Projet OUAH! Une horloge DCF77 sur FPGA
accompagnée de son simulateur sur Pico W (écrit en C)

Développement en VHDL sous Radiant de Lattice

Jacques Ehrlich

I. Resume

Dans cet article on décrit une horloge DCF77 réalisée 100% par le matériel (hardware). Elle est implantée sur un FPGA et décrite en VHDL. Cette réalisation n'inclut pas le récepteur et le démodulateur du signal DCF77. Elle concerne exclusivement le décodage du signal disponible en sortie du démodulateur et l'affichage sur des 7 segments et sur des leds des heures, minutes, mois, jour du mois, année, jour de la semaine et horaire d'été ou d'hiver. Un circuit imprimé est également proposé. On peut se procurer facilement des récepteurs-démodulateurs DCF77 sur le marché. Mais en France, la réception des signaux DCF77 émis depuis l'Allemagne est difficile du fait de la faible puissance reçue. Elle se fait bien *outdoor* mais s'avère quasiment impossible *indoor*. Aussi pour la mise au point de Ouah ! ai-je-eu besoin d'un simulateur du signal DCF. Ce simulateur, écrit en C, implanté sur Raspberry Pico W peut être intégré dans le même boitier que la carte Ouah ! Ainsi l'horloge peut être utilisée *outdoor* avec le récepteur DCF77 ou *indoor* avec le simulateur à condition que la Pico W soit dans la portée d'un réseau Wifi.

II. Introduction

OUAH ! est l'acronyme ironique de **Oh Une Autre Horloge !**

En effet dans la littérature et les colonnes d Elektor notamment, les projets d'horloges fondées sur le décodage du signal DCF77 ne manquent pas.

Mais je rêvais de développer une version 100% matérielle fondée sur l'utilisation d'un FPGA afin de poursuivre mon autoformation sur cette technologie, autoformation que j'avais commencé avec le projet GONOGO publié également sur le site d'Elektor Lab. En fait il d'agit plutôt d'une reformation car j'avais déjà utilisé professionnellement les FPGA dans les années 80 lors de la mise sur le marché des premiers circuits XILINX. Quarante ans plus tard, la pénurie de semiconducteurs étant passée par là, il était devenu impossible de trouver des FPGA à prix et délais raisonnables. A cela s'ajoutait qu'un FPGA monté sur une carte et entouré d'un certain nombre de composants sont autant de composants SMD délicats à monter qui nécessitent des moyens et une technique que je ne possède pas.

Mais grâce à l'article de Mathias Claussen paru dans Elektor n°494, j'ai découvert la carte Upduino3.1 et c'est ainsi que le rêve est devenu réalité. En effet, cette carte présente plusieurs avantages ; bas coût (autour de 40 € TTC au moment où j'écris ces lignes), faible encombrement, faible consommation, très complète, programmable par un simple câble USB et puissance tout à fait honorable puisqu'équipée d'un FPGA Lattice ICE40UP5K doté de pas moins de 5280 cellules ce qui est amplement suffisant pour ce projet.

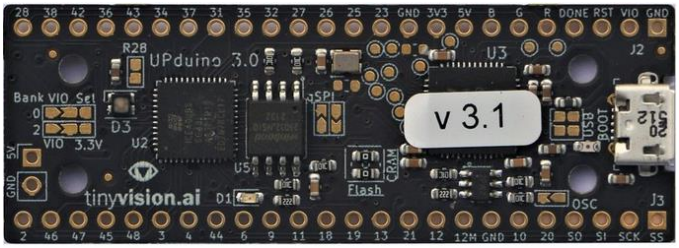


Figure 1 – La carte Upduino3.1

La mise en œuvre des FPGA ne coule pas de source. Outre qu'il est bien sur requis de connaître un langage HDL, elle nécessite la maitrise d'outils de développement dont certains peuvent s'avérer complexes. Je me suis concentré sur deux d'entre eux : APIO sous Linux qui supporte exclusivement une programmation en Verilog et Radiant de Lattice sous Windows qui supporte aussi bien la programmation Verilog que VHDL. Ces deux outils sont bien documentés et finalement assez simples à mettre en œuvre.

Ainsi, cet article a pour objet non seulement de décrire une implémentation de OUAH! fondée sur l'Upduino3.1 mais aussi les modalités de mise en œuvre à l'aide de l'outil Radiant. Finalement une implantation complète sur circuit imprimé ne faisant usage que de composants traversants est proposée.

III. Le signal DCF77

On trouve sur internet de nombreux sites décrivant le contenu de la trame DCF77. On pourra notamment consulter Wikipedia [1]. Les informations sont codées en BCD dans des trames émises toutes les minutes à raison d'une impulsion par seconde. La durée des impulsions est de 100 ms pour la transmission d'un 0 et 200 ms pour la transmission d'un 1. Il faut 59 secondes pour transmettre la totalité des informations requises. Il n'y a pas d'impulsion entre la 59ème impulsion de la trame N et la 1ère impulsion de la trame N+1 : en d'autres termes, la 60èmeimpulsion est omise ce qui laisse deux secondes de séparation entre deux trames. C'est ce qui permet la synchronisation du décodeur. La Figure 2 décrit le codage des différentes informations contenues dans une trame. A noter que la trame émise à l'instant t indique l'heure de l'instant 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 60^{e} impulsion supplémentaire. Le silence est reporté à la 61^{e} et dernière seconde. Puis, débute l'heure suivante.)	:39	8		:59	0	Marque de la minute : aucune modulation.

Figure 2 – Codage d'une trame DCF 77 – (source Wikipedia)

IV. LA CARTE OUAH !

La Figure 5 décrit le schéma de la carte Ouah ! Il est relativement simple puisque tout se situe dans le code VHDL du FPGA. On distingue principalement trois parties : la carte Updiuno, l'affichage par afficheurs 7 segment et leds et l'alimentation secteur pouvant facultativement être sauvegardée par une pile 9V.

Les heures, minutes, mois, jour du mois et année sont affichés sur 8 afficheurs 7 segments.

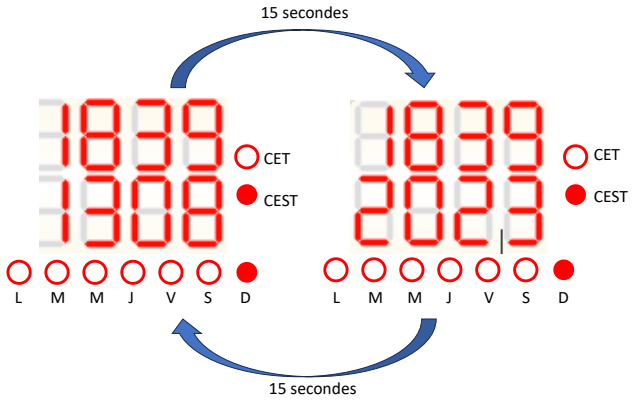


Figure 3 – Affichage des informations

Quatre afficheurs sont dédiés aux heures et minutes et quatre autres afficheurs affichent en alternance mois et jour du mois puis année. Afin de minimiser le câblage et à cause du nombre limité de sorties du FPGA, j'ai eu recours au multiplexage pour l'affichage sur afficheurs 7 segments.

Le jour de la semaine et l'heure d'été-hiver sont affichés sur une série de 7 leds et une autre de 2 leds. La Figure 3 illustre l'affichage obtenu à 18h39 le dimanche 13 août 2023 (heure d'été).

Huit sorties de la carte Updiuno, seg0 à seg7, sont utilisées pour contrôler les 7 segments (a,b,c,d,e,f,g) et le point décimal (dp non utilisé actuellement). Les signaux seg0 à seg7 sont envoyés sur tous les afficheurs mais seul l'afficheur sélectionné sera allumé. Cette sélection se fait par les sorties de la carte Updiuno notées cs0 à cs7. Sept sorties de la carte Updiuno dow0 à dow6 servent à contrôler les leds jours de la semaine (L,M,M,J,V,S,D) et deux sorties notées CET et CEST permettent l'allumage des leds heure d'été/hiver. Une led a été prévue pour signaler la bonne détection du signal DCF77 et trois points de test TP1 à TP3 sont également disponibles.

Il convient de noter qu'il n'existe pas de logique de RESET intégrée au FPGA. C'est au concepteur de prévoir cette logique ce qui est fait par le circuit U2 et le bouton poussoir SW1. Quant à l'effet du RESET c'est aussi au concepteur de le décrire dans son code VHDL

L'alimentation est très classique et consiste en un transformateur 220/12v 100 mA dont la tension alternative de sortie est redressée par un pont à 4 diodes. Ce redressement double alternance est ensuite filtré par C1 et envoyé à travers la diode D2 aux entrées de deux régulateurs 5v/100 mA, U3 et U12, dont les sorties subissent un dernier filtrage par C3 et C6. U12 et C6 ne sont utiles que si le simulateur sur carte PicoW est intégré au projet. Dans le cas contraire, ils seront omis mais il faudra câbler un strap entre +5V et +S5V. Au cas où l'on débranche momentanément le secteur, une pile 9V peut prendre le relais. Sa tension est également appliquée à l'entrée du régulateur 5V à travers la diode D3, permettant ainsi d'assurer la continuité du fonctionnement. Le couple D2-D3 garantit que les deux sources d'alimentation ne débitent pas l'une dans l'autre. Cette astuce simple provient du site Sonelec Musique de Remy Maillard [2], véritable caverne d'Ali Baba et je ne remercierais jamais assez son auteur pour tout ce qu'il apporte à la communauté des électroniciens.

Le brochage de la carte Updiuno est indiqué sur la Figure 4. Il convient de noter que les numéros de broches (1 à 48) indiqués sur le schéma de la carte Ouah (Figure 5) sont celles de la carte Updiuno et non celles du FPGA. La correspondance entre broches de la carte UpDuino et broches du FPGA est indiquée sur la Figure 4 par les chiffres situés dans les losanges gris. Elle est également inscrite sur le circuit imprimé de la carte Updiuno. A titre d'exemple, la broche n° 25 de la carte Updiuno (coin inférieur droit sur la Figure 4) est reliée à la broche n°2 du FPGA.

Attention, cette figure que l'on trouve sur Internet contient une erreur. Il convient d'inverser les broches 42 et 41. La broche 42 est relié au GND ; quant à la broche 41 elle délivre le signal d'horloge 12 MHz.

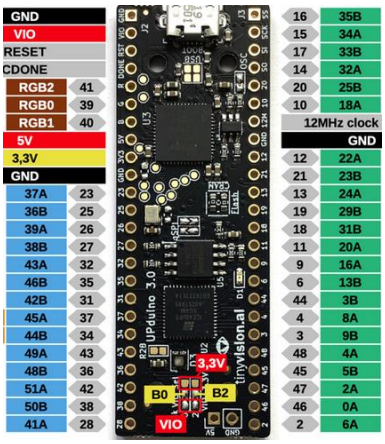


Figure 4– Brochage de la carte UpDuino

ATTENTION ! DANGER

La carte Ouah contient une alimentation intégrée alimentée en 220 v. Par conséquent des précautions d'utilisation s'imposent pour éviter toute électrocution. Voir à ce sujet les recommandations faites dans la section VIII de cet article

Il est de la responsabilité des lecteurs souhaitant réaliser le projet Ouah ! de prendre toutes les précautions nécessaires, l'auteur ne pouvant en aucun cas être tenu pour responsable des conséquences d'un choc électrique.

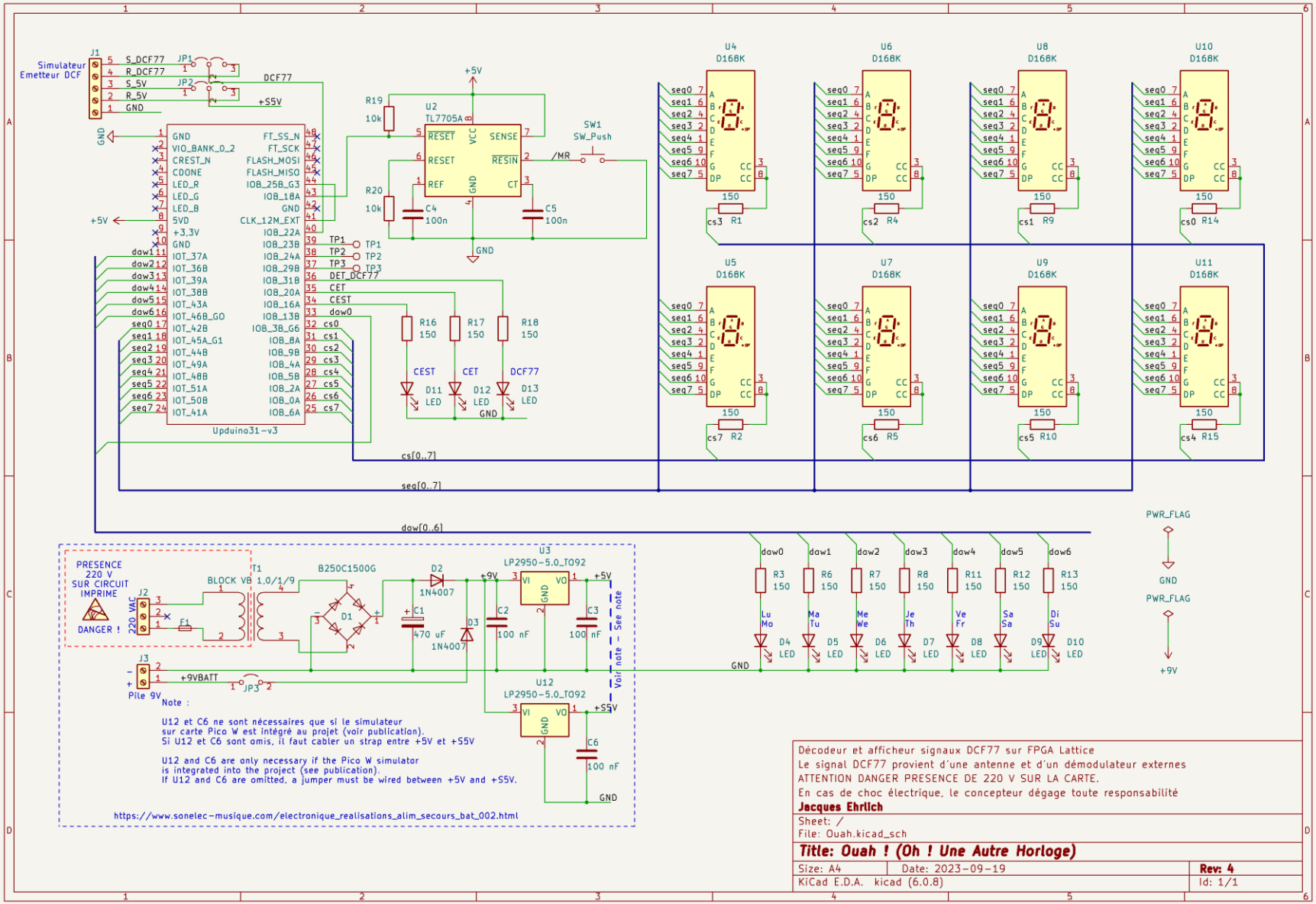


Figure 5 – Schéma de la carte Ouah!

V. LE CODE VHDL

En VHDL il existe deux méthodes de modélisation d'un système : structurelle et comportementale. L'approche structurelle consiste à décrite un schéma logique à partir de composants de base choisis dans une bibliothèque ou créés par le concepteur (portes, registres, compteurs, mémoire etc). On décrit finalement la connectivité entre ces composants un peu comme on le fait quand on décrit un schéma en vue d'une simulation sous SPICE. En revanche, l'approche comportementale consiste à décrire le système par son comportement. Ce comportement est décrit dans un ensemble de PROCESS qui s'exécutent en parallèle. La description comportementale possède quelques similitudes avec la description d'un algorithme dans un langage de programmation mais les programmeurs pourront être déroutés par le fait que les lignes de codes VHDL s'exécutent en parallèle. Avec ma modeste expérience du VHDL, je suis moi-même souvent surpris quand je compare le résultat obtenu au résultat attendu. Dans tous les cas, cela est dû à une mauvaise compréhension de ma part de l'exécution parallèle des instructions VHDL.

A. Rendons à César ce qui lui appartient.

Pour l'écriture du code VHDL, je me suis inspiré d'un rapport d'étude réalisé par des étudiants de l'université Paris 11 dans le cadre d'une formation d'Ingénieur Maître [3], rapport fort bien fait que je vous invite à lire.

J'ai néanmoins étendu cette étude en exploitant la totalité (ou presque) de la trame DCF77 ce qui permet l'affichage des heures, minutes, mois, jour du mois, jour de la semaine, année, horaire d'été (CEST) et horaire d'hiver (CET). La technique d'affichage est également très différente, notamment par l'introduction du multiplexage déjà évoqué plus haut. Enfin je ne désespère pas d'améliorer la fiabilité dans une prochaine version du code VHDL, en introduisant un contrôle de parité sur les informations contenues dans les trames DCF77, ce qui n'a pas été fait dans l'étude citée ci-dessus.

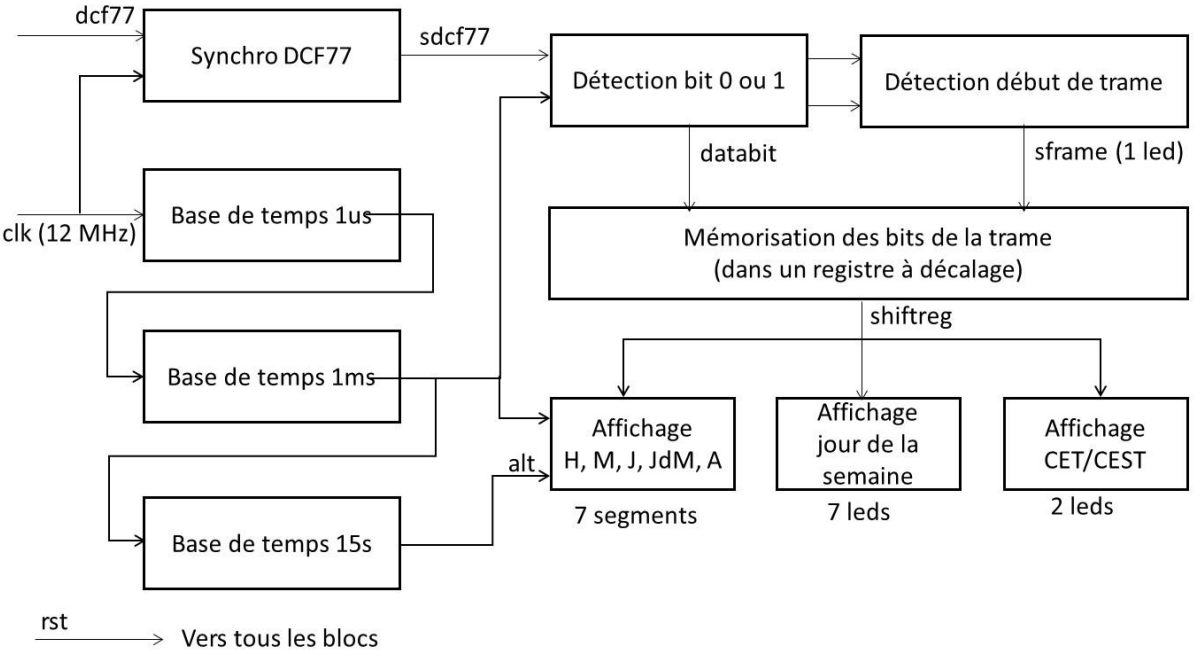
B. Un bref survol du code

Le schéma synoptique ci-dessous illustre le fonctionnement du système. A partir de l'horloge 12 MHz intégrée sur la carte Upduino, sont générées des signaux rectangulaires de rapport cyclique 1 et de période 1 us (process `MICROSEC`), 1 ms (process `MILLISEC`) et 15 s.

Afin d'avoir un système synchrone, le signal `dcf77` sortant du récepteur (ou du simulateur) est synchronisé avec l'horloge 1 us (process `SYNCDCF`). Le signal résultant `sdcf77` combiné avec l'horloge 1ms constituent les entrées de deux blocs : l'un pour la détection de début de trame (process `SFD`) et l'autre pour la détection des 0 et des 1 (process `BITDETECT1` et `BITDETECT2`) qui dépendent de la largeur des impulsions de trames (100 ms pour un 0 et 200 ms pour un 1). Dès que le début de trame est détecté, le contenu de la trame est mémorisé dans un registre à décalage. Il ne reste plus qu'à extraire dans ce registre les champs requis et décoder les codes BCD pour calculer les différentes informations à afficher.

Pour les heures, minutes, mois, jour du mois est année, cela est réalisé dans le process `MULTIPLEX`. Ce process décode les bits extraits du registre à décalage et sélectionne successivement les 8 afficheurs 7 segments grâce au signal `selector` qui s'incrément modulo 8 toutes les ms. Ainsi un afficheur 7 segment est allumé pendant 1 ms toutes les 8 ms. Grâce à la persistance rétinienne on obtient l'impression d'un allumage simultané des 8 afficheurs. Quant à l'allumage des diodes jours de la semaine et été-hiver, cela se passez dans les process `DAYOFWEEK` et `CETCEST`.

Chers lecteurs, je n'entrerai pas davantage dans la description du code VHDL. Après tout, ce projet a aussi un



objectif d'autoapprentissage et je vous laisse à votre propre lecture, analyse et compréhension du code. Je pourrai répondre aux questions que vous pourrez poser sur le site Elektor Lab à la page de ce projet. Et si d'aventure des améliorations peuvent être proposées je serai très heureux de les connaitre et de bénéficier ainsi de votre expérience.

VI. Mise en œuvre sous Lattice Radiant

A. Installation

L'installation de Radiant sous Windows ne pose aucune difficulté. Il faut aller sur le site de Lattice [4] et suivre les indications très claires qui sont données. Vous aurez à remplir un formulaire de licence en fournissant l'adresse MAC de l'ordinateur sur lequel sera installé le logiciel. Attention : il faut choisir la *licence node locker* sauf si vous prévoyez d'installer Radiant sur un serveur (ce qui permet ensuite son utilisation à partir de plusieurs ordinateurs connectés sur le réseau local). Quelques minutes plus tard on reçoit par mail un fichier qu'il faut installer dans le répertoire indiqué. Ensuite on télécharge gratuitement le logiciel et on lance l'installeur en acceptant toutes les options proposées par défaut.

B. Utilisation

Au lancement de Radiant, il est proposé soit d'ouvrir un projet existant, soit d'en créer un nouveau. Ce sera cette seconde option qui sera choisie pour une première utilisation ce qui lance un assistant (w*izard)* permettant de renseigner le projet : nom, type de FPGA utilisé etc. Si vous possédez déjà le code source VHDL, l'assistant vous propose de l'inclure dans le projet sinon, il sera toujours possible de le faire plus tard. Il en va de même pour le fichier de contraintes qui permet d'affecter les entrées et sorties aux broches du FPGA. Vous devez ensuite préciser le type de FPGA utilisé et le packaging. Dans notre cas il s'agit du ICE40UP5K en package SG48.

Radiant permet plusieurs implémentations d'un même projet ; c'est très pratique si vous voulez tester différentes options sans modifier un code VHDL déjà opérationnel. La première implémentation est créée automatiquement lors de la création du projet.

Cette phase d'initialisation étant passée, il faut ensuite fournir le code VHDL si cela n'a pas été fait avec l'assistant. Vous pouvez l'éditer dans Radiant ou bien importer le code existant, ce que nous allons faire. Dans la fenêtre de gauche il faut sélectionner *Input File* sous *impl_1* et faire un clic droit puis *Add* puis *Existing File* et chercher sur votre ordinateur la plus récente version du fichier `ouah.vhd` que vous aurez récupéré sur le site d'Elektor Lab ou sur Github [5]. Ceci fait, il ne reste plus qu'à compiler ce code ce qui se fait en cliquant sur la flèche verte du bandeau supérieur. Quatre phases s'enchaînent alors automatiquement s'il n'y a pas d'erreur : *Synthesize Design, Map Design, Place and Route Design* et *Export Files.* En revanche, en cas d'erreur, le processus s'interrompt dans la phase où l'erreur a été rencontrée. Il faut alors consulter les messages d'erreur[1]s dans l'onglet *Report* puis effectuer les corrections requises et relancer la compilation. Si vous utilisez le fichier disponible en téléchargement la compilation se fera de bout en bout sans erreur.

La compilation étant achevée, Radiant a affecté de façon aléatoire les entrées et sorties sur les broches du FPGA. Il faut donc modifier cette affectation afin de la rendre compatible avec le schéma de la carte OUAH !. Pour cela, on dispose de deux possibilités : utiliser l'éditeur de contraintes (Menu *Tools* puis *Device Constraint Editor*) ou bien éditer le fichier de contraintes (fichier .pdc) : toute modification par l'éditeur de contraintes se répercute sur le fichier de contraintes et inversement toute modification du fichier de contraintes sera visible sous l'éditeur de contraintes. J'ai opté pour l'édition du fichier de contraintes. On va donc forcer Radiant à générer le fichier de contraintes. Pour cela on appelle l'éditeur de contraintes et dans la colonne de gauche de l'éditeur on clique sur l'icône *Back Annotate Assignement*. Ensuite on clique sur l'icône disquette (bandeau supérieur) ce qui permet d'enregistrer un fichier de contraintes, puis on ferme l'éditeur de contraintes. Le fichier de contraintes est désormais accessible sous la rubrique *Post Synthetisys Constraint File* et par un double clic sur le nom du fichier on peut l'ouvrir pour le modifier. Pour éviter ce travail fastidieux, vous pouvez tout simplement importer le fichier de contraintes disponible sur Githun. Ceci fait, il faut relancer le processus de compilation et finalement on obtiendra un fichier *bitstream* prêt à être téléchargé dans la carte.

C. Programmation du FPGA

Le programmateur s'installe automatiquement en même temps que Radiant. Il peut être appelé depuis Radiant ou bien en tant que programme indépendant. Avant de lancer le programmateur vous devez connecter la carte Updiuno3.1 à l'ordinateur avec un simple câble USB (le plus court possible).

Lancez le programmateur (menu *Tools* puis *Programmer*) et cliquez sur le bouton *Detect Cable* et vérifiez que le message *Board with FTDI-USB Host Chip Detected* est affiché.

Ensuite, dans la fenêtre supérieure gauche du programmateur double cliquez dans le champ *Operation* ce qui ouvre une nouvelle fenêtre. Dans le champ *Target Memory* sélectionnez *External SPI Flash Memory puis* vérifiez que le paramétrage est conforme aux indications de la Figure 6 et enfin cliquez sur OK.

Pour lancer la programmation, il faut alors aller dans le menu *Run* et sélectionner *Program Device* et après quelques secondes le FPGA de la carte UpDuino3.1 est programmé.

D. Création de nouvelles implémentations

Comme indiqué en introduction dans cette section, le logiciel Radiant autorise la création d'une infinité d'implémentations. Je suggère d'utiliser cette possibilité chaque fois que vous voudrez faire évoluer le code VHDL tout en préservant les versions précédentes. Ainsi en cas de bug, il sera toujours possible de revenir vers une version antérieure fonctionnelle.

Il convient de signaler que le programmateur ne va pas

[1]

automatiquement chercher le code de l'implémentation en cours d'utilisation. Il faudra sélectionner le fichier `.bin` correspondant dans la fenêtre de paramétrage du programmateur.

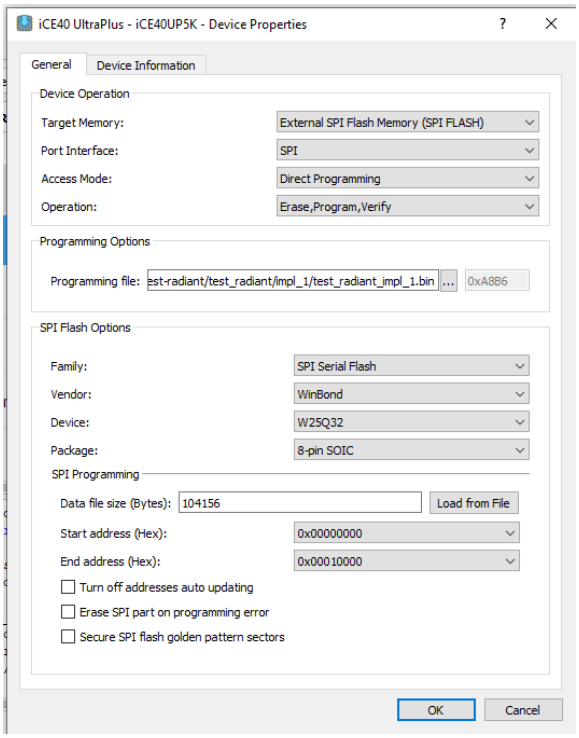


Figure 6 – Paramétrage du programmateur

VII. Realisation de la carte OUAH !

Après un maquettage sur une plaque de prototypage Verboard, j'ai dessiné le circuit imprimé sous Kicad. Ma première idée était de réaliser une carte au même format que celle du projet Cloc 2 publié dans Elektor de Mars/Avril 2023, puis d'intégrer la carte dans le même modèle de boitier. Mais j'ai assez vite constaté que c'était « mission impossible » à cause de l'encombrement de la carte Upduino d'une part et de l'alimentation intégrée d'autre part. Une option eut été d'utiliser une alimentation externe de type chargeur de téléphone portable. Mais les essais réalisés dans la phase de prototypage ont montré que ce type d'alimentation à découpage empêchait toute réception du signal DCF77 du fait du rayonnement électromagnétique de ces chargeurs.

J'ai donc dû me résoudre à augmenter les dimensions de la carte OUAH ! tout en gardant l'objectif de la loger dans le même type de boitier que celui du projet Cloc 2.0, mais de dimensions supérieures. C'est ce qui explique que la forme de la carte est très similaire à celle du projet Cloc 2.0.

Tous les fichiers de fabrication sont disponibles sur le site d'Elektor Lab ou sur Github [5] et sont prêts à être envoyés chez votre fabricant de circuit imprimé préféré.

Concernant le câblage, aucune difficulté particulière puisque tous les composants sont traversants. Il est impératif de mettre la carte Upduino et les afficheurs 7 segments sur support.

VIII. Prevention des chocs electriques

ATTENTION ! DANGER

La carte Ouah contient une alimentation intégrée alimentée en 220 v. Par conséquent des précautions d'utilisation s'imposent pour éviter toute électrocution. Il convient d'appliquer la recommandation sui suit.
Il est de la responsabilité des lecteurs souhaitant réaliser le projet Ouah ! de prendre toutes les précautions nécessaires, l'auteur ne pouvant en aucun cas être tenu pour responsable des conséquences d'un choc électrique.

Après avoir totalement câblé la carte OUAH et avant toute mise sous tension, il convient de lui adjoindre une carte d'isolation qui servira aussi de blindage pour protéger le module de réception DCF77 de tout rayonnement électromagnétique Celle-ci est découpée dans une plaque époxy cuivre 1 face épaisseur 16/10.

Cette carte sera découpée exactement selon le même contour et trous de fixation que la carte OUAH ! et sera fixé contre celle-ci côté époxy contre côté cuivre et séparée par quatre entretoises de 3 mm d'épaisseur. Voir la vue en coupe sur la Figure 7.

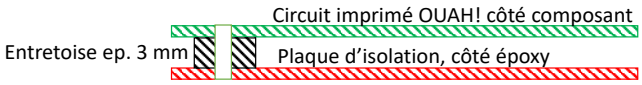


Figure 7 – Fixation d'une contre-plaque d'isolation

Les deux plaques seront rendues solidaires par des vis et écrous téflon diamètre 3 mm. On peut alors manipuler la carte sous tension sans risque de choc électrique par contact avec les soudures côté cuivre. En revanche, côté composants, il reste une possibilité de choc électrique par contact avec les vis du bornier à vis recevant le câble secteur (Figure 9). Celui-ci devra être protégé par un ruban adhésif isolant ou tout autre moyen.

IX. RACCORDEMENT ET ESSAIS PRELIMINAIRES

Pour les essais préliminaires, je conseille d'utiliser le simulateur de signaux DCF77 décrit plus loin dans cet article. Celui-ci se raccordera au bornier 5 pôles situé en bas à droite sur la Figure 9. On utilisera les pôles GND et S5V pour son alimentation et le pôle S77 pour le signal DCF. Il faudra aussi positionner les deux cavaliers JP1 et JP2 en position S. On peut ensuite connecter le cordon 220 v sur le connecteur 3 pôles[2] situé en bas à gauche sur la Figure 9 puis brancher celui-ci sur une prise secteur. Après une pression sur le bouton RESET on doit voir s'afficher des 0 sur les 8 afficheurs 7 segments. Après une ou deux minutes, heure et date devraient s'afficher correctement.

Cette étape étant franchie avec succès, on peut alors connecter le récepteur DCF77. Celui-ci se raccordera au connecteur 5 pôles situé en bas à droite sur la Figure 9. On utilisera les pôles GND et R5V pour son alimentation et la pôle R77 pour le signal DCF77. Il faudra aussi positionner les deux cavaliers JP1 et JP2 en position R. Si la réception est correcte, après une ou deux minutes, heure et date devraient s'afficher correctement. Dans le cas contraire, il faudra rechercher – parfois non sans mal - la position optimale de l'antenne en ferrite pour obtenir un bon fonctionnement.

Si une pile est utilisée pour assurer la continuité du fonctionnent en cas de coupure secteur, il faudra la connecter au bornier 2 pôles en respectant les polarités comme indiqué sur la Figure 9. Ne pas oublier de placer un cavalier sur JP3.

X. MISE EN BOITE

La carte OUAH ! a été intégrée dans un boitier Hammond 1591T en Polycarbonate, cristal, 150 x 46 x 80 mm.

La Figure 8 est une vue de côté montrant la carte et à sa contreplaque isolante, le tout monté sur des entretoises de 15 mm fixées sur le fond du boitier. Ces entretoises permettent de conserver un vide dans lequel il sera possible de mettre la pile, le simulateur et le récepteur DCF77. Malheureusement les essais réalisés à ce stade n'ont jamais permis de recevoir le signal DCF77 dès lors que l'antenne est placée dans le boitier. Donc je ne recommande pas cette solution et préconise de n'intégrer que le simulateur et la pile. Quant au récepteur, il sera placé à l'extérieur du boitier. Selon l'intérêt suscité par ce projet, je publierai ultérieurement, lors d'une mise à jour de cet article la solution que j'aurai finalement adopté. Je suis aussi preneur de vos propres réalisations.

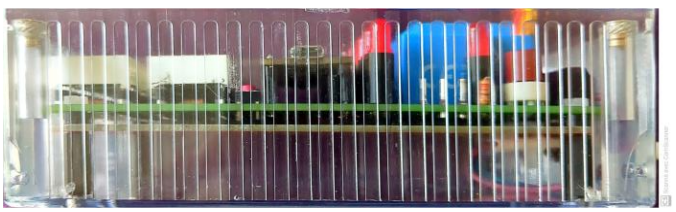


Figure 8 – integration dans le boitier : vue de côté
(on distingue la carte d'isolation située sous la carte Ouah!)

[2] Le pôle du milieu est non connecté.

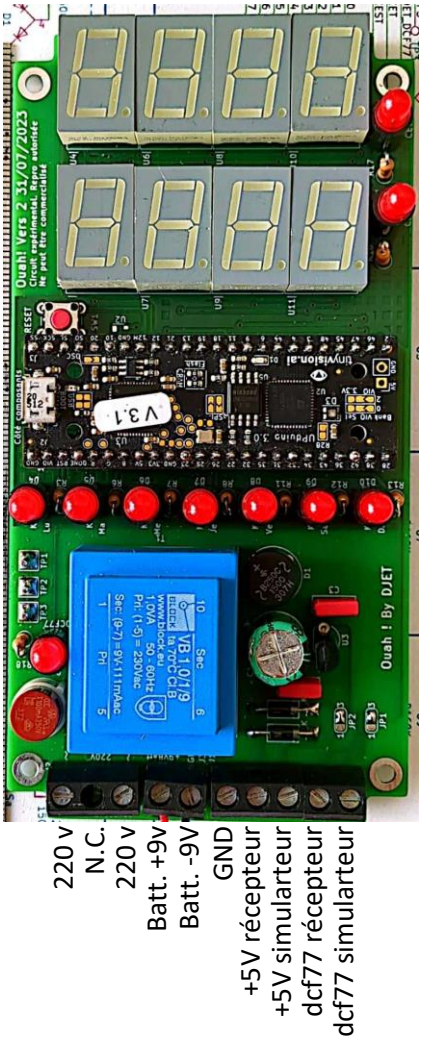


Figure 9 – Vue de dessus de la carte Ouah!

XI. LE SIMULATEUR

Comme cela a été indiqué, il est indispensable de disposer d'un simulateur de signal DCF77 pour mettre au point une horloge DCF77.

A. Le simulateur pour Raspberry Pi

Dans un souci de simplicité, une première version a été développée en C sur Raspberry Pi dans l'environnement Pi OS (une Pi Zéro W est tout à fait suffisante). Le code source est disponible sur Elektor Labs ou sur Github [5] dans le fichier rpi_dcf77sim.c.

Dans ce qui suit on suppose le Rpi alimenté par sa prise USB, connecté à votre réseau local en Wifi ou Ethernet et mis à la date et à l'heure locale (en France, UTC + 1 ou UTC +2 selon la période de l'année.)

A partir d'un PC sous Windows ou d'un système sous Linux, il faudra établir une liaison SSH avec le Rpi et créer un répertoire dédié rpi_dcfsim77 puis installer le code source dans ce répertoire, le compiler puis l'exécuter avec les commandes suivantes :

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

$ ./rpi_dcf77sim

Le simulateur affiche sur la console la date et l'heure courante, puis la traduction dcf77 sous forme d'une suite de zéro et de 1. Il suffit de raccorder la carte OUAH ! à la broche GND et à la broche GPIO22 du Rpi pour recueillir le signal DCF77.

B. *Le simulateur pour Pico W*

La version du simulateur décrite ci-dessus, si elle est adaptée en phase de mise au point, ne l'est pas pour un usage continu et une intégration dans le boitier de la carte OAUH ! à cause de sa consommation excessive. C'est ce qui m'a conduit à porter le code C sur la carte Pico W. C'est un travail relativement conséquent qui pourrait justifier en soi un article spécifique. Il est toutefois possible pour les lecteurs ayant une déjà la maitrise du développement en C sous Visual Studio Code (VSCode), de compiler le code source fourni après l'avoir modifié pour prendre en compte le SSID et le PASSWORD de leur propre réseau Wifi. Il convient de noter que ce code fait usage de codes développés par la fondation Raspberry : il s'agit des codes figurant dans les exemples fournis avec VSCode pour Pico et PicoW : `ntp_client et rtc.`

La configuration d'un projet sous Visual Stufio Code est loin d'être triviale. Afin de simplifier cette tâche, j'ai choisi d'intégrer le projet `simdcf77` dans les projets exemples fournis avec Visual Studio Code. Ceci permet de bénéficier des fichiers `CMakeLists.txt` correctement configurés. Voici comment procéder sur un PC sous Windows.

1. Installer VSCode (voir les tutoriels sur le site de la fondation Rpi).
2. Lancez VSCode et ouvrez le dossier des exemples puis cliquez sur le dernier fichier `MakeLists.txt` (situé après le fichier `.gitignore` et ajoutez les lignes suivantes juste après la 1ère ligne :
   ```
   set(PICO_BOARD pico_w)
   set(WIFI_SSID "votre_ssid")
   set(WIFI_PASSWORD "votre_password"
   ```
   Sauvez et attendez que VSCode reconstruise la configuration.
3. Localiser sur votre disque dur le répertoire `build` et ouvrez ce répertoire (Figure 10-a).
4. Ouvrez les sous répertoires `pico_w` puis `wifi ;`
5. Copiez dans ce répertoire le dossier `simdcf77` que vous aurez récupéré sur le Github [5] du projet Ouah ! (Figure 10-b).
6. Dans VSCode ouvrez le sous-dossier `pico_w` puis le dossier `wifi` puis le dossier `simdcf77.`
7. Cliquez sur le fichier `picow_symdcf77.c` et recherchez dans ce fichier les lignes contenant
   ```
   #define WIFI_SSID "xxxxx"
   #define WIFI_PASSWORD "xxxx"
   ```
   et remplacez les XXXX par vos SSID et PASSWORD puis sauvez la version modifiée (Figure 10-c).
8. Cliquez ensuite sur l'icône CMake située sur la barre d'actions verticale située à l'extrême gauche.
9. De nouveau ouvrez les dossiers et sous dossiers pour atteindre `symdcf77` et lancez la compilation du projet en cliquant sur l'icône située à droite de la `ligne pico_symdcf77_backgound`. (Figure 10-d).
10. Avec l'explorateur de fichier Windows, ouvrez `build->pico_w->wifi->symdcf77` vous trouverez le fichier : `picow_symdcf77_background.uf2` à installer par un simple glissé-déposé sur la carte PicoW selon la procédure décrite dans les tutoriels de la carte Pico ou Pico W disponibles sur le site de la fondation Rpi (Figure 10-e).
11. Après quelques instants, la connexion de la carte PicoW à votre réseau Wifi devrait se faire puis vous verrez clignoter led intégrée à la carte. Le simulateur est prêt à être connecté (broches 38 : GND, 39 : VSYS et 29 : GPIO22) à la carte Ouah !

XII. Conclusion et futurs travaux

Le bon fonctionnement d'une horloge DCF77 reste très tributaire de la qualité de réception du signal DCF. Celle-ci est assez médiocre en France avec les récepteurs que l'on trouve à bas coût. Le fonctionnement *indoor* est quasi impossible. En *outdoor* il est nécessaire de trouver la bonne orientation de l'antenne et d'être en milieu ne subissant pas de perturbations électromagnétiques. Mes prochains travaux porteront sur l'amélioration de la qualité de réception par ajout d'un préamplificateur d'antenne sélectif sur la fréquence 77,5 kHz.

Des améliorations du code VHDL sont également prévues pour prendre en compte les bits de parité émis dans la trame DCF77.

Toutes suggestions sur des améliorations pouvant être apportées à ce projet sont bienvenues.

XIII. References

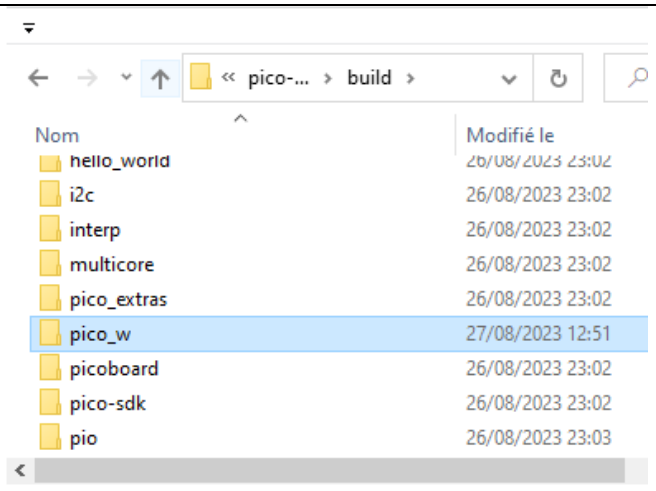
[1] https://fr.wikipedia.org/wiki/DCF77
[2] https://www.sonelec-musique.com/
[3] http://fimcachan.free.fr/vhdl/horloge2003.pdf
[4] https://www.latticesemi.com/LatticeRadiant?pr031521
[5] https://github.com/duodiscus92/projet-ouah/tree/master
[6] http://www.radioman33.com/pages/les-boites-de-jeu-radio/boite-de-montages-philips-ee20.html

XIV. Biographie

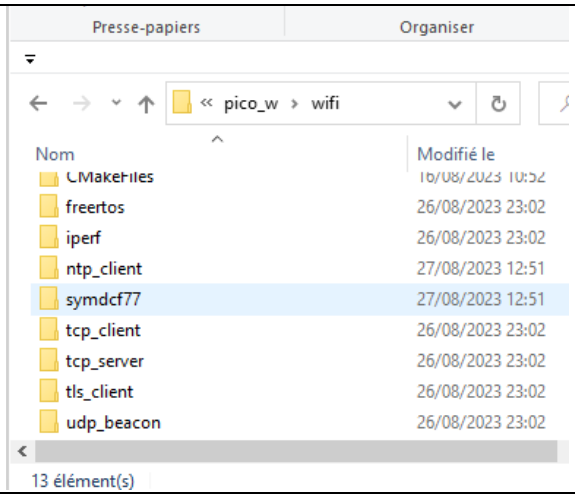
Dr. Jacques Ehrlich est Directeur de Recherche Emérite à l'Université Gustave Eiffel et consultant. Il a dirigé jusqu'en 2014, le LIVIC un laboratoire de recherche sur les systèmes avancés d'aides à la conduite et le véhicule autonome et connecté. Il a coordonné pendant huit ans (2012-2019) un groupe international d'experts internationaux sur l'exploitation des réseaux routiers et les systèmes de transports intelligents (STI) à l'association mondiale de la route (PIARC).

C'est un passionné d'électronique qu'il découvrit à 15 ans avec la boite de construction Philips EE20 [6], mais aussi d'informatique et de vol en planeur autant de disciplines qu'il pratique comme hobbys et loisirs de retraité.

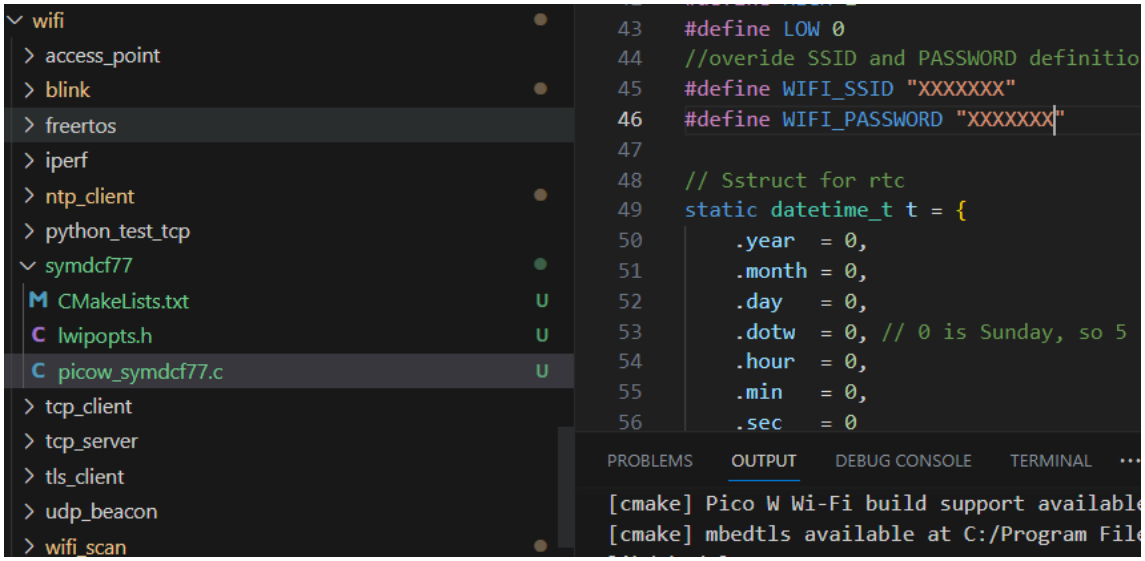
Site web : www.djet.fr



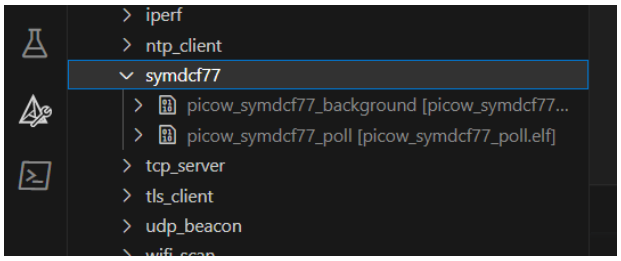
a) Contenu du répertoire build



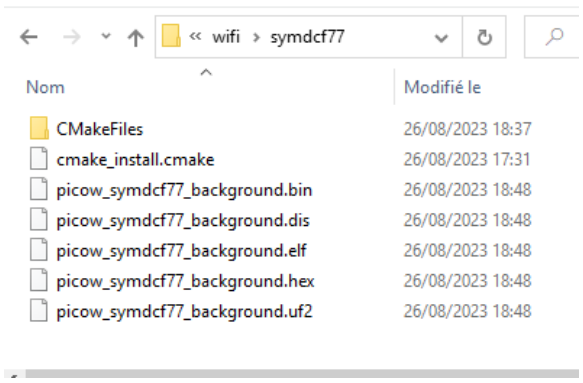
b) Installation du repertoire symdcf77



c) Dans VSCode, modification du fichier picow_symdcf77



d) Lancement de la compilation du projet



e) Fichier uf2 à glisser deposer dans la Pico W

Figure 10 – Etapes de la création du projet picow_symdcf77

XV. ANNEXE 1 LISTE DES COMPOSANTS

Quantités	Référence(s)	Valeur	Désignation	Observations
1	C1	470 uF	Condensateur polarisé	
4	C2, C3	100 nF	Condensateurs	
1	D1	B250C1500G	Pont à diodes	
2	D2, D3	1N4007	Diode	
10	D4, D5, D6, D7, D8, D9, D10, D11, D12, D13	LED	LED rouge	
1	F1	LITT 37201000411	Fusible radial 100 mA retardé	A monter sur support
1	J1	Screw_Terminal_01x05	Bornier à vis 5 pôles	
1	J2	Screw_Terminal_01x03	Bornier à vis 3 pôles	
1	J3	Screw_Terminal_01x02	Bornier à vis 2 pôles	
2	JP1, JP2	Jumper_3_Open	Support pour cavalier 3 pôles	
1	JP3	Jumper_2_Open	Support pour cavalier 2 pôles	
8	R1, R2, R4, R5, R9, R10, R14, R15	150	Résistance	
10	R3, R6, R7, R8, R11, R12, R13, R16, R17, R18	150	Résistance	
2	R19, R20	10k	Résistance	
1	SW1	SW_Push	Bouton Push	
1	T1	BLOCK VB 1,0/1/9	Transformateur 220V/9V	
3	TP1, TP2, TP3	TestPoint	Point de test	
1	U1	Upduino31-v3	Carte Upduino 3.1	A monter sur support
1	U2	TL7705A	Générateur reset TL7705A	A monter sur support
1	U3	LP2950-5.0_TO92	Régulateur 5V	A monter sur support
8	U4, U5, U6, U7, U8, U9, U10, U11	D168K	Afficheurs 7 segments	A monter sur support
1	Boiter Polycarbonate Crystal	Hammond 1591T	Boitier 150 x 46 x 80mm IP54	
1	Récepteur DCF77	Kundo XT 950 051	Récepteur DCF77	
1	Optionnel	Raspberry Pico W	Simulateur DCF77	Voir note ci-dessous

Note :