TP VHDL : Simulation & Synthèse

I. Documents et rendus

I.A. Manuel utilisateur de la carte

https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=234&No=1021&PartNo=4

I.B. Documentations techniques du MAX10

https://www.intel.fr/content/www/fr/fr/programmable/products/fpga/max-series/max-10/support.html

I.C. Dépôt Web

L'archive que vous venez de télévcharger contient tous les fichiers nécessaires au TP :

- ce document au format PDF
- des fichiers :
  - ./modelsim/sim_*.do : scripts pour compiler et simuler l'entité « * ». Le fichier « wave_*.do » est attendu. Voir § VI.C
  - ./sources/template_*.vhd : fichiers temporaires à utiliser pour débuter l'entité correspondante
  - ./synthese/* : fichiers du projet Quartus

Voir § III.B.1.

I.D. Rendu

Une fois votre TP terminé, vous ferez une archive 7z (ou zip, pas de rar ni aucun autre format) de votre dossier projet :

- dossier « modelsim » ([D1]). Ne pas inclure
  - sous-dossier « work »
  - fichiers *.wlf
- dossier « sources » ([D2]).
- dossier « synthese » ([D3]). Ne pas inclure
  - sous-dossier « db »
  - sous-dossier « incremental_db »

Cette archive doit faire moins de 1Mo (test sur ma machine : 513Ko pour le projet complet).
Votre rapport au format DOCX ou PDF devra lui-aussi être fourni.
Un lien spécifique à chaque trinôme vous sera communiqué où vous pourrez déposer vos fichiers.

II. Présentation

II.A. L'horloge

Vous allez étudier une horloge affichant les heures, minutes et secondes. Pour cela, vous devrez modéliser en langage VHDL un système ayant le schéma bloc suivant :

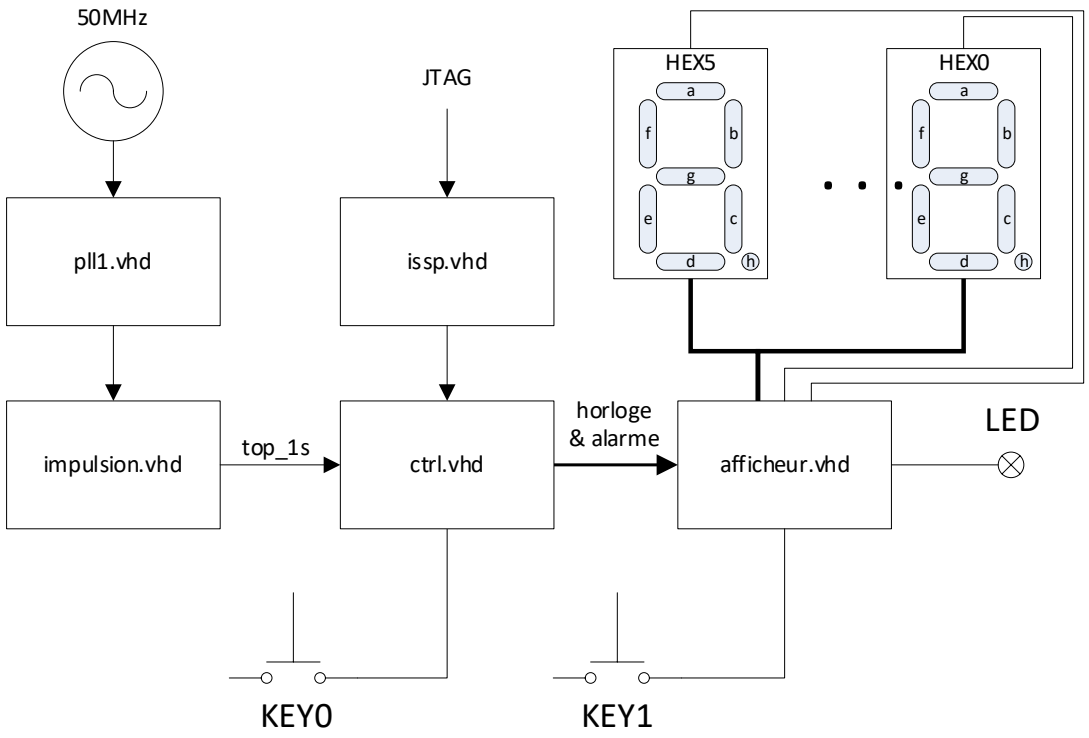


Figure 1 : Principe de réalisation

Les signaux « clk » et « rst » sont envoyés à tous les modules. Leur connexion n'est pas indiquée sur la Figure 1 par souci de lisibilité.

II.B. L'oscillateur

L'oscillateur présent sur la carte est cadencé à 50 MHz.

II.C. Le module impulsion

Ce module génère une impulsion d'une durée d'un cycle d'horloge (à 50 MHz) chaque seconde.

II.D. Le module contrôle

Ce module effectue l'opération de comptage à partir de son entrée d'horloge à 50 MHz et de l'impulsion à 1s.

Le bouton KEY0 permet de charger les heures de l'horloge et de l'alarme, celles-ci étant fournies par l'outil « In-System Sources & Probes ».

II.D.1. Chargement

Le chargement des heures (horloge & alarme) s'effectue par l'appui sur le bouton poussoir KEY0 filtré par un anti-rebonds que vous devrez coder.

Pour information :

- le bouton poussoir envoie un état logique haut au repos
- le bouton poussoir envoie un état logique bas lorsque l'on appuie dessus

II.D.2. LEDs

Lorsque l'alarme se déclenche, les 10 LEDs rouges de la carte doivent clignoter au rythme 1 seconde allumé / 1 seconde éteint. Proposez une méthode triviale n'utilisant aucun compteur pour réaliser cette fonction (seuls des comparateurs sont autorisés).

II.E. Le module afficheur

Ce module pilote les 6 afficheurs 7-segments.
Les segments sont allumés par l'envoi d'un état logique bas par le FPGA.

II.E.1. Horloge / Alarme

L'affichage de l'horloge est actif par défaut : bouton poussoir KEY1 relâché.
L'affichage de l'alarme est actif lorsque le bouton poussoir KEY1 est appuyé.

III. Codage & Simulation

III.A. Contraintes

Bibliothèque	Interdite	Autorisée
std_logic_arith	×	
std_logic_unsigned	×	
std_logic_signed	×	
numeric_std		×

III.B. Déroulement

III.B.1. Arborescence

Décompressez l'archive qui vous est fournie. Ce dossier doit être sur un disque local (qui pourra ensuite être archivé et copié sur votre disque réseau ou sur une clé USB).

Vérifiez que vous avez bien les sous-dossiers suivants :

- modelsim (noté [D1])
- sources (noté [D2])
- synthese (noté [D3])

Très important : veillez bien à ce que les dossiers ne contiennent **ni espace, ni caractère accentué**, et qu'ils soient situés sur un disque **local**.

III.B.2. Création du projet ModelSim

Lancez ModelSim.

Créez un projet MPF dans [D1] (File / New / Project).

III.B.3. Impulsion

III.B.3.1 Module de synthèse

En vous basant sur la Figure 1 et notamment la liste des entrées/sorties, créez un fichier impulsion.vhd (dans [D2]) générant une impulsion à chaque seconde. Vous pouvez vous aider du fichier « template_impulsion.vhd » fourni. **Respectez les règles du § VI.A**.

- l'entité doit s'appeler « impulsion » et son architecture « rtl »
- le code doit être synthétisable
- la période de répétition de l'impulsion doit être contrôlée par le paramètre *generic* « CLK_DIVIDER » de type « natural »
- vous ne devez utiliser qu'un unique process ressemblant à ceci :

```
process(rst,clk)
begin
if rst='?' then

elsif rising_edge(clk) then
  instructions
end if;
end process;
```

La forme d'onde de cette impulsion est définie comme suit :

- état haut : durée 1 cycle d'horloge
- état bas : « CLK_DIVIDER -1 » cycles d'horloge

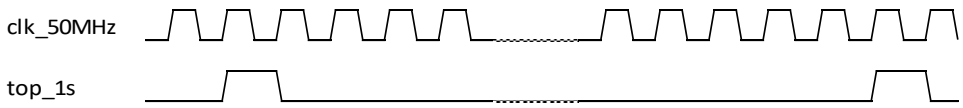


Figure 2 : Forme d'onde de l'impulsion de 1 s

III.B.3.2 Testbench

Toujours dans [D2], créez un fichier impulsion_tb.vhd.

- l'entité doit s'appeler « impulsion_tb » et son architecture « testbench »
- mappez CLK_DIVIDER à une valeur très faible (comme 100 par exemple)
- vous n'avez aucune autre contrainte

III.B.3.3 Questions

1) Ajoutez ce fichier au projet de ModelSim, définissez l'ordre de compilation. Voir § VI.B

2) En faisant attention à la taille de votre compteur (nombre de bits) par rapport à la plus grande valeur que l'on pourra y charger (donc pour atteindre la seconde), simulez votre module « impulsion ». Documentez votre code et rédigez votre rapport.
   ✋ N'oubliez pas de créer/sauvegarder votre fichier « wave_*.do » qui vous servira lors de l'audit !

3) Expliquez pourquoi on ajoute cette constante qui aura évidemment la « vraie » valeur en synthèse (50.10^6) alors qu'en simulation elle a une si petite valeur. Pour vous aider :
   a. Essayez par exemple de faire exécuter la simulation jusqu'à la seconde. Quel temps a-t-il fallu à ModelSim ?
   b. Est-ce gérable si vous voulez simuler plusieurs centaines de secondes ?

4) Essayez de généraliser cette dualité (valeur de simulation // valeur de synthèse) en énonçant les contraintes qu'il faudra veiller à respecter.

III.B.4. Contrôle

III.B.4.1 Module de synthèse

En vous basant sur la Figure 1 et notamment la liste des entrées/sorties, créez un fichier ctrl.vhd (dans [D2]) qui réalise le comptage des heures / minutes / secondes pour l'horloge. Il réalise aussi le chargement de l'horloge et de l'alarme.

- l'entité doit s'appeler « ctrl » et son architecture « rtl »
- le code doit être synthétisable
- vous ne devez utiliser qu'un unique process ressemblant à ceci :

```
process(rst,clk)
begin
if rst='?' then

elsif rising_edge(clk) then
  instructions
end if;
end process;
```

- la méthode d'optimisation des compteurs présentée en cours n'est pas conseillée ici !

III.B.4.2 Testbench

Toujours dans [D2], créez un fichier « ctrl_tb.vhd »

- l'entité doit s'appeler « ctrl_tb » et son architecture « testbench »
- vous n'avez aucune autre contrainte

III.B.4.3 Questions

1) Ajoutez ce fichier au projet de ModelSim, définissez l'ordre de compilation. Voir § VI.B.
2) Simulez votre module « ctrl ».
3) Documentez votre code et rédigez votre rapport en incluant notamment :
   a. une capture d'écran montrant la durée équivalente à 1 seconde (cette mesure doit apparaître entre 2 curseurs de ModelSim sur la fenêtre wave). On note δ_{su} cette mesure.
   b. idem pour les dizaines de seconde, notée δ_{sd}
   c. idem pour 1 minute, notée δ_{mu}
   d. idem pour les dizaines de minute, notée δ_{md}
   e. le chargement horloge/alarme (le module ISSP sera remplacée par une constante)
4) Vérifiez que les relations entre les δ_x sont correctes (par exemple, $\delta_{mu} = 60.\delta_{su}$)
5) Rédigez votre rapport.

III.B.5. Afficheur

III.B.5.1 Module de synthèse

En vous basant sur la Figure 1 et notamment la liste des entrées/sorties, créez un fichier afficheur.vhd (dans [D2]) qui réalise la gestion des afficheurs 7-segments.

- l'entité doit s'appeler « afficheur » et son architecture « rtl »
- il est recommandé d'utiliser uniquement de la logique combinatoire pour la fonction de conversion
- le code doit être synthétisable
- vous ne devez utiliser qu'un unique process ressemblant à ceci :

```
process(rst,clk)
begin
if rst='?' then

elsif rising_edge(clk) then
  instructions
end if;
end process;
```

III.B.5.2 Testbench

Toujours dans [D2], créez un fichier « top_tb.vhd ».

- l'entité doit s'appeler « top_tb » et son architecture « testbench »
- vous n'avez aucune autre contrainte

Il vous est bien demandé de faire le testbench du top-level ; ce dernier contenant donc maintenant les trois modules que vous venez de développer (ici, l'intérêt de faire un testbench dédié à l'afficheur est quasi-nul).

III.B.5.3 Questions

1) Ajoutez ce fichier au projet de ModelSim, définissez l'ordre de compilation. Voir § VI.B
2) Validez rapidement que les modules « impulsion » et « ctrl » sont toujours fonctionnels.
   Dans votre rapport, indiquez uniquement que vous avez fait cette question.
3) Simulez votre module « afficheur ». Documentez votre code et rédigez votre rapport.

III.B.6. pll1

Pour connecter les différents modules, un reset asynchrone est nécessaire. Aucune source physique sur la carte ne permet d'obtenir un tel signal.

Par ailleurs, l'utilisation directe d'une horloge externe (ici l'oscillateur 50MHz) est déconseillée.

Il convient donc d'instancier une PLL (*ALTPLL* dans IP Catalog) configurée pour recevoir une horloge à 50MHz et fournir une horloge interne à 50MHz (même fréquence). Il faut donc :

- désactiver l'option « Create a 'areset' input to asynchronously reset the PLL »
- activer l'option « Create 'locked' output »

Proposer (et utiliser) une méthode afin de créer le signal de reset asynchrone nécessaire à partir du signal 'locked' fourni par cette PLL.

IV. Synthèse

IV.A. Travail préparatoire

La cible technologique est un FPGA MAX10 de la société Intel. Effectuez une description détaillée de la structure de cette famille de composants. Quels types d'opérations peuvent-ils supporter ?

IV.B. Contraintes

~~~ATTENTION~~~

> Il vous est expressément demandé de ne modifier aucune contrainte, ni réglage de Quartus, en dehors de celles demandées aux § IV.C.2 et IV.C.3.

IV.B.1. Questions

1) Module *top*
   a. Mettez à jour la valeur de la constante CLK_DIVIDER pour la synthèse (voir la ligne 24 du fichier « template_top.vhd »). Vous ne devez remplacer que le « y » par votre valeur.
2) Module *afficheur*
   a. Il vous est demandé de piloter chaque sortie par un registre. À combien estimez-vous le nombre de registres nécessaires pour ce module ?

IV.C. Synthèse logique

IV.C.1. Utilisation de Quartus

Double-cliquez sur le fichier « tp_vhdl.qpf » pour lancer Quartus et ouvrir automatiquement le projet. Les contraintes et la liste des fichiers VHDL sont déjà configurées.

IV.C.2. Mode AREA

Sélectionnez l'option « Area » (menu Assignments / Settings / Analysis & Synthesis Settings). Compilez les codes sources VHDL (menu Processing / Start Compilation)

Commentez le rapport de synthèse obtenu.

IV.C.3. Mode SPEED

Sélectionnez l'option « Speed » (menu Assignments / Settings / Analysis & Synthesis Settings). Compilez les codes sources VHDL (menu Processing / Start Compilation)

Commentez le rapport de synthèse obtenu.

IV.D. Implémentation physique

Affichez la fenêtre de messages (View / Utility Windows / Messages). Vérifiez que vous n'avez aucune erreur, ni « critical warning », et un unique « warning ».

Voici un aperçu :

Si cette image correspond à ce que vous avez, vous pouvez passer à la suite.

Dans le cas contraire :

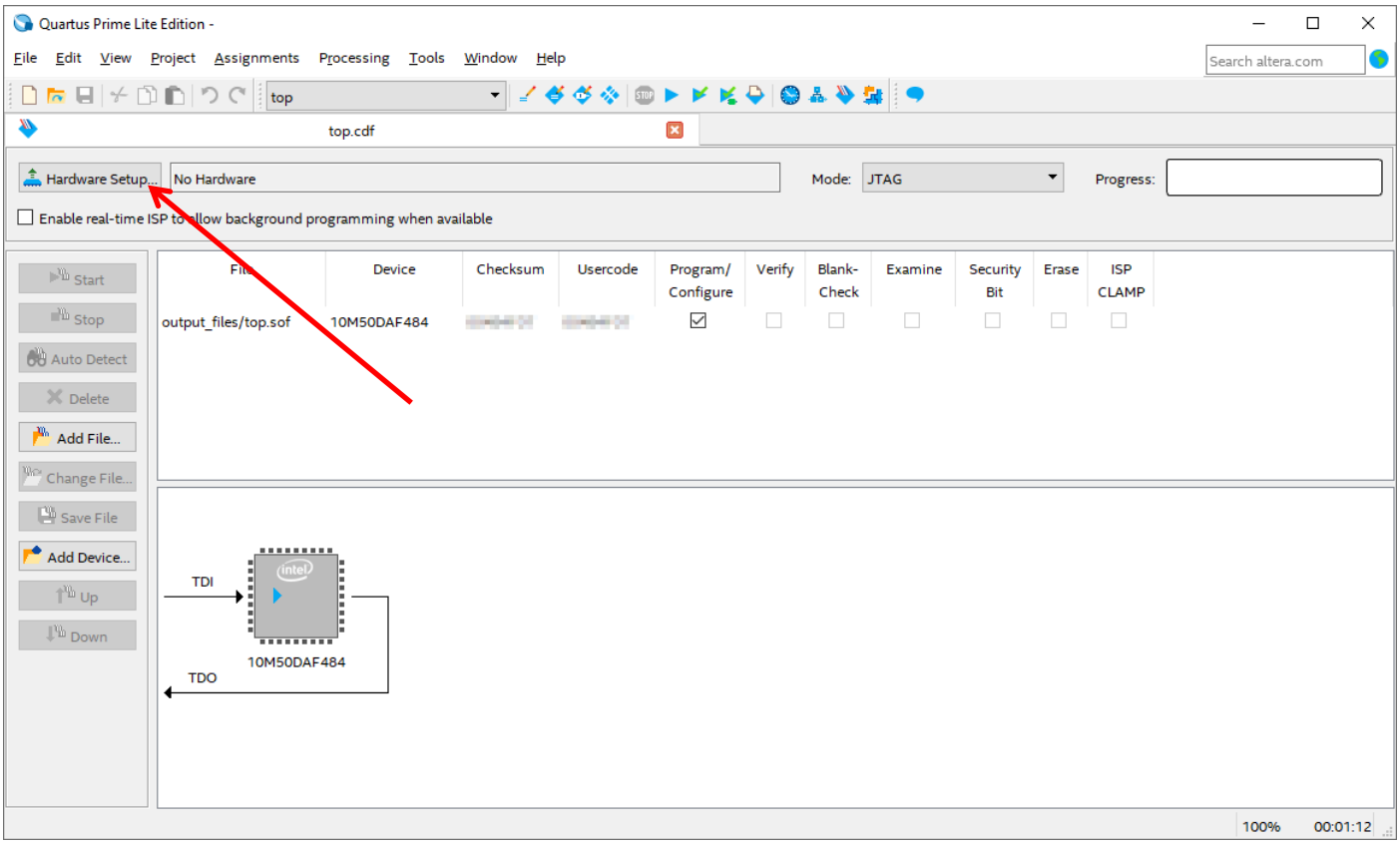
- **vous ne devez pas passer aux étapes suivantes de ce paragraphe**
- **vous ne devez pas programmer la carte DE0 !!!**

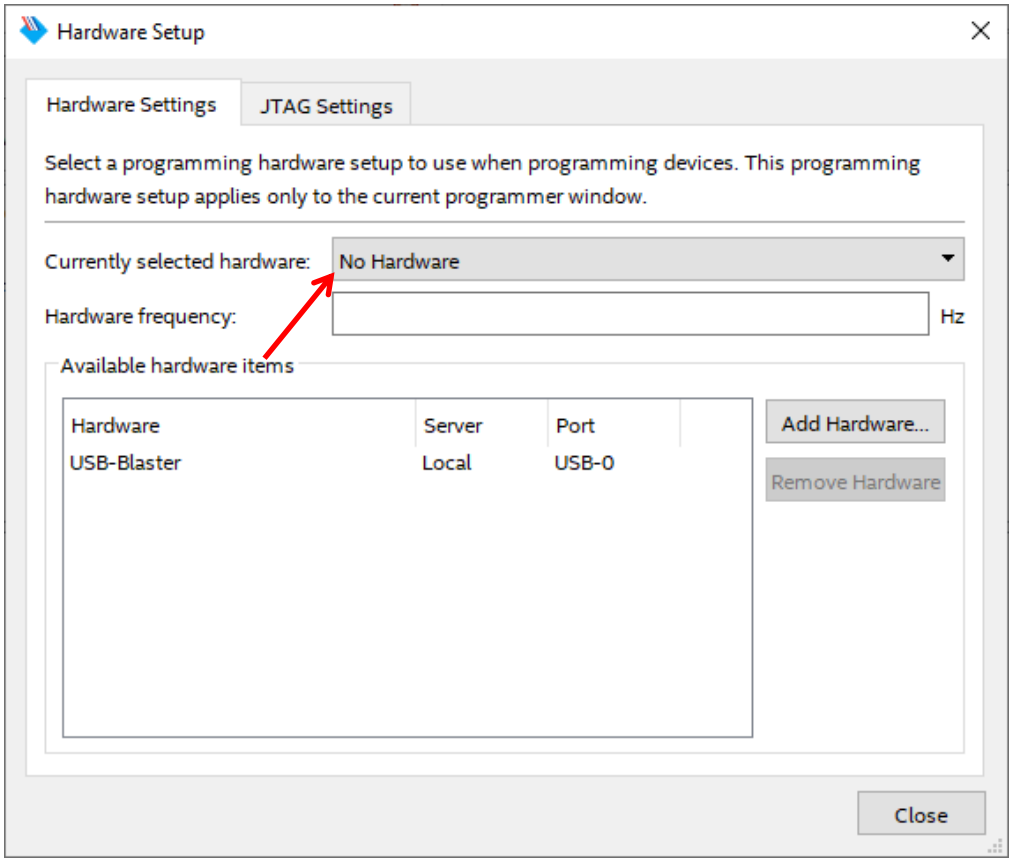
Vous pouvez néanmoins continuer au § V.

Connectez le câble USB à la DE0-Nano pour la mettre sous tension.
Sous Quartus, ouvrez le programmeur :

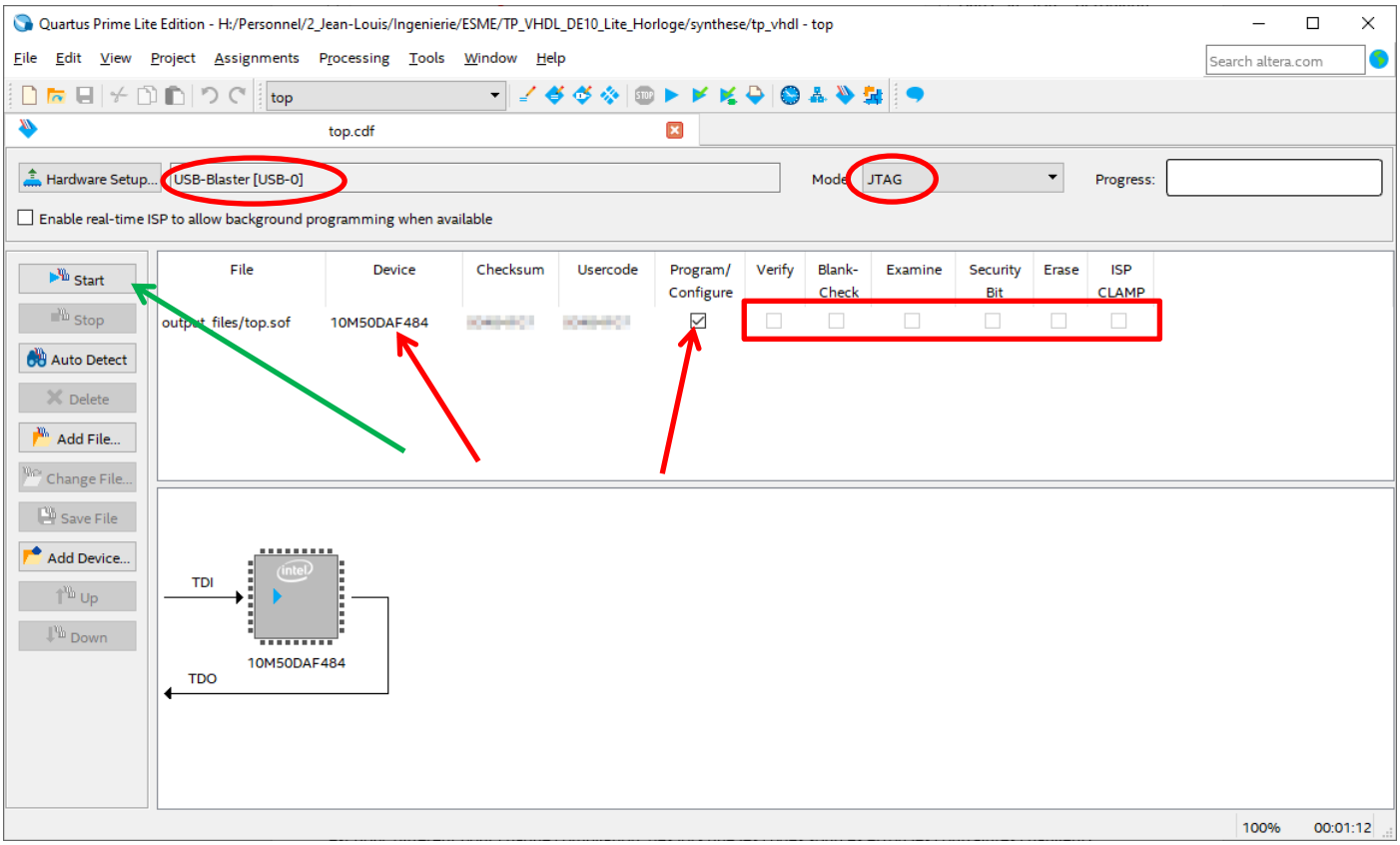
- soit par le menu Tools / Programmer
- soit par l'icône du Programmer : 

Puis dans « Hardware setup », sélectionnez le câble « USB Blaster », comme décrit ci-dessous :





Dans la liste déroulante, sélectionnez « USB-Blaster [USB-x] » (où x est un nombre a priori quelconque).



Vérifiez que toutes les informations en rouge soient correctes. Le checksum correspond au CRC32 du fichier SOF (il est donc différent pour chaque compilation, dès lors que les codes sources et/ou les contraintes changent).

Vous pouvez alors programmer le FPGA en cliquant sur le bouton « Start » (flèche verte)

V. Analyse du module « impulsion »

1) Expliquez pourquoi on n'a pas généré un signal dont le rapport cyclique est proche de 50% mais un signal qui ne présente qu'un unique cycle actif sur sa période. Pour vous aider :
   a. Quel rapport cyclique approximatif caractérise une horloge ? La réciproque est-elle vraie (dans le cas général) ?
   b. Si un signal est une horloge, quel type de ressource physique précise va-t-il utiliser dans le MAX10 ? Et si ce signal n'est pas une horloge ?
   c. En se basant sur un ordre de grandeur du MAX10 de votre carte, en quelles quantités sont les ressources évoquées au point précédent ?

2) La Figure 2 représente un chronogramme « de principe ». Que manque-t-il entre un front montant de *clk_50MHz* et une transition de *top_1s* ? Si l'on avait généré un signal de rapport cyclique 50%, est-ce que cette nouvelle horloge aurait pu être considérée comme synchrone et en phase avec *clk_50MHz* ? En supposant que l'on ait plusieurs fois ce type de relation (clk_1 vers clk_2, …, clk_{n-1} vers clk_n) que dire de la relation entre clk_1 et clk_n ? Cela est-il réaliste et viable dans un système complet ?
   Pour cette question, vous inclurez :
   - un 1er chronogramme pour illustrer la relation *clk_50MHz // top_1s*
   - puis un 2nd chronogramme pour les relations clk_1//clk_2 + clk_2//clk_3 + clk_{n-1}//clk_n
   - enfin un 3ème chronogramme pour la relation clk_1//clk_n en incluant une donnée synchrone de clk_1 et une donnée synchrone de clk_n.

3) La règle suivante vous paraît-elle justifiée : « deux horloges qui ne sont pas strictement identiques (donc ne provenant pas de la même source physique) doivent être considérées comme quelconques, sans aucune relation ni de fréquence ni de phase » ?

VI. Annexe

VI.A. Règles de codage VHDL

Ce paragraphe contient quelques règles élémentaires que vous devrez respecter :

- un objet statique (*constant, generic*) est écrit entièrement en majuscules
- un objet dynamique (*signal, port*) est écrit principalement en minuscules
- tout objet (*generic, port, signal*) doit, lors de sa déclaration, être suivi d'un commentaire pertinent
- les objets de type *variable* sont interdits
- toutes les entités doivent être mappées en instanciation directe (cf. *template_top.vhd*)

VI.B. ModelSim – définir l'ordre de compilation des fichiers

Allez sur la fenêtre « Project ». Si elle n'est pas visible, allez dans la fenêtre principale de ModelSim, puis dans le menu View / Project (x). Vous devriez avoir une fenêtre similaire à celle-ci :

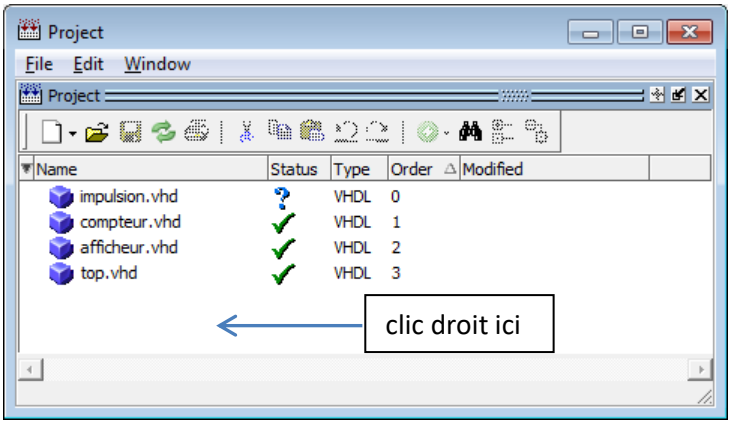


Figure 3 : ModelSim - Fenêtre Project

Puis faites un clic droit dans la partie centrale de cette fenêtre, puis Compile / Compile Ordre. Vous obtenez alors cette fenêtre :

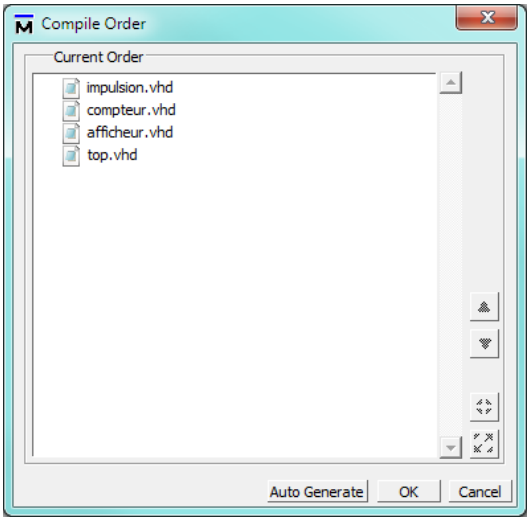


Figure 4 : ModelSim – Compile Order

Sélectionnez ensuite le fichier à déplacer puis cliquez sur la flèche appropriée.

VI.C. ModelSim – Exécuter un script

Dans la console de ModelSim (le « transcript »), pour exécuter le script « sim_ctrl.do », tapez la commande suivante :

```
do sim_ctrl.do
```