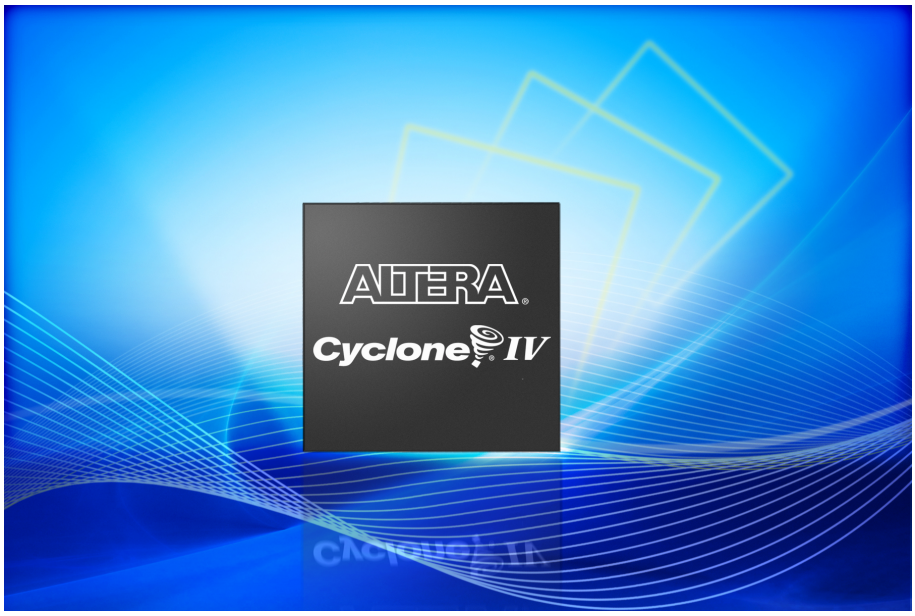
Université de Liège

ELEN0037 - Microélectronique

Année académique 2014 - 2015

Laboratoire FPGA



Remacle Matthieu, Schmitz Thomas, Pierlot Vincent

1 Introduction

1.1 Objectif et instructions

Le but de ce laboratoire est de vous familiariser avec les logiciels nécessaires pour la création de votre projet VHDL dans le cadre du cours de microélectronique, et de la carte de développement DE0-nano.

Au terme de ce laboratoire, il vous sera demandé de créer un rapport écrit, au format PDF, à remettre aux assistants. Ce rapport comprendra l'ensemble de vos réponses aux questions posées tout au long de ce laboratoire, vos codes VHDL et vos schémas.

1.2 Logiciels requis

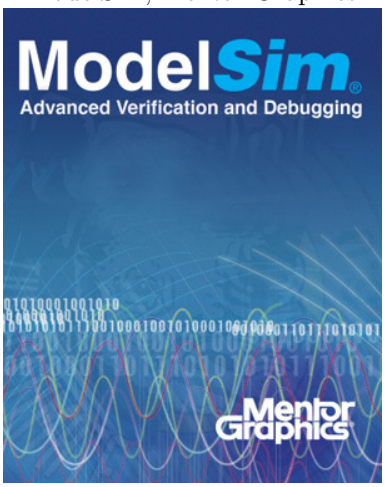
Deux logiciels sont indispensables pour le laboratoire. Ils sont installés sur les machines du R100, mais vous devrez les installer par vous même sur vos machine pour le travail VHDL :

Quartus II, Altera

Version : Web edition

Lien : https://www.altera.com/download/software/quartus-ii-we

ModelSim, Mentor Graphics



Version : Altera Edition

Lien : https://www.altera.com/download/software/modelsim-starter

Il existe une version Linux, en cas de problème lors de l'installation adressez vous aux assistants

1.3 Conventions

Les encadrés suivants seront utilisés au cours de ces notes :

Remarque importante

Remarque très importante, qui doit être impérativement respectée !

Question devant être résolue dans le rapport

2 Considérations théoriques

2.1 Conception d'un programme VHDL

La conception d'un programme VHDL peut être résumée par le diagramme de la figure 1 :

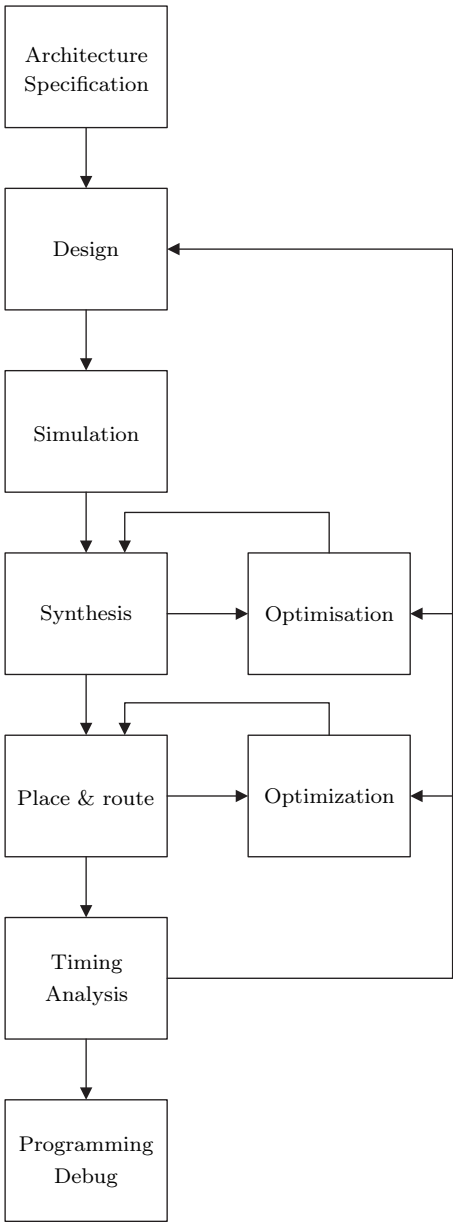


Figure 1 – Diagramme d'un développement d'un projet VHDL

Passons en revue les différentes étapes reprises dans ce diagramme :

Architecture specification : Cette première étape est la plus importante. Elle consiste à écrire les spécifications du programme, ou en d'autres termes, écrire l'ensemble des fonctions que le

programme devra être capable de réaliser. Cette spécification sera très utile pour le debug, car elle permet au final de créer un ensemble de scénarios test que le programme devra pouvoir effectuer sans encombre.

La création de l'architecture consiste à réfléchir à l'organisation du code, et aux ressources nécessaires. En particulier, on commence généralement par la création d'une machine d'état complète, ainsi qu'une liste des entrées/sorties (I/O). Les différents signaux et ressources nécessaires doivent aussi faire l'objet d'une réflexion. À la fin de cette étape, il doit être possible d'avoir une première idée de l'ensemble des ressources nécessaires (pins (pattes) I/O, registres, etc...) à l'implémentation du programme.

C'est grâce à cette réflexion sur l'architecture du programme qu'il sera possible de déterminer sa faisabilité et/ou déterminer le adapté au programme. Dans le cadre de ce laboratoire, et par extension de votre projet, le composant vous est imposée. Il est donc de votre ressort d'évaluer correctement les ressources requises afin de savoir si votre programme est réalisable. Si ce n'est pas le cas, il faut revoir les spécifications à la baisse.

Design (programme) : Cette étape reprend la phase de programmation à proprement parlé. Il convient de suivre scrupuleusement l'architecture que vous avez définie dans l'étape précédente, afin de ne pas utiliser des ressources inutiles.

Simulation : Cette étape, bien que facultative, vous permet de faire un premier debug de votre logiciel, afin d'éliminer les erreurs de conception. A cette étape, il est déjà possible de vérifier si le programme répond aux spécifications de votre projet, via un ensemble de scénarios de test. Le logiciel utilisé au cours de ce laboratoire pour cette étape est ModelSim.

Synthesis : Votre programme créé et simulé, il vous reste à l'implémenter physiquement. La première étape de ce processus est la synthèse, c'est à dire la traduction de votre code VHDL en équations booléennes implémentables dans le composant. Grâce à Quartus II, il vous est possible de réaliser cette étape en un clic. A l'issue du processus, un rapport vous sera présenté avec, entre autres, les ressources nécessaires à l'implémentation du programme. Si ces ressources sont trop importantes pour votre composant, il conviendra d'optimiser votre code pour diminuer la consommation, voir diminuer les spécifications.

Place & Route : Votre programme a été traduit en équations, mais il n'est pas encore mis sous une forme programmable dans le composant. Il faut en effet encore décider de l'emplacement des différents flip-flops utilisés, des signaux de contrôle, etc... Ceci est réalisé, toujours par le logiciel Quartus II, durant cette étape. L'analyse est généralement relativement précise et il est rare qu'un programme qui est analysable ne soit pas implémentable. Toutefois, il est possible que cela arrive. Il convient alors de réaliser encore quelques optimisations si tel est le cas.

Timing analysis : Pour les programmes dont la vitesse est un élément essentiel, il faut également analyser les timings, c'est-à-dire le temps mis par un signal pour passer de l'entrée de le composant à sa sortie (entre autre). Cette étape ne s'appliquera toutefois pas à vos programmes dans le cadre de ce cours.

Programming & debug : Dernière étape : placer le programme sur le composant et vérifier que tout se passe bien. Si la simulation a été bien réalisée, le programme devrait fonctionner du premier coup...

Le composant utilisée pour ce laboratoire est un Cyclone IV EP4CE22F17C6N. Recherchez dans le datasheet les ressources dont vous disposez pour créer votre programme. En particulier, combien d'éléments logiques, combien d'entrées/sorties, etc, disposez vous ?

2.2 Écrire du code propre

Chaque programmeur à son propre style pour coder, mais il convient de respecter certains points clés afin de garder un code propre. Ceci permet de diminuer les erreurs, et de faciliter la relecture. Les commentaires sont également très importants ! Voici une liste, non exhaustive, d'éléments à respecter :

- Indentation du code,
- Choix d'une convention de nomenclature (par exemple : les types commencent par une minuscule et les variables par une majuscule, etc...),
- Choix de noms cohérents et lisibles,
- Commentaire pour chaque élément un peu compliqué,
- etc

Voici un exemple de code lisible. Il correspond à nos choix de nomenclature, mais rien ne vous empêche d'utiliser les vôtres.

```
1  library ieee ;
2  use ieee.std_logic_1164.all ;
3
4  entity cardio is
5    port
6    ( -- Input ports
7      clk       : in  std_logic ;
8      heart     : in  std_logic ;
9      -- Output ports
10     led_fast  : buffer std_logic ;
11     led_slow  : buffer std_logic -- no semicolon!
12   ) ;
13 end entity cardio ;
14
15 architecture cardio_arch of cardio is
16   signal cnt       : integer range 0 to 127 := 0 ;
17   signal heart_old : std_logic := '0' ;
18 begin
19   counter : process( clk )
20   begin
21     if( rising_edge( clk ) ) then
22       if( heart_old = '1' and heart = '0' ) then
23         -- do something!
24         cnt <= 0 ;
25       else
26         if( cnt /= 127 ) then
```

```
27            cnt <= cnt + 1 ;
28          end if ;
29        end if ;
30        heart_old <= heart ;
31      end if ;
32    end process counter ;
33 end architecture cardio_arch ;
```

3 Affichage d'un rectangle rouge sur un écran VGA

3.1 Spécifications

Les spécifications de ce premier programme sont très simples : il s'agit de générer un signal VGA valide de résolution 800×600, et affichant un petit carré rouge au centre de l'écran.

Il n'y a donc qu'un scénario de test extrêmement simple : vérifier que le système génère les bons signaux VGA. En outre, la seule entrée au système est l'horloge cadencée à 50 [MHz]. Il en résultera une simulation très simple.

3.2 Fonctionnement de l'interface VGA

Le protocole VGA est un protocole analogique, utilisé depuis longtemps pour contrôler les écrans CRT (Cathode Ray Tube). Cet aspect est à garder en mémoire pour comprendre les besoins de cette interface : il faut en effet envoyer des données pour une région supérieure à celle requise pour l'affichage, ainsi que des signaux de contrôles de synchronisation verticale et horizontale, qui sont utilisés pour replacer le rayon en position initiale.

La figure 2 illustre les signaux de synchronisation, ainsi que la région active (ce qui sera vraiment visible) et la région inactive, appelée Blanking time.

Ces signaux doivent respecter des timings stricts. Par exemple, pour du 800×600 @ 72 [Hz] :

Caractéristiques générales :
- Horloge pixel : 50 [MHz]
- Fréquence de ligne : 48 077 [Hz]
- Fréquence d'image : 72 [Hz]

Timings pour une ligne :
- Région active : 800 pixels
- Front Porch : 56 pixels
- Sync Pulse : 120 pixels
- Back Porch : 64 pixels

Timings pour l'image :
- Région active : 600 lignes
- Front Porch : 37 lignes
- Sync Pulse : 6 lignes
- Back Porch : 23 lignes

Chaque pixel est codé par trois tensions analogiques, donnant les intensités des sous pixels rouges, verts, et bleus.

Dans ce laboratoire, nous n'allons pas utiliser de DAC, et donc envoyer uniquement des valeurs de type ON/OFF. Dès lors, seules les couleurs suivantes sont accessibles (les chiffres entre parenthèses sont les triplets de valeurs R,G,B) :
- Noir (0,0,0)
- Rouge (1,0,0)
- Vert (0,1,0)
- Bleu (0,0,1)
- Jaune (1,1,0)
- Magenta (0,1,1)
- Cyan (1,0,1)
- Blanc (1,1,1)

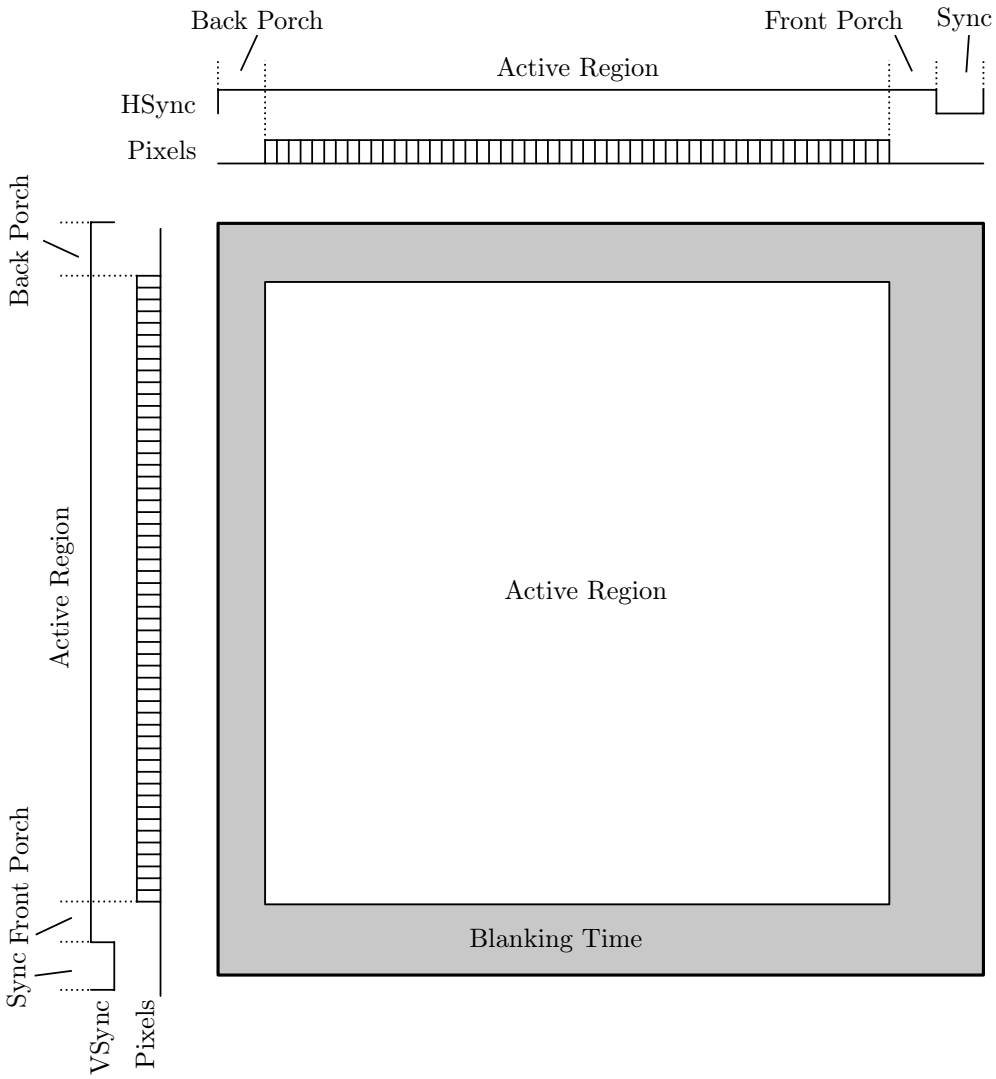


Figure 2 – Signaux VGA

3.3 Architecture

De manière tout à fait générique, pour une image codée en vraies couleurs (24bits), il faudrait générer l'image et l'enregistrer, ce qui demanderait une taille égale à

$$800 \times 600 \times 3 = 1.44\,[\text{MB}]$$

La FPGA dispose à son bord d'un peu moins de 600 [Kbits] de mémoire, ce qui est donc loin d'être suffisant. Bien qu'il soit également possible de créer des cellules mémoires à partir de LE (Logic Element), ceci diminuerait la puissance de calcul disponible, et donc une meilleure solution est à envisager. En revanche, la carte en elle-même embarque une mémoire SDRAM de 32 [MBytes], permettant ainsi de stocker plusieurs images.

Dans le cadre de ce premier programme, la situation est beaucoup plus simple : puisque l'on crée un carré de couleur uniforme, il suffit de détecter le moment où l'on rentre dans la zone du carré (en regardant où le pixel en cours d'envoi se situe), et de mettre la sortie Rouge à 1. Dans le cas contraire, on la laisse à 0.

Pour cette réalisation, un seul processus sera nécessaire. Il sera exécuté à 50 [MHz], et donc à chaque coup d'horloge de la carte. Des compteurs permettront de comptabiliser le numéro de ligne et le numéro de pixel sur la ligne, et à partir de là, les signaux de synchronisation seront générés.

3.4 Programme

Le code suivant effectue les opérations demandées :

```
1  library ieee ;
2  use ieee.std_logic_1164.all ;
3  use ieee.std_logic_ARITH.all ;
4  use ieee.std_logic_UNSIGNED.all ;
5
6  entity vga is
7  Port (
8        CLOCK_50            : in std_logic ;
9        GPIO_1_D            : out std_logic_vector( 33 downto 0 )
10       ) ;
11 end entity vga;
12
13 architecture vga_arch of vga is
14       --Sync Signals
15   signal h_sync           : std_logic ;
16   signal v_sync           : std_logic ;
17       --Video Enables
18   signal video_en         : std_logic ;
19   signal horizontal_en    : std_logic ;
20   signal vertical_en      : std_logic ;
21       --Color Signals
22   signal red_signal       : std_logic ;
23   signal green_signal     : std_logic ;
24   signal blue_signal      : std_logic ;
25       --Sync Counters
26   signal h_cnt    : std_logic_vector(10 downto 0) := (others => '0') ;
27   signal v_cnt    : std_logic_vector(10 downto 0) := (others => '0') ;
28
29 begin
30
31   vga_gen : process
32   begin
33
34   wait until( ( CLOCK_50'event ) and ( CLOCK_50 = '1' ) ) ;
35
36   --Generate Screen
37   if ( v_cnt >= 0 ) and  ( v_cnt <= 799 ) then
38     red_signal <= '0' ;
39     green_signal <= '0' ;
```

```
40      blue_signal <= '0' ;
41        if ((( v_cnt >= 200 ) and ( v_cnt <= 400 )) and (( h_cnt >= 300 )
              and ( h_cnt <= 500 ))) then
42          red_signal <= '1' ;
43        end if ;
44    end if ;
45
46    --Generate Horizontal Sync
47    if ( h_cnt <= 975 ) and ( h_cnt >= 855 ) then
48      h_sync <= '0' ;
49    else
50      h_sync <= '1' ;
51    end if ;
52
53    --Reset Horizontal Counter
54    if ( h_cnt = 1039 ) then
55      h_cnt <= "00000000000" ;
56    else
57      h_cnt <= h_cnt + 1 ;
58    end if ;
59
60    --Reset Vertical Counter
61    if ( v_cnt >= 665 ) and ( h_cnt >= 1039 ) then
62      v_cnt <= "00000000000" ;
63    elsif ( h_cnt = 1039 ) then
64      v_cnt <= v_cnt + 1 ;
65    end if ;
66
67    --Generate Vertical Sync
68    if ( v_cnt <= 642 ) and ( v_cnt >= 636 ) then
69      v_sync <= '0' ;
70    else
71      v_sync <= '1' ;
72    end if ;
73
74    --Generate Horizontal Enable
75    if ( h_cnt <= 799 ) then
76      horizontal_en <= '1' ;
77    else
78      horizontal_en <= '0' ;
79    end if ;
80
81    --Generate Vertical Enable
82    if ( v_cnt <= 599 ) then
83      vertical_en <= '1' ;
84    else
85      vertical_en <= '0' ;
```

```
86    end if ;
87
88    video_en <= horizontal_en and vertical_en ;
89
90    --Assign Physical Signals To VGA
91    GPIO_1_D(1) <= red_signal   and video_en ;
92    GPIO_1_D(3) <= green_signal and video_en ;
93    GPIO_1_D(5) <= blue_signal  and video_en ;
94    GPIO_1_D(7) <= h_sync ;
95    GPIO_1_D(9) <= v_sync ;
96
97  end process vga_gen ;
98
99  end architecture vga_arch ;
```

Essayez de bien comprendre le code, nous y reviendrons plus tard

4 Création, simulation et programmation d'un projet

4.1 Découverte de Quartus II

Quartus est la suite logicielle développée par Altera. Elle vous permettra de réaliser, au sein d'une interface unique, toutes les tâches envisageables sur un composant logique programmable de type CPLD/FPGA :

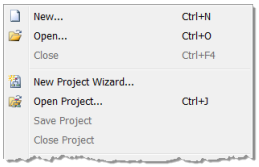
- Ecriture du programme via non seulement du VHDL ou Verilog, mais aussi via des machines d'états, un schéma ou encore des blocs diagrammes, etc,
- Génération des fonctions logiques à partir des sources, avec la génération de nombreux rapports,
- Programmation du composant,
- Analyses diverses et variées : consommation, temps de propagation, ressources, etc.

En outre, de nombreux blocs sont disponibles : le processeur NIOS (codable en C), vous permettant d'implémenter un processeur classique sur FPGA, différents blocs comme des PLL's, des fonctions DSP, des interfaces diverses, etc.

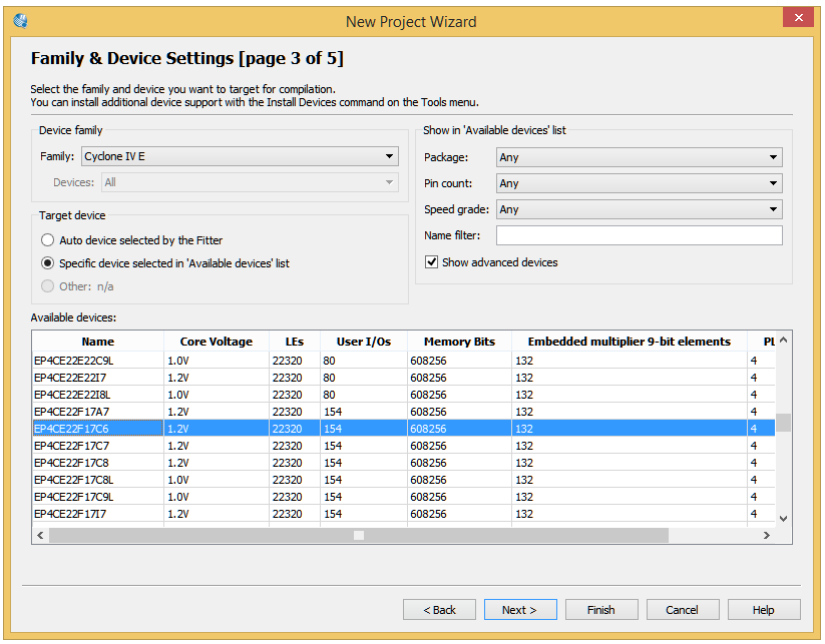
Interface

Ce logiciel fonctionne avec la notion de Projet. Un projet reprend l'ensemble des codes sources, fichiers de programmation, fichiers de contraintes, etc, nécessaires pour la programmation d'un composant. Vous aurez donc un seul projet pour votre laboratoire. La création d'un projet est relativement simple :

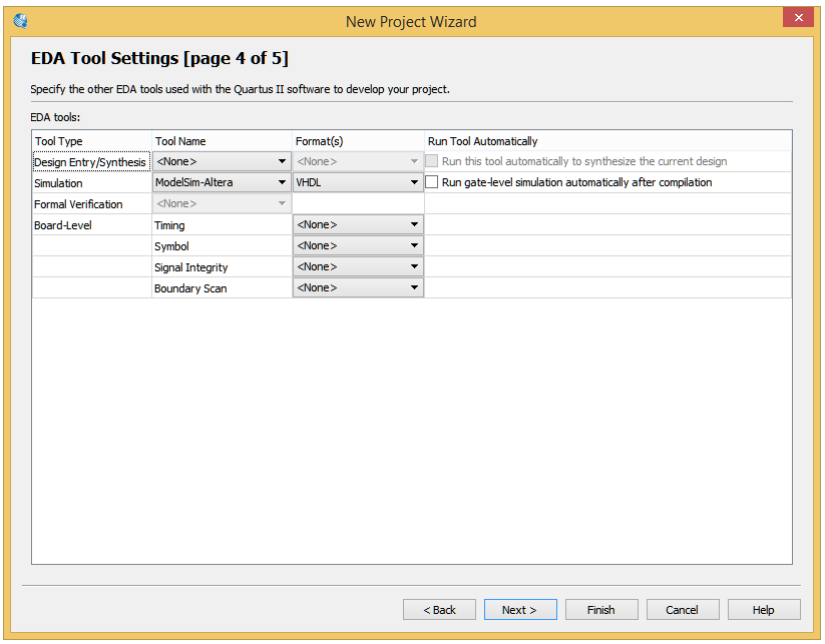
- Une fois le logiciel démarré, cliquez sur File → New Project Wizard...



- Une nouvelle fenêtre s'ouvre. En cliquant sur suivant, vous pouvez configurer les différentes variables de votre projet, à commencer par l'emplacement des fichiers, et le nom (mettez ici vga). Ensuite, vous pouvez sauter l'étape d'ajout de fichiers (nous en créerons un plus tard) pour passer à la configuration du composant, un EP4CE22F17C6 :



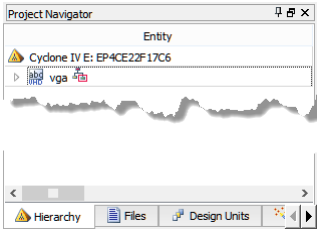
— Enfin, dans la partie Simulation, choisissez ModelSim Altera :



Le projet étant créé, vous avez maintenant accès à la fenêtre principale de Quartus II. Cette fenêtre reprend un certain nombre d'informations :

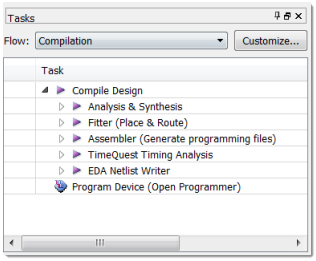
Navigateur de projet : Cette fenêtre affiche un résumé de la hiérarchie du projet. Dans le cadre de ce laboratoire, la hiérarchie restera simple, puisque nous n'aurons qu'un seul fichier.

Dans cette fenêtre, il est également possible de passer en revue les différents fichiers du projet, etc.

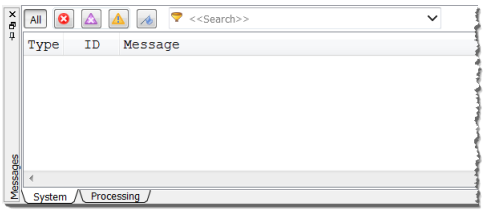


Tâches : Cette fenêtre fournit l'ensemble des informations concernant les tâches à effectuer pour avoir un projet fonctionnel. Chaque tâche (par exemple : la synthèse) peut être effectuée indépendamment en double-cliquant sur celle-ci. Si une tâche nécessite qu'une autre tâche soit effectuée, elle le sera également.

Enfin, il est possible via cette fenêtre d'accéder aux rapports de compilation, vous indiquant ce qui s'est produit lors de l'exécution d'une tâche. Pour ce faire, il suffit de cliquer sur la petite flèche afin de dérouler la tâche, et cliquer sur un des rapports proposés.



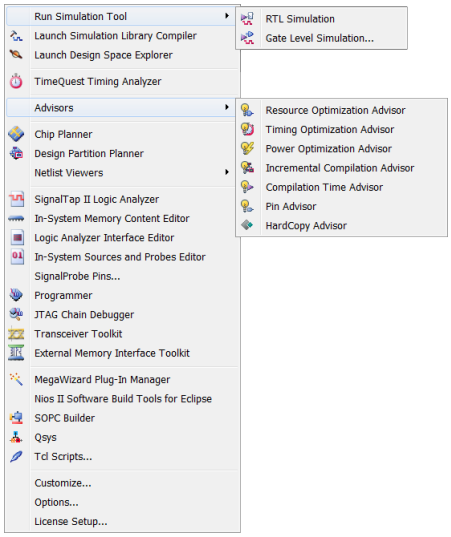
Messages : L'ensemble des messages générés (par exemple : les erreurs de compilation, etc) seront affichés ici.



Outils : Les outils importants, tels que le lancement d'une compilation, d'une analyse de timing, ou encore le pin planner se trouvent dans cette barre.

Si une des fenêtres n'est pas disponible sur votre écran, vous pouvez la faire apparaître en cliquant sur View → Utility Windows et en cliquant sur l'élément indisponible.

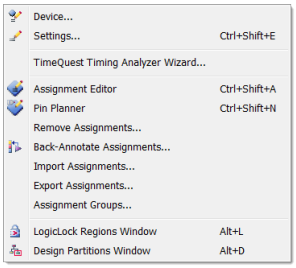
Dans la barre supérieure, d'autres outils sont disponibles sous la partie Tools :



Ainsi, c'est ici que vous trouverez, entre autre :

— Le programmateur,
— L'analyseur de timing,
— Le Plugin-in Manager,
— Les advisors,
— Le lancement d'une simulation.

Enfin, on peut aussi parler du menu Assignments, qui reprend tout ce qu'il vous faudra pour attribuer les entrées/sorties de votre programme, ainsi que les options du projet :



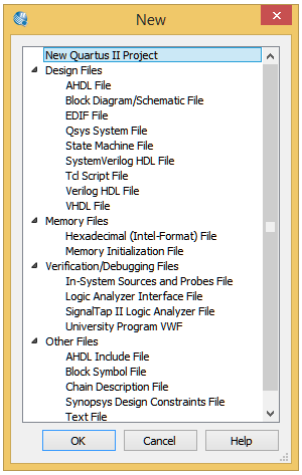
Vous noterez pour finir que les fonctionnalités de la barre d'outils sont présentes dans les menus déroulants.

Configuration de Quartus II

L'intégralité de ces étapes se passe dans le logiciel Quartus II. Toutefois, il est nécessaire de le configurer pour lui dire d'utiliser ModelSIM pour la simulation. Ceci peut être fait en allant sur Tools → Options. Dans la fenêtre qui s'affiche, allez sur l'onglet EDA Tool Options, et vérifiez que le chemin de ModelSim-Altera est défini. Si ce n'est pas le cas, définissez-le.

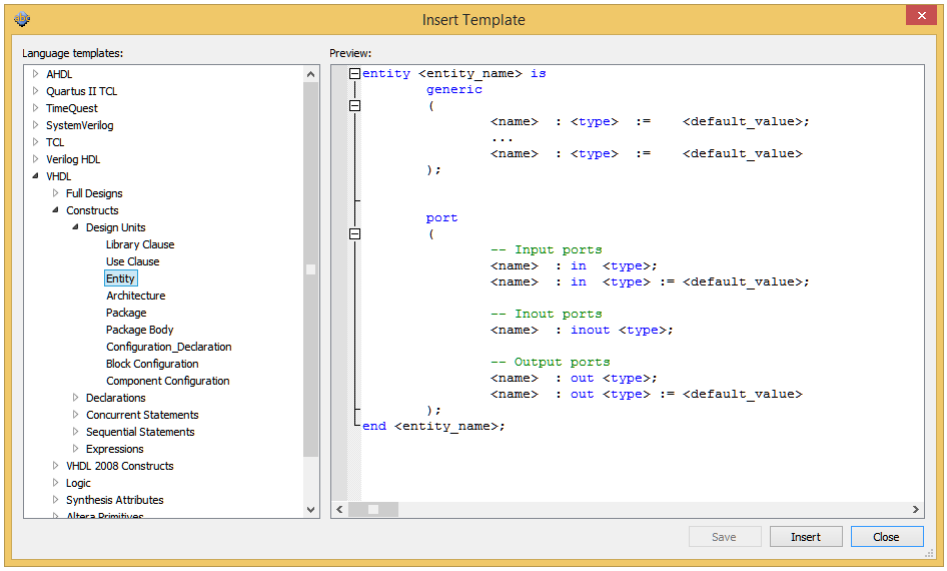
Edition du fichier VHDL

Il est maintenant temps de créer notre carré rouge. Pour ce faire, cliquez sur File → New... et dans la fenêtre qui s'affiche, sélectionnez VHDL file.



Un fichier vide s'ouvre dans l'espace de travail, il ne vous reste plus qu'à réécrire le code d'exemple créé un peu plus tôt. Plutôt que de copier le programme, nous allons le réécrire. Quartus II fourni en effet une série de Templates permettant de faciliter l'écriture de code VHDL. Vous pouvez y accéder via le bouton . Dans la nouvelle fenêtre qui s'ouvre, choisissez VHDL → Constructs → Design Units → Entity.

Appuyez ensuite sur Insert et la construction est ajoutée à votre fichier VHDL. Il ne vous reste plus alors qu'à remplacer les éléments entre <> à votre guise. N'oubliez pas que vous pouvez aussi utiliser la fonction Find & Replace (via CTRL-H) de l'éditeur de texte pour aller plus vite.



Sauvegardez votre projet. Une fois votre entité recopiée, vous pouvez vérifier que ce que vous avez écrit est correct, via le bouton . Si la compilation est réussie, un message s'affichera dans la fenêtre de message et un rapport de compilation (vide) dans l'espace de travail. Finalement, recopiez l'intégralité du code, et vérifier que la syntaxe est correcte.

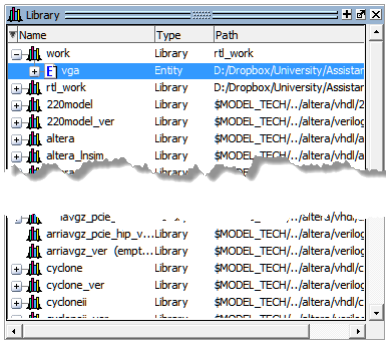
Votre fichier étant syntaxiquement correct, il est maintenant temps de lancer les étapes d'analyse et de simulation en cliquant sur le petit triangle mauve.

4.2 Découverte de ModelSim

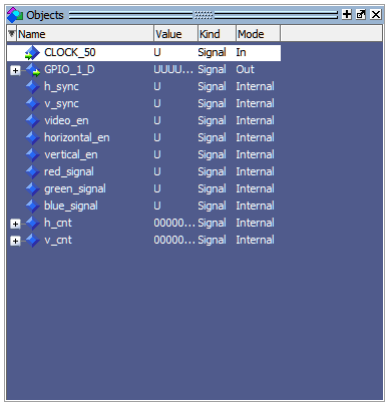
Interface

Afin de vérifier le fonctionnement du système, il est possible de le simuler. Pour ce faire, appuyez sur le bouton  , qui lancera le programme ModelSim avec tous les paramètres préchargés. Tout comme pour Quartus, l'espace de travail est découpé en différentes zones, les plus importantes étant celles-ci :

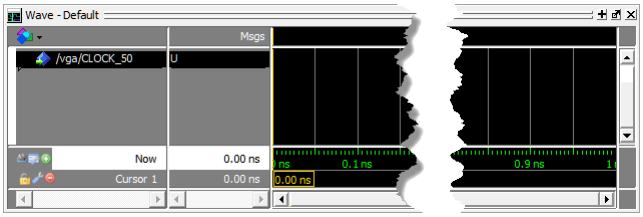
Bibliothèques : Toutes les libraires nécessaires sont reprises dans cette fenêtre. En particulier, votre programme se trouve dans la librairie Work. Double-cliquez sur le nom de votre programme pour lancer la simulation.



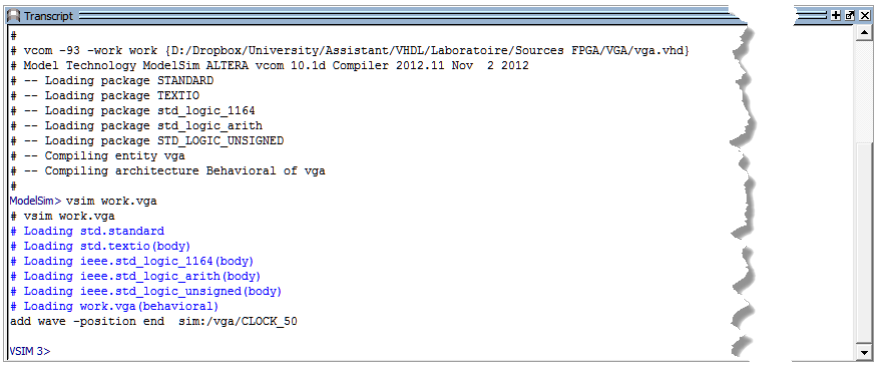
Objets : Tous les signaux disponibles apparaîtrons dans cette fenêtre. Vous pouvez, avec la souris, les glisser-déposer dans la fenêtre Wave pour afficher le résultat de la simulation de ces signaux.



Wave : Tous les résultats de simulation demandés sont accessibles dans cette fenêtre. Ici, l'objet `CLOCK_50` a été déplacé (Si la fenêtre n'apparaît pas, faites View → Wave).



Console : Il est possible de piloter le logiciel à l'aide d'une console, c'est-à-dire en lui entrant une série de commandes. Les messages, du même type que ceux de Quartus, sont affichés ici.



Lorsque vous avez double-cliqué sur votre programme dans la fenêtre Libraries, vous avez démarré une simulation. En réalité, votre simulation a été initialisée, et se trouve au temps 0, mais il ne s'est encore rien passé. En effet, il faut tout d'abord donner au logiciel des stimuli, c'est-à-dire déterminer les entrées du système. Avec ModelSim, il y a plusieurs façons de travailler. En voici 3 :

— Lancer la simulation sans aucune information, puis lui donner les stimuli au fur et à mesure ;
— Lancer la simulation avec des stimuli généré graphiquement à l'avance ;
— Lancer la simulation avec un fichier de TestBench, qui reprend l'ensemble des stimuli.

Dans ce laboratoire, nous verrons la première et la dernière possibilité. En effet, le fichier de Testbench est en réalité beaucoup plus court et simple à écrire que de créer des stimuli à la main. La première possibilité peut être pratique pour débuguer une situation étrange : on pose quelques stimuli, on stoppe la simulation, on regarde ce qu'il se passse, puis on impose d'autre stimuli, on relance la simulation, etc. Voyons d'abord cette première façon de faire.

Lancement d'une simulation

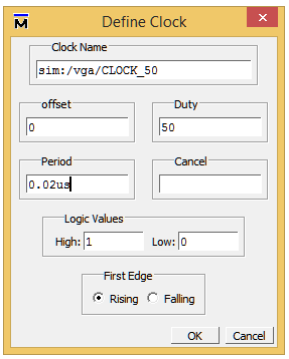
Vous utiliserez principalement deux possibilités :

— Stimulus de type horloge,
— Stimulus de type valeur forcée.

Une fois votre simulation démarrée, commencez tout d'abord par déplacer tous les objets dans la fenêtre Wave, ce qui va permettre de visualiser tous les signaux. Remarquez que les entrées, les sorties, mais également les signaux internes sont disponibles, ce qui est très pratique pour déboguer une situation problématique.

Ensuite, faites un clic droit sur l'élément `CLOCK_50`, puis dans le menu qui apparaît, choisissez Clock :

Dans le champ Period, indiquez 0.02us.



Enfin, démarrez la simulation pour une durée de 20ms. Vous pouvez le faire soit en saisissant dans le champ approprié une durée puis en cliquant sur le bouton :

Soit en entrant `run 20 ms` dans la console. Vous pouvez zoomer sur le résultat avec O et I.

Grâce à ces outils, simulez votre carré rouge. Est-ce que le signal est bien conforme à vos attentes ?

Les Testbenches

Maintenant, nous allons voir la seconde possibilité de simulation : l'utilisation de fichiers de Testbench. Il s'agit de fichiers VHDL, indiquant quelle entité est testée, et exécutant différents processus de test. L'unité testée est appelée Unit Under Test (UUT), ou encore Device Under Test (DUT). Le code suivant reprend un fichier pour tester notre entité vga :

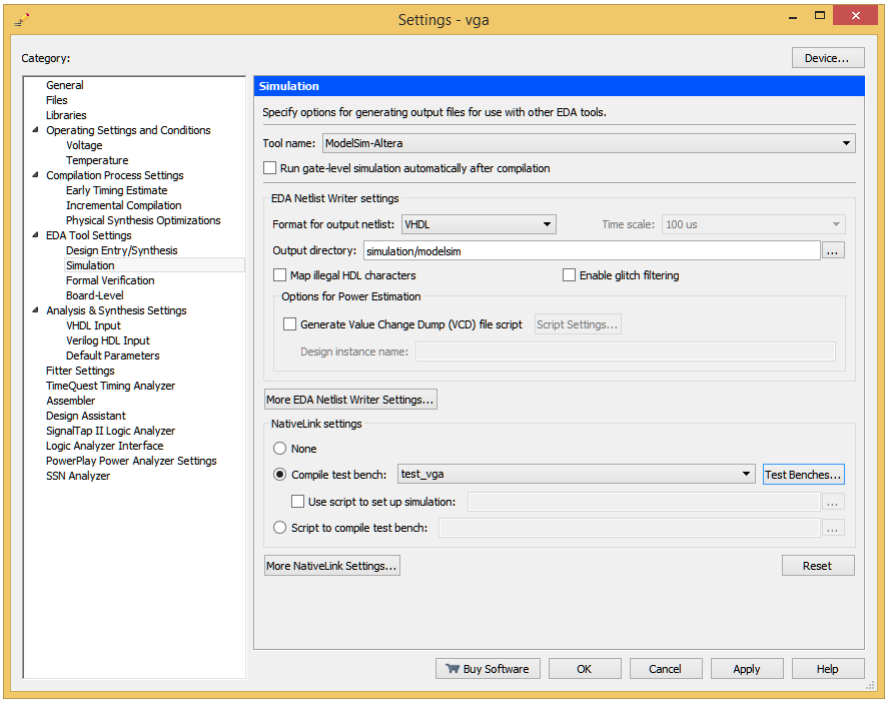
```
1  library ieee ;
2  library std ;
3  use ieee.std_logic_1164.all ;
4  use ieee.std_logic_textio.all ;
5  use ieee.std_logic_unsigned.all ;
6  use std.textio.all ;
7
8  entity test_vga  is
9  end ;
10
11 architecture test_arch of test_vga is
12   signal CLOCK_50 : std_logic ;
13   signal GPIO_1_D : std_logic_vector( 33 downto 0 ) ;
14   constant frames : integer := 2 ;
15
16   -- Description of vga
17   component vga
18   port (
19     CLOCK_50          : in std_logic ;
```

```
20      GPIO_1_D          : out std_logic_vector( 33 downto 0 )
21      ) ;
22    end component;
23
24    -- Begining of the architecture: port map
25    begin
26    DUT   : vga
27      port map (
28      CLOCK_50      => CLOCK_50,
29      GPIO_1_D      => GPIO_1_D
30      ) ;
31
32    -- Processes declaration
33    clk_stimulus: process
34    begin
35      for i in 1 to frames * 833334 loop
36        CLOCK_50 <= '0' ;
37        wait for 0.01us ;
38        CLOCK_50 <= '1' ;
39        wait for 0.01us ;
40      end loop ;
41      wait ;
42
43    end process clk_stimulus ;
44
45  end architecture test_arch ;
```

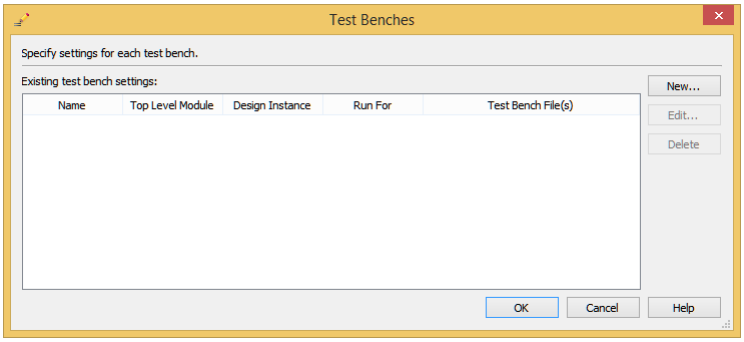
La structure du fichier est la suivante :

- Déclaration d'une entité vide pour la simulation,
- Déclaration des signaux internes qui correspondent aux entrées et sorties de la DUT,
- Connection de la DUT au sein de notre entité de simulation,
- Déclaration de processus de tests.

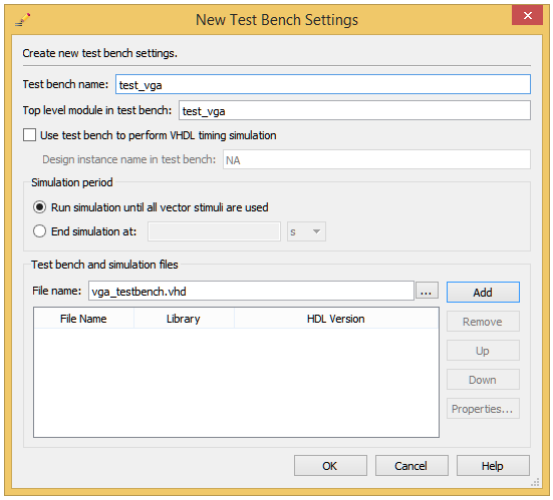
Il ne reste plus qu'à ajouter le fichier décrit ci-dessus dans notre projet. Pour ce faire, créez un nouveau fichier VHDL, collez-y le code et enregistrez le. Ensuite, dans Quartus, faites Assignements → Settings. Dans la fenêtre qui s'ouvre, choisissez la partie Simulation sous EDA Tools Settings :



Ensuite, dans la partie NativeLink settings, cochez Compile test bench, et cliquez sur le bouton Test Benches.

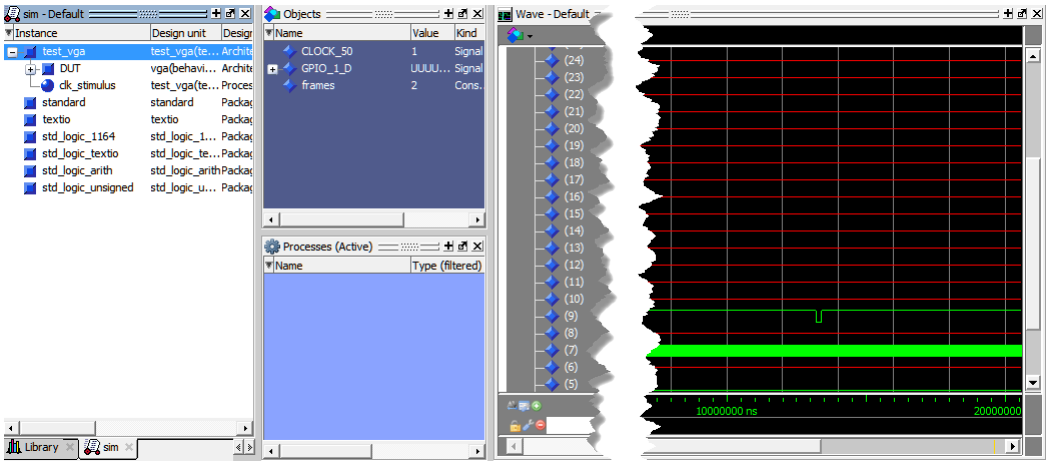


Dans la fenêtre qui s'ouvre, cliquez sur New... pour enfin obtenir ceci :



Indiquez, en suivant l'exemple de la capture d'écran, un nom pour le Testbench. Cochez la durée de la simulation (jusqu'à la fin, ou jusqu'à une certaine période), et enfin, grâce au bouton ..., allez chercher votre fichier de test. N'oubliez pas de cliquer sur Add.

Revenez ensuite à la fenêtre principale de Quartus en sauvant bien vos données. Cliquez ensuite sur . La simulation démarre et, après avoir ajusté le zoom dans la fenêtre Wave grâce aux touches O et I, vous devriez obtenir ceci :



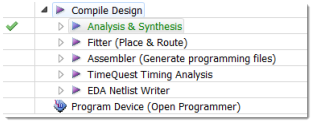
Comme le montre la capture, votre simulation a été générée avec les informations contenues dans le fichier de Testbench. Ici toutefois, toutes les I/O's ne sont pas utilisées, et seulement 3 changent d'état, ce qui est correct.

Votre système est maintenant simulé, et fonctionne comme vous le souhaitez. Vous pouvez donc passer à l'étape de synthèse et de programmation ! Quittez donc ModelSim.

4.3 Analyse et Place & Route du fichier

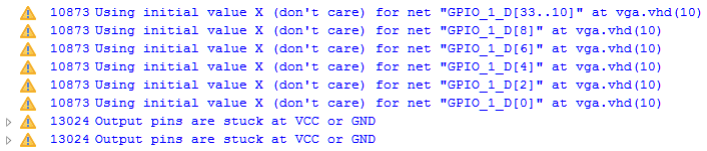
Analyse

Durant cette étape, le logiciel va analyser votre fichier VHDL et le traduire en éléments logiques implémentables dans la FPGA. Pour démarrer l'analyse, il suffit de cliquer sur Analysis & Synthesis dans la fenêtre de tâches. Si cette opération est réussie, vous devriez voir un petit ✔, comme ceci :



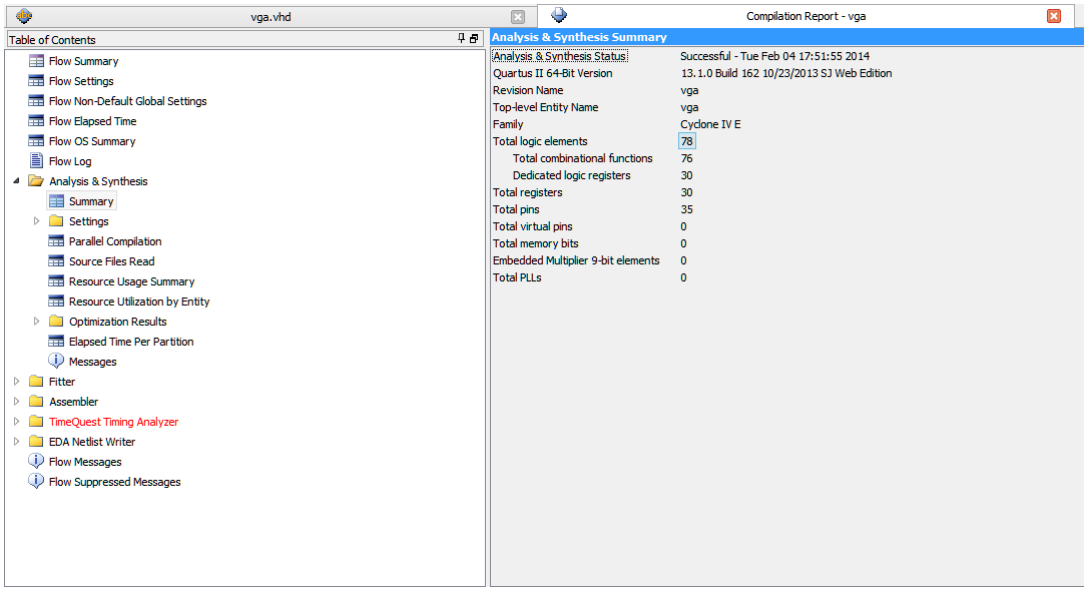
En outre, une fenêtre de rapport de compilation s'est ouverte dans votre espace de travail. Si ce n'est pas le cas, vérifiez à nouveau votre code VHDL.

Dans la fenêtre de messages, vous devriez voir un grand nombre d'informations. Cliquez sur le bouton pour n'inclure que les warnings. Vous devriez voir ceci :



Expliquez la cause de tous les warnings.

Une fois le fichier analysé, un grand nombre d'informations sur ce qui a été réalisé est maintenant disponible. Vous pouvez accéder au rapport de compilation, s'il n'est pas déjà ouvert, avec un clic droit sur Analysis & Synthesis puis View Report :



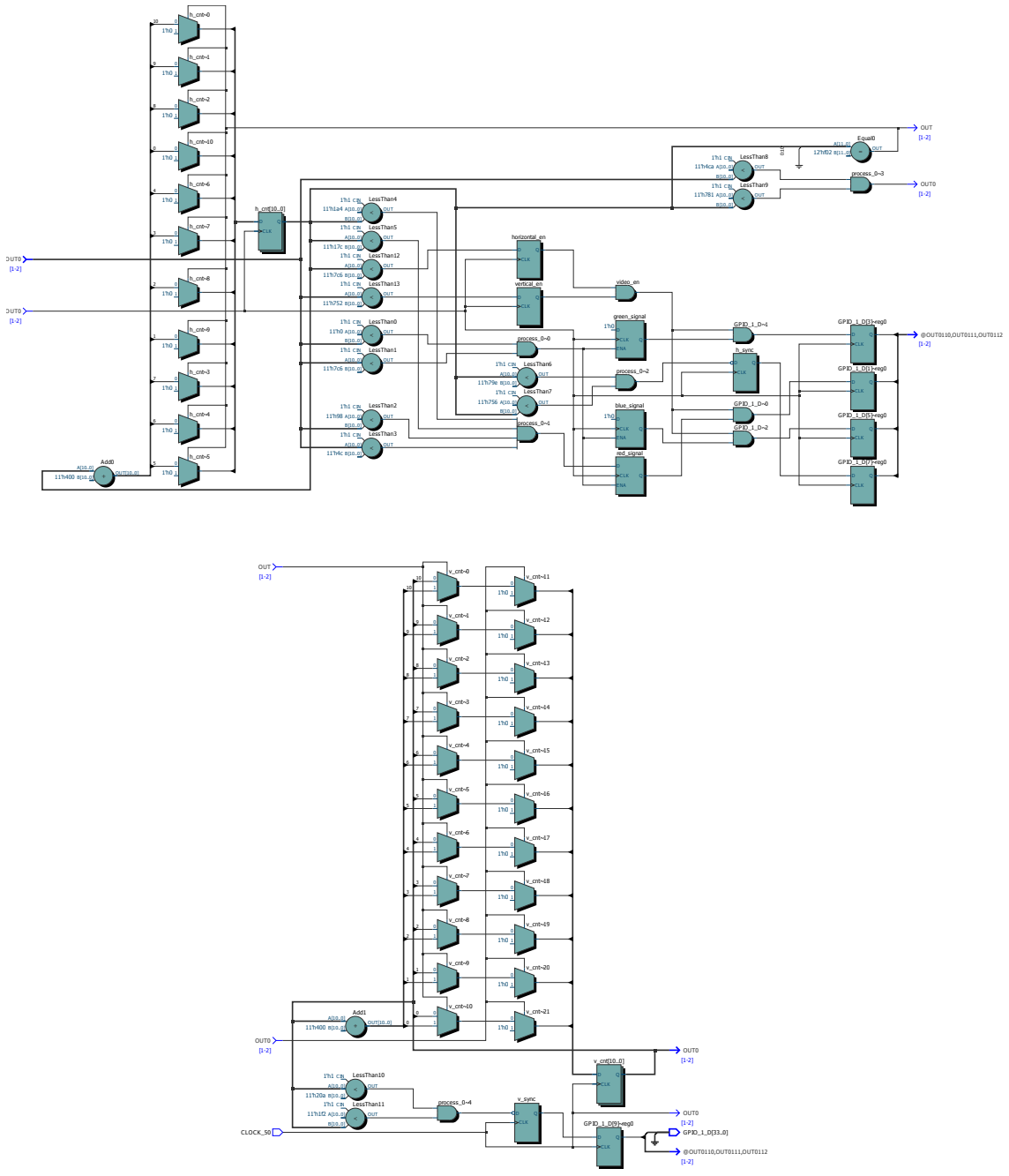
Cette fenêtre reprend deux informations capitales :

— Le nombre d'éléments utilisés par votre programme,
— Le nombre d'I/O's de votre système.

Ces deux nombres doivent bien entendu être compatibles avec le matériel dont vous disposez ! Sinon il faudra modifier votre code pour utiliser moins de ressources. Vous pouvez aussi, par curiosité, aller voir les autres parties du rapport, comme le Ressource Usage Summary.

Parmi les autres informations disponibles, il est possible de voir l'implémentation physique du système. Allez dans Tools → Netlist Viewers → RTL Viewer :

Attention, il peut y avoir deux pages.



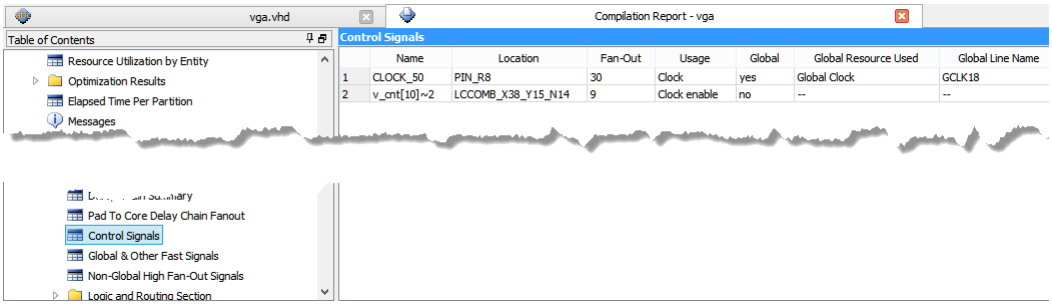
Vous pouvez ici voir la traduction de votre code en terme d'utilisation de registres, multiplexeurs, etc.

Cette fonction est très utile. Grâce aux différents outils de cette fenêtre, essayez de trouver dans le code VHDL les lignes faisant référence (la première ligne suffit) aux différents registres.).

La dernière étape consiste à assigner les différents signaux de votre système à des entrées/sorties précises du composant. Seules certaines entrées/sorties de la FPGA sont accessibles sur la carte de développement. Vous devrez donc placer toutes vos signaux sur ces entrées/sorties. Il faudra

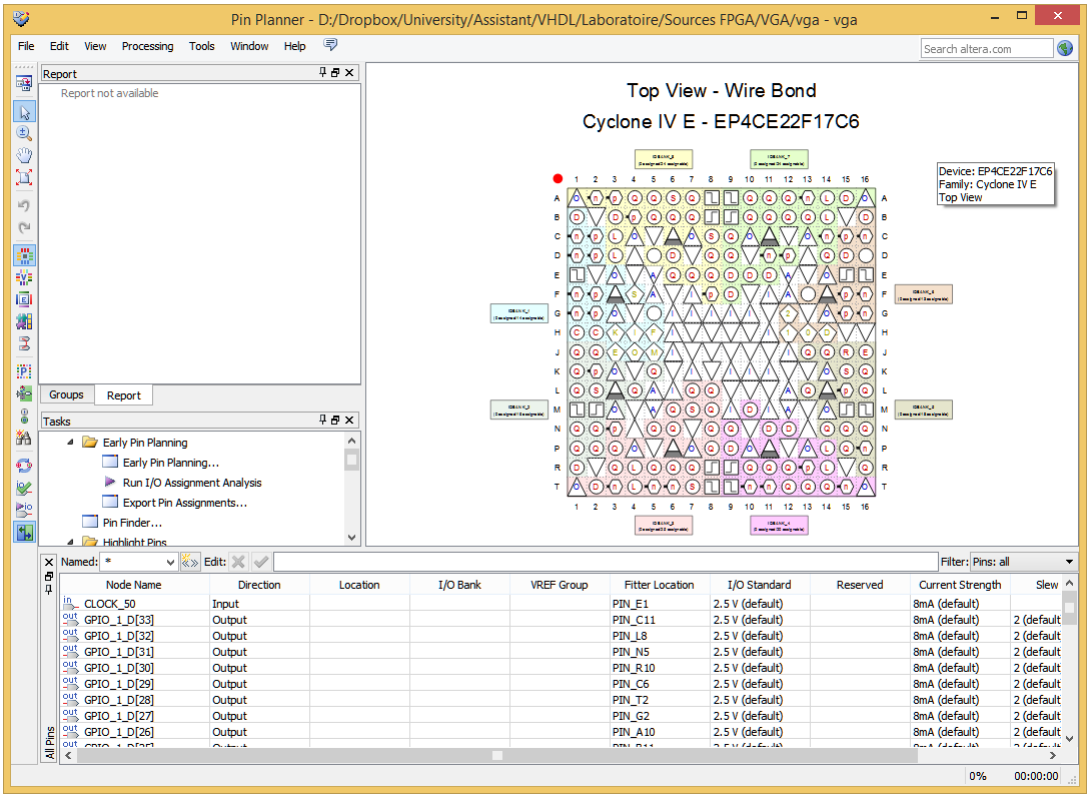
également indiquer au logiciel que faire des entrées/sorties non utilisées, et enfin indiquer quelle tension utiliser.

Dans ce projet, il y a deux types de signaux : les signaux classiques, et les signaux d'horloges. Un signal est considéré comme signal d'horloge par Quartus de manière automatique lorsque vous utilisez la fonction `rising_edge()`, entre autre. De plus, il est important de savoir qu'il ne peut y avoir que deux signaux d'horloge par composant. Vous devez être capable, à partir du code que vous avez écrit, de déterminer quel signal est un signal d'horloge. Mais vous pouvez avoir confirmation dans Quartus en allant dans le rapport du Fitter, puis de naviguer dans Ressource Section, Control Signal :



Les signaux d'horloge sont signalés par une valeur Clock dans le champ Usage.

Maintenant, il faut indiquer au logiciel l'emplacement de chacun de vos signaux sur le composant. Pour ce faire, cliquez sur le bouton , ce qui ouvre la fenêtre suivante :



Il est possible de déplacer un signal (par exemple `CLOCK_50`) sur une entrée/sortie de la FPGA. Attention toutefois à respecter les entrées/sorties de la DE0-nano ! N'oubliez pas de vous référer

au schéma de la carte pour placer vos entrées/sorties au bon endroit.

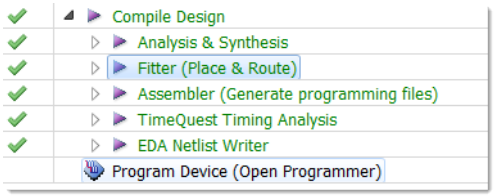
Enfin, vous pouvez également choisir la tension de fonctionnement d'une entrée/sortie, grâce à la colonne I/O Standard. Vérifiez que tout est en 3.3-V LVTTL (default).

Trouvez le manuel de la DE0-nano, et placez les signaux de sorties utilisés dans le projet vga aux bons endroits.

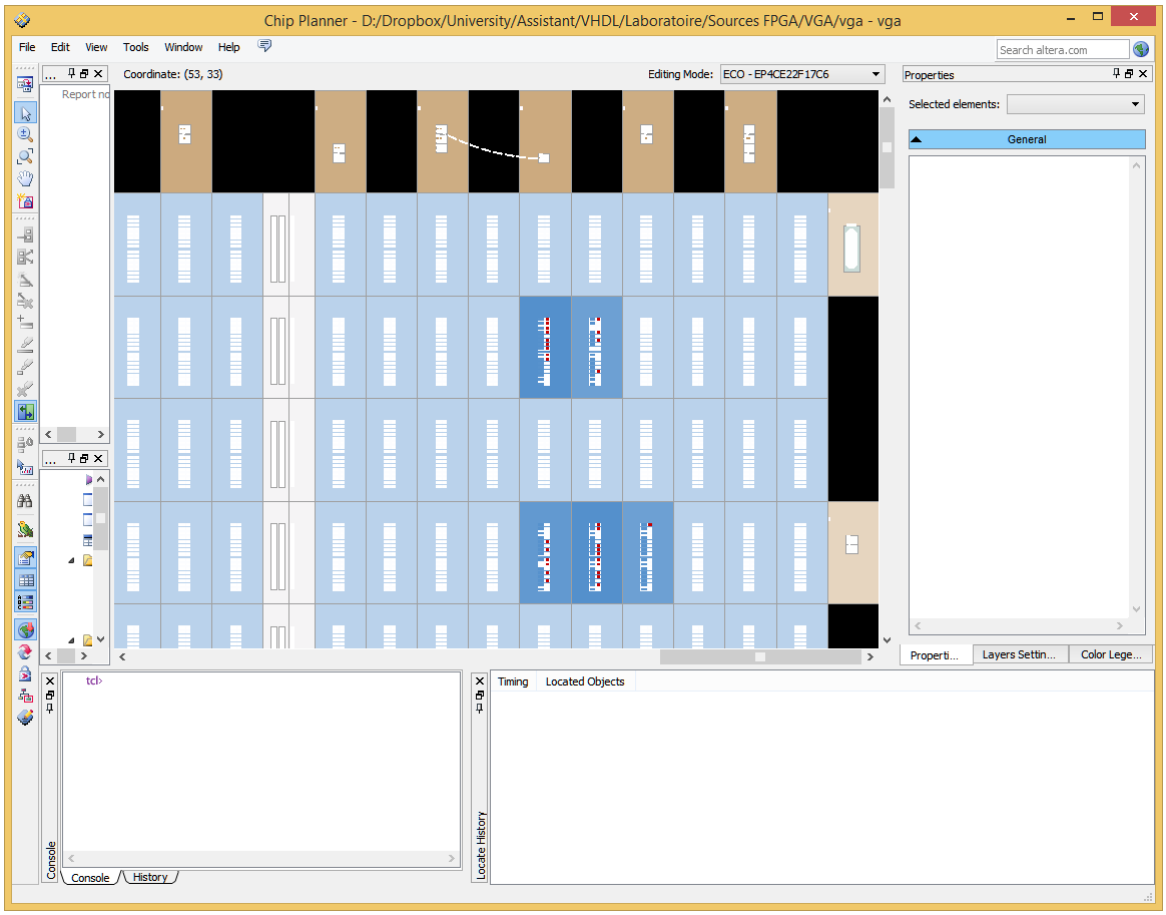
Synthèse du fichier

A ce stade, votre fichier a été codé, simulé et est donc fonctionnel. Il a également été analysé, et il ne reste plus qu'a le synthétiser, c'est-à-dire le mettre dans une forme compréhensible par le FPGA. Si les deux autres étapes ont été réalisées avec succès, il y a peu de chance que votre fichier ne passe pas cette étape.

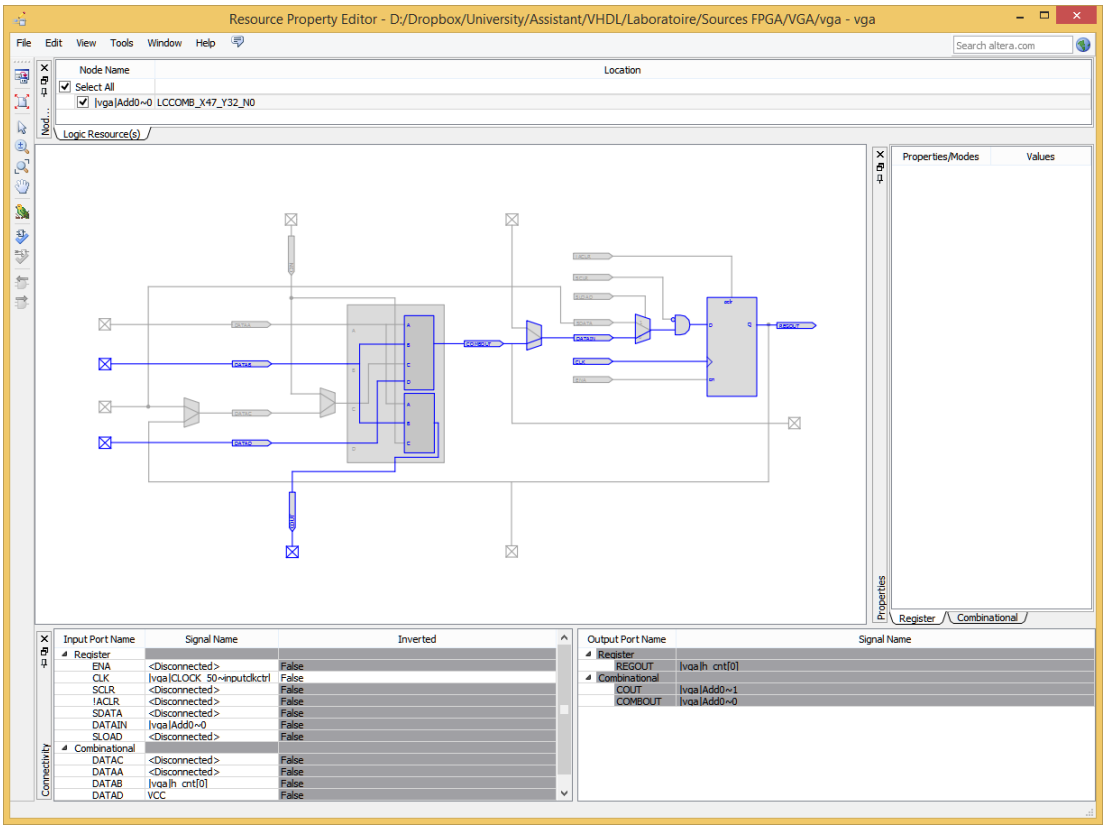
Cliquez sur le bouton pour lancer la compilation complète et enfin obtenir ceci :



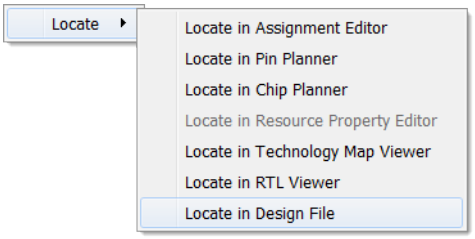
A la fin de cette étape, un certain nombre de rapports sont disponibles. Ainsi, il est par exemple possible de voir la façon dont le programme a été implémenté sur le composant. Pour ce faire, cliquez sur (Chip Planner), ce qui ouvre la fenêtre suivante :



Plus un bloc est bleu foncé, plus il est utilisé. Vous pouvez ainsi vous déplacer dans le programme afin de voir exactement comment votre système a été implémenté. Vous pouvez par exemple voir les équations logiques, les registres, etc. Double-cliquez sur un bloc pour ouvrir une nouvelle fenêtre avec le détail du bloc. Par exemple :



A chaque fois, il est possible de voir où l'élément à été déclaré dans le code, en faisant un clic droit, puis Locate → Locate in Design File :



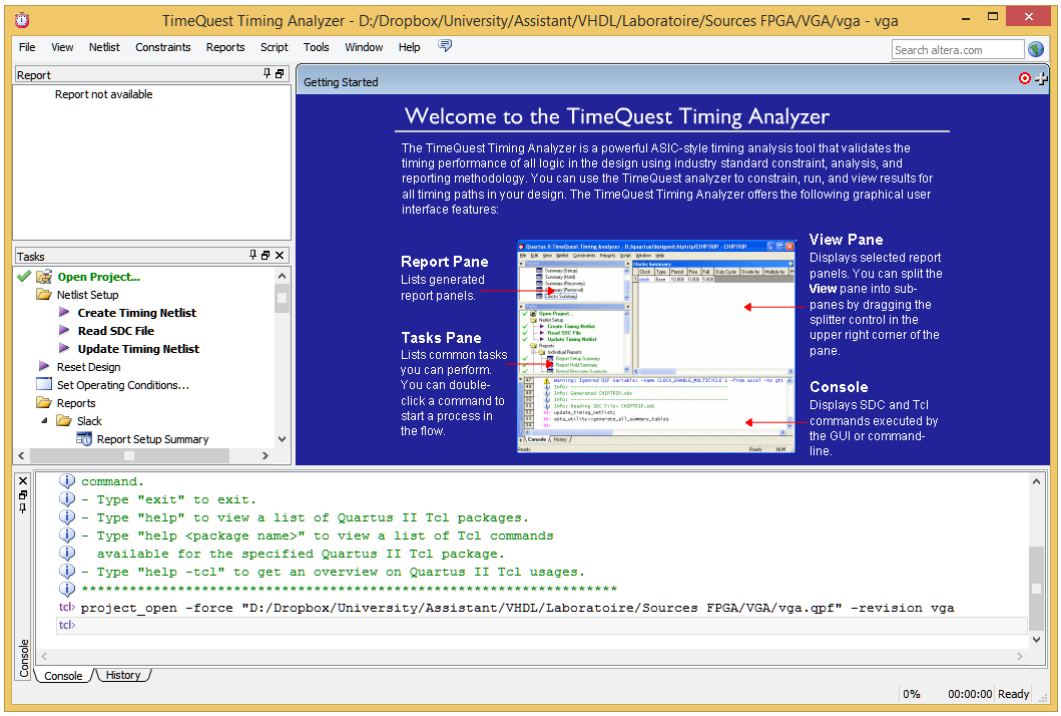
4.4 Analyse des délais (timings)

Dans certains programmes, il est important que les délais soient respectés. Ainsi, il faudra parfois que le délai du chemin entre une entrée et une sortie soit borné à une valeur maximale, ou bien que le décalage temporel entre une sortie valide et l'horloge reste dans une certaine limite, ou bien encore que l'horloge puisse être suffisamment rapide.

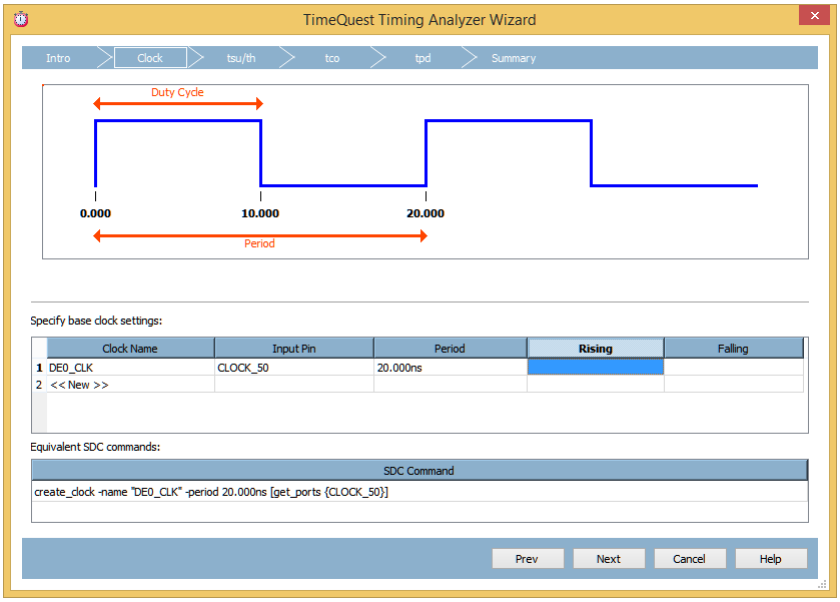
L'analyse des délais peut également influencer la compilation : imposer des délais très stricts sur certains chemins permettra de les placer plus près des I/O's, par exemple.

Dans cette optique, Quartus fourni l'outil TimeQuest, permettant de faire de nombreuses analyses. Dans le cadre de ce laboratoire, nous nous limiterons à imposer un délai maximum entre l'horloge d'entrée et la disponibilité d'une valeur correcte sur notre sortie.

Compilez l'ensemble de votre programme. Une fois ceci fait, cliquez sur pour lancer TimeQuest. Une nouvelle fenêtre apparaît :



Afin de nous simplifier la vie, nous allons utiliser un assistant (wizard). Cliquez sur Tools → TimeQuest Timing Analyzer Wizard. Dans la nouvelle fenêtre qui s'affiche, faites Next :



Cette première fenêtre permet d'indiquer les horloges du programme. Dans notre cas c'est très simple, il n'y en a qu'une, avec une période de 0.02 [us]. Il est également possible de faire une horloge asymétrique, mais nous n'allons pas utiliser cette fonctionnalité ici.

Les deux fenêtres suivantes permettent d'indiquer les délais des entrées/sorties par rapport aux horloges. C'est donc très utile lorsque le FPGA reçoit des entrées qui sont synchrones avec une de ses horloges. Enfin, la troisième fenêtre permet de contraindre les chemins qui ne sont pas affectés par des horloges, autrement dit, des chemins purement combinatoires.

Les délais en question (repris en figure 3) sont :

— T_{su} (setup time) : temps minimum pendant lequel l'entrée doit être stable avant le flanc montant. Ce temps contraint l'arrivée la plus tardive d'un signal à l'entrée d'un registre, et donc le chemin combinatoire le plus long,

— T_h (hold time) : temps minimum pendant lequel l'entrée doit être stable après le flanc montant. Ce temps contraint l'arrivée la plus précoce du signal à l'entrée d'un registre,

— T_{co} (clock to output) : temps entre le flanc montant et la disponibilité des données en sortie,

— T_{pd} (propagation delay) : temps de propagation d'un chemin purement combinatoire.

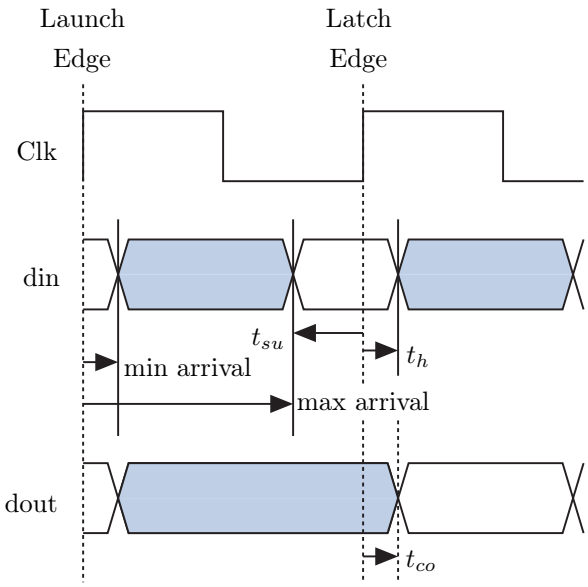
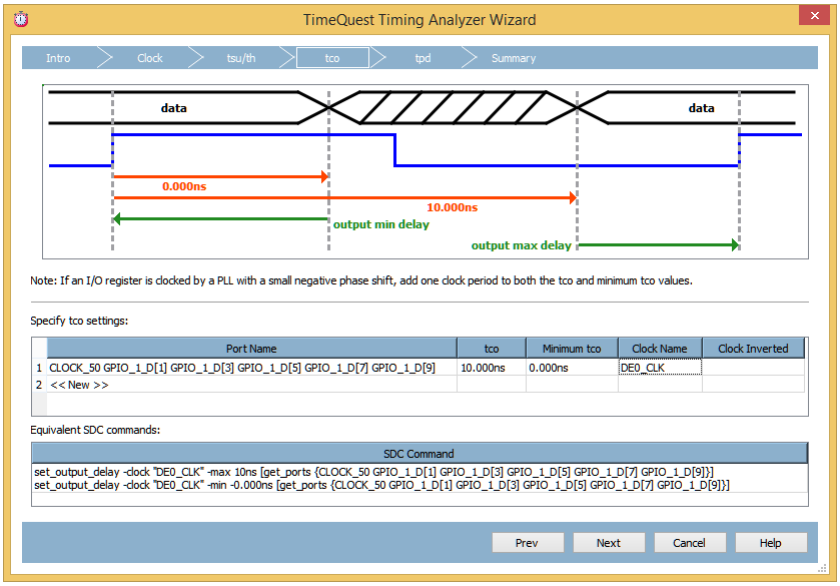


Figure 3 – T_{su}, T_h et T_{co}

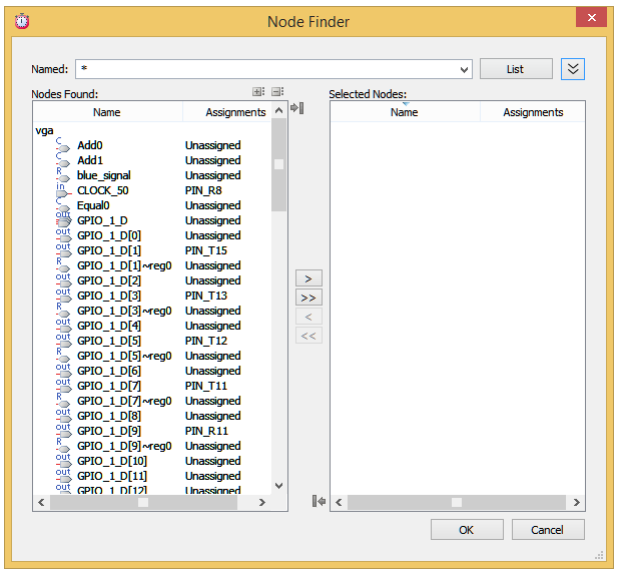
Nous n'avons pas ici de chemin combinatoire pur puisque tout est référencé à l'horloge `CLOCK_50`. Appuyez donc sur Next 2 fois pour atteindre la fenêtre permettant de régler les T_{co} :



Dans ce laboratoire, nous allons utiliser l'horloge comme entrée, et tester l'entièreté des sorties.

Pour ce faire, double-cliquez sur la première ligne, colonne Port Name, puis sur le bouton . Dans la fenêtre qui apparaît, cliquez sur List.

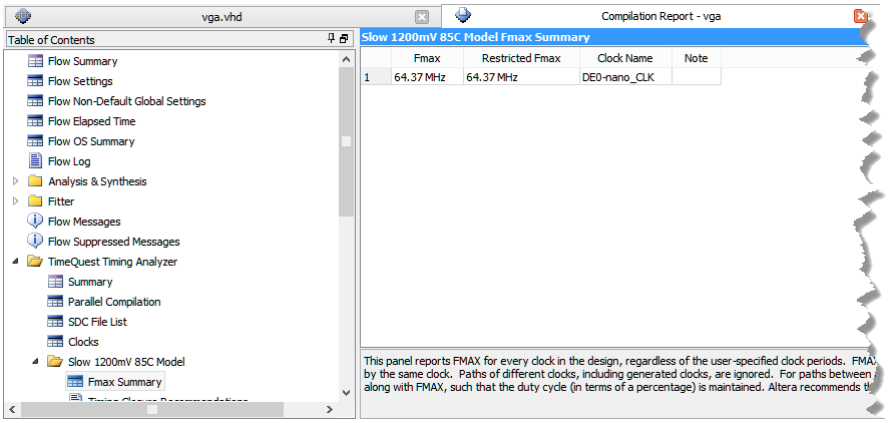
A noter qu'il est possible de cliquer sur le bouton pour régler plus finement l'effet du bouton List, par exemple en n'affichant que les I/O's attribuées.



Choisissez les GPIO qui sont attribuées, puis transférez-les dans la colonne de droite. Cliquez ensuite sur OK. Indiquez un T_{co} de 10 [ns], et n'oubliez pas de remplir la colonne `Clock Name` avec l'horloge créée dans la seconde fenêtre. Cliquez deux fois sur suivant, puis Finish, et quittez TimeQuest.

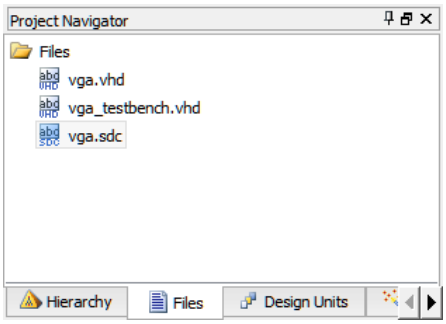
Lors du choix de la durée de T_{co}, regardez attentivement la fenêtre du Wizard : vous constaterez que les Timings sont indiqués sur le schéma, mais également que les commandes SDC sont affichées.
Changez la valeur de 10 [ns], et observez le changement de la commande SDC : la commande indique le Output Max Delay, et non pas le T_{co}. Ce qui veut dire que si vous souhaitez changer les timings par après via la modification du fichier source et sans passer par le Wizard, vous indiquerez le délai de sortie, et non pas le T_{co}. Jouez un peu avec les différentes valeurs pour comprendre cette subtilité fondamentale, qui sera indispensable pour répondre à la question qui suit.

Redémarrez la compilation (avec un T_{co} de 10 [ns]), puis affichez le rapport TimeQuest Timing Analyser → Slow 1200V 85C Model → Fmax Summary :



Ce rapport vous indique qu'il est possible d'augmenter la fréquence d'horloge jusqu'à 64.37 [MHz] sans enfreindre les différentes contraintes de timings. Ce même rapport indique également une fréquence de 67.11 [MHz], mais pour le modèle Slow 1200V 0C Model.

Enfin, nous allons modifier le fichier .sdc directement afin d'appliquer des contraintes trop serrées. Dans le Project Navigator, choisissez l'onglet Files, et double-cliquez sur le fichier vga.sdc :



Dans ce fichier, changez les contraintes T_{co} comme suit, et recompilez.

```
1 set_output_delay -clock "DE0-nano_CLK" -max 17ns [get_ports {GPIO_1_D
    [1] GPIO_1_D[3] GPIO_1_D[5] GPIO_1_D[7] GPIO_1_D[9]}]
2 set_output_delay -clock "DE0-nano_CLK" -min 0ns [get_ports {GPIO_1_D[1]
    GPIO_1_D[3] GPIO_1_D[5] GPIO_1_D[7] GPIO_1_D[9]}]
```

Cette fois-ci, l'analyse de timing ne réussi plus.

Observez tous les rapports qui vous sont soumis. A quoi servent-ils ? Que sont les différents modèles ? Essayez avec différentes valeurs d'horloge et de délais de sortie minimum et maximum. Que se passe-t-il ? Écrivez vos constatations de manière détaillée, et expliquez le fonctionnement des analyses de timings ainsi que les résultats qui peuvent être déduits. Aussi, regardez l'effet des contraintes de timings sur le Fitter. Que se passe-t-il ?

4.5 Si Timequest Timing analysis wizard n'est pas présent

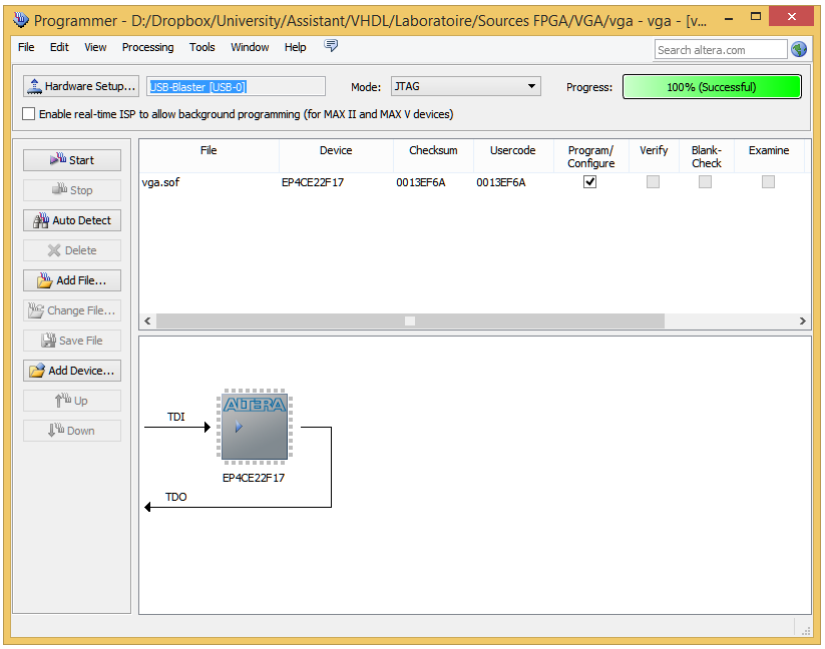
— cliquez sur Timequest
— cliquez sur Create timing netlist

— Dans le menu Constraints, cliquez sur create clock
— Donnez le nom `DE0_CLK` et réglez la période sur 20ns
— Cliquez sur targets pour associer la clock à `DE0_CLK`
— Cliquez sur Run
— Dans le menu Constraints cliquez sur Set ouptut delay
— Renseignez le nom `DE0_CLK`
— cliquez sur maximum
— Donnez 10ns comme valeur de delay
— cliquez sur target et ensuite sur list, sélectionnez les ports voulus.
— cliquez sur Run
— dans Timequest, descendez tout en bas de l'onglet Task et cliquez sur write sdc file
— sauvegarder le, il ne vous reste qu'à l'ajouter au projet en allant dans les settings de votre projet, onglet Timequest

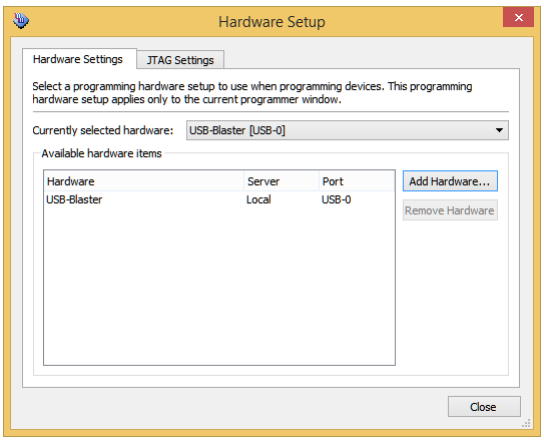
4.6 Programmation

La toute dernière étape consiste à programmer le FPGA. Pour se faire, connectez le programmateur à votre PC. Sous Windows, l'installation des pilotes n'est pas forcément automatique : vous devrez le faire manuellement, en allant dans votre gestionnaire de périphériques, et en indiquant l'emplacement des drivers. Ceux-ci se trouvent dans le répertoire quartus/drivers de votre installation de Quartus II.

Ouvrez le programmateur en double-cliquant sur Program Device (Open Programmer), dans la partie Tasks de Quartus. Une nouvelle fenêtre s'ouvre. Vous pouvez toutefois l'intégrer dans le logiciel sous la forme d'un onglet en faisant Window → Attach Window.



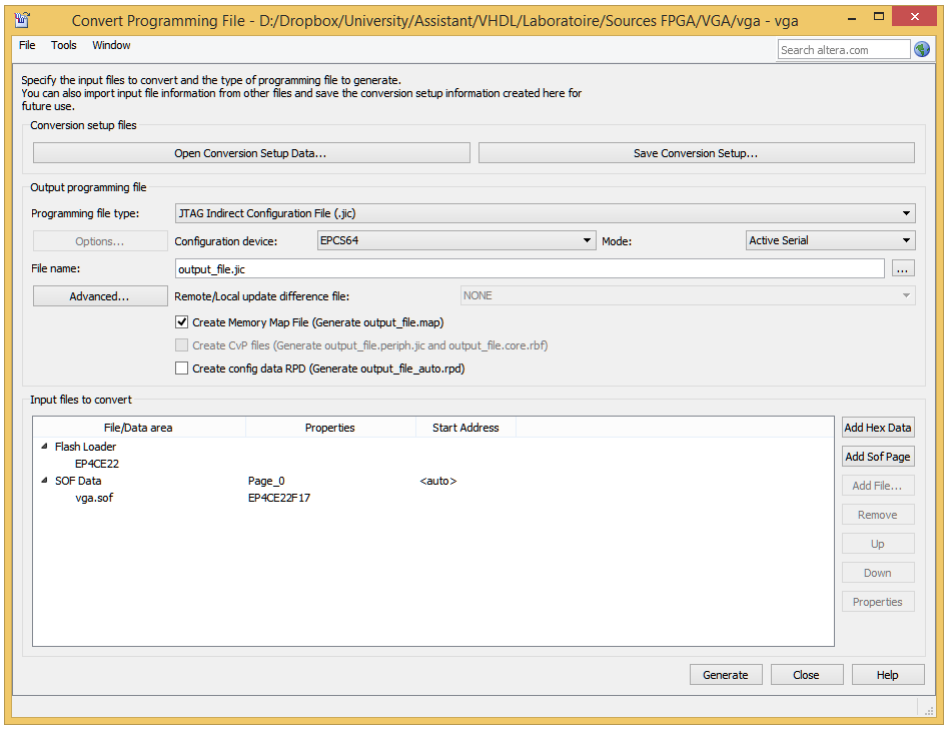
Cliquez sur le bouton Hardware Setup... pour configurer votre programmateur. Une nouvelle fenêtre s'ouvre, et choisissez USB-Blaster dans Currently Selected Hardware :



Faites Close pour revenir à la fenêtre de programmation, et cliquez tout simplement sur Start pour lancer la programmation. Si tout s'est bien passé, votre composant est programmé, et si vous branchez le cable VGA, vous devriez voir un carré rouge s'afficher à l'écran.

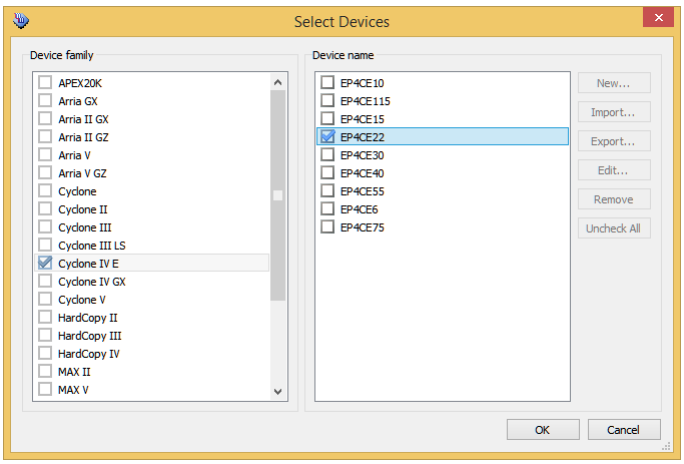
Toutefois, bien que le composant soit programmé, ce n'est pas suffisant. En effet, à la prochaine coupure de courant, le FPGA perdra sa configuration. Heureusement, la carte de développement contient une mémoire flash, qui permet donc au FPGA de charger sa configuration au démarrage. Il faut toutefois faire quelques paramétrages pour utiliser cette mémoire.

Faites files → Convert Programing File. Une nouvelle fenêtre s'ouvre alors :

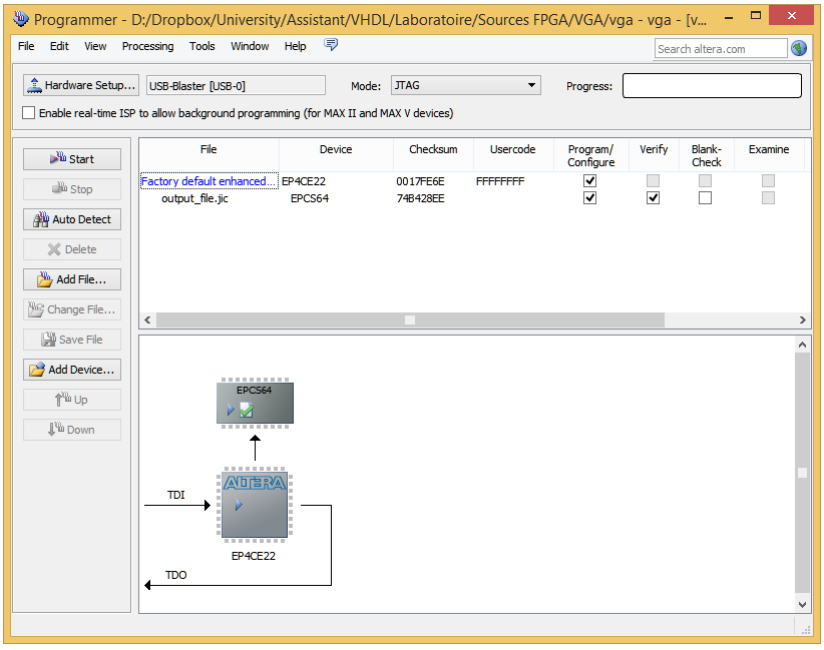


Dans Programming file type, choisissez JTAG Indirect Configuration File (.jic). Ensuite, pour la Configuration device, choisissez EPCS64.

Une fois ceci configuré, cliquez dans la partie blanche du bas de la fenêtre sur Flash Loader, puis sur les boutons de droite, faites Add Device. Dans la nouvelle fenêtre, choisissez notre FPGA :



Pour le SOF Data, faites Add File, et choisissez le fichier compilé, qui porte donc l'extension .sof. Enfin, cliquez sur Generate. Réouvrez le programmeur, et supprimez le fichier vga.sof. Ensuite, cliquez sur , et choisissez cette fois le fichier .jic. Enfin, dans les cases à cocher, sélectionnez Program/Configure et Verify :



Reprogrammez le système. Cette fois ci, la tâche devrait être un peu plus longue, tandis que le programme ne devrait pas démarrer automatiquement, comme la fois précédente. Enlevez la prise USB de la carte et rebranchez : votre carré rouge apparait !

4.7 Exercice

Modifiez le programme du carré rouge pour créer un carré dont la couleur passe par les 8 couleurs accessibles avec notre système simple, en alternance toute les secondes.

Programmez le FPGA avec un code persistant (donc sur la mémoire Flash), et faites une analyse de timings rapide pour essayer de trouver la fréquence maximale de votre programme.

5 Debug d'un projet

5.1 Introduction

Il arrive parfois qu'il soit très difficile de créer une simulation, voir impossible. Un cas fréquent est lors de l'utilisation de périphériques externes, comme par example l'accéléromètre présent sur la carte.

Cet accéléromètre utilise le protocole SPI, et dès lors, générer des signaux valides pour simuler ses réponses via un TestBench est particulièrement délicat.

Un autre cas possible est lorsqu'on utilise le code d'un tiers. Ce code est parfois peu, voir pas du tout commenté, ou alors suit une logique qui semble propre au programmeur.

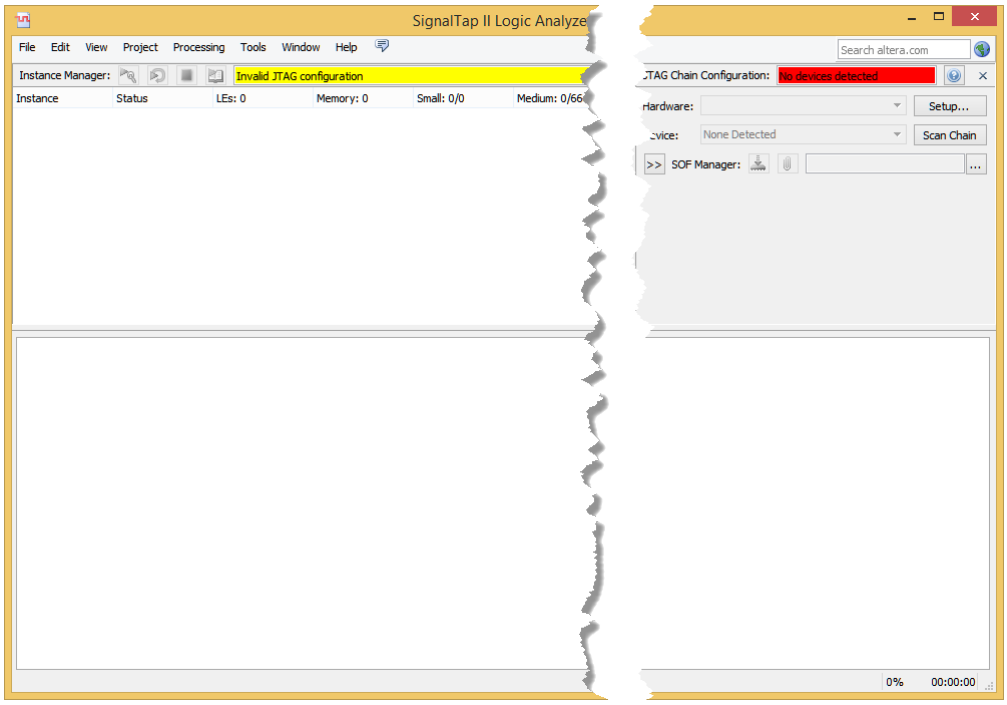
Dans cette dernière partie, nous allons donc voir comment analyser les signaux générés dans le FPGA après l'avoir programmé, sans pour autant utiliser un oscilloscope. Ansi, il sera possible d'observer chaque signal d'un programme, même si vous n'avez à priori aucune idée de l'utilité de ces signaux.

5.2 Analyse des signaux générés au sein du FPGA

Pour réaliser cette tâche, Quartus dispose d'un outil très pratique : SignalTap. Cet outil utilise la mémoire embarquée du FPGA pour enregistrer les signaux générés en interne, puis les retransmet sur le port USB et les affiche.

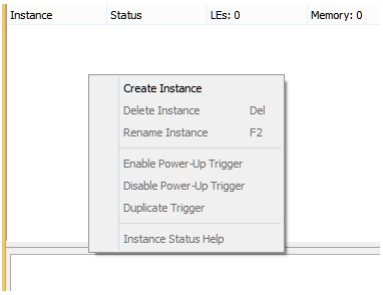
Il est possible également de définir certains triggers pour déclencher uniquement aux instants voulus. Bien entendu, tout ceci vient au coût d'une surcharge (overhead) sur votre programme.

Allez dans Tools, puis cliquez sur SignalTap II Logic Analyzer. Une nouvelle fenêtre apparaît :

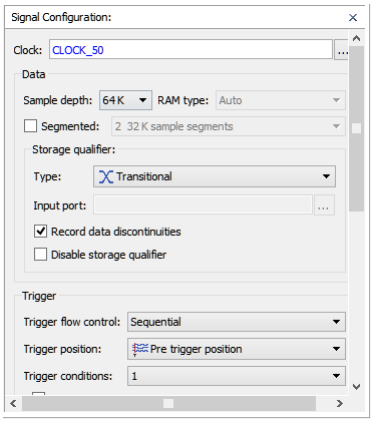


La première chose à faire est de configurer le programmateur. Dans la partie droite de la fenêtre, cliquez sur Setup, et configurez votre USB-Blaster comme lors de la programmation du composant. Vous devriez voir un message sur fond jaune "Invalid JTAG configuration", ce qui indique que le composant n'est pas bien configuré. Ceci est normal puisque nous n'avons encore chargé aucun code compatible avec SignalTap.

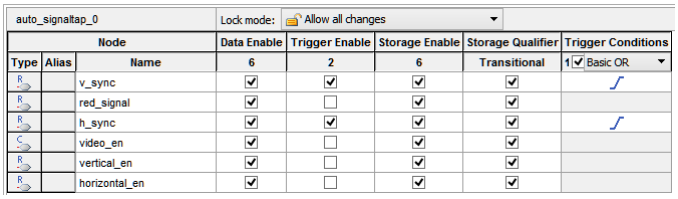
Dans la partie blanche supérieur de la fenêtre, faites un clic droit, puis Create Instance :



La fenêtre se modifie alors et une instance est ajoutée. Dans Signal Configuration, configurez l'horloge et indiquez un Sample depth de 64K :



Enfin, sur la partie gauche de la fenêtre, faites un clic droit pour ajouter des signaux, et ajoutez les signaux suivants :

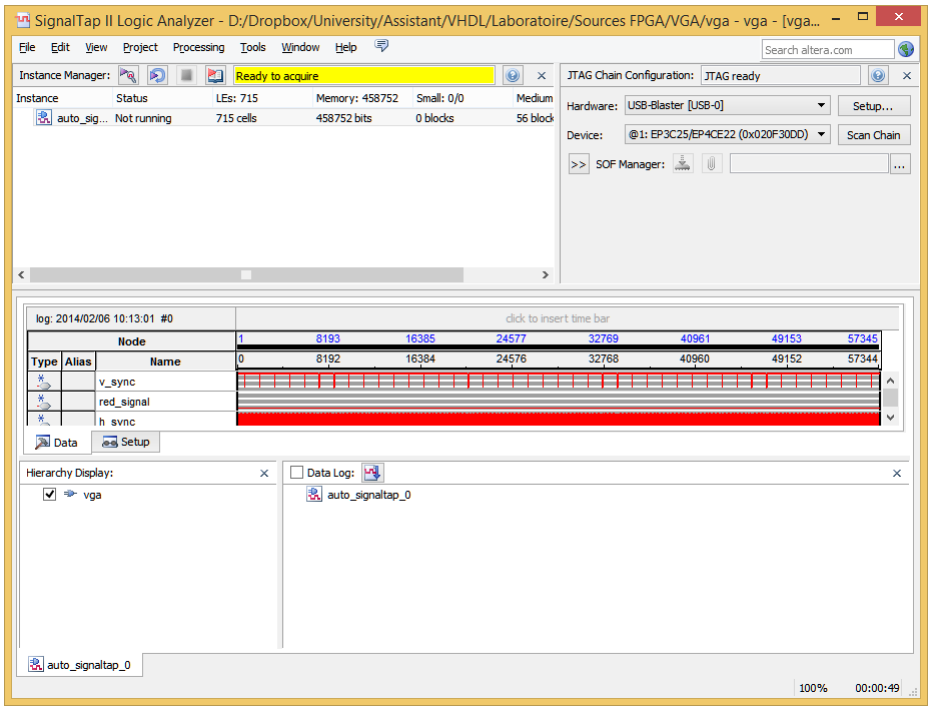


Plus vous ajouterez de signaux à observer, plus les besoins en mémoire seront élevés, et donc plus il faudra diminuer le Sample Depth. Il est également possible d'utiliser des triggers très variés en cliquant sur la colonne Trigger Conditions. Dans ce cas ci, nous allons utiliser un Basic OR, sur les rising edge du signal `v_sync` et `h_sync`.

Enfin, toujours dans la même fenêtre, sélectionnez votre instance, compilez votre programme comme d'habitude. Vous remarquerez que celui-ci à augmenté de taille au niveau des ressources

utilisées. Programmez le composant sans toutefois programmer la Flash, de manière à ce que le programme se lance automatiquement, et retournez dans SignalTap.

Appuyez sur le bouton pour démarrer l'acquisition, ce qui devrait vous donner ceci :



Tous les signaux indiqués ont été enregistrés pendant l'exécution, puis transférés au logiciel. Vous pouvez maintenant zoomer et analyser le comportement en détail.

5.3 Analyse du code d'un tiers

Chargez le projet Gsensor disponible sur le site web. Grâce à tous les outils vu au long de ce laboratoire, essayez de comprendre le fonctionnement de ce projet. Que fait-il ? Comment le fait-il ? Quelles sont les variables importantes ? En particulier, replacez des commentaires dans le code.

Commencez par afficher le RTL viewer pour voir l'emboitement des différents morceaux de code. Ensuite, allez lire le datasheet de l'accéléromètre de la carte, un ADXL345. Essayez de comprendre le code, puis, grâce à des analyses SignalTap, vérifiez que ce que vous avez compris correspond aux observations : changez quelques morceaux de code, et observez les réactions.

Montrez au travers de votre rapport, avec des analyses de signaux, des diagrammes, etc, que vous avez compris ce code.

Cet exercice prendra plus de temps que la durée du laboratoire, vous devrez donc rendre votre rapport au plus tard 2 semaines après votre passage.

6 Exercice final

Une fois que vous avez compris le code précédent, modifiez le pour afficher un carré rouge, qui se déplacera sur les axes X et Y en fonction de la position de votre carte de développement, qui agira donc comme une télécommande.