OUAH! project: a DCF77 clock on FPGA
with its simulator companion on Pico W (written in C)

VHDL development with Radiant from Lattice

Jacques Ehrlich

I. Summary

In this article we describe a DCF77 clock realized 100% by hardware. It is implemented on an FPGA and described in VHDL. This realization does not include the receiver and the demodulator of the DCF77 signal. It concerns exclusively the decoding of the signal available at the output of the demodulator and its display on 7 segments and LEDs of the hours, minutes, month, day of the month, year, day of the week and daylight saving (CEST/CEST) time. A printed circuit is also proposed. DCF77 receiver-demodulators are readily available on the market. But in France, reception of DCF77 signals transmitted from Germany is difficult outdoors due to the low power received and almost impossible indoors. Also, for the development of OUAH! did I need a DCF signal simulator. This simulator, written in C, implemented on Raspberry Pico W can be integrated in the same enclosure as the Ouah! Thus the clock can be used outdoor with the DCF77 receiver or indoor with the simulator provided that the Pico W is within the range of a Wi-Fi network.

II. Introduction

OUAH! is the French ironic acronym for **Oh ! une autre horloge** that you can translate by **Wow ! another clock.**

Indeed, in the literature and the columns of Elektor in particular, there is no shortage of clock projects based on the decoding of the DCF77 signal.

But I dreamed of developing a 100% hardware version based on the use of an FPGA to continue my self-training on this technology, self-training that I had started with the GONOGO project also published on the Elektor Lab site. In fact, it's more of a retraining because I had already used FPGAs professionally in the 80s when the first XILINX circuits were put on the market. Forty years later, the shortage of semiconductors having passed through there, it had become impossible to find FPGAs at reasonable prices and deadlines. In addition, a FPGA mounted on a board and surrounded by some components are all SMD components difficult to mount and which require means and a technique that I do not have.

But thanks to Mathias Claussen's article in Elektor #494, I discovered the Upduino3.1 board and that's how the dream came true. Indeed, this board has several advantages; low cost (around €40 incl. VAT at the time of writing), small footprint, low power consumption, very complete, programmable via a simple USB cable and quite respectable power since it is equipped with a Lattice ICE40UP5K FPGA that contains no less than 5280 cells which is more than enough for this project.

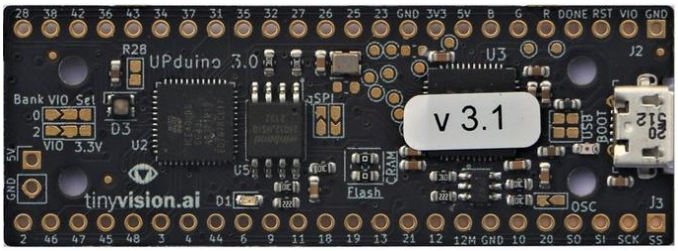


Figure 1– The Upduino3.1 board

The implementation of FPGAs is not so obvious. In addition to the fact that it is of course required to know an HDL language (i.e Verilog or VHDL), it requires the mastery of development tools, some of which may prove to be complex. I focused on two of them: APIO under Linux which exclusively supports Verilog programming and Radiant from Lattice under Windows which supports both Verilog and VHDL programming. These two tools are well documented and ultimately quite simple to implement.

Thus, the purpose of this article is not only to describe a development of OUAH! based on the Upduino3.1 but also how to implement it using the Radiant tool. Finally, a complete realization including a printed circuit with THT components is proposed.

III. The DCF77 signal

There are many sites on the internet describing the content of the DCF77 frame. One can in particular consult Wikipedia [1]. The information is coded in BCD in frames transmitted every minute at the rate of one pulse per second. The duration of the pulses is 100 ms for the transmission of a 0 and 200 ms for the transmission of a 1. It takes 59 seconds to transmit all the information required. There is no pulse between the 59th pulse of frame N and the 1st pulse of frame N+1: in other words, the 60th pulse is omitted which leaves two seconds of separation between two frames, thus allowing synchronization of the decoder. Figure 2 describes the coding of the different information contained in a frame. Note that the frame transmitted at time t indicates the time of time t+1 minute.

Bit	Poids	Signification	Bit	Poids	Signification	Bit	Poids	Signification
:00	1	Début de minute, toujours 0.	:20	S	Début du codage du temps, toujours 1.	:40	10	Jour du mois (suite)
:01	1	Témoins d'alerte civile[5] fournis par le *Bundesamt für Bevölkerungsschutz und Katastrophenwarnung*. Peut contenir également des données de prévision météo[6]	:21	1	Minutes 00–59	:41	20	
:02	1		:22	2		:42	1	Jour de la semaine Lundi=1, Dimanche=7
:03	1		:23	4		:43	2	
:04	1		:24	8		:44	4	
:05	1		:25	10		:45	1	N° du mois 01–12
:06	1		:26	20		:46	2	
:07	1		:27	40		:47	4	
:08	1		:28	p. 1	Bit de parité paire sur les bits 21–27.	:48	8	
:09	1		:29	1	Heure 0–23	:49	10	
:10	0		:30	2		:50	1	Année dans le siècle 00–99
:11	0		:31	4		:51	2	
:12	0		:32	8		:52	4	
:13	0		:33	10		:53	8	
:14	0		:34	20		:54	10	
:15	R	Bit d'appel, permet d'alerter les employés de PTB à Brunswick, responsables du DCF77, ou Émetteur de réserve	:35	p. 2	Bit de parité paire sur les bits 29–34.	:55	20	
:16	A1	1 = Annonce (pendant 1h) un basculement heure d'été/hiver au début de la prochaine heure	:36	1	Jour du mois. 01–31	:56	40	
:17	Z1	**Z1** et **Z2** indiquent le décalage horaire de l'heure émise par rapport au temps UTC.	:37	2		:57	80	
:18	Z2	si **Z1Z2 = 01** : UTC+1h = CET = heure d'hiver si **Z1Z2 = 10** : UTC+2h = CEST = heure d'été	:38	4		:58	p. 3	Bit de parité paire sur les bits 36–57.
:19	A2	1 = Annonce (pendant 1h) l'ajout d'une seconde intercalaire à la fin de l'heure (il y aura une 60e impulsion supplémentaire. Le silence est reporté à la 61e et dernière seconde. Puis, débute l'heure suivante.)	:39	8		:59	0	Marque de la minute : aucune modulation.

Figure 2– Coding of a DCF 77 frame – (source Wikipedia)

IV. THE OUAH! BOARD

Figure 5 shows the schematic of Ouah! It is relatively simple since everything is in the VHDL code of the FPGA. There are mainly three parts: the Updiuno card, the display by 7-segments and LEDs and the power supply which can optionally be backed up by a 9V battery.

The hours, minutes, month, day of the month and year are displayed on eight 7-segment displays.

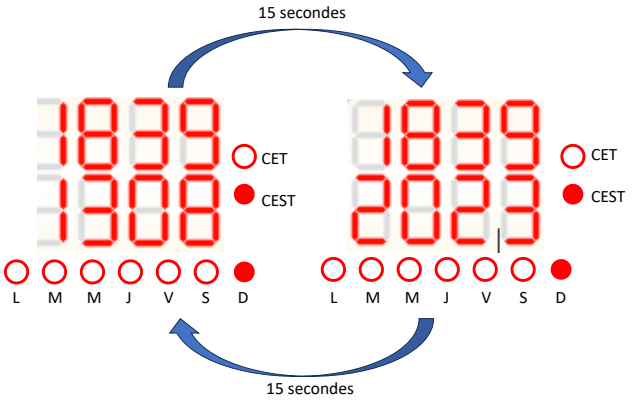


Figure 3– Information Display

Four displays are dedicated to the hours and minutes and four others alternately display month and day of the month then year. To minimize the wiring and because of the limited number of FPGA outputs, I used multiplexing to display information on the eight 7-segment displays.

The day of the week and CET/CEST (daylight saving time) are displayed on a series of 7 LEDs and another of 2 LEDs. As an example, Figure 3 shows how is displayed, August 13, 2023 at 18:39 (CEST in France).

Eight outputs of the Updiuno board, `seg0` to `seg7`, are used to control the 7 segments (a,b,c,d,e,f,g) and the decimal point (dp not currently used). The signals `seg0` to `seg7` are sent to all the displays but only the selected display will be lit. This selection is made by the outputs of the Updiuno board noted `cs0` to `cs7`. Seven outputs of the Updiuno `dow0` to `dow6` are used to control the day of the week LEDs (M,T,W,T,F,S,S) and two outputs noted CET and CEST allow the lighting of the daylight saving time LEDs. In addition, one LED is provided to inform on the correct detection of the DCF77 signal and three test points TP1 to TP3 are also available.

Note that there is no built-in RESET logic into the FPGA. It is up to the designer to provide this logic, which is done by the circuit U2 and the push button SW1. As for the effect of the RESET: it must be described in the VHDL code.

The power supply is very classic and consists of a 220/12v 100 mA transformer whose alternating output voltage is rectified by a 4-diode bridge. This full-wave rectification is then filtered by C1 and sent through diode D2 to the input of two 5v/100 mA regulators (U3, U12), the outputs of which undergo a final filtering by C3 and C6 respectively. If you won't integrate the Pico W simulator (described later in this paper) into Ouah!, then U2 and C6 can be omitted but a strap must be wired from +5V to +S5V instead. If the mains is temporarily disconnected, a 9V battery can take over. Its voltage is also applied to the input of the 5V regulator through diode D3, thus ensuring continuity of operation. The D2-D3 pair ensures that the two power sources do not flow into each other. This simple trick comes from the Sonelec Musique site of Remy Maillard [2], a veritable Ali Baba's cave and I could never thank its author enough for all that he brings to the electronics community.

The pinout of the Updiuno board is shown in Figure 4. It should be noted that the pin numbers (1 to 48) shown on the Ouah! board schematic (Figure 5) are those of the Updiuno board and not those of the FPGA. The correspondence between pins of the UpDuino board and pins of the FPGA is indicated in Figure 4 by the numbers located in the gray diamonds. It is also written on the printed circuit of the board. As an example, pin no. 25 of the Updiuno board (lower right corner in Figure 4) is connected to pin no. 2 of the FPGA.

Warning, this figure found on the Internet contains an error. Pins 42 and 41 should be reversed. Pin 42 is connected to GND; as for pin 41, it delivers the 12 MHz clock signal.

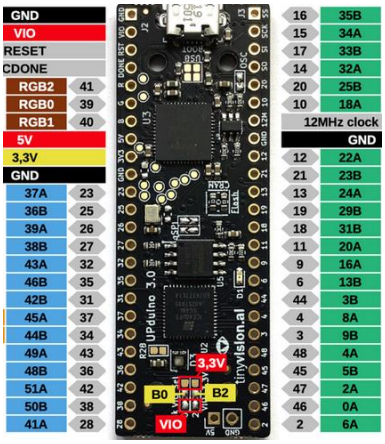


Figure 4- UpDuino board pinout

WARNING! DANGER

The OUAH! board contains an integrated power supply powered by 220 v.
Precautions for use are therefore necessary to avoid any electrocution. Please, see the recommendations made in section VIII of this paper.
It is the responsibility of readers wishing to carry out the Ouah! to take all the necessary precautions, the author can in no way be held responsible for the consequences of an electric shock.

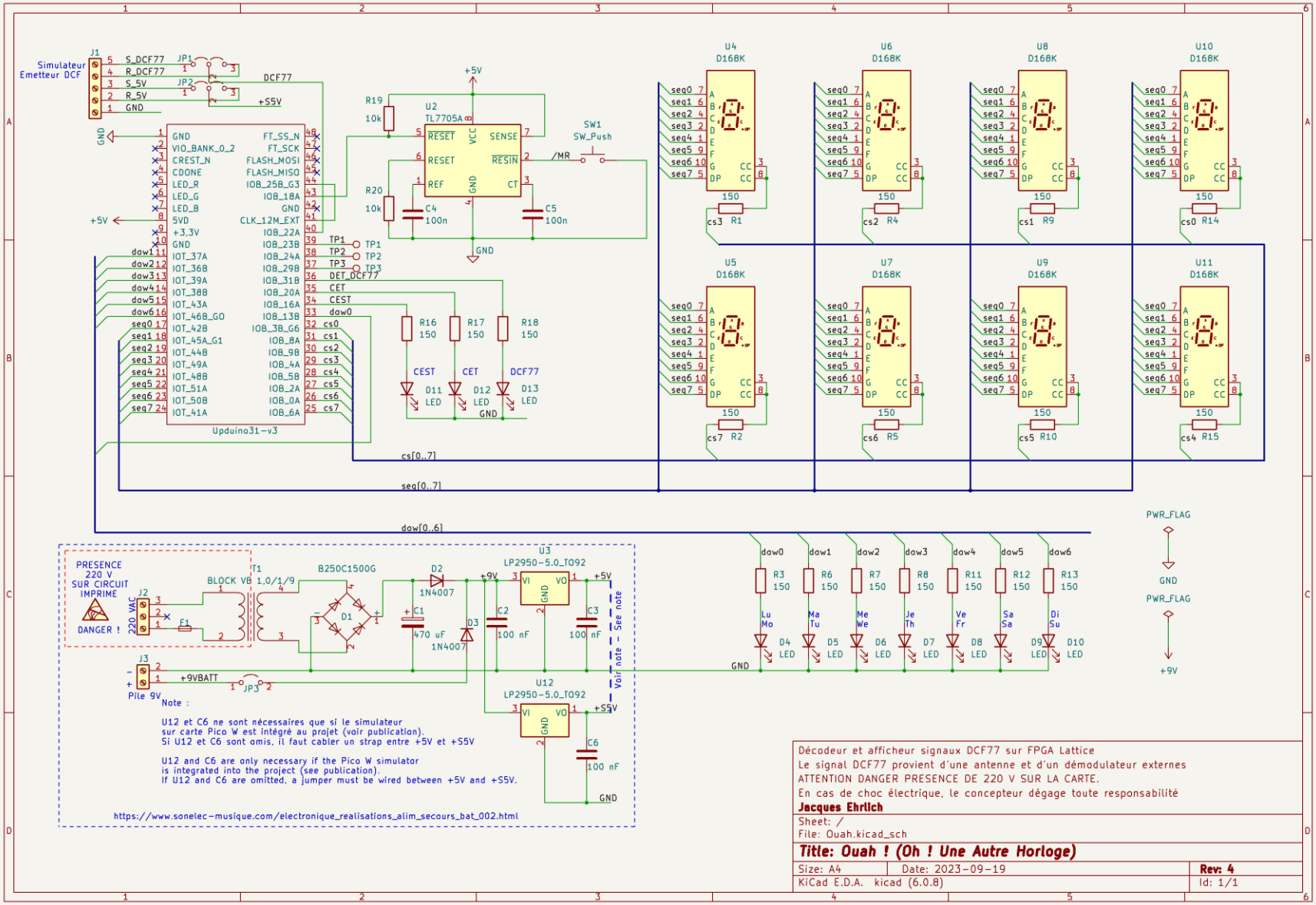


Figure 5– Ouah !

V. VHDL CODE

In VHDL there are two methods of modelling a system: structural and behavioural. The structural approach consists in describing a logic diagram using basic components chosen from a library or created by the designer (gates, registers, counters, memory, etc.). We finally describe the connectivity between these components a bit like we do when we describe a schematic for a simulation under SPICE. On the other hand, the behavioural approach consists in describing the system by its behaviour. This behaviour is described in a set of PROCESS that run in parallel. The behavioural description has some similarities to the description of an algorithm in a programming language, but programmers may be confused by the fact that the lines of VHDL code run in parallel. With my modest experience of VHDL, I am often surprised when comparing the result obtained to the expected result. Either way, this is due to my misunderstanding of parallel execution of VHDL instructions.

A. Let's give Caesar what's his (French expression).

For writing the VHDL code, I was inspired by a study report produced by students from the University of Paris 11 as part of a Master Engineer course [3], a very well-done report that I recommend reading.

I nevertheless extended this study by exploiting all (or almost) of the DCF77 frame which allows the display of hours, minutes, month, day of the month, day of the week, year, daylight saving time (CEST/CET). The display technique is also very different, by the introduction of multiplexing already mentioned above. Finally, I do not despair of improving reliability in a future version of the VHDL code, by introducing a parity check on the information contained in the DCF77 frames, which was not done in the study above mentioned.

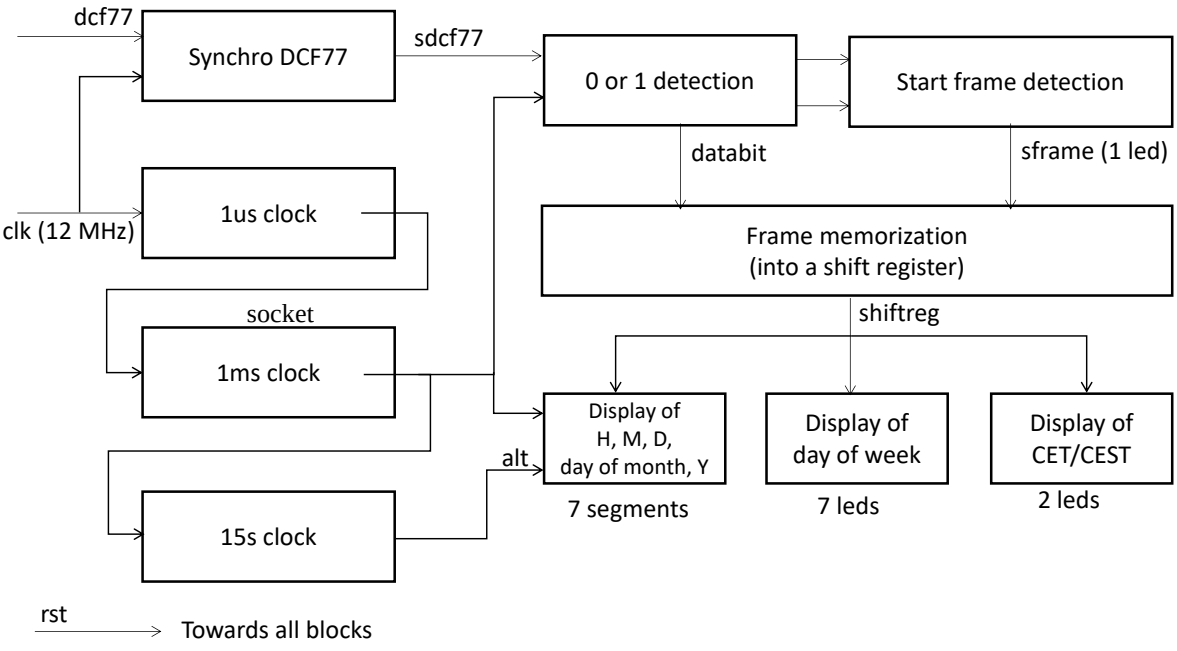
B. A brief overview of the code

The synoptic diagram below illustrates the operation of the system. From the 12 MHz clock integrated on the Upduino board, are generated various rectangular signals with a duty cycle of 1 and a period of 1 us (`MICROSEC process`), 1 ms (`MILLISEC process`) and 15 s.

In order to get a synchronous system, the `dcf77` signal coming from the receiver (or the simulator) is synchronized with the 1 us clock (`SYCDCF process`). The resulting signal `sdcf77` combined with the 1ms clock constitute the inputs of two blocks: one for the detection of the start of frame (`SFD process`) and the other for the detection of 0s and 1s (`BITDETECT1` and `BITDETECT2 processes`) which depend on the width of the frame pulses (100 ms for a 0 and 200 ms for a 1). As soon as the beginning of frame is detected, the content of the frame is stored in a shift register. Then it remains is to extract the required fields from this register and decode their BCD codes to calculate the various information to be displayed.

For the hours, minutes, month, day of the month and year, this is done in the `MULTIPLEX process`. This process decodes the fields extracted from the shift register and successively selects the eight 7-segment displays thanks to the `selector signal` which is incremented modulo 8 every ms. Thus a 7-segment display is lit for 1 ms every 8 ms. Thanks to the retinal persistence one obtains the impression of a simultaneous lighting of the 8 displays. As for the lighting of the day of the week and summer-winter diodes, this happens in the `DAYOFWEEK` and `CETCEST` processes.

Dear readers, I will not go further into the description of the VHDL code. After all, this project also has a self-learning objective and I leave it to your own reading, analysis and understanding of the code. I am at disposal to answer any questions you may ask on the Elektor Labs site on the page for this project. And if by chance, improvements can be proposed I will be very happy to know them and thus benefit from your experience.



VI. Implementation under Lattice Radiant

A. Installation

Installing Radiant on Windows is straightforward. You must go to the Lattice website [4] and follow the very clear instructions that are given. You will have to complete a license form by providing the MAC address of the computer on which the software will be installed. Warning: you must choose the *node locker license* unless you plan to install Radiant on a server (which then allows its use from several computers connected to the local network). A few minutes later you will receive by email a file that must be installed in the indicated directory. Then you can download the software for free and launch the installer by accepting all the options offered by default.

B. Use

When launching Radiant, it is proposed either to open an existing project or to create a new one. It will be this second option which will be chosen for a first use which launches an assistant (wizard*)* allowing to fill in the project: name, type of FPGA used etc. If you already have the VHDL source code, the wizard suggests that you include it in the project; otherwise, you can always do it later. The same goes for the constraints file which allows you to assign the inputs and outputs to the pins of the FPGA. You must then specify the type of FPGA used and the packaging. In our case it is the ICE40UP5K in SG48 package.

Radiant allows multiple implementations of the same project; this is very handy if you want to test different options without modifying an already working VHDL code. The first implementation is created automatically when the project is created.

Once the initialization phase is done, you must then provide the VHDL code if this has not been done with the wizard. You can edit it in Radiant or import the existing code, which we'll do. In the left window you must select *Input File* under *impl_1* and right click then *Add* then *Existing File* and search on your computer for the most recent version of the `ouah.vhd` file that you will have recovered from the Elektor Labs site or from Github . Then, all that remains to do is to compile this code, which is done by clicking on the green arrow in the upper banner. Four phases are then linked automatically if there is no error: *Synthesize Design, Map Design, Place and Route Design* and *Export Files.* On the other hand, in the event of an error, the process is interrupted in the phase where the error was encountered. It is then necessary to consult the error messages in the *Report* tab then make the required corrections and restart the compilation. If you use the file available for download, the compilation will be done from start to finish without error.

With the compilation complete, Radiant randomly assigned the inputs and outputs to the FPGA pins. This assignment must therefore be modified to make it compatible with the layout of the OUAH! board. There are two possibilities for this: use the constraints editor (Menu *Tools* then *Device Constraint Editor*) or edit the constraints file (.pdc file): any modification by the constraints editor affects the constraints and conversely any modification of the constraints file will be visible under the constraints editor. I opted for editing the constraints file. We will therefore force Radiant to generate the constraints file. To do this, call the constraints editor and in the left column of the editor click on the *Back Annotate Assignment icon*. Then click on the disk icon (upper banner) to save a constraints fileand close the constraints editor. The constraints file is now accessible under the menu *Post Synthetisys Constraint File* and by double clicking on the name of the file it can be opened to modify it. To avoid this tedious work, you can simply import the constraints file available on Elktor Labs site or Github [5]. Then, you must restart the compilation process and finally you will get a *bitstream file* ready to be downloaded to the Upduino.

C. Programming the FPGA

The FPGA programmer is automatically installed along with Radiant. It can be called from Radiant or as a standalone program. Before launching the programmer, you must connect the Upduino3.1 board to the computer with a simple USB cable (as short as possible).

Launch the programmer (*Tools* menu then *Programmer)* and click on the *Detect Cable* button and check that the message *Board with FTDI-USB Host Chip Detected* is displayed.

Then, in the upper left window of the programmer double click in the *Operation* menu which opens a new window. In the *Target Memory* field select *External SPI Flash Memory* and then check that the setting is as shown in Figure 6 and finally click OK.

To start programming, you must then go to the *Run* menu and select *Program Device* and after a few seconds the FPGA of the UpDuino3.1 board is programmed.

D. Creation of new implementations

As indicated in the introduction to this section, the Radiant software allows the creation of an infinite number of implementations. I suggest using this possibility whenever you want to evolve the VHDL code while preserving previous versions. Then, in case of a bug, it will always be possible to return to an earlier version which works well.

It should be noted that the FPGA programmer will not automatically look for the code of the implementation in use. You will have to select the corresponding `.bin` file in the FPGA programmer settings window (Figure 6.)

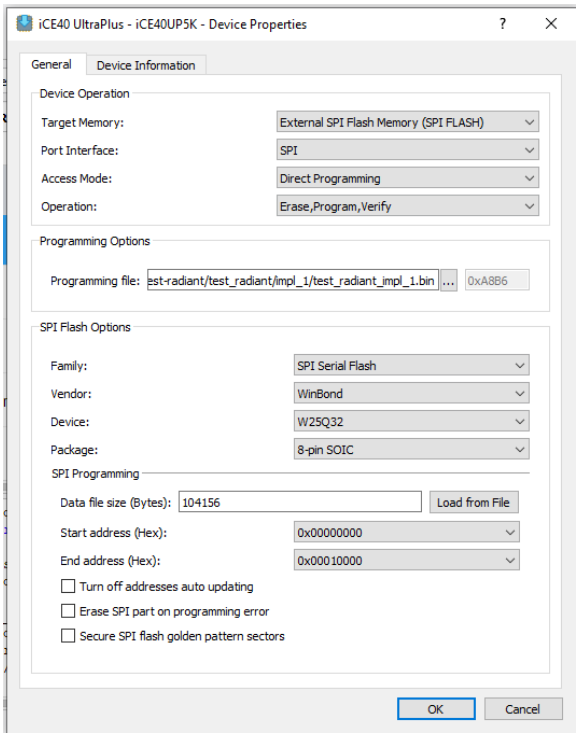


Figure 6– FPGA programmer Setup

VII. Realization of the OUAH! board

After modelling on a Verboard prototyping card, I designed the printed circuit using Kicad. My first idea was to make a card in the same format as the Cloc 2 project published in Elektor in March/April 2023, then to integrate the board into the same box model. But I quickly realized that it was "mission impossible" because of the size of the Updiuno board on the one hand and the integrated power supply on the other hand. One option would have been to use an external power supply such as a mobile phone charger. But the tests carried out in the prototyping phase showed that this type of switching power supply prevented any reception of the DCF77 signal due to the electromagnetic emission of these chargers.

So, I had to resolve to increase the dimensions of the board while keeping the objective of housing it in the same type of case as that of the Cloc 2.0 project, but of larger dimensions. This explains why the shape of the board is very similar to that of the Cloc 2.0 project.

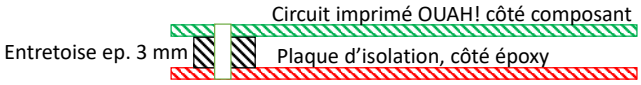
All manufacturing files are available on the Elektor Lab site or on Github [5] and are ready to be sent to your favourite PCB manufacturer.

Regarding the wiring, no difficulty since all the components are THT. It is imperative to put on sockets F1, U1 to U11 and the Updiuno board.

VIII. Prevention of electric shocks

After having completely wired the OUAH card and before any power-up, it is advisable to add an insulation board which will also enhance electromagnetic compatibility This is cut out of a one side copper epoxy plate, thickness 16/10.

This card will be cut to the exact same outline and mounting holes as the OUAH! Board and will be fixed against it epoxy side against copper side and separated by four 3 mm thick spacers. See sectional view in Picture 7.



Picture 7 – Attachment of an insulating counter plate

The two plates will be held together by Teflon screws and nuts with a diameter of 3 mm. It is then possible to handle the OUAH! board without risk of electric shock by contact with the welds on the copper side. However, on the component side, it remains a possibility of electric shock by contact with the screws of connector receiving the mains cable (Figure 9). It must be protected by insulating tape or any other means.

WARNING! DANGER

The OUAH! board contains an integrated power supply powered by 220 v. therefore precautions for use are necessary to avoid any electrocution. The above-mentioned recommendation should be applied.
It is the responsibility of readers wishing to carry out the OUAH! to take all the necessary precautions, the author can in no way be held responsible for the consequences of an electric shock.

IX. Connection and preliminary tests

For the preliminary tests, I recommend using the DCF77 simulator described later in this article. You must connect it to the 5-pole connector located at the bottom right in Figure 9. We will use the GND and S5V poles for its power supply and the S77 pole for the DCF77 signal. It will also be necessary to position the two jumpers JP1 and JP2 in position S. You can then connect the 220V cable to the 3-pole connector located at the bottom left in Figure 9 and then plug it into a mains socket. After pressing the RESET button, 0s should be displayed on the eight 7-segment displays. After about one minute or two, time and date should be displayed correctly.

Once this step has been successfully completed, the DCF77 receiver can then be connected. It must be connected to the 5-pole connector located at the bottom right in Figure 9. We will use the GND and R5V poles for its power supply and the R77 pole for the DCF77 signal. It will also be necessary to position

the two jumpers JP1 and JP2 in position R. If the reception is correct, after one or two minutes, time and date should be displayed correctly. Otherwise, it will be necessary to find – sometimes with some difficulties – the optimum position of the ferrite antenna to obtain proper operation.

If a battery is used to ensure continuity of operation in the event of a mains failure, it must be connected to the 2-pole connector, with respect to the polarities as shown in Figure 9. Do not forget to place a jumper on JP3.

X. INTEGRATION INTO A BOX

The card OUAH! has been integrated in a Hammond 1591T enclosure in Polycarbonate, crystal, 150 x 46 x 80 mm.

Figure 8 is a side view showing the card and its insulating counter plate, all mounted on 15 mm spacers fixed to the bottom of the box. These spacers make it possible to keep a free space to put the battery, the simulator and the DCF77 receiver. Unfortunately, the tests carried out up to now have never made it possible to receive the DCF77 signal when the antenna is placed in the box. So, I recommend integrating only the simulator and the battery. As for the receiver, it will be placed outside the box. According to the interest aroused by this project, I will publish later the solution that I will have finally adopted. I am also interested in your own achievements.

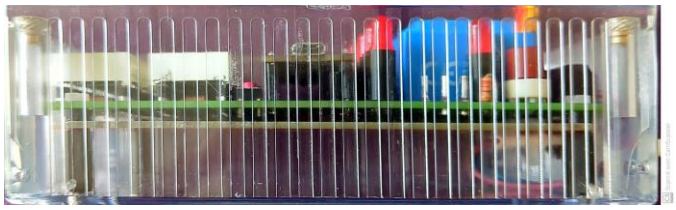


Figure 8– integration in the enclosure: side view
(we can see the isolation card located under the OUAH! board)

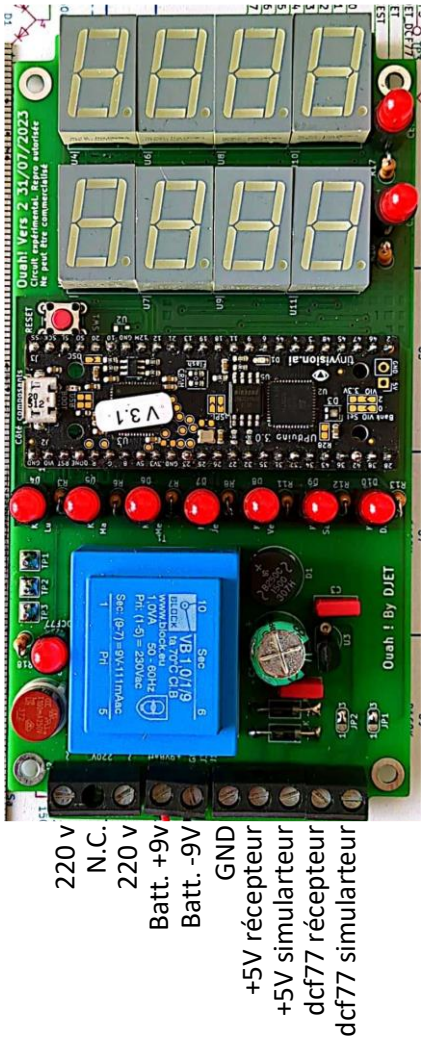


Figure 9– Top view of the OUAH!

XI. THE SIMULATOR

As indicated, it is essential to have a DCF77 signal simulator to develop a DCF77 clock.

A. The simulator for Raspberry Pi

For the sake of simplicity, a first version was developed in C on Raspberry Pi under the Pi OS environment (a Pi Zero W is quite sufficient). The source code is available in the `rpi_dcf77sim.c file` on Elektor Labs and on Github [5]

In the sequel, we assume that the Rpi is powered by its USB socket, connected to your local network via Wifi or Ethernet and set to the local date and time (in France, UTC + 1 or UTC +2 depending on CET/CEST

From a PC under Windows or a system under Linux, it will be necessary to establish an SSH connection with the Rpi and create a dedicated directory `rpi_dcfsim77` then install the source code in this directory, compile and then execute it with the following commands:

```
$cd rpi_dcf77sim
$gcc rpi_dcf77sim.c -o rpi_dcf77sim -lwiringPi
$./rpi_dcf77sim
```

Then the simulator displays the current date and time on the console, then the dcf77 translation in the form of a series of zeros and 1s. Simply connect the board OUAH! to the GND pin and to the GPIO22 pin of the Rpi to get the DCF77 signal.

B. *The simulator for Pico W*

The version of the simulator described above, if it is well suited in the development phase, is not suitable for continuous use and integration in the enclosure of the OAUH board! because of its excessive size and consumption. This is what led me to port the C code to the Pico W board. It's a relatively substantial piece of work that could justify a specific article. However, it is possible for readers who already have experience developing in C under Visual Studio Code (VSCode), to compile the source code provided after modifying it to consider the SSID and PASSWORD of their own Wifi network. It should be noted that this code makes use of codes developed by the Raspberry foundation: these are the codes appearing in the examples provided with VSCode for Pico and PicoW: `ntp_client and rtc.`

Configuring a project under Visual Stufio Code is far from being trivial. To simplify this task, I have chosen to integrate the `simdcf77 project` into the example projects provided with Visual Studio Code. This allows to benefit from correctly configured `CMakeLists.txt` files. Here's how to do it on a Windows PC.

1. Install VSCode (see the tutorials on the Rpi foundation site).
2. Launch VSCode and open the examples folder then click on the last `MakeLists.txt file` (located after the `.gitignore file` and add the following lines just after the 1st line :
   ```
   set(PICO_BOARD pico_w)
   set(WIFI_SSID "your_ssid")
   set(WIFI_PASSWORD "your_password")
   ```
   Save and wait for VSCode to rebuild the configuration.
3. Locate the `build` directory on your hard disk and open this directory (Figure 10-a).
4. Open the subdirectories `pico_w` then `wifi` and copy the `simdcf77` folder that you will have retrieved from the Elektor Labs site or from the Github [5] (Figure 10-b).
5. In VSCode open the `pico_w` sub-folder then the wifi folder then the `simdcf77` folder.
6. Open `picow_symdcf77.c` file, edit the lines containing
   ```
   #define WIFI_SSID "xxxxx"
   #define WIFI_PASSWORD "xxxx"
   ```
   by replacing the XXXX with your SSID and PASSWORD. Then save the modified version (Figure 10-c).
7. Then click on the CMake icon on the left vertical action bar.
8. Again, open the folders and sub-folders to reach `symdcf77` and launch the compilation of the project by clicking on the icon located to the right of the `line pico_symdcf77_backgound` (Figure 10-d).
9. With the Windows file explorer, open `build->pico_w->wifi->symdcf77` where you will find the file: `picow_symdcf77_background.uf2` to be installed by a simple drag and drop onto the PicoW card according to the procedure described in the Pico or Pico W tutorials available on the Rpi Foundation website (Figure 10-e).
10. After a few moments, the connection of the PicoW card to your WiFi network should be done, then you will see blinking the green LED on board the Pico W.
11. Now, the simulator is ready to be connected to the OUAH! Board according to their respective pinout (Figure 4 and Figure 9):
    a. GND of the PicoW to GND of the 5-pole connector of OUAH!
    b. VSYS (pin 39) of the PicoW to S5V pole of the 5-pole connector of OUAH!
    c. GPIO22 (pin 29) of the PicoW to the S77 pole of the 5-pole connector of OUAH!

XII. Conclusion and Future Work

The proper functioning of a DCF77 clock remains highly dependent on the reception quality of the DCF signal. This is quite mediocre in France with the receivers found at low cost. Indoor operation is almost impossible. Outdoors, it is necessary to find the correct orientation of the antenna and to be in an environment not subject to electromagnetic pollution. My next work will focus on improving reception quality by adding a selective antenna preamplifier on the 77.5 kHz frequency.

Improvements to the VHDL code are also planned to consider the parity bits transmitted in the DCF77 frame.

Any suggestions on improvements that can be made to this project are welcome.

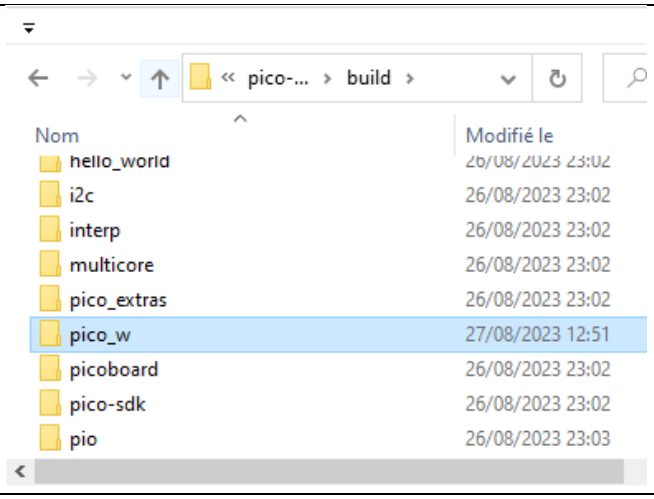
XIII. References

[1] https://fr.wikipedia.org/wiki/DCF77
[2] https://www.sonelec-musique.com/
[3] http://fimcachan.free.fr/vhdl/horloge2003.pdf
[4] https://www.latticesemi.com/LatticeRadiant?pr031521
[5] https://github.com/duodiscus92/projet-ouah/tree/master

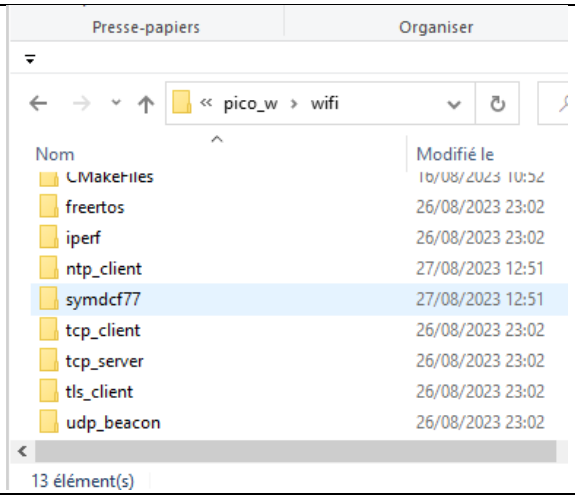
XIV. Biography

Dr. Jacques Ehrlich is Research Director Emeritus at Gustave Eiffel University. Until 2014, he directed the LIVIC, a research laboratory on advanced driving assistance systems (ADAS) and connected autonomous vehicles (CAV). He coordinated for eight years (2012-2019) an international group of experts on road network operation and Intelligent Transport Systems (ITS) at the World Road Association (www.piarc.org). He is currently a freelance consultant in the field of ITS and CAV (www.djet.fr)

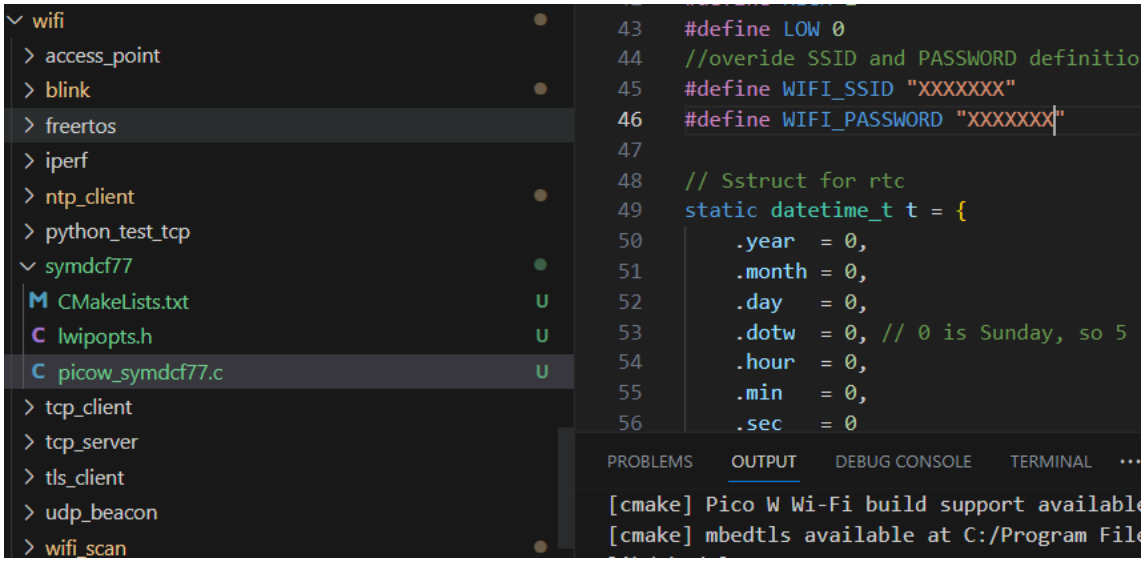
He is passionate about electronics, computer science and glider flight, all disciplines that he practices as hobbies and retired leisure activities.



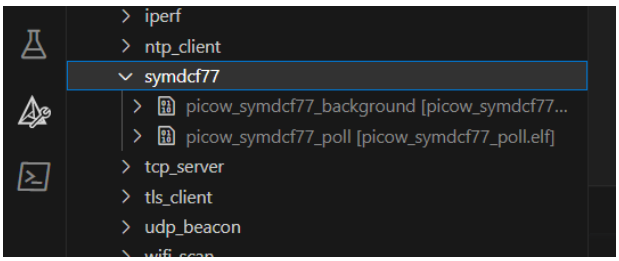
a) Contents of the build directory



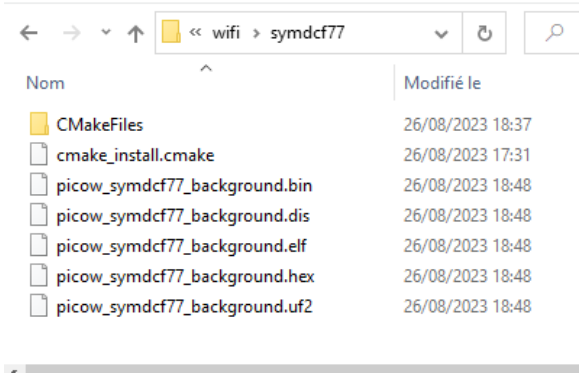
b) Installation of the symdcf77 directory



c) In VSCode, modifying the picow_symdcf77 file



d) Launching the project compilation



e) uf2 file to drag and drop into Pico W

Figure 10– Steps in creating the picow_symdcf77 project

XV. ANNEX 1 LIST OF COMPONENTS

Quantities	References)	Value	Designation	Comments
1	C1	470uF	Polarized capacitor	
4	C2, C3	100nF	Capacitors	
1	D1	B250C1500G	diode bridge	
2	D2, D3	1N4007	Diode	
10	D4, D5, D6, D7, D8, D9, D10, D11, D12, D13	LEDs	red LED	
1	F1	LITT 37201000411	Delayed 100 mA radial fuse	To be mounted on a socket
1	D1	Screw_Terminal_01x05	5 pole screw terminal block	
1	J2	Screw_Terminal_01x03	3-pole screw terminal block	
1	D3	Screw_Terminal_01x02	2-pole screw terminal block	
2	JP1, JP2	Jumper_3_Open	Support for 3 pole jumper	
1	JP3	Jumper_2_Open	Support for 2 pole jumper	
8	R1, R2, R4, R5, R9, R10, R14, R15	150	Resistance	
10	R3, R6, R7, R8, R11, R12, R13, R16, R17, R18	150	Resistance	
2	R19, R20	10k	Resistance	
1	SW1	SW_Push	push button	
1	T1	BLOCK VB 1.0/1/9	220V/9V transformer	
3	TP1, TP2, TP3	Test Point	test point	
1	U1	Upduino31-v3	Upduino 3.1 Board	To be mounted on a socket
1	U2	TL7705A	Reset generator TL7705A	To be mounted on a socket
1	U3	LP2950-5.0_TO92	5V regulator	To be mounted on a socket
8	U4, U5, U6, U7, U8, U9, U10, U11	D168K	7-segment displays	To be mounted on a socket
1	Polycarbonate Crystal case	Hammond 1591T	Enclosure 150 x 46 x 80mm IP54	
1	DCF77 Receiver	Kundo XT 950 051	DCF77 Receiver	
1	Optional	Raspberry Pico W	DCF77 simulator	See note below

Note :