INTRODUCTION AU LANGAGE DE DESCRIPTION DE MATÉRIEL – VHDL

I- INTRODUCTION.

Apparu au début des années quatre-vingt, le langage de description de matériel **HDL** (**H**ardware **D**escription **L**angage) est un langage qui sert à décrire le matériel avec pour objectif :

- ***La spécification,***
- ***La modélisation,***
- ***La simulation,***
- ***La documentation,***
- ***La synthèse logique….***

Avec ce langage, l'information est transformée par des composants qui sont utilisés de façon statique et reliés par des signaux. Le langage **HDL** reflète certains attributs des autres langages textuels, mais il en diffère considérablement car il est basé sur un modèle de flux de données dans lequel les Entrées/Sorties sont connectées à une série de blocs de fonctions via des signaux.

Quelques années plus tard, en 1985, le DoD, Département américain de la défense met dans le domaine public le langage **VHDL 7.2** et propose à l'**IEEE** (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers = Institut des ingénieurs électriciens et électroniciens) de transformer le langage en norme, chose faite le 10 Décembre 1987 sous le nom peu commercial de **IEEE 1076-1987**.

Le langage sera plus connu sous le nom naturel de **VHDL** :

VHSIC (Very High Speed Integrated Circuit)
Hardware
Description
Langage.

Depuis, les outils logiciels de développement proposant les compilateurs/simulateurs ne cessèrent d'évoluer. Leurs performances aujourd'hui permettent la conception et le développement simplifié des circuits numériques de plus en plus complexes.

II- STRUCTURE D'UN MODÈLE VHDL.

Un modèle **VHDL** est constitué de trois parties principales :

- ***La spécification d'appel :***
  - De bibliothèques utiles ; mot-clef : **LIBRARY**,
  - De contenus à exporter ; mot-clef : **USE**.
- ***La spécification d'entité*** ; mot-clef : **ENTITY**. Cette spécification correspond à la vue externe du modèle. Elle correspond au symbole dans les représentations schématiques.
- ***La spécification de l'architecture de l'entité*** ; mot-clef : **ARCHITECTURE**. C'est la vue interne du modèle. Elle est toujours associée à une entité.

```
LIBRARY ressource_lib ;
USE resource_lib.pack_lib.all ;
ENTITY Nom IS
        …… ;
END ENTITY Nom ;
ARCHITECTURE archi1 OF Nom IS
        …… ;
END archi1 ;
```

A- SPÉCIFICATION D'APPEL.

Cette spécification permet de s'appuyer sur des bibliothèques contenant des descriptions courtes et hiérarchisées dont on est sûr de la justesse syntaxique. Les mots de cette spécification sont **LIBRARY** et **USE**.

Exemple.

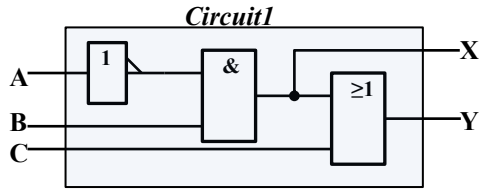
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
```

L'extension **".all"** permet d'exporter tout le contenu de la bibliothèque **"STD_LOGIC_1164".**

B- SPÉCIFICATION D'ENTITÉ.

Cette spécification permet, après appel de bibliothèques utiles, de définir les **paramètres génériques** et les **ports** (mot-clé : **PORT**) par lequel l'entité pourra ***recevoir*** et ***envoyer de l'information*** à et vers son environnement.

Exemple.




```
ENTITY Circuit1 IS
        PORT(A, B, C : IN   STD_LOGIC;
                X, Y  : OUT  STD_LOGIC);
END Circuit1;
```

C- SPÉCIFICATION D'ARCHITECTURE.

La spécification d'architecture permet de définir le fonctionnement du modèle par l'intermédiaire :

- d'instanciations de composants (c'est-à-dire en définissant une netlist d'objets),
- d'instructions concurrentes permettant de manipuler l'information numérique,
- et d'instructions simultanées mettant en jeu les valeurs analogiques du modèle.

L'architecture peut manipuler les informations disponibles sur les ports de l'entité.

Exemple.

```
ARCHITECTURE Logique OF Circuit1 IS
BEGIN
        X <= (NOT (A) AND B);
        Y <= (NOT (A) AND B) OR C;
END Logique;
```

III- CONVENTION LEXICALE.

✓ VHDL ne fait aucune différence entre les majuscules et les minuscules. Mais pour la bonne lisibilité il est souvent préférable d'écrire les mots réservés du langage en majuscules et le reste en minuscules.

✓ Les commentaires sont liés à la ligne et sont totalement ignorés par le compilateur. Ils commencent par deux tirets **"--"** et finissent à la fin de la ligne.

✓ Une instruction se termine toujours par un point-virgule **" ;"**.

✓ Les littéraux de **type caractère** peuvent être exprimés grâce aux 95 caractères imprimables sur les 128 caractères du code **ASCII**. Ils seront notés entre apostrophes. ***Exemple :*** 'a', 'B',…

✓ Les littéraux de **type chaînes de caractères** seront exprimés à partir d'une suite de caractères entre guillemets. L'opérateur de concaténation de chaînes est le "&". ***Exemple :*** "Le langage " & "VHDL."

✓ Les littéraux **décimaux** seront exprimés avec les chiffres, les **réels** devront obligatoirement faire apparaître le point décimal. Il est également possible d'exprimer les nombres en notation scientifique.

Exemple : les entiers **12345**, **21e3** ou encore **21E3**. Les réels **123.45**, **21.5e6** ou encore **21.5E6**

✓ On peut aussi exprimer la valeur d'un littéral dans n'importe quelle base. Pour cela il suffit d'écrire la base, de poser un #, d'écrire la valeur de cette base et de fermer par le #.

Exemple : 2#**10110**# est un entier écrit en base 2 et qui vaut 13. 16#**1AF**# est un entier hexadécimal qui vaut 431.

✓ Pour faciliter l'expression de vecteurs de bits, il est possible de les exprimer grâce à des chaînes plus compactes. ***Exemple :*** B **"00101011"** est un vecteur de bits qui vaut (**'0'**, **'0'**, **'1'**, **'0'**, **'1'**, **'0'**, **'1'**, **'1'**)

IV- LES OPÉRATEURS.

VHDL définit 6 classes d'opérateurs et les niveaux de priorité associés. L'évaluation d'une expression commence par la priorité la plus haute.

a) **Opérateurs logiques : NOT**, **AND**, **OR**, **NAND**, **NOR**, **XOR**, **XNOR**,

Décalages logiques: **SLL** (décalage logique à gauche), **SRL** (décalage logique à droite).

Décalages arithmétiques : **SRA** (décalage arithmétique à droite), **SLA** (décalage arithmétique à gauche).

Rotation : **ROL** (rotation à gauche), **ROR** (rotation à droite).

b) **Opérateurs relationnels:** =, /=, >, <, >=, <=

Les opérateurs **égalité** (=) et **différent** (/=) s'appliquent sur tous les types sauf le type file. Les **inégalités** (< et >, >= et <=) sur tous les types scalaires et les vecteurs d'éléments entiers ou énumérés c'est-à-dire un type de données qui consiste en un ensemble de constantes appelées énumérateurs.

c) **Opérateurs arithmétiques:** +, –, *, /, **MOD** (Modulo), **REM** (Remainder), **ABS**, **

L'opérateur ** élève entier ou réel à une puissance entière. Les opérateurs **MOD** et **REM** sont définis pour le type entier.

L'opérateur **MOD** délivre le reste de la division avec le signe du diviseur tandis que l'opérateur **REM** délivre le reste avec le signe du dividende.

d) La concaténation.

La concaténation utilise l'opérateur **&** et s'applique aux vecteurs de taille quelconque.

Priorité des opérateurs.

Priorité							
↓	AND	OR	NAND	NOR	XOR		--logique
	=	/=	<	<=	>	>=	--relationnel
	+	–	&				--addition
	+	–					--signe
	*	/	MOD	REM			--multiplication
	**	ABS	NOT				--divers

TP. LANGAGE DE DESCRIPTION DE MATÉRIEL – VHDL : PROGRAMMATION.

☞ **Ce TP est constitué de deux parties :**

A- Programmation en langage VHDL d'un décodeur BCD-7Segments

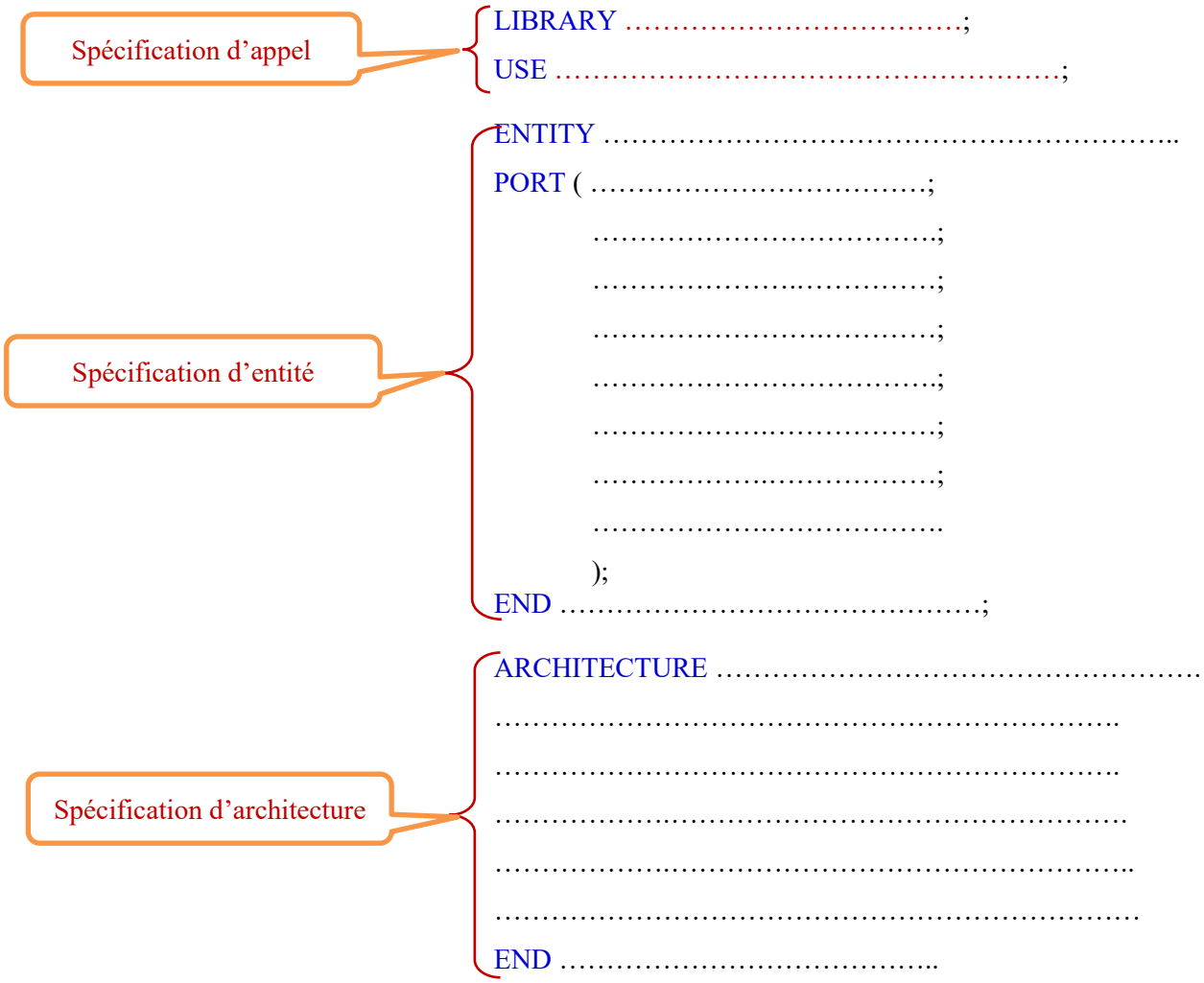
B- Programmation en langage VHDL des bascules RS, D et JK.

A- PROGRAMMATION EN LANGAGE VHDL D'UN DECODEUR BCD-7Segments.

1- En utilisant les opérateurs du langage **VHDL**, réécrire en fonction des entrées binaires **A**, **B**, **C** et **D** les équations des segments **Sa**, **Sb**, **Sc**, **Sd**, **Se**, **Sf** et **Sg** de l'afficheur 7 segments.

Avec les opérateurs traditionnels	Avec les opérateurs du langage VHDL
Exemple : $Y = \left(\overline{A} \cdot B\right) + C$	Y <= ((**NOT** A) **AND** B) **OR** C;
Sa =	Sa <=

2- Donner ci-dessous le programme en langage **VHDL** du décodeur BCD-7Segments à implanter dans le circuit **FPGA Cyclone II : EP2C20F484C7** de la carte de développement **DE1 ALTERA**.



3- Saisir le programme avec le logiciel **"Quartus II Web Edition"**. ***Le circuit logique à programmer est le circuit FPGA Cyclone II – EP2C20F484C7.***

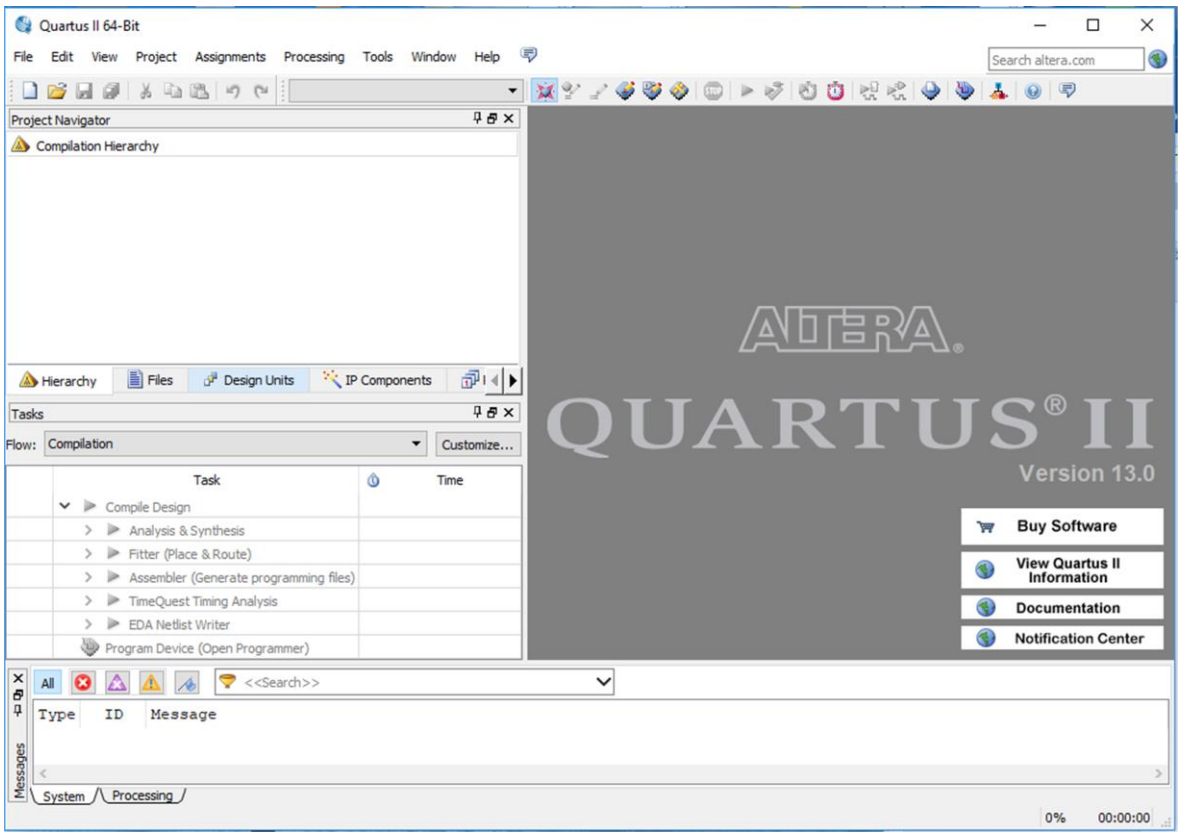
4- Réaliser la simulation afin de vérifier le bon fonctionnement du décodeur BCD-7Segments. **Donner et commenter quelques copies d'écran des résultats de la simulation.**

5- Programmer le circuit **FPGA** et vérifier le bon fonctionnement du décodeur BCD 7-Segments.

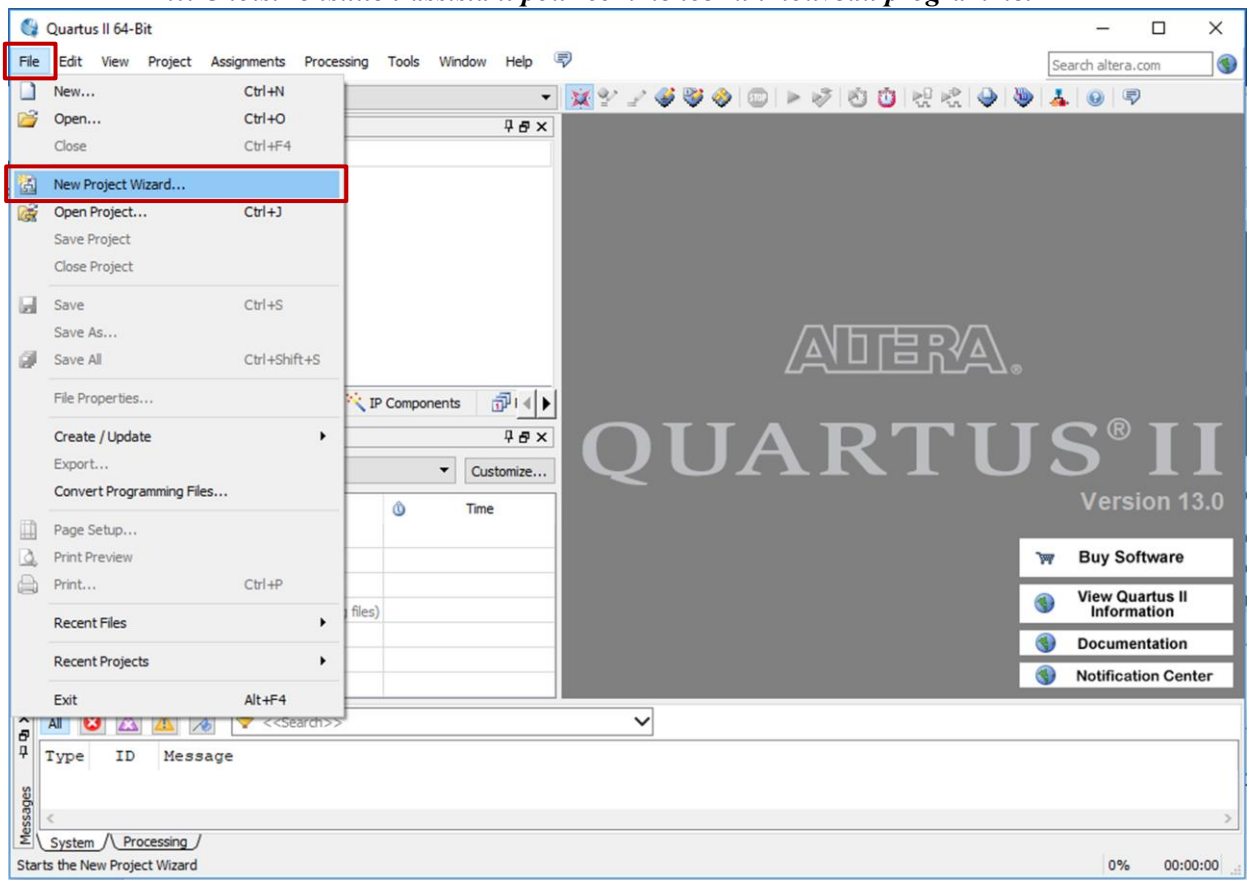
6- **Conclure.**

QUARTUS : Outil de développement et de téléversement de programmes pour les puces Altera.

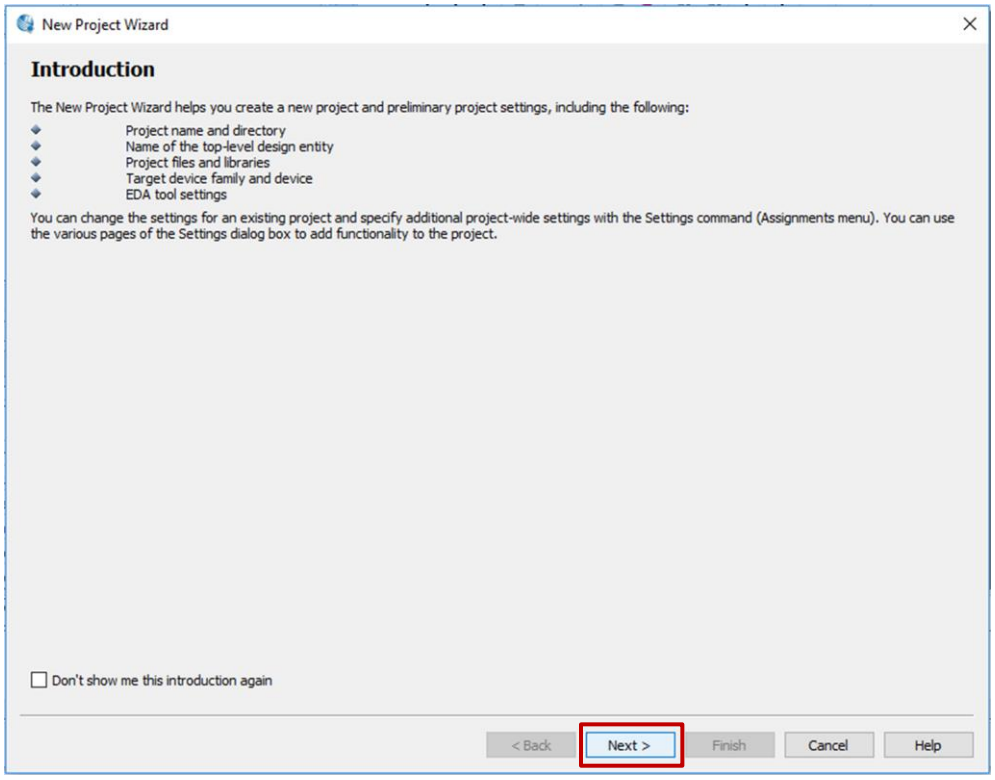
☞ Lancer le logiciel en cliquant sur l'icône **"QUARTUS II"** 



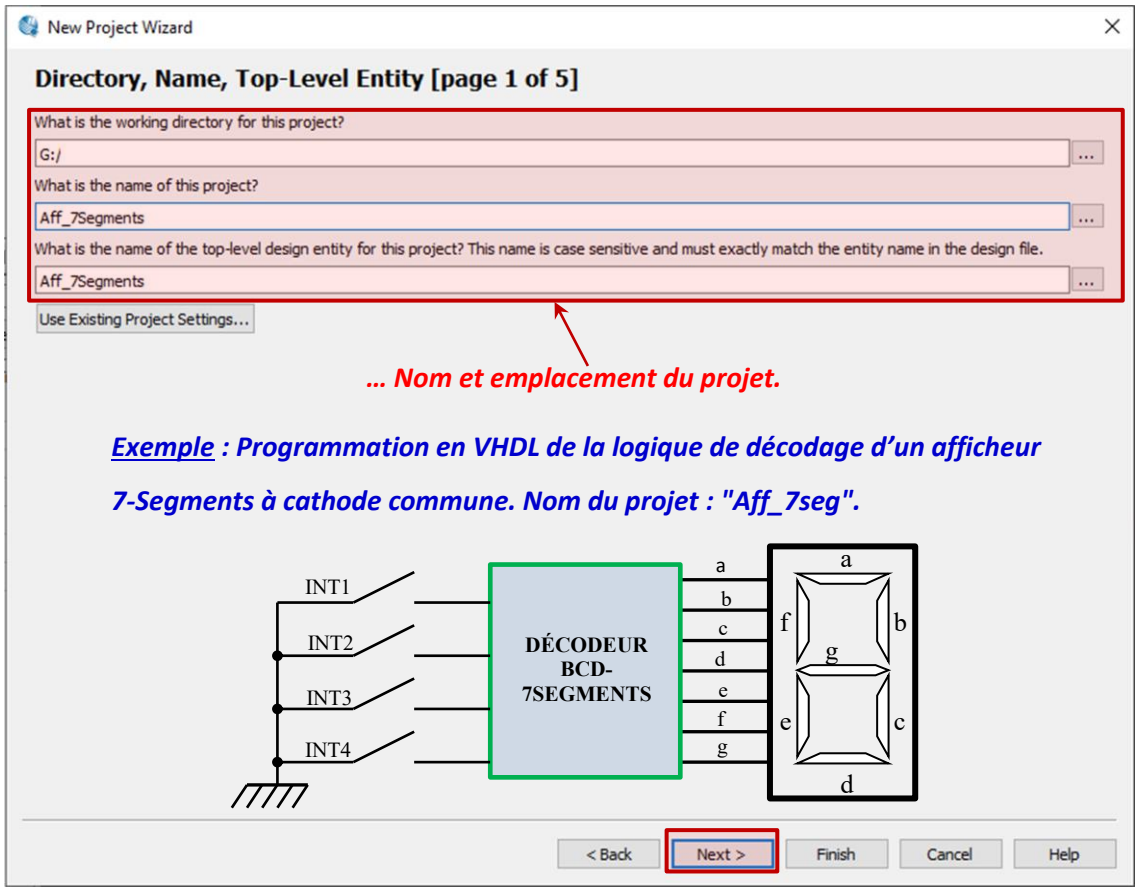
… Choisir ensuite l'assistant pour commencer un nouveau programme.



☞ ***Suivre les cinq étapes proposées par l'assistant.***

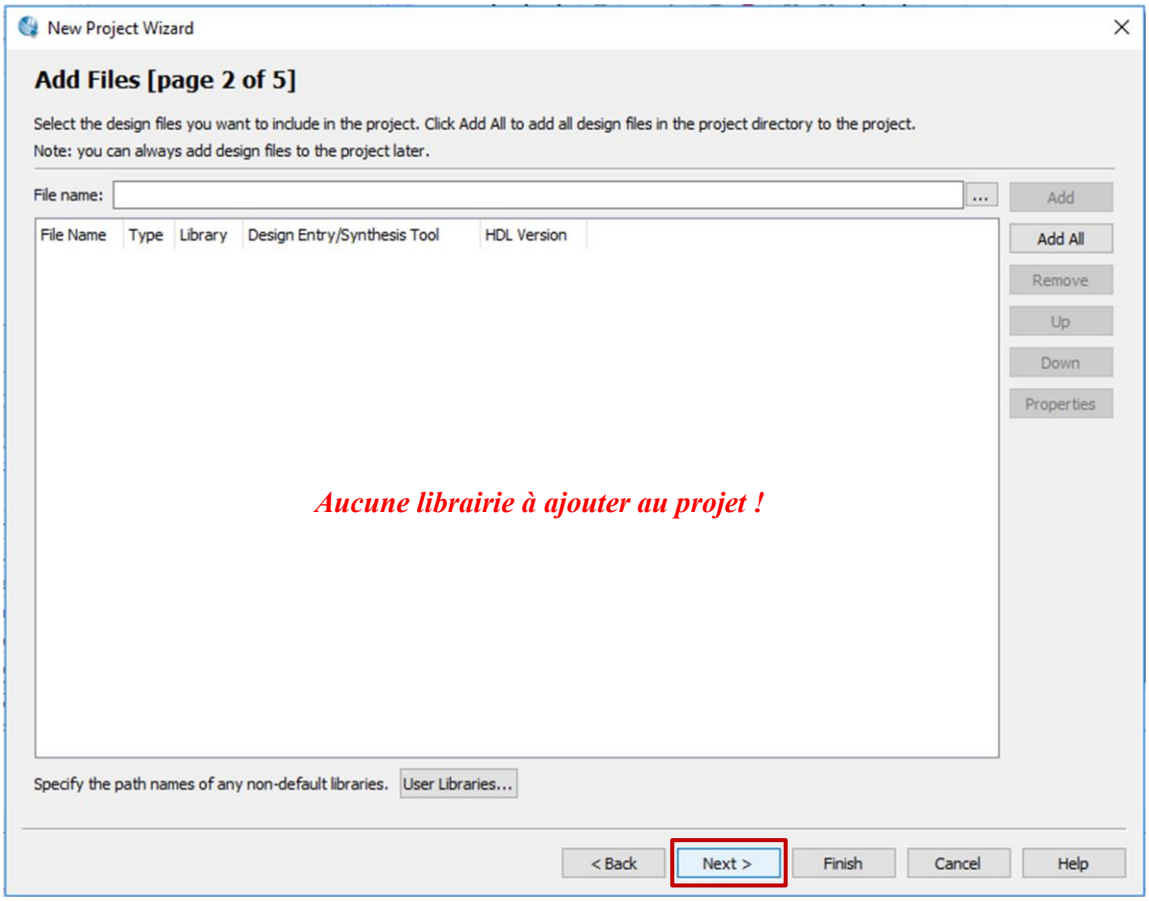


➢ ***Première étape.***

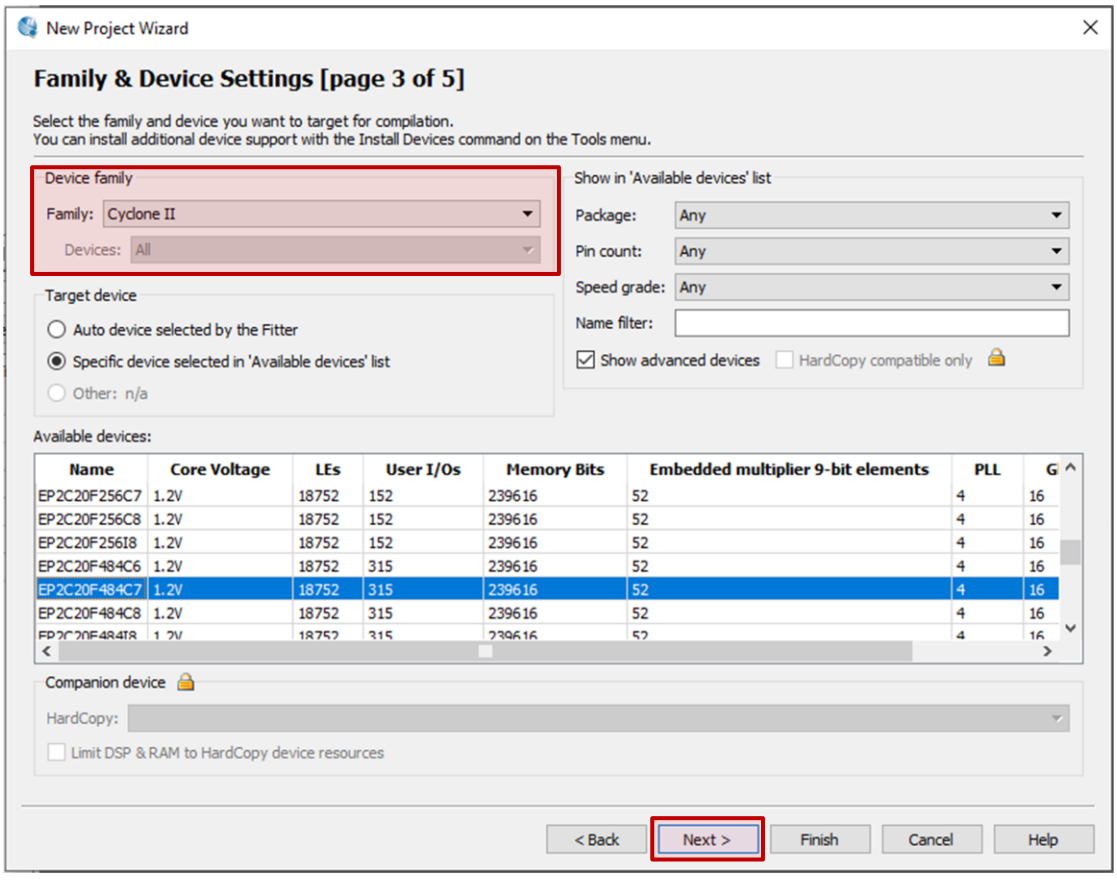


Remarque : ***L'afficheur utilisé est un afficheur cathode commune !***

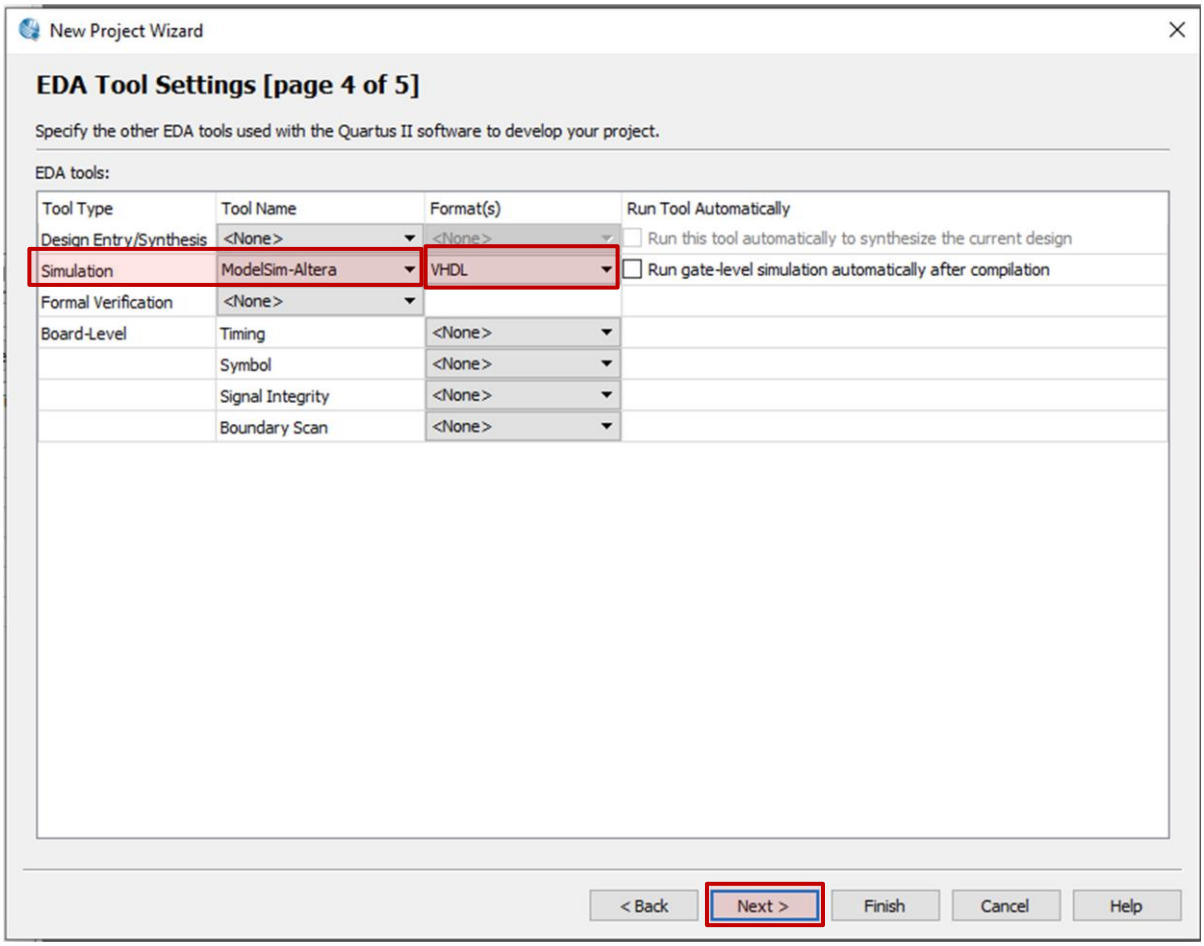
➢ ***Deuxième étape.***



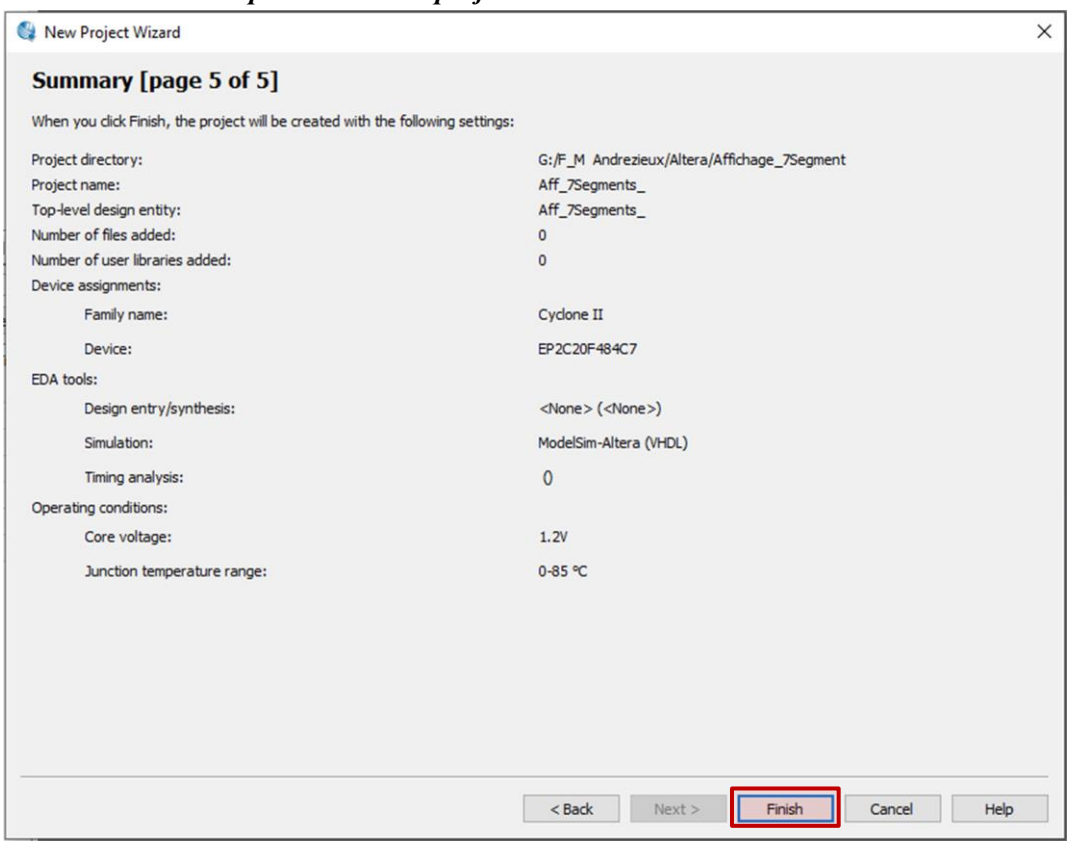
➢ ***Troisième étape.***



➢ ***Quatrième étape.***

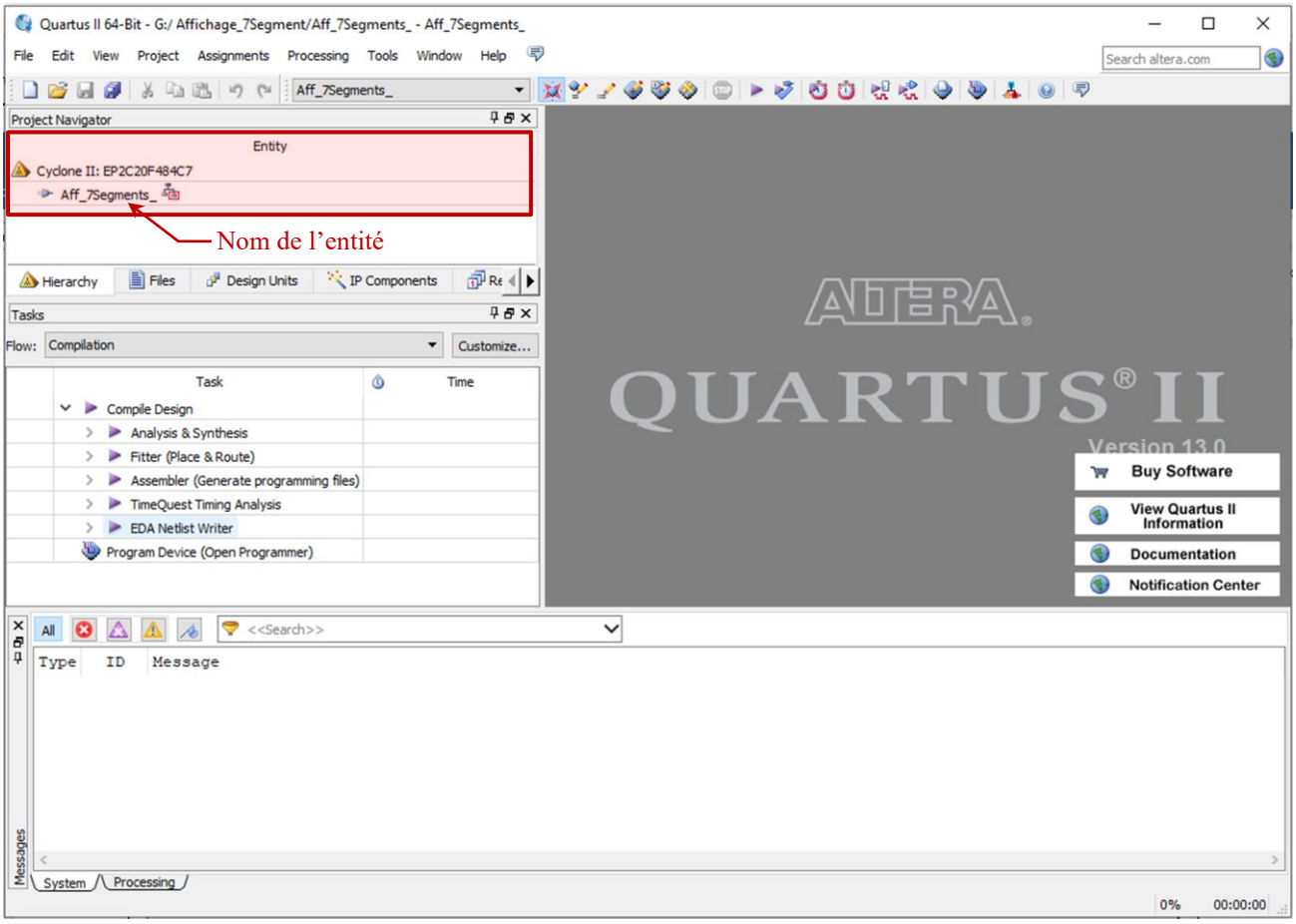


➢ ***Cinquième et dernière étape : résumé du projet.***

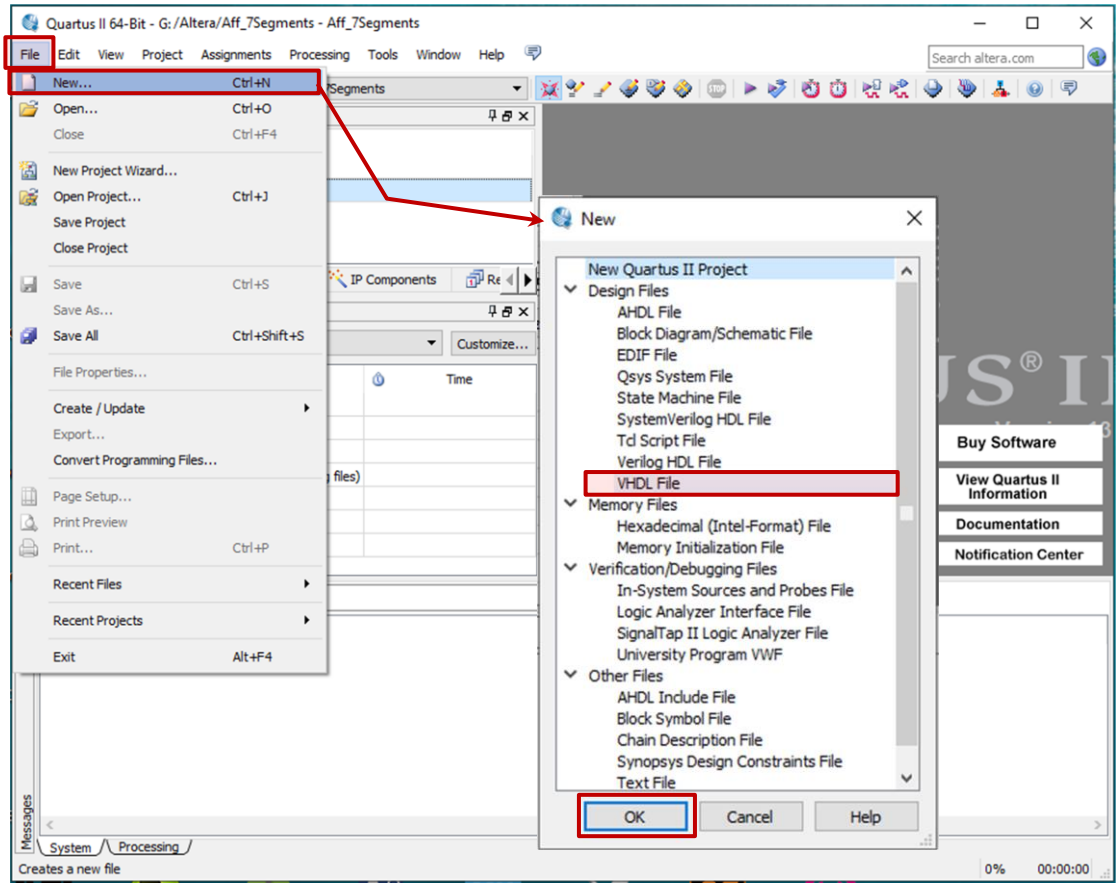


Cliquer sur **"Finish"** pour fermer l'assistant.

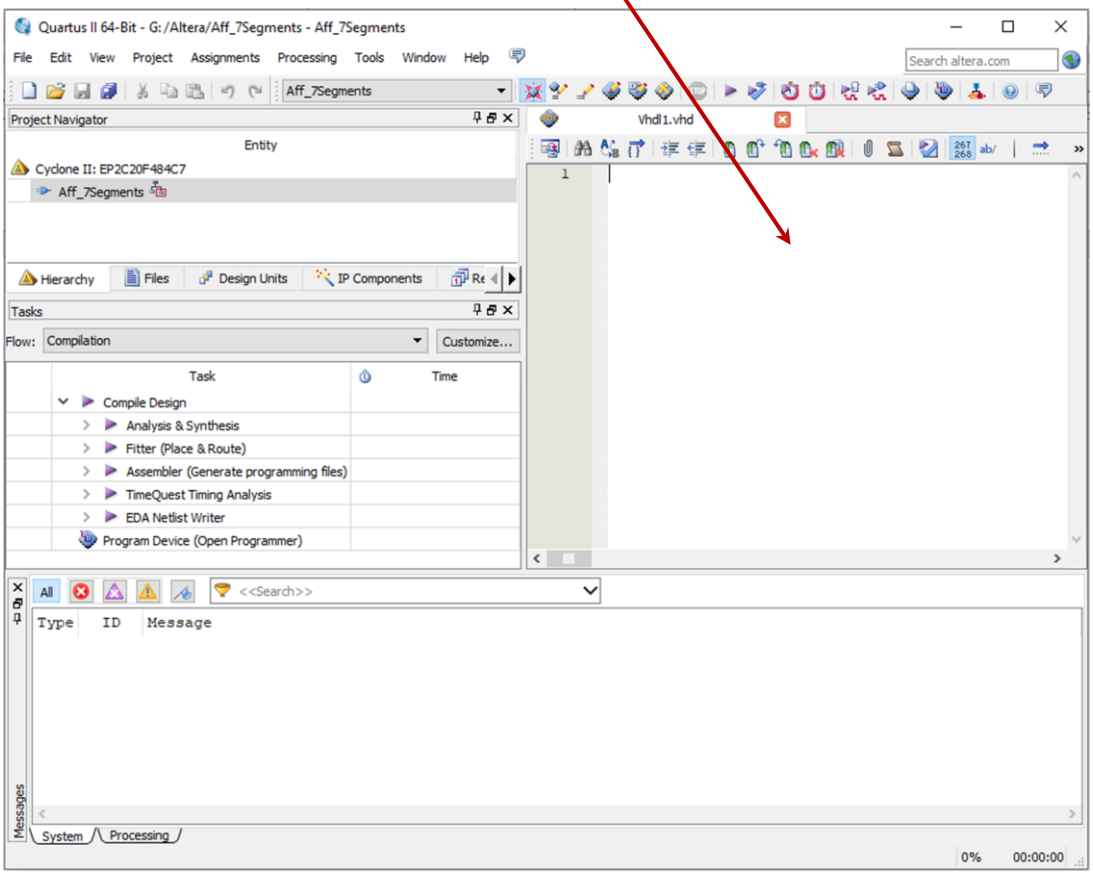
Environnement de développement "QUARTUS".



☺ Dans l'arborescence du projet **(Project Navigator)** figure déjà le nom de l'entité **(Entity)**. Pour choisir le langage de programmation, on utilisera le menu **"File"** de la barre d'outils.



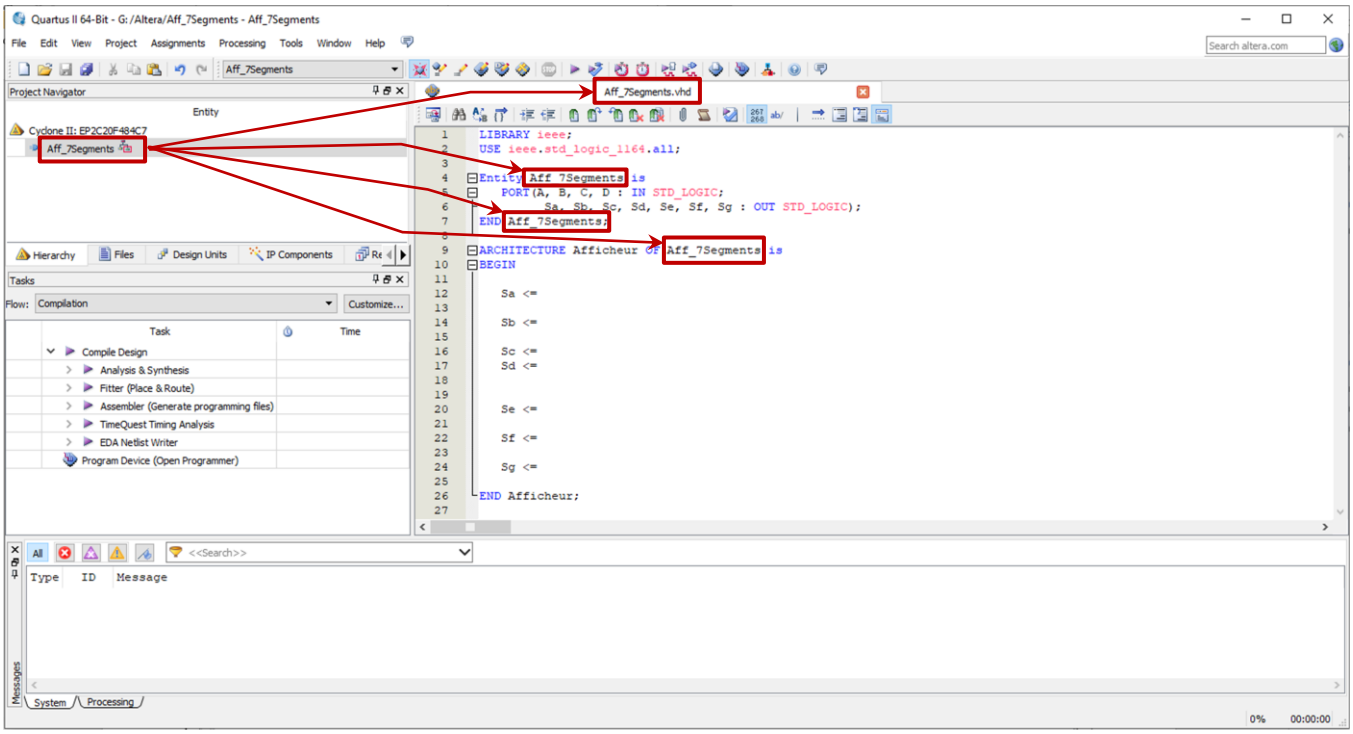
- **Fenêtre d'édition du programme VHDL.**



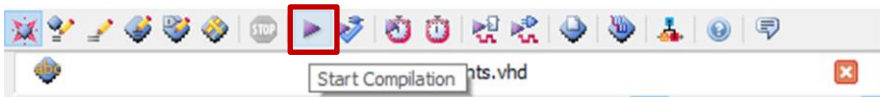
- **Dans la fenêtre d'édition, saisir le programme du décodeur BCD-7Segments.**

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

Entity Aff_7Segments is
        PORT(A, B, C, D : IN STD_LOGIC;
                Sa, Sb, Sc, Sd, Se, Sf, Sg : OUT STD_LOGIC);
END Aff_7Segments;

ARCHITECTURE Afficheur OF Aff_7Segments is
BEGIN

        Sa <= ……..
……;

        Sb <= ……..
……;

        Sc <= ……..
……;
        Sd <= ……..
……;

        Se <= ……..
……;

        Sf <= ……..
……;

        Sg <= ……..
……;

END Afficheur;
```

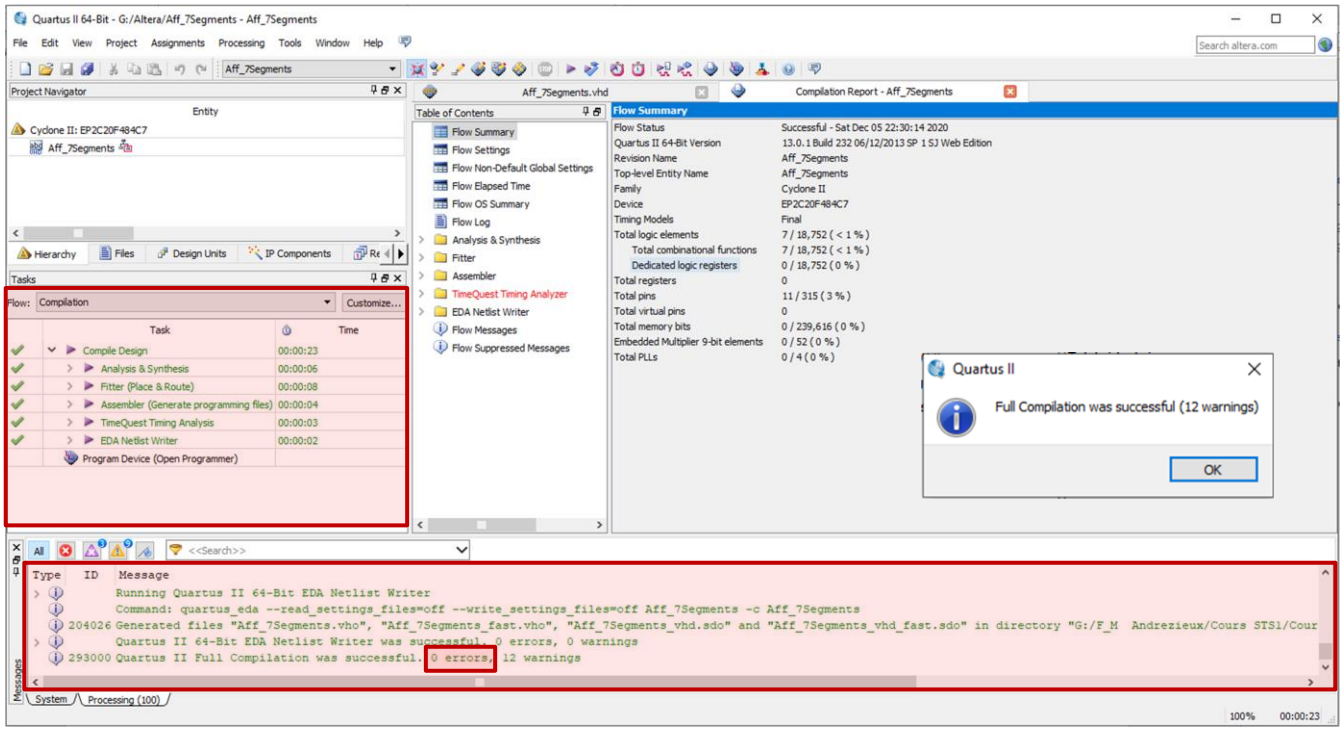
Attention ! ***Sauvegarder votre fichier VHDL en lui donnant le même nom que la spécification entité (Entity).***



☺ **Compiler le programme en cliquant sur l'icône "Start Compilation" de la barre d'outils. Corriger les erreurs éventuelles !**

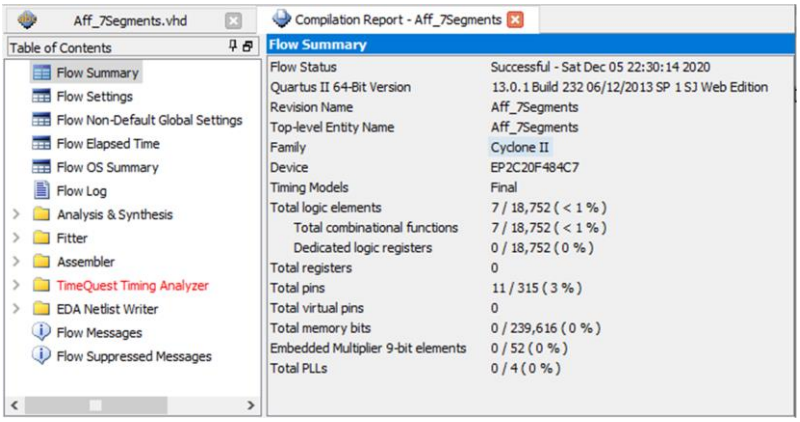


☺ **A la fin de la compilation (sans erreurs dans le programme), les fenêtres "Compilation" et "Message" informent des résultats de la compilation.**



… On ignore les "warnings" !

❖ **Consulter** la fenêtre **"Compilation report - …"** afin d'avoir une idée de l'utilisation et de la capacité du circuit FPGA utilisé **Cyclone II : EP2C20484C7**.



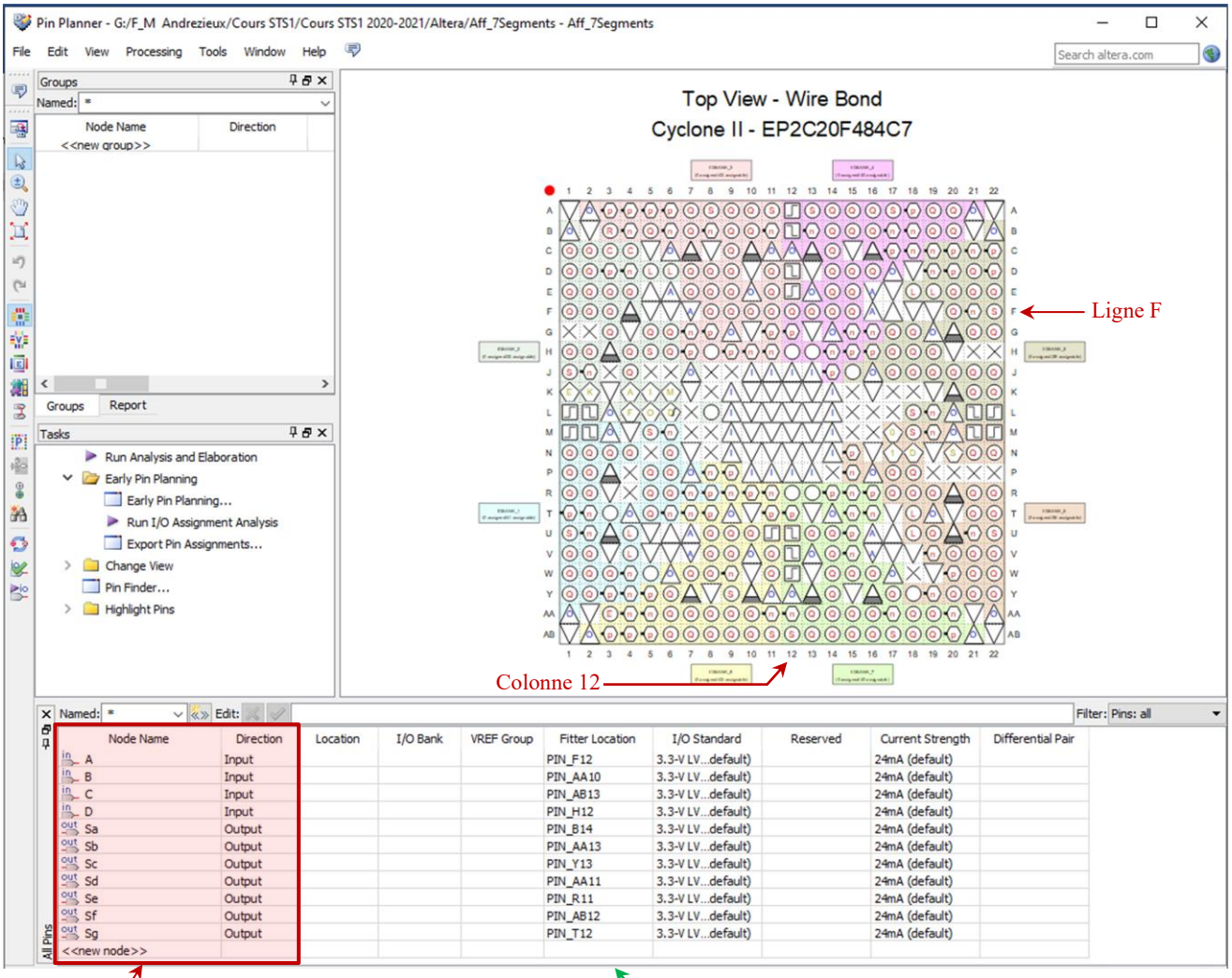
- 11 broches sur 315 ont été utilisées. Il s'agit bien évidemment des 4 entrées **A**, **B**, **C** et **D** du décodeur et des 7 sorties **Sa**, **Sb**, **Sc**, **Sd**, **Se** et **Sf**.
- Moins de 1% d'éléments logiques utilisés.
- …

☺ A la fin de la compilation, un dossier **"work"** est crée …

☺ La fenêtre **"Flow Summary"** résume les éléments du projet et l'utilisation du circuit FPGA Cyclone II EP2C20F484C7.

❖ **<u>Affectation des entrées A, B, C et D et des sorties Sa, Sb, Sc, Sd, Se et Sf</u> :**

Cette opération consiste à affecter chacune des entrées et sorties à une broche **d'entrée/sortie** du circuit **FPGA**. Pour cela **cliquez** sur l'icône **"Pin Planner".** Vous obtenez la fenêtre suivante :



Les entrées-sorties du décodeur BCD-7Segments.

Attribution par défaut des broches du circuit FPGA.

Remarque. Les sorties **Sa, Sb, Sc, Sd, Se, Sf** et **Sg** ont été respectivement affectées par défaut aux broches **PIN_B14, PIN_AA13, PIN_Y13, PIN_AA11, PIN_R12** et **PIN_T12** que l'on pourra repérer sur le **plan du brochage** du circuit FPGA **Cyclone II : EP2C20F484C7.**

De même, les entrées **A, B, C** et **D** sont respectivement affectées par défaut aux broches **PIN_F12, PIN_AA10, PIN_AB13** et **PIN_H12.**

Repérage. **Exemples : PIN_B14** → ligne **B** – colonne **14** ; **PIN_F12** → ligne **F** – colonne **12**.

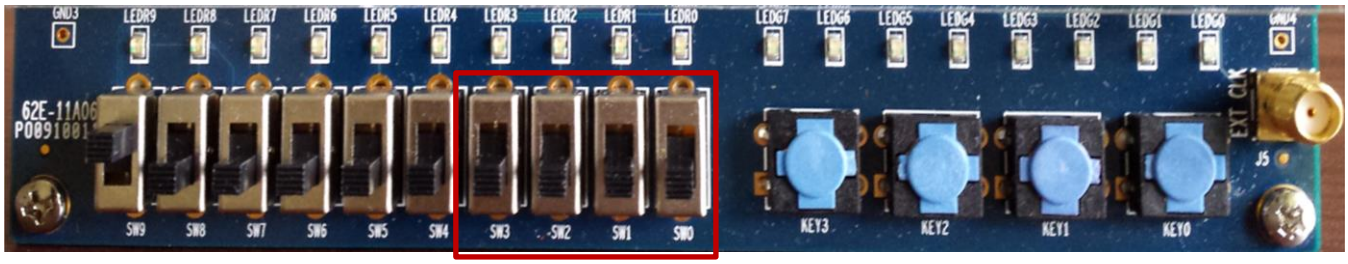
✍ Il faut réaffecter ces entrées et ces sorties afin de les attribuer aux interrupteurs et aux Leds de la carte de développement **DE1 ALTERA**.

❖ **Modification de l'affectation des Entrées A, B, C, D et des sorties Sa, Sb, Sc, Sd, Se, Sf et Sg.**

On va affecter les quatre entrées aux quatre premiers interrupteurs (SW[ ]) de la carte de développement DE1 ALTERA et les sept sorties aux sept segments de l'afficheur 7-Segments **HEX0** : ***afficheur à anode commune***.

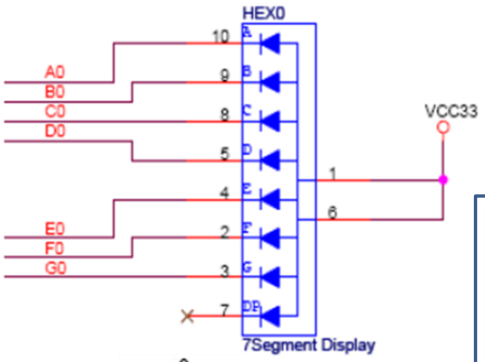
Les tableaux ci-dessous donnent les connexions des interrupteurs, des boutons poussoirs et des Leds aux broches du circuit FPGA **Cyclone II : EP2C20F484C7.**

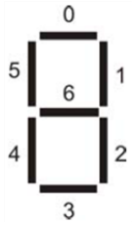
☞ **Les interrupteurs.**



Nom du signal	Broche du FPGA	Description	Entrées du décodeur BCD-7Segments
SW[0]	PIN_L22	Commutateur [0]	A
SW[1]	PIN_L21	Commutateur [1]	B
SW[2]	PIN_M22	Commutateur [2]	C
SW[3]	PIN_V12	Commutateur [3]	D

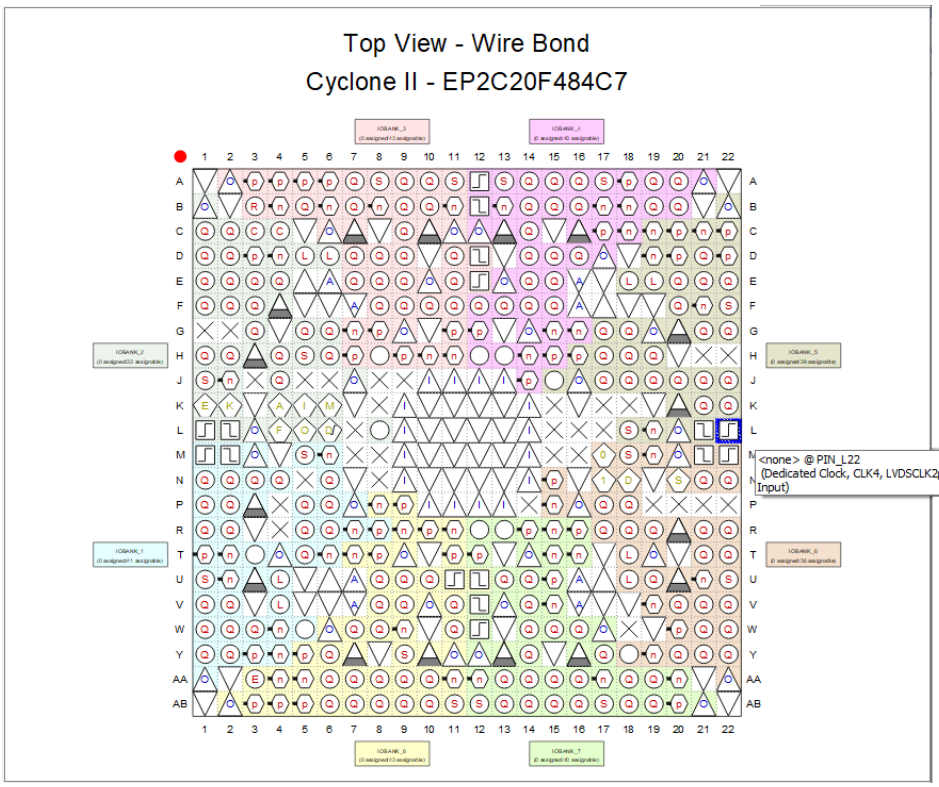
☞ **L'afficheur 7-Segments HEX0 : Afficheur à anode commune.**



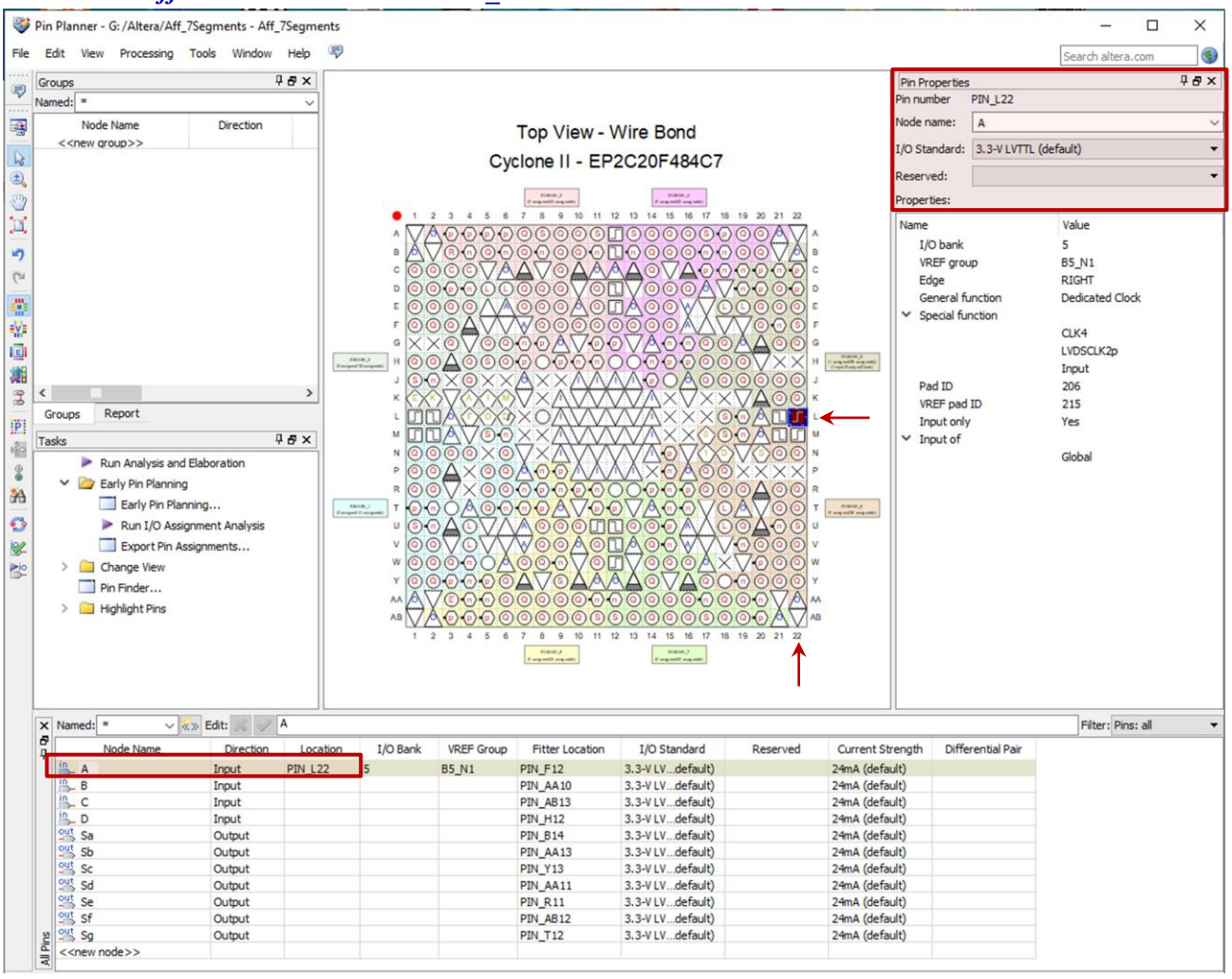


Nom du signal	Broche du FPGA	Description	Sorties du décodeur BCD-7Segments
HEX0[0]	PIN_J2	Segment 0 [0]	Sa
HEX0 [1]	PIN_J1	Segment 1 [1]	Sb
HEX0 [2]	PIN_H2	Segment 2 [2]	Sc
HEX0 [3]	PIN_H1	Segment 3 [3]	Sd
HEX0 [4]	PIN_F2	Segment 4 [4]	Se
HEX0 [5]	PIN_F1	Segment 5 [5]	Sf
HEX0 [6]	PIN_E2	Segment 6 [6]	Sg

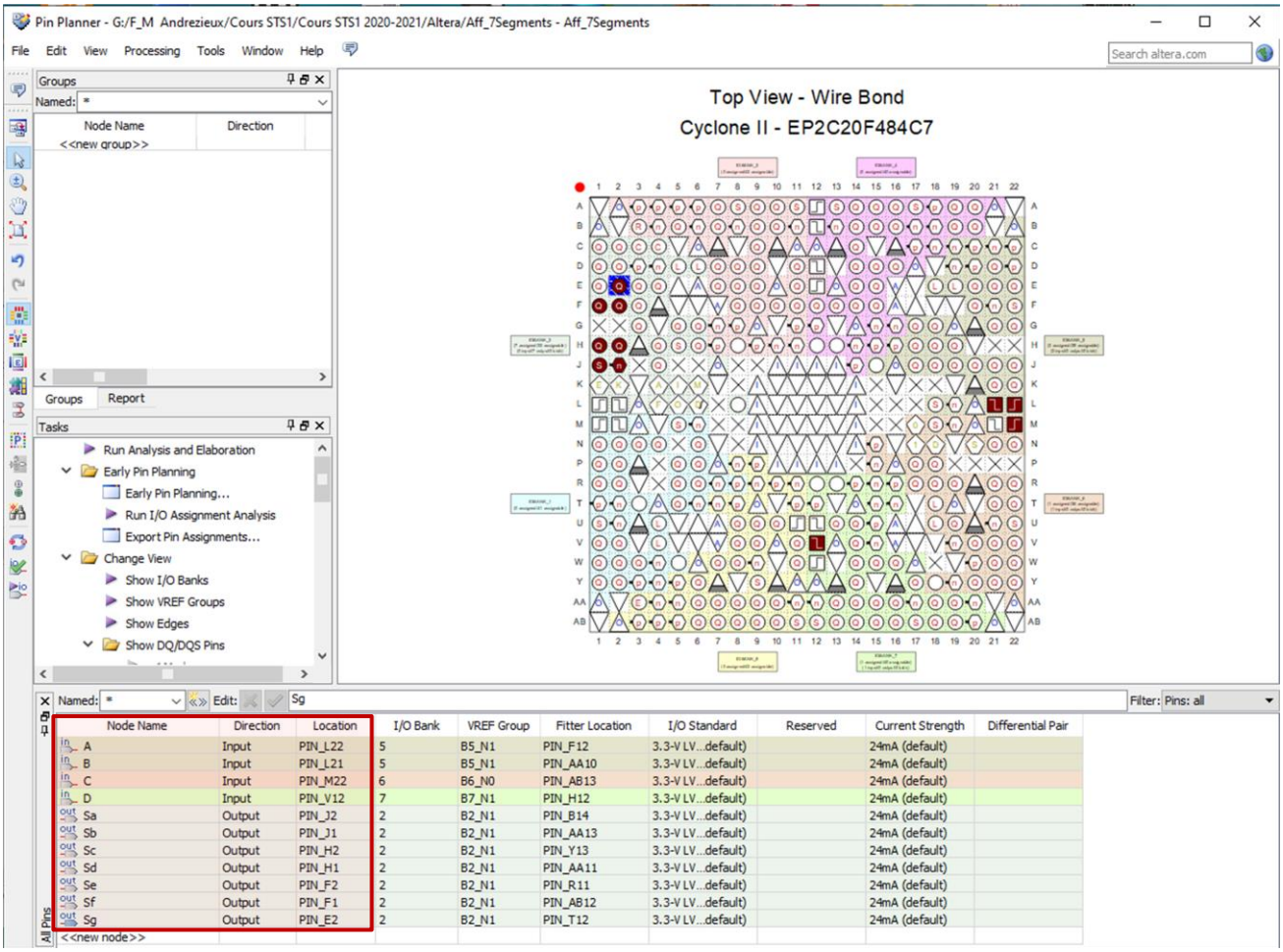
Un double clic sur chacune des broches à utilisées donne accès à ses propriétés ; ce qui permet alors d'affecter la broche à l'une des entrées-sorties du décodeur BCD-7Segments.



Affectation de la broche "PIN_L22" à l'entrée A du décodeur.



Affectation complète des broches du circuit FPGA aux entrées-sorties du décodeur BCD-7Segments.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
A	Input	PIN_L22	5	B5_N1	PIN_F12	3.3-V LV...default)		24mA (default)	
B	Input	PIN_L21	5	B5_N1	PIN_AA10	3.3-V LV...default)		24mA (default)	
C	Input	PIN_M22	6	B6_N0	PIN_AB13	3.3-V LV...default)		24mA (default)	
D	Input	PIN_V12	7	B7_N1	PIN_H12	3.3-V LV...default)		24mA (default)	
Sa	Output	PIN_J2	2	B2_N1	PIN_B14	3.3-V LV...default)		24mA (default)	
Sb	Output	PIN_J1	2	B2_N1	PIN_AA13	3.3-V LV...default)		24mA (default)	
Sc	Output	PIN_H2	2	B2_N1	PIN_Y13	3.3-V LV...default)		24mA (default)	
Sd	Output	PIN_H1	2	B2_N1	PIN_AA11	3.3-V LV...default)		24mA (default)	
Se	Output	PIN_F2	2	B2_N1	PIN_R11	3.3-V LV...default)		24mA (default)	
Sf	Output	PIN_F1	2	B2_N1	PIN_AB12	3.3-V LV...default)		24mA (default)	
Sg	Output	PIN_E2	2	B2_N1	PIN_T12	3.3-V LV...default)		24mA (default)	

Recompiler ensuite votre programme en cliquant sur l'icône **"Start Compilation"** .

Corriger les éventuelles erreurs…

*Afin de s'assurer du bon fonctionnement du **"Décodeur BCD-7Segments"**, on utilisera le simulateur « **ModelSim Altera** ». L'analyse des signaux issus de la simulation permettra de savoir si le décodeur fonctionne correctement ou non. Ci-dessous les broches des afficheurs de la carte de développement Altera.*

Afficheur HEX0

Nom du signal	Broche du FPGA	Description	Affectation
HEX0 [0]	PIN_J2	Segment 0 [0]	Sa
HEX0 [1]	PIN_J1	Segment 1 [1]	Sb
HEX0 [2]	PIN_H2	Segment 2 [2]	Sc
HEX0 [3]	PIN_H1	Segment 3 [3]	Sd
HEX0 [4]	PIN_F2	Segment 4 [4]	Se
HEX0 [5]	PIN_F1	Segment 5 [5]	Sf
HEX0 [6]	PIN_E2	Segment 6 [6]	Sg

Afficheur HEX1

Nom du signal	Broche du FPGA	Description	Affectation
HEX1 [0]	PIN_E1	Segment 0 [0]	
HEX1 [1]	PIN_H6	Segment 1 [1]	
HEX1 [2]	PIN_H5	Segment 2 [2]	
HEX1 [3]	PIN_H4	Segment 3 [3]	
HEX1 [4]	PIN_G3	Segment 4 [4]	
HEX1 [5]	PIN_D2	Segment 5 [5]	
HEX1 [6]	PIN_D1	Segment 6 [6]	

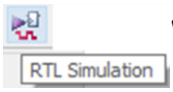
Afficheur HEX2

Nom du signal	Broche du FPGA	Description	Affectation
HEX2 [0]	PIN_G5	Segment 0 [0]	
HEX2 [1]	PIN_G6	Segment 1 [1]	
HEX2 [2]	PIN_C2	Segment 2 [2]	
HEX2 [3]	PIN_C1	Segment 3 [3]	
HEX2 [4]	PIN_E3	Segment 4 [4]	
HEX2 [5]	PIN_E4	Segment 5 [5]	
HEX2 [6]	PIN_D3	Segment 6 [6]	

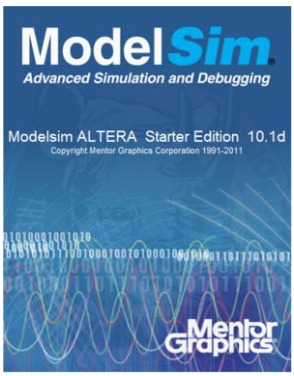
Afficheur HEX3

Nom du signal	Broche du FPGA	Description	Affectation
HEX3 [0]	PIN_F4	Segment 0 [0]	
HEX3 [1]	PIN_D5	Segment 1 [1]	
HEX3 [2]	PIN_D6	Segment 2 [2]	
HEX3 [3]	PIN_J4	Segment 3 [3]	
HEX3 [4]	PIN_L8	Segment 4 [4]	
HEX3 [5]	PIN_F3	Segment 5 [5]	
HEX3 [6]	PIN_D4	Segment 6 [6]	

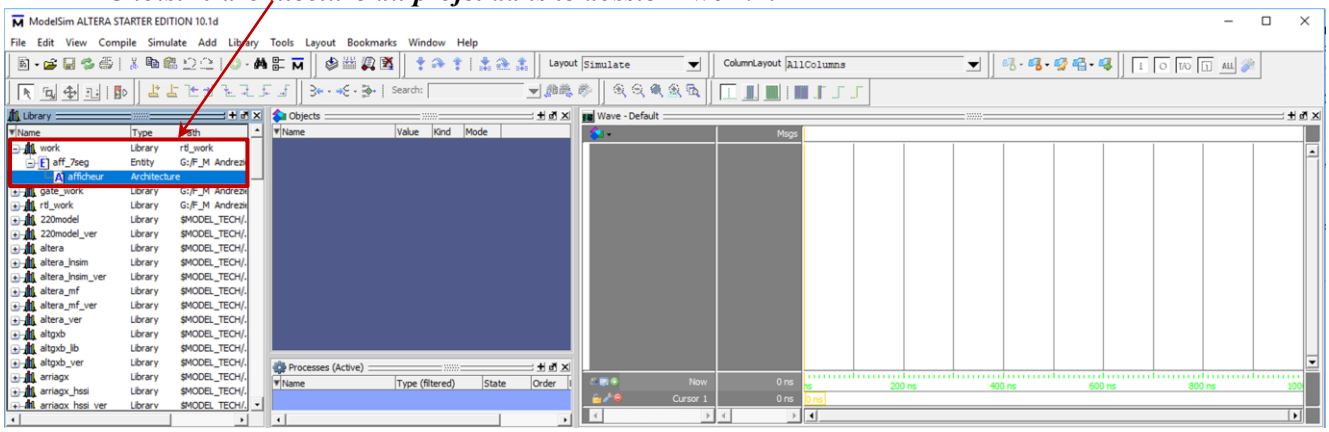
SIMULATION.

☺ Lancer le module de simulation en cliquant sur l'icône 

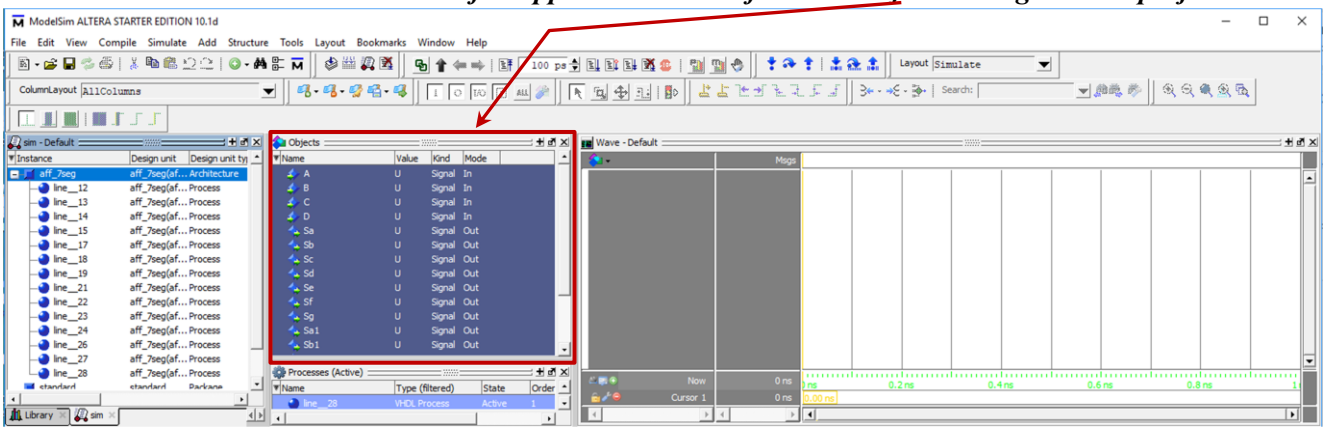
"RTL Simulation" dans la barre d'outils **QUARTUS**.



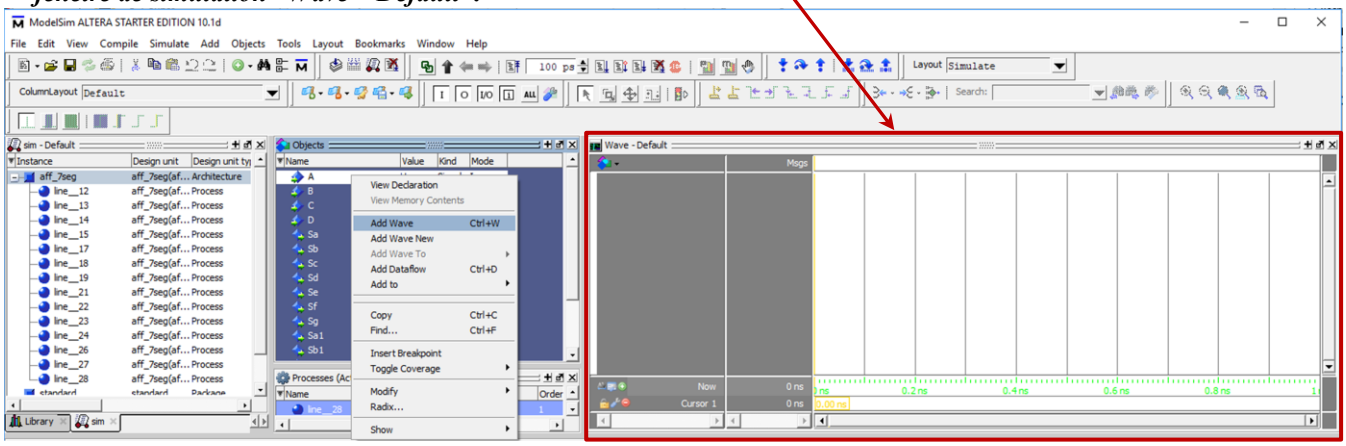
☞ ***Choisir l'architecture du projet dans le dossier "work".***



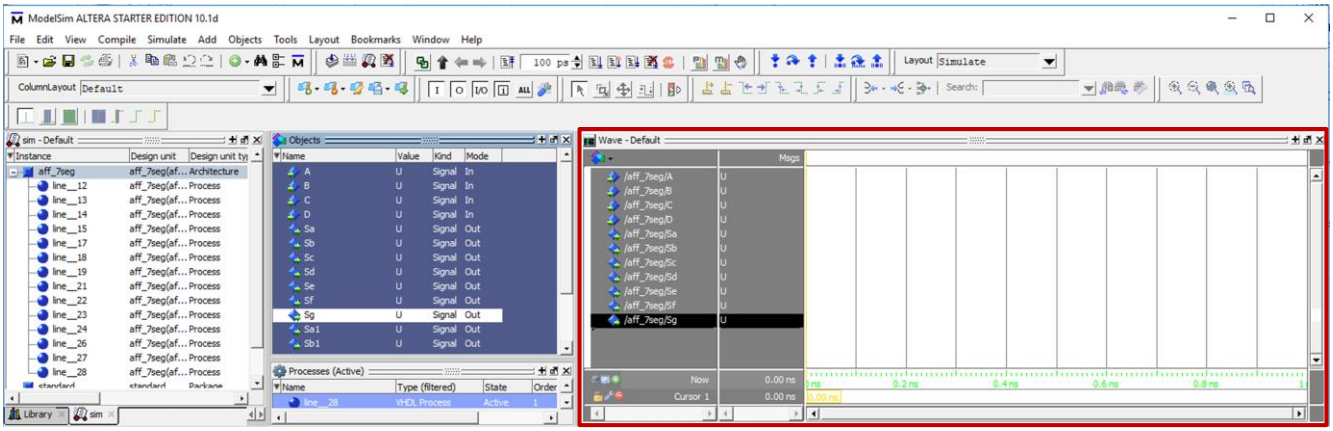
… Un double clic sur "Architecture" fait apparaître dans la fenêtre "Objets" les signaux du projet.



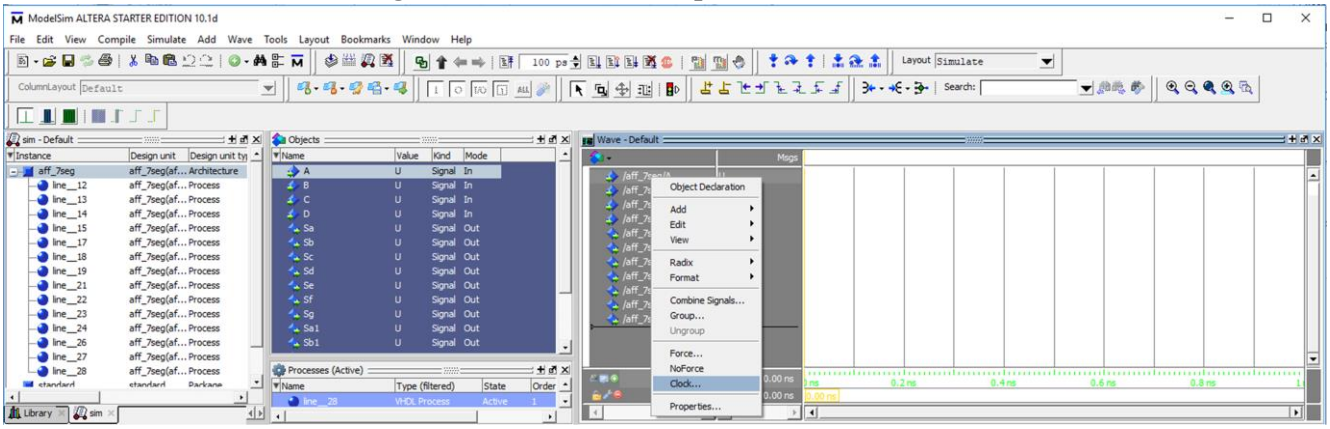
… Faire un clic droit sur chacun des signaux et choisir "Add Wave". Les signaux sont ainsi transférés dans la fenêtre de simulation "Wave – Default".



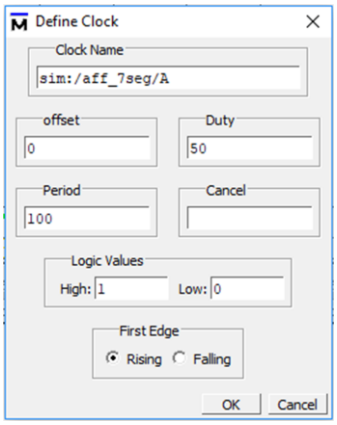
➢ **Paramétrage des signaux "A", "B", "C" et "D".**

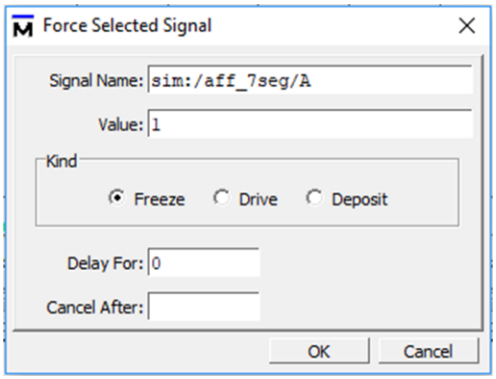


☞ ***Faire un clic droit sur un signal et choisir l'une des options "Force…", "NoForce" ou "Clock…".***

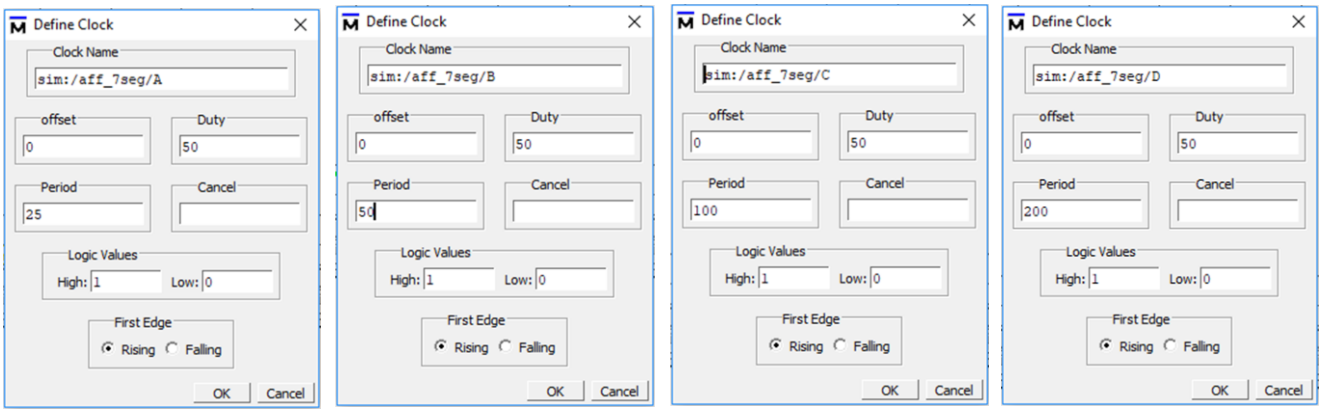


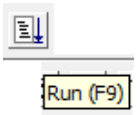
➢ Chacune des options **"Clock…"** et **"Force…"** sera paramétrée en fonction du résultat souhaité de la simulation.



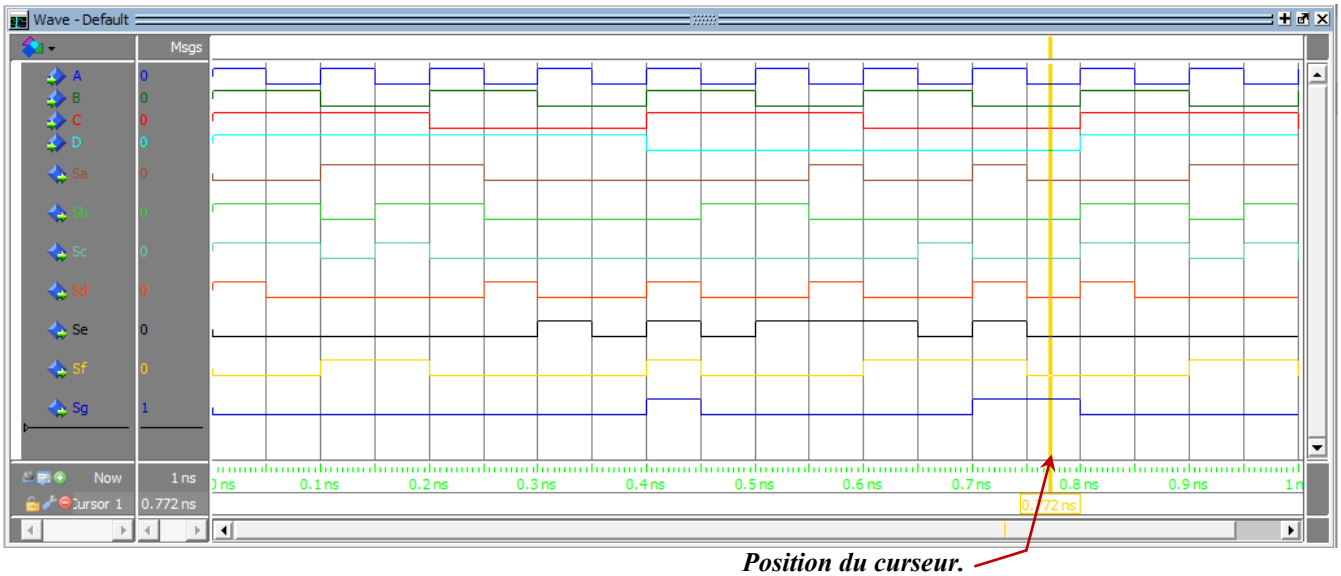


☞ Pour les signaux **A**, **B**, **C** et **D** du décodeur BCD-7Segments, on choisit l'option **"Clock"** avec le paramétrage suivant :



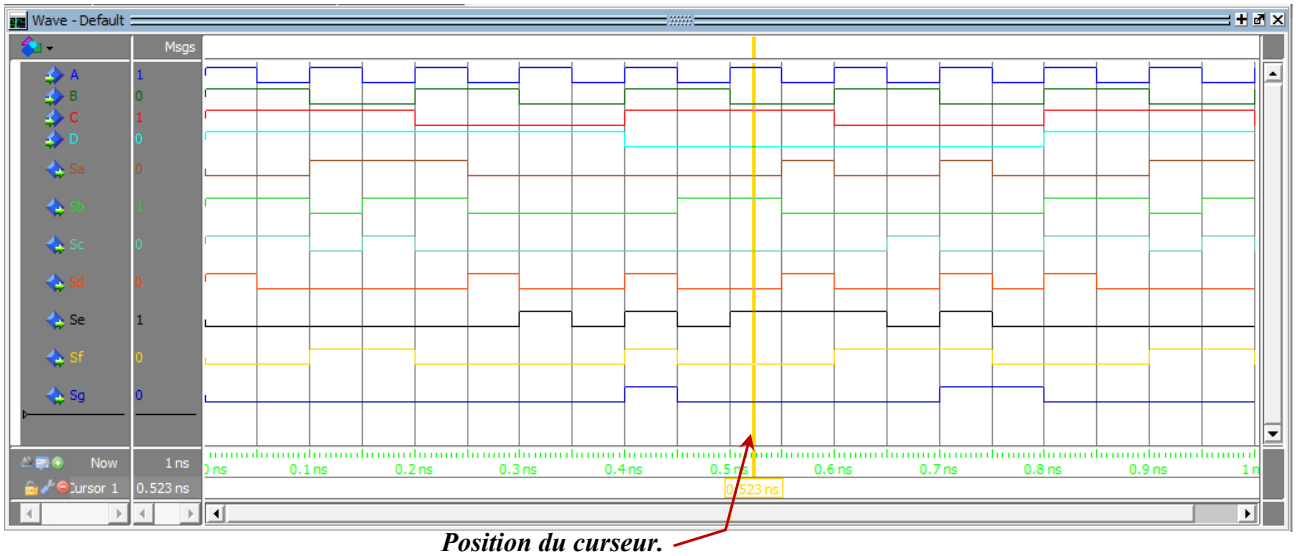
Lancer la simulation en cliquant successivement sur l'icône "Run". 

Résultats.



Position du curseur.

➢ Le positionnement du curseur montre que pour **A = 0** ; **B = 0** ; **C = 0** et **D = 0**, on a **Sa = 0** ; **Sb = 0** ; **Sc = 0** ; **Sd = 0** ; **Se = 0** ; **Sf = 0** ; **Sg = 1**. Seul le segment **Sg est éteint**. On a bien l'affichage du **"0".**



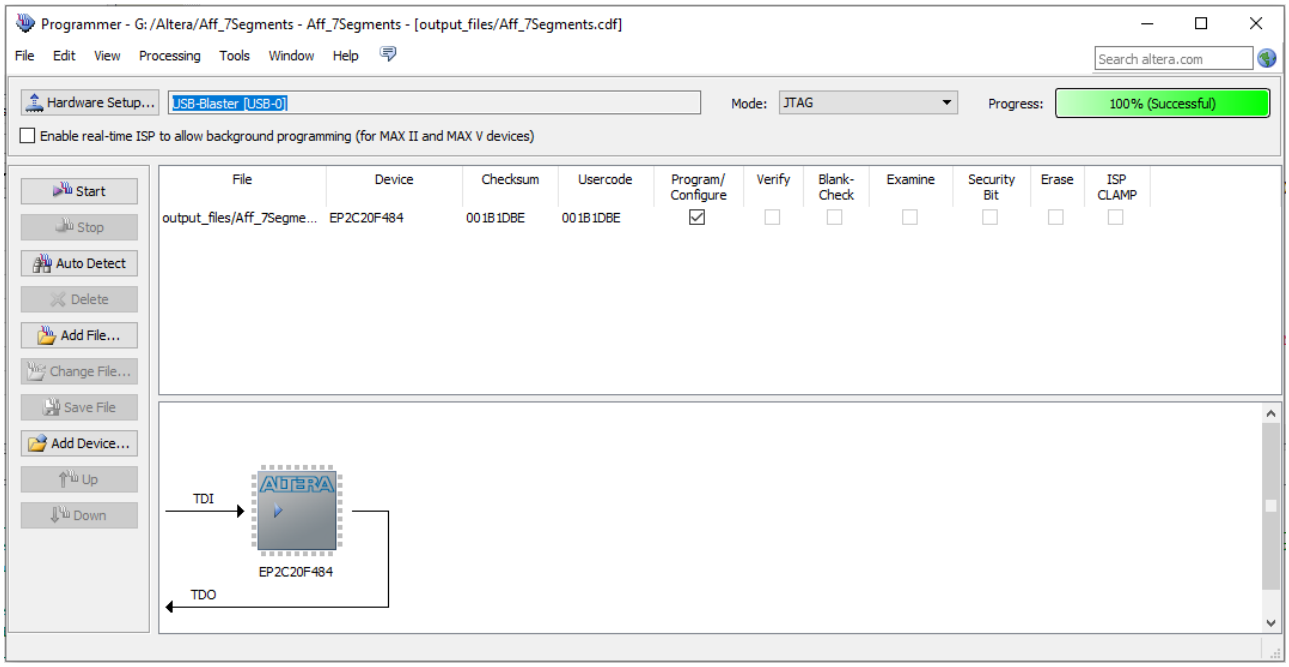
Position du curseur.

➢ Le positionnement du curseur montre que pour **A = 1** ; **B = 0** ; **C = 1** et **D = 0**, on a **Sa = 0** ; **Sb = 1** ; **Sc = 0** ; **Sd = 0** ; **Se = 1** ; **Sf = 0** ; **Sg = 1**. Les segments **Sb et Se sont éteints**. On a bien l'affichage du **"5".**

V- PROGRAMMATION DU CIRCUIT FPGA – FONCTIONNEMENT.

La programmation du circuit FPGA **Cyclone II : EP2C20F484C7** de la carte de développement **DE1 ALTERA** se fait par un clic sur l'icône **"Opens a Programmer window"** de la barre d'icônes du logiciel **Quartus II**.

Après avoir cliqué sur le bouton **"Start"** Start de la fenêtre **"Programmer"**, le fichier est chargé dans le circuit FPGA avec le message **"100% (Successful)"** pour un chargement sans échec !.

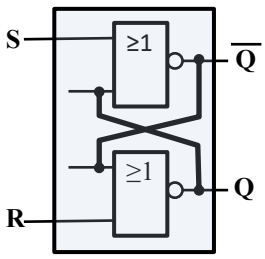


✍ En agissant sur les interrupteurs **SW0, SW1, SW2** et **SW3** de la carte de développement, vérifiez l'affichage des chiffres hexadécimaux sur **l'afficheur HEX0** de la carte de développement Altera.

Interrupteurs **SWx**

B- PROGRAMMATION EN LANGAGE VHDL DES BASCULES RS, D et JK.

1- LA BASCULE R-S : Bascule asynchrone.



➢ Équations des sorties Q et $\overline{Q}$ en fonction des entrées S et R.

- $Q = \overline{R + \overline{Q}}$ (1)
- $\overline{Q} = \overline{S + Q}$ (2)
- (2) dans (1) : $Q = \overline{R + \overline{S + Q}} = \overline{R} \cdot (S + Q)$
- (1) dans (2) : $\overline{Q} = \overline{S + \overline{R + \overline{Q}}} = \overline{S} \cdot (R + \overline{Q})$

Table de vérité.

S	R	Q	$\overline{Q}$	Fonction
0	0	Q	$\overline{Q}$	Mémoire
0	1	0	1	Mise à 0
1	0	1	0	Mise à 1
1	1	0	0	Interdit

☺ ***Ci-dessous les broches du circuit FPGA EP2C20F484C7 attribuées aux interrupteurs, aux boutons poussoirs et aux diodes leds.***

Interrupteurs

Nom du signal	Broche du FPGA	Description
SW[0]	PIN_L22	Commutateur [0]
SW[1]	PIN_L21	Commutateur [1]
SW[2]	PIN_M22	Commutateur [2]
SW[3]	PIN_V12	Commutateur [3]
SW[4]	PIN_W12	Commutateur [4]
SW[5]	PIN_U12	Commutateur [5]
SW[6]	PIN_U11	Commutateur [6]
SW[7]	PIN_M2	Commutateur [7]
SW[8]	PIN_M1	Commutateur [8]
SW[9]	PIN_L2	Commutateur [9]

Boutons poussoirs

Nom du signal	Broche du FPGA	Description
KEY[0]	PIN_R22	Poussoir [0]
KEY[1]	PIN_R21	Poussoir [1]
KEY[2]	PIN_T22	Poussoir [2]
KEY[3]	PIN_T21	Poussoir [3]

Leds rouges

Nom du signal	Broche du FPGA	Description
LEDR[0]	PIN_R20	LED Rouge[0]
LEDR[1]	PIN_R19	LED Rouge[1]
LEDR[2]	PIN_U19	LED Rouge[2]
LEDR[3]	PIN_Y19	LED Rouge[3]
LEDR[4]	PIN_T18	LED Rouge[4]
LEDR[5]	PIN_V19	LED Rouge[5]
LEDR[6]	PIN_Y18	LED Rouge[6]
LEDR[7]	PIN_U18	LED Rouge[7]
LEDR[8]	PIN_R18	LED Rouge[8]
LEDR[9]	PIN_R17	LED Rouge[9]

Leds vertes

Nom du signal	Broche du FPGA	Description
LEDG[0]	PIN_U22	LED Verte[0]
LEDG[1]	PIN_U21	LED Verte[1]
LEDG[2]	PIN_V22	LED Verte[2]
LEDG[3]	PIN_V21	LED Verte[3]
LEDG[4]	PIN_W22	LED Verte[4]
LEDG[5]	PIN_W21	LED Verte[5]
LEDG[6]	PIN_Y22	LED Verte[6]
LEDG[7]	PIN_Y21	LED Verte[7]

☞ **Affectation des entrées-sorties de la bascule RS.**

Les entrées R (reset) et S (set) de la bascule.			
SW[0]	PIN_L22	Interrupteur **SW0**	R (Reset)
SW[1]	PIN_L21	Interrupteur **SW1**	S (Set)

Les sorties Q et Qn de la bascule.			
LEDR[0]	PIN_R20	Led **LEDR0**	Sortie Q
LEDR[1]	PIN_R19	Led **LEDR1**	Sortie inversée Qn

A Faire.

a) Donner le programme VHDL de la bascule RS.

b) Réaliser la simulation et donner le résultat de la simulation. Commenter ce résultat.

c) Programmer la bascule RS simulée sur le circuit **FPGA EP2C20F485C7** et vérifier le fonctionnement.

d) Conclure.

Remarque. Les deux exercices réalisés ci-dessus (**Décodeur** d'un afficheur 7-segments et **Bascule asynchrone R-S**) utilisent des **structures combinatoires.** Avec les équations logiques des sorties de ces circuits, on programme la structure combinatoire étudiée.

Les deux exercices (Bascule D et Bascule JK) qui suivent utilisent des **structures séquentielles** où la présence d'un signal d'horloge rythme les différentes opérations de la structure. On parle de **systèmes synchrones**.

PROGRAMMATION EN VHDL D'UNE STRUCTURE SÉQUENTIELLE.

☺ Dans la description de l'architecture, on utilise souvent le mot clé **"process"**. Il s'agit des différentes tâches d'un programmes VHDL. Ces tâches s'exécutent en parallèle et sont appelées des **processus**.

☺ **Règle de fonctionnement d'un processus :**

1- Un processus est une boucle infinie, lorsqu'il arrive à la fin du code, il reprend automatiquement au début.

2- Un processus doit être sensible des points d'arrêt de façon à le synchroniser. La synchronisation est indiquée par un point d'arrêt. Il existe deux types de points d'arrêts :

- Le processus est associé à une **"liste de sensibilité"** qui contient une liste de signaux qui réveille le processus lors d'un changement d'état d'un des signaux. Sa syntaxe est "**process (liste de signaux)**"
- Le processus a des instructions d'arrêt **"wait"** dans sa description interne. Le wait est sensible soit à un signal soit à un temps physique.

3- Les variables sont internes au processus et son affectées immédiatement, contrairement aux signaux qui eux ne sont pas affectés directement mais par le biais de leur échéancier qui est mis à jour en fin de processus avec la nouvelle valeur et le temps d'affectation qui correspond à un delta-cycle après le signal ayant réveillé le processus.

☞ L'instruction **"wait"** permet de mettre des points d'arrêt dans le corps du processus.

Syntaxe : wait [**S1**, **S2**, …] [**until CONDITION**] [**for DUREE**];

S1 et **S2** sont des signaux, **CONDITION** est une expression générant un booléen et **DUREE** est le temps physique d'attente.

Les signaux d'horloge disponibles.

Nom du signal	Broche du FPGA	Description
CLOCK_27	PIN_D12 ; PIN_E12	Horloge 27 MHz
CLOCK_50	PIN_L1	Horloge 50 MHz
CLOCK_24	PIN_A12 ; PIN_B12	Horloge 24 MHz
EXT_CLOCK	PIN_M21	Entrée horloge externe

2- LA BASCULE D : Bascule synchrone avec entrée de validation (EN) et de mise à "0" ($\overline{RST}$).

➢ **Symbole et table de vérité.**

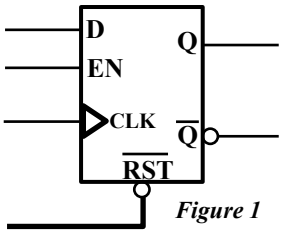
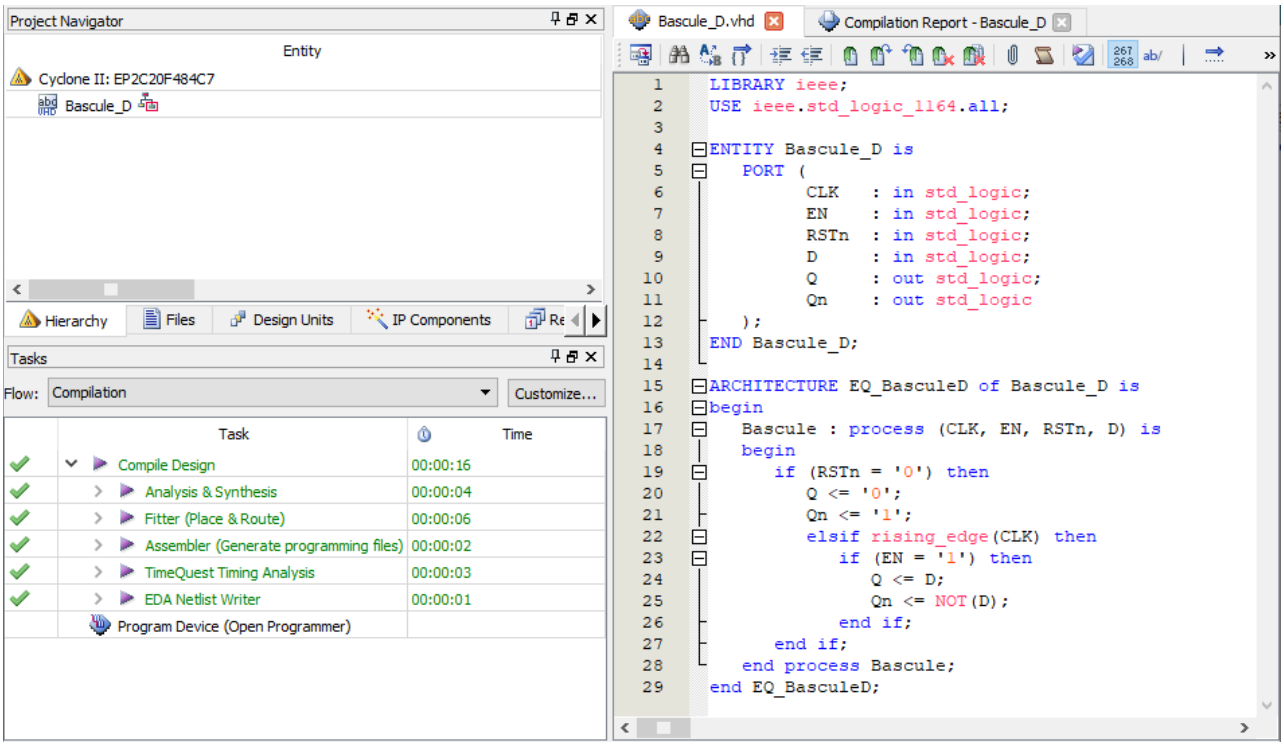


Figure 1

Table de vérité					
CLK	EN	$\overline{RST}$	D	Q	$\overline{Q}$
X	0	0	X	0	1
X	1	0	X	0	1
X	0	1	X	Q_n	$\overline{Q_n}$
↑	1	1	0	0	1
↑	1	1	1	1	0

Équation de la sortie Q : (EN = 1 ; $\overline{RST}$ = 1 et au front montant de CLK), Q = D ; $\overline{Q} = \overline{D}$

Programme VHDL.



☞ **Affectation des entrées-sorties de la bascule D ci-dessus.**

Les entrées **D** (Donnée) ; **EN** (Validation) et **CLK** (Horloge) de la bascule.			
SW[0]	PIN_L22	Interrupteur **SW0**	D (Donnée)
SW[1]	PIN_L21	Interrupteur **SW1**	EN (Validation)
SW[2]	PIN_M22	Interrupteur **SW2**	$\overline{RST}$ (Entrée de forçage à "0")
CLOCK_27	PIN_D12	Horloge 27 MHz	CLK (Horloge)

Les sorties Q et Qn de la bascule.			
LEDR[0]	PIN_R20	Commande diode led **L1**	Sortie Q
LEDR[1]	PIN_R19	Commande diode led **L2**	Sortie inversée Qn

A Faire.

a) Saisir le programme VHDL de la bascule D de la figure 1 ci-dessus.

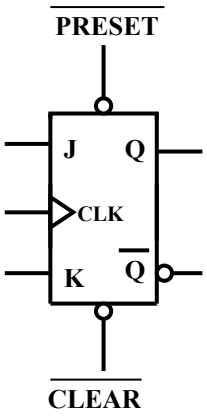
b) Réaliser la simulation et donner le résultat de la simulation. Commenter ce résultat.

c) Programmer la bascule D simulée sur le circuit **FPGA EP2C20F485C7** et vérifier le fonctionnement.

d) Conclure.

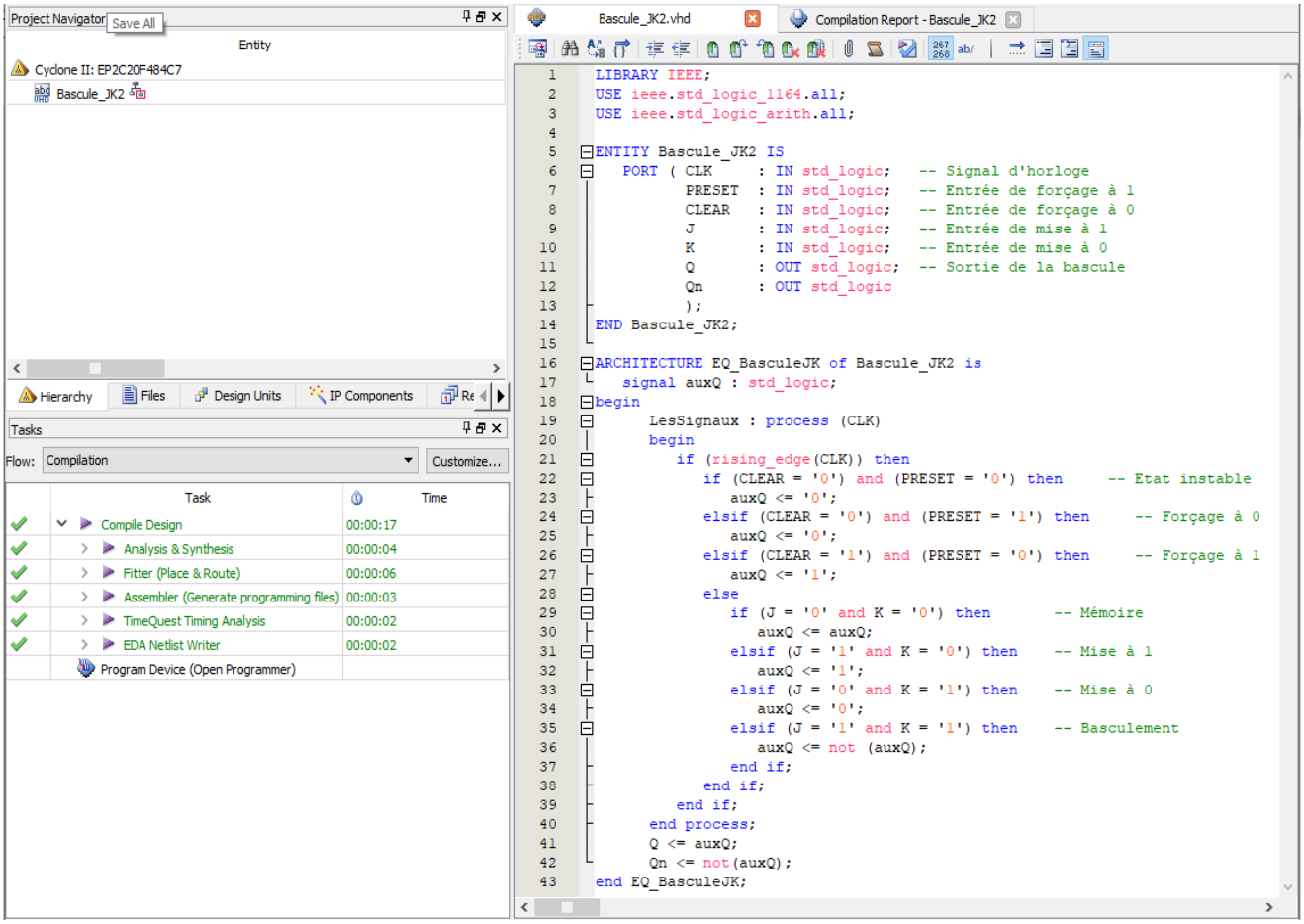
3- **La bascule J-K :** Bascule synchrone avec entrée de **mise à "1"** ($\overline{\text{PRESET}}$) et de **mise à "0"** ($\overline{\text{CLEAR}}$).

➢ **Symbole et table de vérité.**



CLK	J	K	$\overline{\text{PRESET}}$	$\overline{\text{CLEAR}}$	Q	$\overline{Q}$
X	X	X	0	1	1	0
X	X	X	1	0	0	1
X	X	X	0	0	Instable	
H	X	X	1	1	Q	$\overline{Q}$
L	X	X	1	1	Q	$\overline{Q}$
↑	0	0	1	1	Q	$\overline{Q}$
↑	1	0	1	1	1	0
↑	0	1	1	1	0	1
↑	1	1	1	1	$\overline{Q}$	Q

➢ **Programme VHDL.**



☞ **Affectation des entrées-sorties de la bascule JK ci-dessus.**

Les entrées **D** (Donnée) ; **EN** (Validation) et **CLK** (Horloge) de la bascule.			
SW[0]	PIN_L22	Interrupteur **SW0**	J (Mise à "1")
SW[1]	PIN_L21	Interrupteur **SW1**	K (Mise à "0")
SW[2]	PIN_M22	Interrupteur **SW2**	$\overline{\text{PRESET}}$ (Entrée de forçage à "1")
SW[3]	PIN_V12	Interrupteur **SW3**	$\overline{\text{CLEAR}}$ (Entrée de forçage à "0")
CLOCK_27	PIN_D12	Horloge 27 MHz	CLK (Horloge)

Les sorties Q et Qn de la bascule.			
LEDR[0]	PIN_R20	Commande diode led **L1**	Sortie Q
LEDR[1]	PIN_R19	Commande diode led **L2**	Sortie inversée Qn

A Faire.

a) Saisir le programme VHDL de la bascule JK.

b) Réaliser la simulation et donner le résultat de la simulation. Commenter ce résultat.

c) Programmer la bascule JK simulée sur le circuit **FPGA EP2C20F485C7** et vérifier le fonctionnement.

d) Conclure.

Exemple : ***Résultats de la simulation de la bascule JK programmée ci-dessus.***