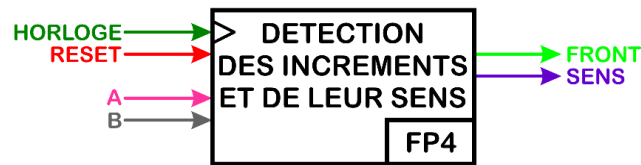




I. PRÉSENTATION DE LA FONCTION FP4

Cette fonction est chargée de détecter chaque front des signaux A et B issus du codeur incrémental et par la même d'en déterminer le sens de rotation.

I.1. DESCRIPTION DU COMPOSANT CODEUR INCRÉMENTAL



Une entrée de RESET synchrone permettra de placer les sorties dans un état lors de la mise sous tension déterminé (initialisation de la logique séquentielle).

Suivant l'évolution des signaux A et B qui sont déphasés de 90°, le signal VALIDATION passe à 1 pendant 1 coup d'horloge à chaque changement de A ou B.

Si le signal A est en avance sur le signal B alors SENS passe à 1, sinon il prend la valeur 0.

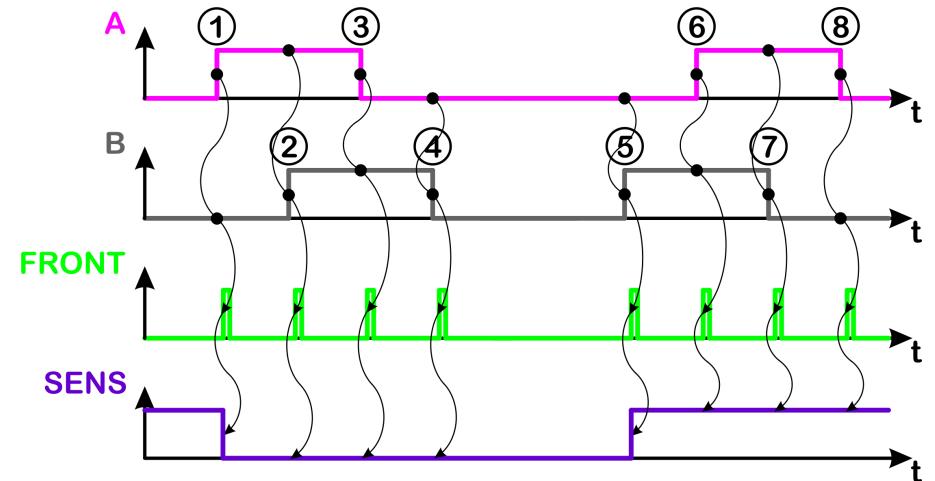
La synthèse peut être faite de plusieurs manières différentes : soit l'écriture directe d'un fichier VHDL soit le dessin d'une machine à états (FSM ou Finite State Machine) ou encore le dessin d'une structure électronique (schéma). Nous en verrons les deux premières solutions.

I.2. CHRONOGRAMMES ATTENDUS

A l'analyse des signaux A et B on remarque qu'il y a 8 cas différents : à chaque front d'un signal, l'autre peut être à 0 ou à 1.

La sortie FRONT passera à 1 fugitivement (durée un coup d'horloge) à chaque fois qu'un front sur A ou B se présentera.

La sortie SENS Passera à 0 pour les cas ① à ④ et à 1 pour les cas ⑤ à ⑧.



II. ÉDITION D'UN FICHIER VHDL

II.1. AJOUT AU PROJET D'UN NOUVEAU FICHIER VHDL

Ajouter un nouveau fichier VHDL au projet. Renommer-le `Codeur_Incremental.Vhd`.

II.2. ÉDITION ET COMPILATION DU FICHIER VHDL

Éditer le fichier VHDL, enregistrer-le puis compiler-le, Corriger les erreurs qui pourraient apparaître.

(Voir listing en annexe pages 9 et 10).

Remarque : le principe est d'utiliser des registres à décalages (basculés D) afin de détecter si un front vient d'avoir lieu sur l'entrée A ou B. Dans ce cas la valeur actuelle est différente de la valeur précédente. Il ne reste plus qu'à lister les différents cas.

III. SIMULATION FONCTIONNELLE AVEC ALDEC OEM

III.1. PRODUCTION DU FICHIER DE TEST

Créer le fichier VHDL de test.

Ajouter à celui-ci les lignes des process de tests afin de faire évoluer les ports d'entrées suivant un jeu d'essai permettant de vérifier tous les cas. (Voir listing en annexe page 11).

Enregistrer celui-ci en gardant le nom proposé.

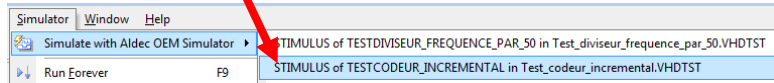
Compiler ce fichier puis compiler le projet complet.

III.2. ASSOCIATION DES FICHIERS

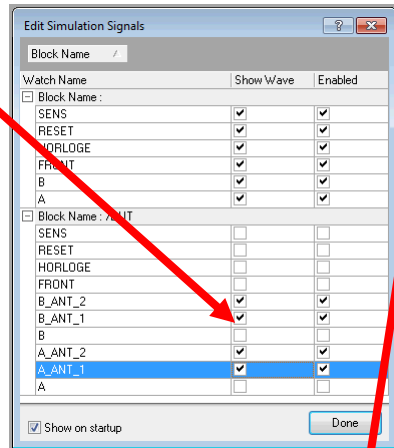
Ajouter le fichier de test dans le Manager Testbenches du simulateur.

III.3. SIMULATION

Lancer la simulation :

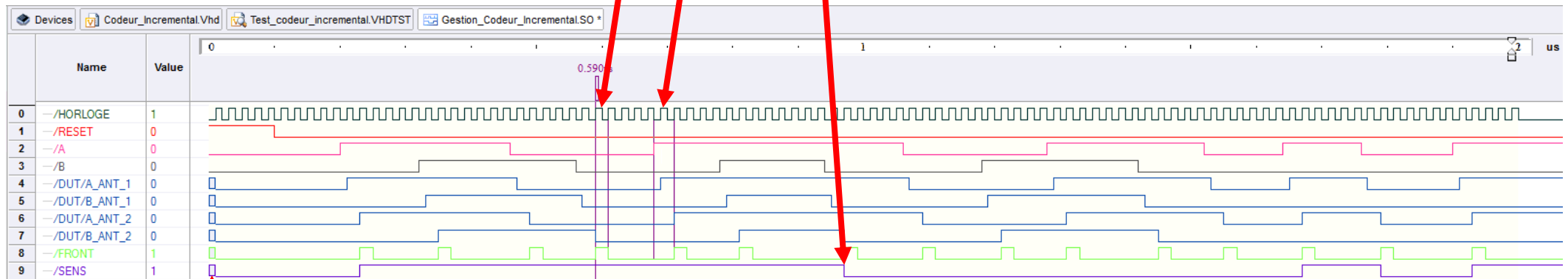


Cocher les signaux à visualiser (show Wave) en plus des ports d'entrées/sorties : (A_ANT_1, B_ANT_1, A_ANT_2, B_ANT_2).



Lancer la simulation sur une durée de 2 μ s.

Afficher la simulation complète.
Replacer les différents signaux dans un ordre logique.



On peut remarquer que les signaux internes (DUT) et les sorties ne sont pas définies avant le premier front montant de l'horloge HORLOGE qui les place dans l'état défini dans le fichier VHDL puisque l'entrée RESET est synchrone.

Analyser les chronogrammes obtenus.

On peut remarquer que le signal FRONT est activé pendant un coup d'horloge à chaque fois qu'un front sur A ou B survient.

Le signal FRONT passe au niveau haut au maximum 2 coups d'horloge après le changement d'un des deux signaux d'entrée A ou B.

Le signal SENS respecte bien l'évolution prévue.

Sauvegarder la fenêtre de simulation.
Sauvegarder le projet complet.

La fonction FP4 est donc validée.

Il peut parfois être utile de proposer d'autres excitations afin d'observer le comportement du programme VHDL, en particulier des cas (ou jeux d'essais) qui normalement ne se produisent pas en fonctionnement normal mais peuvent apparaître dans la réalité (rebonds, etc).

Effectuer une nouvelle simulation en modifiant le fichier de TestBench comme indiqué pour voir l'incidence de rebonds lors du changement d'état de l'entrée A (Voir listing partiel en annexe page 12).



On peut remarquer que les rebonds haute fréquence sont pris en compte par le composant. Le compteur qui suit comptera 7 coups au lieu d'un seul. Il faut donc prévoir un composant qui supprimera les rebonds en amont (FP1 et FP2).

IV. ÉDITION D'UNE MACHINE À ÉTATS

IV.1. PRÉSENTATION RAPIDE

Une machine à états (ou FSM en anglais) est la représentation graphique d'une structure électronique très utilisée en logique séquentielle câblée (séquenceurs utilisant souvent des bascules JK).

Elle fait appel à des états (ou étapes) et des transitions.

Altium ne dispose pas d'éditeur de machines à états. Nous allons donc utiliser celui proposé par ALDEC (le second grand de la synthèse logique).

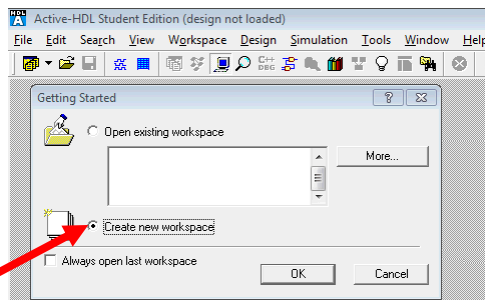
Il faut pour cela installer **ALDEC Active-HDL Student Edition**.

Cette version limitée est gratuite pour les étudiants et les organismes de formation.

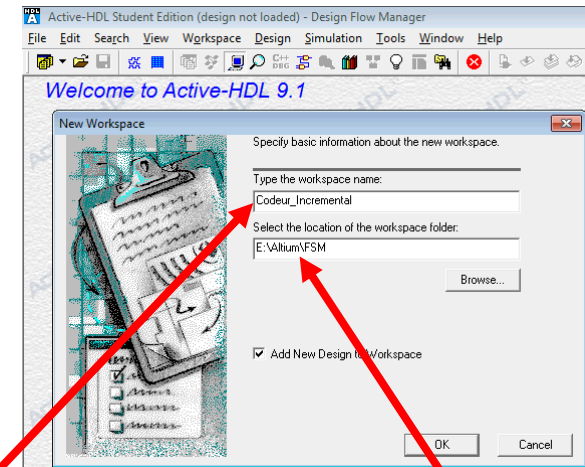
Dans cette partie nous considérerons que le logiciel est correctement installé.

IV.2. CRÉATION D'UN ESPACE DE TRAVAIL

Lancer le logiciel **ALDEC Active-HDL Student Edition** en double-cliquant sur son icône.

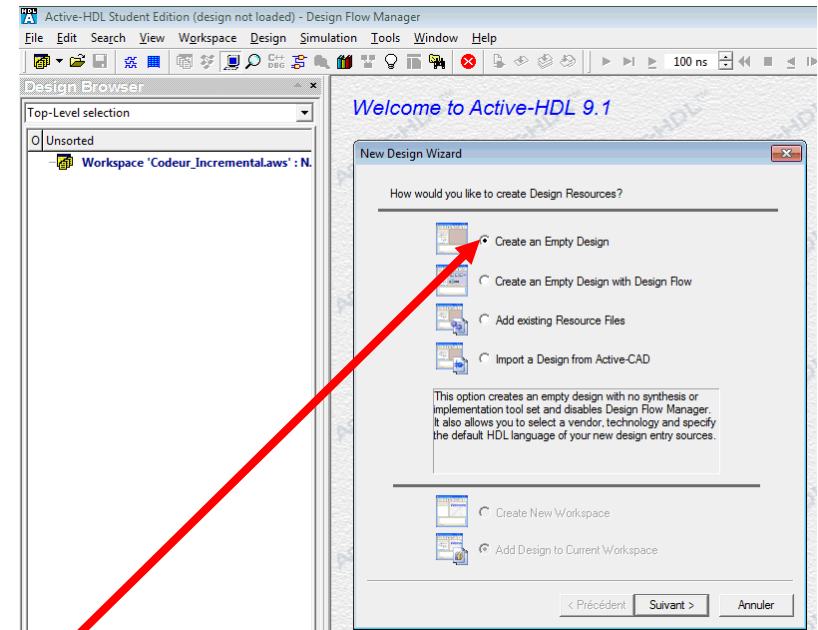


Sélectionner **Create new workspace** puis OK.

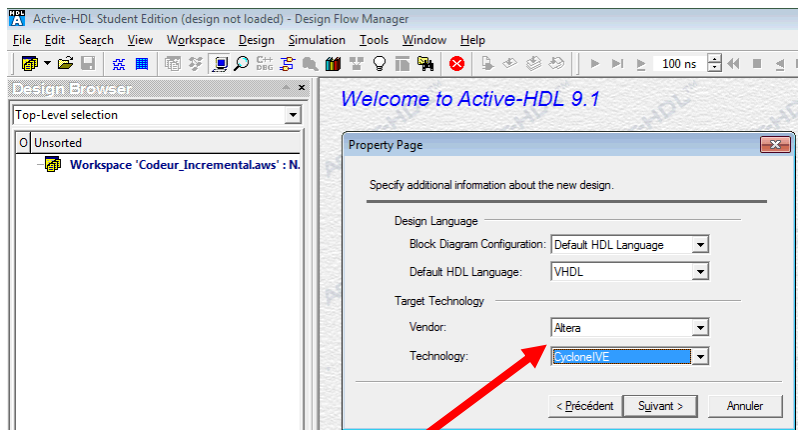


Donner un nom à votre espace de travail (projet).

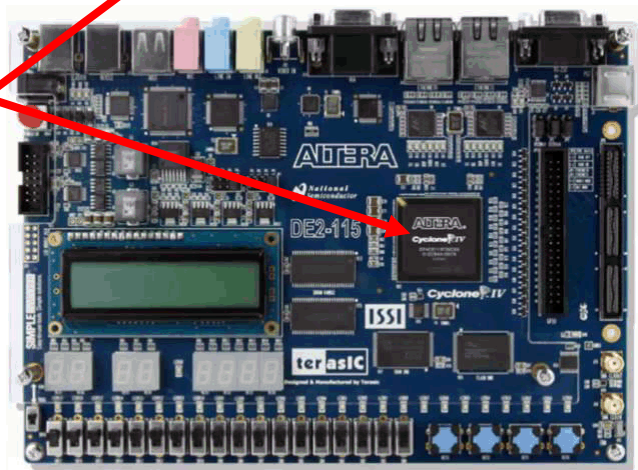
Préciser le chemin à partir duquel un répertoire de travail sera créé (différent de celui du projet Altium). Puis OK.



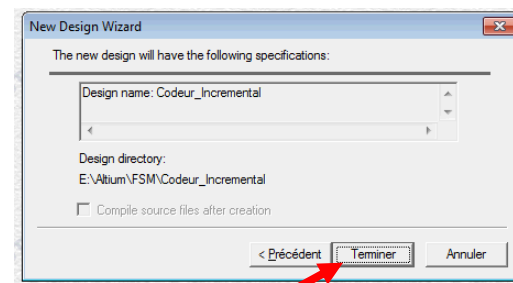
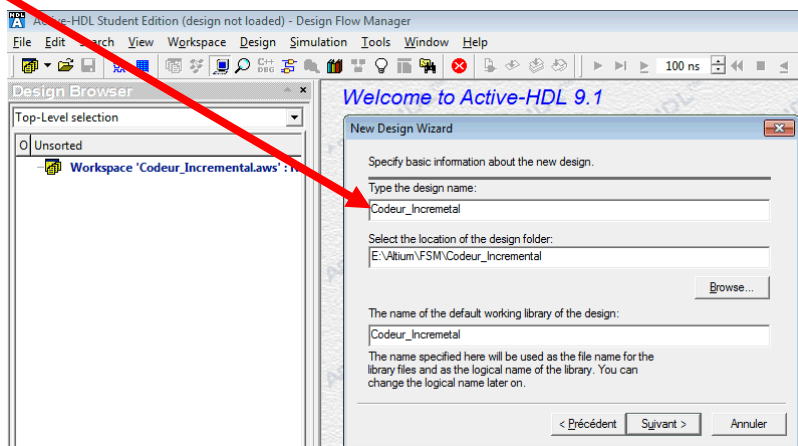
Sélectionner **Create an Empty Design** puis Suivant.



Renseigner, si besoin, ces deux menus déroulants en fonction de la cible utilisée. Ici ce sera un Cyclone IV d'Altera car la carte utilisée est une DE2-115 de TERCASIC. Puis faire suivant.



Nommer le Design Codeur_Incremental puis faire Suivant.



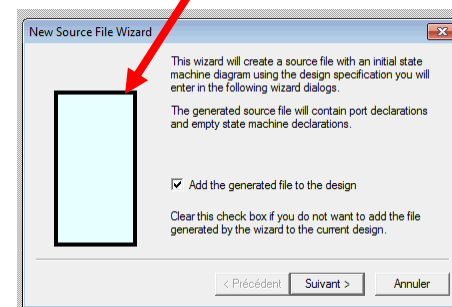
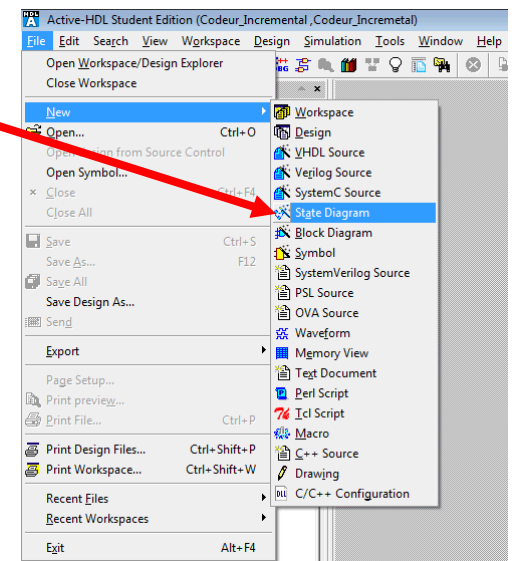
Sur la dernière fenêtre cliquer sur Terminer.

IV.3. CRÉATION DE LA MACHINE À ÉTATS

Créer une nouvelle machine à états :

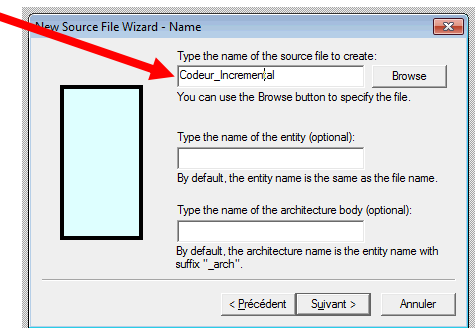
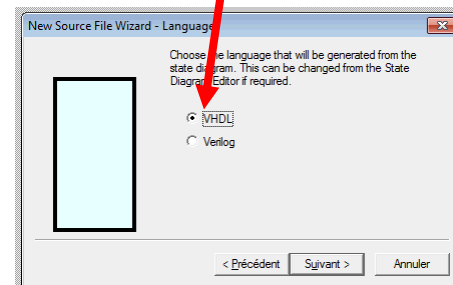
⇒ File ⇒ State Diagram

Un wizard (assistant) va permettre de construire simplement l'entité. Faire Suivant.

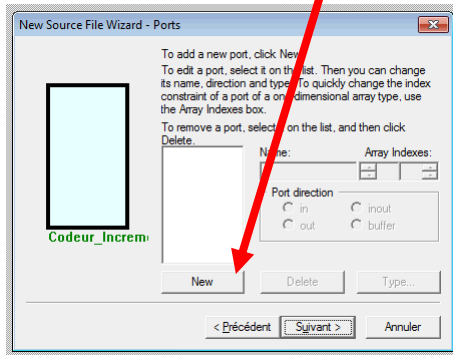


Langage utilisé : VHDL.

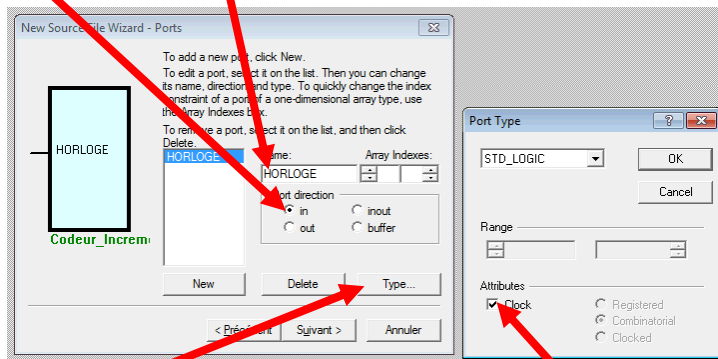
Nom de la machine à états : Codeur_Incremental.



Cette fenêtre permet d'ajouter des ports à l'entité. Faire New.



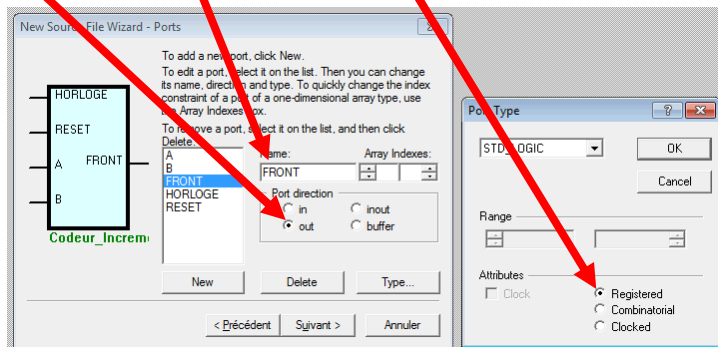
Ajouter en entrée le port HORLOGE.



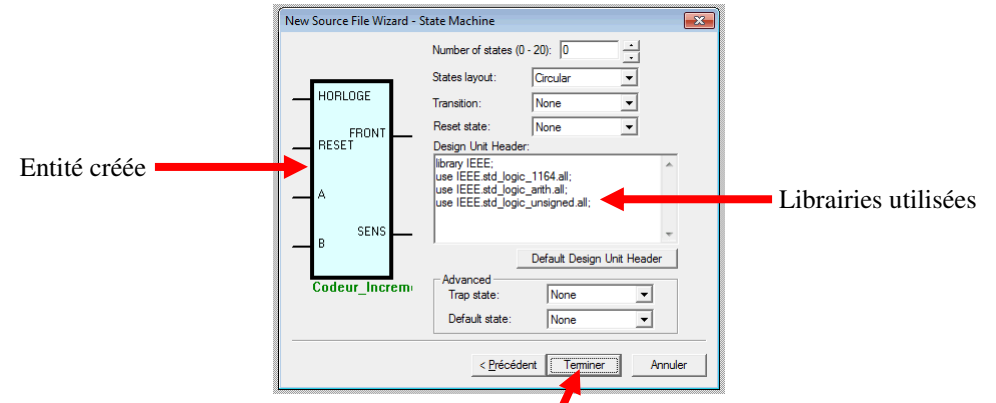
Puis cliquer sur Type pour faire apparaître la fenêtre permettant d'indiquer que c'est un signal d'horloge. Faire OK puis New.

Ajouter les 3 autres entrées : RESET, A et B (par défaut les entrées ne sont pas des signaux d'horloges).

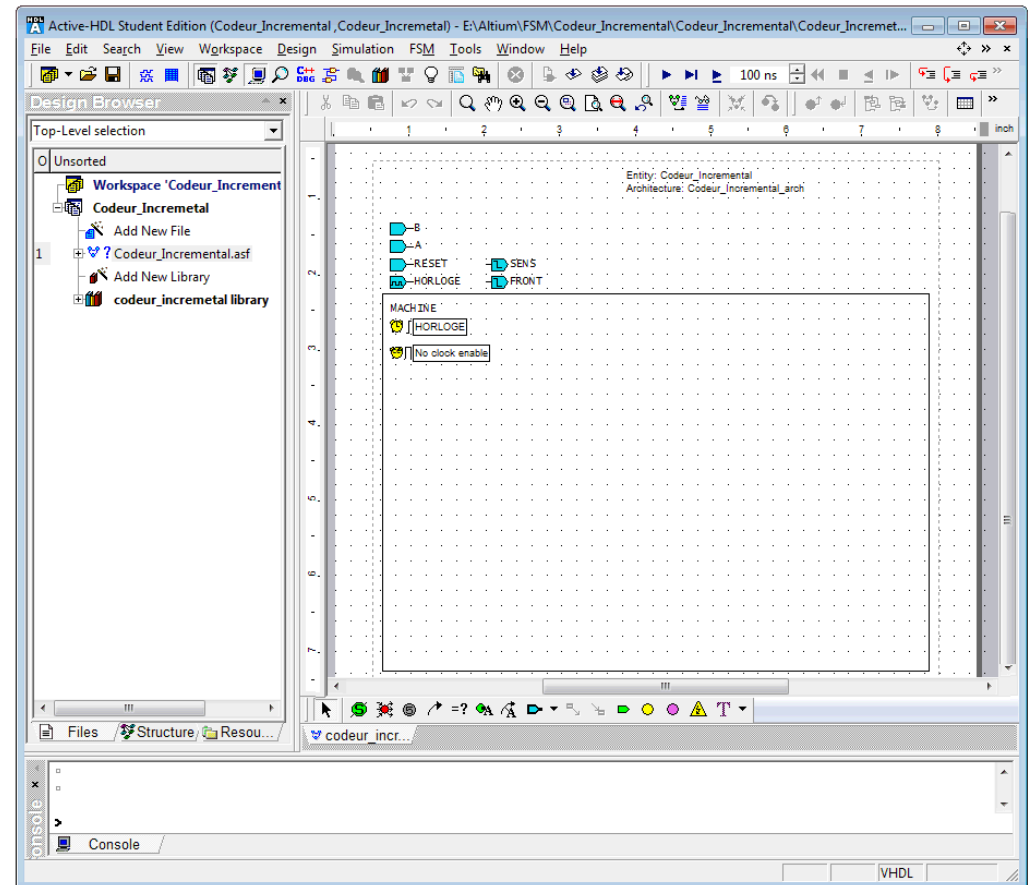
Ajouter en sortie le port FRONT de type Registered (machine de Moore). Faire OK puis New.



Ajouter l'autre sortie : SENS du même type puis faire Suivant.

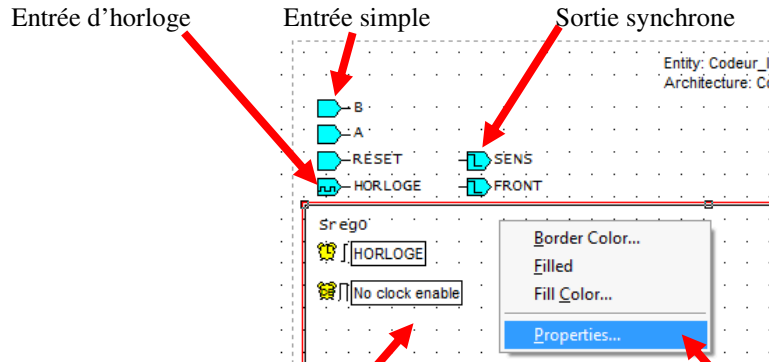
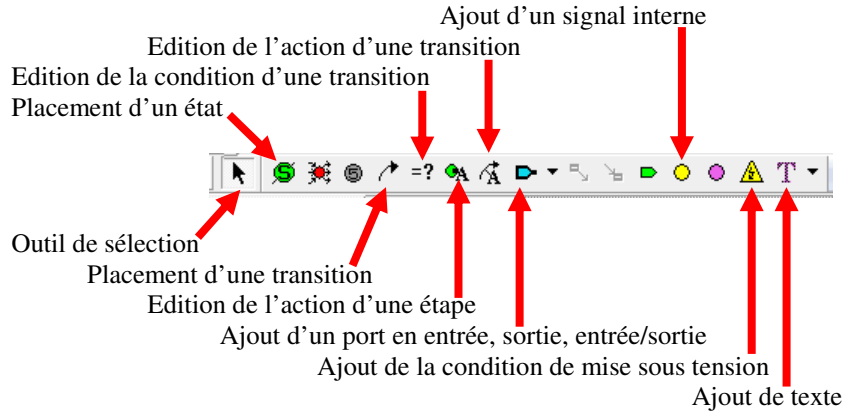


Il est possible de définir d'autres paramètres ici mais ce sera fait manuellement plus tard. Faire terminer.

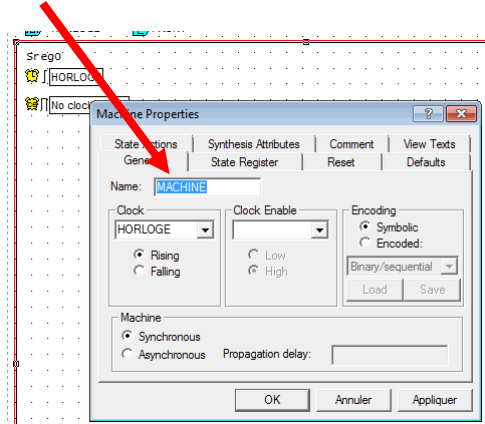


IV.4. DESSIN DE LA MACHINE À ÉTATS

Barre d'outils :

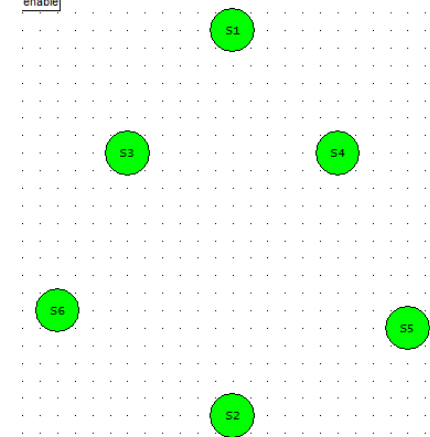
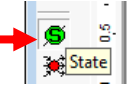


Cliquer droit dans la fenêtre de la machine à états puis cliquer sur Properties... et renseigner les propriétés comme ci-dessous :

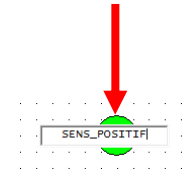
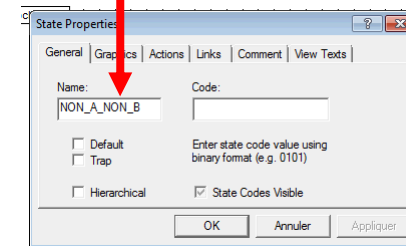


La machine à états complète est donnée en annexe page 12 pour aider au dessin.

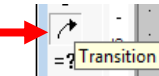
Disposer 6 états comme ci-dessous avec cet outil :



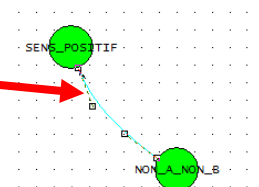
Renommer les états soit en double cliquant dessus soit en cliquant sur le nom comme donné ci-dessous.



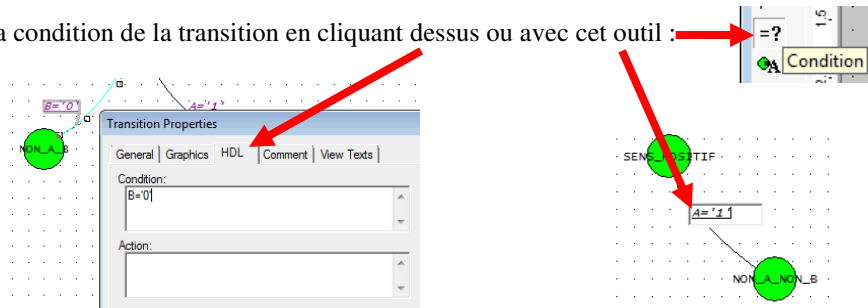
Ajouter les transitions entre les états avec cet outil :



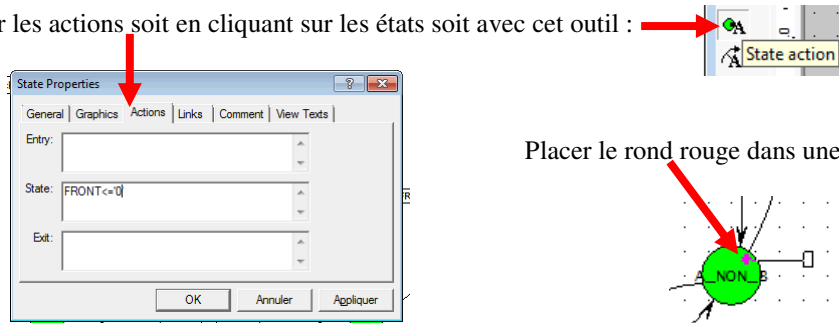
Cliquer sur le bord de l'état de départ puis sur le bord de l'état d'arrivée. Une flèche apparaît. En cliquant dessus on peut faire apparaître deux points qui permettent de courber la transition. Il est aussi possible de déplacer les points de départ et d'arrivée en cliquant dessus.



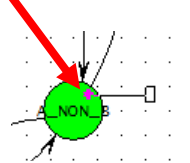
Éditer la condition de la transition en cliquant dessus ou avec cet outil :



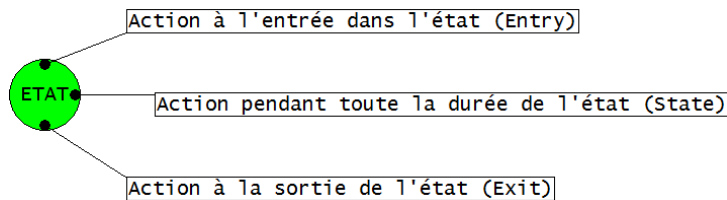
Ajouter les actions soit en cliquant sur les états soit avec cet outil :



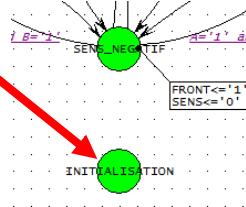
Placer le rond rouge dans une étape



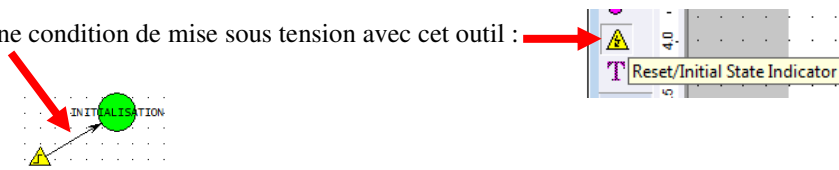
Suivant le besoin 3 types d'actions peuvent être choisis :



Ajouter un dernier état :



Ajouter une condition de mise sous tension avec cet outil :

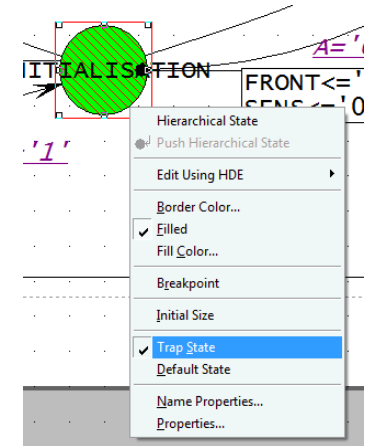


Puis éditer la condition correspondant à cette transition.

Il ne reste plus qu'à définir un des états comme l'état piège, c'est-à-dire que s'il arrive une erreur au niveau de la machine à états c'est dans cet état que la machine retournera.

Dans notre cas il y a 7 états, il faut donc 3 bits au minimum pour les coder. Il reste donc un état inutilisé, si par malheur la machine entrait dedans elle pourrait y rester indéfiniment. Le fait de définir un piège permet d'en sortir le coup d'horloge suivant.

Cliquer droit sur l'état et cocher Trap State.

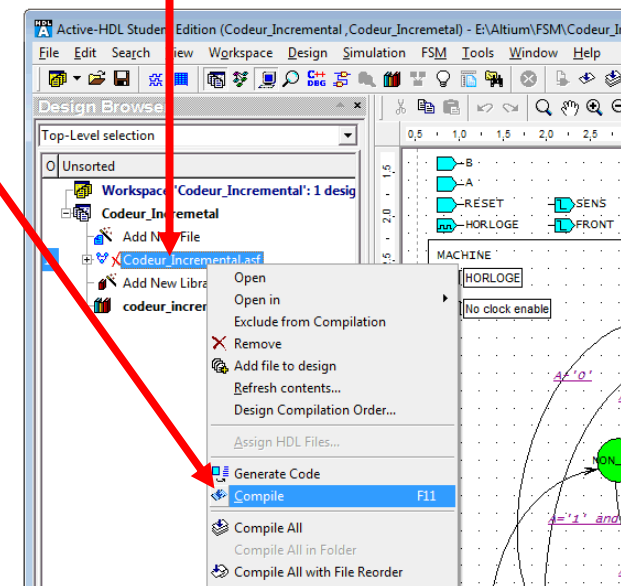


La machine à états est maintenant complète. Enregistrer celle-ci.

A remarquer qu'il n'y a pas besoin d'une grande connaissance de la syntaxe VHDL à par l'écriture et la syntaxe des conditions et des actions.

IV.5. TRADUCTION DU GRAPHIQUE EN VHDL

Dans la fenêtre des fichiers cliquer droit sur le nom de la machine à états et cliquer sur Compiler.



Dans la fenêtre Console on peut connaître le résultat de la compilation :

```
# Warning: FSM2HDL_3127 Codeur_Incremental.asf : State INITIALISATION was selected as the trap state. The coverage pragmas have been omitted.
# HDL code generation completed. 1 error(s), 2 warning(s)
# Double-click on error/warning line to see the source of error/warning.
```

S'il y a des erreurs, les corriger jusqu'à obtenir :

```
# File: .\Codeur_Incremental.vhd
# Compile Entity "Codeur_Incremental"
# Compile Architecture "Codeur_Incremental_arch" of Entity "Codeur_Incremental"
# Compile success 0 Errors 0 Warnings Analysis time : 0.1 [s]
```

Générer le fichier VHDL à partir de la machine à états avec cet outil :

Afficher le fichier VHDL avec cet outil :



Effectuer un copier de tout le fichier généré (CTRL + A, CTRL + C) et coller le fichier dans Codeur_Incremental.vhd du projet Altium à la place du listing édité précédemment.

Compiler celui-ci, il ne devrait pas y avoir d'erreurs.

V. SIMULATION FONCTIONNELLE AVEC ALDEC OEM

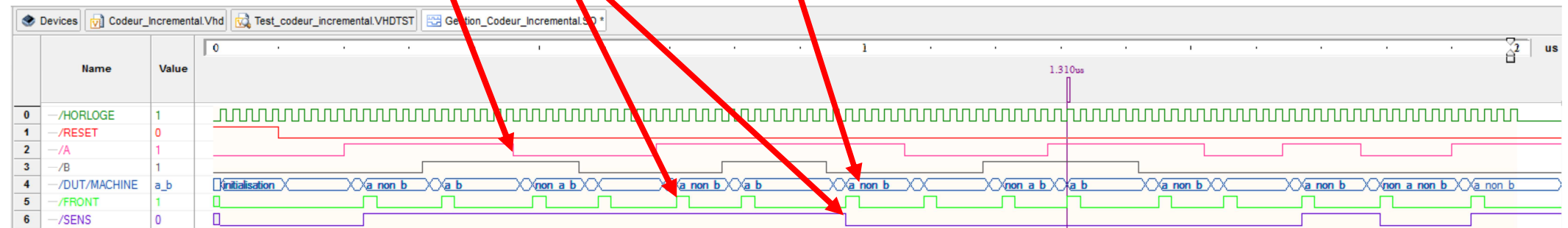
Le nouveau fichier possède les mêmes entrées et sorties et le même fonctionnement que le précédent, il est donc possible de le simuler avec le même fichier de tests. Seuls les signaux internes sont différents.

On peut remarquer que les états de la machine à états apparaissent bien.

Cela peut aider au débogage.

Les sorties FRONT et SENS sont identiques à la première simulation. Les deux solutions, bien que très différentes donnent le même résultat.

Le retard est ici aussi de 2 coups d'horloges au maximum.



Le meilleur choix pourrait alors être celui mettant en œuvre le minimum de ressources et/ ou le plus rapide. Seul le codeur incrémental a été placé (fitter) dans le FPGA, voici les résultats :

Première solution :

Results Summary	
Device Resources - Usage Summary	
Total logic elements	8 / 114,480 (< 1%)
Total registers*	6 / 117,053 (< 1%)
-- Dedicated logic registers	6 / 114,480 (< 1%)
-- IO registers	0 / 2,573 (0%)
IO pins	6 / 529 (1%)
Global clocks	1 / 20 (5%)
* Register count does not include registers inside RAM	
Design Statistics - Timing Summary	
H	580.05 M
Show Results Summary dialog Note: The Results Summary also appears in the Output panel	
Print...	Copy Report Close

Seconde solution :

Results Summary	
Device Resources - Usage Summary	
Total logic elements	37 / 114,480 (< 1%)
Total registers*	9 / 117,053 (< 1%)
-- Dedicated logic registers	9 / 114,480 (< 1%)
-- IO registers	0 / 2,573 (0%)
IO pins	6 / 529 (1%)
Global clocks	1 / 20 (5%)
* Register count does not include registers inside RAM	
Design Statistics - Timing Summary	
H	383.14 MHz
Show Results Summary dialog Note: The Results Summary also appears in the Output panel	
Print...	Copy Report Close

On peut remarquer que la solution avec machine à états est plus gourmande en ressources : 37 éléments logiques au lieu de 8, 9 registres (bascules) au lieu de 6.

La vitesse maximum de l'horloge pour la première solution est de 580 MHz alors que pour la seconde est n'est que de 383 MHz. Dans notre cas ce n'est pas critique.

La fonction FP4 est à nouveau validée.

VI. CRÉATION D'UN SYMBOLE POUR LA FONCTION FP3

Vérifier que la fenêtre de travail est bien le fichier VHDL Codeur_Incremental.Vhd puis créer un symbole avec la commande :

⇒ **Design ⇒ Create Schematic Part From File**

Faire **Ok**. Le symbole apparaît alors à l'écran.

Sauvegarder le sous le nom **Codeur_Incremental.SchLib**

Fermez-le : il nous servira plus tard.

VII. ANNEXES

```
1  -- Projet Gestion_Codeur_Incremental
2  -- CODEUR_INCREMENTAL
3
4  -- Description VHDL d'un composant qui détecte les fronts des entrées A et B
5  -- et le sens de rotation d'un codeur incrémental
6  -- sortie FRONT indiquant qu'un déplacement vient d'avoir lieu (actif à 1)
7  -- sortie SENS indiquant le sens du déplacement qui vient d'avoir lieu
8  -- 08/11/2012 C. D. Lycée Chevroliier Angers
9
10 library IEEE;                                --Liste des librairies prédéfinies utilisées
11 use IEEE.STD_LOGIC_1164.ALL;
12 use IEEE.STD_LOGIC_ARITH.ALL;
13 use IEEE.STD_LOGIC_UNSIGNED.ALL;
14
15 entity CODEUR_INCREMENTAL is                  --Description de l'entité (broches du composant)
16 port (
17     HORLOGE      : in  std_logic;             --Entrée de synchronisation
18     RESET        : in  std_logic;             --Entrée d'initialisation (systèmes synchrone)
19     A             : in  std_logic;             --Entrée A connectée au codeur incrémental
20     B             : in  std_logic;             --Entrée B connectée au codeur incrémental
21     FRONT        : out std_logic;             --Sortie indiquant qu'un déplacement à eu lieu
22     SENS          : out std_logic;             --Sortie indiquant le sens du déplacement
23 );
24 end CODEUR_INCREMENTAL;                       --Fin de la description de l'entité
25
26 architecture ARCH_CODEUR_INCREMENTAL of      --Description de l'architecture
27     CODEUR_INCREMENTAL is                    --Comportement interne du composant (silicium)
28     signal A_ANT_1 : std_logic;              --Valeur du signal A il y a 1 coup d'horloge
29     signal B_ANT_1 : std_logic;              --Valeur du signal B il y a 1 coup d'horloge
30     signal A_ANT_2 : std_logic;              --Valeur du signal A il y a 2 coups d'horloge
31     signal B_ANT_2 : std_logic;              --Valeur du signal B il y a 2 coups d'horloge
32     begin                                     --Début de la description synchrone
33     process (HORLOGE)                       --Partie synchrone sur le signal HORLOGE
34         variable VALEURS : std_logic_vector (3 downto 0); -- Variable servant à la concaténation
35         begin                                 --Fonctionnement complètement synchrone
36             VALEURS := A_ANT_2 & B_ANT_2 &    --Déclaration d'une variable pour concaténer
37                 A_ANT_1 & B_ANT_1;           --les 4 bits pour détection de front
38         if (HORLOGE'event and HORLOGE = '1') --Attente d'un front montant du signal HORLOGE
39             then
40         if (RESET = '1')                     --Si RESET actif (à la mise sous tension)
41             then
42                 A_ANT_1 <= A;                --Initialise les signaux internes
43                 A_ANT_2 <= A;                --Initialise les signaux internes
44                 B_ANT_1 <= B;                --Initialise les signaux internes
45                 B_ANT_2 <= B;                --Initialise les signaux internes
46                 FRONT <= '0';               --Initialise les ports en sorties
47                 SENS <= '0';                --Initialise les ports en sorties
```

```

48     else
49         A_ANT_1 <= A;
50         B_ANT_1 <= B;
51         A_ANT_2 <= A_ANT_1;
52         B_ANT_2 <= B_ANT_1;
53         case VALEURS is
54             when "0010" =>
55                 FRONT <= '1';
56                 SENS <= '1';
57             when "1011" =>
58                 FRONT <= '1';
59                 SENS <= '1';
60             when "1101" =>
61                 FRONT <= '1';
62                 SENS <= '1';
63             when "0100" =>
64                 FRONT <= '1';
65                 SENS <= '1';
66             when "0001" =>
67                 FRONT <= '1';
68                 SENS <= '0';
69             when "0111" =>
70                 FRONT <= '1';
71                 SENS <= '0';
72             when "1110" =>
73                 FRONT <= '1';
74                 SENS <= '0';
75             when "1000" =>
76                 FRONT <= '1';
77                 SENS <= '0';
78             when others =>
79                 FRONT <= '0';
80         end case;
81     end if;
82 end if;
83 end process;
84 end ARCH_CODEUR_INCREMENTAL;
85
--Sinon
--Mémorisation de la nouvelle valeur de A
--Mémorisation de la nouvelle valeur de B
--Mémorisation de l'ancienne valeur de A
--Mémorisation de l'ancienne valeur de B
--Début du choix multiple
--Cas n°1
--Informe qu'un font a été détecté
--Informe du sens positif
--Cas n°2
--Informe qu'un font a été détecté
--Informe du sens positif
--Cas n°3
--Informe qu'un font a été détecté
--Informe du sens positif
--Cas n°4
--Informe qu'un font a été détecté
--Informe du sens positif
--Cas n°5
--Informe qu'un font a été détecté
--Informe du sens négatif
--Cas n°6
--Informe qu'un font a été détecté
--Informe du sens négatif
--Cas n°7
--Informe qu'un font a été détecté
--Informe du sens négatif
--Cas n°8
--Informe qu'un font a été détecté
--Informe du sens négatif
--Dans tous les autres cas
--Pas de font détecté
--Fin du choix multiple
--Fin de la condition sur l'entrée RESET
--Fin de la condition sur l'entrée HORLOGE
--Fin de la description synchrone
--Fin de la description interne

```

```

1  -----
2  -- VHDL Testbench for codeur_incremental
3  -- 2012 11 10 8 57 38
4  -- Created by "EditVHDL"
5  -- "Copyright (c) 2002 Altium Limited"
6  -----
7
8  Library IEEE;
9  Use      IEEE.std_logic_1164.all;
10 Use      IEEE.std_logic_textio.all;
11 Use      STD.textio.all;
12 -----
13
14 -----
15 entity Testcodeur_incremental is
16 end Testcodeur_incremental;
17 -----
18
19 -----
20 architecture stimulus of Testcodeur_incremental is
21     file RESULTS: TEXT open WRITE_MODE is "results.txt";
22     procedure WRITE_RESULTS(
23         A: std_logic;
24         B: std_logic;
25         FRONT: std_logic;
26         HORLOGE: std_logic;
27         RESET: std_logic;
28         SENS: std_logic
29     ) is
30         variable l_out : line;
31     begin
32         write(l_out, now, right, 15);
33         write(l_out, A, right, 2);
34         write(l_out, B, right, 2);
35         write(l_out, FRONT, right, 2);
36         write(l_out, HORLOGE, right, 2);
37         write(l_out, RESET, right, 2);
38         write(l_out, SENS, right, 2);
39         writeline(RESULTS, l_out);
40     end procedure;
41
42     component codeur_incremental
43     port (
44         A: in std_logic;
45         B: in std_logic;
46         FRONT: out std_logic;
47         HORLOGE: in std_logic;
48         RESET: in std_logic;
49         SENS: out std_logic
50     );
51 end component;
52
53 signal A: std_logic;
54 signal B: std_logic;
55 signal FRONT: std_logic;
56 signal HORLOGE: std_logic;
57 signal RESET: std_logic;
58 signal SENS: std_logic;
59
60 begin
61     DUT:codeur_incremental port map (
62         A => A,
63         B => B,
64         FRONT => FRONT,
65         HORLOGE => HORLOGE,
66         RESET => RESET,
67         SENS => SENS
68     );
69
70     GENERATION_HORLOGE:process
71     begin
72         HORLOGE <= '0';
73         wait for 10 ns;
74         HORLOGE <= '1';
75         wait for 10 ns;
76     end process;
77
78     STIMULUS0:process
79     begin
80         -- insert stimulus here
81         RESET <= '1';
82         A <= '0';
83         B <= '0';
84         wait for 100 ns;
85         RESET <= '0';
86         wait for 100 ns;
87         A <= '1';
88         B <= '0';
89         wait for 120 ns;
90         A <= '1';
91         B <= '1';
92         wait for 140 ns;
93         A <= '0';
94         B <= '1';
95         wait for 100 ns;
96         A <= '0';
97         B <= '0';
98         wait for 120 ns;
99         A <= '1';
100        B <= '0';
101        wait for 100 ns;
102        A <= '1';
103        B <= '1';
104        wait for 160 ns;
105
106        A <= '1';
107        B <= '0';
108        wait for 120 ns;
109        A <= '0';
110        B <= '0';
111        wait for 120 ns;
112        A <= '0';
113        B <= '1';
114        wait for 100 ns;
115        A <= '1';
116        B <= '1';
117        wait for 140 ns;
118        A <= '1';
119        B <= '0';
120        wait for 100 ns;
121        A <= '0';
122        B <= '0';
123        wait for 120 ns;
124        A <= '1';
125        B <= '0';
126        wait for 120 ns;
127        A <= '0';
128        B <= '0';
129        wait for 140 ns;
130        A <= '1';
131        B <= '0';
132        wait for 120 ns;
133        A <= '0';
134        B <= '0';
135        wait;
136    end process;
137
138    WRITE_RESULTS(
139        A,
140        B,
141        FRONT,
142        HORLOGE,
143        RESET,
144        SENS
145    );
146 end architecture;
147 -----
148 -----
149 -----

```

```

70 GENERATION_HORLOGE:process
71 begin
72     HORLOGE <= '0';
73     wait for 10 ns;
74     HORLOGE <= '1';
75     wait for 10 ns;
76 end process;
77
78 STIMULUS0:process
79 begin
80     -- insert stimulus here
81     RESET <= '1';
82     A <= '0';
83     B <= '0';
84     wait for 60 ns;
85     RESET <= '0';
86     wait for 80 ns;
87     A <= '1';
88     B <= '0';
89     wait for 60 ns;
90     A <= '0';
91     B <= '0';
92     wait for 60 ns;
93     A <= '1';
94     B <= '0';
95     wait for 60 ns;
96     A <= '0';
97     B <= '0';
98     wait for 60 ns;
99     A <= '1';
100    B <= '0';
101    wait for 60 ns;
102    A <= '0';
103    B <= '0';
104    wait for 60 ns;
105    A <= '1';
106    B <= '0';
107    wait;
108 end process;

```

HORLOGE

RESET

A

B

Active-HDL Student Edition

Entity: Codeur_Incremental

Architecture: Codeur_Incremental_arch

SENS

FRONT

MACHINE

HORLOGE

No clock enable

