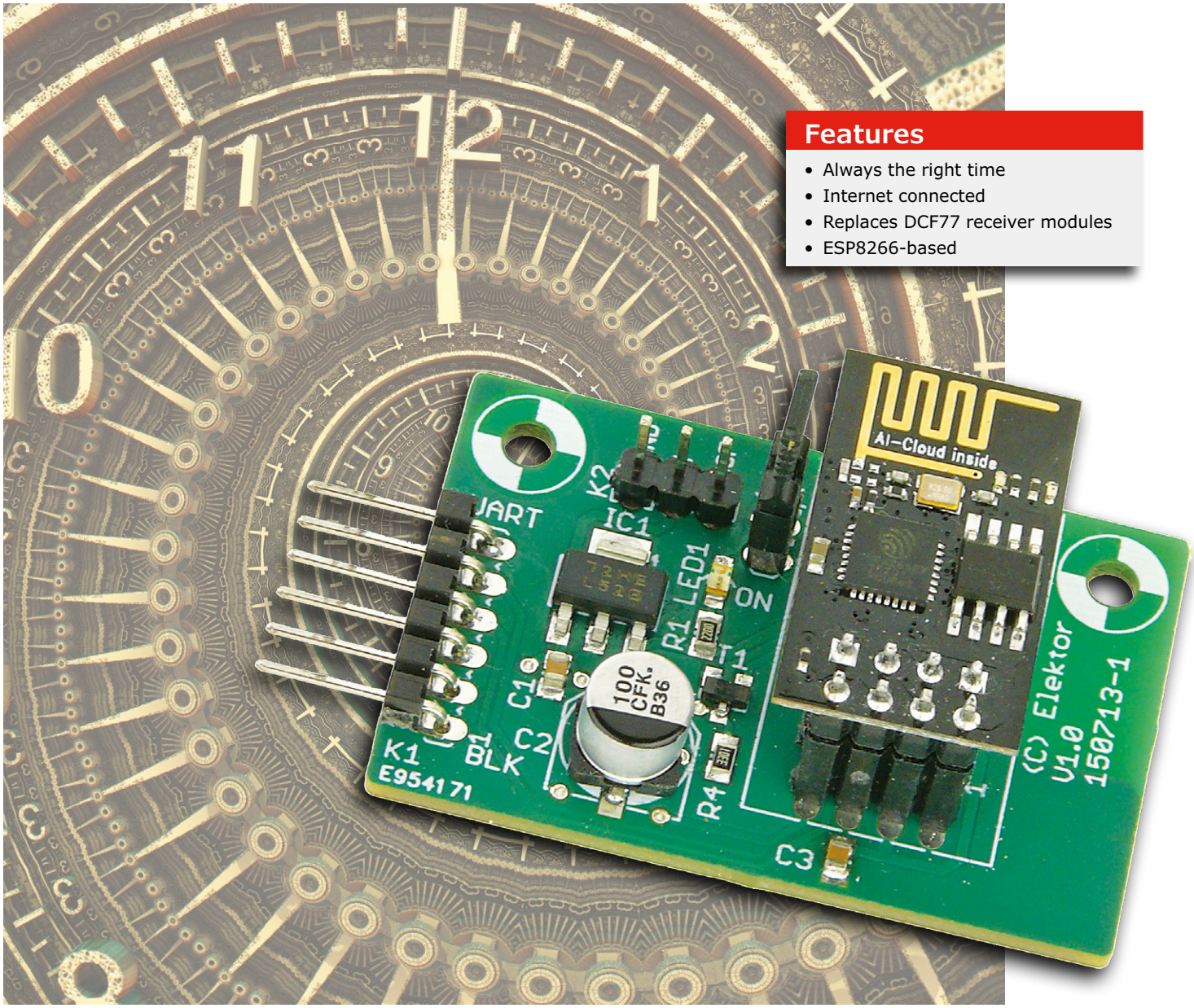
DCF77 Emulator with ESP8266

Replace over-air time by Internet time



By **Massimo Fusari** (Italy) & **Luc Lemmens** (Elektor Labs)

About twenty years ago I recycled and modernised a vintage clock with Nixie tubes built by my father in the nineteen seventies. I replaced the digital logic by a microcontroller and used a DCF77 receiver module instead of the original 50-Hz derived timebase.

Over many years the clock has worked fine, but recently DCF77 reception in my house has worsened due to electromagnetic interference created by modern switching power supplies, I suppose. So I decided to replace the DCF77 receiver by some form of Network Time Protocol (NTP) client.

An ESP-01 ESP8266-based module would be perfect for the job. It is cheap, powerful, and can be easily programmed in Arduino style with all the benefits of open source libraries. The result is a DCF77 receiver emulated by an ESP-01 module connected to my home Wi-Fi network. Only one output pin is required to drive the old Nixie clock.

Although the idea behind this project is quite straightforward, the implementation may be a bit more elaborate. The software may need a few modifications to make it compatible with your DCF77 clock, so Luc at Elektor Labs designed a small printed circuit board (PCB) that makes it very easy to (re)program the ESP-01 module if and when needed. The circuit presented here therefore is a DCF77 emulator and ESP-01 programmer rolled into one circuit.

The hardware is simple

The schematic of the emulator (**Figure 1)** isn't particularly complex. The ESP-01 module is represented here by MOD1, powered by a 3.3-V low-dropout (LDO) voltage regulator (IC1). Jumper JP1 is available for putting the ESP-01 into programming mode.

LED1 indicates that the supply voltage is present, however since the ESP-01 has its own power LED, LED1 (together with R1) may be omitted.

K1, which is pin-compatible with an FTDI cable, permits connecting a USB-to-serial converter **of the 3.3 V type**.

Transistor T1 will make interfacing the emulator to your clock slightly easier as it translates the 3.3-V level of the ESP-01's output to the logic voltage level of your DCF77 clock. Collector resistor R4 may be omitted if the clock at hand already has a pull-up resistor at the input of the DCF77 decoder.

The DCF77 output signal gets inverted by T1, but the software fixes this. You may need to check the documentation of the DCF77 receiver module in your clock to see if the output signal should be inverted or not. Of course, it is always possible to use the good-old 'trial and error' method to find out which polarity works for you, or use an oscilloscope to observe the output signal polarity from your original DCF77 receiver.

All the hard work is done by ReadAndDecodeTime

The program is in the shape of an Arduino sketch. As is customary for these kinds of programs, they first execute a function called `setup` and then continuously repeat a function called `loop`.

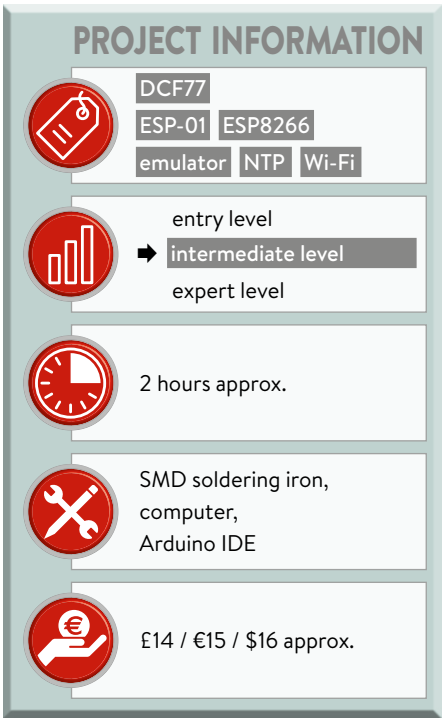
The function `setup` first sets up the ESP-01's inputs and outputs and it initializes some global variables. It also starts a timer that fires every 100 milliseconds. This timer is used to produce the DCF77 encoded bit stream (refer to the Internet for the details about the DCF77 protocol). Finally, a connection to the Wi-Fi network is established.

The function `loop` is very simple and runs just once every minute. When it does, it either connects to the time server to obtain the current time or it tries to reconnect to the network because the connection was lost for some reason.

The reason for the function `loop` to idle is because it delegates all the hard work to the function `ReadAndDecodeTime`. This is the workhorse that really connects to the NTP server to request the time and then converts it into something that can be easily encoded in the DCF77 format. The function `CalculateArray` is assigned with this task. It converts the different values that make up a valid frame into zeroes and ones at the right position in the frame.

The 100-ms timer mentioned above calls the function `DcfOut`. This function reads the bit array filled by `CalculateArray` and delivers the values in the shape of 100-ms (zeroes) and 200-ms (ones) pulses on pin GPIO2.

The conversion of NTP time to DCF77



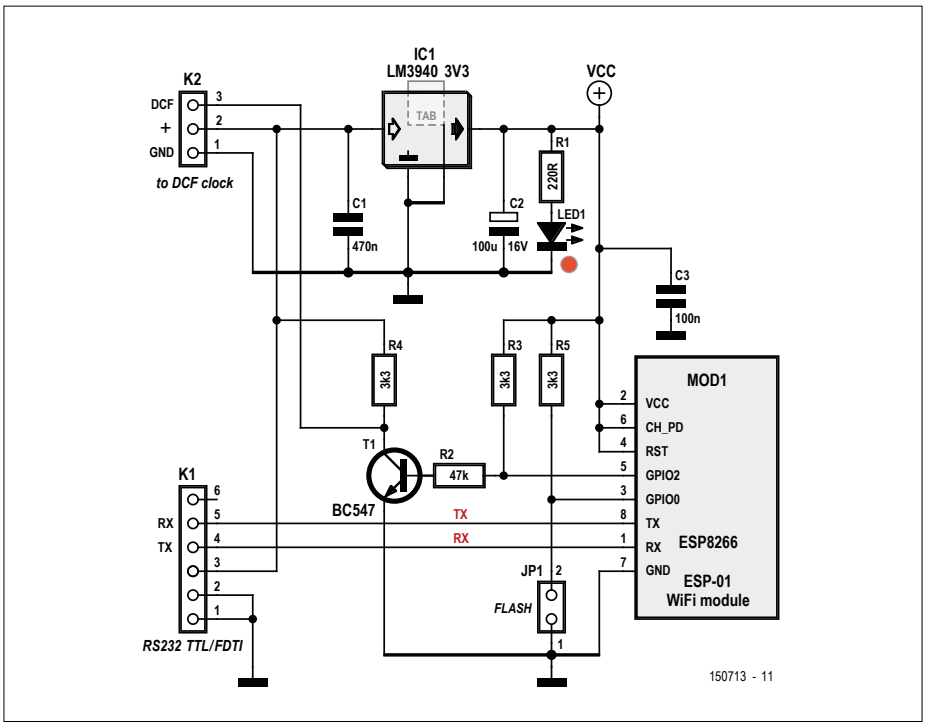


Figure 1. The intelligence of the DCF77 emulator resides in the ESP-01 Wi-Fi module.

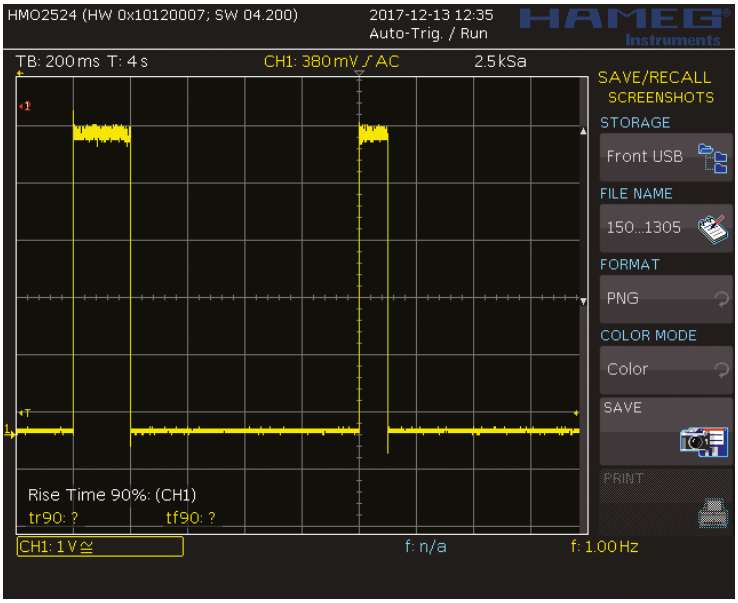


Figure 2. The pulses produced by the emulator are timed perfectly. A logic one (left pulse) is 200 ms, a zero is 100 ms and the time between each pulse is one second.

time might seem a trivial task but it's likely more complicated than you think. First of all, the time received from the NTP server is the number of seconds elapsed since 1900. Second, the library used for time calculations uses Unix time — the number of seconds elapsed since 1970 — so NTP time must first be converted to Unix time. Third, the DCF77 protocol transmits the time of the next minute ("At the third stroke, the time will be…"), not the actual time. Finally, the NTP time received is probably not on a minute boundary. To correct for this we subtract two minutes from the received time and send them to the clock, followed by a third complete minute. This way we ensure that the clock receives enough data to be able to synchronize to it and extract the correct time.

Construction

Although many parts are SMT (surface mount technology) types, soldering and assembling will not be too difficult.

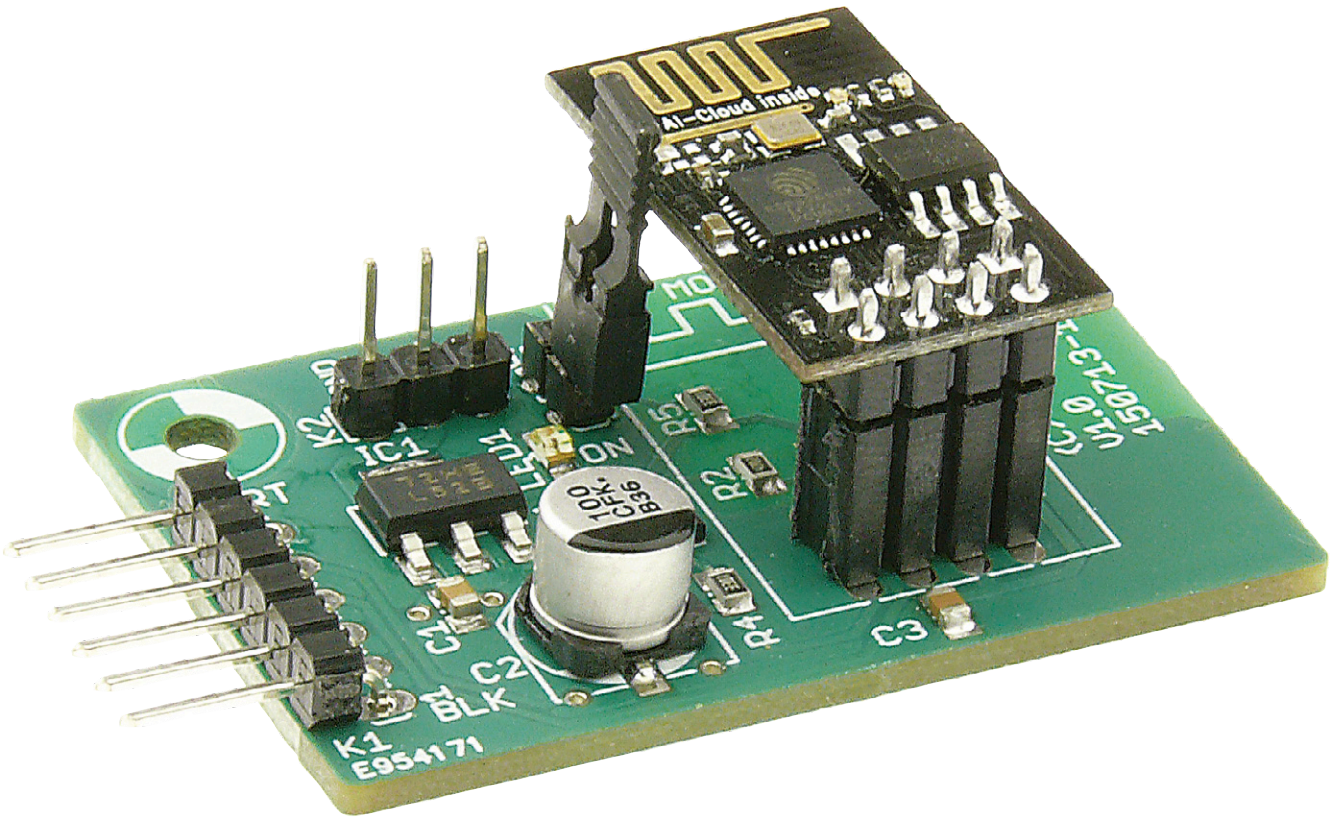
Even though the BOM includes an 8-way (2×4) socket strip for mounting the ESP-01 module, to save space (height) and improve mechanical stability it is preferable to solder the ESP-01 directly on the main PCB without this socket. You can also temporarily mount a socket and remove it once you are sure that everything is working fine.

The sketch has to be adapted to your setup

After assembling the board, it is time to program the ESP-01 module with the Arduino sketch ESP8266_NTPtoDCF available as a free download from [1]. If you have never programmed an ESP8266-based module before with the Arduino IDE, it is necessary to install the ESP8266 Boards Pack age first. More on this subject can be found on [2] and many other websites, as well as in articles.

Three things need to be set up to make the sketch work with your DCF77 clock:

1. the credentials for your Wi-Fi network;
2. the URI of the time server used for synchronization;
3. the polarity of the DCF77 emulator's output.



The ESP-01 needs to connect to your Wi-Fi network to enable it to collect time information from an NTP server. To make this work the credentials for the Wi-Fi network must be entered at the top of the sketch:

```
char ssid[] = “your_network_name”;
   // your network SSID (name)
char pass[] = “network_password”;
   // your network password
```

The URI of the NTP time server must be defined too:

```
const char* ntpServerName = “0.nl.
   pool.ntp.org”;
```

In this case a server for the Dutch (NL) time zone is specified, but you may prefer another one.
The polarity of the DCF77 emulator output signal is defined in the function `DcfOut`:

```
case 0:
  if (PulseArray[PulseCount]!=0)
    digitalWrite(LedPin,0);
  break;
case 1:
  if (PulseArray[PulseCount]==1)
    digitalWrite(LedPin,1);
  break;
case 2:
    digitalWrite(LedPin, 1);
  break;
```

Shown here is the active-high version (output normally logic low, pulses are logic high). For an active-low output signal (output normally logic high, pulses logic low) invert the ‘0’ and the ‘1’s of the three `digitalWrite` commands.

Programming the ESP-01

Before we continue, an important remark first: never connect power to K2 when a USB-to-serial converter is powering the circuit through K1 (or the other way around). Disrespecting this No-No will short two power supplies, and may damage your DCF77 clock, your computer or both. Don’t come crying to us.
Close jumper JP1 and connect a 3.3-V FTDI-cable-compatible USB-to-serial converter between K1 and your computer. JP1 must be closed at power on to switch the ESP-01 module to flash (programming) mode. In the Arduino IDE, on ‘Tools’ menu, select ‘Generic ESP8266 Module’ as board type and the correct COM port number of your USB-to-serial interface. Compile and upload the sketch. When the upload is completed open the ‘Serial Monitor’ in the Arduino IDE (again in the ‘Tools’ menu). The ESP module will echo useful information to your computer screen to check if the DCF77 simulator is doing its job correctly. First it will show if the ESP-01 is able to connect to your network. If it isn’t, verify that you entered the correct network SSID and password in the sketch.
Once a connection has been established, the time server is read and decoded time information is displayed. Please note that the time on your screen is two minutes fast — this lead is needed to synchronize the DCF77 clock in time!
If the connection to the time server is not made, you may have misspelled its URI. Correct it in the sketch.
Every time you make modifications to the sketch it must be recompiled and reprogrammed into the ESP-01 module. Remember to power cycle the circuit (unplug and plug K1) to put the module back in programming mode.

Install the emulator in the clock

Once the information in the Serial Monitor is correct, the circuit is ready to be installed in your DCF77 clock.
Remove the programming cable from K1 and open jumper JP1. Remove the old DCF77 receiver from your clock. In many cases it will be a separate module with three wires (power, 0 V and DCF77 signal). Connect K2 of our circuit to the now unconnected input of the clock.
If you want to be sure that the output of our DCF77 emulator produces a valid DCF77 encoded signal, there are numerous Arduino-based (test) projects on the Internet with DCF77 decoders. We tested our prototype with the sketch found at [3]. ⏮

(150713)

Web Links

[1] www.elektormagazine.com/labs/dcf77-emulator-with-esp8266-elektor-labs-version-150713

[2] https://github.com/esp8266/Arduino

[3] https://arduino-hannover.de/2012/06/14/dcf77-empfanger-mit-arduino-betreiben/

[4] www.elektormagazine.com/150713

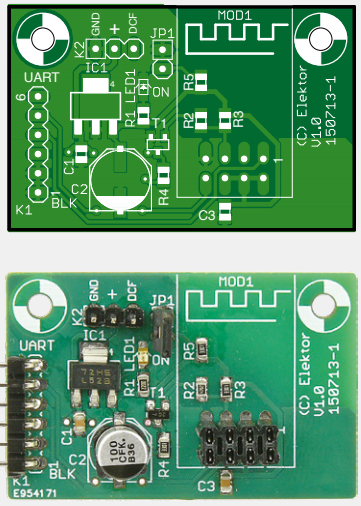
COMPONENT LIST

Resistors
All 0805, 5%, 0.1 W
R1 = 220Ω
R2 = 47kΩ
R3,R4,R5 = 3.3kΩ

Capacitors
All 0805
C1 = 470nF
C2 = 100µF, 16V, radial can SMD, 6.3mm diameter
C3 = 100nF

Semiconductors
IC1 = LM3940IMP-3.3/NOPB
LED1 = LED, red, 0805
T1 = BC847C

Miscellaneous
JP1 = 2-pin pinheader, 0.1” pitch
Jumper, 2-way, 0.1” pitch
K1 = 6-pin pinheader, 0.1” pitch
K2 = 3-pin pinheader, 0.1” pitch
MOD1 = ESP-01 Wi-Fi module
MOD2 = 8-way (2x4) pinheader socket, 0.1” pitch
PCB 150713-1 v1.0



FROM THE STORE

→ 150713-1
bare DCF77 emulator PCB

→ 150445-91
ESP-01 module