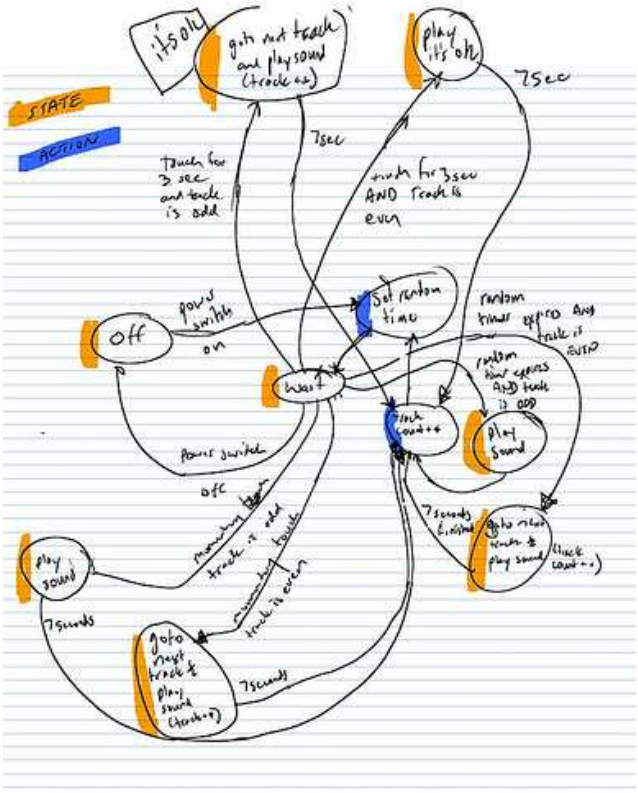
SIN-FPGA – DESCRIPTION PAR MACHINE A ETATS



Documents ressources: http://www.altera.com/literature/lit-index.html
Introduction to Quartus II : intro_to_quartus2.pdf
Documentation QUATUS : quartusii_handbook.pdf
Documentation KIT DE2 : DE2_UserManuall.pdf
Mode d'emploi du simulateur : ModelSim Tutorial : modelsim_tutorial_ug.pdf
Data sheet : http://www.altera.com/literature/lit-cyc2.jsp

Logiciels :
ALTERA QUARTUS II et Mentor Graphic ModelSim :
https://www.altera.com/download/software/quartus-ii-we
https://www.altera.com/download/software/modelsim

Matériel : Carte ALTERA DE2 : http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html

QUARTUS dispose d'un outil de développement graphique de description par machine à états. Après le dessin, la machine à états est codée par QUARTUS en langage de haut niveau (VHDL ou VERILOG)

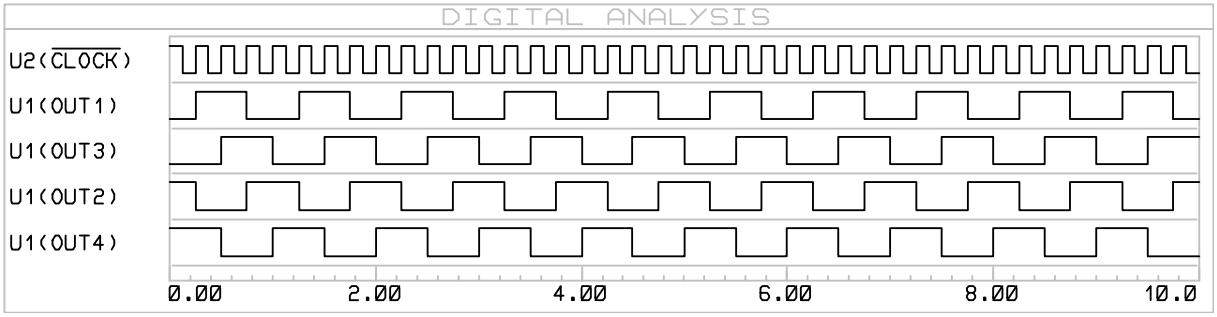
La validation du comptage ou du décomptage des véhicules entrant dans le parking (le projet support) nécessite un séquencement particulier avec une réaction sur front montant, le transfert d'une impulsion de comptage ou décomptage si le nombre de place le permet.
La description par machine à états est alors particulièrement adaptée et fera l'objet du dernier exercice de ce TP.
Le TP fait appel à des connaissances simples du langage VHDL. Consulter le livre de J.Weber et S.Moutault http://books.google.fr/books?id=AKoIOwjcqnUC

1) Résoudre un problème séquentiel par MAE dans QUARTUS II

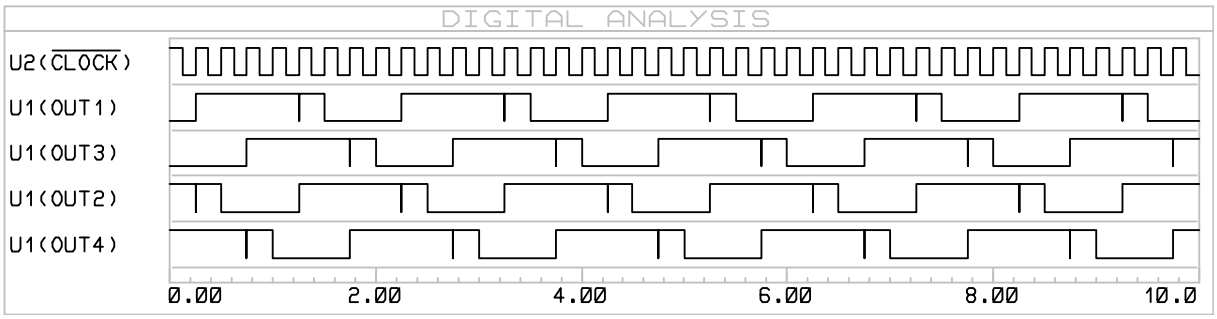
On désire réaliser la logique d'une commande de moteur pas à pas avec la possibilité d'effectuer la rotation en pas entier ou en demi-pas. Le moteur tournera uniquement dans le sens horaire.
http://fr.wikipedia.org/wiki/Moteur_pas à pas

Phases de commande des 4 bobines du moteur, le problème est similaire avec un moteur bipolaire.

Pas entier :



Demi-pas

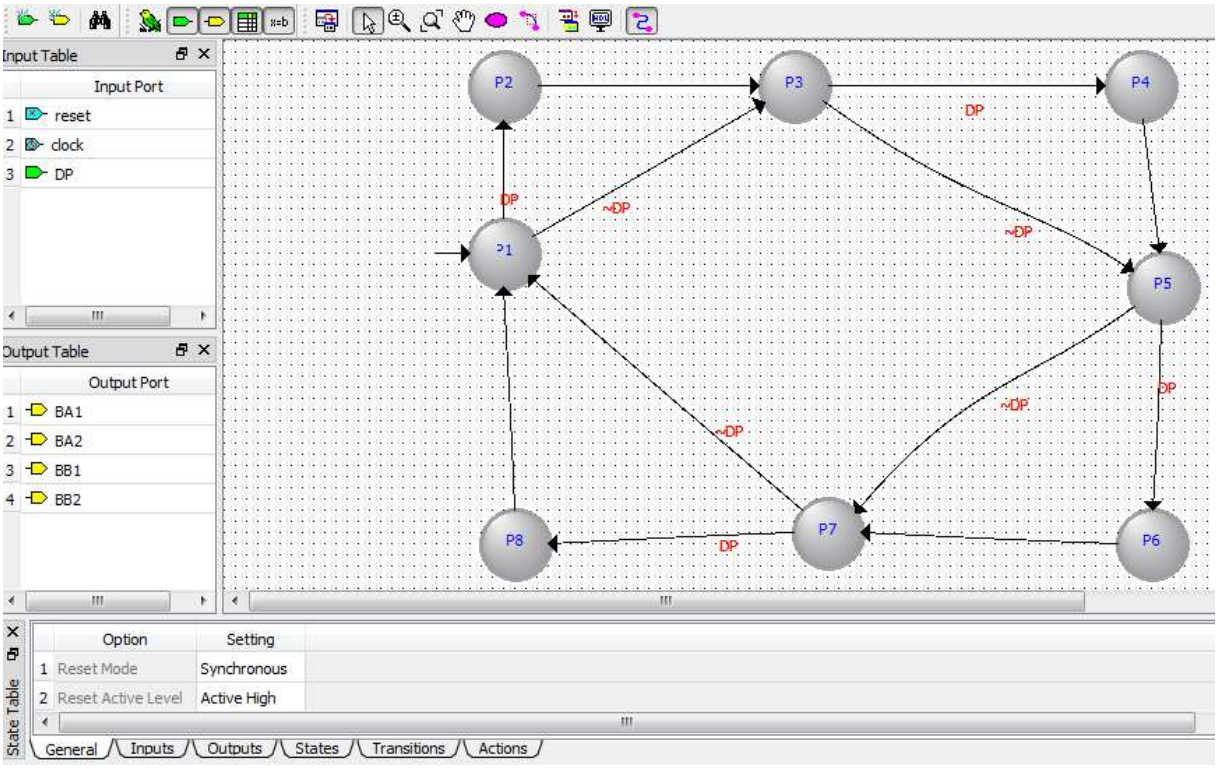


Les chronogrammes ci-dessus font apparaitre les quatre phases du mode pas entier et les huit phases du mode demi-pas.

2) Description par machine à état : commande de moteur pas à pas

Créer un projet Quartus appelé motpp_mae pour un FPGA EP2C35672C6 (équipant la carte DE2)

Cliquer « File – New » ou  puis sélectionner «State Machine File »
Une fenêtre de l'éditeur graphique de machine à état est créé



En suivant l'architecture ci-dessus :

- Créer l'entrée DP qui permettra le choix entre pas entier et demi-pas (Symbole vert)
- Créer les quatre sorties pour les bobines (Symboles jaunes)
- Placer les 8 états de la machines (cercle rose) et les renommer (double clic), l'état P1 possède par défaut une flèche sur la gauche, c'est l'état après la remise à zéro de la machine.
- Placer les liens entre les états (icône lacet), certaines transitions sont conditionnelles, double-clic sur le lien permet d'éditer les conditions.

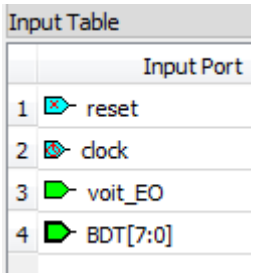
ex :

DP : la transistion est valide pour DP=1

~DP : la transistion est valide pour DP=0

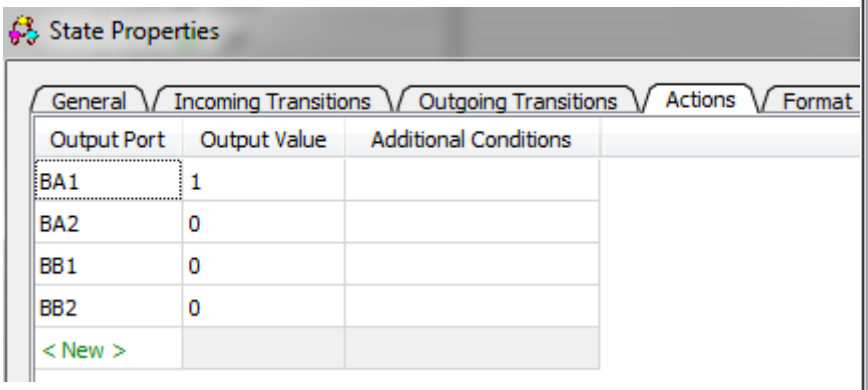
OPERATEURS	
==	Est égale
!=	Est different
<=	Est inférieur ou égal
<	Est strictement inférieur
>=	Est supérieur ou égal
>	Est strictement supérieur
&	Et logique
|	Ou logique
^	Ou exclusif
~&	NAND
~|	NOR
~^	NON Ou exclusif
~	NON

Pour créer un bus (ex :BDT) en entrée ou en sortie :
BDT[7:0] (donc ici 8bits)



Il est alors possible de créer des conditions de transition sur ce bus.
Ex : BDT==128 ou BDT>= 'b010011100

Indiquer l'état des sorties pour chaque état de la machine, conformément aux chronogrammes ci-dessus.
Double clic sur l'état puis onglet « action ».



La machine à états est terminée, créer le fichier VHDL correspondant, en cliquant sur  .
Le fichier généré sera utilisé lors de la simulation, dans le cas d'un projet ayant de multiples sources il est nécessaire de faire apparaitre ce fichier dans le projet.
Dans le project Navigator, onglet Files, Clic-Droit sur Files puis ajouter le fichier motpp.vhd.

Il est indispensable de vérifier la description VHDL, des connaissances minimum du langage sont donc nécessaires. L'analyse d'une machine à état est relativement simple, identifier dans le programme ci-dessous :

- La déclaration des entrées, sorties et des signaux.
- La partie « mise à jour des états »
- L'effet du RESET sur les sorties
- Pour chaque état, la ou les conditions de passage à un autre état, l'état des sorties.

State Table

	Output Port	Output Value	In State
1	BA1	0	P1
2	BA1	1	P2
3	BA1	1	P3
4	BA1	1	P4
5	BA1	0	P5
6	BA1	0	P6
7	BA1	0	P7
8	BA1	0	P8
9	BA2	0	P1
10	BA2	0	P2
11	BA2	0	P3
12	BA2	1	P4
13	BA2	1	P5
14	BA2	1	P6
15	BA2	0	P7
16	BA2	0	P8
17	BB1	0	P1
18	BB1	0	P2
19	BB1	0	P3
20	BB1	0	P4
21	BB1	0	P5
22	BB1	1	P6
23	BB1	1	P7
24	BB1	1	P8
25	BB2	1	P1
26	BB2	1	P2
27	BB2	0	P3
28	BB2	0	P4
29	BB2	0	P5
30	BB2	0	P6
31	BB2	0	P7
32	BB2	1	P8

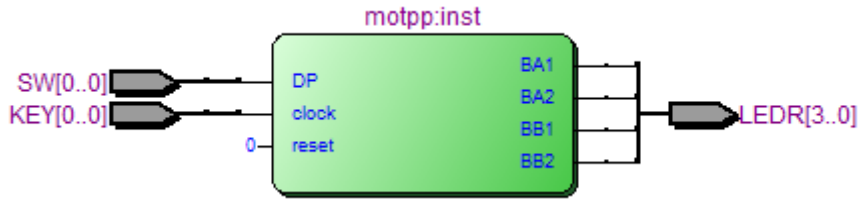
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY motpp IS
    PORT (
        reset : IN STD_LOGIC := '0';
        clock : IN STD_LOGIC;
        DP : IN STD_LOGIC := '0';
        BA1 : OUT STD_LOGIC;
        BA2 : OUT STD_LOGIC;
        BB1 : OUT STD_LOGIC;
        BB2 : OUT STD_LOGIC
    );
END motpp;

ARCHITECTURE BEHAVIOR OF motpp IS
    TYPE type_fstate IS (P1,P2,P3,P4,P5,P6,P7,P8);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;
```

```
PROCESS (fstate,reset,DP)
BEGIN
   IF (reset='1') THEN
       reg_fstate <= P1;
       BA1 <= '0';
       BA2 <= '0';
       BB1 <= '0';
       BB2 <= '0';
   ELSE
       BA1 <= '0';
       BA2 <= '0';
       BB1 <= '0';
       BB2 <= '0';
       CASE fstate IS
          WHEN P1 =>
              IF ((DP = '1')) THEN reg_fstate <= P2;
              ELSIF (NOT((DP = '1'))) THEN reg_fstate <= P3;
              -- Inserting 'else' block to prevent latch inference
              ELSE reg_fstate <= P1;
              END IF;
              BA1 <= '0';
              BA2 <= '0';
              BB1 <= '0';
              BB2 <= '1';
          WHEN P2 =>
              reg_fstate <= P3;
              BA1 <= '1';
              BA2 <= '0';
              BB1 <= '0';
              BB2 <= '1';
          WHEN P3 =>
              IF ((DP = '1')) THEN reg_fstate <= P4;
              ELSIF (NOT((DP = '1'))) THEN reg_fstate <= P5;
              -- Inserting 'else' block to prevent latch inference
              ELSE reg_fstate <= P3;
              END IF;
              BA1 <= '1';
              BA2 <= '0';
              BB1 <= '0';
              BB2 <= '0';
          WHEN P4 =>
              reg_fstate <= P5;
              BA1 <= '1';
              BA2 <= '1';
              BB1 <= '0';
              BB2 <= '0';
          WHEN P5 =>
              IF ((DP = '1')) THEN reg_fstate <= P6;
              ELSIF (NOT((DP = '1'))) THEN reg_fstate <= P7;
              -- Inserting 'else' block to prevent latch inference
              ELSE reg_fstate <= P5;
              END IF;
              BA1 <= '0';
              BA2 <= '1';
              BB1 <= '0';
              BB2 <= '0';
          WHEN P6 =>
              reg_fstate <= P7;
              BA1 <= '0';
              BA2 <= '1';
              BB1 <= '1';
              BB2 <= '0';
          WHEN P7 =>
              IF ((DP = '1')) THEN reg_fstate <= P8;
              ELSIF (NOT((DP = '1'))) THEN reg_fstate <= P1;
              -- Inserting 'else' block to prevent latch inference
              ELSE reg_fstate <= P7;
              END IF;
              BA1 <= '0';
              BA2 <= '0';
              BB1 <= '1';
```

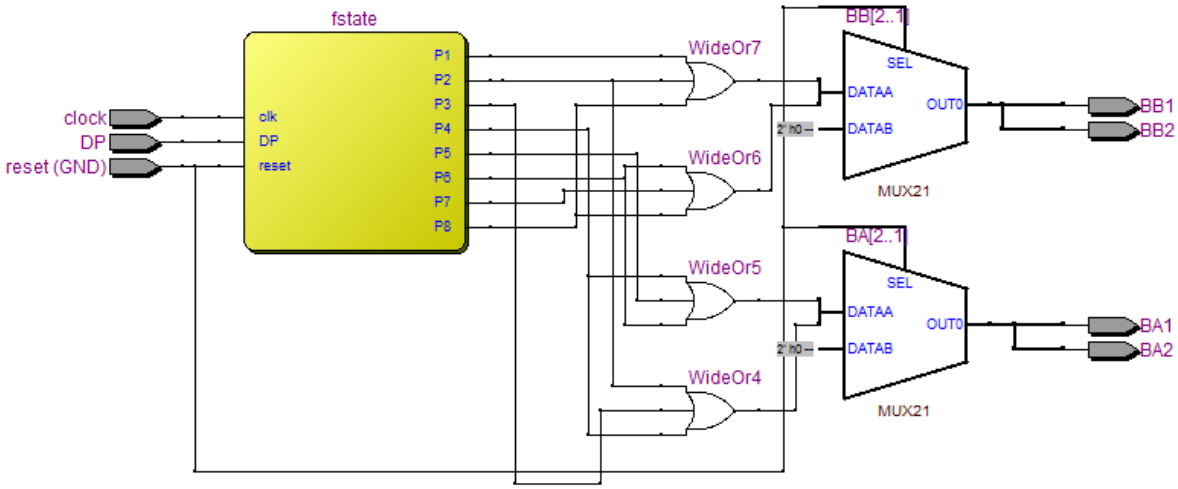
```
                BB2 <= '0';
            WHEN P8 =>
                reg_fstate <= P1;
                BA1 <= '0';
                BA2 <= '0';
                BB1 <= '1';
                BB2 <= '1';
            WHEN OTHERS =>
                BA1 <= 'X';
                BA2 <= 'X';
                BB1 <= 'X';
                BB2 <= 'X';
                report "Reach undefined state";
        END CASE;
    END IF;
END PROCESS;
END BEHAVIOR;
```

3) *Structure logique de la machine à état.*

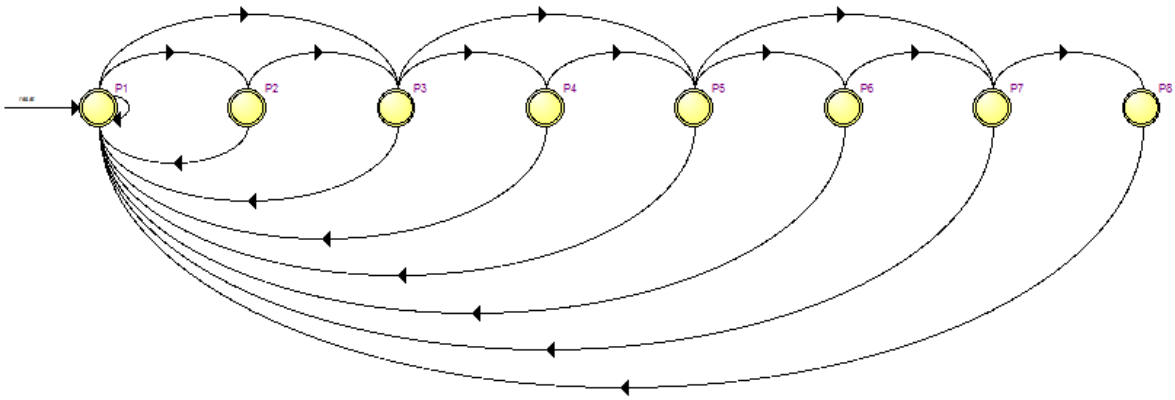
Tools – Netlist Viewvers –RTL Viewer affiche le schéma de la structure fonctionnelle (comportementale ou behavior)



Un double-clic sur l'instance permet de voir sa structure.

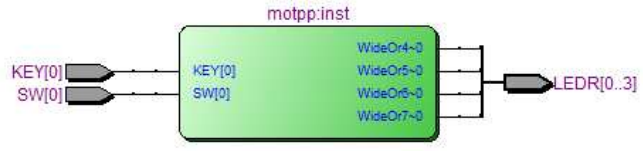


Un double clic sur la machine à état (fstate) permet de retrouver la description initiale.

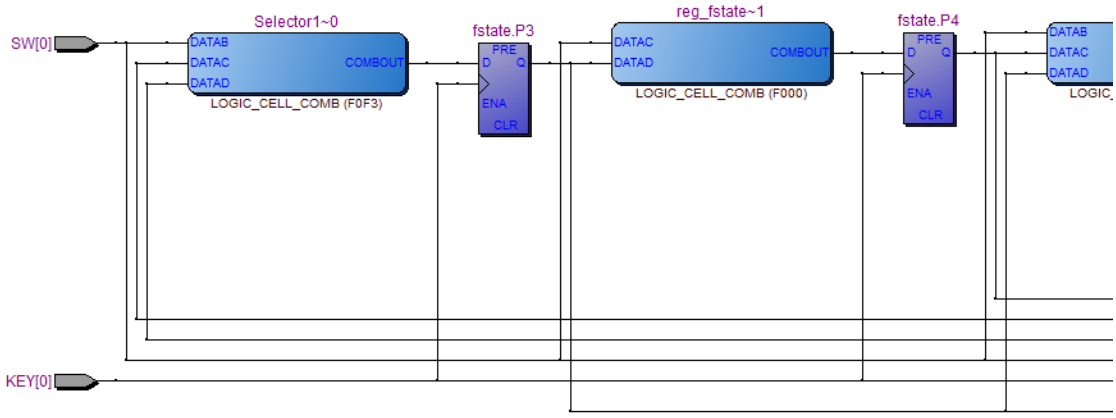


4) *Structure synthétisée dans le FPGA de la machine à état.*

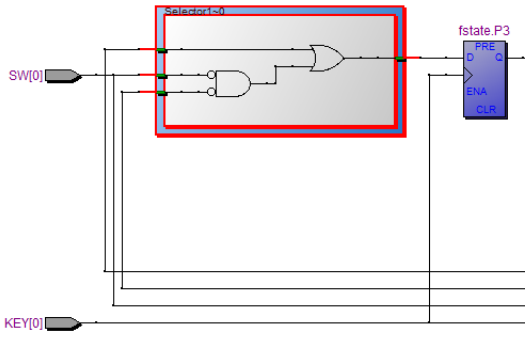
Tools – Netlist Viewvers –Technologie Map Viewer affiche le schéma de la structure synthétisée.



Double-clic sur l'instance affiche les LOGIC_CELL et les bascules mises en œuvre.



La structure interne des LOGIC_CELL peut être visualisée (double-clic)



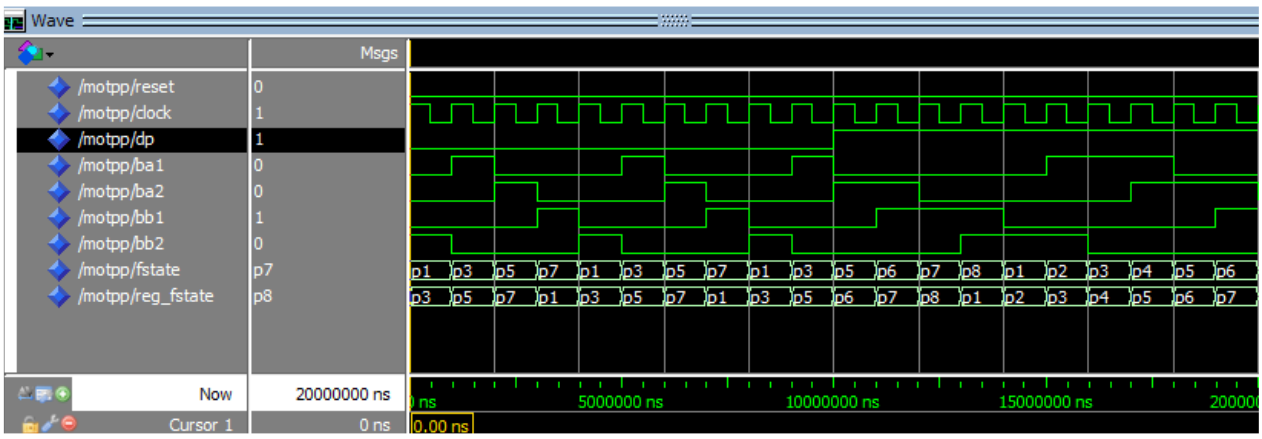
5) *Simulation fonctionnelle de la machine à états*

Le fichier VHDL généré est simulable dans ModelSIm. Effectuer une simulation fonctionnelle (RTL)

Cliquer Tools – Run EDA simulation tools – EDA RTL simulation

Dans ModelSim, Cliquer Simulate – Start Simulation - work- behavior – OK

Ajouter tous les signaux, forcer le RESET et DP à 0, placer une horloge 1ms sur clock et effectuer une simulation de 10ms, forcer DP à1 , effectuer une simulation de 10ms.



Le simulateur présente les signaux de commande du moteur pas à pas ainsi que l'état de la machine et l'état suivant, conformément au code VHDL.

6) *Simulation temporelle de la machine à état*

La machine à états doit être insérée dans un schéma qui permettra entre autre la définition des broches du FPGA cible.

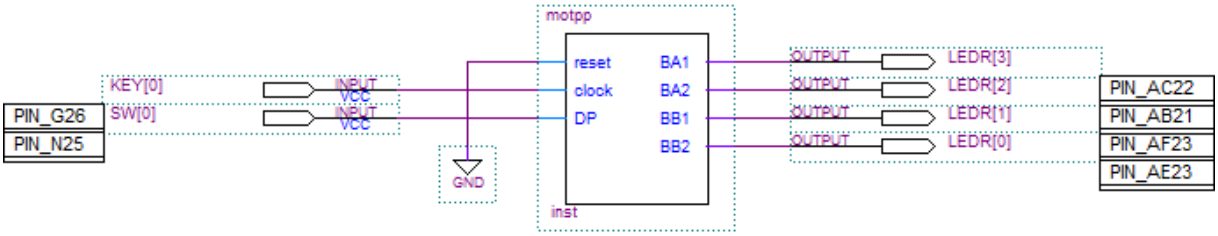
Créer un nouveau composant à partir du fichier motpp.vhd :

Cliquer File – Create/Update – Create Symbol File for Current File

Créer un nouveau schéma « motpp_MAE.bdf » mettant en œuvre ce composant.

Charger les assignations par défaut : Assignment – Import assignments selectionner DE2_pin_assignments.csv. (Voir CD accompagnant la carte DE2) http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=30&PartNo=4

Nommer ensuite les broches d'E/S comme suit :



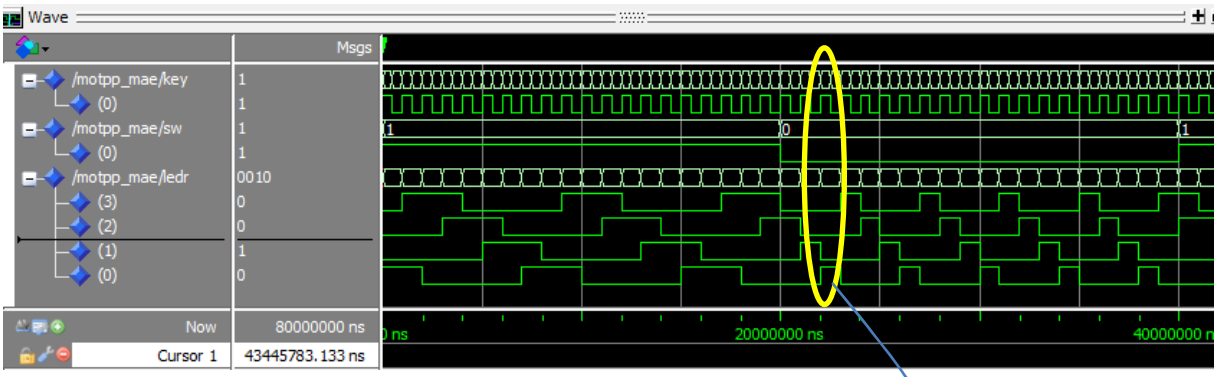
On peut vérifier par un double-clic sur le composant quelle est sa description.

Définir ce schéma comme description de plus haut niveau

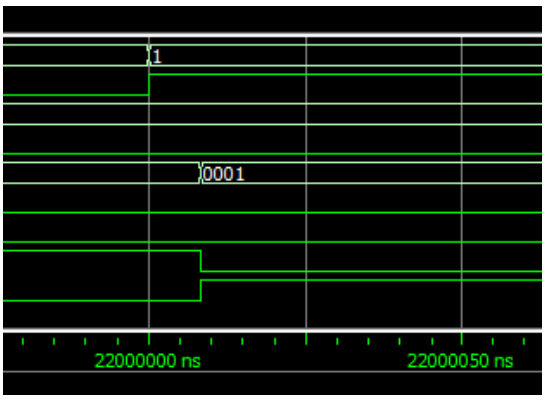
Dans le Project Navigator clic-droit sur motpp_MAE.bdf puis « Set As Top Level Entity »

Lancer maintenant ModelSIm en simulation « GATE LEVEL » « Slow model »

Placer une horloge 1ms sur key[0], une horloge 40ms sur sw[0], effectuer une simulation de 80ms.



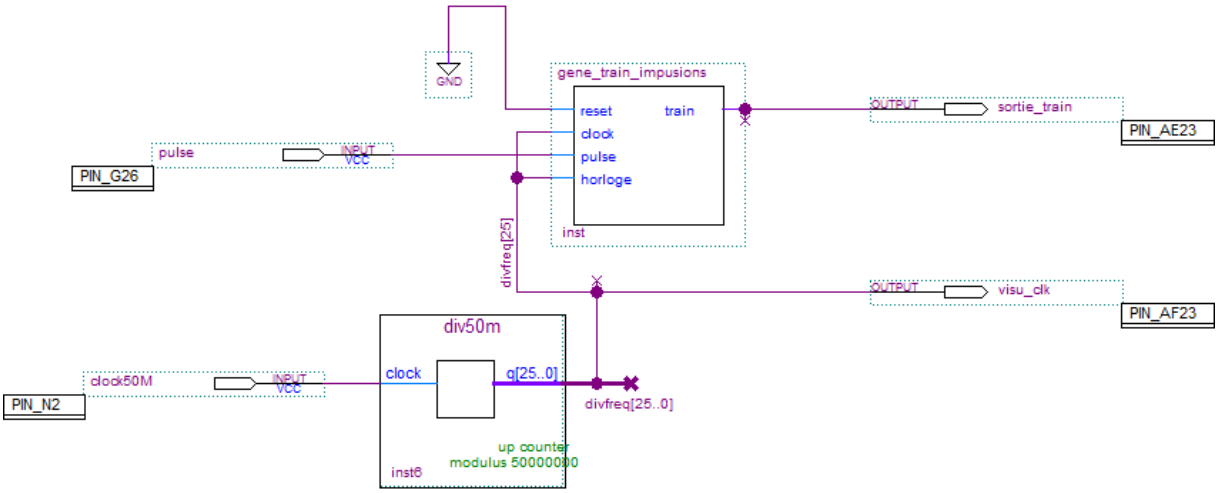
Les signaux représentant les états de la machine ne sont plus accessibles, Quartus ayant optimisé la description en temps et en ressource. En revanche les retards et temps de propagation peuvent être mesurés, la simulation représentant un fonctionnement réel.



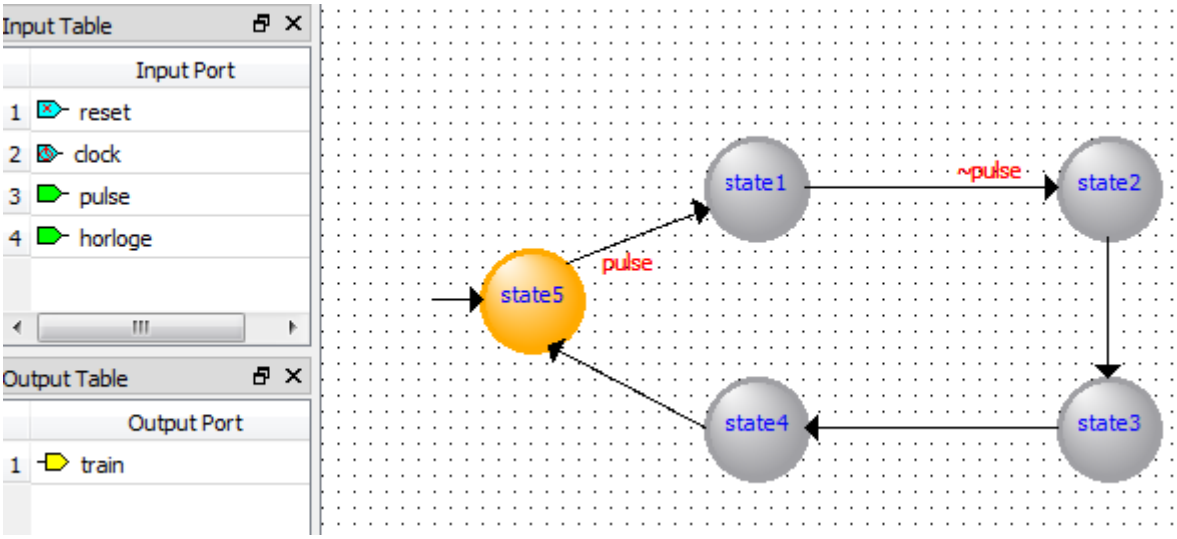
Il ne reste plus qu'à effectuer les essais sur le matériel.

7) *Exercice : Générateur de train d'impulsions*

On se propose de réaliser un générateur de train de trois impulsions sur la LED LEDR[0] lors de l'appui sur la touche KEY[0] du KIT DE2, suivant le schéma ci-dessous :



Réaliser, simuler et tester sur une carte DE2 le générateur de train d'impulsions avec une période d'une seconde et correspondant à la description suivante :



Lors du front descendant sur « pulse », « l'horloge » est transmise trois fois sur la sortie train.
(L'onglet action de l'état comporte alors l'information train=horloge)

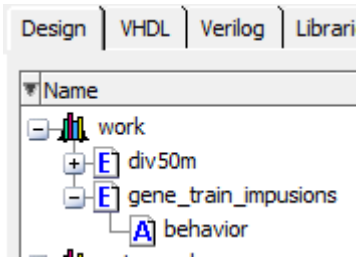
En raison de la division par 50M de l'horloge 50Mhz, la simulation sera effectuée sans le diviseur par 50.000.000.

La division de l'horloge du KIT DE2 par $50x10^6$ entraine une simulation « GATE » très longue.

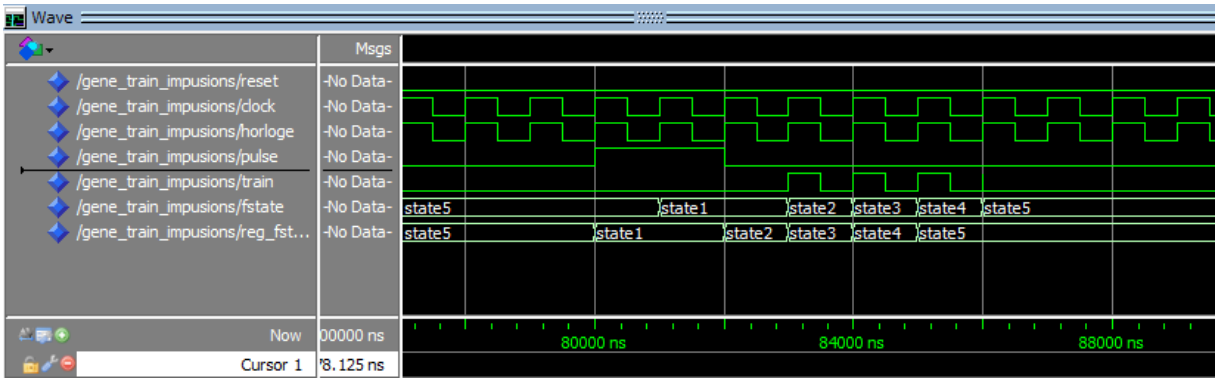
La simulation comportementale permet de simuler chaque description indépendamment, elle va permettre de valider le fonctionnement comportemental de la machine à états.

Lancer une simulation EDA-RTL avec ModelSim.

Dans ModelSIm , Start Simulation- work – choisissez « behavior » de la MAE « gene_train_impulsions »



Avec une horloge 1us sur « clock » et « horloge » et une horloge 20ms, rapport cyclique 20% sur pulse on doit obtenir :

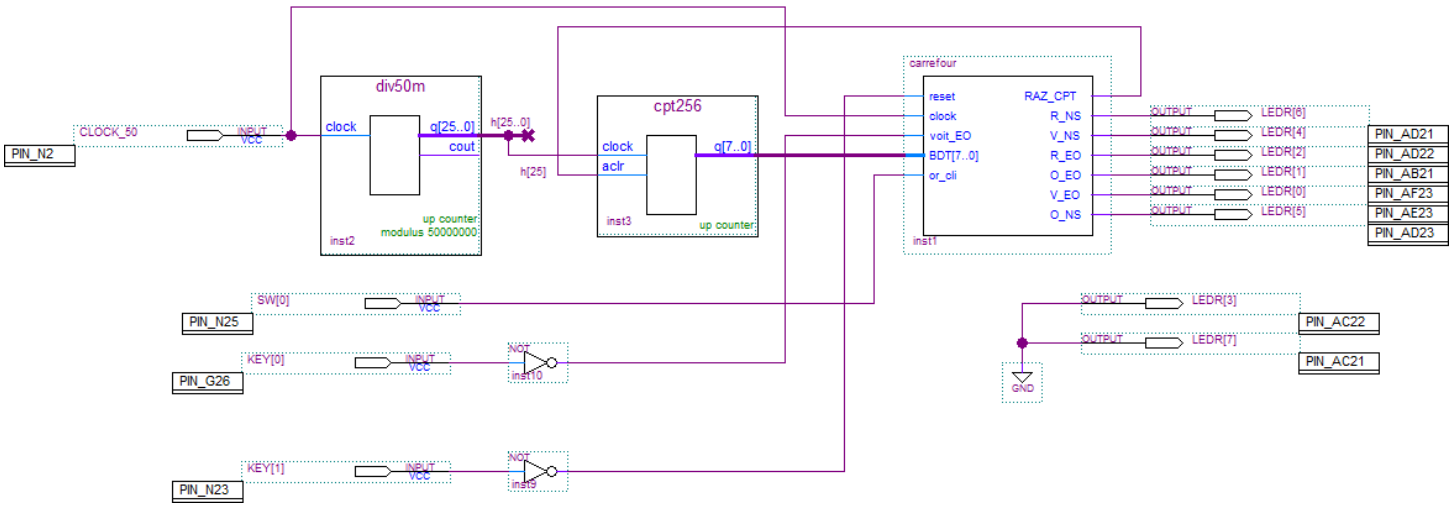


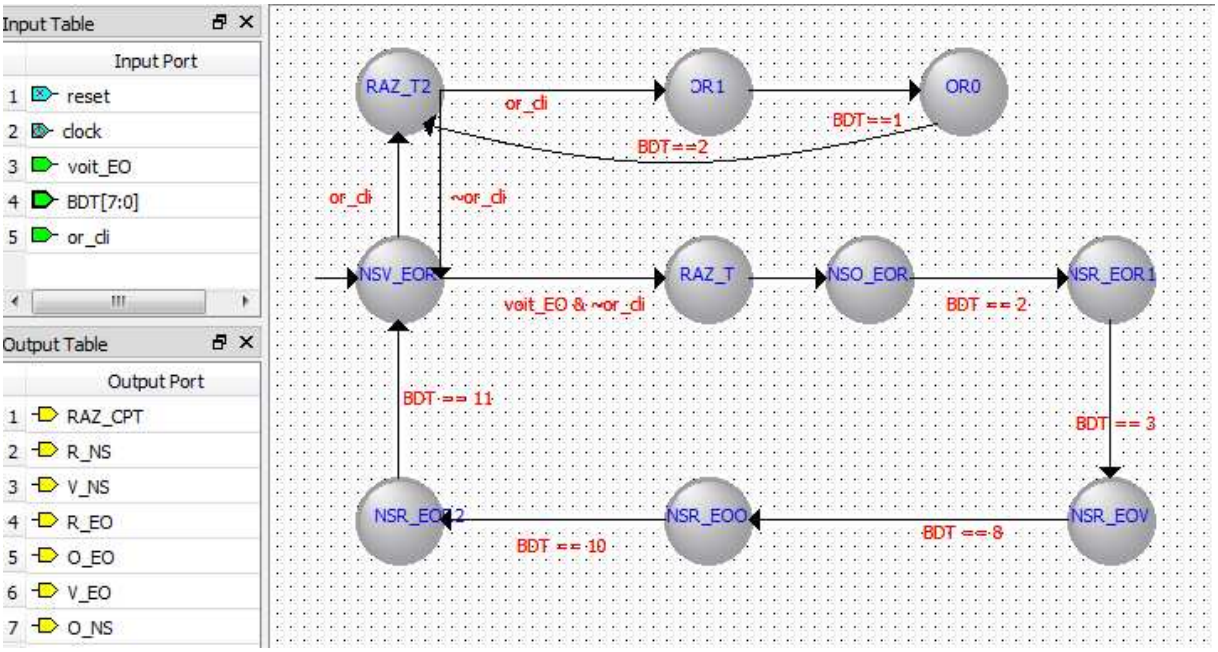
Essayer ensuite le générateur de train d'impulsion sur le KIT DE2.

8) *Exercice : gestionnaire de carrefour*

Le carrefour possède deux axes de circulation.
La voie Nord-Sud (NS) est prioritaire et au vert par défaut.
Un détecteur de véhicules est placé sur la voie Est-Ouest (EO). Lorsqu'un véhicule se présente, la voie NS passe à l'orange pour 2 secondes puis au rouge, après 1 seconde la voir EO passe au vert pour 5 secondes (cela pour accélérer la simulation), puis à l'orange pour 2 secondes, puis au rouge, après 1 secondes la voir NS passe au vert et le cycle peut recommencer.
Une commande (or_cli) permet de passer les deux voies à l'orange clignotant, période 2 secondes.

Une base de temps est ici nécessaire, elle est réalisée à l'aide d'un compteur de secondes qui peut être remis à zéro par la machine à états et dont la sortie peut être utilisée dans les tests de transitions.

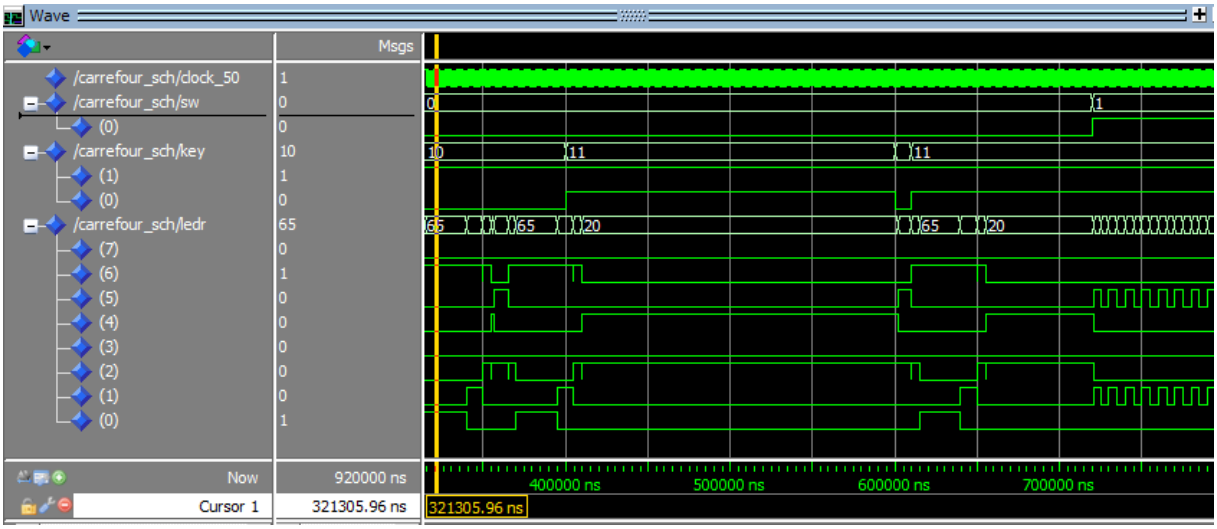




Réaliser la machine à état et le schéma du gestionnaire de carrefour.

Le différentiel entre le temps du FPGA (20ns) et le temps des feux du carrefour (la seconde) entraine une simulation très longue. Pour être pragmatique (uniquement pour la simulation) le diviseur par $50x10^6$ est transformé en diviseur par 50 et l'horloge passe d'une période de 20ns à une période de 20ms.

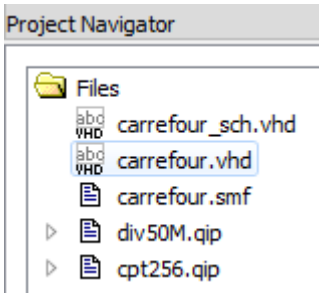
Simulation « GATE » : Cette simulation montre l'évolution des signaux après synthèse dans le FPGA, les « glitchs » ont la largeur d'une période d'horloge.



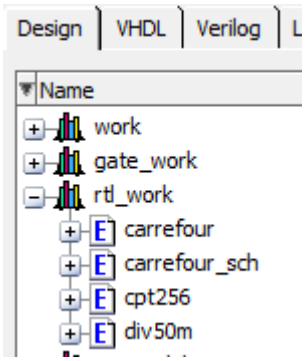
La simulation fonctionnelle est réalisée à partir de fichiers en langage de haut niveau (VHDL, VERILOG). La machine à état ainsi que les compteurs sont décrits en VHDL.

Il est nécessaire de remplacer le schéma du gestionnaire de carrefour par sa description en VHDL.

Afficher le schéma puis « file – Create/update – Create HDL design file for current file », choisir VHDL. Le fichier carrefour_sch.vhd est créé. Il faut remplacer le schéma par la description VHDL dans le Project Navigator.



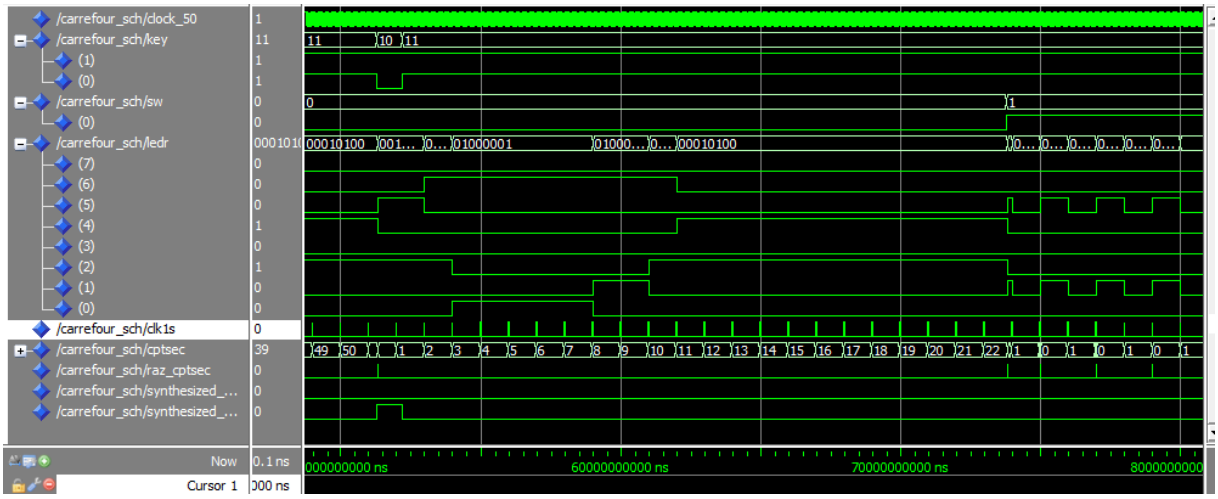
On peut maintenant réaliser une simulation fonctionnelle.
Dans ModelSim faire « Simulate - start simulation », dans rtl_work, il est possible de simuler la MAE « carrefour », ainsi que les deux compteurs indépendamment. Pour simuler l'ensemble, sélectionner « carrefour_sch ».



La commande "do test.do" lance la simulation

Fichier test.do :

```
vsim -do carrefour_run_msim_rtl_vhdl.do -l msim_transcript -gui rtl_work.carrefour_sch
add wave /*
force -freeze sim:/carrefour_sch/clock_50 1 0, 0 {10000000000 ps} -r 20ms
force -freeze sim:/carrefour_sch/key(1) 1 0
force -freeze sim:/carrefour_sch/key(0) 1 0
force -freeze sim:/carrefour_sch/sw(0) 0 0
run 25000ms
force -freeze sim:/carrefour_sch/key(0) 0 0
run 1000ms
force -freeze sim:/carrefour_sch/key(0) 1 0
run 15000ms
force -freeze sim:/carrefour_sch/sw(0) 1 0
run 15000ms
```

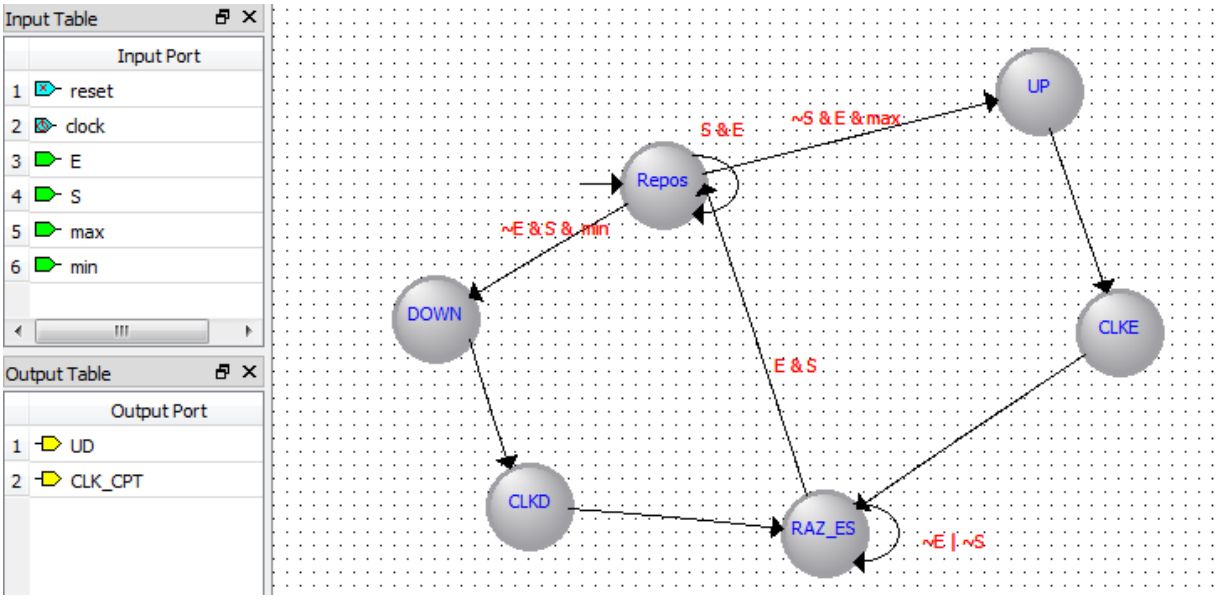


Ne pas oublier de changer le diviseur par 50 en diviseur par $50x10^6$ avant de faire les essais sur le matériel !

9) *La machine à état dans le projet : gestion de parking*

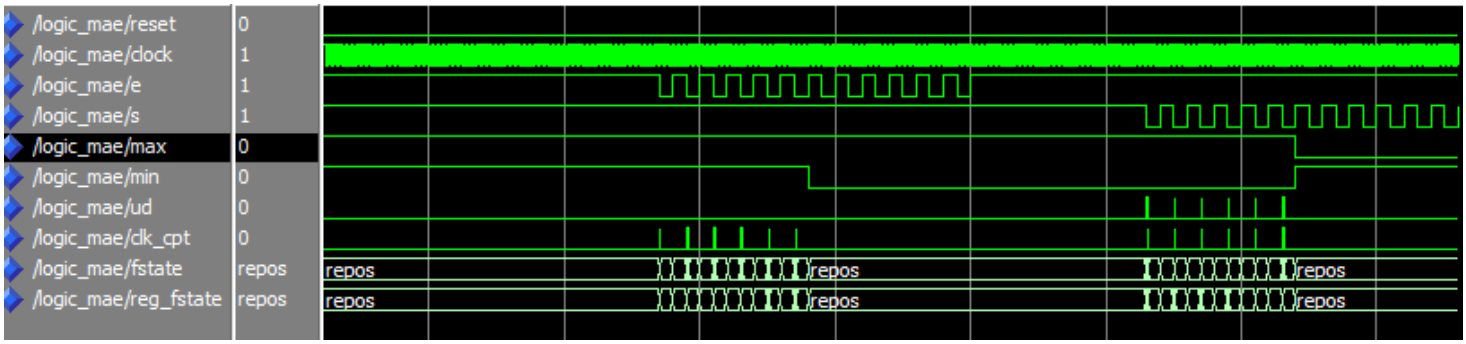
- La détection d'un véhicule en entrée s'il reste au moins une place dans le parking une impulsion en mode décomptage sur le compteur/décompteur
- La détection d'un véhicule en sortie si le nombre de place restant n'est pas au maximum provoque une impulsion en mode comptage sur le compteur/décompteur

La machine à état du détecteur de véhicules :



E et S, détecteurs de véhicules, actifs à l'état bas.
max =0 indique un parking plein.
min=0 indique un parking vide.
UD, comptage/décomptage, comptage si UD=1
clk_cpt, horloge du compteur/décompteur

	Output Port	Output Value	In State
1	UD	0	DOWN
2	UD	0	RAZ_ES
3	UD	1	UP
4	UD	0	Repos
5	UD	1	CLKE
6	UD	0	CLKD
7	CLK_CPT	0	DOWN
8	CLK_CPT	0	RAZ_ES
9	CLK_CPT	0	UP
10	CLK_CPT	0	Repos
11	CLK_CPT	1	CLKE
12	CLK_CPT	1	CLKD



Réaliser la machine à état dans le projet « parking » et effectuer une simulation fonctionnelle.