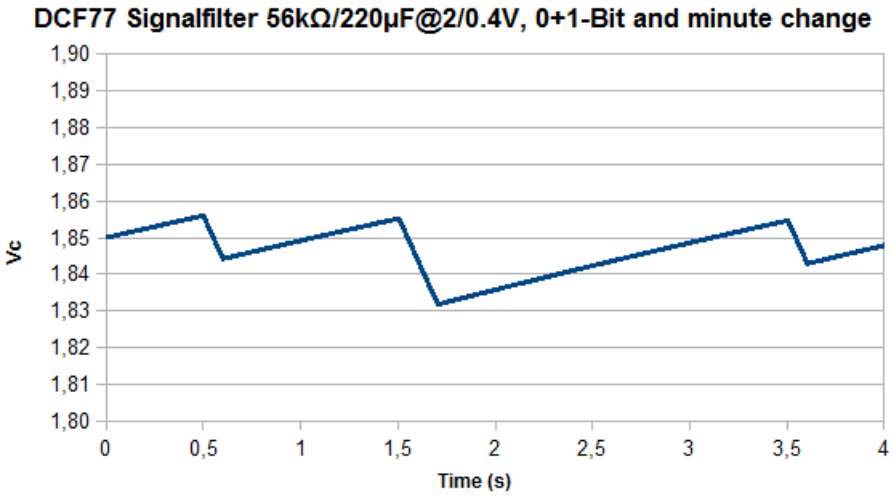
From that simulation the ripple of the voltage on the rectifier capacitors was also calculated. It is below 0.25 mV and remains below one digit of a 10 bit ADC. Only if the capacitor values down to one tenth or the reduction of the resistor down by a factor of 20 yields

one ADC digit. This would increase the amplitude drop speed, but would also decrease the voltage level.

Even though the rectifier RC does not produce humming I added the 10k/470nF RC filter. In practice humming was larger than simulated here, so this filter was necessary for a clean signal.

The rectifier hardware and the calculation tool can also be used for smaller frequencies, e. g. for the 32.768 kHz IF of a superhet. It doesn't work with an IF of 455 kHz or 10.7 MHz, though.

The amplitude drop when receiving zeroes and ones and during the 2-second long missing drop when the minute is over leads to a reduction of the long-term average of the signal, when averaged over a time period of longer than one minute (as done in an AVR or in a long-term RC filter). The following parameters were used in this simulation:



- Signal DC of
  - 2 V without amplitude drop,
  - 0,4 V with amplitude drop,
- averaging by an RC filter of
  - R=56 kΩ,
  - C=220 µF.

  with a time constant of t = 8,5 s.

Such a RC combination would be chosen if the DCF77 signal would be used to adjust the gain of the RF or IF amplifier, e. g. in a TCA440 or for a diode attenuator. In those cases the reception of a zero or a one shall not lead to a relevant gain adjustment.

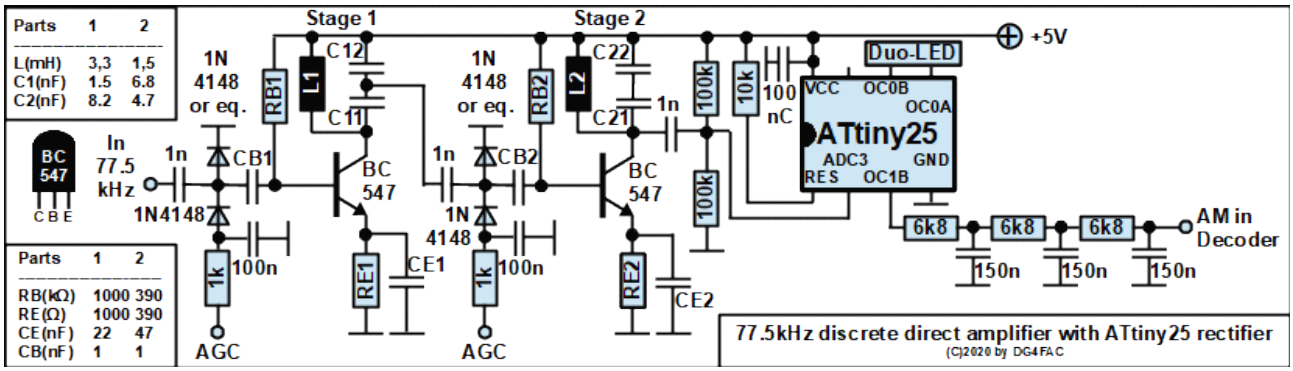
One can see that the average voltage is at 1.85 V (approx. 93% of the 2 V on the input) and differs only by +/-10 mV during a zero or a one. During a minute change, the level change is slightly higher and around +/-25 mV.

From that one can see that long-term averaging (via software or with an RC filter) is an appropriate method.

2.2.2 Rectifier with an ATtiny25

The rectification with an ATtiny25 controller is in detail described in chapter 7. The assembler software for the ATtiny25 can also be found there.

This can directly be attached to the second stage of the amplifier and does not need an emitter follower like the diode rectifier, due to high resistance of the ADC3 input stage in the ATtiny25.



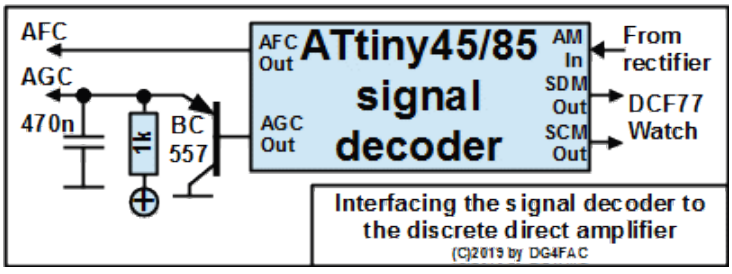
The signal of the second amplification stage is fed to the AD converter of the ATtiny25. This measures the amplitude of the signal with a very high sampling frequency, rectifies it (by subtracting 0x0200 from the 10-bit ADC result and inverting the result if negative = rectification), detects the maximum from a large series of measurements (256), averages those over 16 maxima, divides it by 2 and runs the 8-bit TC1 with that in PWM mode.

TC1 produces a pulse-width modulated signal that is averaged by a three-stage RC filter, that produces a stable DC voltage with a very low ripple of between 0 and 5 mV, depending from the amplitude on the ADC3 input. This DC is transferred to the decoder controller's AM in that derives a) the DCF77 bits and b) the AGC and AFC signals from that.

Do not try to integrate the functions that the tn25 rectifier performs into the decoder's controller: the very fast ADC sampling rate eats up the complete clocking of the controller with 8 MHz, so that there is no time left for the complex functions of the decoder.

The Duo-LED that can be attached to the rectifier controller (red/yellow or red/green 2-pin-Duo-LED, red anode to OC0B, resistor of 270Ω) can serve as a signal strength indicator (with increasing green brightness, if signal strength is too large to regulate the amplifier it turns fully green). If you do not need that you can switch this off by changing the software's configuration.

2.3 Automatic regulation

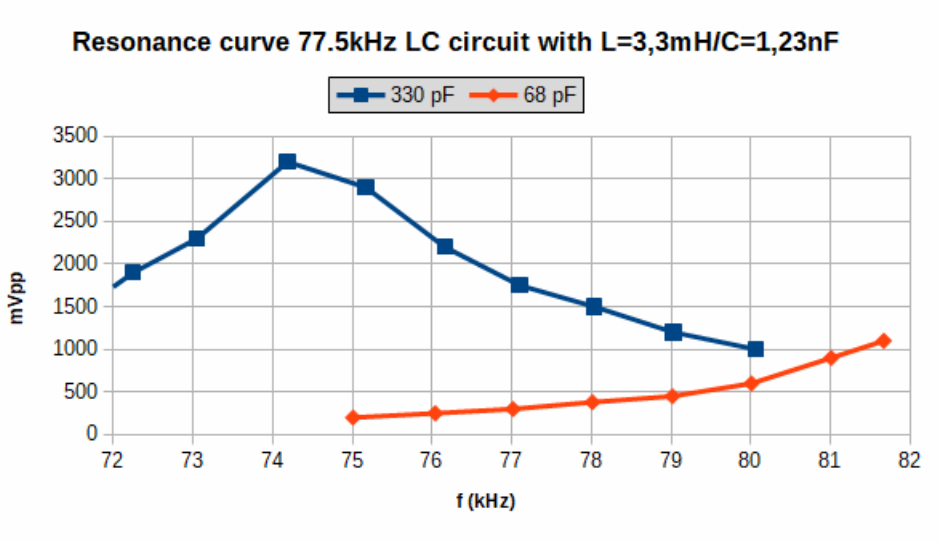


For the automatic regulation of the frequency (AFC) and of the gain (AGC) as well as for the complete decoding of the DCF77 signal, including a serial interface for transmitting the data to another controller that displays date and time received, a controller of the type ATtiny45 has been developed. This reads the rectified voltage, analyzes the voltages, derives controls and adjusts AFC and AGC via two PWM channels and, by detection and checking of voltage drops, decodes the zeroes and ones received from DCF77.

The description of that TN45 controller can be found here.

2.4 The pass-band curve of LC filters

In order to determine the filter properties of the two LC resonance circuits in the collector of the amplifier stages a generator was designed and built that allows to measure those filters around 77.5 kHz. It produces sine waves with adjustable frequencies between 70 and 80 kHz. The amplitude of the oscillator is 4 Vpp at an operating voltage of 5 V.



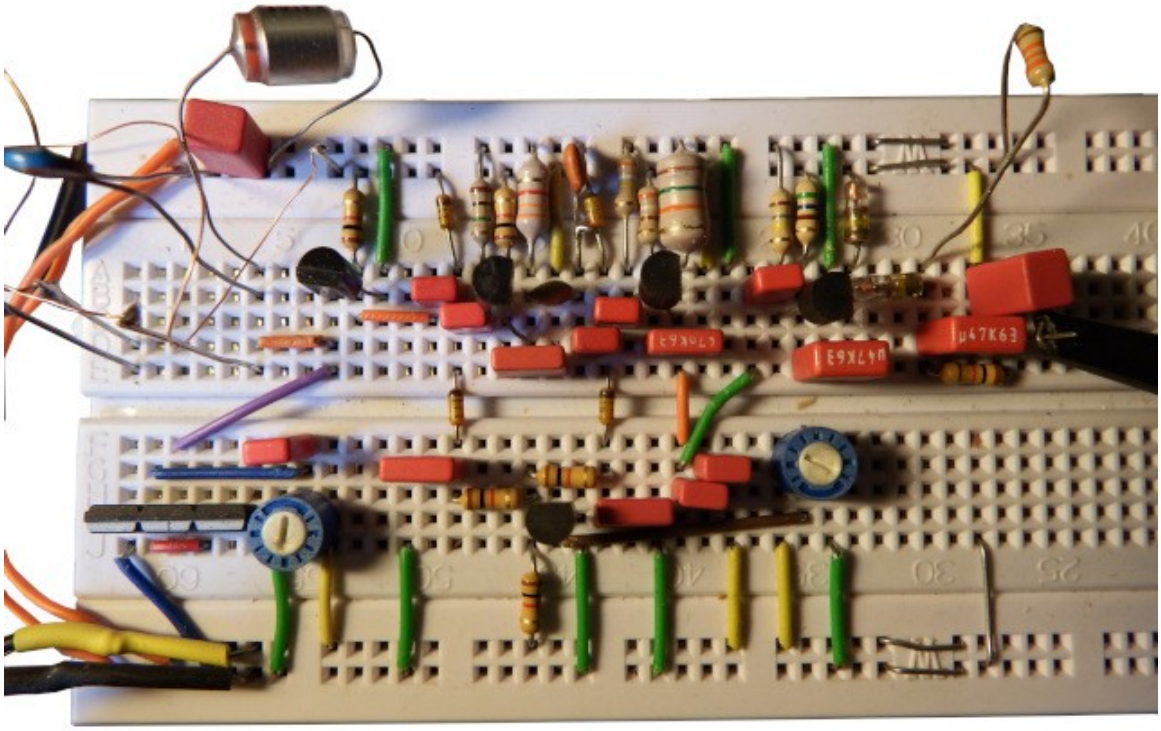
Those are the filter curves with different coupling capacitors. The maximum of resonance with a 330pF capacitor is not at 79 kHz (as calculated) but by 5 kHz lower at 74 kHz. This is, on one hand, due to the coupling capacitor (when fully in parallel 70.2 kHz), but is also due to straying effective values of L (5%) and C (10%).

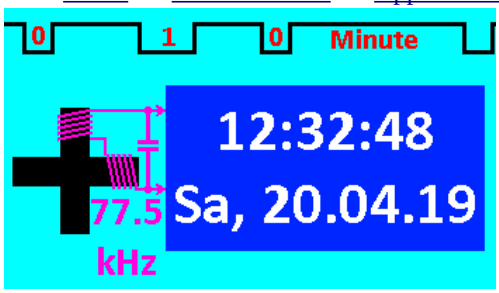
The curve is rather broad and not very steep. It covers +/-2.5 kHz for the amplitude drop down to half (3 dB).

When decreasing the coupling capacitor to 68 pF (with a Z_C of approximately 30 kΩ) the resonance frequency increases. Reactance of the LC is larger than 11 kΩ at resonance.
Due to the high bandwidth of the LC circuit it does not make much sense to adjust the two LC circuits for 77.5 kHz. Compared to a simple resistor in the collector, an un-adjusted LC circuit is an immense advantage. Especially RF far away of the 77.5 kHz (short wave, 90 kHz power supplies, etc.) are not amplified.

Those who need it more narrow, because their power supply, energy saving lamp or old valve TV transmits at 80 kHz, can use a superhet with a more narrow filter, as also described on this web page.

This is the transistor amplifier on the breadboard. Make sure that the cross antenna is at least in a distance of 15 to 20 cm of the inductivities to avoid feedback and self-oscillation.

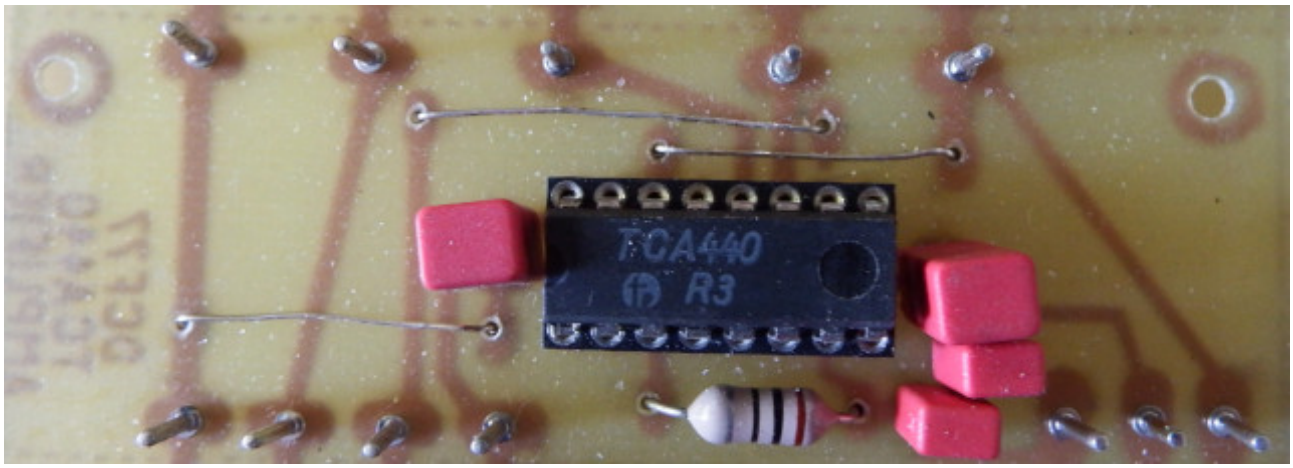




Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 TCA440 amplifier

3 A DCF77 receiver direct amplifier with a TCA440



Lots of parts are needed for a gain-adjustable receiver amplifier for 77.5 kHz (see here). So using an integrated circuit can reduce the number of necessary parts. As currently AM radio reception are slowly dying out, the production of such ICs is very limited. There are only ICs in production, that

- include AM and FM receivers in one chip (e. g. CD2003 and many more). For DCF77 the FM part is unnecessary and consumes current for nothing,
- offer very primitive amplifiers without gain regulation, unusable for changing voltage levels (e. g. TA7642 ZN414), or that
- are unavailable in normal electronic shops (such as the MAS6181 or the TDA1572), or that
- require a large number of external capacitors to work (e. g. SA602/612).

The way out from that dilemma is to use ICs that are not produced any more but are still available in specialized shops. Old fashioned AM superhet ICs fit our needs for DCF77 reception perfectly. For a direct receiver concept, only the internal pre-amplifier, oscillator and mixer is unnecessary and has to be disabled. The integrated IF amplifier, equipped with a gain control input pin, can be used to amplify the 77.5 kHz directly with enough gain.

Here the ancient TCA440 is used. It needs only a few external components. It is still available in specialized electronic shops, search for it on the internet.

3.1 The TCA440

The TCA440 integrates, in its 16 pin DIP case

1. it provides a differential amplifier on its pins 1 and 2, that can be used as a RF input stage, its gain can be reduced applying positive voltages on pin 3,
2. it receives a symmetric oscillator frequency on its pins 4 and 5 and couples it back on pin 6,
3. it has an additive mixer on board, with its outputs on pins 16 and 15,
4. integrates a gain regulated IF amplifier with symmetric inputs on its pins 12 and 13 and its output on pin 7,
5. has a gain amplifier stage on pin 9, that regulates the IF gain and has an output for attaching an instrument to display field strength on pin 10.

The pins 14 (plus) and 8 (GND) provide the operating voltage. The IC integrates

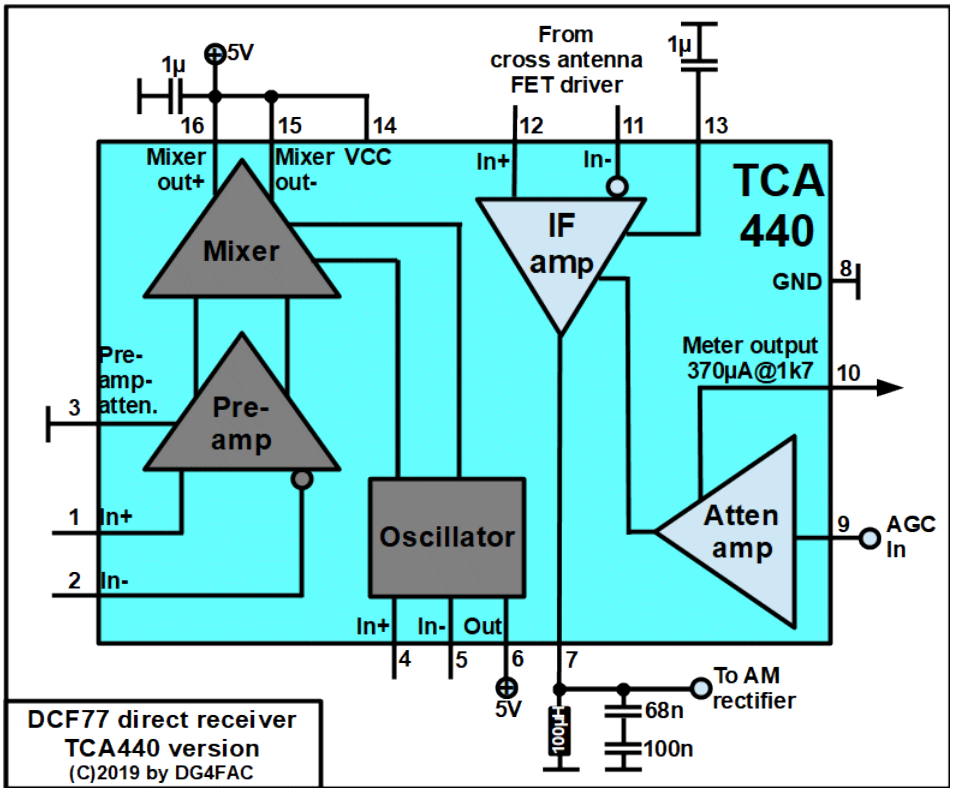
- 34 transistors,
- 21 diodes, and
- 53 resistors.

Copying of this using discrete parts would be an extreme effort, filling a Euro sized board.

Additional data, the internal structure and applications as medium wave receiver can be seen in the data-sheet by Siemens.

3.2 Schematic for a DCF77 direct amplifier with TCA440

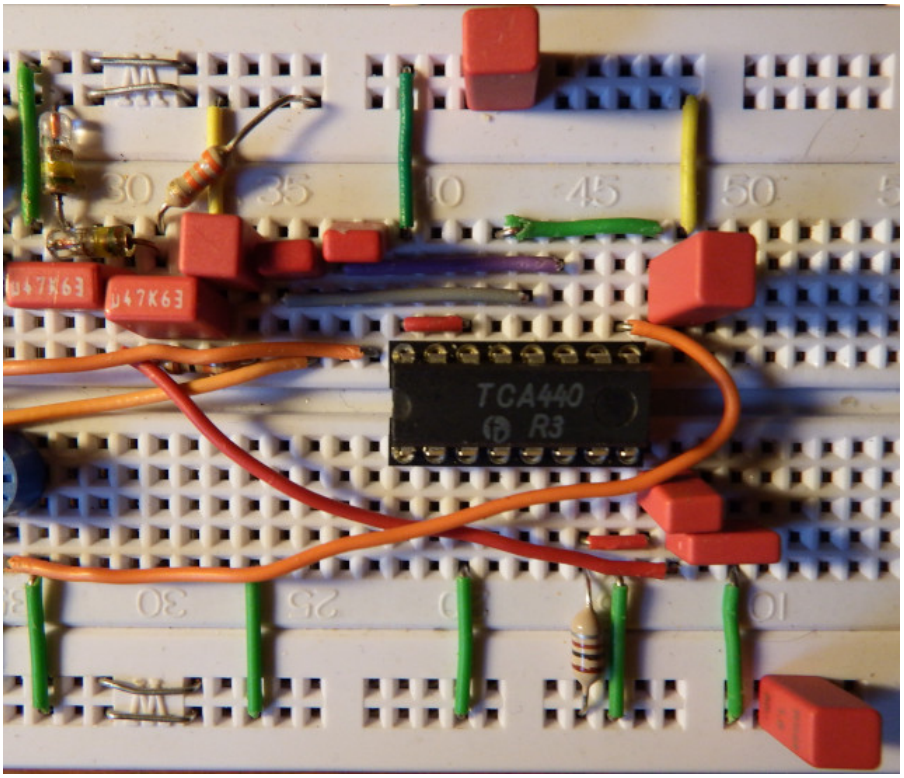
The received signal from the cross antenna, with frequency regulation and buffered with an N-FET from here is symmetrically applied to the IF amplifier of the TCA440. Pre-amp, oscillator and mixer are switched off and are not used here.



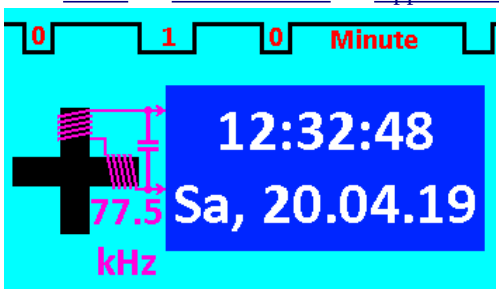
The output of the IF amplifier goes to a low-resistance (approx. 2 kΩ) resonance circuit. The AM-RF is then rectified with a double diode rectifier already described here.

Frequency (of the cross antenna receiver) and gain regulation can be made using trim resistors or with an ATtiny45 controller as described here.

Note that the TCA440 has a gain of 1.0 if you leave the AGC input open. To wake-up the TCA440, apply a lower voltage to the AGC pin, and he wakes up and gains.



That is how the amplifier and rectifier is mounted on a simple breadboard.



Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 superhet 32.768 kHz IF

4 DCF77 superhet receiver with xtal filter

Those who want to have the Mercedes of a DCF77 receiver, home-brew themselves a superhet with a crystal filter! The DCF77 receiver RF signal (e. g. from a cross antenna) of 77.5 kHz is

1. amplified in a pre-amp, then
2. mixed with an oscillator signal to form a different frequency (here: 32.768 kHz), which is then
3. filtered with an LC circuit and a crystal, after that
4. amplified in an Intermediate Frequency (IF) amplifier, its output then
5. is again filtered with an LC circuit and rectified in a two-diode stage as shown here with the generated DC filtered in an RC stage, and then
6. the DC is measured, checked and decoded in an ATtiny45 controller, with time and date information serially transmitted to
7. be received, decoded and displayed on an ´LCD.

With that, you can be absolutely shure that no one besides you (and me, of course) has such a home-brewed Mercedes in its garage: it is unique and perfect.

4.1 Advantages of a superhet over any other concepts

Superhets are better than direct receivers because the Intermediate Frequency (IF) can be filtered with a small bandwidth (here: of a few Hz). So any interferences from other sources (random noise, strong RF from nearby short wave transmitters, from switching power supplies or switched power saving lamps as well as all other electromagnetic fields can be completely sorted out and eliminated. So it is possible to receive the DCF77 signal in a very far distance and in a noisy environment, where other receivers do not work.

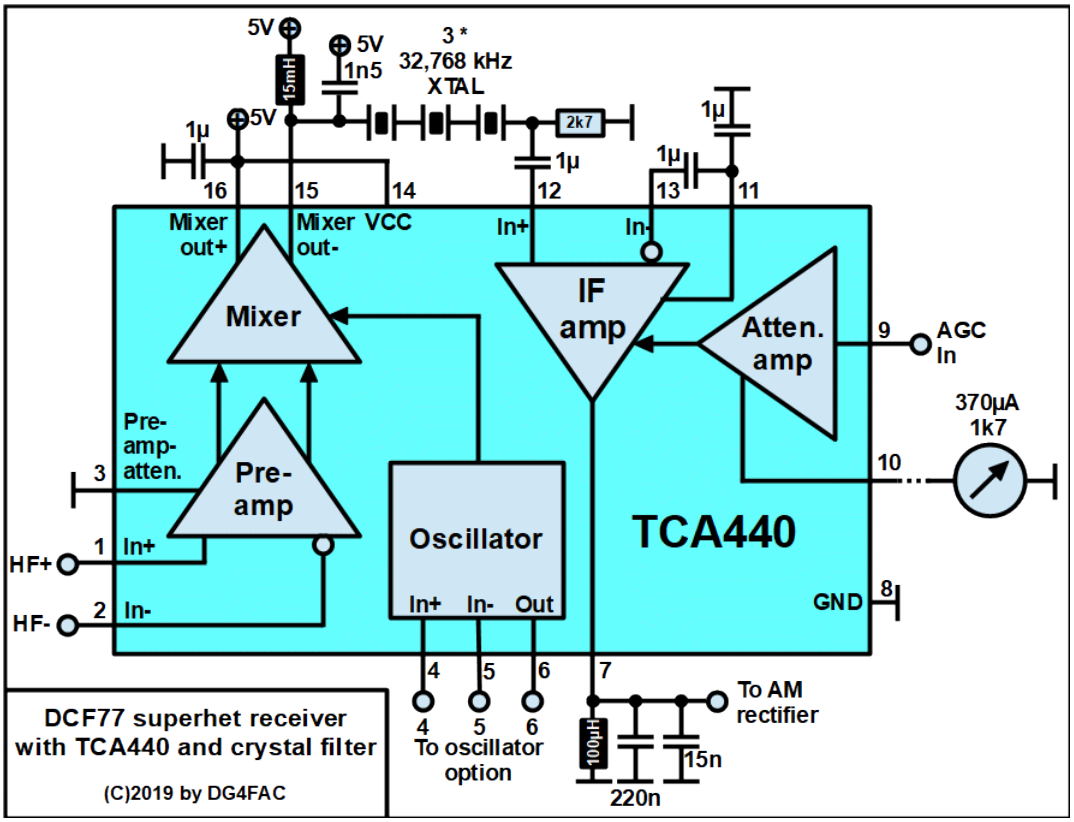
As the IF amplifier works on a different frequency, the IF signal can be amplified without getting self-oscillation. This also makes it more sensitive than direct receiver concepts.

4.2 The superhet schematic

This is the schematic of the Mercedes.

The symmetric output signal from the cross antenna's FET buffer stage is fed into the pre-amplifier stage of a TCA440 on its pins 1 and 2. The gain reduction of the pre-amp stage on pin 3 is turned off. IF you are in the absolute near-field of DCF77 (say: less than 10 km) you can apply 1 or 2 V here to not drive the mixer stage into an overload.

On the oscillator pins 4 and 5 the oscillator signal of 77.5 + 32.768 = 110.268 kHz is supplied. This signal ois either generated in an LC circuit (by using the oscillator output signal on pin 6, see here) or with an xtal oscillator (see here).



The mixer products are filtered with a LC circuit made of a fixed coil of 15 mH and a capacitor of 1.5 nF. To filter the only product of interest, the 32.768 kHz, one or up to three 32kHz xtals follow. The properties of such a crystal filter are in detail shown here.

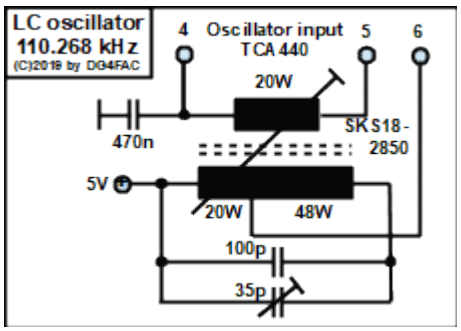
The output of the crystal filter is fed into one of the two symmetric input pins (pin 12) of the IF amplifier, with the other input on pin 13 being blocked to ground potential via a 1µF capacitor.

The emitter output of the IF amplifier on pin 7 is connected with a second LC combination with L=100µH and two parallel capacitors of 220 nF and 15 nF. The signal is then fed into a 2-diode rectifier and RC filter stage to yield the amplitude as DC. This is further measured and analyzed in a controller as described here. The superhet comes in two variations: with the oscillator signal

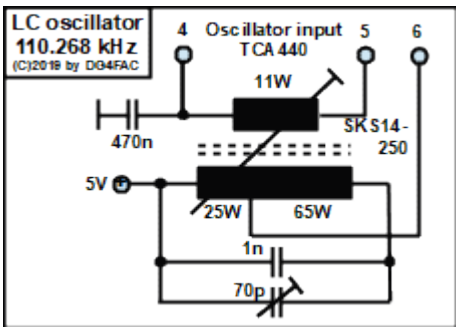
1. produced by a LC combination, or
2. with a crystal oscillator and rectangle-to-sine filter.

4.2.1 TCA440 with the internal LC oscillator circuit

If you want to use the built-in oscillator in the TCA440 the following is necessary. Prepare an 18mm ferrite core with an AL value of 2,850 nH per winding[2]. The core can be trimmed with a screw or with the trim capacitor to 110.268 kHz. Use a frequency counter or the rectified DC to adjust.



An alternative to that would be to choose a 14-mm ferrox cube core with an AL of 250 nH/w^2. The components change slightly, with a trimming range from 108.4 to 112.0 kHz.



4.2.2 TCA440 with an external oscillator

The previous documents on this had an error, the oscillator now works fine.

4.2.2.1 Concept using a crystal oscillator as basis

LC resonance circuits are slightly temperature sensitive, so that its frequency has to be adjusted from time to time. Under long term operation and with some aging of parts, this has some disadvantages. An alternative to the LC would be to generate the oscillator frequency from a xtal-controlled base frequency, by dividing that with a fixed rate.

Of course, there are no 110.268 or 44.732 kHz on the market. This solution here uses a xtal clocked AVR as a rectangle generator.

An AVR, here an ATtiny25, is clocked by an external xtal oscillator. Its timer/counter 0 works as a divider (in CTC mode), divides the clock frequency by a fixed rate and toggles the compare outputs A and B on compare match. By starting outputs A and B with different start conditions, it produces counter-phased rectangles. The rectangles are filtered with an RC network, that yields sine waves that can drive the TCA440's oscillator inputs with a symmetric sine wave signal.

4.2.2.2 Selecting the crystal frequency

Digital dividers can only divide by integer values. Therefore the xtal frequency, divided by the divider, has to fit nearest to the desired TCA440 oscillator frequency. To find the nearest fit I have listed all available xtals in a spreadsheet and did some calculations with those.

In the table the higher (110.268 - 77.5 = 32.768 kHz as well as the lower (77.5 - 44.732 = 32.768 kHz are considered. The divider is calculated, the divider determined and the factually generated frequency f is as well as its absolute deviation in percent and in +/- Hz is listed. The table is available here as OpenOffice file.

The table holds a second sheet, listing xtal oscillators only. There are fewer, but also some frequencies that are not available as discrete xtal. The output of the crystal oscillator is connected to the ATtiny25's XTAL1 pin. The CLKOUT fuse can, but must not be activated. The calculation sheet "xtal_oscillator" lists that. In this mode the ATtiny25 can be operated with 5 V, the reduction of the operating voltage is unnecessary.

Please note that the divider toggles the OC0 pins, so that two toggles are necessary for one wave. The frequency therefore is half of the compare value (+1).

Xtal (MHz)	77.5 + 32.768 = 110.268 kHz				77.5 ∝ 32.768 = 44.732 kHz			
	Divider	f is (kHz)	Delta %	Delta (Hz)	Divider	f is (kHz)	Delta %	Delta (Hz)
1.843200	8	115.200	4.47	4932.0	21	43.886	1.89	-846.3
2.000000	9	111.111	0.76	843.1	22	45.455	1.62	722.5
2.097152	10	104.858	4.91	-5410.4	23	45.590	1.92	858.3
2.457600	11	111.709	1.31	1441.1	27	45.511	1.74	779.1
2.500000	11	113.636	3.05	3368.4	28	44.643	0.20	-89.1
3.000000	14	107.143	2.83	-3125.1	34	44.118	1.37	-614.4
3.072000	14	109.714	0.50	-553.7	34	45.176	0.99	444.5
3.276800	15	109.227	0.94	-1041.3	37	44.281	1.01	-450.9
3.579545	16	111.861	1.44	1592.8	40	44.744	0.03	12.3
3.686400	17	108.424	1.67	-1844.5	41	44.956	0.50	224.1
3.686411	17	108.424	1.67	-1844.1	41	44.956	0.50	224.2
3.932160	18	109.227	0.94	-1041.3	44	44.684	0.11	-48.4
4.000000	18	111.111	0.76	843.1	45	44.444	0.64	-287.6
4.096000	19	107.789	2.25	-2478.5	46	44.522	0.47	-210.3
4.194304	19	110.376	0.10	108.4	47	44.620	0.25	-111.7
4.433619	20	110.840	0.52	572.5	50	44.336	0.88	-395.8
4.915200	22	111.709	1.31	1441.1	55	44.684	0.11	-48.4
5.000000	23	108.696	1.43	-1572.3	56	44.643	0.20	-89.1
5.068800	23	110.191	0.07	-76.7	57	44.463	0.60	-268.8
5.120000	23	111.304	0.94	1036.3	57	44.912	0.40	180.3
5.200000	24	108.333	1.75	-1934.7	58	44.828	0.21	95.6
6.000000	27	111.111	0.76	843.1	67	44.776	0.10	44.1
6.000000	27	111.111	0.76	843.1	67	44.776	0.10	44.1
6.144000	28	109.714	0.50	-553.7	69	44.522	0.47	-210.3
6.400000	29	110.345	0.07	76.8	72	44.444	0.64	-287.6
6.553600	30	109.227	0.94	-1041.3	73	44.888	0.35	155.7
7.372800	33	111.709	1.31	1441.1	82	44.956	0.50	224.1
8.000000	36	111.111	0.76	843.1	89	44.944	0.47	211.8
8.000000	36	111.111	0.76	843.1	89	44.944	0.47	211.8
8.867238	40	110.840	0.52	572.5	99	44.784	0.12	52.0
9.216000	42	109.714	0.50	-553.7	103	44.738	0.01	5.9
9.830400	45	109.227	0.94	-1041.3	110	44.684	0.11	-48.4
10.000000	45	111.111	0.76	843.1	112	44.643	0.20	-89.1
10.000000	45	111.111	0.76	843.1	112	44.643	0.20	-89.1
10.240000	46	111.304	0.94	1036.3	114	44.912	0.40	180.3
10.700000	49	109.184	0.98	-1084.3	120	44.583	0.33	-148.7
11.000000	50	110.000	0.24	-268.0	123	44.715	0.04	-16.6
11.059200	50	110.592	0.29	324.0	124	44.594	0.31	-138.5
12.000000	54	111.111	0.76	843.1	134	44.776	0.10	44.1
12.000000	54	111.111	0.76	843.1	134	44.776	0.10	44.1
12.288000	56	109.714	0.50	-553.7	137	44.847	0.26	114.7
12.750000	58	109.914	0.32	-354.2	143	44.580	0.34	-151.6
14.000000	63	111.111	0.76	843.1	156	44.872	0.31	139.8
14.318000	65	110.138	0.12	-129.5	160	44.744	0.03	11.7
14.745600	67	110.042	0.21	-226.2	165	44.684	0.11	-48.4
15.000000	68	110.294	0.02	26.1	168	44.643	0.20	-89.1

Xtal (MHz)	77.5 + 32.768 = 110.268 kHz				77.5 ∞ 32.768 = 44.732 kHz			
	Divider	f is (kHz)	Delta %	Delta (Hz)	Divider	f is (kHz)	Delta %	Delta (Hz)
16.000000	73	109.589	0.62	-679.0	179	44.693	0.09	-39.3
18.000000	82	109.756	0.46	-511.9	201	44.776	0.10	44.1
18.432000	84	109.714	0.50	-553.7	206	44.738	0.01	5.9
20.000000	91	109.890	0.34	-377.9	224	44.643	0.20	-89.1

When mixing the 77.5 kHz input signal with the higher frequency (+32.768 = 110.268 kHz) the 15 MHz xtal has the smallest deviation (0.02%, +26.1 Hz). The crystals 5.0688 and 6.4 MHz deviate by 0.07% or 77 Hz and the xtal 4.194304 MHz by 0.1% or 108 Hz. When mixing with the lower frequency (77.5 - 32.768 = 44.732 kHz) the 9.216 MHz- crystal fits best, with 0.01% or 5.9 Hz deviation.

When mixing with 44.732 kHz the first harmonic (89.46 Hz) is in the wider range of the input frequency. Therefore interferences cannot be excluded, therefore the 15 MHz xtal was chosen. This deviates by 26.1 Hz upwards.

4.2.2.3 Rectangles to sine waves

Any ATtiny has an 8-bit counter/timer with OC0A and OC0B output. The two pins can generate a symmetric output signal: OC0B generates the opposite signal by starting with a high instead of a low port-bit. So, the oscillator signal can be fed symmetrically to the oscillator input of the TCA440.

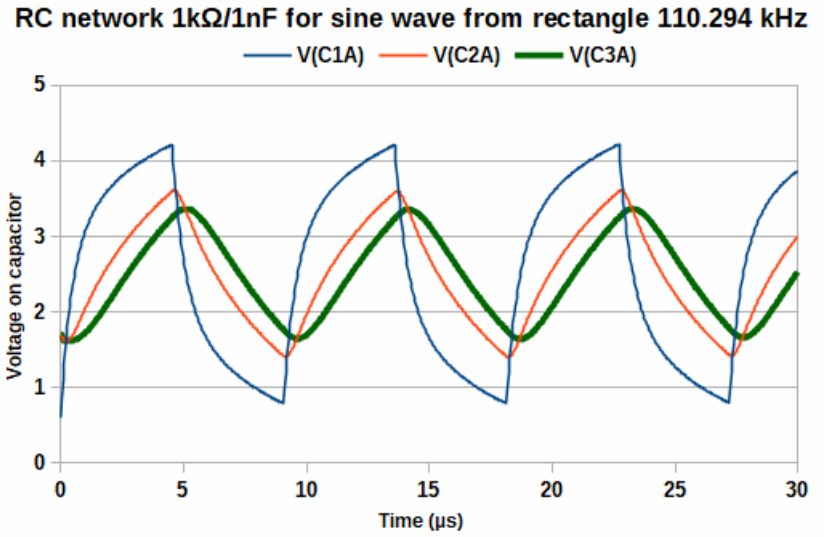
On both outputs, OC0A and OC0B, rectangular signals are made. Using those rectangles for mixing would have adverse consequences, as rectangles consist of all uneven harmonics of the base frequency. It is better if those harmonics are filtered off by use of a three stage RC network.

The calculation spreadsheet OpenOffice file here has a sheet named "Oscillator_coupling", where I played with different RC combinations. To have a large-enough signal the filter should not damp the base frequency too much, but the third (and beyond) harmonic.

To limit the number of components three RC filter stages have been combined. Finally I selected a combination of 1kΩ and 1nF. The loss of amplitude is limited and the harmonics are well suppressed with that.

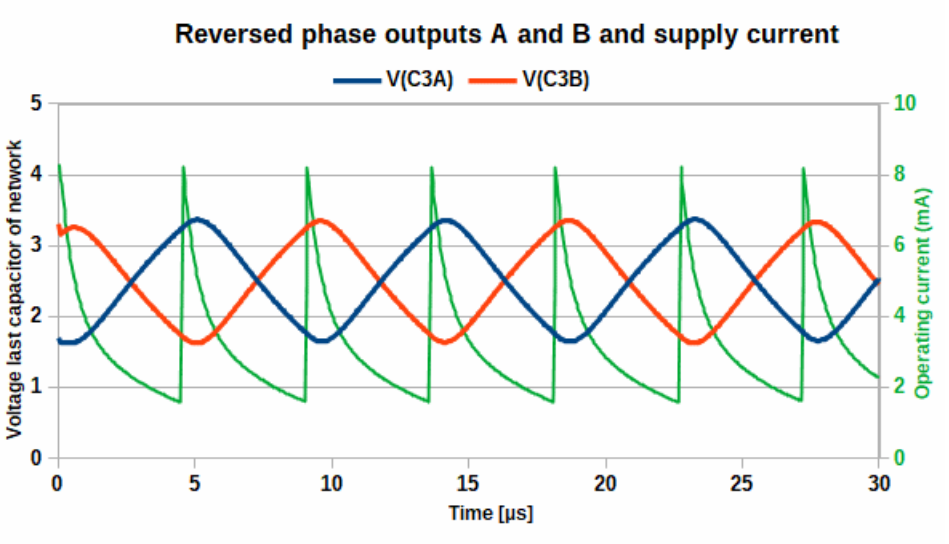
This displays the filter effect of the three stages, as calculated with the spreadsheet.

The first stage (V(C1), blue curve) still is nearly fully reaching the operating voltage limits. In the second stage (V(C2), the red curve) the amplitude swing is smaller and the form is nearer to a sine



wave. In the third stage (V(C3), green curve) the amplitude loss is lower and the wave is a nearly perfect sine.

Displayed above is only one signal, the second is reversed, as can be seen in this picture. This includes, in green and on the right side scale, the operating supply currents through the PWM output pins during these phases. These are roughly 3.2 mA, but reach slightly above 8 mA in the peaks.

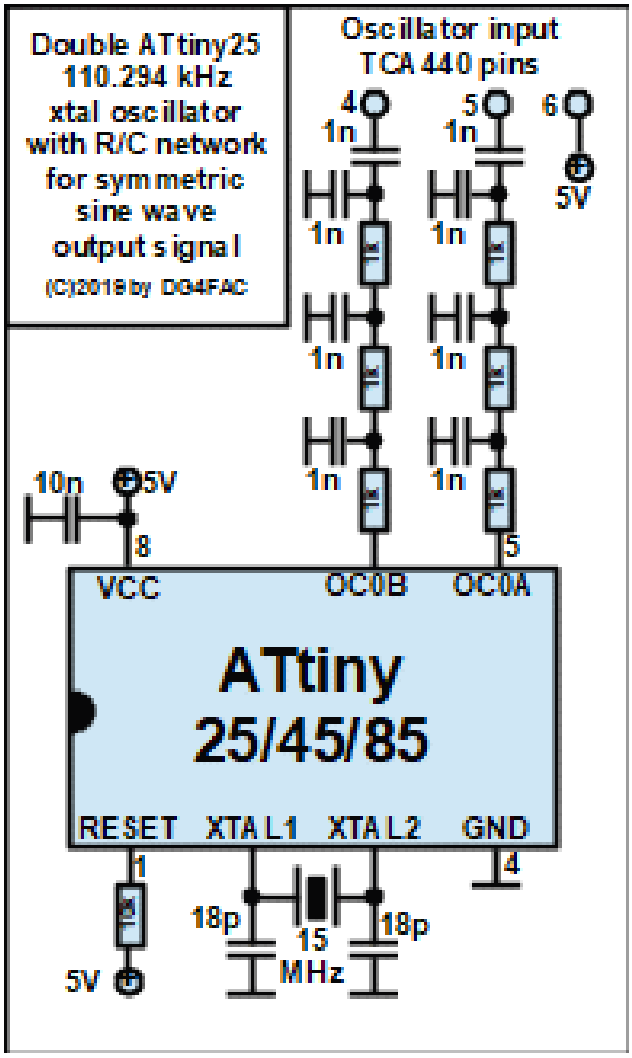


With that signal mixing can be made.

4.2.2.4 The schematic with a discrete crystal

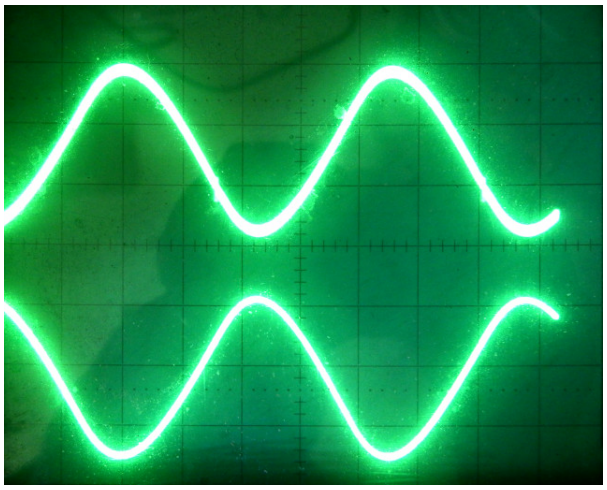
The schematic is rather simple: the timer outputs OC0A and OC0B generate the reversely clocked rectangle of 110.294 kHz, to be filtered in three RC stages. The oscillator inputs on pin 4 and 5 of the TCA440 receive that signal.

The xtal of 15 MHz is connected to the XTAL inputs, each with a capacitor of 18 pF to GND.

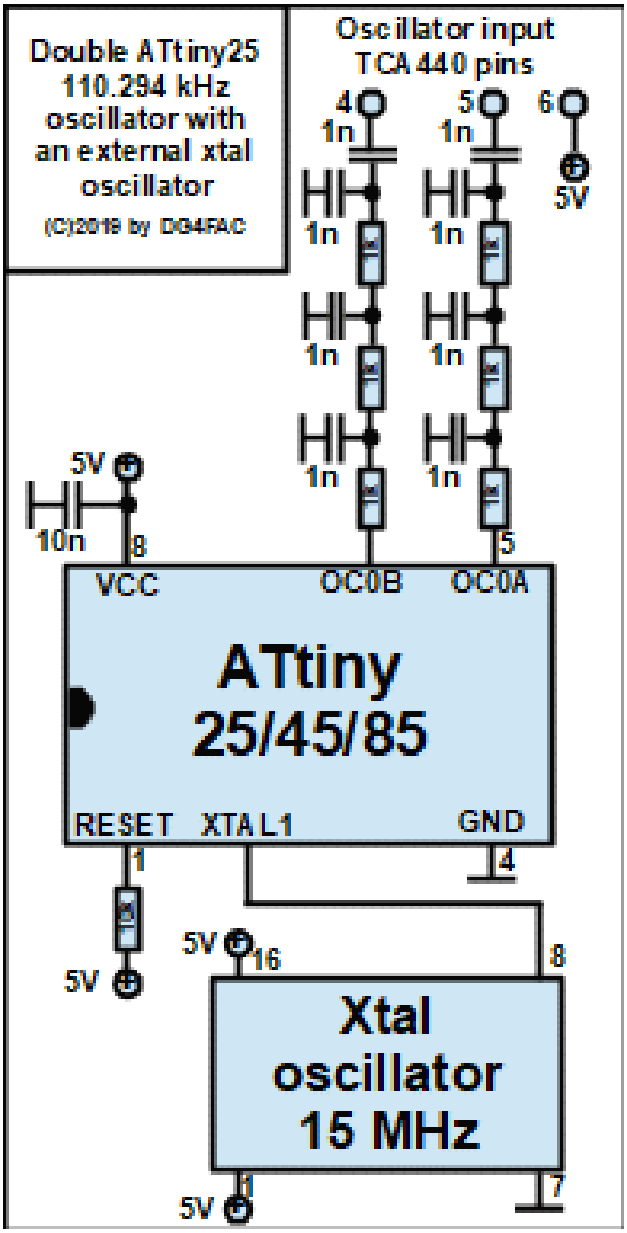


4.2.2.5 The schematic with an integrated xtal oscillator

This is the schematic using an integrated crystal oscillator. That works at 5 V operating voltage and does not have any other limitations like above described. Even though this type of xtal oscillators produce a horrible rectangular signal (anything else than a steap up and down), it works perfect with an ATtiny25.



These are the two sine waves of the two generated signals. Looks clean.



4.2.2.6 Software for the ATtiny25

The software for the ATtiny25 consists of a few lines assembler:

1. The two output pins OC0A and OC0B are configured as outputs.
2. The port register of OC0A is cleared, the one for OC0B is set to one (reversed signal)./li>
3. Both compare values are set to the divider factor (divider minus 1).
4. In the control port TCCR0A of timer TC0 the CTC mode is set and both output pins are defined to toggle on compare match.
5. In control port TCCR0B the timer is started with a prescaler value of 1.
6. The sleep mode of the controller is set to idle mode, the SLEEP instruction is executed and the controller is not needed any further.

The source code is listed here and can be downloaded in assembler format here.

;

```
; *********************************
; * Xtal oscillator for TCA440    *
; * 15 MHz ==> (77.5+32.768 kHz)  *
; * (C)2019 by DG4FAC             *
; *********************************
;
.nolist
.include "tn25def.inc" ; Define device ATtiny25
.list
;
; *********************************
;     H A R D W A R E
; *********************************
;
; Device: ATtiny25, Package: 8-pin-PDIP_SOIC
;
;                 _________
;          1 /         |8
; RESET o--|RESET  VCC|--o +5 V
; XTAL1 o--|PB3    PB2|--o
; XTAL2 o--|PB4    PB1|--o Osc out -
;   0 V o--|GND    PB0|--o Osc out +
;          4|_________|5
;
; *********************************
;   F I X E D   C O N S T A N T S
; *********************************
;
.equ clock = 15000000 ; 15 MHz
.equ fosc = 77500+32768 ; Added
.equ divider = (clock+fosc)/(fosc*2)
.equ cCtc = divider - 1 ; CTC value
;
; *********************************
;        R E G I S T E R S
; *********************************
;
.def rmp = R16 ; Multipurpose register
;
; *********************************
;  M A I N   P R O G R A M   I N I T
; *********************************
;
.cseg
.org 000000
;
Main:
  sbi DDRB,DDB0 ; PB0 direction output
  cbi PORTB,PORTB0 ; Clear OC0A output
  sbi DDRB,DDB1 ; PB1 direction output
  sbi PORTB,PORTB1 ; Set OC0B output
  ldi rmp,cCtc ; Write CTC value
  out OCR0A,rmp ; to compare register A
  out OCR0B,rmp ; and B
  ldi rmp,(1<<WGM01)|(1<<COM0A0)|(1<<COM0B0) ; CTC mode, toggle OC0A
  out TCCR0A,rmp ; in TC0 control port A
  ldi rmp,(1<<CS00) ; Prescaler = 1
  out TCCR0B,rmp ; in TC0 control port B
  ldi rmp,1<<SE ; Sleep enable, idle mode
  out MCUCR,rmp
Loop:
```

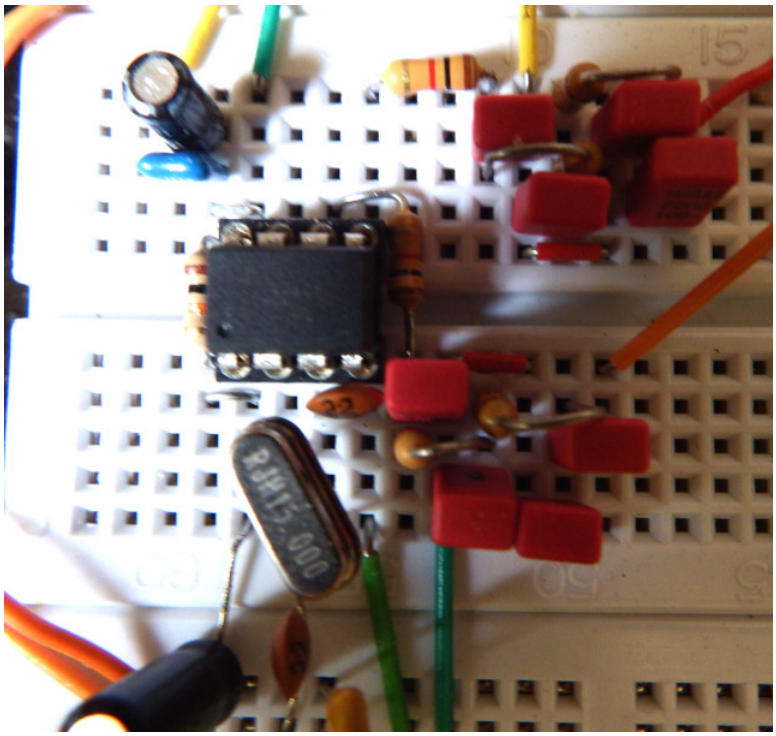
```
  sleep ; Go to sleep
  rjmp loop
;
; End of source code
```

4.2.2.7 Fuses of the ATtiny25

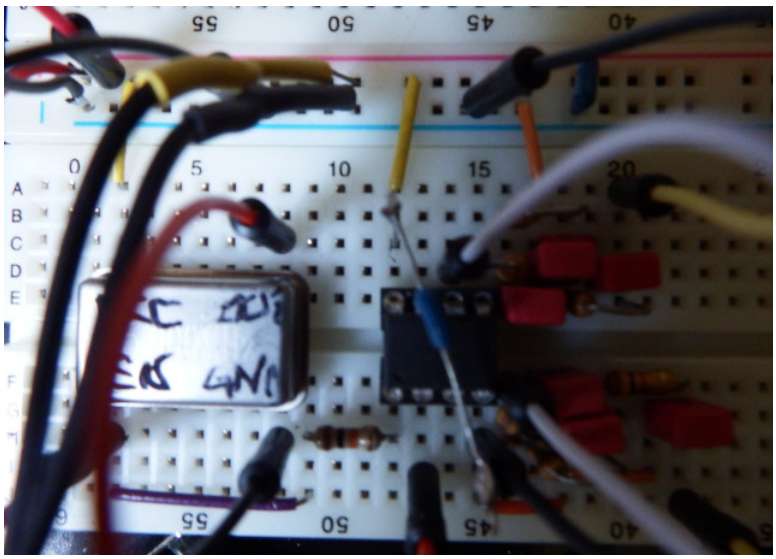
Prior to or after programming the flash the fuses of the ATtiny25 have to be set to work with the external xtal or xtal oscillator. The following fuses have to be set with a discrete crystal:

1. CLKDIV8 has to be disabled.
2. The clock frequency has to be set to an external oscillator of more than 8 MHz.

4.2.2.8 Mounting the xtal sine wave generator



This is the sine wave generator on a breadboard, here with a discrete crystal. The six capacitors right to the ATtiny25 form the three RC networks for sine wave filtering.



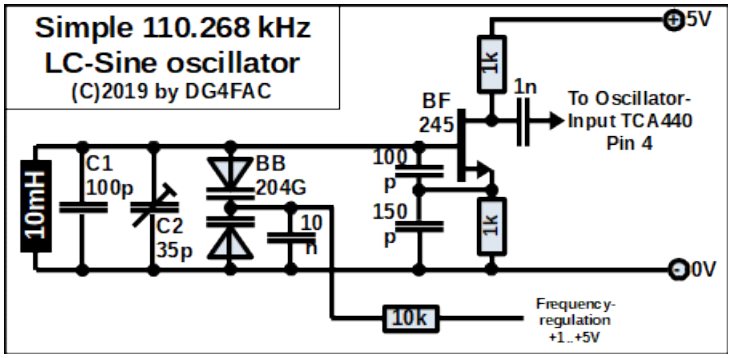
This is the version with the integrated xtal oscillator.

4.1.3 LC-VCO-Oscillator with ATtiny25 controller

The mixer frequency for the DCF77 superhet with a TCA440 has to work exactly at 77.5+32.768 = 110.268 kHz, with deviations of only a few Hz. To achieve this with an LC oscillator, its frequency has to be measured exactly and, in case it differs by more than +/-5 Hz, it has to be re-adjusted. This can be done with an ATtiny25.

4.1.3.1 Design of the LC-VCO-Oscillator

Firstly, building an LC oscillator is a simple task: an appropriate coil L and a convenient capacitor C has to be brought to oscillate. This requires one FET and generates a nice sine wave.



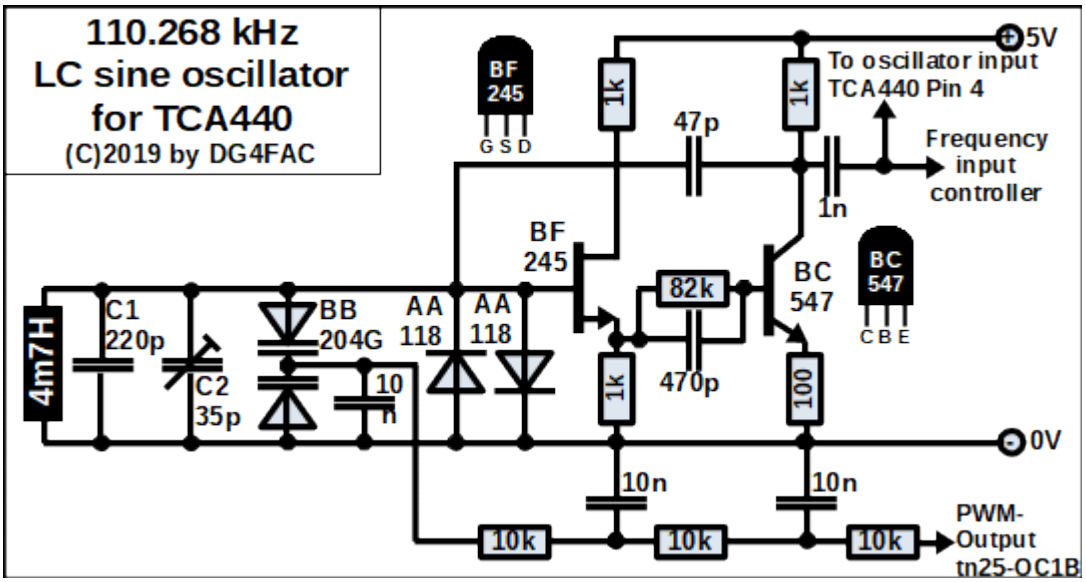
That is how such a simple LC oscillator looks like. It works as follows.

Depending from the capacitive voltage divider with the two capacitors from gate to source and from source to ground the FET produces a nice sine wave on the drain. On the source pin, the sine wave is rather distorted, and, if the divider ratio is changed a little bit, also the drain sine wave is rather distorted. The FET goes into saturation and distorts the clean sine wave. That comes from the large amplitude on the gate, which disturbs the function of the varactor diodes.

This has the disadvantage that the frequency regulation with the varactor diodes does not work good enough and is very far from predictable, even though those are reversed double diodes.

Conclusion: impracticable for a reliable operation.

To achieve an orderly operation, the amplitude of the LC circuit has to be kept as low as possible. In this design, the amplitude is limited by two Germanium or Schottky diodes on the LC circuit, that limit the amplitude at +/-0.2 V. The FET does not amplify in this design, an additional inverting amplifier follows to feedback enough HF to allow oscillation for which the inversion is necessary.



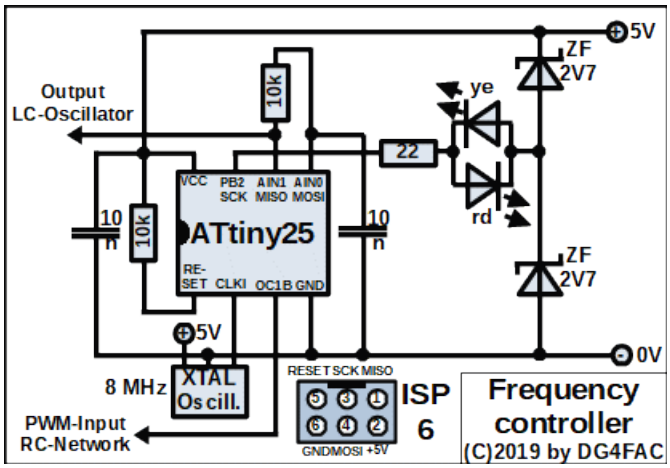
4.1.3.2 Frequency measurement and -regulation

To measure the frequency and to regulate the voltage of the varactor diodes for a constant frequency of 110,268+/-5 Hz an ATtiny25 follows. The controller is clocked with an 8 MHz crystal oscillator, the internal RC oscillator is not exact enough for that task.

The timer/counter TC0 in the ATtiny25 generates the measuring time clock: the pulses are counted for exactly 0.5 seconds long. The 0.5 seconds are achieved by dividing the controller clock of 8 MHz

1. by 256 in the prescaler, and
2. by 125 in the counter in CTC mode, and
3. by 125 in a register.

As the analog comparer is used to detect pulses, each sine wave produces two analog comparer changes, so the 24-bit wide frequency counter registers directly hold the frequency in Hz.

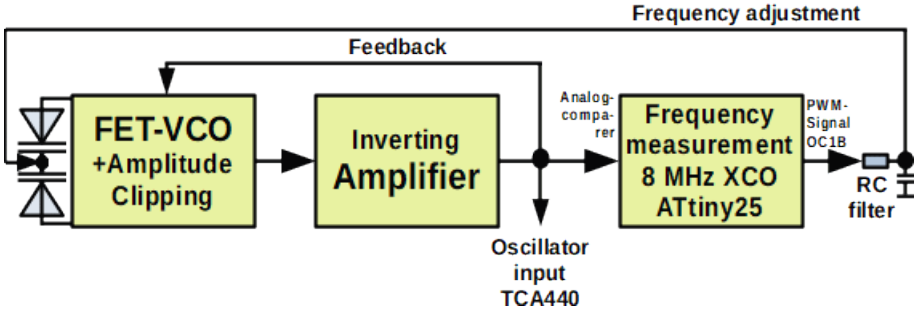


If this is by more than 5 Hz smaller, the PWM value of the 8-bit counter TC1 in OCR1B is increased by one. That increases the PWM output voltage after the RC filter by 5V/256 = 19.5 mV. This increasing voltage decreases the capacity of the varactor diode by approximately 0.02 pF and increases the LC frequency accordingly.

If the measured frequency exceeds 110,268 Hz by more than 5 Hz, the OC1B value is decreased by one, the smaller voltage on the varactor diodes increases their capacity and lowers the frequency of the LC accordingly. By that the oscillator frequency is kept within that narrow bandwidth.

To signal adjustments made and the correctness of the frequency two LEDs (or one double LED red/green) are build in. The yellow LED signals that the frequency is too small, the red signals exceeding frequency. If the oscillator works correct, both LEDs are switched off. If you do not need that and your oscillator works correct, just remove the LED and its current-limiting resistor.

This is the overall schematic of the frequency regulated oscillator.



4.1.3.3 Programming the ATtiny25

The program for the ATtiny25 is written in Assembler. The source code can be downloaded as assembler source text here and can be viewed in the next chapter.

The source code consists of the following functional parts:

1. Adjusting the hardware:
    - initiating the stack for interrupt handling,
    - initiating the LED output pin,
    - starting the timer/counter TC1 as asynchronous PWM (with 100µs wait time for PLL synchronization), with OC1B as output pin and with 255 in compare B (+5 V on the output,),
    - starting the timer/counter 0 as gate timer for frequency measurement, with interrupt enable,
    - starting the analog comparer for detecting edges on the input, with interrupt enable, and
    - enabling the interrupt flag in the status register.
2. The two interrupt routines
    - for frequency measurement via analog comparer: an 8-bit register counts the interrupts, on overflow an additional 16-bit counter is increased,
    - for the gate time of the frequency measurement: a divider register, starting with 125, is decreased, it it reaches zero
        - the register divider is restartet with 125,
        - the current 24-bit counter state is copied to three other registers, and
        - the 24-bit counter registers are cleared.
3. The comparison with the lower and upper limit of the target frequency:
    - Comparing the copied 24-bit counter with the smaller frequency limit: if the measured frequency is smaller the PWM frequency is increased (if not already at 255) and the yellow LED is switched on by setting its data direction bit and clearing the output bit,
    - Comparing the copied 24-bit counter with the higher frequency limit: if the frequency is larger than that, the PWM value is decreased (if not already smaller than the lower limit) and the red LED is switched on by setting its direction- and port-bit,
    - if both cases are not true, the LED is switched off by clearing its direction bit.

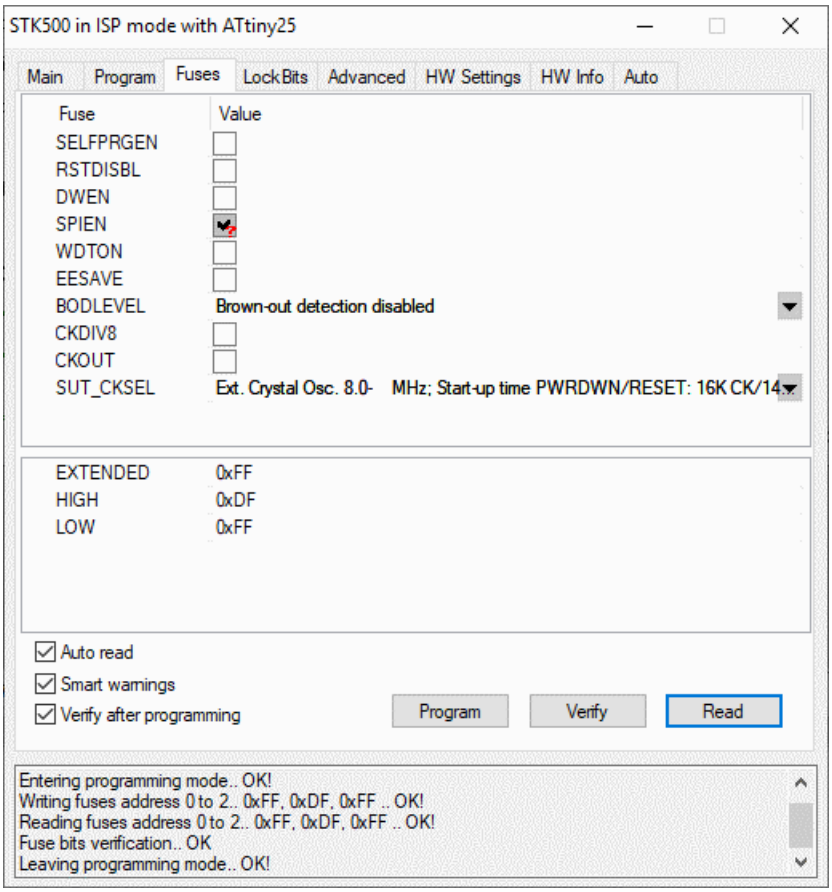
For both interrupt service routines their duration has been added in clock cycles.

The analog comparer interrupt occurs every 4.5 µs, so after 9 µs one interrupt event would be missing causing a difference of 1 Hz. That is the case if the TC0 interrupt is longer than 72 clock cycles.

An interrupt loss of the TC0 gate timer can only occur after 512 clock cycles.

Both interrupt service routines are fast enough to not interfere with each others.

The whole program has 137 words and fits very well into the flash memory of the ATtiny25. Do no forget, prior to or after programming the flash, to change the oscillator fuse of the ATtiny25, otherwise it would work with only 1 MHz.



4.1.3.4 Connecting the LC oscillator to the TCA440

The output of the LC oscillator on the collector of the BC547 is coupled to the TCA440's oscillator input on pin 4 via a capacitor of 1 nF. The differential input on pin 5 of the TCA440 is deactivated with a 1 or 10 nF capacitor to ground, therefore the oscillator input is unsymmetrical and is made symmetrical only by the emitter resistor in the TCA440's oscillator input stage.

4.1.3.5 The source code for the ATtiny25

This is the assembler source code for the LC-VCO's ATtiny25. The original source code in assembler format is here.

```
;
; ********************************
; * LC Oscillator with frequency  *
; * regulation via PWM&varicap    *
; * (C)2019 avr-asm-tutorial.net  *
; ********************************
;
.nolist
.include "tn25def.inc" ; Define device ATtiny25
.list
;
; ********************************
;  D E B U G G I N G   S W I T C H
; ********************************
;
.equ Yes = 1 ; Set debug on
.equ No = 0 ; Set debug off
;
; Do not compare, blink in 500 ms
.equ debug_blink500ms = No ; Yes = blink
;
; Blink on measuring on analog compare int
;   Red and green LED blink very fast if comparer
;   int occurs, counting is disabled
.equ debug_blinkaci = No ; Yes = blink
;
; ********************************
```

```
;        H A R D W A R E
; *********************************
;
; Device: ATtiny25, Package: 8-pin-PDIP_SOIC
;                 _________
;             1 /         |8
;    RESET o--|RESET  VCC|--o +5V
; Xtal osc o--|CLKI   PB2|--o SCK/LED
; PWM out  o--|OC1B   PB1|--o MISO/AIN1
;        0V o--|GND    PB0|--o MOSI/AIN0
;             4 |_________|5
;
; *********************************
;  P O R T S   A N D   P I N S
; *********************************
;
.equ pPwmD = PORTB ; PWM direction port
.equ bPwmD = DDB4 ; PWM direction port pin
.equ pLedO = PORTB ; LED output port
.equ pLedD = DDRB ; LED direction port
.equ bLedO = PORTB2 ; LED output port pin
.equ bLedD = DDB2 ; LED direction port pin
.equ pLedI = PINB ; LED blinking port
.equ bLedI = PINB2 ; LED blinking port pin
;
; *********************************
;   A D J U S T A B L E   C O N S T
; *********************************
;
; Clock rate of external crystal oscillator
.equ clock=8000000 ; Define clock frequency
.equ cOscFreq = 77500 + 32768 ; Frequency of the LC oscillator, Hz
.equ cOscTol = 5 ; Frequency tolerance +/-, Hz
.equ cMinVolt = 2000 ; Minimum voltage of PWM, mV
;
; Use sheet clock in dcf77_lcosc_tn25.ods for the following
.equ cGateTime = 500 ; Gate time for frequency measurement in ms
.equ cPresc = 256 ; Prescaler (from spreadsheet)
.equ cCtcDiv = 125 ; CtcDivider (from spreadsheet)
;
; *********************************
;  F I X  &  D E R I V.  C O N S T
; *********************************
;
; Check clock
.set cClockCorrect = clock==4000000
.set cClockCorrect = cClockCorrect || (clock==4194304)
.set cClockCorrect = cClockCorrect || (clock==4915200)
.set cClockCorrect = cClockCorrect || (clock==5120000)
.set cClockCorrect = cClockCorrect || (clock==6553600)
.set cClockCorrect = cClockCorrect || (clock==7372800)
.set cClockCorrect = cClockCorrect || (clock==8000000)
.set cClockCorrect = cClockCorrect || (clock==16000000)
.if cClockCorrect
  .message "Clock is correct"
  .else
  .error "Incorrect clock setting!"
  .endif
;
; Define frequency measurement constants
.equ cTc0Clk = clock / cPresc ; Define prescaler from clock
.equ cDiv = cTc0Clk / cCtcDiv / 2 ; Register divider
.if cDiv > 256
  .error "cDiv is too large!"
  .endif
.if cCtcDiv == 256
  .equ cCtcCmp = 0
  .else
```

```
  .equ cCtcCmp = cCtcDiv-1
  .endif
.equ cDelta = cDiv*(cCtcCmp+1)*cPresc ; Clock calculated
.if cDelta != clock
  .message "Clock divider has division rest, inaccurate second"
  .endif
.equ cMeasFreq = 1000 / cGateTime ; Measuring frequency
.if cMeasFreq < 1
  .error "Measuring gate time too long"
  .endif
;
; Oscillator constants
.equ cOscLow = cOscFreq - cOscTol ; Smallest tolerable frequency
.equ cOscMax = 2*cOscTol + 1 ; Largest tolerable frequency
;
; Minimum voltage of PWM
.equ cMinPwm = (cMinVolt * 256) / 5000 ; Minimum PWM value
;
; *********************************
;       R E G I S T E R S
; *********************************
;
; free: R0 to R14
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
; free: R17 to R29
.def rFlag = R17 ; Flag register
  .equ bOver = 0 ; Measuring cycle is over flag
; free: R18
.def rDiv = R19 ; Register divider
.def rFrq0 = R20 ; Measured frequency result, byte 1
.def rFrq1 = R21 ; dto., byte 2
.def rFrq2 = R22 ; dto., byte 3
.def rCnt0 = R23 ; LSB of 24 bit counter
.def rCnt1 = R24 ; HSB, used as 16 bit counter
.def rCnt2 = R25 ; MSB of 16 bit counter
; free: R26 to R31
;
; *********************************
;           S R A M
; *********************************
;
; No SRAM used
;
; *********************************
;         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 ; PCI0
        reti ; OC1A
        reti ; OVF1
        reti ; OVF0
        reti ; ERDY
        rjmp AciIsr ; ACI
        reti ; ADCC
        reti ; OC1B
        rjmp Oc0AIsr ; OC0A
        reti ; OC0B
        reti ; WDT
        reti ; USI_START
```

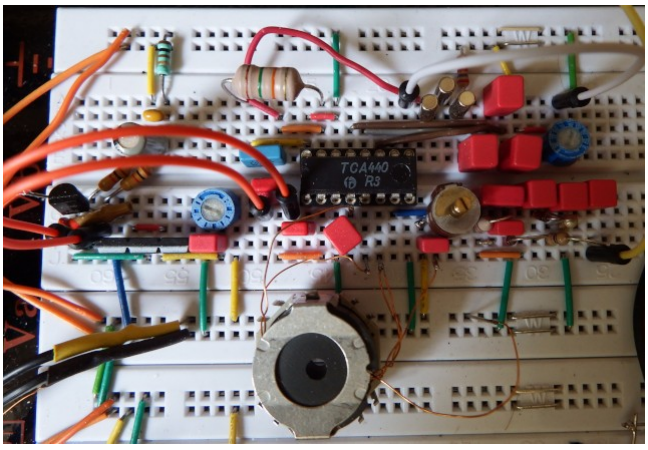
```
        reti ; USI_OVF
;
; *********************************
;  I N T - S E R V I C E   R O U T .
; *********************************
;
; Counts changes of the analog comparer
;   Occurs every 4.5344 us at 110.268 kHz
AciIsr: ; 7 clocks for int and vector jump
  .if debug_blinkaci == Yes
    sbi pLedI,bLedI ; Blink LED
        reti
    .endif
  in rSreg,SREG ; Save SREG, +1 = 8
  inc rCnt0 ; Count LSB, +1 = 9
  brne AciIsr1 ; +1/2 = 10/11
  adiw rCnt1,1 ; Count HSB/MSB, +2 = 12
AciIsr1: ; 11/12 clocks
  out SREG,rSreg ; Restore SREG, +1 = 12/13
  reti ; +4 = 16/17
; Maximum 17 clock cycles
;   17 clock cycles in 4.5344 us = 4 MHz min.
;
; 2 Hz counter interrupt service routine
;   counts for 0.5 seconds and reads
;   counting result
Oc0AIsr: ; 7 clocks for int and vector jump
  in rSreg,SREG ; Save SREG, +1 = 8
  dec rDiv ; Decrease divider, +1 = 9
  brne Oc0AIsr1 ; Not yet zero, +1/2 = 10/11
  sbr rFlag,1<<bOver ; Set bOver flag, +1 = 11
  ldi rDiv,cDiv ; Restart cDiv, +1 = 12
  mov rFrq0,rCnt0 ; Copy counter, +1 = 13
  mov rFrq1,rCnt1 ; +1 = 14
  mov rFrq2,rCnt2 ; +1 = 15
  clr rCnt0 ; Clear counter, +1 = 16
  clr rCnt1 ; +1 = 17
  clr rCnt2 ; +1 = 18
Oc0AIsr1: ; 11/18 clock cycles
  out SREG,rSreg ; Restore SREG, +1 = 12/19
  reti ; +4 = 16/23
;
; *********************************
;  M A I N   P R O G R A M   I N I T
; *********************************
;
Main:
  ldi rmp,Low(RAMEND)
  out SPL,rmp ; Init LSB stack pointer
  ; Init I/O ports
  sbi pLedD,bLedD ; Turn LED output on
  sbi pLedO,bLedO ; Turn red LED on
  ; Start TC1 as async PWM
  sbi pPwmD,bPwmD ; Set OC1B as output
  ldi rmp,255 ; Start with the highest PWM stage
  out OCR1B,rmp ; in compare port B
  ldi rmp,255 ; End value for PWM, 8-Bit PWM
  out OCR1C,rmp ; in output compare register C
  ldi rmp,(1<<PLLE)|(1<<LSM) ; Enable PLL in low speed mode
  out PLLCSR,rmp ; in PLL control register
  ; Wait for 100 microseconds
  ;   n = 2+4*(z16-1) + 3
  ;   n = 2+4*z16-4+3 = 4*z+1
  ;   4*z16 = n-1
  ;   z16 = (n-1)/4
  ;   n @ 8MHz = 800
  .equ z16 = (clock/10000+2)/4
  ldi rCnt2,High(z16) ; Wait for 100 us, MSB
```

```
  ldi rCnt1,Low(z16) ; dto., LSB
PllWait:
  sbiw rCnt1,1 ; Count down
  brne PllWait ; Wait further
  ldi rmp,(1<<PLLE)|(1<<LSM)|(1<<PCKE) ; and PCK
  out PLLCSR,rmp ; in PLL control register
  ldi rmp,(1<<PWM1B)|(1<<COM1B1) ; PWM B enabled, High to low
  out GTCCR,rmp ; in general timer control register
  ldi rmp,(1<<PWM1A)|(1<<CS12) ; PWM A, High/Low, Prescaler=8
  out TCCR1,rmp ; in TC1 control register
  ; Start TC0 as gate timer
  ldi rDiv,cDiv ; Start software divider
  ldi rmp,cCtcCmp ; Set compare A value
  out OCR0A,rmp ; in compare A
  ldi rmp,1<<WGM01 ; Set CTC mode 3
  out TCCR0A,rmp ; in TC0 control port
  clr rmp
  .if (cPresc == 1) || (cPresc == 64) || (cPresc == 1024)
    sbr rmp,1<<CS00
    .endif
  .if (cPresc == 8) || (cPresc == 64)
    sbr rmp,1<<CS01
    .endif
  .if (cPresc == 256) || (cPresc == 1024)
    sbr rmp,1<<CS02
    .endif
  out TCCR0B,rmp ; to TC0 control port B
  ldi rmp,1<<OCIE0A ; Enable interrupt on compare A
  out TIMSK,rmp ; in timer int mask
; Sleep mode idle
  ldi rmp,1<<SE ; Sleep enable
  out MCUCR,rmp ; in microcontroller control port
  ; Init analog comparer as frequency input
  ldi rmp,(1<<AIN1D)|(1<<AIN0D) ; Disable digital inputs
  out DIDR0,rmp ; in analog disable port register
  ldi rmp,1<<ACIE ; Enable analog comparator interrupts
  out ACSR,rmp ; in analog comparer status register
;
; Enable interrupts
       sei ; Enable interrupts
;
; *********************************
;    P R O G R A M   L O O P
; *********************************
;
Loop:
  sleep ; Go to sleep
  nop ; Delay on wake-up
  sbrc rFlag,bOver ; bOver flag clear?
  rcall Measured ; Frequency measurement
  rjmp Loop
;
; Frequency measurement complete
Measured:
  cbr rFlag,1<<bOver ; Clear flag
.if debug_blink500ms == Yes
  sbi pLedI,bLedI
  ret
  .endif
  ldi rmp,Byte1(cOscFreq) ; Byte 1 of cOscLow
  sub rFrq0,rmp ; Subtract measured frequency, LSB
  ldi rmp,Byte2(cOscFreq) ; Byte 2 of cOscLow
  sbc rFrq1,rmp ; dto., HSB
  ldi rmp,Byte3(cOscFreq) ; Byte 3 of cOscLow
  sbc rFrq2,rmp ; dto., MSB
  brcs MeasuredLow ; Frequency too small, increase
  ldi rmp,Byte1(cOscMax) ; Byte 1 of upper bound
  sub rFrq0,rmp ; Subtract upper bound, LSB
```

```
  ldi rmp,Byte2(cOscMax) ; Byte 2 of upper bound
  sbc rFrq1,rmp ; Subtract upper bound, HSB
  ldi rmp,Byte3(cOscMax) ; Byte 3 of upper bound
  sbc rFrq2,rmp ; Subtract upper bound, MSB
  brcc MeasuredHigh ; Frequency too high, decrease
  cbi pLedD,bLedD ; LED off
  ret
MeasuredLow:
  ; Measured frequency too low, increase
  in rmp,OCR1B ; Read compare value
  inc rmp ; Increase compare value
  brne MeasuredSetPwm ; Not the max. value, set PWM
  dec rmp ; Decrease again
  cbi pLedO,bLedO ; LED to yellow
  rjmp MeasuredSetPwm
MeasuredHigh:
  ; Measured frequency too high, decrease
  in rmp,OCR1B ; Read compare value
  dec rmp ; Decrease
  cpi rmp,cMinPwm ; Smaller than minimum PWM
  brcc MeasuredSetPwm ; Not smaller than min., set PWM
  inc rmp ; Increase again
  sbi pLedO,bLedO ; LED to red
MeasuredSetPwm:
  out OCR1B,rmp ; Write new value to TC1 compare B
  sbi pLedD,bLedD ; LED pin as output
  ret
;
; End of source code
;
Copyright:
.db "(C)2019 by Gerhard Schmidt",0,0
.db "C(2)10 9ybG reahdrS hcimtd",0,0
```

4.2.4 Mounting the superhet

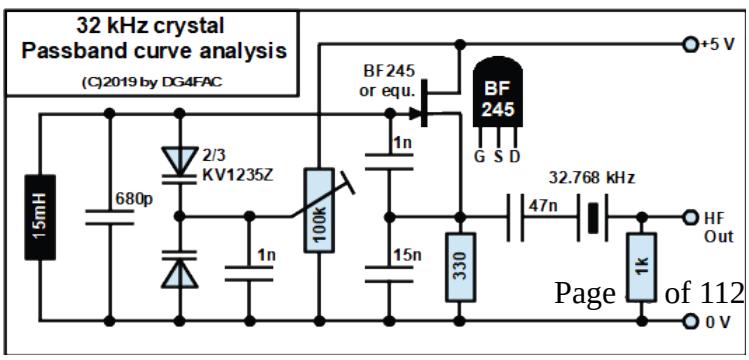
That is how the capacitor and xtal grave looks alike on a breadboard, here with a LC oscillator.



To the left the buffer stage with the FET can be seen (the antenna can not be seen). The frequency of the input stage can be adjusted with the left trim resistor. Then the TCA440 with the oscillator coils follow. Above to the right the three tiny crystals and the 1µF grave can be seen. On the lower part the three 470 µF capacitors of the rectifier can be seen. The trim resistor to the right regulates the gain of the IF amplifier.

4.3 The xtal filter for 32.768 kHz

To measure the filter properties of 32.768kHz crystals, one can use this oscillator. It generates a 32kHz sine wave signal with an adjustable frequency. The adjustment is made

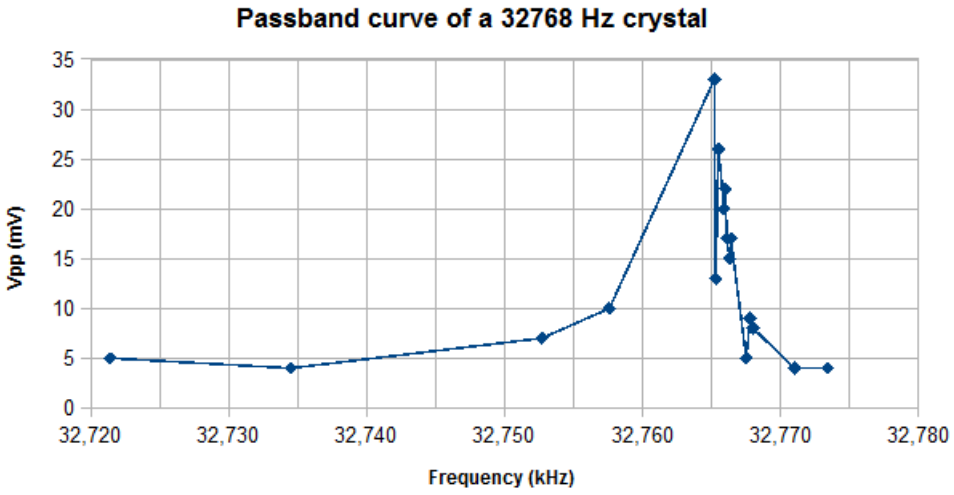


with Medium Wave varactor diodes, for which a BB212 or a variable capacitor for medium wave can also be used.

The crystal is fed with the low-resistance signal output of the sine wave generator and has an output resistor of 1kΩ.

This is the resulting pass-band curve. It is less than 10 Hz wide, especially the falling edge is rather steep.

When measuring slightly above the resonance frequency a moderate feedback on the oscillator took over control, so one single data point showed an unexpected value.

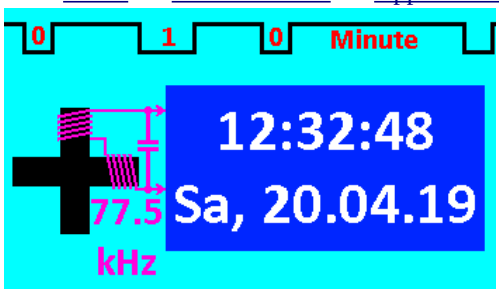


Remarkable is that the selectivity far from the resonance is rather limited. This is caused by the stray capacity of the crystal. Therefore the crystal filter shall always be combined with an LC filter, to reduce frequencies far from the xtal resonance.

4.4 Automatic control of the DCF77 signals

The gain control as well as the frequency adjustment can, for test purposes, be adjusted with resistor trimmers. A usual trim potentiometer with 270° is sufficient.

More comfortable is when a micro-controller does that work. Measuring, adjusting and control of the AGC and AFC can be done with an ATtiny45, as shown here in detail.



Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 controller with ATtiny45

5 DCF77 controller with ATtiny45

To ensure that the DCF77 clock owner does not have to adjust its frequency and gain steadily and with a trim resistor, and to ensure that the DCF77 information does not have to be decoded by counting and assembling single bits, a small controller has been developed that does all that: taking care for the receiver and decode the DCF77 bits.

The results of that control are

1. two PWM signals:
    1. a gain control signal, ranging from 0 to 255, where larger numbers decrease the gain of the receiver, so that enough, but not too large DC from the AM rectifier results,
    2. a frequency control signal, ranging from 0 to 255, which increases the varactor diode's capacity with increasing values and frequently tries out, if an increased or decreased voltage increases the drop difference of the rectified DCF77 signal, by that keeping the input frequency of the ferrite antenna always on the center of DCF77's transmit frequency,
2. the checking that amplitude drop and pause times of the DCF77 signal are within the correct expected times of 100 or 200 ms (for a zero or one bit received), of 800 resp. 900 (for an inactive pause with high amplitude following reception of a one or zero bit) or of a high amplitude for either 1800 or 1900 milliseconds (minute change following reception of the 59th or last one or zero bit), times plus and minus a selectable tolerance percentage,
3. to collect DCF77 bits, if the 100 or 200 ms long amplitude drops occur, and to store those 59 bits per minute in a correct row in the SRAM,
4. on a minute change correctly received: to check all parity bits of DCF77 (minutes, hours, date) for correctness,
5. to convert the received bits for minutes, hours, weekday, day, month and year from BCD to binary format, and to
6. send all those information, including error messages and status information, over a one-way two-wire interface to another controller, that can receive and display all that on an LCD.

5.1 Why assembler? Why an ATtiny45 and nothing else?

Lots of things to do for

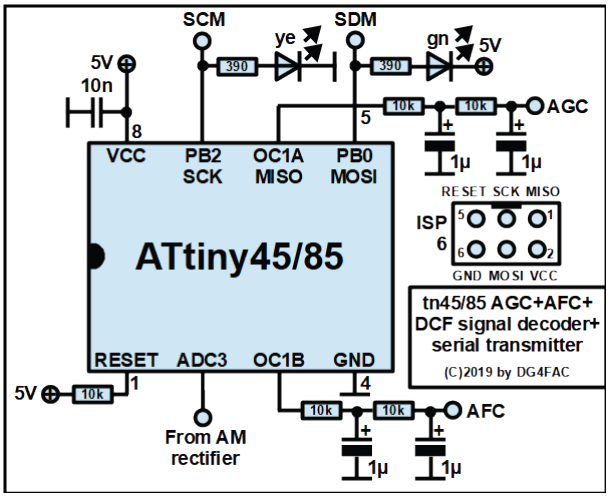
- an 8-bit-8-pin ATtiny45 controller with max. 2,048 instruction memory words and 256 bytes of SRAM storage space,

- but manageable tasks if you do all that in assembler and not in one of the memory-space eating, time-wasting and completely inadequate library-focused styles of high-level-languages such as C or Bascom,
- a controller, but using the wonderful instruction set that the two Norwegians have designed reduces CPU instruction steps from three or four, as necessary in less well-designed PIC assembler, down to one, speeding up execution and to work with the default 1 MHz clock rate,
- such a small and slow controller, speeded up by the use of fast interrupts to ensure that all different hardware tasks are executed timely and exactly then when needed, without wasting any time for unnecessary wait loops,
- the six pins I/O:
    - two to switch a PWM output on and off very fast (a very special and exclusive feature of the ATtiny25/45/85, with their very fast internal 64/32 MHz PWM oscillator, allowing to use small capacitors to generate low-humming as well as very fast-responding PWM signals, do not try this with another ATtiny or a PIC as it does not work with such high speed,
    - another two pins for the one-way serial communication with the other controller, allowing a lean and fast sending baud-rate, just enough to fit into the overall time schedule and to allow the receiver to collect those bits in an interrupt-driven scheme,
    - one pin to measure the amplitude DC, coming from the receiver's rectifier stage, by use of the internal AD converter, with conversion initiated in constant time intervals to ensure that the measuring rate fits well with the amplitude changes of DCF77,
    - the RESET pin, that allows In-System-Programming of the ATtiny45 during the software design stage, without having to remove the chip.

The additional controller to display those signals on an LCD was necessary to have all the pro's of the ATtiny45 PWM features and to allow the use of other controllers for the display or if you need any further DCF77-date-and-time-dependent switching only. If you need only two or three channels to be switched on and off only during weekdays you can use an ATtiny13 instead: connect serial clock with INT0, serial data with any other pin and the two or three output pins with your switches. The DCF77 controller says which weekday currently is and which time and date. Even if DCF77 is off: the status messages allow to detect such failures and to switch to a software-driven clock scheme instead.

5.2 The schematic of the ATtiny45 controller for DCF77

This is all you need. It works with an ATtiny45 or ATtiny85.



The two PWM outputs OC1A and OC1B generate the two voltages for AGC and AFC and use the high-speed-PWM features of the ATtiny45/85. The High-Speed PWM oscillator of 64 MHz is switched by software to low speed (32 MHz is sufficient), then divided by 8 in the prescaler, to yield a PWM frequency of

31.25 kHz. Both outputs are connected to a two-stage RC network of 10kΩ and 1µF. This ensures that digital humming is small enough and that the analog voltages produced react fast enough to any changes of the PWM value.

The rectified and RC filtered (0.01 ms) AM DC signal from the DCF77 receiver is fed to the AD converter channel ADC3. Conversion is clocked by TC0's timer overflow, that occurs every 16.4 ms. The amplitude drop of 100 ms length for a transmitted zero results in six, for a one in approximately twelve measurements. A zero and a one, followed by a minute change pause of approximately 1,800 ms, happen within three seconds or over 183 measurements. The storage of such measurement results, in order to calculate averages and to measure pulse durations, requires 183 SRAM bytes. That is why the ATtiny25 does not fit.

The whole time and date detection of the DCF77 signal as well as the AGC and AFC control is based on this collection. No further RC filters are necessary (like this wou?d be the case to detect the minute change in the DCF77 signal, see the chapter on how-it-works below).

Serial output of the results is done with the pins PB0 (data master, SDM) and PB2 (clock master, SCM). As communication in backwards direction is unnecessary, both outputs are always master and active. The two LEDs can be used to view active signal traffic.

The ISP interface can be used to program the chip within. It is not necessary if pre-programmed ATtiny45/85 are used.

5.3 Functioning

5.3.1 Start-up phase

When the controller starts, it absolves a start-up phase. This adjusts the AGC and the AFC to start-up values. Both start with decimal 255 in their PWM channels or +5V. On every complete batch of 155 measurements the AGC value is decreased by eight (maximum 32 batches or 81 seconds). The decrease stops if either

1. a minimum of 0.5 V has been reached, or
2. a maximum of 2.5 V has been exceeded, or
3. zero has been reached.

In any case the first approach of AGC adjustment is over then, further adjustment is taken over by each completed batch by

1. increasing the PWM value, if the maximum is larger than 2.5 V, or
2. decreasing the PWM value, if the maximum is smaller than 2.0 V.

In the second phase the AGC value is adjusted. After each batch the AGC PWM is decreased and the difference between maximum and minimum is stored in SRAM. This phase is stopped when the AFC PWM value reaches zero (which takes another 81 seconds).

Software then searches for the first maximum value in the stored values, starting from the difference at the PWM value of 255. This is done because, in case of a larger ferrite coil, a second maximum can occur that relates to one of the two coils, which can be even stronger than the combined one, but signal strength at that decreases very fast if the di-

rection of the ferrite changes slightly. If the first maximum is identified, the AFC PWM is written to that value.

Further frequency adjustment is then done on each batch completion: the AFC PWM value is increased, and then decreased, and the difference of the maximum and minimum decides, whether the last change is repeated (if the difference is larger) or if the direction changes (if the difference is smaller). If the maximum has not been found during the AFC scan period, the frequency scan is repeated over and over again.

5.3.2 Detection of zero/one bits and minute change

During the AGC and AFC scan periods normal checking of incoming single values is omitted. Only if both the bGScan and bFScan flags are set, the single values are checked whether a level change has happened. To do that, the last three measurements, as stored in three registers and updated whenever a new measurement is completed, are compared with the average value. The average value is updated whenever a complete batch has been measured, it is the maximum value minus the minimum value.

If the last values were inactive (higher than average) it is waited for three succeeding values below average. If those are detected

- the detection direction bit is changed to increasing values,
- the length, over which the inactive high period lasted, is calculated, if this
    - is below the minimum period of a zero, or
    - in between the maximum period of a zero and the minimum period of a one, or
    - in between the maximum period of a one and the minimum period of a pause, or
    - in between the maximum period of a pause and the minimum period of a minute change, or
    - beyond the maximum period of a minute change

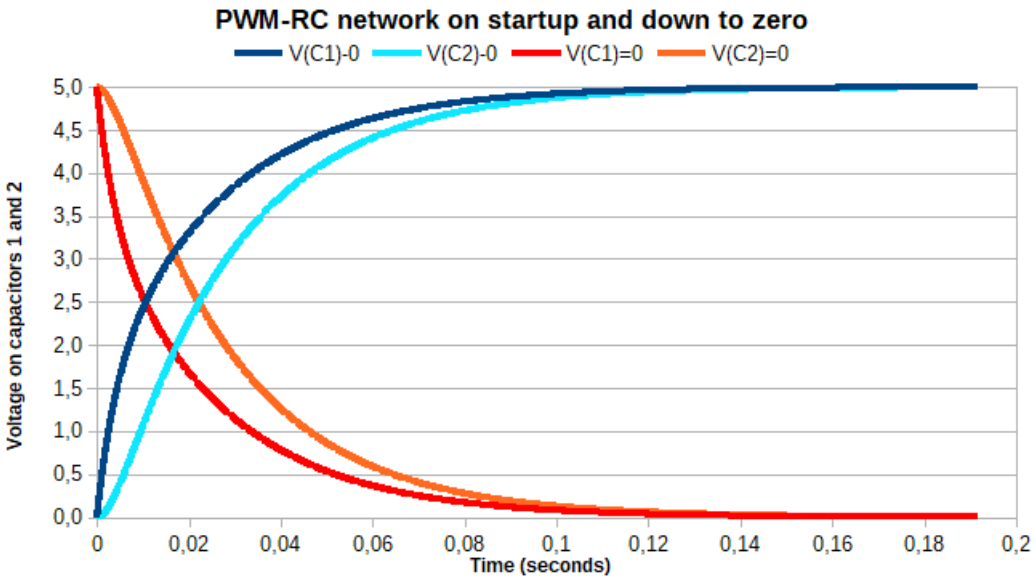
  an error condition has occurred and is reported with an **E** code via the serial interface.
- if the duration is in between the minimum for a minute change and its maximum, it is checked whether exactly 59 bits have been received. If that is the case, the complete set of DCF77 bits is converted (see below), otherwise an error is send.

If the previously recognized stage was a signal coming in (three values were below average) the last three values are checked for a signal end. If all three are above average the detection of high-to-low signals is switched on. The signal duration of the low-signal is checked for the minimum length of a zero and its maximum as well as for a one with its minimum and maximum. If a correct zero or one has been identified, the respective bit is shifted into the bit storage and counted. If not, respective error messages are send via the serial interface.

5.3.3 Generation and properties of the PWM signals

The two outputs OC1A and OC1B generate PWM signals for adjusting gain and frequency. 8-bit timer TC1 is in asynch mode: the PWM clock is at 64 MHz, slowed down by two and is divided by 8. The PWM width is 256 stages, so the resulting PWM frequency is 64 MHz / 2 / 8 / 256 = 15.625 kHz or a PWM period of 64 µs.

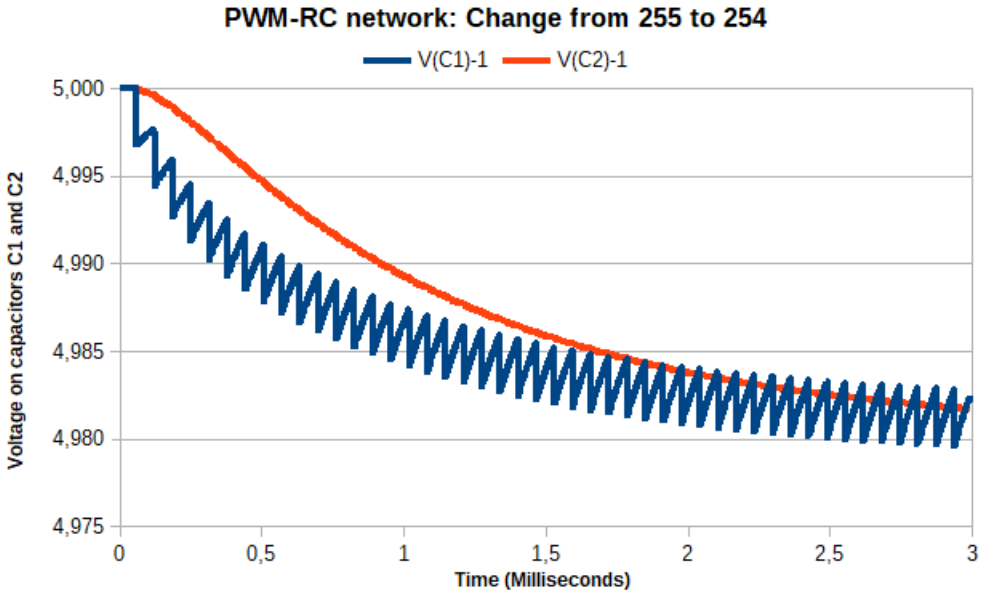
To filter the harmonics, a double RC filter with 10kΩ and 1µF follow. The filter was simulated with the Libre-Office spreadsheet here. On start-up, when both capacitors are unloaded and the PWM is set to 255, the following voltage increase happens.



The increasing curve shows that the RC filter has nearly complete the end point after 0.1 seconds and completely after 0.2 s. That is fast enough to not having to wait on start-up for the stabilized voltage. The second capacitor follows slightly behind, but is also fast enough.

The change speed from full load down to zero (not exactly zero, as the PWM has a minimum of 1, which corresponds to a minimum of 19.5 mV), is similar. The end value is reached within 0.2 s.

To demonstrate the speed of change for a small difference in PWM values, this shows the voltages on both capacitors if the PWM is switched from 255 down to 254 (normal switching is by one unit, 19.5 mV). The first capacitor shows voltage drops (humming) of 3 to 4 mV by the single low phase among 255 high phases, the second capacitor is completely free of this humming.

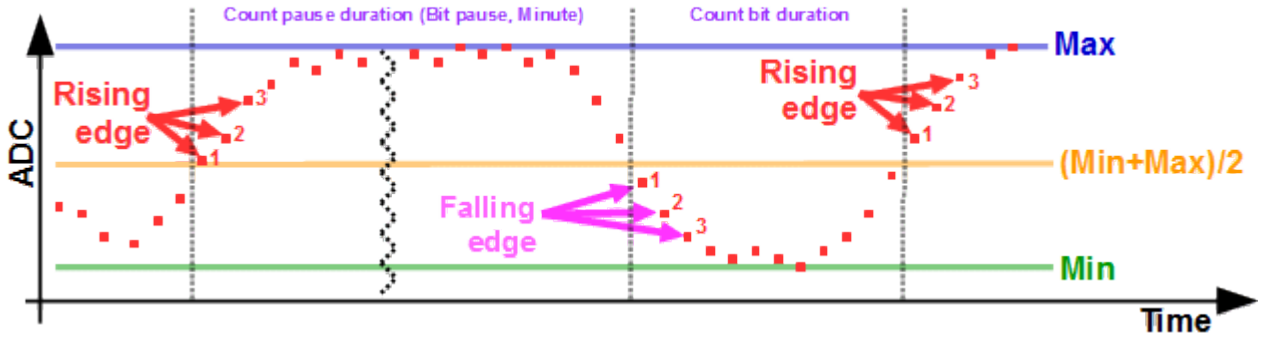


The voltage on C2 is a bit delayed, but by less than 5 ms. That is fast enough for the AFC and AGC adjustment, the next ADC measurement after 16.384 ms will already reflect the new voltage setting.

Do not try this with a different type of ATtiny or a different controller, it does not work due to lower PWM frequencies.

5.3.4 Measuring and evaluation of the AM DC signals

To ensure that measuring the amplitude voltage in constant time slices the AD conversion is started by the overflow of timer TC0 using the **ADATE** bit of the ADC. The TC0 prescaler is set to 64, so that the conversion starts all 256 * 64 * 1,000 / 1,000,000 = 16.384 ms. The conversion, with an ADC clock prescaler of 128, needs 1.664 ms (see section **Timing** in the source code).



Reading of the ADC values is performed within the ADC's interrupt service routine:

1. only the upper 8 bits of the result are read (ADLAR is activated),
2. the value read is written to the registers rLast, so that the last three measurements are available in the registers rLast1 to rLast3, because the amplitude drop detection is based on those three values (outside the ISR),
3. the value is written to an SRAM buffer **sBuffer**, that is adjusted to a length of 155 measurements, corresponding to the last 155 * 16.384 ms = 2.54 seconds.

If the buffer is full, the flag **bBufFull** in the flag register **rFlag** is set. Outside the interrupt service routine this bit is recognized and the complete buffer is searched for the maximum and minimum values. The difference between the two extremes is then

1. evaluated if the minimum difference in **cAmVoltDelta** has been reached. If this is not the case, error message **E0** (DCF77 signal time-out) is issued over the serial interface, further bit evaluation is blocked by setting the **bMin** flag in the flag register and a frequency scan is re-started by clearing the **bFScan** flag,
2. divided by two and added to the minimum value. This value is used as compare value to decide whether the signal strength is above or below the average value.

The maximum, average and difference values are applied until the next buffer is filled (after 2.54 seconds). From the measured difference value the AGC adjustment is derived on every buffer-full event:

1. The PWM value is either increased or decreased in the next period, depending from the result of the last increase or decrease.
2. If the last change was an increase:
    - it is checked whether the difference between max and min has increased in that period, if that is the case, the next higher value is written to the PWM compare register,

- if that is not the case and the difference is smaller or equal the change direction is reversed to decrease and the next lower PWM value is written to the PWM compare register.

3. If the last change was a decrease the opposite is performed.

That means, that in case that the difference is equal, the direction is changing each time the buffer is full, and that the PWM value constantly goes up and down by one unit.

The AGC PWM is adjusted as follows:

1. if the maximum value is smaller than 2.0 V, the gain is increased by decreasing the PWM compare value by 1,
2. if the maximum value is larger than 2.5 V, the gain is decreased by increasing the PWM compare value by 1,
3. if the maximum is in between 2.0 and 2.5 V, the PWM value is not changed.

If neither **bGScan** nor **bFScan** are cleared, the edge detection is active. Edge detection works as follows:

1. The flag **bHi** in the flag register stores if the last edge was a low-to-high transition, the register **rTransH:rTransL** holds the SRAM buffer position when the last transition happened.
2. If **bHi** is set, an amplitude drop is to be detected. It is checked whether the last three values in **rLast1**, **rLast2** and **rLast3** are all below the average value. If that is not the case the detection routine ends and waits for the next value. If it is the case,
    - the flag **bHi** is cleared,
    - the difference between the current and the last transition in the buffer is calculated and evaluated whether the duration of the high-state was, within the tolerances, a correct pause between the last zero or one bit and the start of the next bit or a correct minute change. In the latter case the DCF77 bits are checked whether 59 were received and the DCF77 time and date are checked, converted and issued.
    - the current position in the sBuffer is written to **rTransH:rTransL**.
3. If **bHi** is clear, the next rising edge is to be detected. This is the case if all **rLast** registers are above average. If this is the case,
    - the flag **bHi** is set,
    - the duration of the low signal is calculated by comparing the current **sBuffer** position with the start position in **rTransH:rTransL**. If this is within the tolerance area of a zero or a one, the respective bit is shifted into the DCF77 bits received and counted. If not, respective error codes are transmitted over the serial connection,
    - **rTransH:rTransL** is updated.

5.3.5 Serial transmission

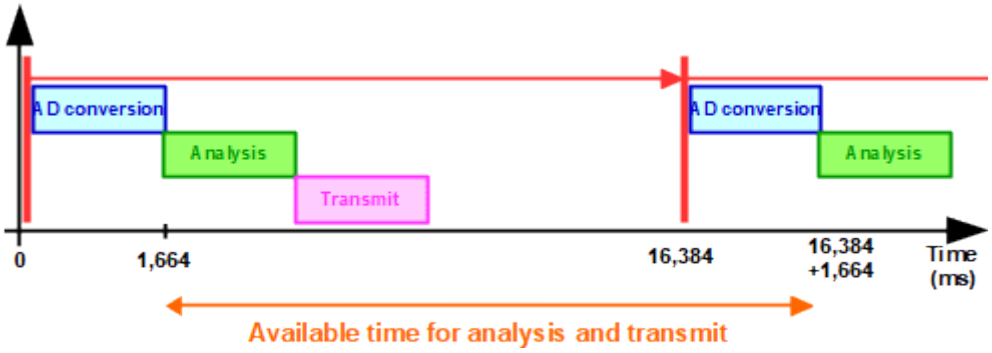
The serial transmit routine is called whenever the ADC reports a single measurement result (any 16.384 ms) and after all actions to be taken are completed. The messages to be send are written into a ring buffer in SRAM, with two bytes for each message. The buffer input and output addresses are held in two register pairs. On calling the send routine it is

checked whether input and output addresses are equal. If that is the case, nothing is send.

If there are messages to be send,

1. the message is copied (2 bytes),
2. it is send bit-by-bit (starting with bit 15 down to 0) over the serial pins by
   - placing the bit onto the serial data output pin,
   - waiting for a certain delay time for settlement,
   - setting the serial clock output pin high for a certain time, then clear again,

   and for all 16 bits,
3. increasing the message output address by two.



The receiver has to be able to receive each bit within 50 µs (at 10 kBd) resp. 25 µs (at 20 kBd). A PCINT in the receiver, with shifting the bits and counting, takes 27 clock cycles, so the receiver has to work with at least a clock rate of 540 kHz at 10 kBd or 1.08 MHz at 20 kBd. So the default is set to 10 kBd to be compatible with a 1 MHz clock rate of the receiver.

If necessary, any 16.384 ms a data set transmit requires 800 or 1,600 µs time. For flag handling and all other operations nearly the complete time between this and the next ADC event is available (16.3 milliseconds), because the AD conversion does not need any action. Transmit baud-rates of 10 or 20 kBd are compatible, the lowest possible baud-rate would be roughly 8 ms for 16 Bits = 2 kBd.

Lots of different information have to be send to the receiver. To ease processing of those in the receiver, the 16 bits were divided into the MSB, consisting of an ASCII character, and a second LSB that holds an additional parameter. All message codes are listed in the table, in "" enclosed characters are ASCII, in () enclosed values are binary numbers.

Parameter	High Byte	Low Byte	Duration (ms)
Statusmeldungen			
Restart	"R"	0	**16.4**
Frequency scan completed	"C"	0=not ok, 1=ok	
Signal strength	"S"	(AGC value)	
Frequency	"F"	(AFC value)	
Received DCF77 time			**98.3**
Minutes	"m"	(Minutes)	
Hours	"h"	(Hours)	
Weekday	"W"	(Weekday)	
Day	"D"	(Day)	
Month	"M"	(Monath)	

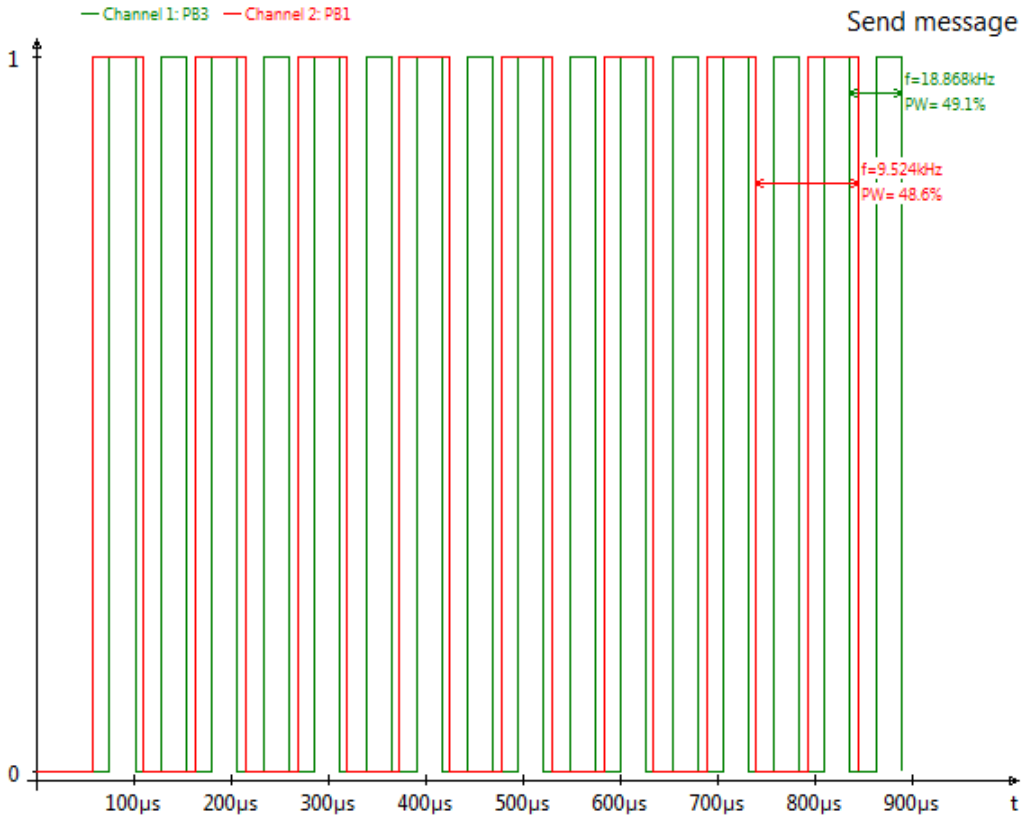
Parameter	High Byte	Low Byte	Duration (ms)
Year	"Y"	(Year)	
Bit monitoring			**16.4**
Zero received	"0"	0	
One received	"1"	1	
DCF77 error messages			**16.4**
Time-out	"E"	"0"	
Short signal	"E"	"1"	
Between zero/one	"E"	"2"	
Between one/pause	"E"	"3"	
Between pause/minute	"E"	"4"	
Longer than minute	"E"	"5"	
<>59 bits	"E"	"6"	
Number of bits	"B"	(Number of received bits)	
Parity minutes odd	"E"	"7"	
Minute ones > 9	"E"	"8"	
Minutes > 59	"E"	"9"	
Parity hours odd	"E"	"A"	
Hours ones > 9	"E"	"B"	
Hours > 23	"E"	"C"	
Parity date odd	"E"	"D"	
Weekday = 0	"E"	"E"	
Day = 0	"E"	"F"	
Day ones > 9	"E"	"G"	
Day > 31	"E"	"H"	
Month = 0	"E"	"I"	
Month ones > 9	"E"	"J"	
Month > 12	"E"	"K"	
Year ones > 9	"E"	"L"	
Year > 99	"E"	"M"	
Debugging messages			**32.8**
Buffer filled	"a"	(Average value)	
Delta max - min	"d"	(Difference)	

All transmitted time and data of a successfully evaluated DCF77 signal set require approximately 100 ms. This can be used to adjust the seconds counter of the receiver clock to synchronize the start of the next minute.

The debugging messages are only send if the respective debugging switches are set to **Yes**.

The signals that the transmitter produces look like shown here (left: 10 kBd, right: 20 kBd). Send here was 0xAAAA (ones and zeroes). The data signal on PB0 (SDM) is red, the clock signal on PB2 (SCM) is green.


Channel 1: PB3
Channel 2: PB1
Send message
1
0
f=9.615kHz
PW= 48.1%
f=4.831kHz
PW= 49.3%
250µs
500µs
750µs
1ms
1.25ms
1.5ms
1.75ms
t

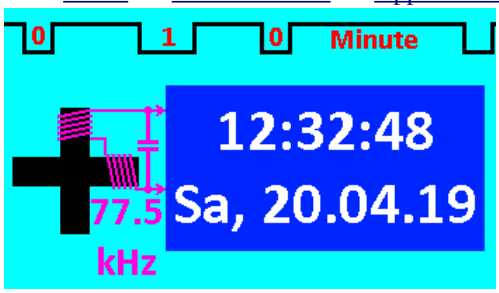

Channel 1: PB3
Channel 2: PB1
Send message
1
0
f=18.868kHz
PW= 49.1%
f=9.524kHz
PW= 48.6%
100µs
200µs
300µs
400µs
500µs
600µs
700µs
800µs
900µs
t

5.4 Software

$$

5.5 Operation experiences

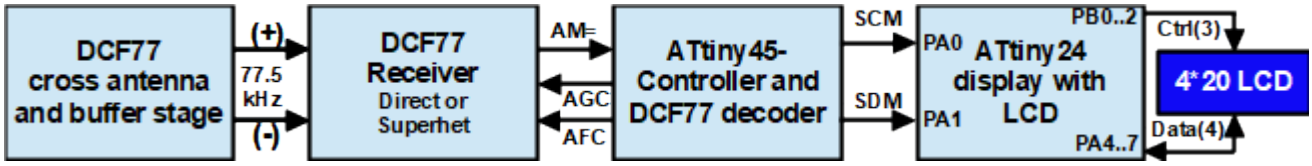
$$



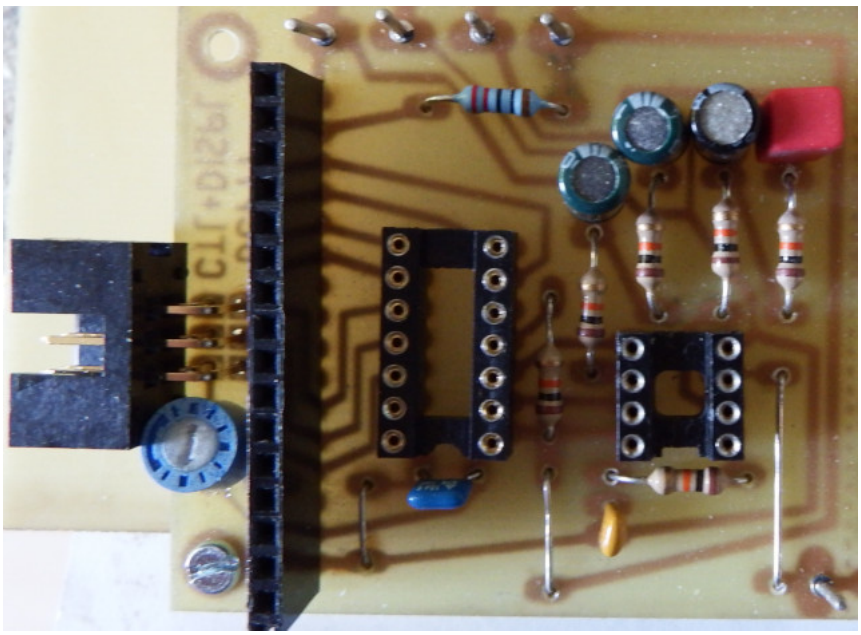
Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 display with ATtiny24

6 DCF77 display with an ATtiny24



This device



1. receives the serial signals from an ATtiny45 decoder and displays those on an LCD,
2. is working with the ATtiny24 experimental device with LCD and can be attached using a simple six pin plug,
3. uses the LCD include software as shown here to drive the LCD, and
4. has an additional seconds counter and display that simply counts from 0 to 59.

The device can be used as a complete display for all DCF77 receivers as described on that webpage, providing time and date from the receiver as well as status and error messages.

The displayed device works a little bit different: the ATtiny24 is mounted, together with the ATtiny25 controller, on one PCB. Wiring is slightly different, but not very much.

6.1 Connecting the device with the receiver

With a six pin flat cable, both sides equipped with a plug, connect

1. pin 1 with GND (minus operating voltage),
2. pin 2 with +5 V operating voltage,
3. pin 3 with the serial clock pin of the ATtiny45 controller (SCM), and
4. pin 6 with the serial data pin of the ATtiny45 (SDM).

See the schematic of the Tiny24-LCD experimental display device here. All you need to do is programming the ATtiny24's flash memory with the software provided here.

6.2 Display

When starting and without any serial input signals the program starts like this.

In line 2 behind **Raw data:** the incoming raw data is displayed. The first character (MSB in **rRxH**) is simple ASCII, then **rRxL** follows in hexadecimal format. If the displayed first character is an **F:** then the PWM value of the frequency adjustment follows as a three-digit decimal.

The incoming DCF77 data for the time (hours and minutes) follow in line 3. The seconds display is internally generated and counts even without DCF77 signal. Behind the ASCII character **S:** the PWM value of the gain adjustment follows in 3-digit decimal format.

Line 4 displays the weekday and the date, with format depending from the language chosen, and any signal errors behind **E:**.

The display changes when the ATtiny45's SCM and SDM are connected.

6.3 Software for the ATtiny24

The software can be downloaded as assembler source code from here or can be displayed in the attachment here.

The following chapters describe the software's functions further.

6.3.1 Reception of the serial signals

Serial reception works as follows:

1. On each level change on the PA0 input pin (SCM signal) a PCINT0 interrupt is executed.
2. If the PA0 pin is high, following level change, a bit on the data input pin PA1 has to be shifted into the register pair for serial signals (**rSerialH:rSerialL**), starting from bit 0 (rotate left). The number of received bits is down-counted from 16, if it reaches zero the content of the serial registers is copied to the register pair **rRxH:rRxL**, the flag **bRxIn** is set and the counter restarts with 16. Any bit reception sets the time-out counter **rSerialTO** to its start value.
3. Outside the PCINT0 routine the time-out-counter is counted down. If it reaches zero, the counter of still-to-be-received-bits is restarted at 16.

The further handling of the 16 bits received in **rRxH:rRxL** is performed outside the Interrupt-Service-Routine. Depending from the MSB in **rRxH** the lower 8 bits in **rRxL** are handled differently. The modes are part of a table **PosTable:**:

1. Each line of the table starts with the respective ASCII character.
2. The two following bytes are the line and the column where information on the LCD will be displayed (zero-based numbers).
3. If the parameter concerns a DCF77 info, its displacement in SRAM follows as a byte. If not, 255 follows.
4. The type of displaying the parameter follows as byte. A zero simply displays the parameter as ASCII character. A 1 displays as a 2-digit decimal, a 2 as 3-digit decimal. A 3 displays as a weekday (zero-based) and a 4
    - shifts the received zero or one into a register buffer, and
    - displays the last four received bits as binary ASCII characters.
5. The closing zero in each line brings the byte count to an even number of bytes.
6. The table ends ends with two zeroes.

Extending the ATtiny45 controller software with further parameters can simply be added to this table. The structure can also be used with other controllers.

6.3.2 Seconds and serial interface time-out

For generating the seconds the timer TC0 is used. It divides the clock signal of 1 MHz (default clock rate) with a prescaler value of 8 and by 125, which yields a 1 ms time interrupt. The interrupt counts down a counter in register pair **rSecH:rSecL** that starts with 1,000. If the counter reaches zero, the flag **bSec** is set and the register pair is restarted with 1,000.

If the seconds flag is set, the seconds are advanced and displayed at the respective position on the LCD. If the seconds reach 60, the seconds counter restarts.

The milli second interrupt also decreases the time-out value. If, after 8 ms, this reaches zero the number of bits-to-be-received restarts with 16.

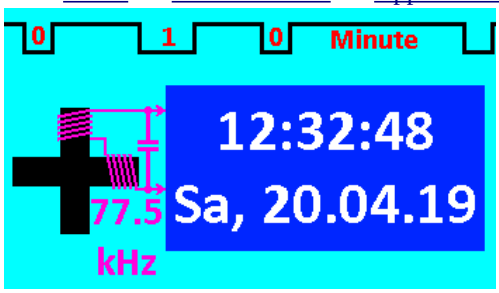
6.3.3 Debugging option

This option, if the debug switch **Debug_display** is set to **Yes**, simulates the reception of serial signals. The bit combination in the constant **cDebug_displayH** and **cDebug_displayL** is tested. The debug switch has to be set to **No** to assemble the final version of the software.

6.4 Assembler source code for the DCF77 display with ATtiny24

This is the software for the DCF77 display with an ATtiny24. The source code can be downloaded from here in asm format. To assemble it needs the Include routines here.

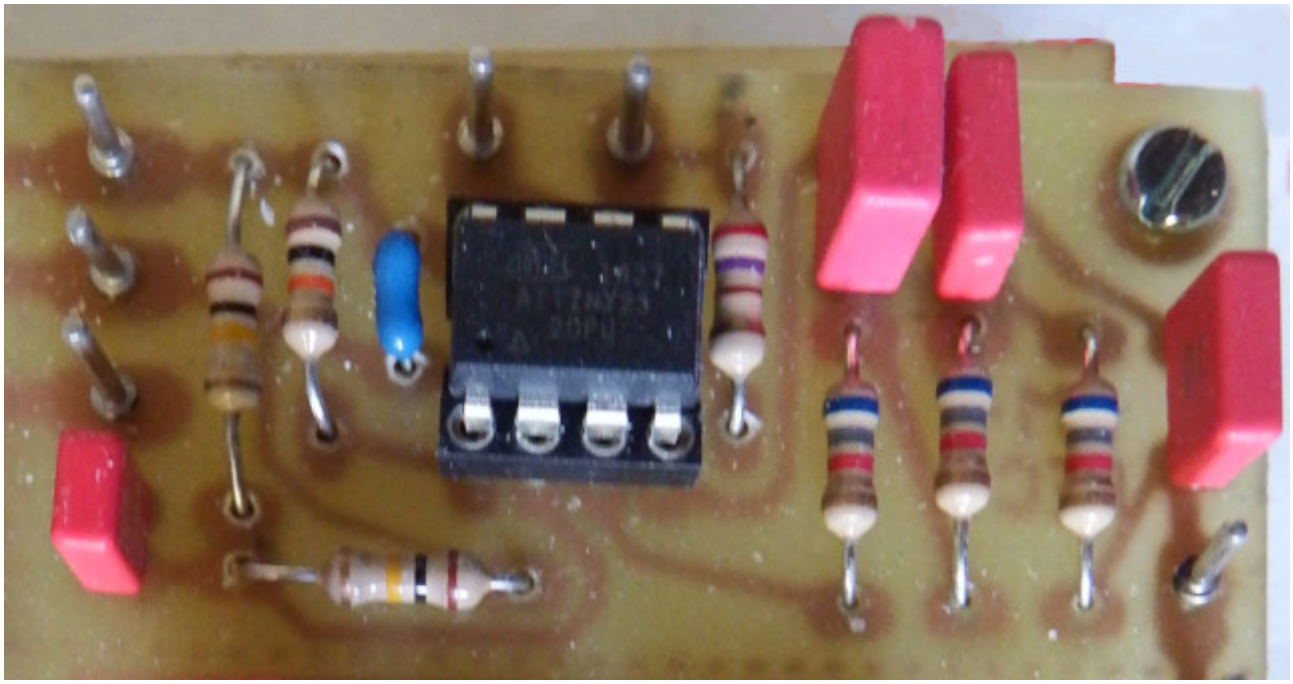
```
;
; (Software still under construction)
;
```



Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

AM rectifier with ATtiny25

7 DCF77 AM rectifier with ATtiny25



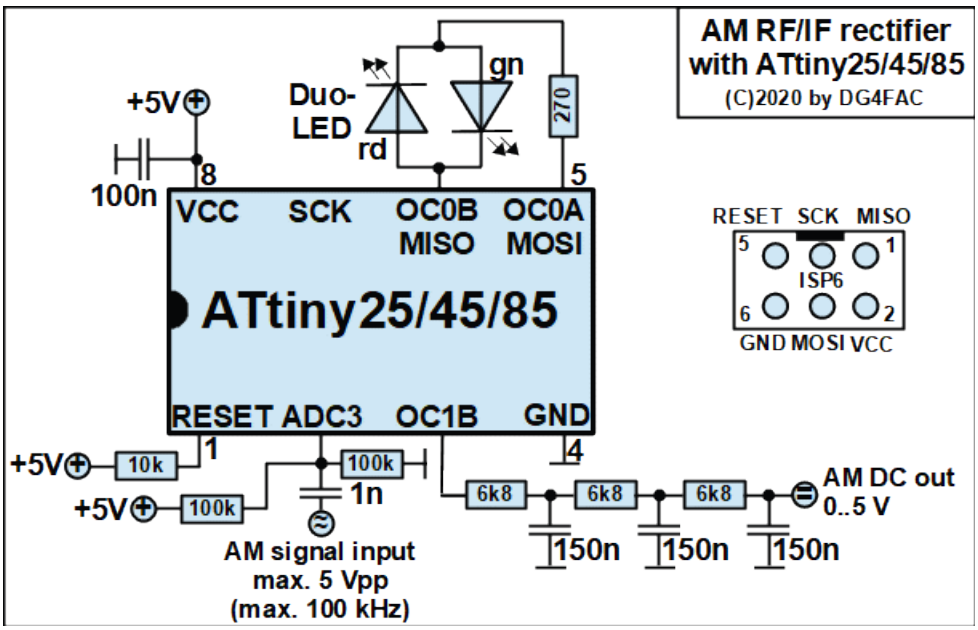
For the DCF77 signal of 77.5 kHz from a direct receiver or for the IF of 32.768 kHz from a superhet one needs a rectifier for AM signals. Usually one uses a diode rectifier. But as diode rectifiers for AM RF/IF always have the disadvantage that their forward voltage of 0.2 to 0.3 Volt won't let them detect signal amplitudes smaller than that 0.2 to 0.3 V, I designed and built a microcontroller rectifier. This device detects amplitudes of down to 5 mV_{pp}.

Note that this design is limited to frequencies below 100 kHz, so the device is not designed and doesn't work correct with an IF of 455 kHz or higher!

7.1 How it works

The rectifier works with a microcontroller, here an ATtiny25, and its built-in ADC channel 3. The controller runs with its built-in RC oscillator at a clock rate of 8 MHz (CLKPR is set to 1 by software).

7.1.1 Hardware



The ADC is working in free running mode, that is: he restarts the next conversion whenever the conversion is complete. The ADC clock prescaler is at 4, so the ADC runs with 2 MHz. As each conversion in free running mode takes 13.5 cycles, the ADC works with a conversion frequency of 2 MHz / 13.5 = 148.2 kHz and one conversion lasts 6.75 µs.

The RF of 77.5 kHz or the IF of 32.768 kHz comes in as a sine wave with an amplitude between 0.0 and 5.0 V_{pp}. The voltage divider with the two 100kΩ resistors divides the operating voltage by two, the 1nF capacitor transfers the RF/IF to the ADC3 input of the ATtiny25, so that the sine wave produces the following 10-bit ADC conversion results from the input voltage:

- no amplitude: ADC result approximately 512,
- positive swing: ADC result larger than 512,
- negative swing: ADC result smaller than 512.

So all the controller has to do is to

1. subtract 2 from the MSB of the result,
2. if that yields the carry flag being set: to invert the result.

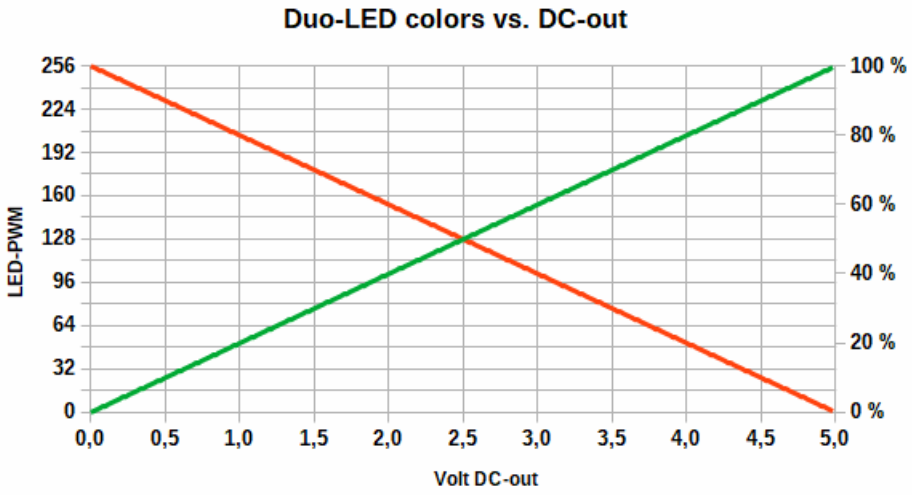
That rectifies the signal: negative swings will get positive, positive ones remain positive (Rectification). The result is nine bits wide (0 to 511).

This is repeated over and over again (with the ADC in free running mode) for 128 times, hence over 128 * 6.75 = 864 µs. The maximum of these 128 measurements is the highest/lowest detected amplitude of the input swing. This maximum is selected from 128 measurements of 28 (32.768 kHz, 30.5 µs per wave) or 67 (77.5 kHz, 12.9 µs per wave) sine waves, so that the maximum is very likely to be detected.

The detected maxima are added up 8 times, the result is divided by 16 to yield an 8-bit average value for the PWM's compare B value. With that the measuring time is 8 * 864 µs = 6.91 ms.

7.1.2 Duo-LED option

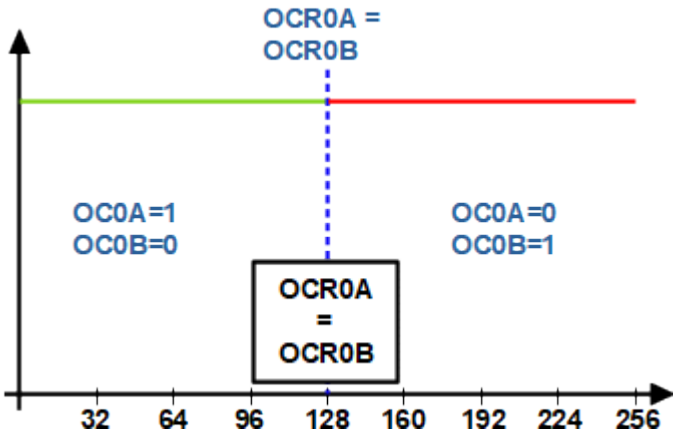
The device can be equipped with a red/ green or red/yellow duo-LED to display the results. Then display on the LED works as follows. With small amplitudes red dominates, the higher the more greener the LED gets.



This is achieved by setting the comparer values of TC0 to the same value, the TC1's PWM value in OCR1B. The OC0-bit-behavior is to set OC0A on the beginning of the PWM cycle and to clear the OC0A on compare match (un-inverted). The OC0B is reversed: it is set at the beginning and cleared on compare match. This yields the following behavior:

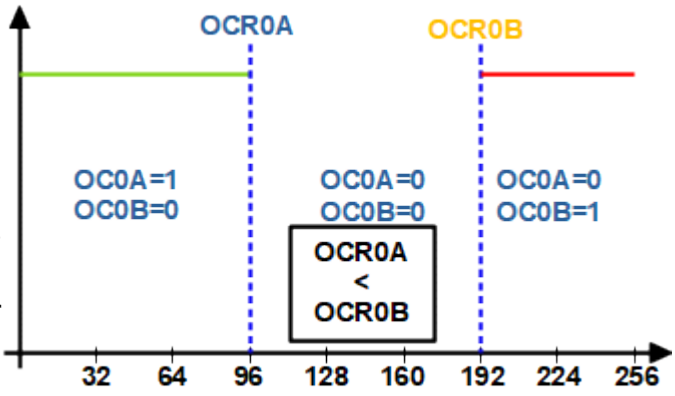
When both comparers have a compare match, the color changes from green to red. If the compare match occurs at 255, the red LED is never on.

The earlier the longer is the LED red on and the green LED off. The time over which one of the two colors are on is always 100%.

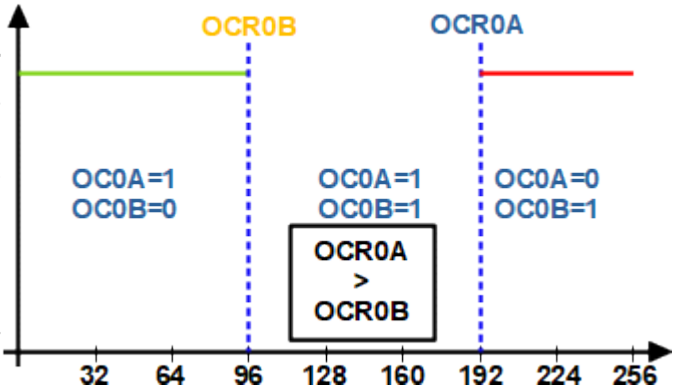


If OCR0A is different from the OCR0B value, the behavior is rather different. As the OC0A output is set at PWM cycle start, while OC0B is cleared, the green LED is turned on.

This stays on until OCR0A is reached: this clears OC0A and, as OC0B is also cleared, switches both LEDs off. This changes later on, when OCR0B is reached: this switches the red LED on. This remains on until the end of the PWM cycle.

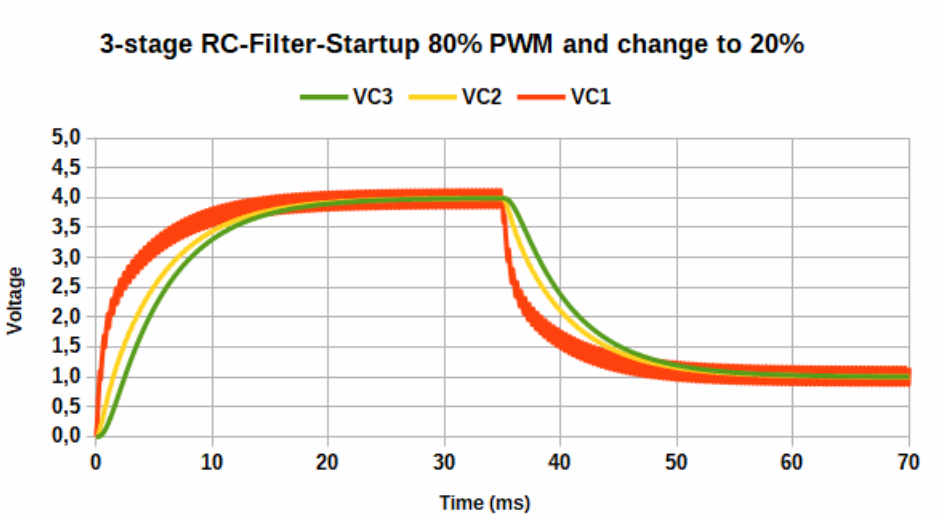


This is also the case if OCR0A is larger than OCR0B: the inactive time now has both pins set. This mode reduces the brightness of both LEDs, the larger the difference between OCR0A and OCR0B the longer the pause with both LEDs off. The duration of the green on and the red on can still be altered, but only to the available rest of the time.



7.1.3 DC Output

TC1 produces the output. On PWM cycle start the output pin OC1B is set, on reaching compare match B it is cleared (positive PWM signal).



The averaged amplitude feeds the comparer of the 8-bit-TC1 timer that runs as PWM with the compare B in fast PWM mode. The timer is clocked by the 8 MHz controller clock and without prescaling. The PWM runs with a frequency of 31.25 kHz, one cycle lasts 32 µs. The PWM value is updated every 13.82 ms / 32 µs = 423 cycles.

The OC1B output signal is filtered by a 3-stage RC filter with R=6k8 and C=150n. The time constant of this RC network is t = 0.69 * R * C = 0.7 ms. The diagram shows the response of the filter on startup with an 80% PWM pulse and to a level change to a 20% PWM pulse after 35 ms.

The approximately 15 to 20 ms that the RC network needs to swing to a new value is short enough to detect 100 ms long amplitude drops for a transmitted 0-bit of DCF77 and a 200 ms long voltage drop for a 1-bit.

The PWM hum of the RC network is

1. stage 1 (VC1, red): 0.29 V_{pp},
2. stage 2 (VC2, yellow): 17 mV_{pp},
3. stage 3 (VC3, green): 5 mV_{pp}.

So the PWM noise on the third stage is roughly one digit of a 10-bit ADC result and the RC provides sufficient filtering.

Averaging over 16 measurements ensures that single failures to detect the maximum of the input signal are smoothly handled.

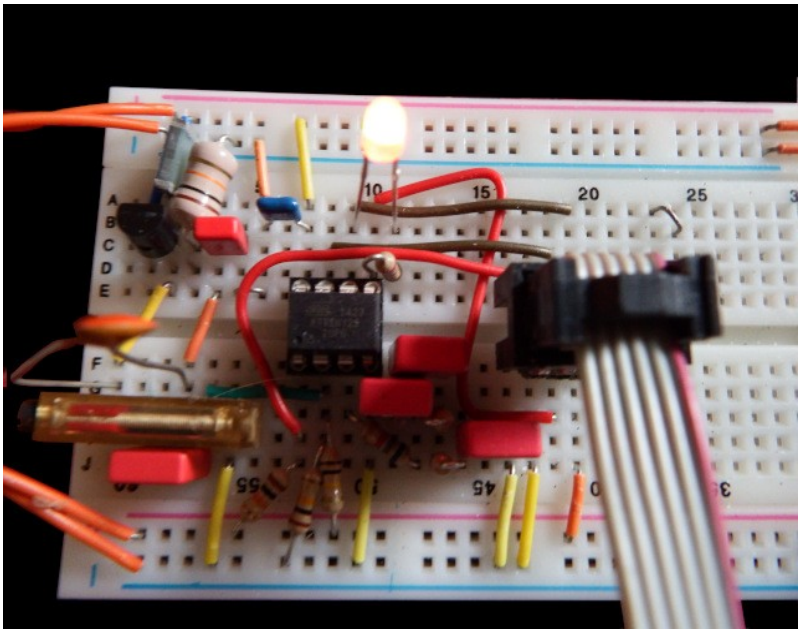
7.1.4 Available resources

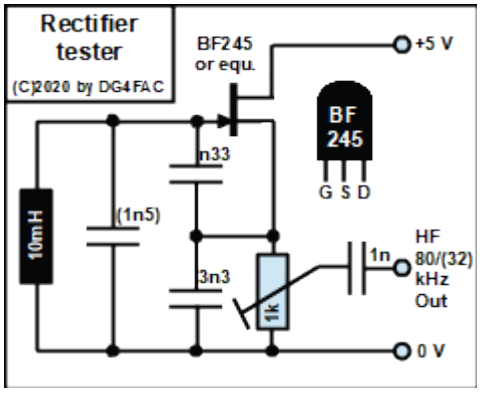
All drawings shown here are available as a LibreOffice Draw file here (see the last drawing for the rectifier).

The LibreOffice Calc file am-rect_tn25.ods provides all calculations that can be useful when changing the design and properties of the device (ADC sampling, clocking, RC-filter and filter response, Duo-LED configuration, etc.). Please note that if you play around with the values therein: do not increase the speed of the AD converter, the prescaler will have to 4 or higher. Otherwise the interrupt service routine is not fast enough to handle all the interrupts and blocks other program steps from being served.

7.2 Testing

I've tested the rectifier by using the generator to the right. It generates a sine wave either with 77.5 kHz (no parallel C) or 32.8 kHz (1n5 parallel to the coil). The signal is not a very nice sine wave, but it fits to the rectifier.





The potentiometer in the source line of the FET allows to dim the signal strength.

This is the rectifier on a test stand. Unfortunately the generator does not provide a full 5 V swing, so you do not get the Duo-LED to full green color.

7.3 Software for the rectifier

The software for the ATtiny25 is written in assembler. The source code is here in assembler source code format and is listed below. No fuses have to be changed, the increase of the clock frequency from the 1 MHz default to 8 MHz is done by software.

The program is interrupt-driven. No sleep mode is used because the fast ADC sampling eats up most of the time.

When assembling make sure that the two settings "DuoLed" and "LedOnly" are set to your needs:

- If you do not need the Duo-LED, switch it off by setting DuoLed to Zero.
- If you want to experiment with the Duo-LED only set LedOnly to 1. If you program the final, don't forget to set it to Zero again.

The program has been tested and works fine as designed.

The original source code for the ATtiny25 rectifier is here. This is only a HTML formatted listing.

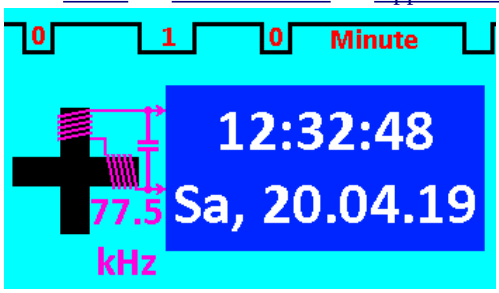
```
;
; ****************************
; * AM rectifier with ATtiny25 *
; * (C)2020 by DG4FAC          *
; ****************************
;
.nolist
.include "tn25def.inc" ; Define device ATtiny25
.list
;
; *********************************
;       P R O P E R T I E S
; *********************************
;
.equ Duoled = 1 ; 1 if Duo-LED attached
.if DuoLed == 1
  .equ DuoledSpeed = 2 ; Speed factor for Duo-LED
    ; 1: 30.5 Hz
    ; 2: 122.1 Hz
    ; 3: 488 Hz
    ; 4: 3.91 kHz
    ; 5: 31.25 kHz (use this when simulating)
  ; Test the Duo-LED only
  .equ LedOnly = 0 ; Display red/green only
  .if LedOnly == 1 ; Define colors
     ; The intensities of the LED colors
     .equ cLedOnlyColor = 0x20 ; Green=0x20, red=0xC0
        .endif
  .endif
.equ MaxAverage = 8 ; Number of maxima to be averaged, 2..128
.equ cMaxCnt = 32 ; Count of measurements for detecting maximum
;
; Error checking
.if (MaxAverage<2) || (MaxAverage>128)
  .error "Illegal MaxAverage value selected!"
  .endif
.if (2<<(LOG2(MaxAverage)-1)) != MaxAverage
  .error "MaxAverage has to be a power of 2!"
  .endif
.if (MaxAverage*cMaxCnt)>1024
  .message "Warning: Averaging too long for DCF77!"
  .endif
.if cMaxCnt<16
```

```
  .message "Warning: Too few countings for max detection!"
  .endif
.if cMaxCnt>255
  .error "Too many counting events for max detection!"
  .endif
.if (DuoLedSpeed<1)||(DuoLedSpeed>5)
  .error "Wrong DuoLedSpeed setting!"
  .endif
;
; *********************************
;       H A R D W A R E
; *********************************
;
; Device: ATtiny25, Package: 8-pin-PDIP_SOIC
;
;                 _________
;          1 /         |8
;  RESET o--|RESET  VCC|--o VCC, +5V
;  AM-IN o--|PB3    PB2|--o SCK
; DC-OUT o--|PB4    PB1|--o Red Anode, MISO
; 0V GND o--|GND    PB0|--o Green Anode, MOSI
;          4 |__________|5
;
;
; *********************************
;  P O R T S   A N D   P I N S
; *********************************
;
.equ pOutLedD = DDRB ; Portregister of LED
.if DuoLed==1
  .equ bOutLedD = (1<<PORTB0)|(1<<PORTB1)|(1<<PORTB4) ; DC output pin and Duo-Led
  .else
  .equ bOutLedD = 1<< PORTB4 ; Only the DC output for PWM
  .endif
;
; *********************************
;   A D J U S T A B L E   C O N S T
; *********************************
;
;
; *********************************
;  F I X  &  D E R I V.  C O N S T
; *********************************
;
.set cTc0Presc = (1<<CS02)|(1<<CS00)
.if DuoLedSpeed == 2
  .set cTc0Presc = 1<<CS02
  .endif
.if DuoLedSpeed == 3
  .set cTc0Presc = (1<<CS01)|(1<<CS00)
  .endif
.if DuoLedSpeed == 4
  .set cTc0Presc = 1<<CS01
  .endif
.if DuoLedSpeed == 5
  .set cTc0Presc = 1<<CS00
  .endif
;
; *********************************
;       T I M I N G
; *********************************
;
; Clock = 8000000
; ADC prescaler = 2
; ADC conversion steps = 13
; ADC conversion frequency = 307.69 kHz
; ADC conversion time = 3.25 us
; Measurements per sine wave
```

```
;   at 77.5 kHz = 3.97
;   at 32.768 kHz = 9.39
;
; Maximum detection
;   Number of measurements = 32
;   Sampling time = 832 us
;   Sine waves measured
;     at 77.5 kHz = 64.48
;     at 32.768 kHz = 27.28
;
; Averaging maximums
;   Averaging = 16
;   Averaging time = 13.82 ms
;
; *********************************
;       R E G I S T E R S
; *********************************
;
; free: R0 to R8
.def rAvgL = R9 ; Average sum, LSB
.def rAvgH = R10 ; dto., MSB
.def rInL = R11 ; Maximum detected, LSB
.def rInH = R12 ; dto., MSB
.def rMaxL = R13 ; Maximum LSB
.def rMaxH = R14 ; dto., MSB
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
.def rMaxCnt = R17 ; Maximum counter
.def rAvgCnt = R18 ; Average counter
.def rAdcL = R19 ; ADC value read, LSB
.def rAdcH = R20 ; dto., MSB
; free: R22 to R31
;
; *********************************
;           S R A M
; *********************************
;
.dseg
.org SRAM_START
; No SRAM used (only for stack)
;
; *********************************
;         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 ; PCI0
       reti ; OC1A
       reti ; OVF1
       reti ; OVF0
       reti ; ERDY
       reti ; ACI
       rjmp AdccIsr ; ADCC interrupt
       reti ; OC1B
       reti ; OC0A
       reti ; OC0B
       reti ; WDT
       reti ; USI_START
       reti ; USI_OVF
;
; *********************************
```

```
;  I N T - S E R V I C E   R O U T .
; ********************************
;
AdccIsr:
  in rSreg,SREG ; Save SREG
  in rAdcL,ADCL ; Read ADC LSB
  in rAdcH,ADCH ; dto., MSB
  subi rAdcH,0x02 ;
  brcc AdcIsr1 ; No carry
  neg rAdcL ; Negative LSB
  com rAdcH ; dto., MSB
AdcIsr1:
  cp rAdcL,rMaxL ; Compare with maximum
  cpc rAdcH,rMaxH
  brcs AdcIsr2 ; Smaller
  mov rMaxL,rAdcL
  mov rMaxH,rAdcH
AdcIsr2:
  dec rMaxCnt ; Count measurements
  brne AdcIsr3
  ldi rMaxCnt,cMaxCnt ; Restart counter
  mov rInL,rMaxL ; Copy maximum, LSB
  mov rInH,rMaxH ; dto., MSB
  clr rMaxL ; Restart maximum, LSB
  clr rMaxH ; dto., MSB
  out SREG,rSreg ; Restore SREG
  set ; Set input flag
  reti
AdcIsr3:
  out SREG,rSreg ; Restore SREG
  reti
;
; ********************************
;  M A I N   P R O G R A M   I N I T
; ********************************
;
Main:
.ifdef SPH ; If an ATtiny85 is used
  ldi rmp,High(RAMEND)
  out SPH,rmp ; Init MSB stack pointer
  .endif
       ldi rmp,Low(RAMEND)
       out SPL,rmp ; Init LSB stack pointer
; Increase clock to 8 MHz
  ldi rmp,1<<CLKPCE ; Enable CLKPR change
  out CLKPR,rmp ; in CLKPR
  clr rmp ; No prescaler
  out CLKPR,rmp ; in CLKPR
; Start values
  clt ; Input flag off
  clr rMaxL ; Maximum clear, LSB
  clr rMaxH ; dto., MSB
  clr rMaxCnt ; One full cycle max detection
  ldi rAvgCnt,MaxAverage ; Average over values
; Output pins as output
  ldi rmp,bOutLedD ; Output pins as output
  out pOutLedD,rmp ; in direction port
; Start Duo-LED
.if DuoLed == 1
  ldi rmp,0x80 ; Set both compare values to zero (LED half)
  out OCR0A,rmp ; Compare A
  out OCR0B,rmp ; Compare B
  ldi rmp,(1<<COM0A1)|(1<<COM0B1)|(1<<COM0B0)|(1<<WGM01)|(1<<WGM00) ; Clear OCR on match,
Fast PWM mode
  out TCCR0A,rmp ; To timer control port A
  ldi rmp,cTc0Presc ; Set prescaler
  out TCCR0B,rmp ; in timer control port
  .endif
```

```
; Start PWM on TC1
  clr rmp ; Compare A and B to zero
  out OCR1A,rmp
  out OCR1B,rmp ; To compare port B
  ldi rmp,0xFF ; 8-bit PWM
  out OCR1C,rmp ; in compare port C
  ldi rmp,(1<<PWM1B)|(1<<COM1B1) ; PWM on OC1B
  out GTCCR,rmp ; in control port GTCCR
  ldi rmp,(1<<CS10) ; Prescaler=1, resolution in OCR1C
  out TCCR1,rmp ; in control port 1
; Init ADC
  ldi rmp,(1<<MUX0)|(1<<MUX1) ; Channel ADC3
  out ADMUX,rmp ; To ADC mux
  ldi rmp,0 ; ADC free running mode
  out ADCSRB,rmp ; in control port B
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADATE)|(1<<ADIE)
  out ADCSRA,rmp ; Start first conversion in control port A
; Enable interrupts
       sei ; Enable interrupts
;
; ********************************
;    P R O G R A M   L O O P
; ********************************
;
Loop:
  brtc Loop ; Input bit not set
;
; Input the next maximum
Input:
  clt ; Clear input flag
.if (DuoLed==1)&&(LedOnly==1)
  ldi rmp,cLedOnlyColor ; Set color
  out OCR0A,rmp
  out OCR0B,rmp
  .else
  add rAvgL,rInL ; Add result, LSB
  adc rAvgH,rInH ; dto., MSB
  dec rAvgCnt ; Decrease counter
  brne Loop ; If not zero, continue
   ; Calculate average
  ldi rmp,LOG2(MaxAverage)+1
Input0:
  lsr rAvgH ; Divide average by 2
  ror rAvgL
  dec rmp ; Count down
  brne Input0 ; Continue dividing
; Set PWM value
  out OCR1B,rAvgL ; Set PWM compare B
  .if DuoLed==1
    out OCR0A,rAvgL
    out OCR0B,rAvgL
    .endif ;
  .endif
InputOut:
  ldi rAvgCnt,MaxAverage ; Restart counter
  clr rAvgL ; Clear average sum
  clr rAvgH
  rjmp Loop
;
; End of source code
;
; (Add Copyright information here, e.g.
; .db "(C)2020 by Gerhard Schmidt  " ; Source code readable
; .db "C(2)20 0ybG reahdrS hcimtd  " ; Machine code format
;
```

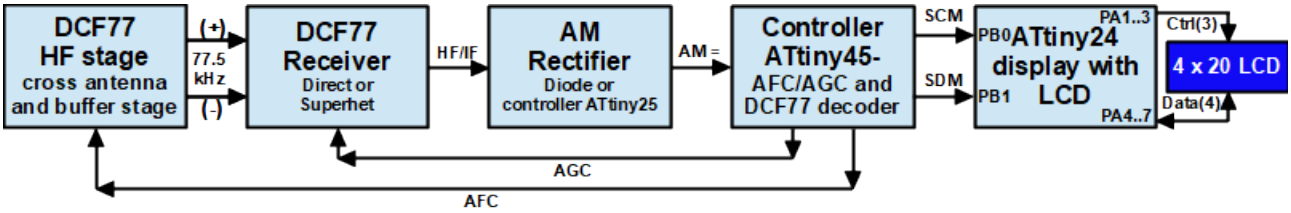


Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 PCB layouts

8 DCF77 PCB layouts

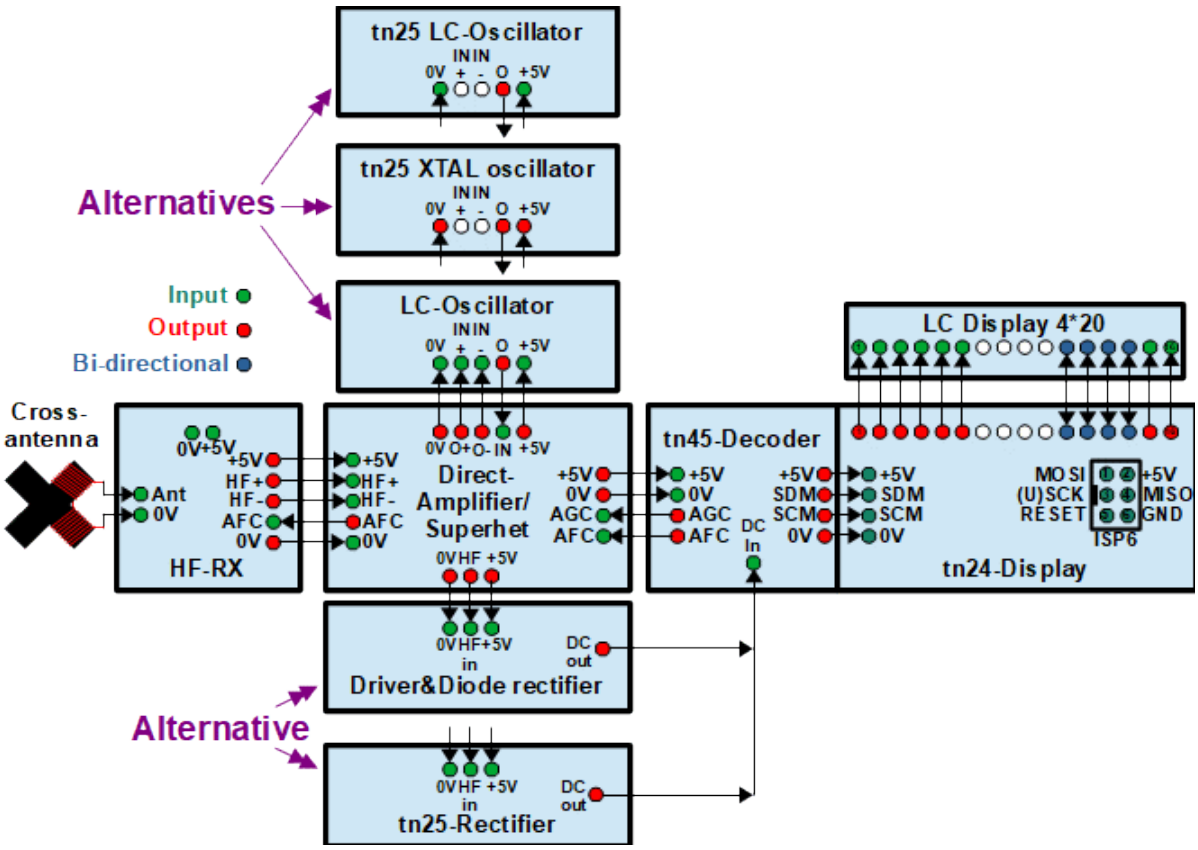
All described receivers use an RF input stage, though slightly different for large and for small ferrite coils, as well as the ATtiny25 controller plus the ATtiny24 display stage. I have developped a versatile PCB that allows to suit to all the variations that are described here. A PCB with any desired concept can be scissored. All you need is a graphics designer, such as PaintNet or GIMP for example, or, even better and more comfortable, the TGIF vector graphics program for Linux.



All variations fit well on a half Euro-size PCB (80x100 mm), as shown below.

8.1 Modules and connections

This here shows the different modules and their related connections.



8.1.1 The HF-RX module

This module has a size of 20x80 mm. It consists mainly of the FET and the capacitor diodes for frequency adjustment. Two different variations, that differ only in the diode's connections, are here available:

a. with anti-parallel diodes, e. g. with a KV1235Z and only two diodes used, or a BB212, for larger ferrite coils, or
b. with parallel diodes, e. g. with the three KV1235Z diodes or three BB112, for smaller ferrite coils.

This module has the following external connections:

1. the power supply for the whole receiver of +5 V, and
2. the cross antenna (or any other ferrite coil).

The module has the following in- and outputs:

I. the operating voltage GND and + 5V to the amplifier/superhet,
II. the two symmetric RF outputs HF+ and HF-,
III. the AFC input for adjusting the frequency.

8.1.2 The Direct-Receiver modules

The two modules for directly receiving DCF77 with transistors or the one with the TCA440's IF amplifier are both of a size of 30x80 mm. The following connections are available:

- To the RF receiver module:
    1. the operating voltage of +5 V, and
    2. the two symmetric RF inputs, and
    3. the AFC output,
- To the controller- and display module:
    1. the operating voltage of +5 V, and
    2. the two inputs AFC and AGC,
- To the rectifier module:
    1. the operating voltage of +5 V, and
    2. the RF/IF output.

8.1.3 The Superhet receiver modules

(These modules are still under construction.)

8.1.4 The AM rectifier modules

The two alternative rectifier modules, either with diodes or with an ATtiny25 controller, rectify the RF or IF and hand the result over to the controller/display module. The operating voltage for that stage stems from the Amplifier/Superhet stage.

8.1.5 The Control- and Display-module

This module is identical for all receiver variations. The controller ATtiny25 measures the voltage that comes from the rectifier, and generates (via PWM plus RC filter) the AGC- and the AFC-voltages. The AFC voltage line crosses the amplifier/superhet module to get to the RF-RX stage. The DCF77 results are transmitted via the internal Two-Wire-Interface to the ATtiny24, which displays the results on the connected LCD.

This module has additionally an ISP6 interface connector. This can supply the operating voltage from an attached ISP programming adapter, if the programming adapter provides 5 Volt. If one of the three controllers shall be flashed, the programming pins RESET, MOSI, MISO and SCK can be temporarily attached to the ISP6 plugs using enameled wires.

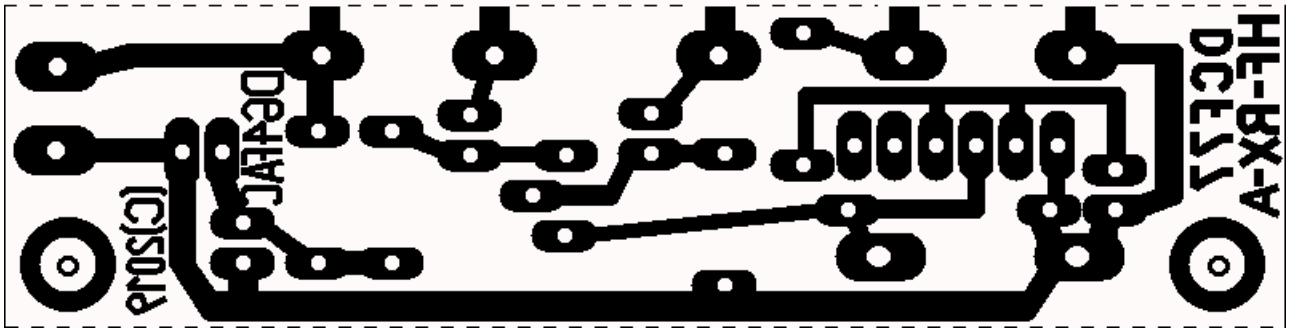
8.2 Links to the PCB layouts

The following parts are incomplete because not all layouts are complete.

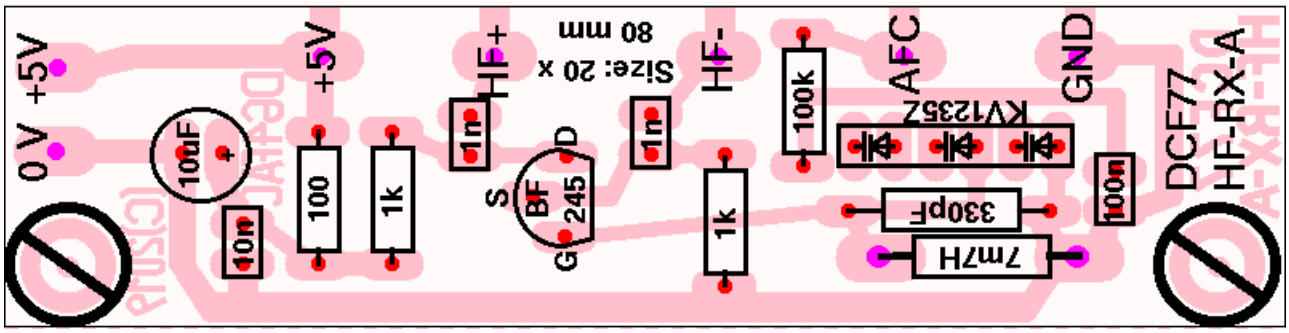
To download the files place the mouse over the entry and, in the context menu, select "Save linked file as ...".

8.2.1 Layouts for the modules

The PCB layouts of the modules are available here as gif graphics files, e. g. for the RF-RX part those look like:



The associated component placement plan looks like this:

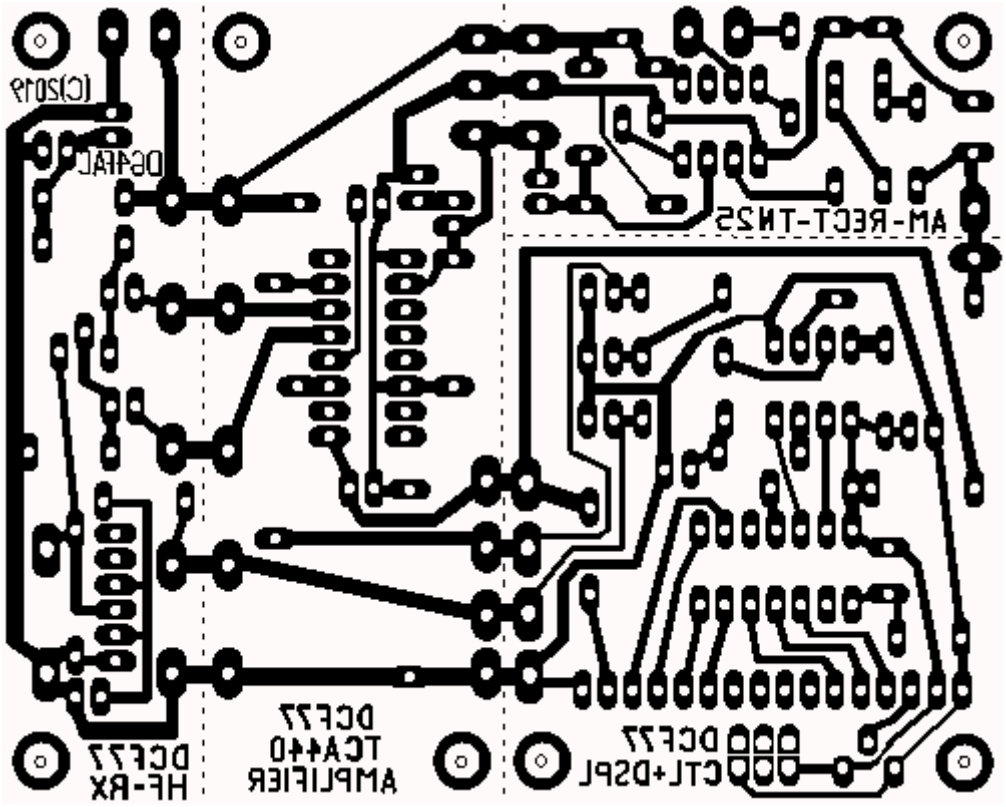


Module	Size	Variation	PCB layout		Component plan	
	mm		gif	tgif-obj	gif	tgif-obj
HF-RX	20x80	Anti-parallel	HF-RX-A	HF-RX-A	HF-RX-A	HF-RX-A
		Parallel	HF-RX-P	HF-RX-P	HF-RX-P	HF-RX-P
Direct receiver	30x80	with Transistors	Trans	Trans	Trans	Trans
		with TCA440	TCA440	TCA440	TCA440	TCA440
Superhet	30x80	internal LC-oscillator	SuperLC	SuperLC	SuperLC	SuperLC
		divided XTAL	SuperXT	SuperXT	SuperXT	SuperXT
	50x80	regulated LC-oscillator	SuperRegLC	SuperRegLC	SuperRegLC	SuperRegLC
AM rectifier	50x23	with diodes	Diodes	Diodes	Diodes	Diodes
		with ATtiny25	Rect25	Rect25	Rect25	Rect25
Controltn45Display-tn24	50x57		CtrlDisp	CtrlDisp	CtrlDisp	CtrlDisp

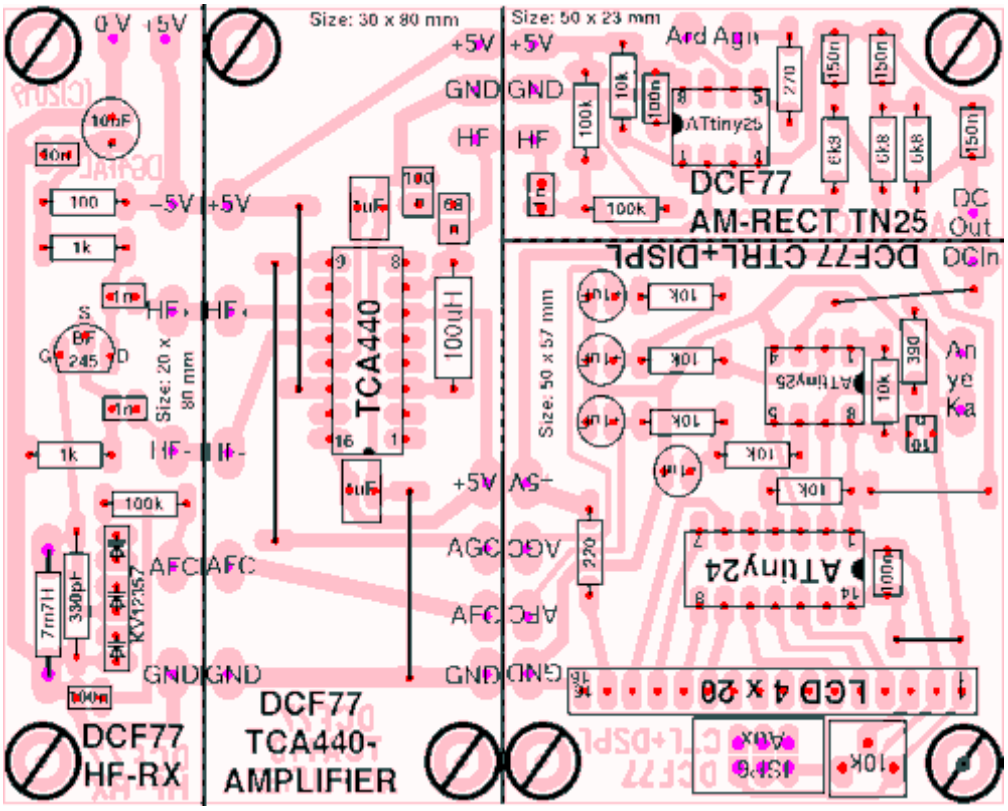
8.2.2 Complete PCB layouts with combinations

All PCBs linked here are 100x80 mm (half Euro-Format), except the superhet with regulated LC oscillator (120x80 mm).

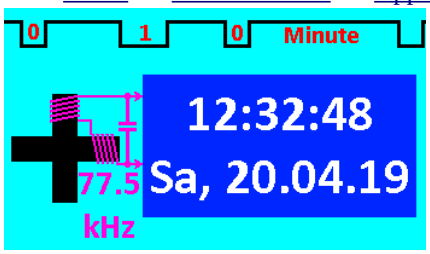
Here for example is the combination of HF-RX-a, TCA440-Direct receiver, ATtiny25-Rectifier and ATtiny25-Controller/ATtiny24-Display on one single PCB. Layout and component placement are downsized (the originals below are double that size;!)



This is the associated component placement plan.



Short name	HF-RX		Direct receiver		Superhet with oscillator option			Rectifier		Con-trol + Dis-play	PCB layout		Component placements	
	Anti-paral-lel	Paral-lel	Tran-sis-tors	TCA 440	Inter-nal LC	XTAL Divi-der	Regu-lated LC	Dio-des	AT-tiny 25		as gif	for tgif	as gif	for tgif
Direct Trans.		X	X					X		X	Trans	Trans	Trans	Trans
Direct TCA440	X			X					X	X	TCA-440	TCA-440	TCA-440	TCA-440
Super-LC	X				X				X	X	Super-LC	Super-LC	Super-LC	Super-LC
Super-XTAL		X				X			X	X	Super-XT	Super-XT	Super-XT	Super-XT
Super-LC25	X						X		X	X	Super-RegLC	Super-RegLC	Super-RegLC	Super-RegLC



Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and ATxmega

DCF77 Alarm clock with ATmega324PA

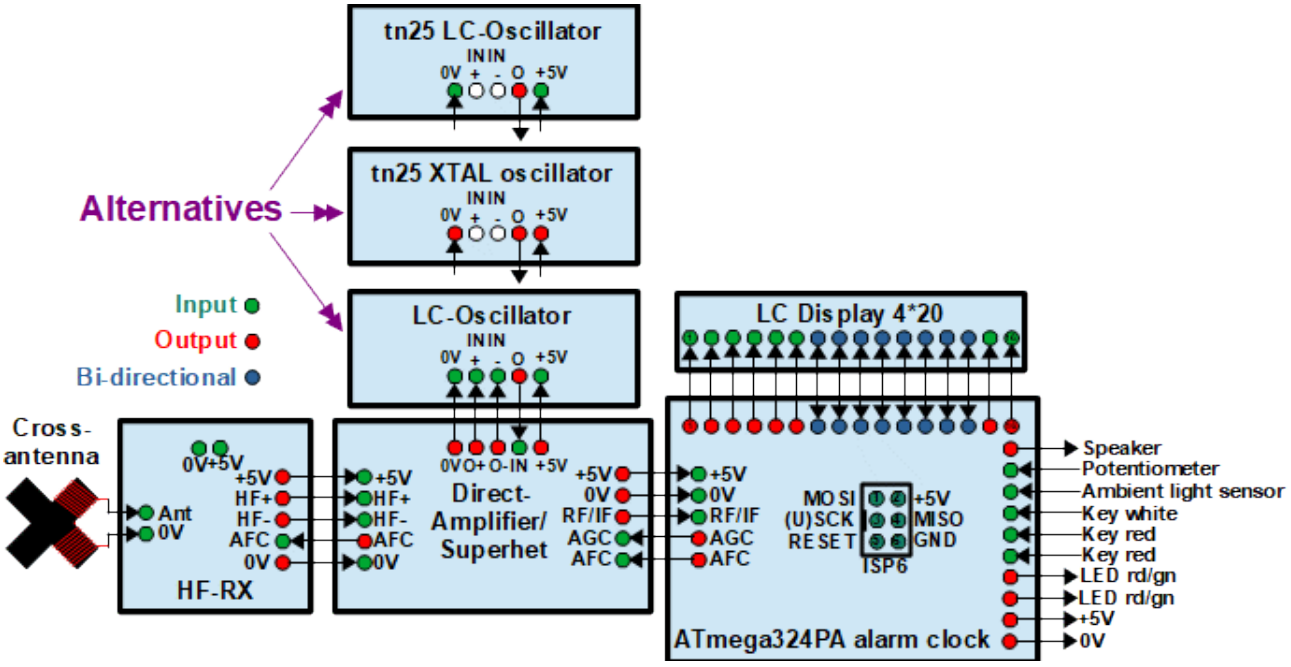
9 DCF77 Alarm clock with ATmega324

Highly experimental! Not tested yet!

With that project I will try to integrate

- the AM rectifier for 77.5 kHz RF or 32.868 kHz IF,
- the controller that controls the frequency of the input stage over the AFC line as well as the RF/IF amplifier stages via the AGC line and decodes the DCF77 signals, and
- the display that shows date and time received.

The picture shows the connections of the integrated module.



As this yields a full feature alarm clock, it should also include:

- an XTAL driven time base that advances the clock even without a DCF77 signal over longer periods (or even never),
- the opportunity to adjust date and time with three keys and a potentiometer, so that the clock works correct even completely without DCF77,
- setting an alarm time, at which melodies are played on a small speaker.

- the opportunity to measure ambient light intensity and to adjust the backlight of the LCD with that.

The ten connections to the right show the external components.

The project is an idea and not yet finished.

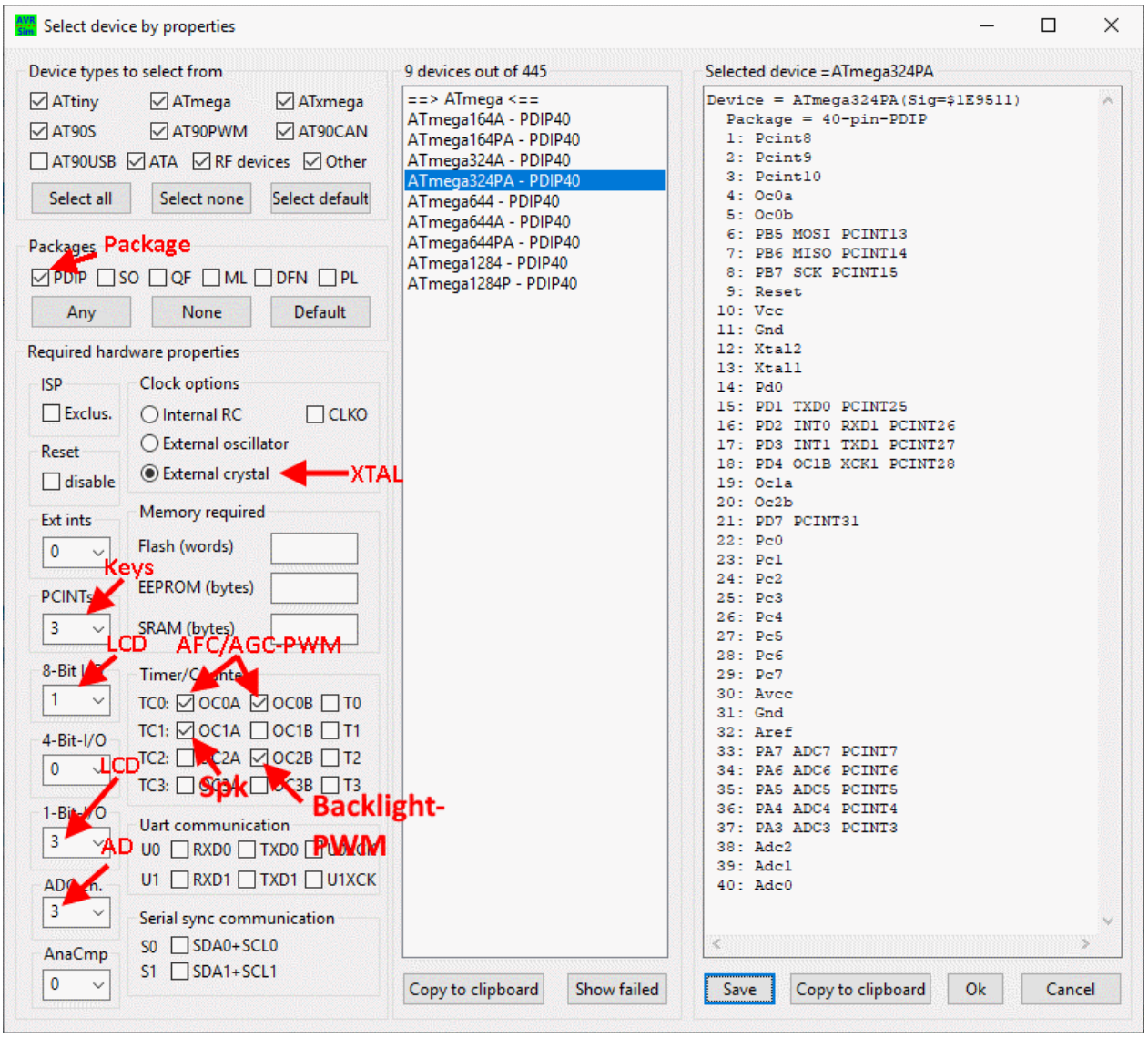
9.1 Selecting the controller

In order to perform all these tasks the controller must have

1. two PWM outputs at a high frequency for the AFC and AGC PWM,
2. a PWM with a frequency of 100 to 200 Hz for the backlight,
3. a 16-bit-timer in CTC mode with an OC output pin for generating the alarm tone and the music to be played,
4. two xtal pins to clock the controller with a crystal,
5. an 8-bit bidirectional port for the data transfer to the LCD and for reading the busy flag of the LCD as well as three single port pins for the control of the LCD,
6. three pins for the three keys, that can interrupt whenever one of the key is pressed (either INTn or PCINTn),
7. three analog converter inputs for RF/IF amplitude measurement (high speed ADC channel), ambient light sensor and a potentiometer to adjust date, time and alarm time digits.

As one of timers also has to provide the 5-ms pulse for DCF77 signal analysis and for the derived second pulse, I selected TC2 for that purpose. As this requires mixed CTC (to arrive at exact divider results) as well as PWM operation, OCR2B was selected for the LCD backlight PWM.

This is avr_sim's device selection window with all necessary hardware features. Not many different devices fulfill all the required properties. All of them end with a 4. A price comparison with my preferred electronics dealer showed the ATmega324PA is my preferred selection.



9.2 The hardware

This is the complete hardware schematic for the alarm clock.

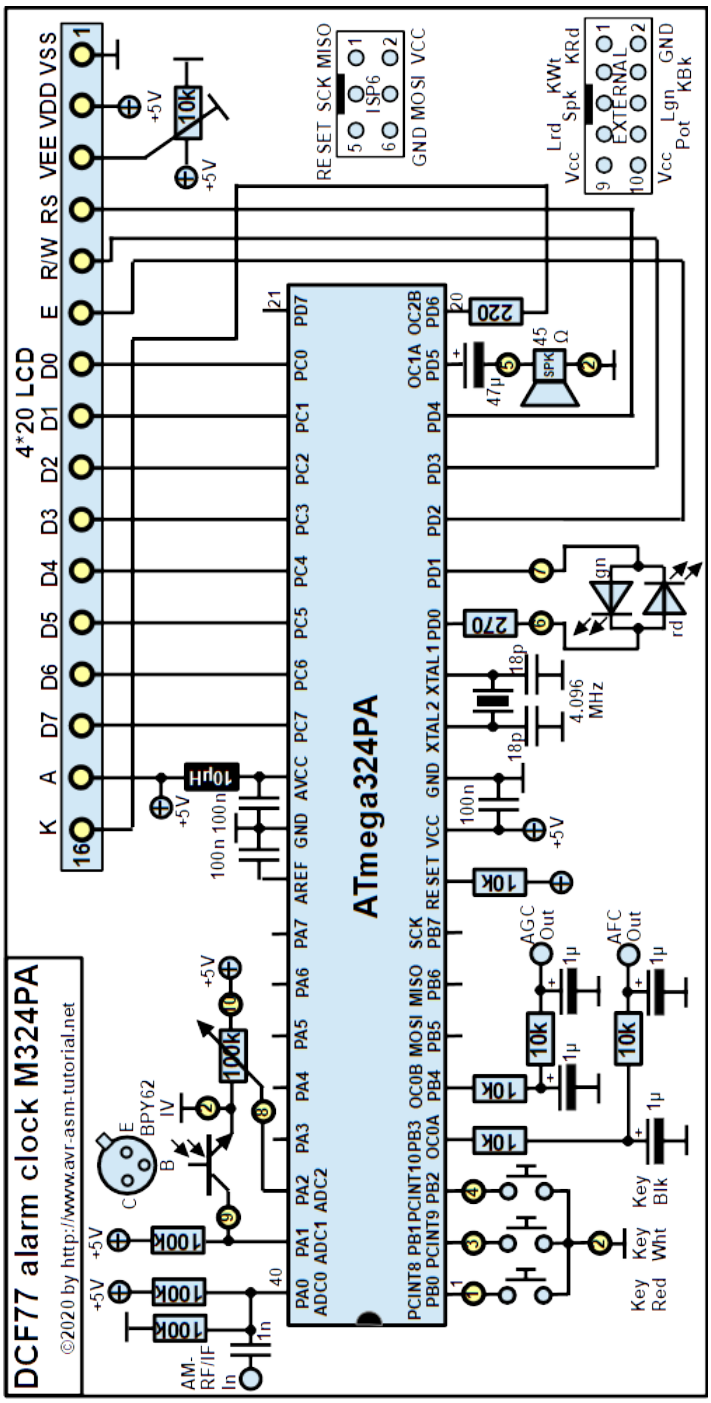
The three ADC channels ADC0, ADC1 and ADC2 are connected to the following external devices:

- the RF or IF signal of the DCF77 receiver, to be measured for its amplitude internally, as ADC0 input channel,
- ADC1 has attached the ambient light sensor transistor, to be mounted to the outside of the clock's box,

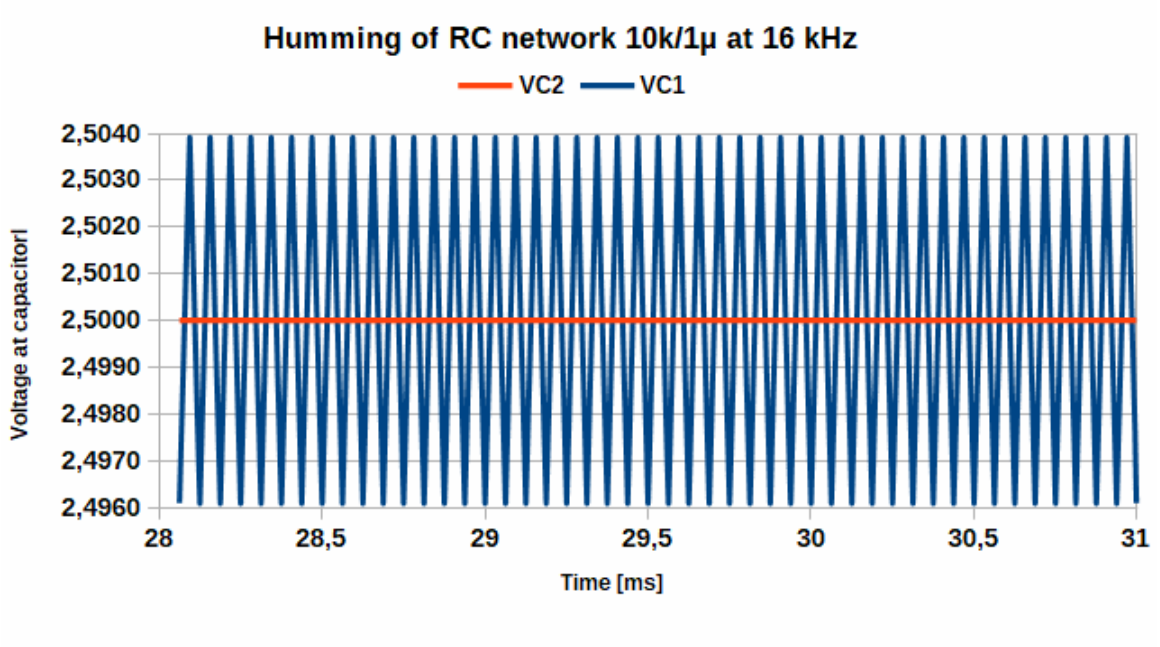
- on ADC2, the potentiometer is attached that allows to select numbers between 0 and 59 (for seconds and minutes of the time and the alarm time) or between 0 and 23 (for hours) as well as all other digits to be adjusted by the user).

Usually the ADC measures the RF/IF amplitude in a very fast manner and is in free running mode with interrupts enabled. From time to time (the ambient light sensor every 1.28 seconds, the potentiometer every 250 ms) the free running mode and the interrupt generation is stopped and the additional ADC channel is measured in polling mode.

The device is clocked with a 4.096 MHz crystal on the XTAL1 and XTAL2 pins, so that the operation of the clock over weeks and months without synchronization with DCF77 is exact enough for an alarm clock and the clock does not require re-adjustment.



The AFC- and AGC-signals are produced by OC0A and OC0B via a two-stage RC network. The PWM signal works at 16 kHz, while the RC's half frequency is by 1000-fold lower at 14.5 Hz, so that the RC networks filters the signal good enough. While the humming on the first capacitor is at 7.82 V_{pp}, the second capacitor is at 10 μV_{pp} humming, as the analysis in the sheet "rc_hum" in the LibreOffice calc file here shows.



The three keys are connected to PB0, PB1 and PB2. Those have their pull-up resistors set by software and throw PCINT interrupts to be served. After a key has been pressed, further PCINT interrupts are blocked for a dead-time of 100 milli-seconds, so that no spurious signals can confuse the software.

All external components are plugged in via a 10-pin box connector, for which the pinning is given in the lower right.

To program the ATmega324PA within the system, a standard ISP6 connector has been added. This is not necessary in the final device, but eases program changes.

9.3 Mounting the alarm clock

TBD

9.4 Software for the alarm clock

9.4.1 Download of the complete final software

TBD

9.4.2 Software for hardware testing

TBD

9.4.2.1 Testing the crystal clock and the ISP interface

TBD

9.4.2.2 Installation and testing of the LCD

TBD

9.4.2.3 Testing the LEDs

TBD

9.4.2.4 Testing the AFC and AGC signal generation

TBD

9.4.2.5 Testing the keys

TBD

9.4.2.6 Testing the speaker

TBD

9.4.2.7 Testing the RF/IF rectifier

TBD

9.4.2.8 Testing the ambient light sensor

TBD

9.4.2.9 Testing the potentiometer

TBD

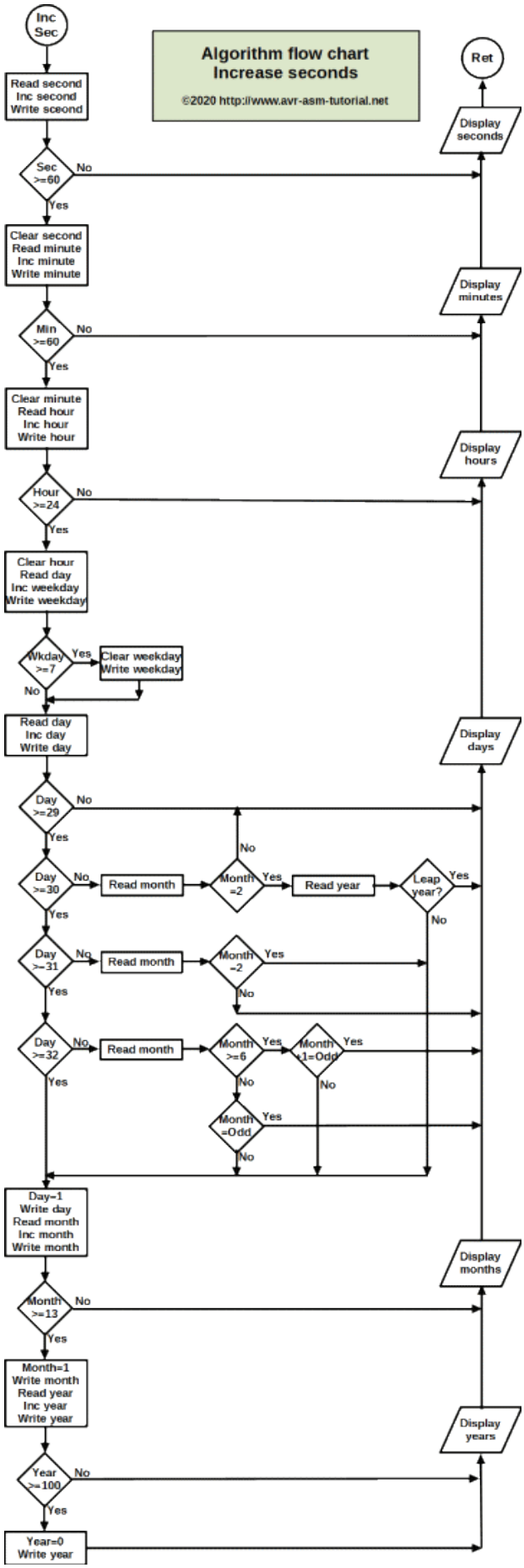
9.4.3 Software for the alarm clock

9.4.3.1 Date and time

The time that is independent from DCF77 synchronization is derived from the system clock of 4.092 MHz by TC2. TC2 is prescaled by 256 and the CTC mode in compare A divides the clock signal by 80. That delivers a signal of f = 4.096 MHz / 256 / 80 = 200 Hz. The register rSecDiv counts from 200 down to zero. If that reaches zero, the time and date has to be advanced by one second.

Time and date is completely located at the beginning of the SRAM. Time and dates are stored in binary form, each component in one byte, in a total of seven bytes. These bytes are overwritten when DCF77 received a correct and complete time/date set.

This is the flow that increases the current time and date by one second. The algorithm to the left does the increase to the date and time components, the algorithm to the right



updates all changed digits on the LCD. All software flow drawings are available in the LibreOffice draw file here.

Most of the increases are straight forward and simple. Only the calculation of the days of the current month is a little bit more complex, thanks to a pope around the year 1500.

Realization uses the pointer register pair X (R27:R26) that is increased each time the next time/date component is reached.

The display of the components on the LCD starts with setting the cursor to the line and column where the component is located. All components, excluding the day abbreviation, are displayed by calling the routine LcdDec2, that is part of the include routine for LCD operation lcd.inc.

9.4.3.2 DCF77 analysis

TBD

9.4.3.3 LCD operation

TBD

9.4.3.4 Adjusting date, time and alarm time with the keys

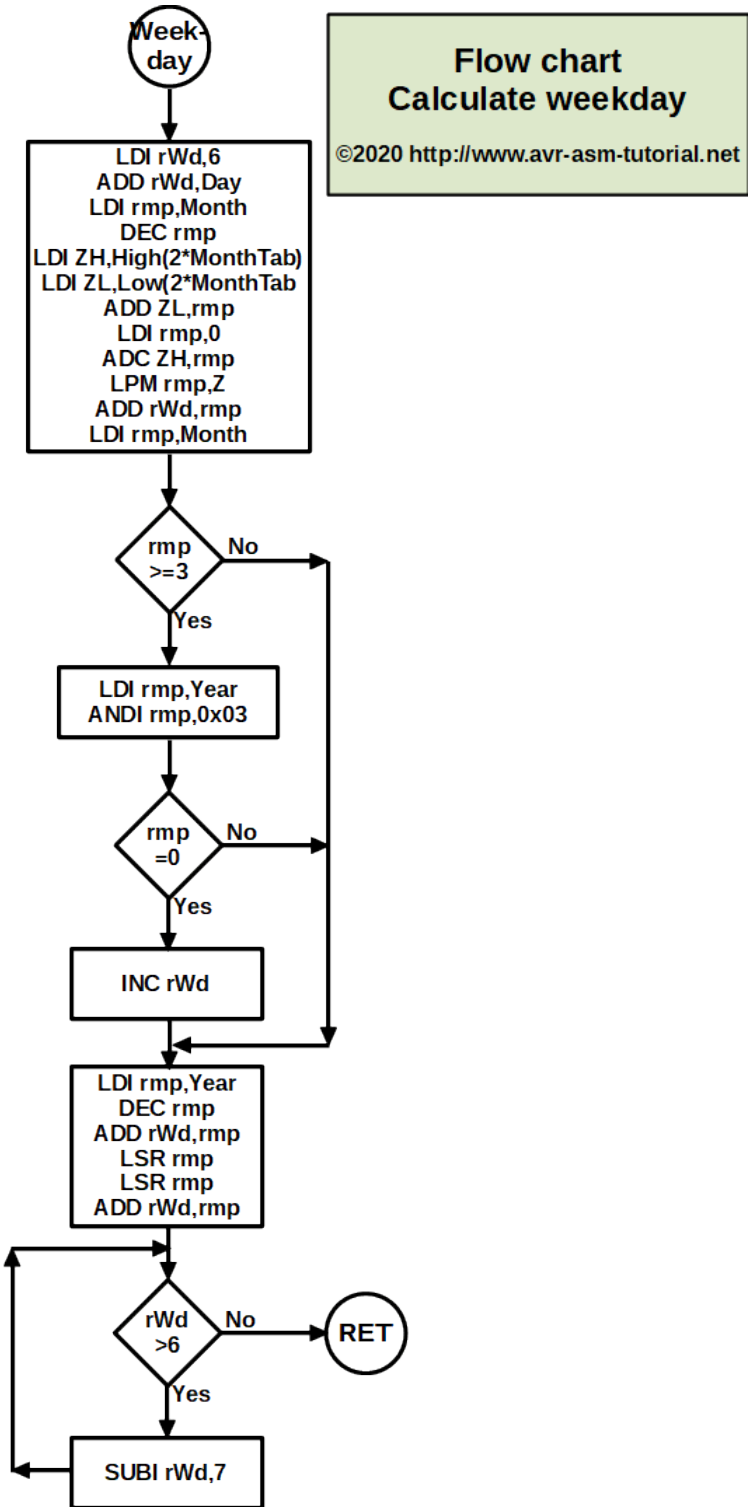
TBD

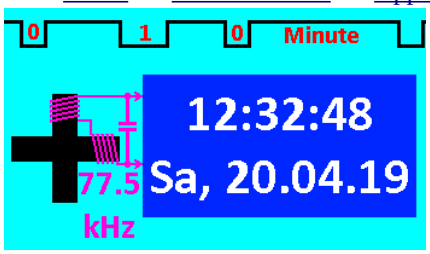
If the user adjusts the date, we can ease that by calculating the weekday from the adjusted date, so that the weekday is automatically correct. To the right you see the flow scheme of that calculation.

It is rather simple and based on the weekday on 01.01.2000, which was a Saturday. For each day and for each year since then (difference minus one) we add one to the result. The difference caused by the month are a little more complicated. It is held in a table in the flash memory. The difference to be added for each month (minus one) is read from that table. As the table is based on a non-leap-year with 365 days, we have to add one for a leap year and for the months above February. Finally we have to add the number of leap years since 2000 to the current year (minus one), which is simply the (year minus one) divided by four.

At the end we repeatedly subtract 7 until the result is smaller than 7 (Modulo calculation).

Those who want to play around with the calculation, here is the source code in asm format.





Applications of
AVR single chip controllers
AT90S, ATtiny, ATmega and
ATxmega

DCF77 direct receiver with regulated OpAmp

This project is experimental. I don't know if it really works as planned here.

10 DCF77 AM direct receiver with gain-regulated OpAmp and ATtiny25

DCF direct receivers need a regulated gain amplifier. The reasons for that are that

1. the information in the DCF77 signal is encoded in the amplitude, so amplification must now be too high to avoid amplitude clipping,
2. too high amplification leads to self-oscillation of the amplifier. At necessary gains of 5,000 and higher this is an issue.

This concept here uses

- two operational amplifiers, that amplify the DCF77 signal, and
- uses an FET/resistor divider to regulate its gain, and uses
- an oscillation signal from an ATtiny25 to produce the negative voltage to increase the resistance between the Drain and the Source of the FET and so to reduce the gain of the OpAmp (AGC).

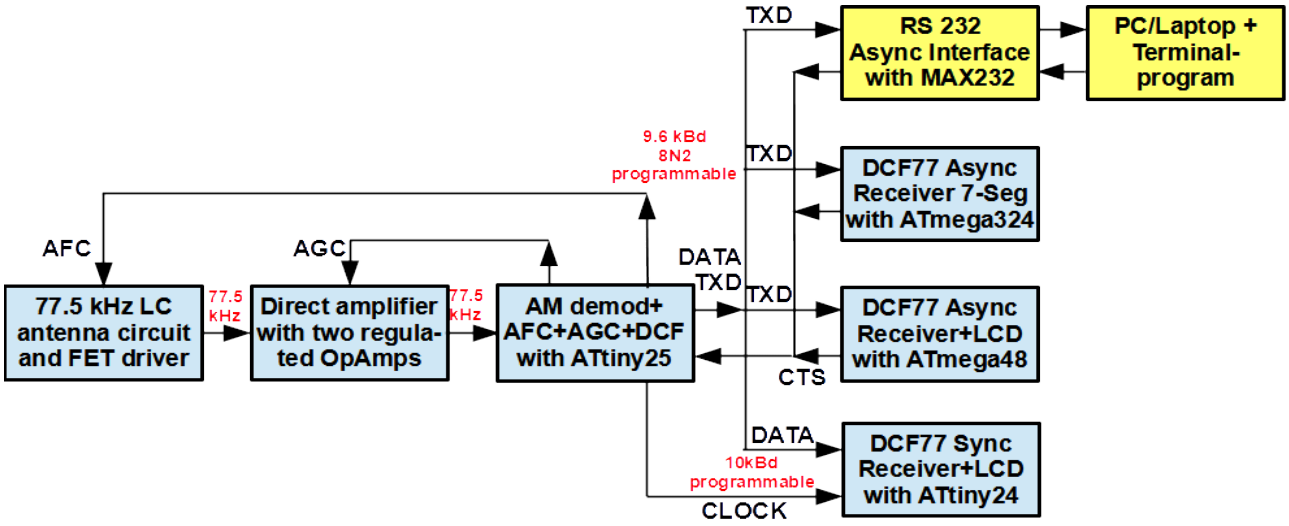
The output of the ATtiny25 on the two pins 3 and 7 can be configured as follows:

0. no signal, both pins remain low, or
1. original DCF77 signal on pin 3, inverted DCF77 signal on pin 7, no decoding or change in the signal, or
2. decoded time and date transmitted over two synchronous output pins, or
3. decoded time and date transmitted over an asynchronous output pin, if the respective Clear-To-Send pin is on.

The constant **cTxContent** determines which content is transmitted in modes 2 or 3:

0. nothing, or
1. decoded time transmitted as serial ASCII each minute, in the format "T14:59" as derived from the decoded DCF77 signal, or
2. decoded time and date as serial ASCII each minute, in the format "T14:59D12/20/20WSu", or
3. the latter, but additional receiver status and debugging info added as lines, which do not start with a T.

Conversion of the ME(S)T time to UTC can be configured within the software by setting **cUtc** to one.



This receiver here therefore is a versatile piece of equipment: you can use it as a simple DCF77 receiver that produces a High/Low signal to be decoded with another piece of equipment, you can attach a simple display with a LCD to it and you can display DCF77 date and time information via a terminal program on a PC or Laptop. What ever your needs are, the software is designed to fit these all.

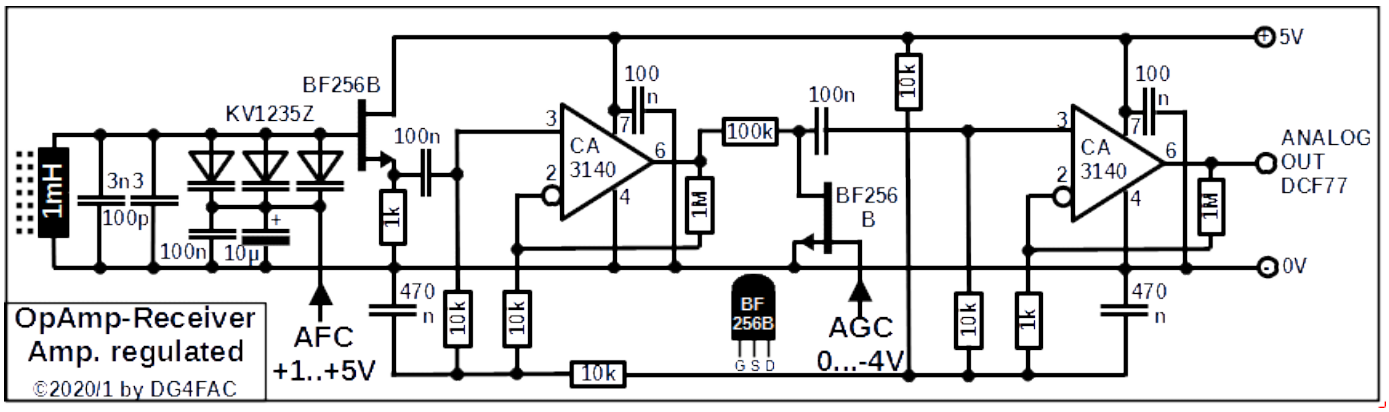
To display the results you can use the RS232 and your PC or laptop or

1. a sync receiver with an ATtiny24 and an LCD here, or
2. an async receiver with an ATmega48 and an LCD here, or
3. an async receiver with an ATmega324 and a muxed 4-digit 7-segment LED here.

All drawings are available as Libre-Office-Draw file here.

10.1 Hardware of the regulated OpAmp receiver

10.1.1 Receiver hardware schematic



This is all you need:

- The ferrite antenna is brought to resonance with a fixed C and three varactor diodes in parallel to adjust the resonance, if you do not need that part you can replace the three varactors, the whole RC network from OC0A of the ATtiny25 and the capaci-

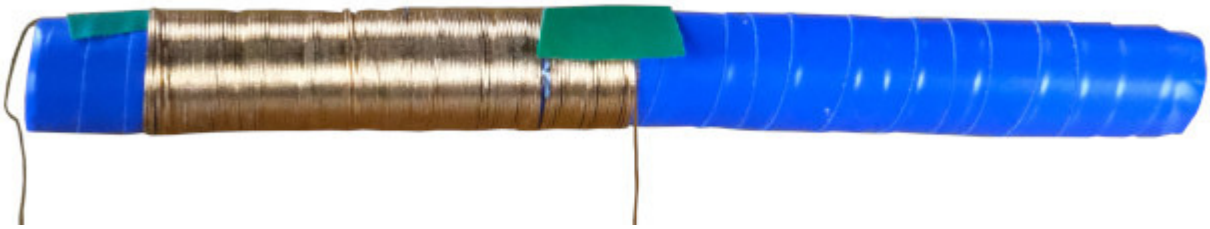
tors on the cathodes of the varactors. And you can possibly add a trim capacitor to do the resonance trimming manually.

- As I do not like the ferrite circuit to be on a different voltage than 0 Volt, and as the CA3140 does not really work as amplifier with its positive input at 0 V (only if you shift the offset voltage on Pins 1 and 5 with a 4k7 trim potentiometer widely to one side - which is not really a stable operation) I decided
    1. to divide the amplification load into two portions: one does a pre-amplification by 100, the second stage by 1,000,
    2. to operate both OpAmps at half the operating voltage (+2.5 V),

  and to couple the ferrite LC circuit with a capacitor onto the medium voltage. That meant to add a FET driver stage that provides a low resistance signal.
- In the first stage the direct amplification is done with the operational amplifier CA3140. Do not use a 741 instead, it does not work properly with a single 5V supply. Gain is fixed at 100 by the two resistors.
- Before the signal enters the second stage, a FET driven voltage divider reduces the signal gain. The FET's gate can have between 0 and -4.5V, the 100k is the upper half of the voltage divider. On start-up and with small negative voltages the signal gain is very small.
- The remaining signal behind the FET attenuator is then fed into a second OpAmp. If you don't need the gain of 1,000, because your DCF77 signal strength is already large enough, you can either skip this second OpAmp (within the near-field of DCF77) or you can reduce its gain by increasing the 1k resistor to 10k or 100k.

The analog signal then is transferred to the controller's ADC3 input pin (see below).

10.1.2 How the antenna circuit works



The 1 mH ferrite coil has been made from a 10-cm ferrite rod, which was covered over its hole length with two to three layers of adhesive tape, on which a coil with 0.25 mm copper wire was twisted. To find out how many windings would be necessary for 1 mH I stopped at 100, 130 and 160 windings and measured the inductivity. With these values I determined the specific inductivity per winding^2 **AL**. These values are shown in the table.

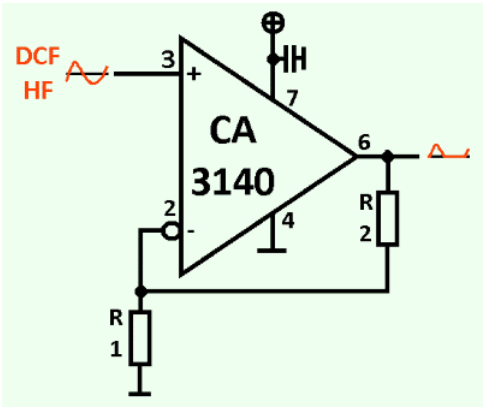
Windings	Inductivity [mH]	AL [nH per w^2]
100	0.40	39.72
130	0.75	44.56
160	1.05	40.88

The AL value, therefore, is roughly 41 nH/winding^2. For a 2 mH coil roughly 220 windings would have been necessary.

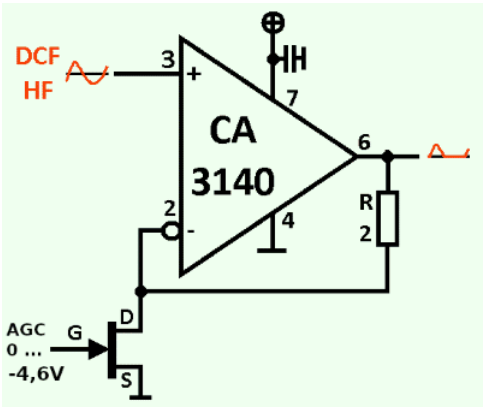
Ohm's resistance of the 160 windings is 5.6 Ω, inductive resistance at 77.5 kHz is 510 Ω. The C needed for resonance at 77.5 kHz is 4.03 nF. Preferred Styroflex capacitors of 3.3 nF are in the market, so with the three varactor diodes at 243 pF the resonance can be reached. A 3.9 Styroflex would also be fine, but the varactors would then be very small in capacity. You can even use any different capacitors besides Styroflex, because the capacity is regulated anyway so that accuracy and long-term reliability play no roll anyway.

The LC circuit's resistance, when L and C are in resonance at 77.5 kHz, is around 50 kΩ or higher. So usual transistors or PNP/NPN OpAmp entry stages are not recommendable, the OpAmp should have a FET on the input.

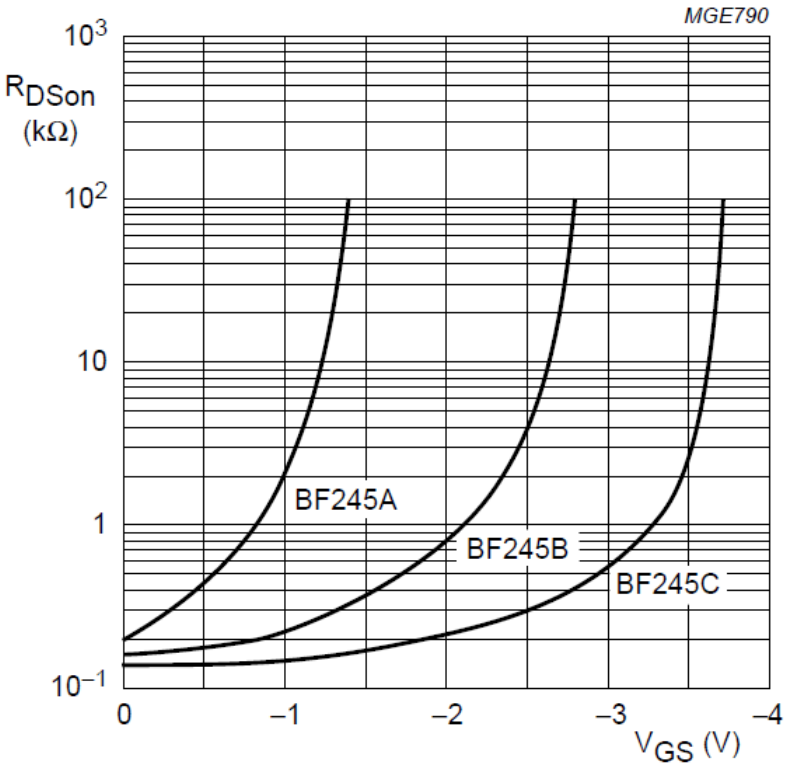
10.1.3 How the regulated OpAmp works



In the original design displayed here I tried to replace the resistor R1 in a linear amplifier stage with a FET. Its gate voltage can be varied between zero and minus 4.6 V. That varies the resistance of the Drain-to-Source pins of the FET between some 100 Ω and up to more than 1 MΩ.

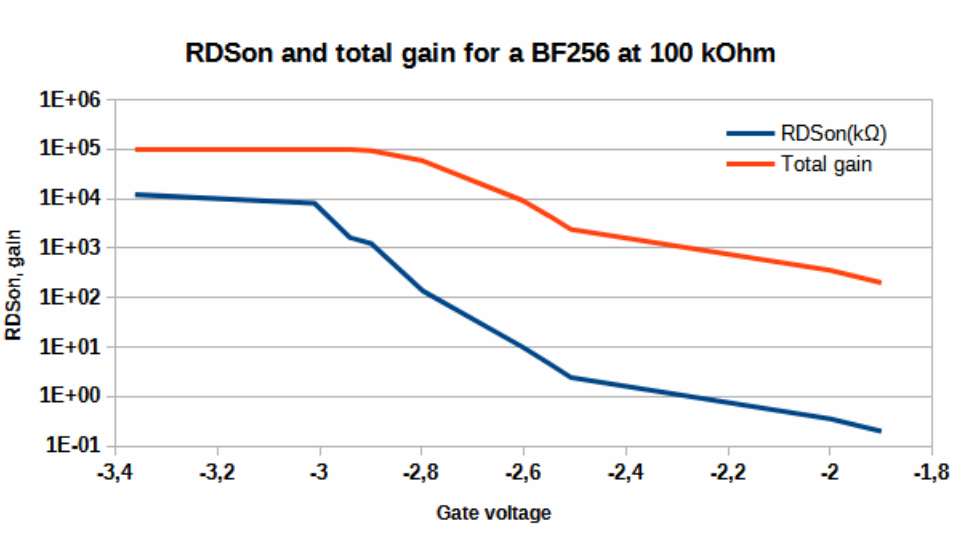


This shows the variation of RDSon of the three BF245 FETs. The curve starts at roughly 200 Ω (for a BF245A) or less and, with rising negative voltages on the gate pin, rises to more than 100 kΩ. The BF245C needs higher negative voltages than the A or B type for the same RDSon.



This type of diagram is not available for other JFETs. Usually only IDSS is listed, which is the current through the Drain-Source, with the gate at 0 V and at a certain voltage (mostly 15 V) on the drain. This is useful for calculating RDSon at 0 Volt, but yields too high and unrealistic resistances (the 15 V are unre-

alistic). Therefore I have measured a BF246B by adding a resistor on the drain and measuring the drain's voltage at different gate voltages. The diagram shows for the limited number of measurements that the current through the drain yields realistic RDSon values (in red) and that the calculated OpAmp gain is between one and 100,000.



Unfortunately, the CA3140 does not really work as an amplifier with the positive input pin on ground. Only if you un-balance the bias voltage with a 4k7 trim potentiometer on pins 1 and 5 (with the middle of the potentiometer on Ground) very widely to one side, the amplification starts. The trim range over which the CA3140 then amplifies is very narrow. If you leave this range, the CA3140 starts swinging wildly. The range where it works is depending from the gain and even reacts on the varactor diodes of the attached LC circuit. Nothing you can rely on without running into serious trouble.

I had to change the concept. After trying out different solutions, the schematic shown above has yielded a reliable amplifier. If you reduce the capacitors, e. g. down to 1 nF too much noise and instability is the consequence. Which is to be expected with a high amplification rate of 100,000, because already 10 µV noise already make up 1 V if amplified by 100,000. The positive side is that the FET-decoupled LC circuit, adjusted to DCF77, hasn't reacted on my energy saving lamp, that produces lots of noise near 77.5 kHz (which causes usual commercial receivers to malfunction).

A gain of 100,000 means that 10 µV HF are amplified to 1 V on the output. This amplification is high enough in the near-field and in some distance to DCF77.

As you can see from the diagram of the RDSon of the BF256 the amplification is steeply rising/falling in the middle of the curve. That means that already small change of the negative base voltage of the FET lead to the signal being too small or too large. By adjusting the gain of the two OpAmp stages, you can change the regulating voltage to get to the less steep sides of the diagram. Just change the resistors from the negative input to ground, e. g. to 1 or 10 or 100 kΩ. As both stages do not invert the input signal, even an amplification of 1,000 does not lead to back-feed and wild swinging.

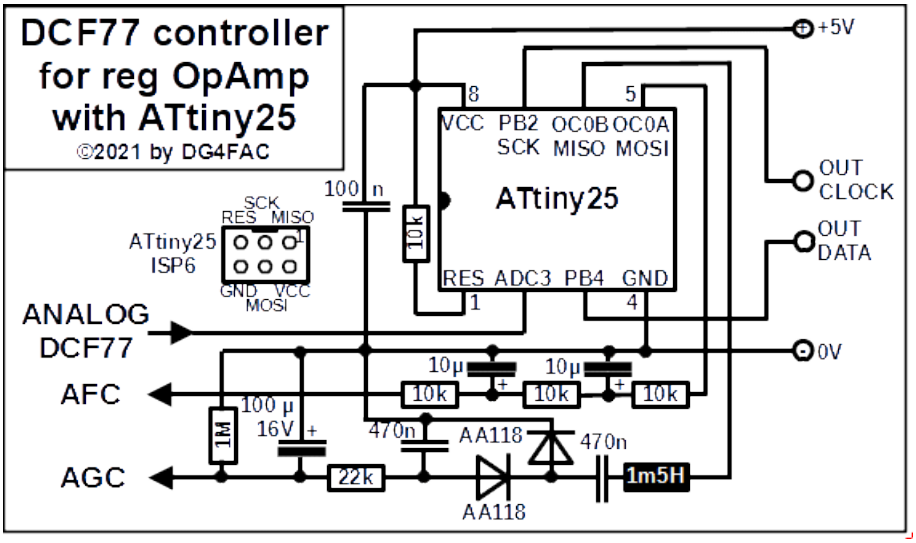
10.1.4 Serial receivers for display

The serial signals that are produced by the ATtiny25 are invisible. Only if you add additional hardware, you can display that. Those who don't want to use a PC or Laptop and their terminal program have the following opportunities to see what is going on:

1. a receiver for sync serial signals with an ATtiny24 and an attached LCD module here, or
2. a receiver for async serial signals with an ATmega48 and an LCD here, or
3. a receiver for async serial signals with an ATmega324 and a very large 7-segment LED display for displaying the received time here.

Anyone can find its appropriate solution from that, the tn25 delivers all you need to have.

10.2 Hardware of the ATtiny25 controller



Control and regulation as well as analyzing the DCF77 signals is the task of an 8-pin ATtiny25. You can also use an ATtiny45 or 85, if you want to waste unused flash and EEPROM memory space.

The following functions are implemented:

1. measuring the amplitude of the DCF77 signal on pin ADC3 (pin 2),
2. generating the AFC control voltage on the OC0B output pin (pin 6), and
3. generating and outputting the desired results via pin 3 (either the original DCF77 amplitude state, the data for sync serial transmission or the TXD signal for async transmission) and pin 7 (either the inverted DCF signal, the Clock signal for serial sync or the CTS input for async serial signals).

These hardware functions are described nearer in the following chapters.

10.2.1 Measuring the DCF77 signal

The ATtiny25 measures continuously the maximum amplitude value of the OpAmp's output on his ADC3 input. As the signal swings around the medium voltage of 2.5 V, these values are rectified mathematically, so that even unsymmetrical signals can be measured correct.

The software then determines two values from 256 of such measurements:

1. the maximum amplitude value (above or below the average), and
2. the average of the signal.

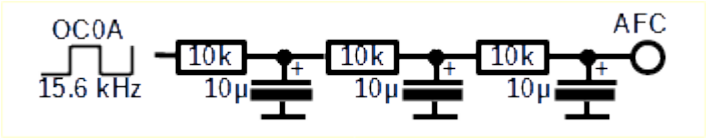
The second value should be around +2.5V (8-bit-ADC value = 128) and serves only for the mathematical rectification of the signal.

The first value controls the AGC: if this value is higher than the pre-selected value of 2.0V (positive = 4.5V, negative = 0.5V, ADC between 230 and 25), the AGC voltage generator

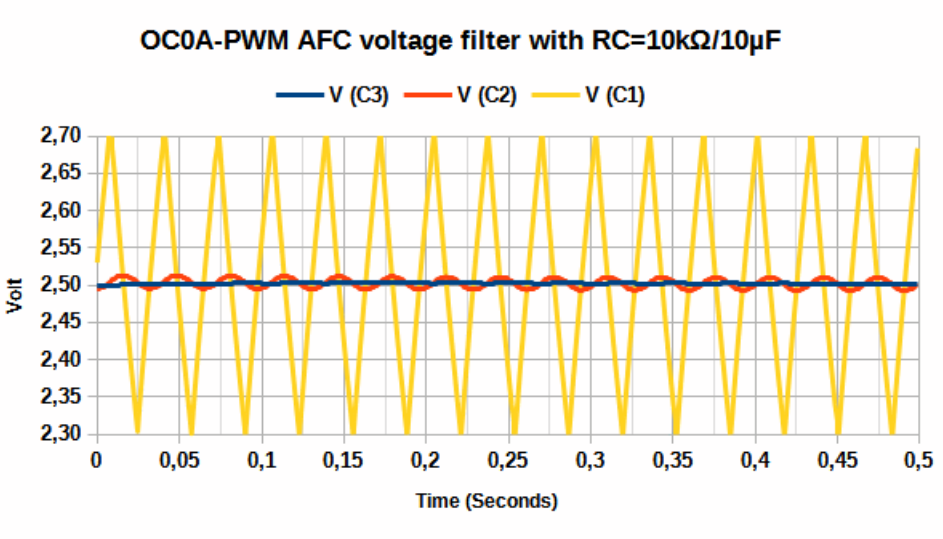
is switched off. This decreases slowly the Gate voltage of the FET and the amplitude gets smaller.

The further treatment of the measured maximum values is described in the software section.

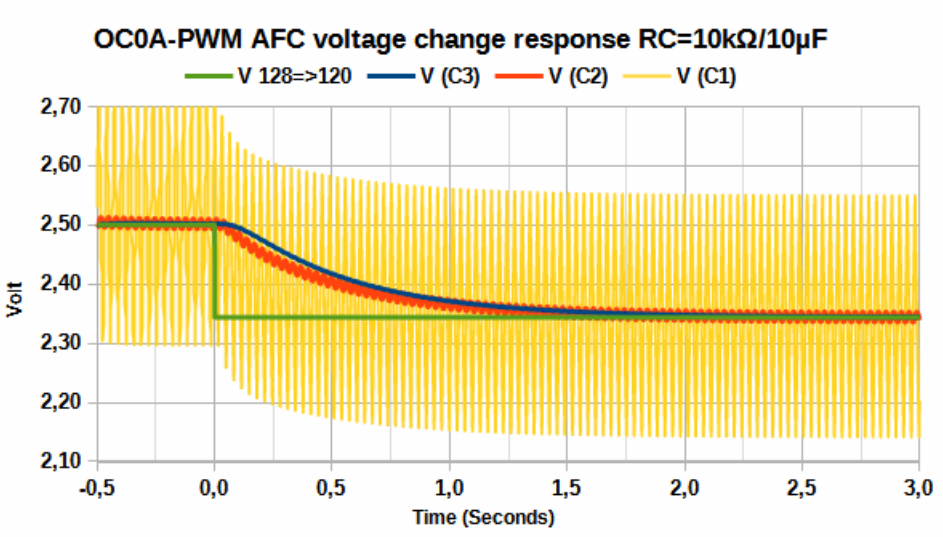
10.2.2 Generating and filtering the AFC voltage



The OC0A-PWM generates a pulse-width-modulated 8-bit-rectangle, which is filtered with a three-stage RC network. The output controls the three varicaps on the LC circuit.

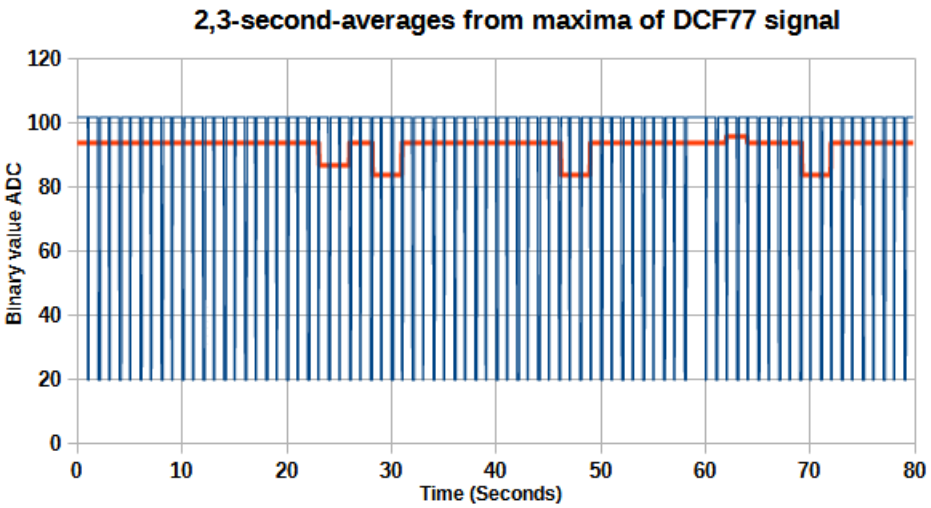


The three stages are necessary to remove PWM noise from that signal. In the first stage there are still 420 mV ripple (see calculation sheet **opampreg_OC0A** in the Libre-Office Calc file here). In the second stage this goes down to 19 mV, in the final stage to 3.6 mV. This ripple varies the LC input stage by roughly 7.2 Hz, small enough for operating the LC stage and below the 256-stage resolution of the PWM (40 Hz per digit).



The very good filter properties of the RC have an adverse effect: it delays adoption to changed PWM values. The delay when changing the PWM value by 8 lasts roughly 2 seconds, as shown in this diagram. Changes by 8 occur at start-up when the AFC is scanned to find the first approach to the optimum value. That means that the software has to wait for two seconds until the changed voltage has settled.

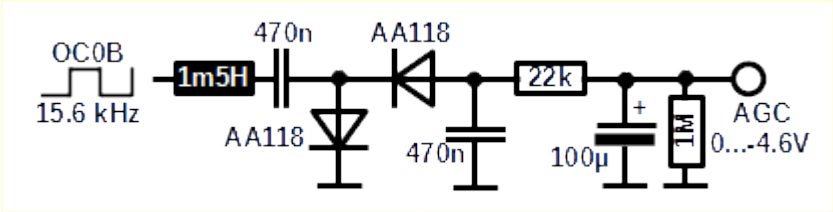
Scanning is performed at the beginning. The OCR0A value is set to 255 and is reduced by 8 in 28 stages. After the delay the maximum and its average is measured for 256 times (=65,536 measurements, 2.3 seconds long). The difference between the maximum and its average helps to identify whether the input signal at this frequency is a steady-amplitude transmitter or is DCF77. This because the low-amplitude phases of DCF77 reduce the average, and in 2.3 seconds at least one amplitude reduction takes place. This is simulated in the following diagram over 80 seconds long. The OCR0A value at the maximum difference between maximum and its average is finally selected as start value for the AFC and written to the OCR0A portregister.



As each scan value is settling over 2.3 seconds and then measured over 2.3 seconds, the whole scan phase lasts 129 seconds or two minutes long.

The further adjustment of the AFC is performed during normal operation. Here, the OCR0A values are varied by +/-1. More about the scan phase and AFC adjustment during operation can be found in the software section.

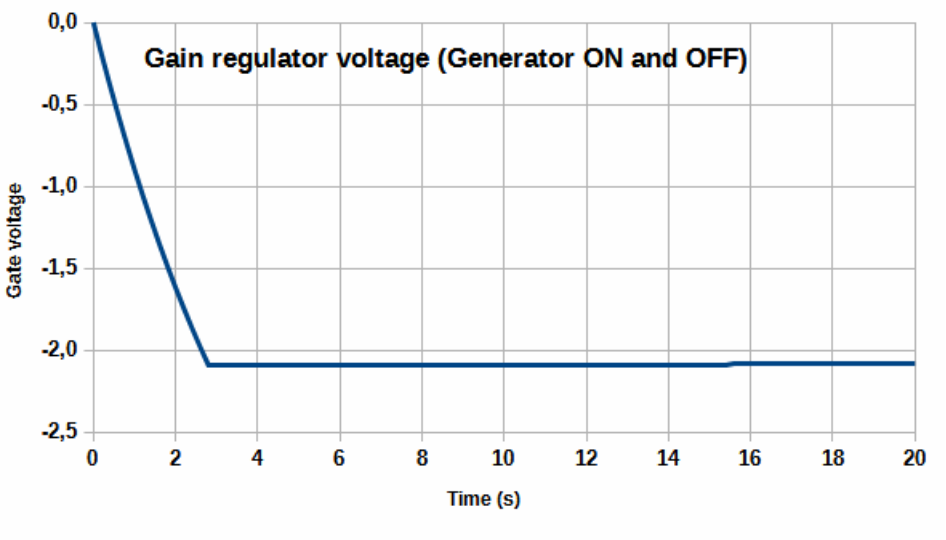
10.2.3 Generation and filtering of the AGC voltage



The generation of the negative control voltage for the gate of the FET (AGC) is done with a coil, two capacitors of 470 nF and two Germanium-Diodes. If OC0B produces a rectangle, this generator delivers -4.6 V, if permanently on. This is enough for all types of FET (A to C types). Driving the OCR0B-PWM output with other OC0B compare values than 0x7F has nearly no effect, the control voltage has to be regulated by switching the output pin OC0B off and on, the long time constant of 10 MΩ * 470 µF is long enough to not change control voltages too fast, e. g. when DCF77 is reducing its amplitude to transmit a zero (100 ms) or a one (200 ms), and to smooth the signal during pump phases (when full, the AC on that 470 µF capacitor is at less than 1 mV).

If the generator is off, the capacitor unloads via the resistor. This increases the negative gate voltage to more positive values, RDSon of the FET gets smaller, the divider between the OpAmp stages lets less amplitude through to the second OpAmp and so reduces the overall gain.

At start-up, the capacitor is unloaded and RDSon has its lowest value. So, on start-up no input signal in the second stage is present. Only after the OC0B-generator has produced enough load, the second stage starts amplifying. Loading ends when the maximum allowed amplitude has been reached.



During normal operation, if the long-term amplitude maximum falls below a minimum value, the generator is again switched on. As the 2.3-second average always includes one or two amplitude drops, the generator does not react on short-term drops.

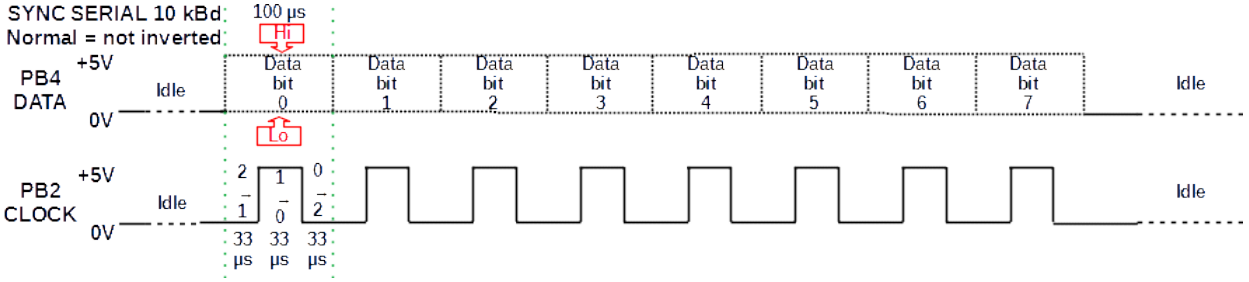
On more details on the algorithm that controls the AGC voltage see the software section below.

10.2.4 Output of results

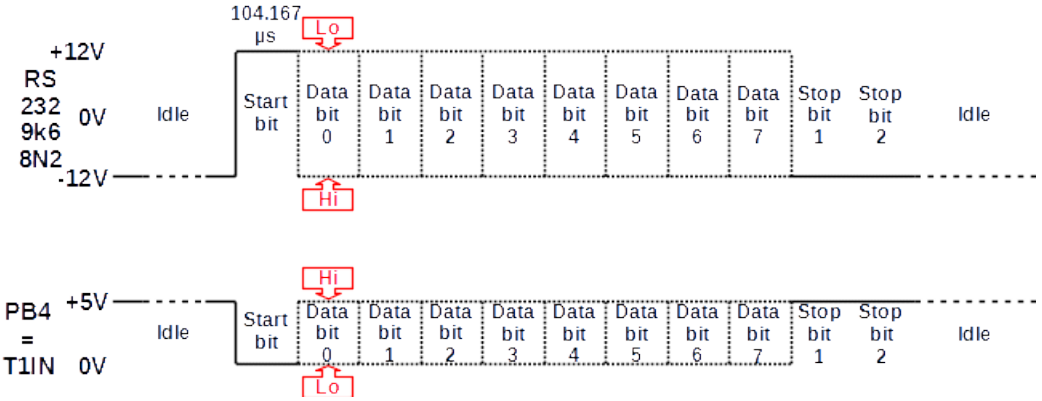
The two pins PB2 and PB4 produce the output signal, depending from the selections in the source code:

0. If the constant **cTxMode** is zero, both output pins are permanently low.
1. If the simplest method 1 is selected, pin 3 (Data) follows the DCF77's signal strength: normally high, when a zero or one is transmitted it goes low. Note that for recognizing a low at least three maximum values below the long-term maximum average is necessary, so that a delay of 30 ms occurs. For those who need an inverted signal: just use the output pin 7 (Clock/TXD) or set constant **cRevert** to one.
2. If serial synchronous output is selected with **bTxMode** = 2, data and messages are send over the two pins. The bits of the character stream are placed on the DATA pin 3, starting with bit 0, the CLOCK pin 7 is activated for one third of the time per bit (baud rate) and after another third the next bit is placed on the DATA pin. All content is send as ASCII characters. The baud rate can be adjusted by the constant **cBaud** (in Baud or bit per second).

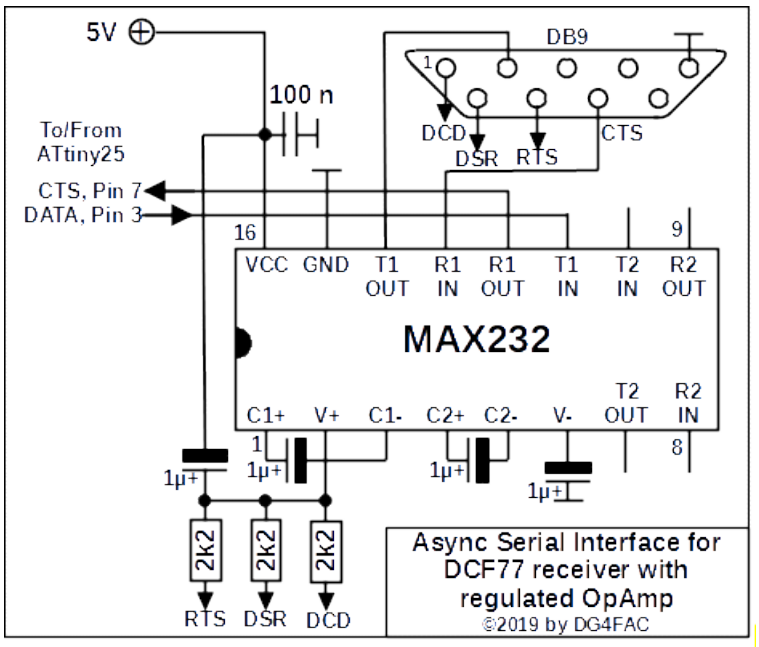
   The scheme shows transmission of a 0xAA byte with 10 kBd. The three phases allow the receiver to prepare and perform its actions. Each eight bits form one character, all bits are send without additional pauses in between. The end of a transmitted line is finalized with a carriage return and a line feed character.



3. In mode 3 all results and messages are send in asynchronous mode, with a start bit and two stop bits (1N8), over the DATA/TXD pin:



This shows such a transmission over a RS232 line, with a baud rate of 9k6. The +/- 12V level over the RS232 line can be seen. Such a level converter looks like this.



Only the CTS signal of the RS232 is used: it has to be activated to allow transmission. Baud rates are adjusted with the **cBaud** constant, **cRevert** = 1 inverts both signals. Lines are finalized with CR and LF again.

The selection of what shall be transmitted can be adjusted in **cTxContent**:

0. Zero transmits nothing.
1. One transmits the time in the format "Thh:mm:ss", preceeded by a **T**. It follows a **M** for Mid-European-Time, a **S** for Mid-European-Summer-Time or a **U** for UTC. A CR+LF follows.
2. Two transmits the time, preceeded by **T**, then M/S/U for the time format, then a **D** for the following date. If cEN in the ATtiny25 is one, the date is send as "MM/DD/

YY", if zero as "DD.MM.YY". Then a **W** and the two characters of the weekday follow, finalized by CR+LF.

3. Three additionally sends relevant messages on the status of the controller such as the adjusted AFC-PWM value as "F = 123" and the state of the AGC generator as "G = On/Off" and so-called E numbers with DCF77 time/date conversion errors, as well as time and date in the long format like above.

10.3 The software

10.3.1 Software download

The software is still under construction. $$To be done$$

10.3.2 Software overview

The software has to do the following:

1. The DCF signal that comes in from the ADC3 input pin has to be converted to a digital value, has to be rectified (as positive distance from the averaged mean value) and from that
    - the maximum of 256 measurements of the amplitude has to be identified, and
    - the average of those 256 measurements has to be calculated.

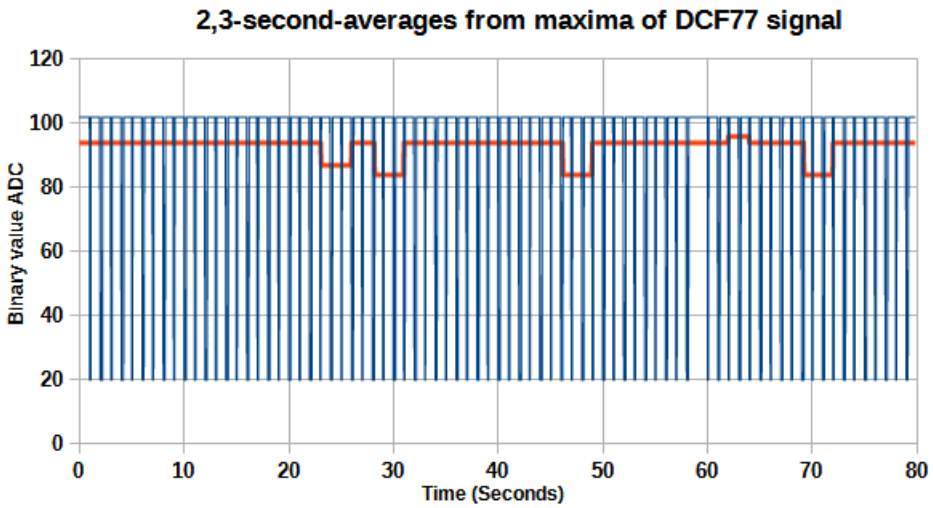
   This is all done within the interrupt service routine. To speed things up, only the 8 most significant bits are read from the ADC and averaging sums up those raw values in a 16-bit register pair. After 256 measurements (roughly 7.3 ms) the MSB of the sum is copied to the register **rAvg**. This value is only used to determine the rectifier's mid value, to be subtracted from. If subtraction of the average from the ADC raw result sets the carry flag, the value is complemented (subtracted from zero). This value is then compared with the previous maximum in **rMaxM** and, if larger or equal, replaces this maximum. After 256 measurements the value is copied to **rMax** and the **bAdc** flag is set. Further processing is outside the interrupt service routine.
2. Outside the ISR it is checked whether the AGC generator is currently on (**bGain** = 0). If this is the case and if the **rMax** value is equal or above the selected value in **cMaxLevel** (2 $V_{rectified}$ = 4 Vpp) the AGC generator is switched off, the flag **bGain** is set and the long-term averaging of the maximum is restarted.
3. If not currently charging the AGC, the long-term maximum is calculated by summing up 256 maximum values. The MSB of this sum is written to **rLTMax**. If this value is smaller than the minimum level in **cMinLevel** (0.5 V) the AGC generator is switched on again and the flag **bGain** is cleared.
4. If not, the DCF77 signal recognition is performed as follows:
    - If the previous amplitude was high (flag **bHigh** = 1) and if the short-term maximum is smaller than the long-term average maximum, a cycle counter is decreased (from three down to to zero). If that is the case, the amplitude has changed to low. In that case **bHigh** is cleared and the cycle counter is checked whether the high phase counter is within the bounds of a minute change. If so, a minute change is performed.

- If the previous amplitude was low (flag **bHigh** = 0), the same mechanism takes place. If this phase ends, with the three-counter reaching zero, it is checked whether the previos low-phase was in the range of a zero or a one.

5. If output options are enabled, the respective outputs have to be written to the output buffer and buffer transmission is started. These parts are only assembled if so enabled.

To adjust the AFC voltage,

1. a scan at the beginning finds the raw area, where DCF77 transmits, and
2. during the decoding running, the PWR is running with one digit lower and one digit higher values, which allows a fine identification.



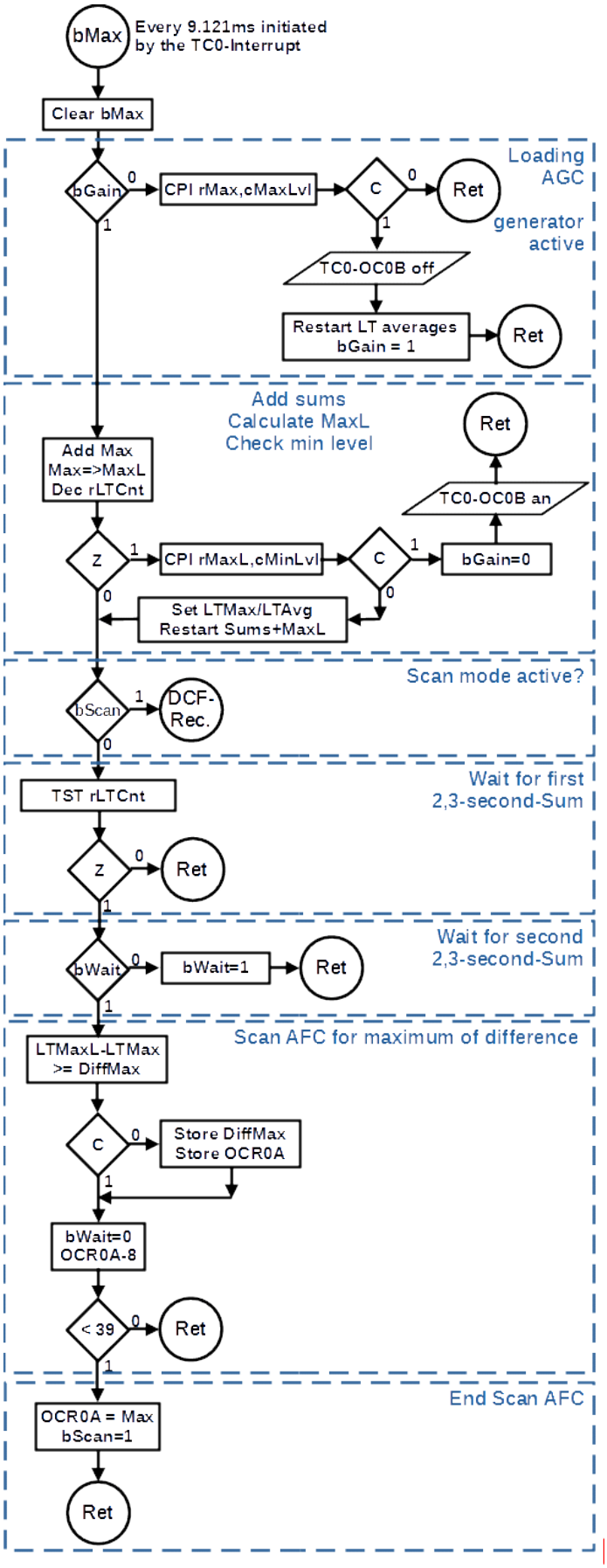
The scan works as follows. It starts with the highest possible voltage that TC0 can produce (OCR0A at 255 respective +5V, lowest capacity of the varactors, highest frequency of the LC)- After waiting for 2.3 seconds to accommodate the RC network to the new value, the 256 maxima are measured, for which the maximum and the average are calculated. If the difference between the maximum and its average is higher, this value is saved. Then the value of the PWM is decreased and the same procedure is repeated. If the PWM value reaches its minimum (by default 39), the scan ends and the OCR0A value at the detected maximum is written to OCR0A.

See below for a flow diagram of this scan.

When running normal, the PWM value is varied with -/+ 1 digit and the same difference is used to re-adjust the AFC to one of the values -1, 0 or +1.

10.3.3 The AD conversion of the input signal

The AD converter runs in free auto-start mode, which means: it restarts the AD whenever the previous result has been read. As the ADC has to detect the maximum of the 77.5 kHz DCF77 wave, it runs nearly as fast as possible (2 MHz controller clock, AD prescaler = 4, AD frequency roughly 35 kHz). So it measures approximately one value per two sine waves on the input. An analysis shows that this is sufficient and does not produce too much erronous values. Result fetch, maximum detection and average calculation is performed within the ISR of the ADC.



The timing is shown in the calculation sheet **DCF77clocking** of the Libre-Office file here, with the parameters in the sheet **opampreg_ADC**. The sheet **opampreg_maxdetection** simulates the maximum detection over 256 measurements. As the ADC is restarted only after the previous result has been read, the effective number of conversion cycles is 14.25. Sampling 256 measurements lasts roughly 7,3 ms.

To avoid lengthy division routines, the average calculation over 256 measurements is held as simple as possible. Because only 8-bit values are relevant here, the **ADLAR** bit of the ADC is set and only the high byte of the result is read.

The detection of the maximum and the calculation of the average are done within the interrupt service routine. This needs between 27 and 32 clock cycles for that. If the controller would be at its default 1 MHz clock and the AD prescaler would be two, one measurement would last 28 controller clock cycles. That means that there is no time left to do something else, e. g. for transmitting or calculating long-term averages, but to collect AD values and restart the ADC. Therefore the default clock of the ATtiny25 of 1 MHz is in-

creased to 2 MHz, for which the clock prescaler **CLKPR** is set to four at the beginning. At 2 MHz clock the controller provides enough spare time for the 27 to 32 clock cycles of the ISR as well as for other purposes such as transmitting.

10.3.4 OC0A and OC0B adjustment: The AFC scan at start-up

This flow diagram shows the scan phase, that starts after beginning and that is performed outside the ISR. The ISR of TC0 initiates this every 10 ms by setting its flag bit, if its down-counter reaches zero.

At start-up the 470 µF capacitor is not loaded and starts loading. In this phase it is only checked if the maximum is below the threshold level. If this is the case, nothing else happens. If the threshold has been reached, the **bGain** bit is set, the generator is switched off, the averages are restarted and normal processing can begin.

By summing up 256 short-term measurements of the maximum an average of the maximum over 2.3 seconds is calculated. If those 256 measurements are available, it is checked whether the maximum are below the selected min level. If that is the case, the generator is restarted again and **bGain** is cleared.

If the 256 values have enough amplitude, the MSB of the maximum and the MSB of the sum is stored in the long-term registers for further use.

At start-up the raw adjustment of the frequency has to be performed. During this time, the **bScan** bit is clear. Scanning starts with 255 in the OCR0A port.

This phase is necessary because there could be other transmitters within the LC reception band:

1. At first: the tenth harmonic of the TC0-PWM, that produces the AFC voltage and that generates the AGC voltage from time-to-time is at 78.12 kHz and not very far from DCF77. So better not waste space and place the antenna circuit as far away from the ATtiny25 and the PWM components as possible. Anyway, this does not produce an amplitude-modulated signal but a steady noise.
2. Here at my location south of Frankfurt a strong signal on 80 kHz is seen, for which I do not know where it is coming from.
3. My energy saving lamp transmits constantly near 70 kHz.

It is insufficient to measure only the maximum of the received waves, because you would be stuck to these constant-carrier-signals instead of DCF77. So it is the second-pulses that are the relevant detection criteria for DCF signals. Only the signals where the maximum is by 10 or 20% larger than the average maximum promise to the DCF77 carrier.

As long as the scan is active a **bWait** bit = 0 lets the scan continue for another 2.3 seconds. Only if this bit has become one, the value is read and compared with the previous ones. Each of the 28 measurements in the scan phase is evaluated for this difference, the maximum difference and its associated OCR0A value are stored.

In each scan step the OCR0A value is decreased by 8 until it becomes smaller than 39 (AFC = 0.76 V). If that is reached, the scan ends and the optimal OCR0A value is written to OCR0A.

10.3.5 The AFC in normal operation

After absolving the raw scan, the following algorithm adjusts the AFC voltage.

1. The long-term difference between the long-term maximum minus the average maximum is calculated and stored in SRAM (measurement 1).
2. The current OCR0A value is decreased by one and, after a wait period, the long-term difference is measured and registered (measurement 2).
3. Then the OCR0A value is increased by two and the same procedure is performed (measurement 3).

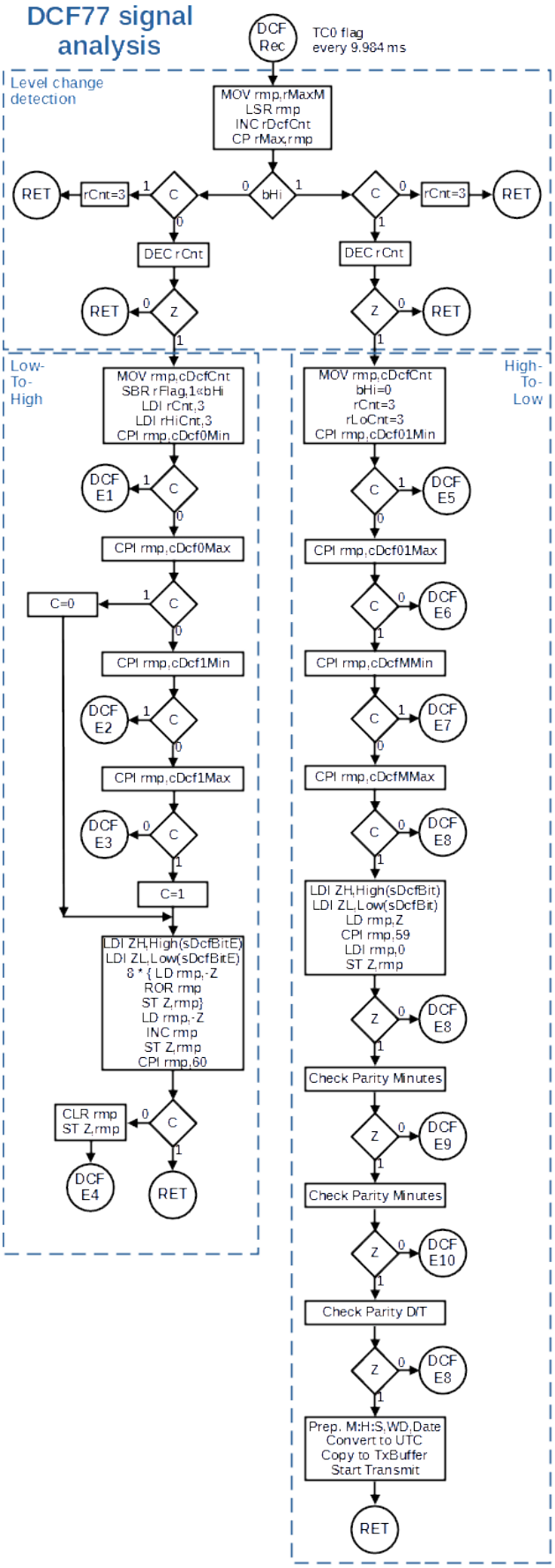
The maximum of the three stored values is selected as next approach.

10.4 Analysis of the DCF77 signals

This is the flow diagram of the DCF77 signal analysis. Displayed here is solely the recognition of zeros, ones and the minute changes.

It starts with halving the long-term maximum, this value is used to determine zeros and ones and high-amplitude phases. The duration counter **rDcfCnt** is increased by one.

Further processing depends from whether we are in a high phase of the amplitude (**bHi** = 1) or not. If in a low amplitude phase (left part of the diagram), it is checked whether the short-term maximum is still lower than halve the long-term maximum. If that is the case, the counter **rCnt** is restarted at three.

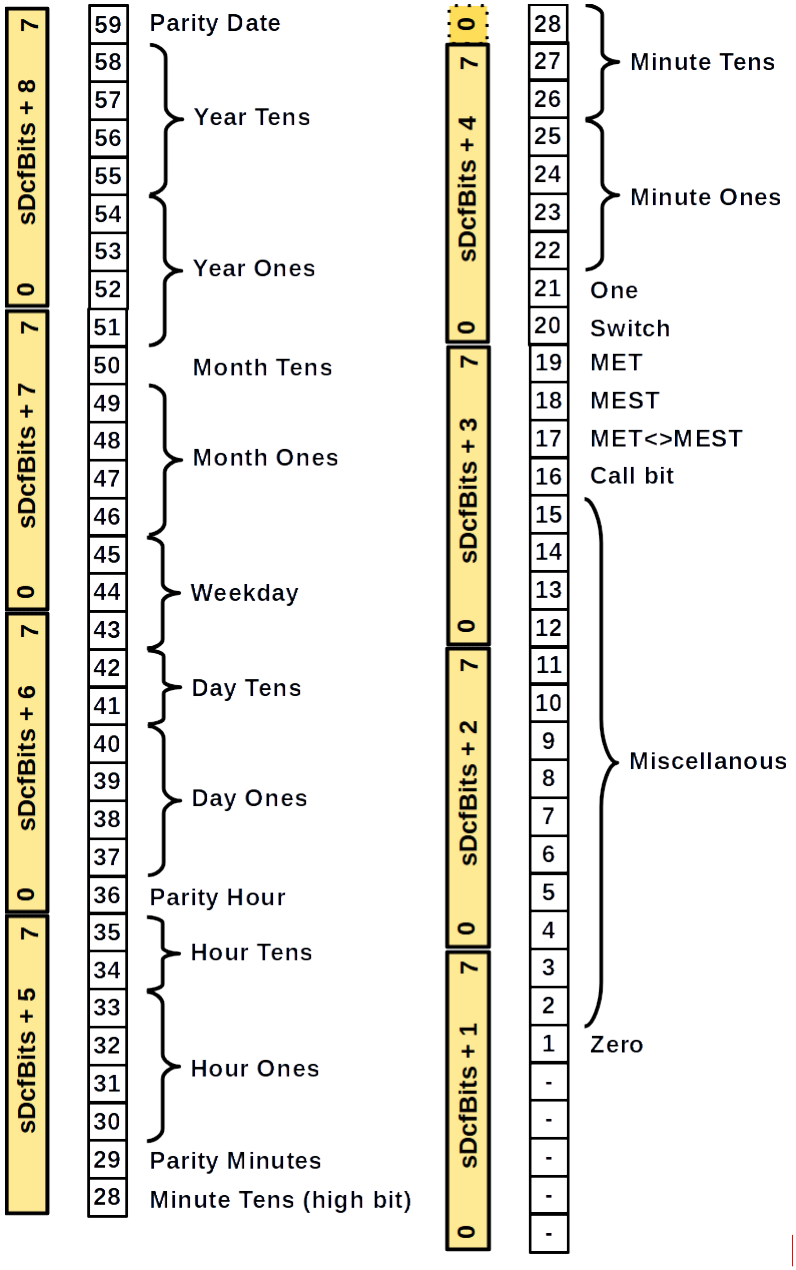


If not, the counter **rCnt** is decreased. If that reaches zero (after three consecutive highs), the direction bit **bHi** is set and the duration of the low phase is checked if it is within a zero or one range. If it is not in these two ranges (too short, between a zero and a one, longer than a one), an error is generated and processed (with output, if so configured).

Correct zeros and ones are written to the carry flag **C** and are shifted into the eight DCF bit storages in the SRAM (sDcfBitsN). The number of received bits is increased, if 59 is exceeded another error results.

During a high phase (right side of the diagram) the same recognition of a level change takes place. If so, the duration is compared with the range of a minute change (1800 ms - tolerance to 1900 ms plus tolerance plus one). If the duration was shorter than a minute change, it is checked whether the signal was between 800 ms - tolerance and 900 ms plus tolerance plus one. These signals appear between single bits and are ok (no error message triggered).

If a minute change has occurred, it is checked whether exactly 59 bits had been registered. Then it is checked whether the parity bits for hours, minutes and dates are correct. If that is also the case, the DCF bits are converted to time and date in ASCII (see next chapter) and, if so configured, converted to UTC (see overnext chapter). If the time and date are completed, those are send over the serial interface, if so configured.



10.4.1 Conversion of the DCF77 data bits to time/date

This is the location of all 59 bits that result from right-shifting of all bits in SRAM.

The following remarks to that:

- The 59th bit (Parity date) includes all date bits altogether, the parity bits for hours and for minutes include only these bits.

- Information, that is completely inside a byte, such as the year-tens, can be copied and shifted right (in this case three times), the upper bits be cleared with an ANDI on the lower bits (in this case 0x0F), and an ASCII-Zero added to yield the decimal digit.
- Bits that span over more than one byte have to be copied and shifted left over Carry into the higher byte first.
- The two bits 21 and 1 have to be one and zero, otherwise an error has happened.
- The bits 19 to 17, that deal with the summer and winter time, are only needed for the UTC conversion.

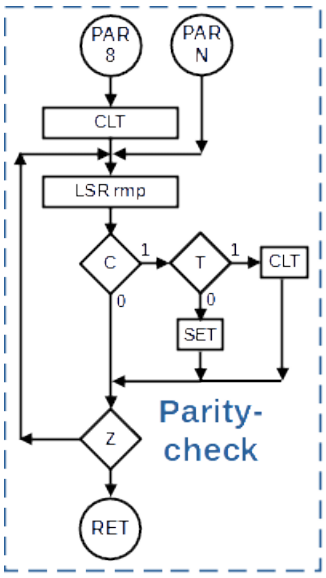
When accessing such very long records of bytes the AVR's ability to access bytes, where Y and Z point to the base, with a distance and with LDD/STD is very helpful. To use this feature I grouped the time/date record as follows and added distance values.

The SRAM table is structured systematically to optimize transparency and understandably.

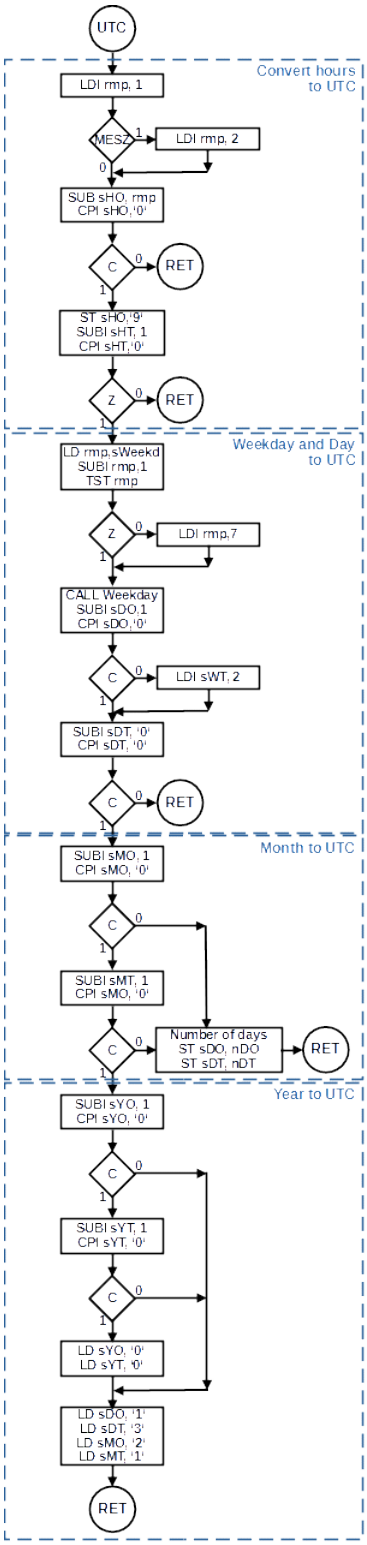
The column **Dist** gives the displacement for accessing the byte with LDD r,Y/Z+d or STD Y/Z+d, r. The formulation LD R16,Y+dStrT yields the content of the table on the position dStrT, if Y points to **sDcf**.

Dist	Name	Category	Content	Use
0	sDcf	Counter	Received bits	n bits
1	dByte8	DCF77 bits	Year, Parity-Date	Shift-register for DCF77 bits
2	dByte7		Year, Month, Weekday	
3	dByte6		Weekday, Day, Par-Hr	
4	dByte5		Hour,Par-Min,Minute	
5	dByte4		Minute, One, Switch	
6	dByte3		ME(S)T, Call, Misc.	
7	dByte2		Misc.	
8	dByte1		Zero, Misc.	
9	dErr	Error nmbr	0 (no error) to NDcfErr	Error
10	dStrT	Long-format-string	'T' or 'Z'	
11	dStrHT		Hour-Tens	
12	dStrHO		Hour-Ones	
13	dStrSep1		':'	
14	dStrMiT		Minute-Tens	
15	dStrMiO		Minute-Ones	
16	dStrSep2		':'	
17	dStrST		Second-Tens	
18	dStrSO		Second-Ones	
19	dStrUtc		'M/S/U'	
20	dStrWd1		'M','T/D','W/M','T/D','F','S','S'	
21	dStrWd2		'o','u/i','e/r','h/o','a','u/o'	
22	dStrSep3		'D'	
23	dStrDT		Month-Tens/Day-Tens	
24	dStrDO		Month-Ones/Day-Ones	
25	dStrSep4		'/' or '.'	
26	dStrMoT		Day-Tens/Month-Tens	
27	dStrMoO		Day-Ones/Month-Ones	
28	dStrSep5		'/' or '.'	
29	dStrYT		Year-Tens	
30	dStrYO		Year-Ones	
31	dStrCR		Carriage return 0x0D	
32	dStrLF		Line feed 0x0A	
33	dStr0		ASCII-Null 0x00	

10.4.2 Checking the parity



This is the parity checking rountine's flow diagram. Normally, when hours and minutes parities are checked, the T flag is cleared on entry. The data byte, including its parity bit, is send into the routine and it ends with a cleared T flag, if the parity is fine. If you have to check further bytes, like in the date parity, rather call ParN which uses the previous state of the T flag instead.



10.4.3 Conversion of the DCF77 date and time to UTC

This is rather lengthy if you want to convert time and date to UTC in a correct manner, so that the displayed date is also correct when the day already changed in ME(S)T time, but not yet in UTC time. That requires going back in the date, which is more or less complicated on the 1st of January, where anything changes, including the years.

10.5 Serial transmission of results and status

Serial sending uses TC1 as baud rate generator and for the whole timing of the transmission process. As the 8-bit TC1 in the ATtiny25 allows prescaler values between 1 and 16,384, a rather accurate timing can be achieved. Baud rates beyond 50,000 get more rough, but who needs such high rates at all. The inaccuracy is listed in the symbol table that gavrasm as well as avr_sim produce at the end of their listing. The two constants **cBaudEff** (effective Baud rate) and **cBaudDiff** in 0,01% resolution can be see. The 9k6 async baud rate comes out with 9,615 Bd, which is by 0.16% too high. Such small differences are insignificant for a robust RS232 interface.

During operation a number of result (each minute) and status messages (each second) can come up. Those are written to a transmit buffer in SRAM, which spans from **sBuf** to **sBufEnd**. If the line is complete, carriage return and line feed characters are added as well as an ASCII-Null to end transmission.

If the buffer is ready to transmit,

1. the pointer X points to the buffer start,
2. in the synchronous case **rTxCnt** is set to two,
3. a one is written to the bit counter **rTxBit** to provoke a read-next-character from buffer,
4. the flag **bTx** in the flag register **rFlag** is set,
5. the TC1 counter **TCNT1** is overwritten with zero, and
6. the interrupt mask register bit **OCIE1A** of the counter is set, by that allowing interrupts from TC1.

The complete transmission is performed within the ISR, of which two versions are in the source code and enabled by the constant **cTxMode**:

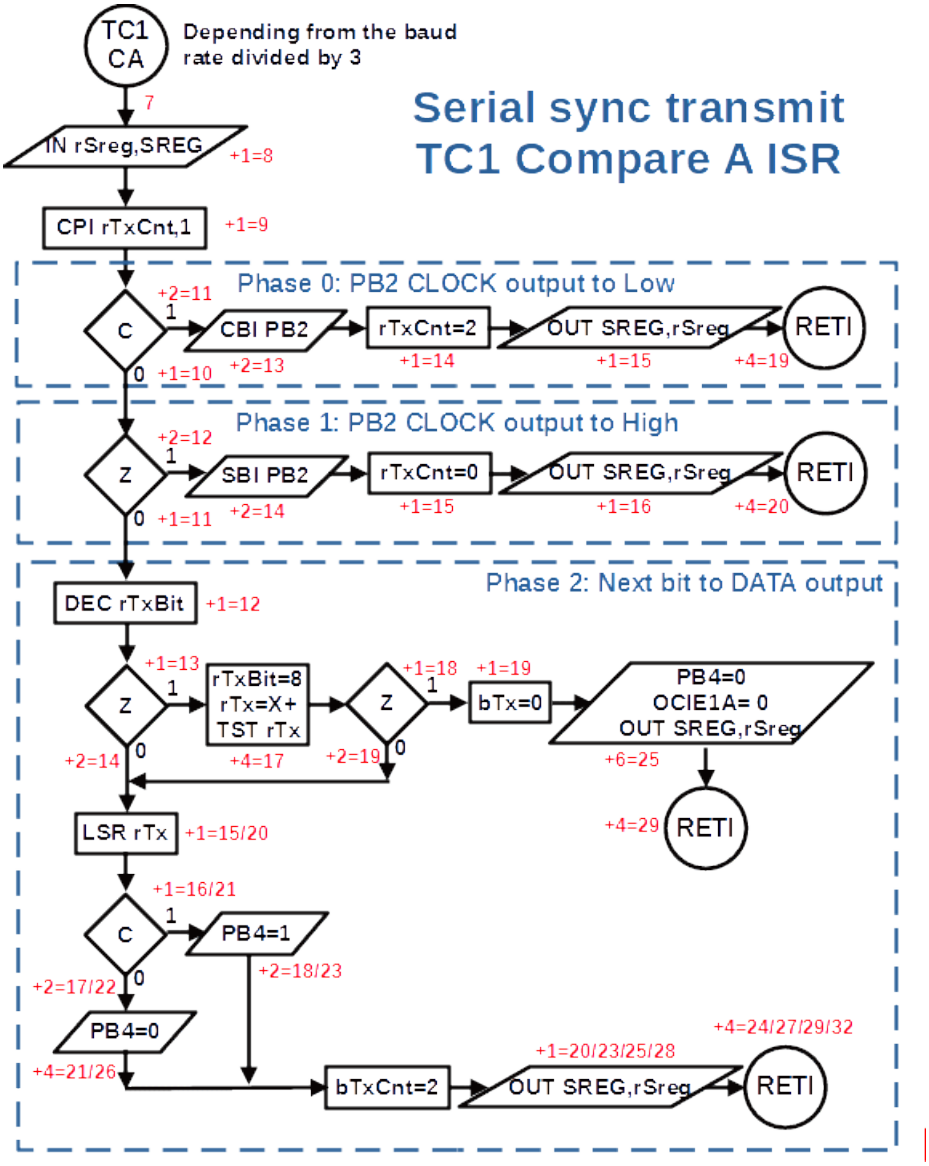
1. **cTxMode**=2 enables the sync mode,
2. 3 enables the async mode.

In both modes the constant b>cReverse = 1 inverts the polarity of both output pins.

Because the async- and the sync mode work different, those are described in two sub-chapters that follow.

10.5.1 Serial transmission in sync mode

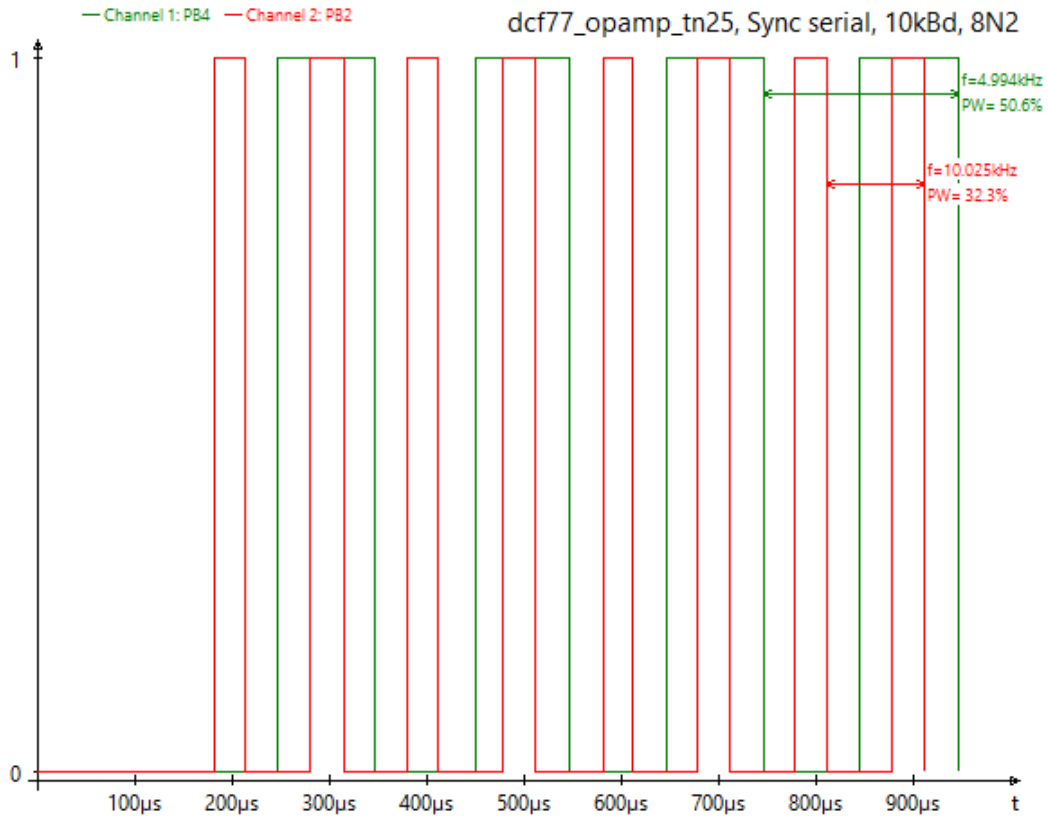
The flow diagram shows the interrupt execution of the TC1 Compare Match. The red numbers display the number of clock cycles required.



In sync mode the interrupt occurs three times more often than in async mode. The phase counter **rTxCnt** decides what to do next. 1 and 0 activate and deactivate the CLOCK output pin. Phase 2 places the next bit onto the DATA pin (lower part of the diagram) and decrements the number of bits to send in **rTxBit**. If that reaches zero, the next ASCII character is read from the buffer. If that is ASCII-Null, the transmit is terminated and the flag **bTx** is cleared as well as the interrupt flag of TC1. If not, the bit is placed to the DATA pin.

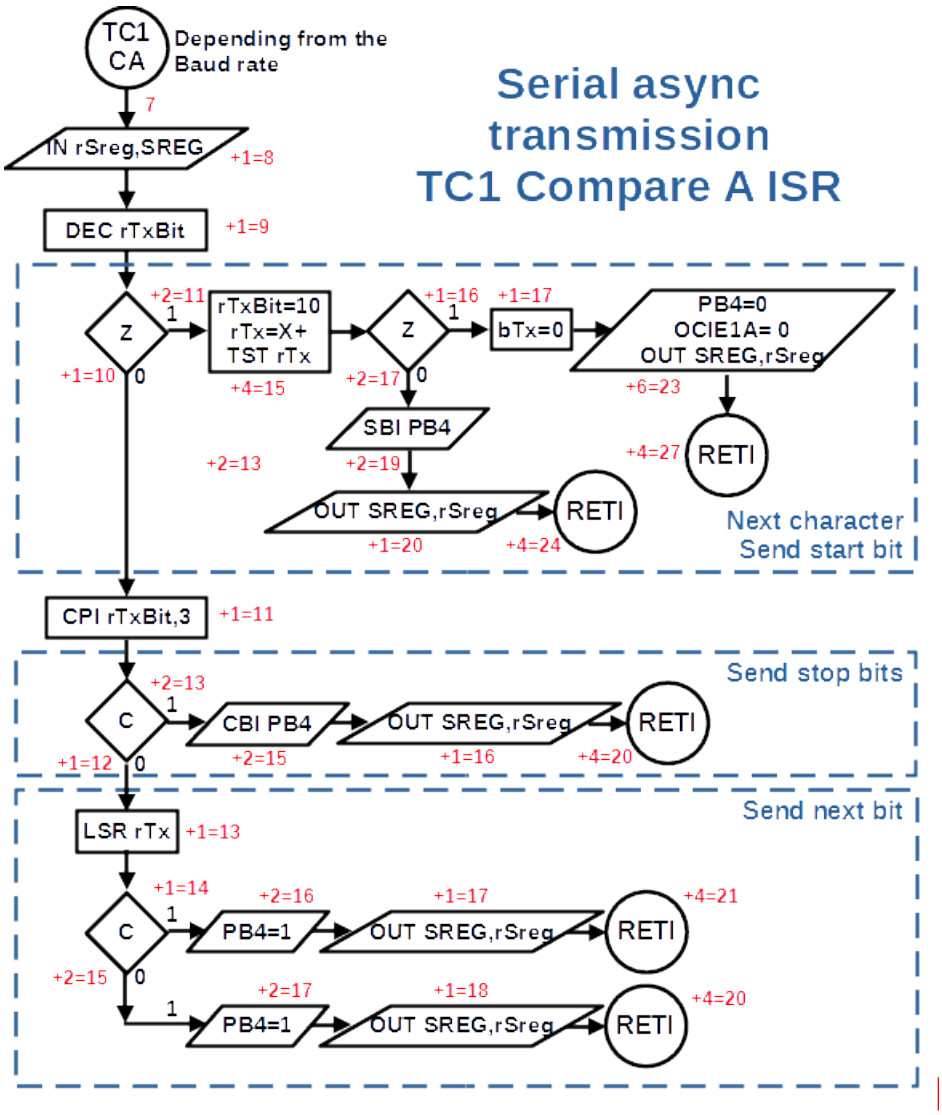
All different execution rows need less than 32 clock cycles. Only in a few cases ADC interrupts can be delayed, but only for short. Only in high baud rates beyond 60 kBd these cases block the complete interrupt scheme. Therefore the source code limits the sync baud rate to below 31 kBd.

If **cTxContent** selects the output of the short time format, the transmit routine needs, at 10 kBd, roughly 8.8 ms for that. When long format is selected, these are 18 ms long. As both occur only once in a minute, those are no relevant occasions.



This shows the transmission of hexadecimal 0xAA (=1010.1010) over the data and clock-Pins in sync mode.

10.5.2 Serial transmission in async mode



The flow diagram shows the interrupt execution in async transmission mode.

In async mode each interrupt stands for one bit. Per 8-bit-character one start bit and two stop bits are added. Due to this the baud rate is 8/11th of the character speed.

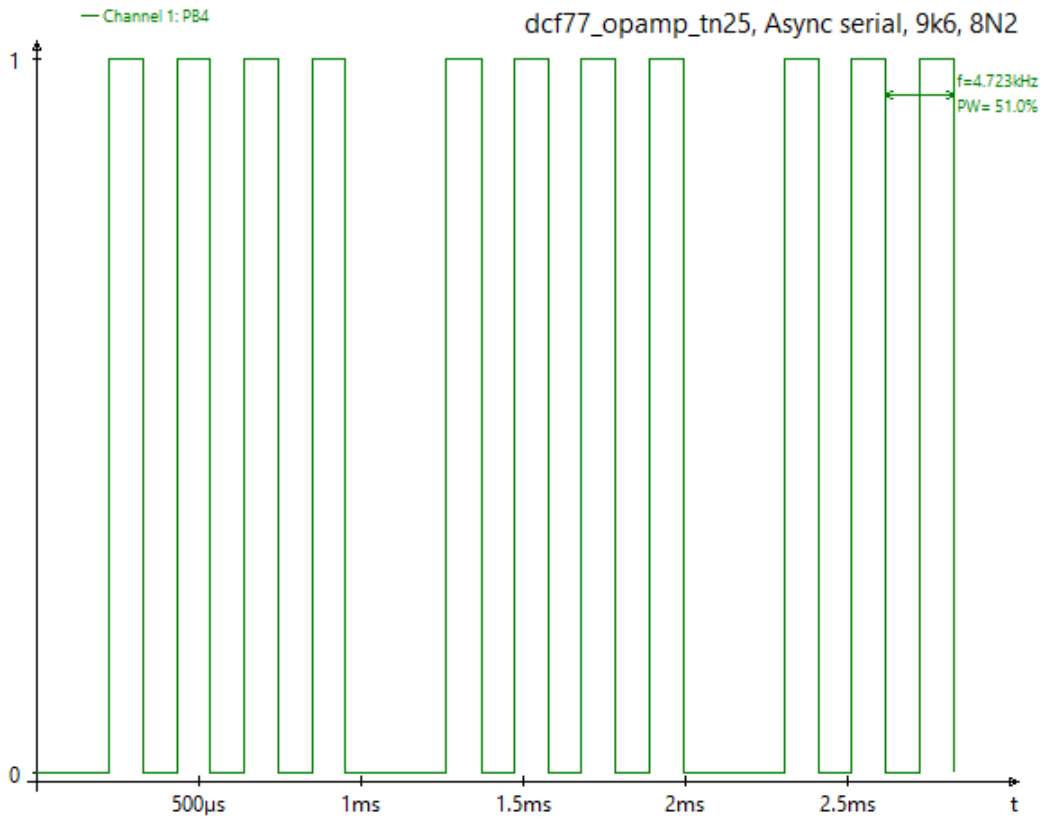
The interrupt service routine starts with decrementing the number of bits in **rTxBit**. If this reaches zero, the next character is read from the SRAM buffer. If that is ASCII-zero transmission ends. If not, the start bit is send (when **cRevert** = 0 the start bit is high, if 1 it is low). If the number of bits to be send is smaller than three stop bits are send. On any other number of bits one further bit is shifted into carry and the TXD output is set or cleared.

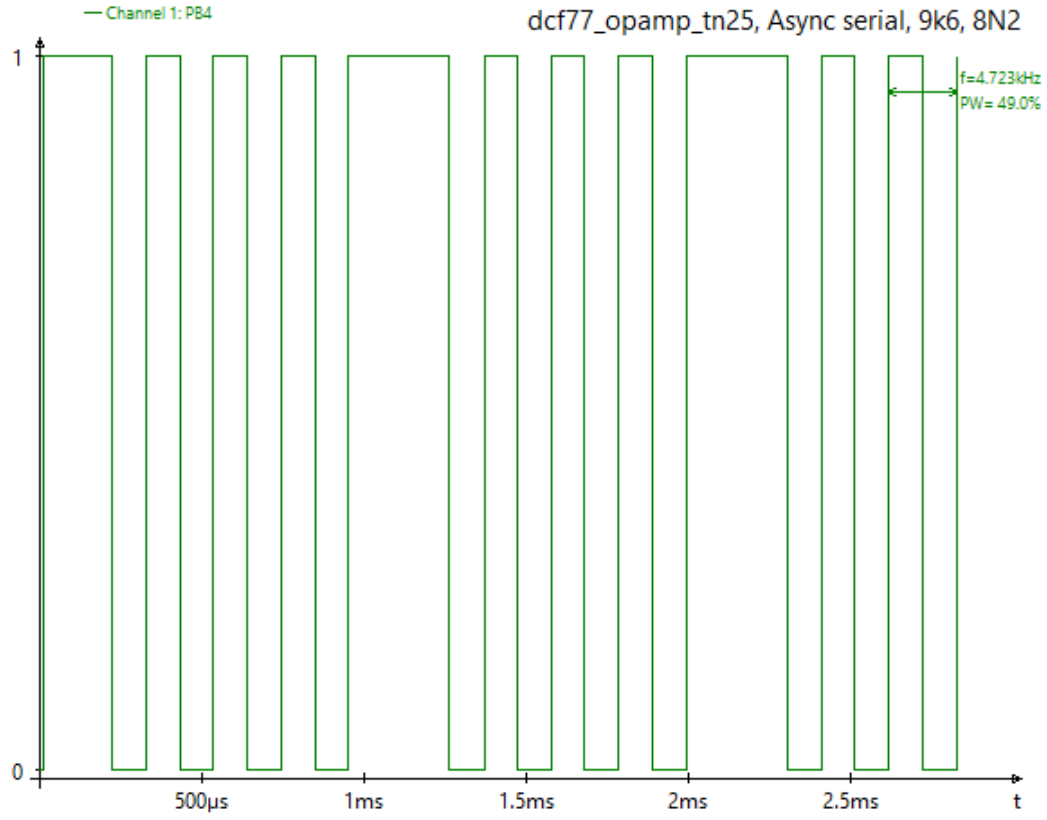
All different flow pathes require at max. 27 clock cycles. The maximum baud rate, at which the transmit blocks any other activities, is 74.1 kBd. If the time for each bit is considered correct, the baud rate for a blockade is reached at 95 kBd. Therefore the baud rate in async mode is limited to 47 kBd in the source code.

The minimum baud rate is below 45 Bd, low enough to start a DCF77 time service on short wave that transmits the DCF77 time/date/weekday via a Radio-Teletype (RTTY) transmitter.

At 9k6 the long format needs 25 ms, the short format 12,6 ms.

The two simulation diagrams show the character 0xAA in async mode at 9k6, to the right the inverted signal how it can be fed into a MAX232.


Channel 1: PB4
dcf77_opamp_tn25, Async serial, 9k6, 8N2
1
0
f=4.723kHz
PW= 51.0%
500µs
1ms
1.5ms
2ms
2.5ms
t

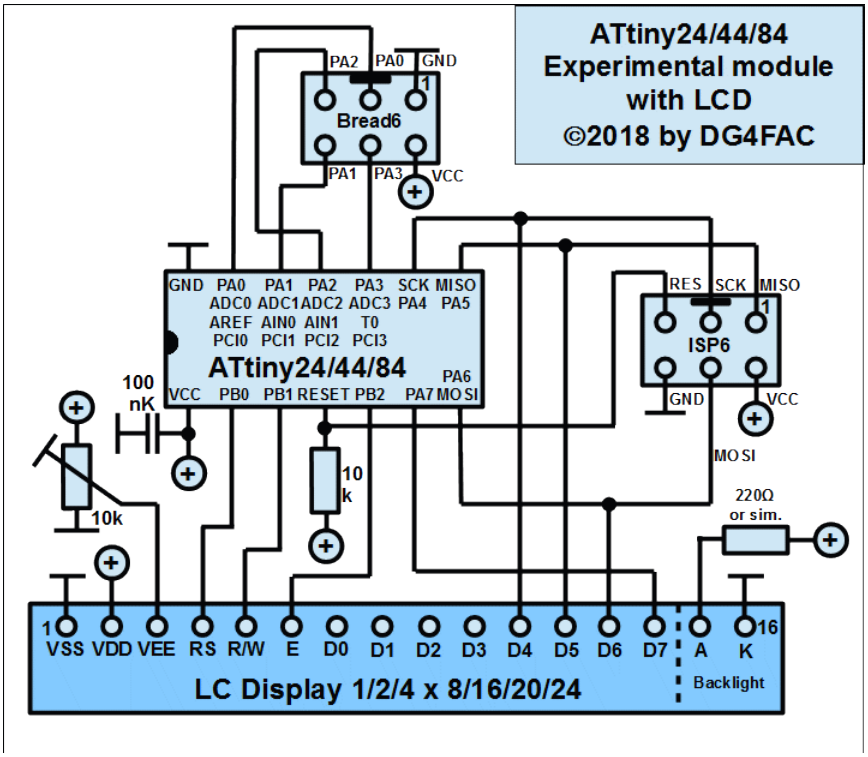

Channel 1: PB4
dcf77_opamp_tn25, Async serial, 9k6, 8N2
1
0
f=4.723kHz
PW= 49.0%
500µs
1ms
1.5ms
2ms
2.5ms
t

10.6 Sync serial receiver and LCD display with ATtiny24

The direct receiver with a regulated OpAmp outputs a sync serial signal in a short or long format, if so configured. This receiver here receives this sync signal and displays the received content on an LCD. With that, time, date, weekday as well as status messages from the receiver can be displayed, depending from the size of the attached LCD.

10.6.1 Necessary hardware

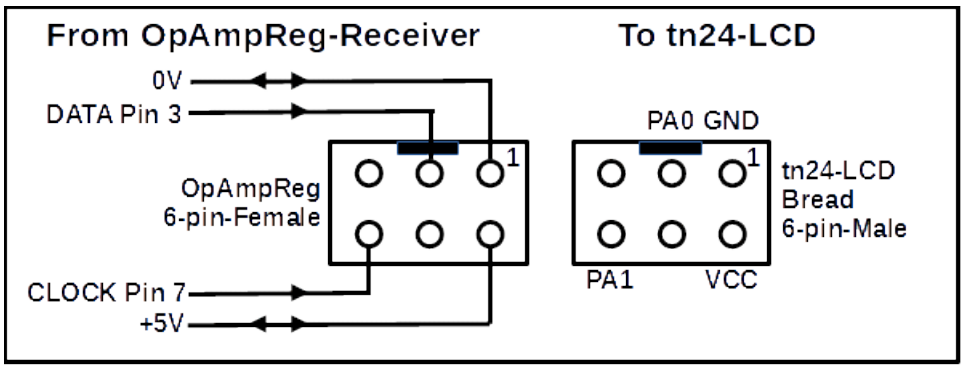
The necessary hardware is published here with a detailed description. It is possible to attach the following LCD types to it:



- Single line, 8 characters: In the OpAmpReg software select the short format, no status messages, Display of the time as: 00:00:00
- Two lines, 16 characters: In the OpAmpReg software select the long format and short status messages, Display: Line 1 = 00:00:00 U/M/S, Line 2 = 01.01.00 Wd E0
- Four lines, 20 characters: In the OpAmpReg software select the long format and long status messages, make sure that cEN is equally defined in both softwares, Display: Line 1 = Time = 00:00:00 U/M/S, Line 2 = Date = 12/31/20 Wd, Line 3 = Long DCF error messages, Line 4 = Status messages

To attach the OpAmp receiver to the tn24lcd board you need only a 6-pin parallel cable that fits into the 6-pin male plug on the tn24lcd board.

That is all. Anything else is done by the software.



10.6.2 The software

The software is written in Assembler, of course. It requires the LCD-Include-Routines to assemble correct. The following entries have to be changed to match with your hardware:

```
;.equ clock = 1000000 ; Clock frequency of controller in Hz
; LCD size:
  .equ LcdLines = 1 ; Number of lines (1, 2, 4)
  .equ LcdCols = 8 ; Number of characters per line (8..24)
; LCD bus interface
  .equ LcdBits = 4 ; Bus size (4 or 8)
  ; If 4 bit bus:
    .equ Lcd4High = 1 ; Bus nibble (1=Upper, 0=Lower)
  .equ LcdWait = 0 ; Access mode (0 with busy, 1 with delay loops)
; LCD data ports
  .equ pLcdDO = PORTA ; Data output port
  .equ pLcdDD = DDRA ; Data direction port
; LCD control ports und pins
  .equ pLcdCEO = PORTB ; Control E output port
  .equ bLcdCEO = PORTB2 ; Control E output portpin
  .equ pLcdCED = DDRB ; Control E direction port
  .equ bLcdCED = DDB2 ; Control E direction portpin
  ;equ pLcdCRSO = PORTB ; Control RS output port
  .equ bLcdCRSO = PORTB0 ; Control RS output portpin
  .equ pLcdCRSD = DDRB ; Control RS direction port
  .equ bLcdCRSD = DDB0 ; Control RS direction portpin
; If LcdWait = 0:
  .equ pLcdDI = PINA ; Data input port
  .equ pLcdCRWO = PORTB ; Control RW output port
  .equ bLcdCRWO = PORTB1 ; Control RW output portpin
  .equ pLcdCRWD = DDRB ; Control RW direction port
  .equ bLcdCRWD = DDB1 ; Control RW direction portpin
; If you need binary to decimal conversion:
  ;.equ LcdDecimal = 1 ; If defined: include those routines
; If you need binary to hexadecimal conversion:
  ;.equ LcdHex = 1 ; If defined: include those routines
; If simulation in the SRAM is desired:
  ;.equ avr_sim = 1 ; 1=Simulate, 0 or undefined=Do not simulate
```

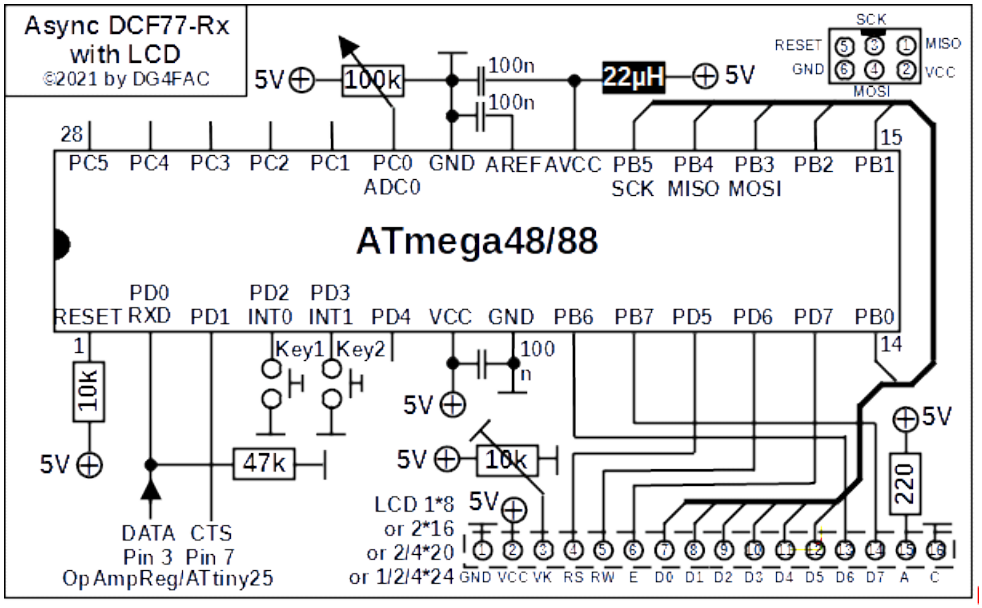
Further properties to be adjusted in the source code are:

- **cEN**: 1 provides english display format,
-

10.7 Async serial receiver and LCD display with ATmega48

The direct receiver with a regulated OpAmp outputs an async serial signal in a long format, if so desired. This receiver here reads this async signal and displays the received content on an LCD. With that, time, date, weekday as well as status messages from the receiver can be displayed, depending from the size of the attached LCD.

10.7.1 Necessary hardware



To not having to write an async serial receiver in assembler I chose an AVR that already has an integrated async receiver on board. Many AVRs have that, I chose an ATmega48 for that. If you have an ATmega88, you can simply replace the m48 by the m88, just change the include header in the source code.

Those who want to clock their DCF77 seconds with an XTAL, cannot use this hardware. The ATmega48 has no complete 8-bit data bus for the LCD when enabling an external crystal, so I decided to skip the crystal here. That makes the watch malfunctioning with 3% inaccuracy. If you want a crystal-driven device you can change to an ATmega324 like here or you'll have to drive the LCD with a four-bit data bus.

The serial signals are received by the Mega48 on its RXD pin. Because the serial signal is directly attached and not inverted, choose **cRevert** = 0 in the source code software for the ATtiny25. This switches the TXD and CTS pins of the ATtiny25 to active high. After the ATmega48 has absolved its initialization, he turns PD1 on, switching Clear-To-Send back on the ATtiny25 on.

The pins PD2 and PD3 are attached to two keys. Those are for adjusting the clock to current time even without a valid DCF77 signal coming in. Pressing key1 starts the adjustment of the hours with an INT0 signal, the correct hour can be selected with the potentiometer that is attached to ADC0. If correct, press key2 (INT1) and the minutes are adjusted. Pressing Key2 again sets the time.

The LCD is attached with its data bus to Port B, the three control inputs are on PD5 to PD7. You can attach any LCD of any size to it, just change the number of rows and columns in the source code. Of course, the size of the LCD determines what is displayed.

Calculation basics for this project are in the Libre-Office-Calc-File here.

10.8 Async serial receiver and LED display with ATmega324

The direct receiver with a regulated OpAmp outputs an async serial signal in a long format, if so desired. This receiver here reads this async signal and displays the received content on an LCD. With that, time, date, weekday as well as status messages from the receiver can be displayed, depending from the size of the attached LCD.

10.8.1 The LED display of the clock

The display used here has been described here. It works with four decimal digits with 28 pieces of 10mm-LEDs, that are organized in seven segments by four LEDs each. Each segment is attached to a constant current driver. In the middle between the four digits a double point is located with two LEDs, altogether 114 LEDs. The original works with an ATmega48.

10.8.2 Necessary hardware

To additionally enable adjustment of the watch via an async serial interface would have resulted in large changes to the original ATmega48 design, because the original already used RXD for a different signal. So I had to change to another controller.

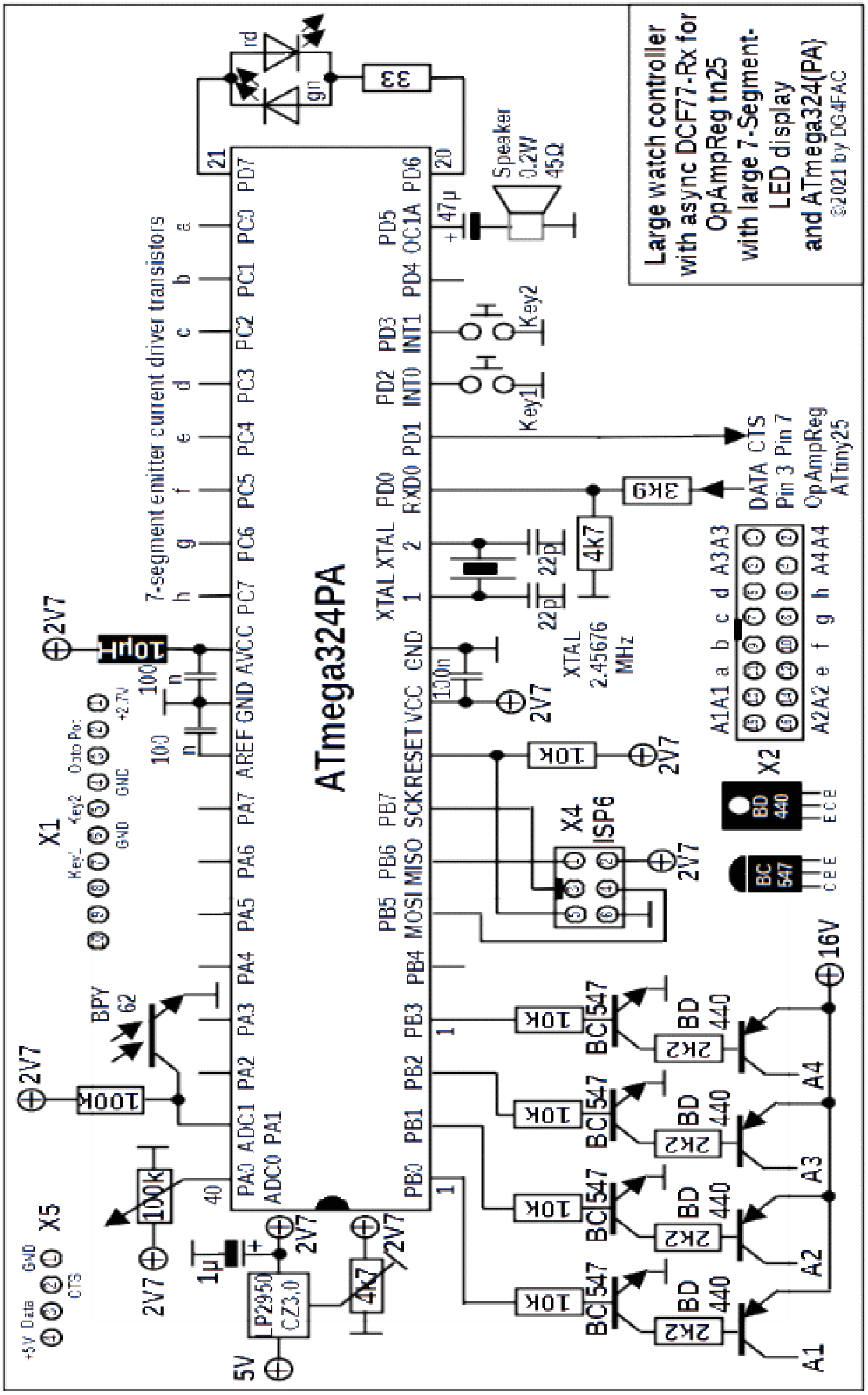
After consulting the AVR selection window in avr_sim I decided to try it with a 40-pin AT324PA, which is commercially available and cheap and has all that is needed for the large watch:

- two crystal pins for clocking the controller with exact second pulses, so that the watch also works without DCF77,
- an 8-bit-bus for multiplexing the seven segments and the double point,
- a 4-bit-bus for driving the anodes of the 7-segment display,
- two INT pins for the two keys,
- two ADC channels for the potentiometer and the background light sensor, and
- the UART input pin RXD0 and a single I/O pin for the CTS output.

The schematic shows how all components are connected to the ATmega324. As a gimmick a red-green dual LED has been added that can display the state of the DCF77 receiver:

- permanently green shows synchronization with DCF77,
- permanently red signals missing synchronization with DCF77,
- blinking red shows read errors on the async interface.

Calculation basics are in the Libre-Office-Calc file here.


Large watch controller
with async DCF77-Rx for
OpAmpReg tn25
with large 7-Segment-
LED display
and ATmega324(PA)
©2021 by DG4FAC
ATmega324PA
7-segment emitter current driver transistors
Speaker
0.2W
45Ω
DATA CTS
Pin 3 Pin 7
OpAmpReg
ATtiny25
XTAL
2.45676
MHz
X1
X2
X4
ISP6
X5
LP2950
CZ3.0
BPY
62
16V