TP FPGA

1 Tutoriel Quartus

En TD, vous avez utilisé le logiciel `Modelsim` pour simuler vos composants VHDL. Pour les tester sur FPGA, vous aurez besoin du logiciel Quartus.

1.1 Création d'un projet

- Lancez Quartus
- Lancez l'assistant de création de projet :

```
File > New Project Wizard
```

- La première page ne sert à rien, cliquez sur `Next`
- Sur la deuxième page, choisissez un nom (eg. `tuto_fpga`) et un chemin. **Attention :** Pas d'espaces ni de caractères spéciaux !
- Cliquez sur `Next`
- Sur la page suivante, sélectionnez `Empty project` puis cliquez sur `Next`
- La page suivante permet d'ajouter des fichier. Nous n'en avons pas besoin, cliquez sur `Next`
- La page suivante est importante, elle permet de choisir le FPGA cible. Faites attention à ne pas faire d'erreur. Le FPGA à sélectionner est le suivant :

```
5CSEBA6U23I7
```

  Ni ~~5CSEBA6U23I7L~~, ni ~~5CSEBA6U23I7S~~. Cliquez sur `Next`
- La page suivante ne nous interesse pas non plus, cliquez sur `Next`
- La dernière page est un récapitulatif, cliquez sur `Finish`

1.2 Création d'un fichier VHDL

- Créez un nouveau fichier

```
File > New
```

- Une fenêtre s'ouvre, sélectionnez

```
VHDL File
```

- Écrivez un composant simple, comme celui-là

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity tuto_fpga is
    port (
        sw : in std_logic;
        led : out std_logic
    );
end entity tuto_fpga;

architecture rtl of tuto_fpga is
begin
    led <= sw;
end architecture rtl;
```

— **Important :** Le composant `top` doit avoir le même nom que celui du projet

1.3 Fichier de contrainte

Sur la DE10-Nano, il y a plusiers LED et plusieurs Switches.

`LED0` est sur la broche `PIN_W15`

`SW0` est sur la broche `PIN_Y24`

Quartus ne peut pas connaître ces informations, il faut donc lui préciser.

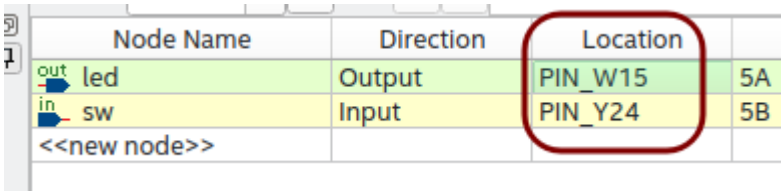
— Avant toute chose, il faut *synthétiser* le projet

> Double-cliquez sur `Analysis & Synthesis`

— Ensuite, cliquez sur :

> `Assignments > Pin Planner`

— Les signaux d'entrée/sortie définis dans l'`entity` VHDL sont listés en bas de la fenêtre qui vient de s'ouvrir.
Configurez-les de la manière suivante :



Node Name	Direction	Location	
out led	Output	PIN_W15	5A
in sw	Input	PIN_Y24	5B
<<new node>>			

— Fermez la fenêtre du `Pin Planner`, ça sauvegarde automatiquement

1.4 Programmation de la carte

— Compilez l'intégralité du projet

> Double-cliquez sur `Compile Design`

Note : Il y a quelques *warnings*. Ce n'est pas grave pour l'instant, nous verrons ça plus tard.

— Lancer l'outil de programmation du FPGA

> `Tools > Programmer`

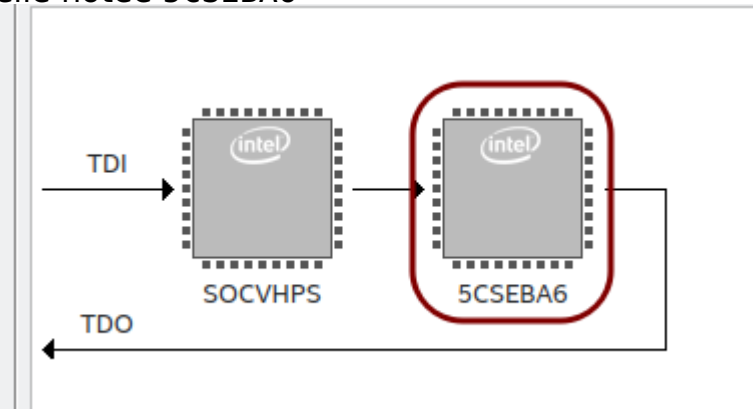
— Une fenêtre s'ouvre :

> Cliquez sur `Auto Detect`

— Deux fenêtres pop-up apparaissent successivements, acceptez les paramètres par défaut

— Vous devriez voir le schéma de deux puces.

— Séletionnez celle notée 5CSEBA6

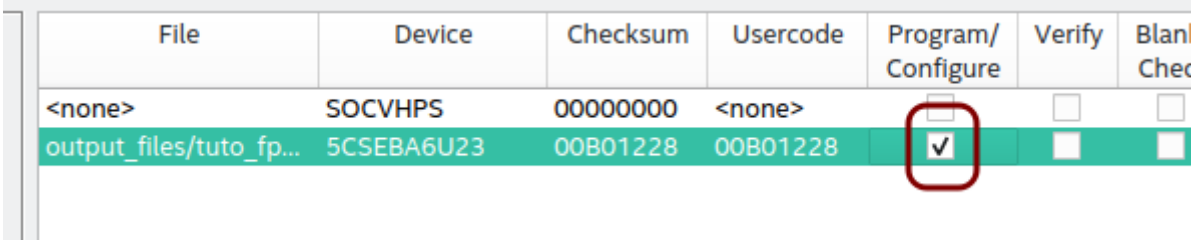


— Chargez le *bitstream*

> `Clic-droit sur la puce > Edit > Change File`

> Séletionnez le fichier `.sof` dans `output_files`

— Cochez la case `Program / Configure` comme sur la figure suivante :



— Programmez la carte

Cliquez sur `Start`

— Ça fonctionne ?

1.5 Modification du VHDL

Comme en TD, nous allons simplement remplacer nos **`std_logic`** par des **`std_logic_vector`**.

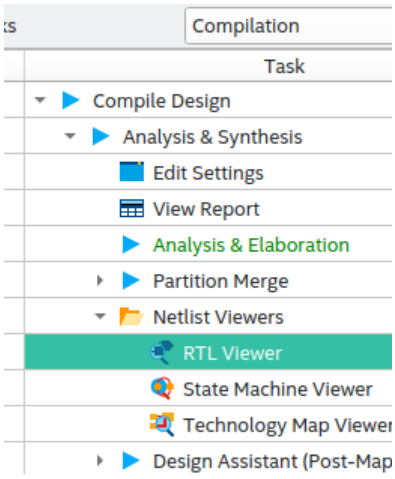
```
library ieee;
use ieee.std_logic_1164.all;

entity tuto_fpga is
    port (
        sw : in std_logic_vector(3 downto 0);
        led : out std_logic_vector(3 downto 0)
    );
end entity tuto_fpga;

architecture rtl of tuto_fpga is
begin
    led <= sw;
end architecture rtl;
```

Vous devrez aussi modifier le fichier de contrainte. Vous pouvez télécharger le systemCD contenant la documentation sur le site de Terasic. Pour aller plus vite – il faut créer un compte pour télécharger la documentation – le `User Manual` est disponible sur Moodle.

1.6 Faire clignoter une LED

Le combinatoire c'est bien, le séquentiel c'est mieux !

— Plusieurs horloges sont disponibles sur la carte. Sur quelle broche est connectée l'horloge nommée `FPGA_CLK1_50` ?
— Le code VHDL ci-dessous permet de faire simplement clignoter une LED
— Tracez le schéma correspondant à ce code VHDL
— Comparez avec le schéma proposé par `quartus` :



Dans la zone de compilation, ouvrir `Compile Design` > `Analysis & Synthesis` > `Netlist Viewers` puis lancer `RTL Viewer`

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity led_blink is
    port (
        i_clk : in std_logic;
        i_rst_n : in std_logic;
        o_led : out std_logic
    );
end entity led_blink;

architecture rtl of led_blink is
    signal r_led : std_logic := '0';
begin
    process(i_clk, i_rst_n)
    begin
        if (i_rst_n = '0') then
            r_led <= '0';
        elsif (rising_edge(i_clk)) then
            r_led <= not r_led;
        end if;
    end process;

    o_led <= r_led;
end architecture rtl;
```

— Ce n'est pas la peine de tester ce code sur la carte, la LED clignote à 50MHz : c'est trop rapide.
— En vous aidant du code ci-dessous, modifiez votre code pour réduire la fréquence :

```vhdl
process(i_clk, i_rst_n)
    variable counter : natural range 0 to 5000000 := 0;
begin
    if (i_rst_n = '0') then
        counter := 0;
        r_led_enable <= '0';
    elsif (rising_edge(i_clk)) then
        if (counter = 5000000) then
            counter := 0;
            r_led_enable <= '1';
        else
            counter := counter + 1;
            r_led_enable <= '0';
        end if;
    end if;
end process;
```

— Proposez un schéma correspondant au nouveau code
— Vérifiez à l'aide de RTL `Viewer`

Comme vous l'avez peut-être remarqué :
— Les entrées commencent par `i_`
— Les sorties commencent par `o_`
— Les registres commencent par `r_`
— Il n'y en a pas ici, mais les signaux internes commencent par `s_`
C'est une bonne habitude à prendre.

Vous noterez également l'utilisation d'un signal de reset : `i_rst_n`.
— C'est important d'avoir un signal de reset, utilisez-le pour chacun de vos registres
— Vous utiliserez le bouton poussoir nommé `KEY0`.
— Sur quelle broche du FPGA est-il connecté ?
— Que sigifie `_n` dans `i_rst_n` ? Pourquoi ?

1.7 Chenillard !

Eh oui, vous vous en doutiez, ça devait arriver à un moment ou à un autre. Vous plongez maintenant dans le grand bain, vous allez devoir concevoir votre propre composant. Sans aide, sans guidage. À vous de jouer :

— Concevez un chenillard !

— Et montrez le résultat (et le code !) à votre encadrant.

2 Petit projet : Bouncing ENSEA Logo

Objectif : Sur la sortie HDMI, faire rebondir le logo ENSEA, comme dans les lecteurs DVD (`https://www.bouncingdvdlogo.com/`).

Déroulement : C'est un genre de mini-projet. Ce texte de TP n'est volontairement pas très guidé. Vous aurez un certain nombre de choix à faire pour arriver au bout.

2.1 Contrôleur HDMI

Objectif : Le contrôleur HDMI génère les signaux nécessaires au *HDMI Transmitter*. Il génère également des signaux en direction de la circuitrie qui génère les images.

1. Récupérez les ressources sur moodle.
   Parmi ces ressources, vous trouverez :
   — Un projet Quartus disposant du pinout déjà configuré.
   — Le fichier `DE10_Nano_HDMI_TX.vhd` : C'est le *top* du projet, il définit les entrées/sorties.
   — Les fichiers `I2C_HDMI_Config.v` et `I2C_Controller.v` : Permettent de configurer le *HDMI Transmitter*. Ils sont déjà instanciés dans le *top*.
   — Le fichier `hdmi_generator.vhd` à compléter. Il est en partie instancié dans le *top*, à compléter également.
   Vous devrez simuler votre contrôleur HDMI avant de le tester sur la carte.
   C'est à vous de créer le testbench et le script de compilation (fichier `.do`).

   Le code de l'**`entity`** est donné en figure 1.

2. Analysez l'**`entity`** :
   — Quel est le rôle des différents paramètres définis en **`generic`** ?
   — Quel est leur unité ?

   Voici le rôle de certain des signaux :
   — `o_new_frame` : Passe à l'état haut pendant 1 cycle d'horloge quand une image a fini d'être transmise
   — `o_pixel_pos_x` et `o_pixel_pos_y` : Positions en X et Y des pixels sur la zone d'affichage active.
   — `o_pixel_address` : Adresse du pixel obtenue par la formule suivante : $\texttt{o\_pixel\_pos\_x} + (\texttt{h\_res} \times \texttt{o\_pixel\_pos\_y})$

3. Rappelez le rôle des autres signaux.

```vhdl
entity hdmi_generator is
    generic (
        -- Resolution
        h_res     : natural := 720;
        v_res     : natural := 480;

        -- Timings magic values (480p)
        h_sync    : natural := 61;
        h_fp    : natural := 58;
        h_bp    : natural := 18;

        v_sync    : natural := 5;
        v_fp    : natural := 30;
        v_bp    : natural := 9
    );
    port (
        i_clk       : in std_logic;
        i_reset_n   : in std_logic;

        o_hdmi_hs   : out std_logic;
        o_hdmi_vs   : out std_logic;
        o_hdmi_de   : out std_logic;

        o_pixel_en      : out std_logic;
        o_pixel_address : out natural range 0 to (h_res * v_res -
        ↪  1);
        o_x_counter     : out natural range 0 to (h_res - 1);
        o_y_counter     : out natural range 0 to (v_res - 1);
        o_pixel_pos_x   : out natural range 0 to (h_res - 1);
        o_pixel_pos_y   : out natural range 0 to (v_res - 1);
        o_new_frame     : out std_logic
      );
end hdmi_generator;
```

Figure 1 – **Entity** du composant `hdmi_generator`

2.1.1 Écriture du composant

Les questions suivantes vont vous guider dans l'écriture du générateur HDMI. Tous les codes seront à écrire dans le fichier `hdmi_generator.vhd`

1. Écrivez un compteur horizontal (`h_count`) qui boucle de 0 à `h_total`, et qui génère le signal de synchronisation horizontal (`o_hdmi_hs`).
2. Testez à l'aide d'un testbench. Vous pouvez réduire artificiellement les tailles d'images pour réduire le temps de simulation.
3. Ajoutez un compteur vertical (`v_count`) qui s'incrémente à chaque cycle du compteur horizontal, et boucle de 0 à `v_total`. Le compteur vertical doit également générer le signal de synchronisation vertical (`o_hdmi_vs`).
4. Adaptez le testbench pour tester l'ensemble.
5. Déterminez les plages de `h_count` et `v_count` où les pixels sont visibles (zone active) et implémentez les signaux `h_act` et `v_act` pour les indiquer.
6. Combinez `h_act` et `v_act` pour produire un signal `o_hdmi_de` indiquant si le générateur est dans une zone active (ligne active et pixel actif).
7. Ajoutez un compteur de pixels actifs (`r_pixel_counter`) pour générer une adresse pixel `o_pixel_address`, qui incrémente uniquement dans les zones actives. Réinitialisez-le au début de chaque nouvelle image.
8. Ajoutez deux compteurs pour générer les signaux `o_x_counter` et `o_y_counter` permettant de compter les pixels actifs.
9. Testez votre composant.

2.1.2 Implémentation sur le FPGA

Dans cette partie, vous complèterez les fichiers du projet quartus pour tester le générateur HDMI dans les conditions réelles.

1. Connectez simplement les signaux `o_x_counter` et `o_y_counter` à deux canaux de couleur (vert et bleu par exemple).
2. Testez et montrez à votre encadrant. Avec un adaptateur HDMI vers USB, vous pouvez visualiser la sortie avec VLC.

2.2 Bouncing ENSEA Logo

Dans cette partie, nous utiliserons une architecture *scanline rendering* pour afficher et déplacer le logo ENSEA. Le logo sera stocké dans une mémoire RAM. Pour aller plus vite, le code de la RAM initialisée est fourni sur Moodle.

1. Instanciez la mémoire dans votre projet
2. Écrivez le code permettant d'afficher l'image en haut à gauche de l'écran.
3. Testez et montrez à votre encadrant.

2.2.1 Déplacer le logo

1. Écrivez le code permettant de déplacer le logo.
2. Testez et montrez à votre encadrant.