

# horloge en VHDL réalisée avec ChatGPT

Jean-François Simon (Elektor)

Jusqu'à présent, je n'avais jamais utilisé de FPGA, ni les langages Verilog ou VHDL. Découvrez dans cet article comment ChatGPT m'a aidé à créer mon premier projet basé sur un FPGA en partant de zéro.

Il y a quelque temps, je me suis procuré une carte DE10-Lite du fabricant Terasic (**figure 1**), contenant un FPGA de la série Intel MAX 10, quelques afficheurs 7-segments, des interrupteurs à glissière, des boutons poussoirs avec leur circuit anti-rebond, des LED, etc. En somme, une carte sympathique et bien équipée pour débuter, mais que j'avais un peu délaissée. L'article de Theo Mulder publié récemment dans Elektor [1] m'y a fait repenser. Cette édition spéciale IA est l'occasion parfaite pour me lancer ! J'ai choisi d'utiliser le langage VHDL ; reportez-vous à l'encadré dédié dans [1] pour vous aider à choisir entre VHDL et Verilog.

Parmi les projets de magazines d'électronique qui apportent toujours, décennie après décennie, une inexplicable satisfaction, les horloges figurent en bonne place. Il y a un côté rassurant à observer l'écoulement régulier des secondes et le clignotement des LED rouges. La DE10-Lite, d'une taille compacte d'environ 8 cm x 10 cm, contient justement un afficheur 7 segments à 6 chiffres et une petite protection en plexiglas. La situation est donc idéale pour faire une horloge à LED ! Evidemment, il est un peu exagéré d'utiliser un composant en boîtier BGA à 484 broches contenant 50 000 éléments logiques pour fabriquer l'équivalent d'une pendule de salle de bains à 5 euros. Mais laissons cela de côté, l'important est d'apprendre et de s'amuser ! Si vous connaissez d'autres exemples amusants d'abus de moyens, faites-le nous savoir dans le courrier des lecteurs.

## Petit avertissement

Cet article contient quelques points de détail, mais ne pourra pas être un tutoriel de bout en bout. Ce serait beaucoup trop long et ennuyeux.



Figure 1. La carte de développement DE10-Lite.

De plus, même si vous tapez *exactement* les mêmes « prompts » que moi dans ChatGPT, je ne peux pas garantir que les réponses que vous obtiendrez seront les mêmes que celles que j'ai reçues, selon que vous utilisez la version gratuite ou payante, selon le modèle (ChatGPT 4, 4o, 4o mini), etc. Voyons plutôt cela comme un retour d'expérience. Je vous invite à essayer vous-même d'utiliser une IA comme ChatGPT pour vos projets ! Il n'y a pas de raison de se limiter aux FPGA, ou à la carte DE10-Lite en particulier, ou à l'utilisation de Quartus, ou au langage VHDL. Suivez votre inspiration et soyez persévérant ! Ce type de projet n'avance qu'en résolvant successivement des problèmes plus ou moins grands.

## Premiers pas

On dit souvent que la première chose à faire pour se familiariser avec le matériel et le logiciel est de faire clignoter une LED. Mais, sur un FPGA, il n'y a pas de fonction prête à l'emploi comme `delay()` sur Arduino. La première étape sera donc encore plus modeste : allumer une des LED de la carte lorsque l'on appuie sur l'un des boutons de la carte. Je me rends donc sur le site ChatGPT et lui donne le *prompt* (instruction) suivant :

*Ecris-moi un code VHDL très simple pour allumer une LED lorsqu'un bouton est pressé sur une carte FPGA DE10-lite.*

La réponse de ChatGPT est présentée dans le **Listing 1**. Pour pouvoir implémenter ce programme dans le FPGA, le chemin est encore long. Voici un aperçu des étapes à suivre. Tout d'abord, téléchargez et installez Quartus Prime Lite [2]. Créez un nouveau projet vide en cliquant sur *New Project Wizard*. Lors de la création du projet, donnez-lui un nom (j'ai choisi « test1 ») et, dans la liste déroulante *Devices*, sélectionnez le FPGA que nous allons utiliser en particulier : le 10M50DAF484C7G. Ensuite, créez un nouveau fichier VHDL (*File* → *New*) puis copiez et collez le code dans ce fichier.

Ensuite, il faut indiquer à Quartus les broches d'entrées-sorties que nous souhaitons utiliser. Je recommande de télécharger le *System CD* fourni par Terasic [3], qui contient de la documentation et des fichiers d'exemple. Le *User Manual* sera d'une grande aide, et



## Listage 1. ChatGPT a généré ce code simple pour allumer une LED.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity LED_Button is
    Port (
        KEY : in std_logic; -- Input button
        LED : out std_logic -- Output LED
    );
end LED_Button;

architecture Behavioral of LED_Button is
begin
    -- Process to control LED
    process(KEY)
    begin
        if (KEY = '0') then -- Active low button (pressed)
            LED <= '1';      -- Turn on LED
        else
            LED <= '0';      -- Turn off LED
        end if;
    end process;
end Behavioral;
```

indique notamment la correspondance entre les numéros de broches et les composants de la carte. J'utilise ici le bouton *KEY0*, connecté à la broche *PIN\_B8* du FPGA, et la LED *LEDR0* (*PIN\_A8*). La façon la plus simple de procéder est d'utiliser le *Pin Planner*, dans le menu *Assignments*. Avant cela, il faut indiquer au système quels sont les signaux présents dans le projet. Une *Compilation* complète (a. dans la **Figure 2**) conviendrait pour cela, mais elle est assez lente. Cliquez donc sur *Start Analysis & Elaboration* (b.), une sorte de compilation partielle, qui fera l'affaire et est plus rapide.

La première erreur ne se fait pas attendre : *Error (12007): Top-level design entity "test1" is undefined*. Par défaut, Quartus donne à l'entité de plus haut niveau le même nom que le nom du projet

(ici « test1 »). Or, le code fourni par ChatGPT décrit une entité nommée *LED\_Button*. Dans le menu *Assignments* → *Settings* → *General* il est possible de spécifier le nom de l'entité de plus haut niveau ; entrez *LED\_Button* au lieu de *test1*. Ensuite, l'analyse et l'élaboration peuvent s'exécuter correctement. Le *Pin Planner* est ensuite utilisé pour spécifier les emplacements *PIN\_B8* pour *KEY* et *PIN\_A8* pour *LED*. Enfin, effectuez une compilation complète (cliquez sur le triangle bleu ou CTRL-L).

## Programmation du FPGA

Pour charger la conception fraîchement compilée dans le FPGA, utiliser le *Programmer* (dans le menu *Tools*). Après la programmation, on peut voir que toutes les LED de la carte, à l'exception de *LEDR0*, sont allumées. Un appui sur *KEY0* allume cette dernière. Victoire ! Pour ceux de nos lecteurs qui voudraient tenter l'expérience, je recommande la vidéo [4], qui donne des informations utiles pour ces premiers pas. L'auteur de la vidéo utilise un FPGA Cyclone IV au lieu d'un MAX 10, pensez donc à adapter ses instructions à votre cas précis.

Les neuf autres LED (*LEDR1* à *LEDR9*) restent allumées car, par défaut dans Quartus, le paramètre suivant est activé : *Reserve all unused pins as inputs tri-stated with weak pull-up*. Ces résistances de tirage limitent le courant à une valeur très faible, mais suffisante pour allumer les LED. Ce paramètre peut être modifié dans le menu *Assignments* → *Device* → *Device and Pin Options* → *Unused Pins*. Par exemple, il peut être réglé sur *As output driving ground* pour forcer un état bas et éteindre les LED. Après recompilation et reprogrammation, elles sont effectivement éteintes, mais certains segments de l'afficheur à 7 segments sont maintenant allumés. En fait, ces segments sont actifs à l'état bas, comme le précise la documentation. Une autre approche consisterait à forcer les LED ou les segments à 1 ou 0 dans le code. Nous verrons cela plus loin.

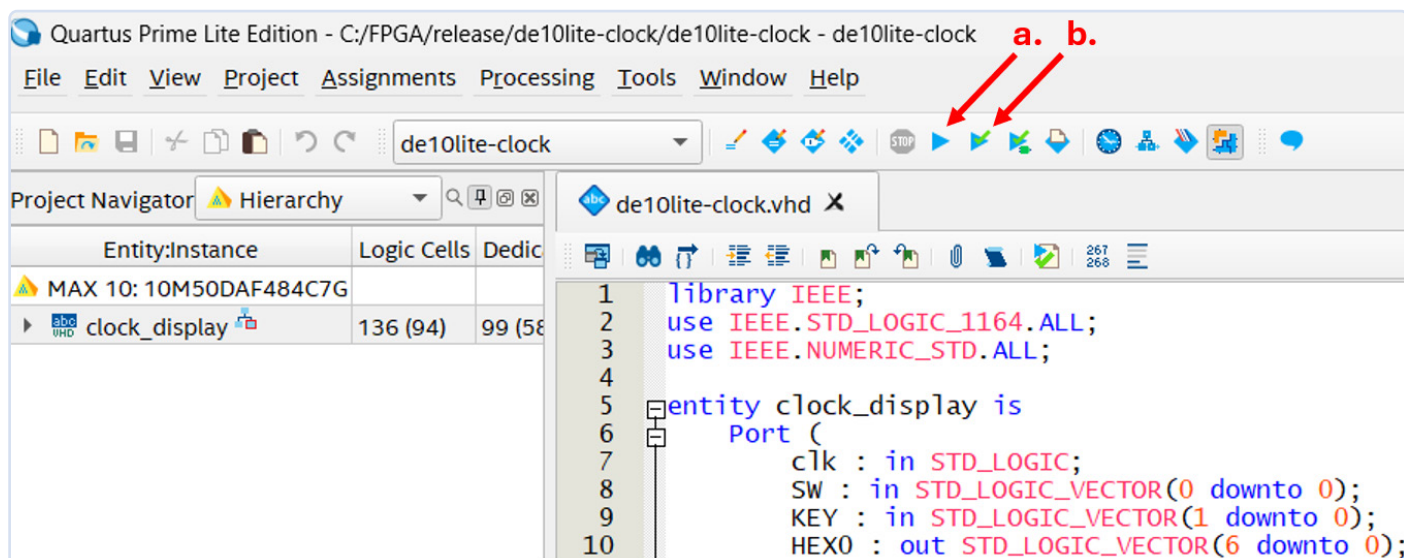


Figure 2. Analyse, élaboration et compilation dans Quartus.



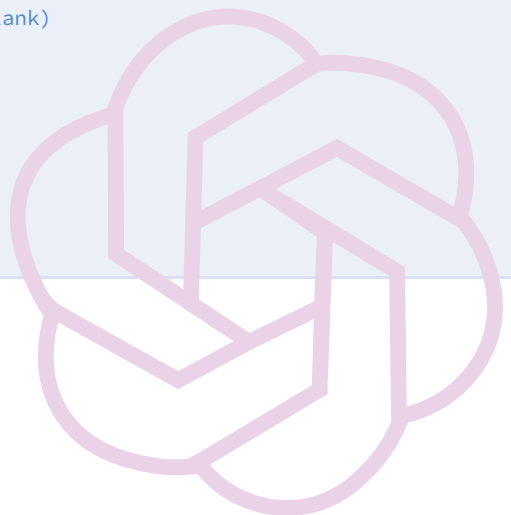
## Listage 2. Commande de l'afficheur à 7 segments.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity seven_segment_display is
    Port (
        clk : in STD_LOGIC; -- Clock signal
        digit : in STD_LOGIC_VECTOR(3 downto 0); -- 4-bit input for the digit (0-9)
        segments : out STD_LOGIC_VECTOR(6 downto 0) -- Outputs for the 7-segment display
    );
end seven_segment_display;

architecture RTL of seven_segment_display is
    signal seg_reg : STD_LOGIC_VECTOR(6 downto 0); -- Register for storing segment data
begin
    -- Sequential logic to update the display register on every clock edge
    process(clk)
    begin
        if rising_edge(clk) then
            case digit is
                when "0000" => -- Digit 0
                    seg_reg <= "0000001"; -- a,b,c,d,e,f are on, g is off
                when "0001" => -- Digit 1
                    seg_reg <= "1001111"; -- b,c are on
                when "0010" => -- Digit 2
                    seg_reg <= "0010010"; -- a,b,d,e,g are on
                when "0011" => -- Digit 3
                    seg_reg <= "0000110"; -- a,b,c,d,g are on
                when "0100" => -- Digit 4
                    seg_reg <= "1001100"; -- b,c,f,g are on
                when "0101" => -- Digit 5
                    seg_reg <= "0100100"; -- a,c,d,f,g are on
                when "0110" => -- Digit 6
                    seg_reg <= "0100000"; -- a,c,d,e,f,g are on
                when "0111" => -- Digit 7
                    seg_reg <= "0001111"; -- a,b,c are on
                when "1000" => -- Digit 8
                    seg_reg <= "0000000"; -- All segments are on
                when "1001" => -- Digit 9
                    seg_reg <= "0000100"; -- a,b,c,d,f,g are on
                when others =>
                    seg_reg <= "1111111"; -- All segments off (blank)
            end case;
        end if;
    end process;

    -- Combinational logic to drive the segments
    segments <= seg_reg;
end RTL;
```





### Listage 3. Entité de plus haut niveau pour afficher un chiffre.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top_level is
    Port (
        clk : in STD_LOGIC; -- Clock input
        segments : out STD_LOGIC_VECTOR(6 downto 0) -- Outputs to the 7-segment display
    );
end top_level;

architecture Behavioral of top_level is
    signal digit_signal : STD_LOGIC_VECTOR(3 downto 0); -- 4-bit digit signal
begin
    -- Instantiate the seven_segment_display module
    U1 : entity work.seven_segment_display
        Port map (
            clk => clk,
            digit => digit_signal,
            segments => segments
        );

    -- Set the digit to "4"
    digit_signal <= "0100"; -- Binary for 4

end Behavioral;
```

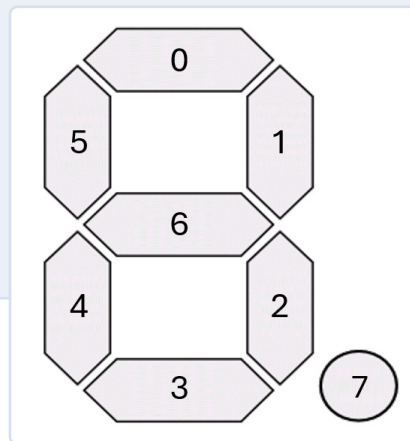


Figure 3. Adressage des différents segments de l'écran.

### Quelques chiffres

Voyons maintenant comment afficher des chiffres sur l'afficheur à 7 segments. J'ai saisi le prompt suivant :

*Ecris-moi un code VHDL capable d'écrire n'importe quel chiffre de 0 à 9 sur l'afficheur 7 segments intégré d'une carte de développement DE10-lite.*

La réponse de ChatGPT comprend du code, ainsi que des explications que j'omets ici faute de place ; le code est visible dans le **listage 2**. Pour faciliter la réutilisation de ce code, je l'ai copié-collé dans un nouveau fichier VHDL appelé (par exemple) *7segment.vhd*. N'oubliez pas de cliquer sur *Add Current File to Project* dans le menu *Project*. Le code fourni est générique, et n'inclut pas le nécessaire pour afficher un chiffre en particulier afin de vérifier le fonctionnement. Je complète donc ma demande : *J'ai collé ce code dans un fichier 7segment.vhd. Montre-moi quel code utiliser dans l'entité de plus haut niveau pour afficher le chiffre « 4 »*. La réponse est donnée dans le **listage 3**.

Voyons cela. L'entité de plus haut niveau, *top\_level*, possède une entrée *clk* pour le signal d'horloge, qui est transmis à l'entité *seven\_segment\_display*. Le signal *digit\_signal* est utilisé pour contenir la valeur du chiffre, encodé en binaire sur 4 bits. Pour le chiffre 4, l'équivalent binaire est 0100. L'entité *seven\_segment\_display* est instanciée et « mappée » avec les signaux de niveau supérieur (*clk*, *digit\_signal* et *segments*). Le *mapping* consiste à relier les signaux entre eux. Par exemple, *digit => digit\_signal* connecte le signal de niveau supérieur *digit\_signal* au port d'entrée *digit* de l'entité *seven\_segment\_display*, et permet ainsi au premier de contrôler le second. Les signaux *clk* et *segments* sont

également mappés de la même manière. Enfin, le nombre à afficher est défini à l'aide de l'instruction *digit\_signal <= "0100"*. Notez la différence entre l'opérateur d'affectation *<=* et l'opérateur d'association *=>*, utilisé notamment pour le *mapping* des signaux.

Avant de pouvoir contempler le résultat, il faut ajouter des *Pin Constraints*, afin d'obliger Quartus à utiliser les broches adéquates, reliées aux segments de l'afficheur. Les segments sont généralement marqués d'une lettre allant de « a » à « g », mais Terasic a choisi de les numéroter de 0 à 6 (**figure 3**). Le manuel d'utilisation nous donne la correspondance des broches ; un extrait est présenté dans le **tableau 1**. Pour cet exemple d'affichage à un chiffre, sept connexions sont nécessaires, il serait donc envisageable d'utiliser le *Pin Planner* à cette fin. Cependant, pour l'horloge finale, nous aurons besoin des six chiffres et des points décimaux, soit 48 connexions.

Il est donc plus pratique de procéder autrement. Chaque contrainte de broche correspond à deux lignes dans le *Fichier de Configuration Quartus* (fichier QSF) : une ligne *set\_instance\_assignment*, qui indique à Quartus le type d'E/S, et une ligne *set\_location\_assignment*, qui indique l'affectation de la broche elle-même. Il est beaucoup plus simple de copier ces contraintes de broches à partir d'un fichier QSF existant. Ouvrir le fichier *DE10\_LITE\_Default.qsf* dans le dossier *Demonstrations/Default* du CD Système. Copier les 96 lignes dédiées à l'affichage 7 segments :



**Tableau 1. Brochage de l'afficheur.**

Signal Name	FPGA Pin No.	Description	I/O Standard
HEX00	PIN_C14	Seven Segment Digit0[0]	3.3-V LVTTL
HEX01	PIN_E15	Seven Segment Digit0[1]	3.3-V LVTTL
HEX02	PIN_C15	Seven Segment Digit0[2]	3.3-V LVTTL
HEX03	PIN_C16	Seven Segment Digit0[3]	3.3-V LVTTL
HEX04	PIN_E16	Seven Segment Digit0[4]	3.3-V LVTTL
HEX05	PIN_D17	Seven Segment Digit0[5]	3.3-V LVTTL
HEX06	PIN_C17	Seven Segment Digit0[6]	3.3-V LVTTL
HEX07	PIN_D15	Seven Segment Digit0[7], DP	3.3-V LVTTL
HEX10	PIN_C18	Seven Segment Digit1[0]	3.3-V LVTTL
HEX11	PIN_D18	Seven Segment Digit1[1]	3.3-V LVTTL
HEX12	PIN_E18	Seven Segment Digit1[2]	3.3-V LVTTL
etc.	etc.	etc.	etc.

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL"
-to HEX0[0]
(...)
(...)
set_location_assignment PIN_L19 -to HEX5[7]
```

Collez ces lignes dans le fichier QSF de votre projet en cours. Le code des listages 2 et 3 nécessite un signal d'horloge. Copiez donc aussi le bloc de 6 lignes dédié à l'horloge interne du FPGA, indiqué par l'en-tête `# CLOCK`. Pour que cela fonctionne, vous devrez renommer `clk` en `MAX10_CLK1_50` et `segments` en `HEX0` dans le fichier VHDL principal (l'entité de plus haut niveau) :

```
entity top_level is
Port (
MAX10_CLK1_50 : in STD_LOGIC;
HEX0 : out STD_LOGIC_VECTOR(6 downto 0)
);
end top_level;
```

Il faut également renommer les mêmes signaux au moment de l'instanciation de notre affichage à 7 segments afin de faire correspondre correctement les signaux d'une entité à l'autre :

```
U1 : entity work.seven_segment_display
Port map (
clk => MAX10_CLK1_50,
digit => digit_signal,
segments => HEX0
);
```

Enfin, il est temps de procéder à une nouvelle compilation. En principe, le code devrait compiler sans erreurs. Si ce n'est pas le cas, signalez ces erreurs à ChatGPT pour qu'il vous aide à trouver le problème. Après la reprogrammation, on obtient l'affichage de la **figure 4**. Ce n'est pas un « 4 » ! Que s'est-il passé ? Inspectons le code responsable de la génération des signaux pour les segments :

```
case digit is
when "0000" => -- Digit 0
seg_reg <= "0000001"; -- a,b,c,d,e,f are on, g is off
```

## Dépannage

Les segments sont actifs à l'état bas ; le commentaire après le double tiret implique que, selon ChatGPT, la série de bits est dans l'ordre suivant : « abcdefg ». Or, selon le *User Manual*, le segment « 0 » (ou « a ») correspond au signal HEX0[0]. En VHDL, les bits sont numérotés en commençant par la gauche, le bit le plus à gauche ayant l'indice le plus élevé et le bit le plus à droite ayant l'indice le plus bas. En d'autres termes, HEX0[0], ou « a », devrait se trouver tout à droite : « gfedcba ». Il faut donc soit en informer ChatGPT et le laisser corriger le problème, soit inverser manuellement toutes les chaînes de bits de gauche à droite : 0000001 devient 1000000, 1001111 devient 1111001, etc. Une fois cela fait, après une nouvelle recompilation et reprogrammation, le chiffre 4 apparaît correctement.

Comme vous le voyez, c'est un peu laborieux. Mais il ne faut pas se décourager, il y a eu du progrès. Nous disposons maintenant d'une entité d'affichage permettant d'afficher le chiffre de notre choix sur le

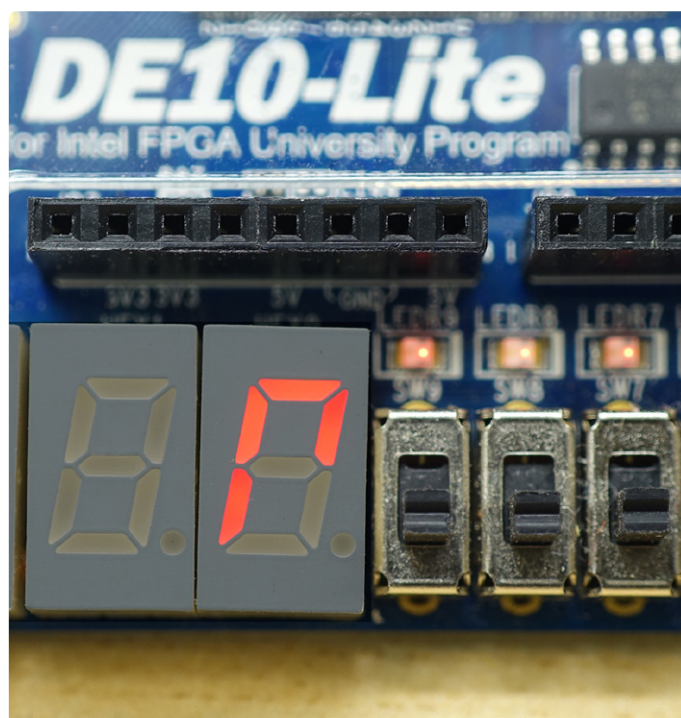


Figure 4. Essai d'affichage d'un chiffre...

## Quelques problèmes

A différents moments durant le projet, ChatGPT a rencontré les problèmes suivants :

- › Génération de code utilisant des fonctions sans les avoir correctement importées, ce qui entraînait des erreurs telles que « object 'to\_unsigned' is used but not declared. »
- › Déclarer un process tel que `process(clk_1hz, reset)` et utiliser par la suite des instructions comme `rising_edge(KEY(0))` à l'intérieur de celui-ci, ce qui provoque une erreur, car KEY ne figure pas dans la liste de sensibilité (la première série de parenthèses dans la déclaration).
- › Générer des verrous (latches). Je vous laisserai chercher pourquoi les latches sont particulièrement problématiques dans les projets à FPGA, mais en résumé, ils peuvent conserver un état sans aucune entrée d'horloge, ce qui les rend imprévisibles. Quartus refuse de compiler du code qui génère des verrous et produit des messages d'erreur.
- › Tenter de modification des signaux à partir de deux processus différents, ce qui entraîne des messages d'erreur.
- › Difficultés pour gérer deux horloges distinctes – où l'horloge principale (rapide) est utilisée pour détecter les pressions sur les boutons avec une certaine réactivité, tandis que l'horloge 1 Hz ne fait qu'incrémenter les compteurs de temps. ChatGPT a parfois confondu les deux, ce qui a entraîné un affichage incohérent, ou bien une très mauvaise réactivité à l'appui sur les boutons.
- › Création de processus sensibles à deux horloges en même temps. Quartus a du mal à déterminer quel front d'horloge doit déclencher les événements.
- › Répondre « Pour résoudre ce problème, nous allons utiliser un flag (drapeau) », et fournir ensuite un code qui n'implémente pas réellement de flag. Également, donner des explications incorrectes sur certains comportements.
- › Avoir des problèmes de logique. Exemple : l'horloge se réinitialise à 00:00 après 19:59:59 au lieu de 23:59:59.
- › Difficultés à résoudre les erreurs de logique lorsqu'elles sont présentes dans plusieurs parties du code à la fois. ChatGPT a tendance à ne corriger l'erreur qu'à un seul endroit, ce qui ne résout pas le problème entièrement et conduit à des cycles répétitifs.

Résoudre tous ces problèmes s'est avéré être une tâche difficile, mais assez amusante et très instructive !

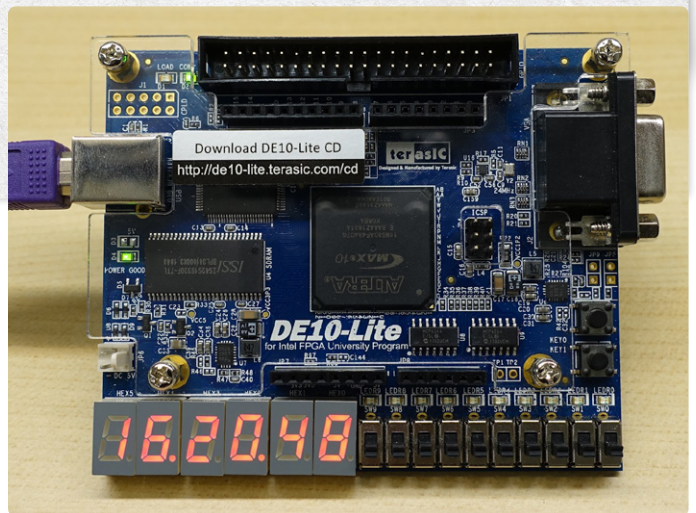


Figure 5. L'horloge terminée.

premier emplacement de l'afficheur. Les *pin constraints* sont en place permettant d'utiliser aisément les cinq autres chiffres. À présent, il n'y a plus qu'à continuer à dialoguer avec ChatGPT pour avancer ! Personnellement, j'aime bien procéder par étapes, je lui ai donc demandé de procéder par petites évolutions, afin d'être certain de bien comprendre. Cependant, il est tout à fait possible de lui faire votre demande finale (ici, une horloge comptant les heures, minutes et secondes sur l'afficheur 7-segments à 6 chiffres d'une carte DE10-Lite) dès le début et de tout faire d'un coup. Cela serait très certainement bien plus rapide.

## Une vraie horloge ?

Voici les étapes que j'ai suivies par la suite, sous la forme des *prompts* donnés à ChatGPT :

- › *Adapte le code pour afficher « 123456 » sur un afficheur à sept segments de 6 chiffres. Les registres d'affichage sont HEX0 à HEX5, un pour chaque chiffre.*
- › *Excellent, merci. Maintenant, écris le code VHDL pour faire une horloge utilisant cet affichage et indiquant l'heure au format hh:mm:ss.*
- › *Maintenant, donne-moi le code que je dois ajouter pour pouvoir régler l'horloge. Je voudrais que l'appui sur KEY[0] incrémente les minutes et KEY[1] incrémente les heures. Pas besoin d'intégrer un anti-rebond dans le code, il y a un circuit dédié sur la carte pour cela.*

Après avoir passé un certain temps à chasser les bugs (voir l'encadré **Quelques problèmes**), en dialoguant avec ChatGPT, et aussi en cherchant sur Google, un code fonctionnel pour une horloge complète a pu être réalisé. Le code final est trop long pour être reproduit ici, je vous invite à le consulter en ligne [5]. L'horloge terminée (**figure 5**) peut être vue en action dans la vidéo [6].

D'abord, l'entité `clock_display` est déclarée, ainsi que toutes ses entrées et sorties. Ensuite, dans le bloc `architecture`, les signaux (analogues aux variables en programmation logicielle) nécessaires à son fonctionnement sont créés : ceux-ci servent à stocker l'heure courante et l'état des boutons. Un premier `process` est utilisé pour créer une horloge de 1 Hz à partir de l'horloge de 50 MHz embarquée sur la DE10-Lite, en comptant jusqu'à 49.999.999 ; lorsque cette valeur est atteinte, le signal `clk_1hz` passe à l'état haut pendant un cycle d'horloge, puis revient à l'état bas.

Un second `process` se charge d'incrémenter les signaux pour les secondes, les minutes et les heures, ainsi que de détecter les pressions sur les boutons. L'heure courante est incrémentée toutes les secondes et les afficheurs à sept segments affichent chaque chiffre à l'aide d'une instance distincte de l'entité `seven_segment_display`. Chaque instance reçoit le chiffre à afficher (sur 4 bits) à son entrée et pilote les segments correspondants pour représenter ce chiffre. Par ailleurs, les deux boutons KEY[0] et KEY[1] sont utilisés pour régler l'heure manuellement. KEY[0] incrémente les minutes et KEY[1] incrémente les heures. L'horloge est remise à zéro lorsque l'interrupteur SW[0] est actionné. Les points décimaux sur l'écran sont utilisés pour séparer les heures, les minutes et les secondes. Ceux des chiffres 0, 1, 3 et 5





sont forcés à 1 afin de les maintenir éteints tandis que les autres sont synchronisés avec le signal d'horloge le plus lent avec les instructions `dpX <= clk_separators`. De cette manière, tous les segments de l'affichage sont explicitement utilisés ; cela vous permet de choisir l'option *Reserve all unused pins as output driving ground* pour éteindre les LED *LEDR0* à *LEDR9* sans perturber l'affichage.


Notez que, par défaut, lorsque le FPGA est programmé, le design est stocké dans la mémoire volatile du FPGA. A la prochaine mise sous tension, le programme de démonstration Terasic est rechargé à partir de la mémoire Flash interne du FPGA. Pour que celui-ci démarre l'horloge directement à chaque mise sous tension, vous devez indiquer au *Programmer* d'utiliser le fichier POF au lieu du fichier SOF ; la programmation est légèrement plus longue.

Au final, l'horloge fonctionne très bien. Au démarrage, si l'interrupteur en bas à droite est en position basse, elle commence immédiatement à compter les secondes. Pour la mettre à l'heure, actionner l'interrupteur pour la maintenir à 00:00:00. Puis, en utilisant une autre horloge comme référence, actionnez à nouveau l'interrupteur au début de la minute suivante pour commencer à compter les secondes. Les secondes sont maintenant réglées. Appuyez ensuite sur les touches KEY0 et KEY1 pour régler respectivement les minutes et les heures.

## Une approche intéressante

Ce projet utilisant ChatGPT pour concevoir une horloge en VHDL a été une expérience fascinante. Il s'agit d'une approche nouvelle par rapport à la méthode traditionnelle consistant à apprendre les bases à l'aide de livres et d'autres ressources et à commencer à programmer en partant de zéro. Avec ChatGPT, vous pouvez commencer avec un code globalement fonctionnel, qui ne contient généralement aucune erreur de syntaxe, ce qui permet d'obtenir des résultats visibles rapidement – un grand avantage pour les impatientes. Cependant, une attention particulière est requise pour résoudre les erreurs de logique et les autres défis qui se présentent en cours de route. Bien entendu, cette méthode n'est pas destinée à remplacer totalement l'apprentissage traditionnel, mais elle le complète efficacement. Les ressources listées à la fin de [1] m'ont été bien utiles. ChatGPT (ou toute autre IA) est un atout supplémentaire dans votre boîte à outils. Il peut présenter assez clairement des informations complexes, ce qui est très utile lorsque les recherches traditionnelles sur internet s'avèrent décevantes (faute de connaître le bon mot-clé par exemple). Il m'a également bien aidé pour analyser du code, en particulier du code non commenté trouvé sur Internet, et pour m'expliquer les algorithmes sous-jacents.

Ceci ne prétend pas être la meilleure façon de faire une horloge ; ce genre de projet peut être abordé de multiples façons et je vous

encourage à essayer vos propres solutions. D'autres outils utilisant l'IA, comme Claude [7], ainsi que des assistants intégrés à votre EDI peuvent être utilisés. Telle qu'elle est présentée ici, l'horloge perd l'heure lorsqu'on la met hors tension. L'ajout d'une batterie de secours constituera un petit exercice pour nos lecteurs. Et maintenant, quoi d'autre ? Pour poursuivre l'apprentissage, il me reste à entreprendre d'autres projets à FPGA. Une étape importante sera d'utiliser la simulation, car il s'agit d'une partie essentielle du développement sur FPGA que je n'ai pas abordée ici. Un avantage inhérent aux FPGA est la portabilité du code : je pourrais ainsi essayer de transférer cette horloge sur une carte plus petite et moins coûteuse, comme la iCE40HX1K-EVB d'Olimex. Cela impliquerait de concevoir un petit PCB pour ajouter des afficheurs à 7 segments, et apprendre les bases du multiplexage en VHDL, car ladite carte n'a pas d'afficheurs intégrés ni même assez de broches d'E/S pour contrôler tous les segments directement. Est-ce bien nécessaire ? Sans doute pas, mais instructif, sûrement ! 

240602-04

## Questions ou commentaires ?

Envoyez un courriel à l'auteur (jean-francois.simon@elektor.com), ou contactez Elektor (redaction@elektor.fr).



## À propos de l'auteur

Jean-François Simon a une passion de longue date pour l'électronique et s'intéresse à des sujets aussi variés que la conception de circuits, le test et la mesure, le prototypage, la rétro-ingénierie, la radio logicielle, etc. Il aime créer, modifier et améliorer ses outils et autres systèmes. Il aime également la mécanique, l'usinage et tout ce qui est technique. [https://x.com/JFS\\_Elektor](https://x.com/JFS_Elektor)



## produits

- > **Alchitry Au FPGA Development Board (Xilinx Artix 7)**  
[www.elektor.fr/19641](http://www.elektor.fr/19641)
- > **Carte de développement FPGA iCEBreaker**  
[www.elektor.fr/20625](http://www.elektor.fr/20625)

## LIENS

- [1] T. Mulder, « FPGA pour les débutants », Elektor 3-4/2024 : <https://www.elektormagazine.fr/230067-04>
- [2] Téléchargement de Quartus Prime Lite : <https://tinyurl.com/downloadquartus>
- [3] System CD de la DE10-Lite : <https://de10-lite.terasic.com/cd>
- [4] Tutoriel : faire clignoter une LED avec un FPGA, étape par étape (vidéo) : <https://www.youtube.com/watch?v=JJ3XkNcLdx8>
- [5] Code source sur GitHub : <https://github.com/jfpsimon/de10lite-clock>
- [6] L'horloge en action (vidéo) : <https://www.youtube.com/watch?v=vdxtk3s0xEA>
- [7] Claude AI : <https://claude.ai>