Flot de conception VLSI/FPGA Synopsys - ModelSim SE

Tutorial (version avril 2010)

Olivier Sentieys, Hélène Dubois
ENSSAT - Université de Rennes 1
sentieys@enssat.fr, dubois@enssat.fr

Sommaire

---

1. Introduction

Ce tutorial a pour objectif l'apprentissage des logiciels **Design Compiler (Synopsys)** et **ModelSim SE (Mentor Graphics)** dédiés à la conception et à la simulation d'ASIC et de FPGA. La version HTML de ce document ainsi que des fichiers d'aide au projet ou au tutorial sont accessibles à l'URL :
http://cairn.enssat.fr/enseignements/Cao/Cao.php

Le logiciel ***Design Compiler de*** **Synopsys** permet la synthèse logique grâce aux outils ***design_vision*** (en mode graphique) et ***dc_shell*** (en mode ligne). Le logiciel **ModelSim SE** permet la simulation temporelle au niveau RT (transfert de registre) ou au niveau porte, à partir des langages VHDL ou Verilog. Ces deux outils sont la référence au niveau de l'utilisation industrielle.
Le tutorial va nous permettre, dans un premier temps, d'appréhender ces logiciels en présentant ses principales composantes. Ensuite, nous nous appuierons sur un exemple concret : un compteur/décompteur 4 bits.

2. Installation de Design Compiler

2.1. Mise en place de l'environnement

- Créez un répertoire $HOME/synopsys
- Copiez l'archive tutorial.tar accessible à l'URL citée plus haut. Ensuite, celle ci peut être décompactée par la commande `tar xfv tutorial.tar` directement dans $HOME/synopsys.
- Modifiez votre fichier **.cshrc** ou **.login** en y ajoutant les lignes suivantes :

```
source /usr/local/Synopsys/environ.csh

source /usr/local/ModelSim/modeltech/environ.csh

setenv LM_LICENSE_FILE 1717@loth2:1717@loth3:1717@loth4:27000@hurin.cnfm.fr
```

- Reconnectez-vous, pour rendre effectives les modifications apportées.

Remarque : **Synopsys** gère les fichiers de manière non structurée. Il place dans le répertoire de lancement tous les fichiers de la bibliothèque associée à ce répertoire.

- IMPORTANT : placez vous **toujours** dans le répertoire $HOME/synopsys/tutorial avant de lancer un des outils de **Synopsys**. Si vous souhaitez travailler dans un autre répertoire, il faudra y copier les fichiers de configurations décrits ci dessous.

2.2- Paramétrage des fichiers de configuration du logiciel

La mise en oeuvre des logiciels de **Synopsys impose la mise en place de fichiers de configuration (*.synopsys_dc.setup et .synopsys_vss.setup*)** dans le répertoire où vous travaillez (i.e. *$HOME/synopsys/tutorial*) ou dans votre $HOME.

- Configuration minimale des outils ***Design Vision*** et ***Design Compiler*** dans le fichier ***.synopsys_dc.setup*** :

```
include  hcmos9gp.script
define_design_lib nom_de_ma_lib -path ./lib
```

Pour faciliter l'utilisation d'une bibliothèque fondeur (appelée *Design Kit*), des scripts ont été écrits. Dans votre fichier ***.synopsys_dc.setup*** le script **hcmos9gp.script** est appelé avec la commande **include hcmos9gp.script**. La bibliothèque

hcmos9gp est celle utilisée pour faire la synthèse et la simulation des composants. Elle correspond à une technologie CMOS 130 nanomètres de STMicroelectronics.

Remarques Ces fichiers de *setup* sont configurés en considérant que vous vous trouvez dans votre *$HOME/synospys/tutorial* et que la bibliothèque se situe elle aussi dans un sous-répertoire (./lib). Il vous faudra toujours travailler dans ce répertoire. Il est important d'avoir identité entre les paramètres de définitions de bibliothèque dans les fichiers ***synopsys_dc.setup*** et ***synopsys_vss.setup*** (i.e. WORK et ./lib).

La configuration du simulateur ModelSim sera vue dans la partie 5, au moment de la simulation.

3. Présentation des éléments principaux de Design Compiler

3.1. Organisation des différents fichiers utilisés par le logiciel

- Extension des fichiers sources
  - les fichiers **.v** sont des fichiers source en langage **Verilog**
  - les fichiers **.vhd** sont des fichiers source en langage **VHDL**
  - les fichiers **.edif** sont des fichiers source en langage **EDIF**
- Extension des fichiers de synthèse et de script
  - les fichiers **.sdf** (ou **.con**) sont des fichiers contenant les contraintes imposées au design ou les temps extraits après simulation
  - les fichiers **.scr** sont des fichiers script
- Extension des fichiers de rapport de compilation et de login
  - les fichiers **.rpt** (ou **.out**) sont des fichiers dans lesquels on trouve les informations sur le design à différents moments de la synthèse logique
  - les fichiers **.log** sont des fichiers contenant toutes les commandes et alias de l'application
  - les fichiers **.db** sont répertoriés dans la base de données du logiciel. C'est également l'extension par défaut de **Synopsys**
  - les fichiers **.syn** sont des fichiers intermédiaires générés durant l'analyse d'un circuit
  - les fichiers **.sim**, **.mra** et **.o** sont des fichiers intermédiaires générés par **vhdlan** et utilisé lors de la phase de simulation
  - les fichiers **.lis** sont des fichiers contenant le code source ainsi que les warnings et les erreurs générés durant l'analyse (**vhdlan**)

Remarque : **Synopsys** n'utilise comme fichiers d'entrées de ses différents outils que des fichiers de type **VHDL** (ou Verilog). Il n'existe pas de méthode graphique pour générer des composants complexes à partir de composants simples. Les fichiers issus de la synthèse logique (netlist) sont également de type **VHDL**.

4. Synthèse logique

Ce tutorial s'appuie sur l'exemple d'un compteur/décompteur 4 bits dont nous allons réaliser l'analyse complète en détaillant toutes les étapes de la synthèse logique.

4.1. Lancement de la synthèse logique

Il existe deux manières d'utiliser l'outil de synthèse :

- la première consiste en l'emploi d'une interface en mode ligne. Pour cela on utilise la commande ***dc_shell*** qui lance l'environnement (apparition du prompt **dc_shell** dans le terminal);
- la deuxième consiste en l'emploi d'une interface en mode graphique. La commande ***design_vision –dcsh_mode*** permet de lancer l'environnement. La fenêtre ***Design Vision*** apparaît alors à l'écran.

Pour les premières utilisations, il est conseillé d'utiliser l'interface graphique qui est nettement plus conviviale que l'interface en mode ligne. Toutes les commandes de ***dc_shell*** peuvent être lues au fur et à mesure de leur exécution dans la fenêtre Console en bas de l'écran. Il est également possible d'écrire des commandes de ***Design Compiler*** (dc_shell) dans la ligne « command line » sous la fenêtre ***Console***.

Dans la suite de ce tutorial, nous utiliserons l'interface en mode graphique, ***Design_vision***. Nous présenterons toutefois quelques commandes de ***Design Compiler*** importantes.

IMPORTANT : la documentation de l'ensemble des outils synopsys est accessible via la commande `sold &` sous unix.

4.2. Synthèse d'un compteur/décompteur 4 bits

4.2.1. Etude du fichier VHDL source

Le fichier VHDL de départ est : compteur.vhd

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

entity compteur is
  generic (size : integer := 16 );
  port ( reset : in Std_Logic;
         clk   : in Std_Logic;
         up    : in Std_Logic;
         load  : in Std_Logic;
         value : in integer range 0 to size-1;
         output: out integer range 0 to size-1) ;
   end compteur;

architecture comportementale of compteur is
   signal count : integer range 0 to size-1;
begin
   synchrone : process (reset,clk)
   begin
     if reset='1' then
            count<=0;
     elsif clk'event and clk='1' then
       if load='1' then
            count<=val;
       elsif up='1' then
            count <= count + 1;
       else
            count <= count - 1;
       end if ;
     end if ;
    end process ;
    output <= count;
```

```
end comportementale;
```

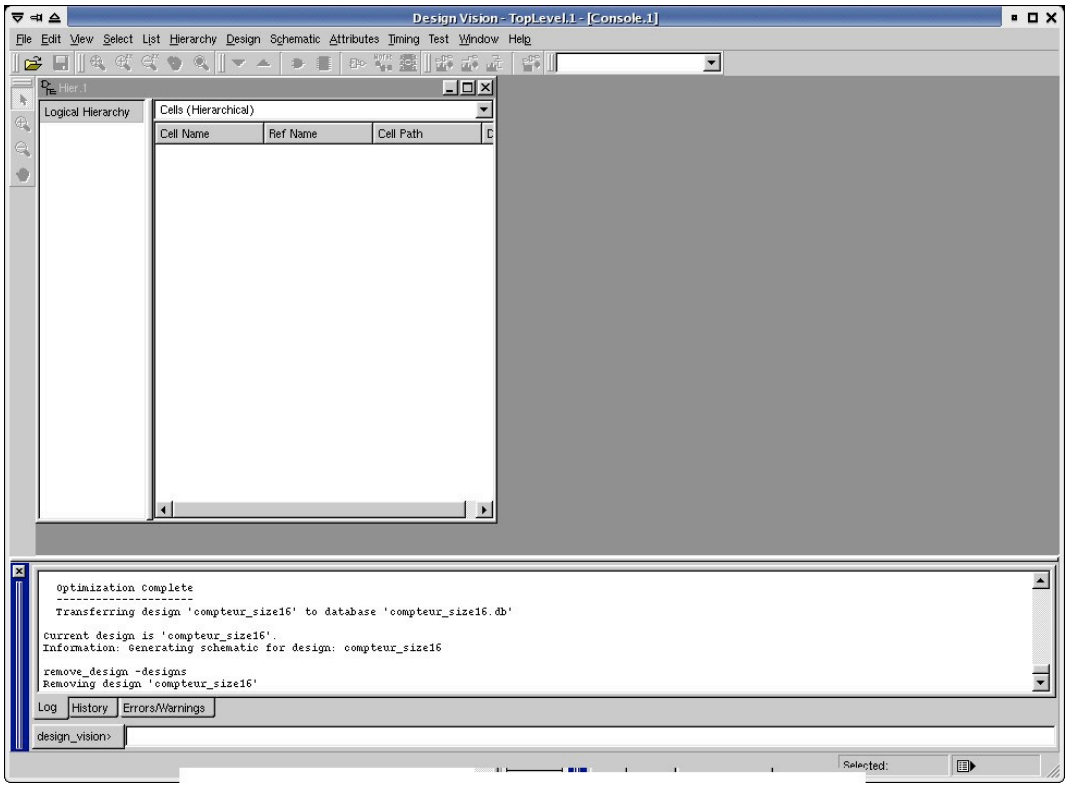
- Tout fichier VHDL (possédant l'extension **.vhd**) doit posséder au moins les lignes suivantes devant chaque **entity**

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

- Le type **STD_LOGIC_VECTOR** n'est pas compatible avec les opérations addition et soustraction. Il faut utiliser le type **INTEGER RANGE 0 to N** ou les types **SIGNED/UNSIGNED**.

4.2.2. Chargement et compilation d'un fichier VHDL dans Design Vision.

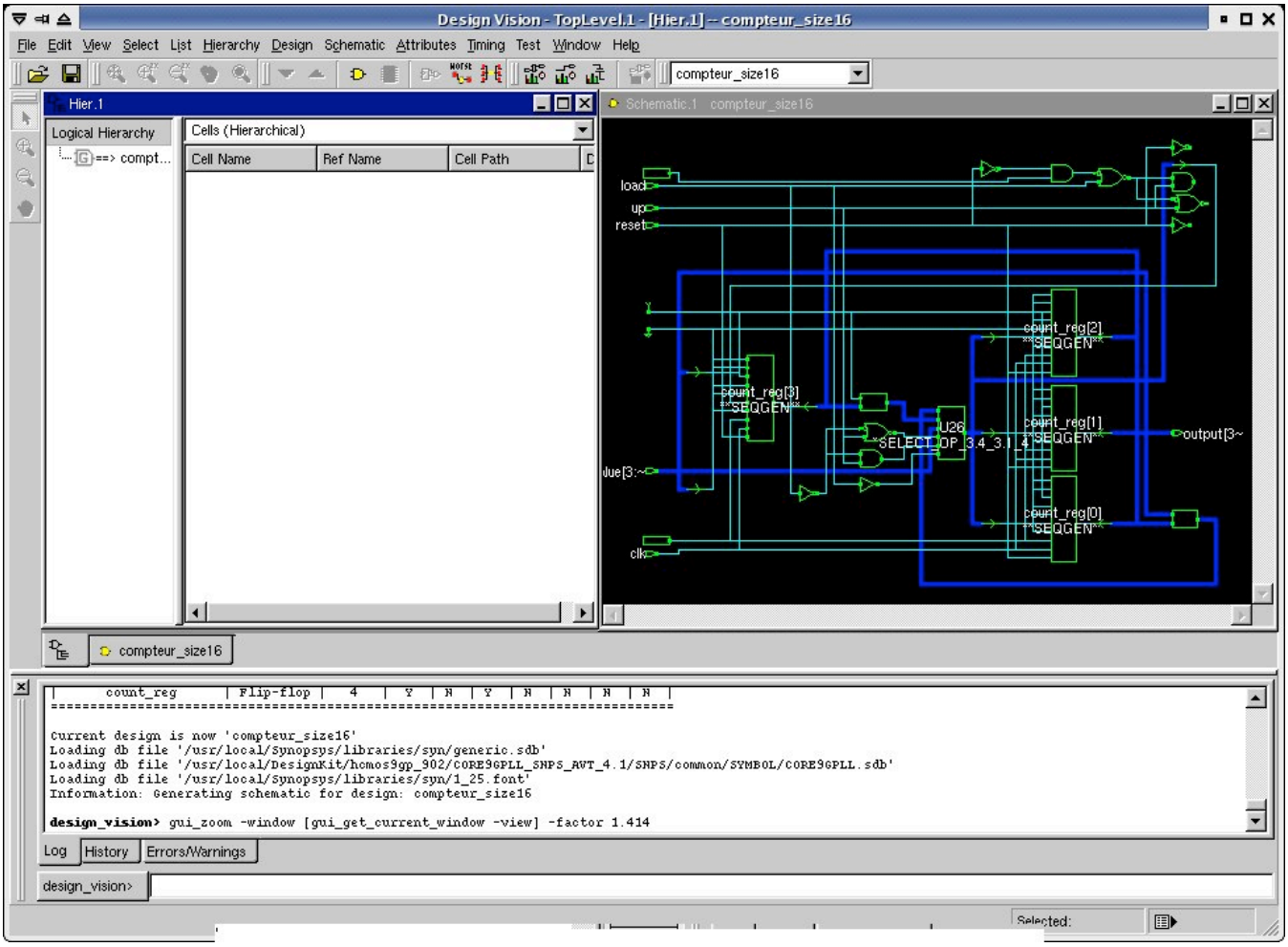
Tapez : **`design_vision -dcsh_mode`** `pour lancer l'outil graphique avec prise en compte du script` hcmos9gp.script.
On peut alors analyser le fichier **.vhd** à l'aide du menu ***File*** **(File -> Analyze)**. Sélectionnez le fichier (grace à ADD) en vérifiant que le modèle VHDL est bien choisi ainsi que la bibliothèque WORK et lancez l'analyse (OK). La syntaxe du fichier est ainsi vérifiée et des fichiers au format intermédiaires sont stockés dans le répertoire de travail **./lib**. On y trouve des fichiers pour la simulation (extension **.sim**) et des fichiers pour la synthèse (extension **.syn**).



4.2.3. Elaboration d'un modèle VHDL

L'élaboration correspond à une synthèse indépendante de la technologie utilisée (elle est non optimisée), il s'agit de la vue RTL. On utilise dans le menu ***File***, le chemin **(File -> Elaborate)**. Choisissez le circuit à synthétiser dans la bibliothèque de travail WORK, précisez la valeur du générique size, puis lancez l'élaboration (OK). Le schéma correspondant peut être alors visualisé par la commande Schematic/NewDesign Schematic view..

Exemple du compteur non optimisé



On notera que le design ne s'appelle plus compteur, mais compteur_size16.

4.2.4 Autre méthode : utilisation de la commande Read

On peut également utiliser la commande ***read*** pour faire ces deux étapes de *analyze* et *elaborate*. Dans ce cas, on n'a pas à donner la valeur de size et le nom du design n'est pas modifié (compteur).

4.2.4. Sauvegarde d'un résultat de synthèse

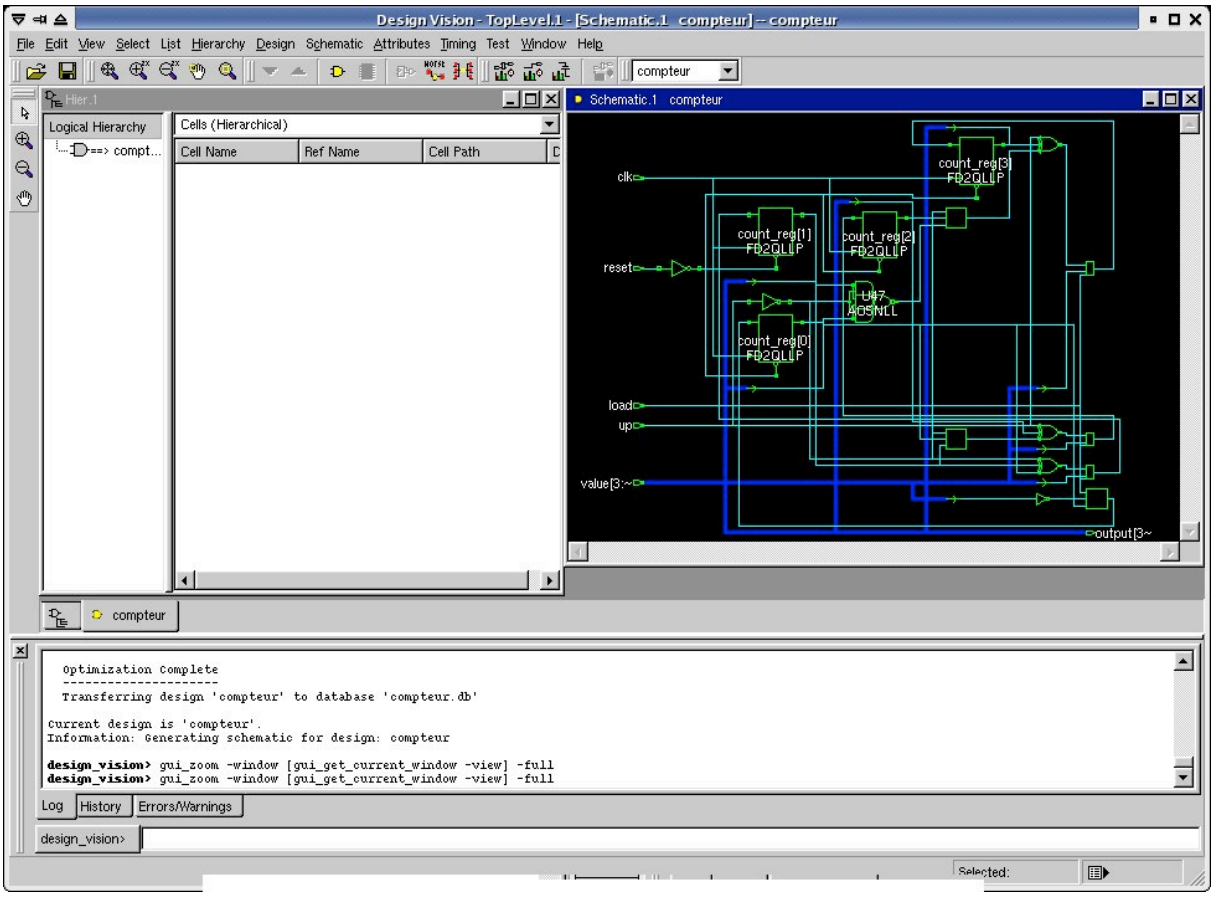
Après l'élaboration, le résultat obtenu peut être sauvegardé afin d'être utilisé sur plusieurs cibles technologiques ou dans le but d'être optimisé suivant plusieurs critères différents. Dans le menu ***File*** **(File -> Save)** sauvegarde le circuit au format **.db**, qui est le format interne à **Synopsys**. Pour notre exemple, il va générer un fichier **compteur.db**.

Pour la simulation temporelle qui va suivre, il faut sauvegarder le fichier synthétisé au format **.vhd en changeant de nom** pour ne pas écraser celui d'origine : **compteur_gate.vhd**.

4.2.5. Compilation et Optimisation d'un circuit

Pour se rendre compte de l'importance de l'étape d'optimisation il est nécessaire d'observer les résultats de compilation **avant optimisation** (cf 4.2.3- compteur non optimisé). On applique maintenant l'étape d'optimisation pour la technologie ciblée **(Design -> Compile Design)** toujours dans la fenêtre ***Design Vision*.** Ouvrez ensuite le nouveau schéma*.*

On observe le résultat suivant qui montre que le design est maintenant une connexion de portes de la bibliothèque ***hcmos9gp***.



- IMPORTANT : la documentation pdf décrivant la bibliothèque de cellules est accessible à : /usr/local/DesignKit/hcmos9gp_902/CORE9GPLL_SNPS_AVT_4.1/doc/databook_1.2V/CORE9GPLL _1.2V_databook.pdf

Il est maintenant nécessaire de sauvegarder le résultat de synthèse en vue d'une simulation de niveau porte ultérieure. Pour cela :

- Exécutez les commandes suivantes dans la ligne de commande design_vision>.
- `change_names -hierarchy -rule vhdl` permet de préparer la *netlist* pour une sauvegarde en VHDL.
- `write_sdf compteur.sdf` permet de sauvegarder les paramètres temporels de simulation au format sdf V2.1.
- Sauvegardez votre compteur optimisé (File -> Save) **compteur_gate.db**.
- Sauvegardez votre compteur optimisé (File -> Save As). Choisissez le format VHDL, sélectionnez "Save All Design in Hierarchy"et appelez votre fichier **compteur_gate.vhd**.

4.2.6. Mise en oeuvre de contraintes

- Les contraintes sont des déclarations explicites qui définissent les objectifs à atteindre pour le circuit considéré. Le logiciel **SYNOPSYS** propose des contraintes de temps, de surface et d'énergie. Par défaut, l'outil minimisera la surface en respectant une éventuelle contrainte de temps.

Lors de la phase d'**optimisation**, **Design Vision** (ou **Design Compiler**) utilise deux modèles de contraintes:

- les *contraintes implicites*, qui sont imposées par la bibliothèque technologique;
- les *contraintes d'optimisation* (ou *contraintes explicites*) imposées par l'utilisateur.

Remarque **Design Compiler** essaie de faire converger les deux modèles de contraintes. Toutefois, en cas de non adéquation, ce sont les contraintes implicites qui sont prioritairement retenues.

Il existe trois façons différentes de spécifier des contraintes:

- écriture de ligne de commande dans **dc_shell>** ou appel d'un script par la commande *include*. C'est le moyen le plus efficace et donc à privilégier.

```
/* Create user defined variables */
designer = "Olivier Sentieys"
company = "ENSSAT"
create_clock {clk} -name clk -period 10.0 -waveform {0 5}
/* Time Budget */
set_dont_touch_network clk
set_clock_uncertainty 0.2 {clk}
set_input_delay 1.0 -max -clock clk {up load val}
set_output_delay 0.5 -max -clock clk {sortie}
/*  Area Constraint */
set_max_area 5000
/* Operating Environment */
set_wire_load_model -name area_18Kto24K
set_driving_cell -library CORE9GPLL -lib_cell IVLL -pin Z {up load val}
set_load load_of(CORE9GPLL/IVLLX4/Z) {sortie}
report_constraints
```

- sélection des menus de contraintes dans la fenêtre **Design Vision**;
  - **(Attributes -> Optimization Constraints -> Design Constraint)** pour spécifier les contraintes en terme de **Surface** (Area), de **Puissance** (Power) ou encore de **Sortance** (Fanout)
  - **(Attributes -> Optimization Constraints -> Timing Constraint)** pour spécifier les contraintes en terme de **Temps de traversée** entre une entrée et une sortie et de **fréquence d'horloge**
- écriture de fichiers source **.vhd** contenant des contraintes (cf.exemple de code ci dessous).

Exemple de l'entity du fichier **compteur.vhd** avec contraintes

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

entity compteur is
port ( reset  : in Std_Logic;
       clk    : in Std_Logic;
       up     : in Std_Logic;
       load   : in Std_Logic;
       val    : in integer range 0 to 15;
       sortie : out integer range 0 to 15) ;
       attribute MAX_AREA of compteur : entity is 70.0;
       attribute MAX_DELAY of sortie : signal is 2.0;
end compteur;
```

4.2.7. Caractéristiques du circuit

- Pour obtenir les paramètres du circuit synthétisé dans **Design Vision :** (Design-> Report).

- Ces rapports peuvent être vus à l'écran ou sauver dans des fichiers txt.

- Par sélection, vous obtenez les paramètres de surface (Area), de consommation (Power), de temps (Timing).

- Sélectionnez Set Options pour faire apparaître la fenêtre ci dessus de droite, puis sélectionnez All Register Data Pins. Pour plus d'informations voir la documentation de synopsys (`sold`).
- Par le menu Analysis->Highlight->Critical Path, vous pouvez tracer le chemin critique du circuit. Pour plus d'informations voir la documentation de synopsys (`sold`).

4.2.8. Script *dc_shell* de compilation d'un fichier VHDL

Après s'être placé en mode **Design Compiler** (commande ***dc_shell*** dans une fenêtre terminal), il suffit de lancer le script shell suivant nommé compteur.scr (commande **include compteur.scr** dans la fenêtre précédente). Ne pas lancer un Design Vision et un dc_shell en même temps. Il est aussi possible de lancer cette commande dans design_vision.

Le fichier **compteur.scr** contient les instructions suivantes:

```
analyze -format vhdl compteur.vhd;
      -- analyse synthaxique du fichier source .vhd
elaborate compteur;
      -- creation du design a partir d'une entite et
      -- d'une architecture VHDL.
      -- "compteur" represente ici le nom de l'entity
uniquify;
      -- elimine toutes les instances du design courant
      -- dans la hierarchie en creant un design unique
      -- pour chaque instance
check_design;
      -- verification de la validite logique du design
      -- (en cas d'erreurs presence de warning)
create_clock {clk} -name clk -period 10.0
      -- contrainte sur la période d'horloge
      -- (à mettre pour obtenir le chemin critique)
compile  -map_effort high;
      -- optimisation du design
      -- rem: il y a 3 modes d'effort de compilation :
      -- low, medium et high
report_area
report_power
report_timing -path full -delay max -max_paths 1 -sort_by group

change_names -hierarchy -rule vhdl
write_sdf compteur.sdf
write -hierarchy -output compteur_gate.db;
write -format vhdl -hierarchy -output compteur_gate.vhd;
      -- sauvegarde de la netlist sous forme de fichier VHDL
```

Vous pourrez modifier ce script pour compiler vos design pendant le projet ; **le gain de temps est réellement appréciable** par rapport à l'utilisation de l'environnement graphique design_vision.

5. Simulation logique avec ModelSim SE de Mentor Graphics

5.1. Introduction et lancement des outils

L'invocation des outils de simulation se fait par la commande **vsim**.

Bien que cela ne soit pas obligatoire, il est conseillé de travailler pas seulement avec des fichiers, mais avec un projet qui contiendra les fichiers mais aussi des configurations liées au projet.

- File->Change Directory et choisir le répertoire courant ($HOME/synopsys/tutorial)
- File->New->Project et donner le nom "tutorial" à ce projet dans le répertoire courant
- Exécuter enfin la commande qui fait le lien avec la bibliothèque de description des composants niveau porte pour Modelsim :

vmap CORE9GPLL
/usr/local/DesignKit/hcmos9gp_902/CORE9GPLL_SNPS_AVT_4.1/MENTOR_MODELSIM/lib_VITAL

Vous pouvez taper cette commande, ou utiliser le fichier init.do qui vous est fourni en tapant la commande do initlib.do. Le simulateur est maintenant configuré.

Avant d'effectuer une simulation, il est nécessaire de créer plusieurs fichiers de test qui constituent le testbench.

5.2. Création des fichiers

Pour simuler un circuit, il faut decriptions VHDL : un fichier de code, un fichier de simulation, un fichier de configuration.

Nous utiliserons le même exemple de fichier code : **compteur.vhd**

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

entity compteur is
   port ( ... ) ;
end compteur;

architecture comportementale of compteur is
   ....
end comportementale;
```

Le fichier de simulation (ou *testbench*) est un fichier **.vhd**, il doit toujours être structuré de la même façon.
Voici un exemple de fichier de simulation : **sim_compteur.vhd**

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

ENTITY bench IS
END bench;

ARCHITECTURE test OF bench IS
COMPONENT compteur
  generic (size : integer := 16 );
  port ( reset : in Std_Logic;
         clk   : in Std_Logic;
          up    : in Std_Logic;
          load  : in Std_Logic;
          value   : in integer range 0 to size-1;
         output : out integer range 0 to size-1) ;
end COMPONENT;
```

```
constant Tclk : time := 10 ns;
signal clk_cpt : std_logic:= '0';
signal reset_cpt: std_logic:= '1';
signal up_cpt  : std_logic:= '1';
signal load_cpt: std_logic:= '0';
signal val_cpt : integer range 0 to 15 := 9;
signal out_cpt : integer range 0 to 15;

begin

cpt : compteur generic map(16) port map (reset_cpt, clk_cpt, up_cpt, load_cpt,
val_cpt,out_cpt);

GENERATE_CLOCK : process
  begin
         clk_cpt<= '0';
         WAIT FOR Tclk/2;
         clk_cpt<= '1';
         WAIT FOR Tclk/2;
  end process GENERATE_CLOCK;

CPT_SIM : process
  begin
         reset_cpt<= '1';
         up_cpt<= '1';
         load_cpt<= '0';
         WAIT FOR 3*Tclk;
         reset_cpt<= '0';
         WAIT FOR 10*Tclk;
         load_cpt<= '1';
         WAIT FOR 10*Tclk;
         load_cpt<= '0';
         WAIT FOR 30*Tclk;
         up_cpt<= '0';
         WAIT FOR 40*Tclk;

         wait;
  end process CPT_SIM;


end test;
```

Le fichier de configuration est aussi un **.vhd**, il est obligatoire mais peut être inclus dans le fichier précédent. Voici un exemple de fichier de configuration : **cfg_compteur.vhd**

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

configuration compteur_cfg of bench is
   for test
      for cpt:compteur
            use entity work.compteur(comportementale);
      end for;
   end for;
end compteur_cfg;
```

Remarque: Les fichiers de configuration et de testbench peuvent être utilisés aussi bien au niveau RTL qu'au niveau porte. Il suffit de modifier le nom de l'architecture dans le fichier *cfg_compteur.vhd*. En effet la netlist générée par Synopsys lors de la synthèse a pour nom d'architecture SYN_nom_de_votre_archi, donc ***SYN_comportementale*** ici au lieu de ***comportementale***

```
configuration compteur_cfg_gate of bench is
 for test
    for cpt:compteur
          use entity work.compteur(SYN_comportementale);
    end for;
 end for;
end compteur_cfg_gate;
```

5.3. Analyse des fichiers

Avant de tester le circuit, il faut l'analyser. Pour cela il nous faut ajouter dans le projet les fichiers VHDL à compiler.

- File->Add to project->compteur.vhd
- Répéter l'opération pour les deux autres fichiers .vhd
- Précisez l'ordre de compilation des fichiers grace à la commande Compile/compile Order.
- Project->Compile All

Il est bien sur possible de compiler les fichiers un par un en cas d'erreur. Un éditeur de texte s'ouvre lorsque l'on double-clique sur les fichiers .vhd.

5.4. Lancement de la simulation

Choisissez l'onglet Library, puis, en double-cliquant sur la configuration compteur_cfg, la simulation se charge. Ceci est équivalent à : Simulate->Start Simulation. Choisir ensuite work.compteur_cfg

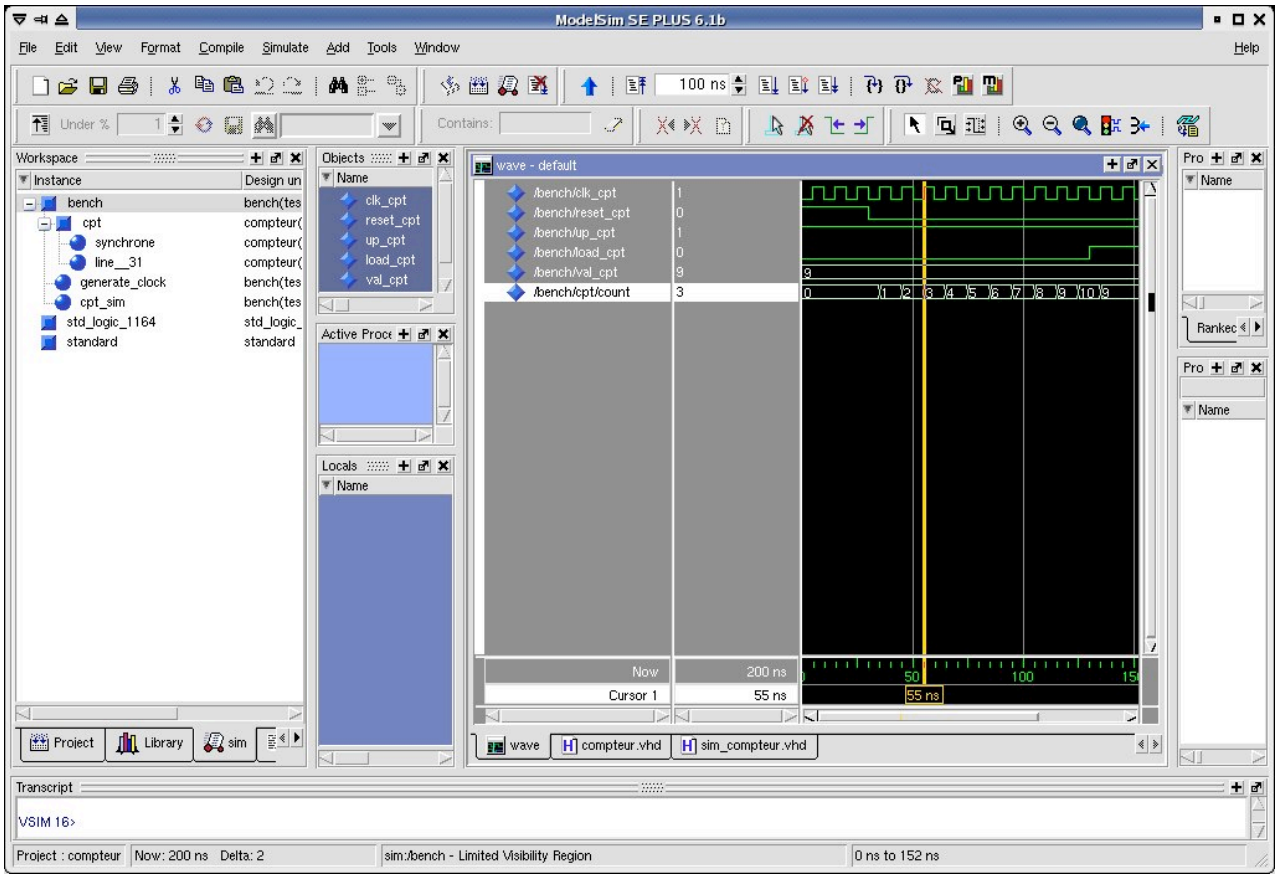
- View/Debug Window/ All permet de faire apparaître l'ensemble des fenêtres de simulation .

Avant de continuer la simulation, une petite explication de cet outil s'impose. L'écran est composé de plusieurs parties:

- La fenêtre WORK SPACE (à gauche) avec ses onglets, qui permet d'accéder aux fichiers, aux entités compilées dans les biblitohèques, etc.
- Le code vhdl
- Les fenêtres "Active Process", "Locals", "Objects" et "Dataflow" permettent de sélectionner les signaux à visulaiser.
- La fenêtre "wave" dessine les chronogrammes

Nous allons maintenant continuer la simulation.

- Dans le fenêtre "Objects", sélectionnez l'ensemble des signaux et « tirez » les dans la fenêtre « Wave »
- Simulate -> Run->Run 100ns permet de faire avancer la simulation de 100 ns (ou utilisez les icônes)
- Répéter cette opération tant que nécessaire... Vous devez voir les sorties de votre compteur s'incrémenter.



Au bout de quelques pas de simulation, vous allez avoir une erreur : d'où vient-elle ? Est elle gênante ?

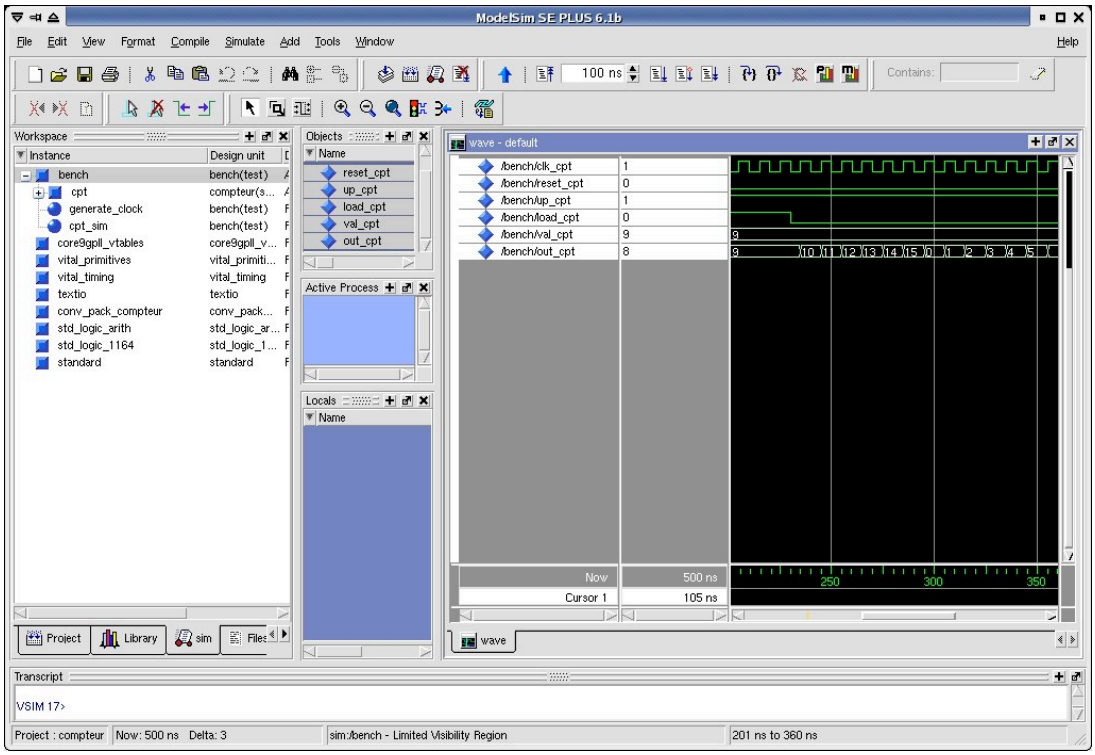
5.5. Simulation au niveau porte

Maintenant que vous venez de faire la simulation au niveau *RTL*, il faut faire la simulation au niveau *porte* avec la netlist **compteur_gate.vhd**. Pour cela supprimez le fichier **compteur.vhd** du projet, puis ajoutez le fichier **compteur_gate.vhd** dans le projet en cours, puis modifiez le fichier *cfg_compteur.vhd* (comportementale en SYN_comportementale) ou créez un nouveau fichier de configuration *cfg_compteur_gate.vhd*. Recompilez l'ensemble. La configuration instancie maintenant la netlist au niveau porte et non plus la spécification RTL d'origine contenue dans compteur.vhd.

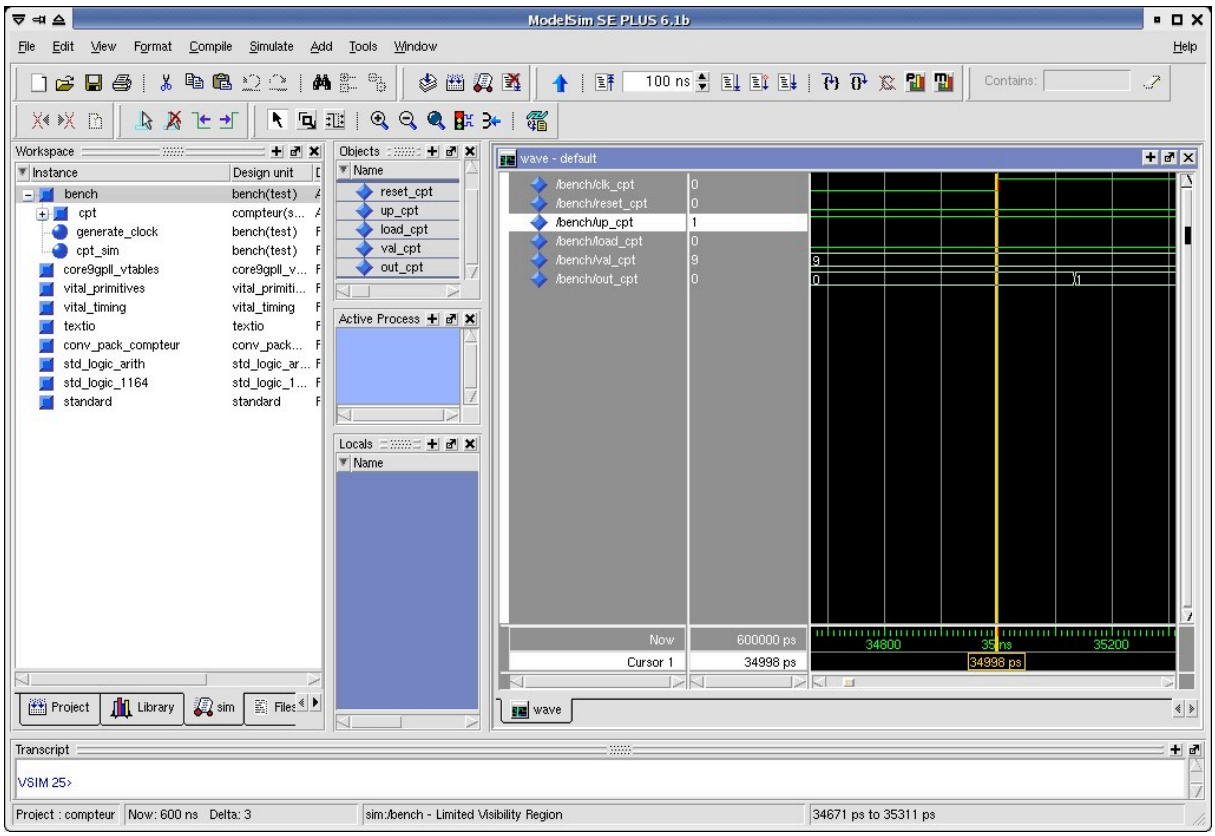
Rappel : il est en outre nécessaire d'avoir généré un fichier au format SDF (Standard Delay Format) **compteur.sdf** sous **`design_vision (paragraphe 4.2.5)`**. Ce fichier contient les délais des portes et les estimations des délais d'interconnexions.

- Simulate -> Start Simulation
- Dans l'onglet **Design**, chargez la configuration du test ***compteur_cfg*** en choisissant **ps** comme résolution du simulateur.
- Dans l'onglet **SDF**, chargez (bouton Add...) le fichier SDF associé au composant à tester (/bench/cpt/).
    - Browse->compteur.sdf
    - Apply to Region->bench/cpt (on applique les temps seulement au composant « cpt » de l'entité « bench »)
    - Cocher l'option *reduce errors to warnings*

Visualisez comme au niveau RTL les signaux d'entré et de sortie du compteur.



Sur cette simulation, l'erreur précédente au bout de quelques cycles de simulation disparait. Pourquoi ?
A partir de ce moment, la simulation prend en compte les portes logiques qui constituent la netlist du compteur et les informations de délais fournies par le fichier SDF. Relancez la simulation en prenant une résolution en ps (onglet design de start simulation) ; en zoomant sur un front d'horloge vous voyez maintenant apparaître des temps de propagation entre l'horloge et les sorties du compteur.



6 Comparaison des différents styles d'écriture en VHDL

Quatre façons de décrire un registre à sortie trois-états sont proposées dans le fichier VHDL **dffs.vhd**.

```
  ------------------------------------------------------------------------
-- A POSITIVE EDGE-TRIGGERED D FLIP-FLOP WITH ACTIVE LOW ASYNCHRONOUS
--                  RESET AND ACTIVE HIGH OUTPUT ENABLE
------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity REG is
port (   Q    : out STD_LOGIC_VECTOR(3 downto 0);
         D    : in  STD_LOGIC_VECTOR(3 downto 0);
         CLK   : in  STD_LOGIC;
         Reset,Oe : in  STD_LOGIC  );
end REG;
-- Premier style d'écriture
architecture REG1 of REG is
  signal Qi : STD_LOGIC_VECTOR(3 downto 0);
begin
     DF : process(CLK) begin
       if CLK'event AND CLK = '1' then
           Qi <= D;
       end if;
     end process DF;
     CB : process(Reset,Oe,Qi)  begin
       if Reset = '0' AND Oe = '1' then
          Q <= (others => '0');
       elsif Reset = '1' AND Oe = '0' then
          Q <= (others => 'Z');
       elsif Reset = '1' AND Oe = '1' then
          Q <= Qi;
       end if;
     end process CB;
end REG1;
-- Deuxieme style d'écriture
architecture REG2 of REG is
  signal Qi : STD_LOGIC_VECTOR(3 downto 0);
begin
     DF : process(CLK) begin
        if CLK'event AND CLK = '1' then
           Qi <= D;
        end if;
     end process DF;
     CB : process(Reset,Oe,Qi) begin
       if Reset = '0' AND Oe = '1' then
          Q <= (others => '0');
       elsif Reset = '1' AND Oe = '0' then
          Q <= (others => 'Z');
       elsif Reset = '0' AND Oe = '0' then
          Q <= (others => '0');
       else
          Q <= Qi;
       end if;
     end process CB;
end REG2;
-- Troisieme style d'écriture
architecture REG3 of REG is
```

```
  signal Qi : STD_LOGIC_VECTOR(3 downto 0);
begin
     DF : process(CLK, Reset) begin
       if Reset = '0' then
           Qi <= (others => '0');
       elsif CLK'event AND CLK = '1' then
          if Oe = '0' then
               Q <= (others => 'Z');
          else
               Qi <= D;
          end if;
       end if;
     end process DF;
     CB : process(Reset,Oe,Qi)  begin
       if Reset = '0' AND Oe = '1' then
          Q <= (others => '0');
       elsif Reset = '1' AND Oe = '0' then
          Q <= (others => 'Z');
       elsif Reset = '0' AND Oe = '0' then
          Q <= (others => '0');
       else
          Q <= Qi;
       end if;
     end process CB;
end REG3;
-- Quatrieme style d'écriture
architecture REG4 of REG is
  signal Qi : STD_LOGIC_VECTOR(3 downto 0);
begin
     DF : process(CLK, Reset)   begin
       if Reset = '0' then
           Qi <= (others => '0');
       elsif CLK'event AND CLK = '1' then
           Qi <= D;
       end if;
     end process DF;
     CB : process(Oe,Qi) begin
       if Oe = '0' then
          Q <= (others => 'Z');
       else
          Q <= Qi;
       end if;
     end process CB;
end REG4;
```

- Effectuez la synthèse puis l'optimisation de chaque registre. Pour cela effectuez les commandes **read** de Design Vision sur le fichier **dffs.vhd**. Effectuer ensuite sur chaque architecture la compilation. Relevez les différences entre les résultats de synthèse. En déduire quelques règles pour la description des circuits séquentiels et des circuits combinatoires.
- Utilisez la fenêtre **Report** pour extraire les paramètres temporels du registre REG4.

Il ne vous reste plus qu'à faire vos propres composants dans le cadre de votre projet, en oubliant pas ce que vous venez de voir dans ce tutorial, ni **les dix commandements de la conception de circuits numériques intégrés**.

Bon courage!