# TP 3 : Additionneur décimal et bascules

**Pour ce TP, il est demandé d'utiliser le projet du TP 2 pour la partie I et de créer un autre projet Quartus pour la partie II**

**Partie I : Additionneur décimal**
**Partie II : les bascules**

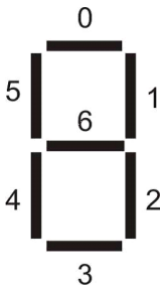
## Partie I : Additionneur décimal sur DE1-SoC

**Un petit rappel sur la description du projet à réaliser :**

On souhaite additionner deux nombres A et B codés sur 4 bits. (Les valeurs de A ou B sont comprises entre 0 et 15). Ces nombres seront sélectionnés sur la carte DE1-Soc à partir des switch **SW[3..0]** et **SW[7..4]** et seront affichés sur les afficheurs 7 segments de la carte DE1-SoC. Une retenue sera prise en compte et l'addition sera effectuée en **Décimal Codé Binaire**, c'est-à-dire que le résultat et les deux nombres de l'addition seront affichés en nombre décimal (et non hexadécimal) sur 2 afficheurs 7 segments. (1 pour les unités, le deuxième pour les dizaines)

### 1) Création d'un transcodeur hexadécimal 7 segments :

- **Décodeur 7 segments**


Créer en VHDL un transcodeur hexadécimal 7 segments permettant d'afficher un chiffre hexadécimal jusque F

Infos :

Le décodeur accepte en entrée un Std_logic_vector sur 4 bits :
**A: in std_logic_vector (3 downto 0);**

Le décodeur génère en sortie un Std_logic_vector sur 7 bits :
**VSeg: out std_logic_vector (6 downto 0)**

L'entité sera nommée *dec_7seg.*

A savoir que les afficheurs utilisés sont de type anodes communes :
**L'allumage des segments est assuré par un état logique '0'.**

Pour le développement VHDL de ce composant, plusieurs solutions VHDL sont possibles.

**En utilisant un process :**

Utilisation de l'instruction 'if ':

Ex :

```
PROCESS(a)
BEGIN
IF (a="0000" ) THEN
AFF <= "0111111";
ELSIF (a="0001" ) THEN
AFF <= "0000110";
...
ELSE
AFF <= "1111001;
END IF;
```

Utilisation de l'instruction 'Case' :

Ex:

```
PROCESS(a)
BEGIN
CASE a IS
WHEN "0000" => AFF <= "0111111";
...
...
WHEN OTHERS => AFF <= "1111001";
END CASE;
END PROCESS ;
```

**Sans utilisation de process :**

Utilisation de l'instruction 'With , When' :

Ex :

```
WITH a SELECT
AFF<="0111111" WHEN "0000",
 "0000110" WHEN "0001",
...
"1111001" WHEN Others;
```

Pour écrire le code VHDL, utiliser un fichier vhdl sur quartus et enregistrer le fichier dans le répertoire : TP2>AddBCD

- Simuler l'entité avec modelsim

**Simulation de l'entité avec modelsim** :

- Placer également dans ce répertoire le fichier Test Bench nommé : TB_dec_7Seg.vhd
- Refaire un nouveau projet avec Modelsim (comme pour la partie 2 du TP2) en choisissant le même répertoire que précédemment et en ajoutant au projet les deux fichiers .vhd.
- Faire une simulation du composant avec modelsim et vérifier si le signal OK_TB reste à 'True' pour toute la simulation.

## 2) Application

- Ajouter au projet Quartus les deux fichiers VHDL :
  Add_BCD.vhd
  DEC_7seg.vhd

- Recompiler sous Quartus et en faire deux composants Symbol graphiques.

- Implémenter sur la carte DE1-SoC le composant Dec_7seg avec votre choix d'entrées et de sorties.

On veut ensuite réaliser une application sur la carte DE1-SoC sous forme graphique, permettant de faire la somme de deux nombres décimaux compris entre 0 et 15.

Le projet global sera nommé top_ADD_BCD.

- Le premier nombre sera composé à l'aide des interrupteurs SW[0] à SW[3] et affiché en valeur décimale sur HEX0 pour les unités et HEX1 pour les dizaines.

- Le deuxième nombre sera composé à l'aide des interrupteurs SW[4] à SW[7] et affiché sur HEX2 pour les unités et HEX3 pour les dizaines.

- Le résultat sera affiché sur HEX4 (dizaine) et HEX5 (unités)

Faire une simulation du projet puis implémenter le projet sur la carte DE1-SoC

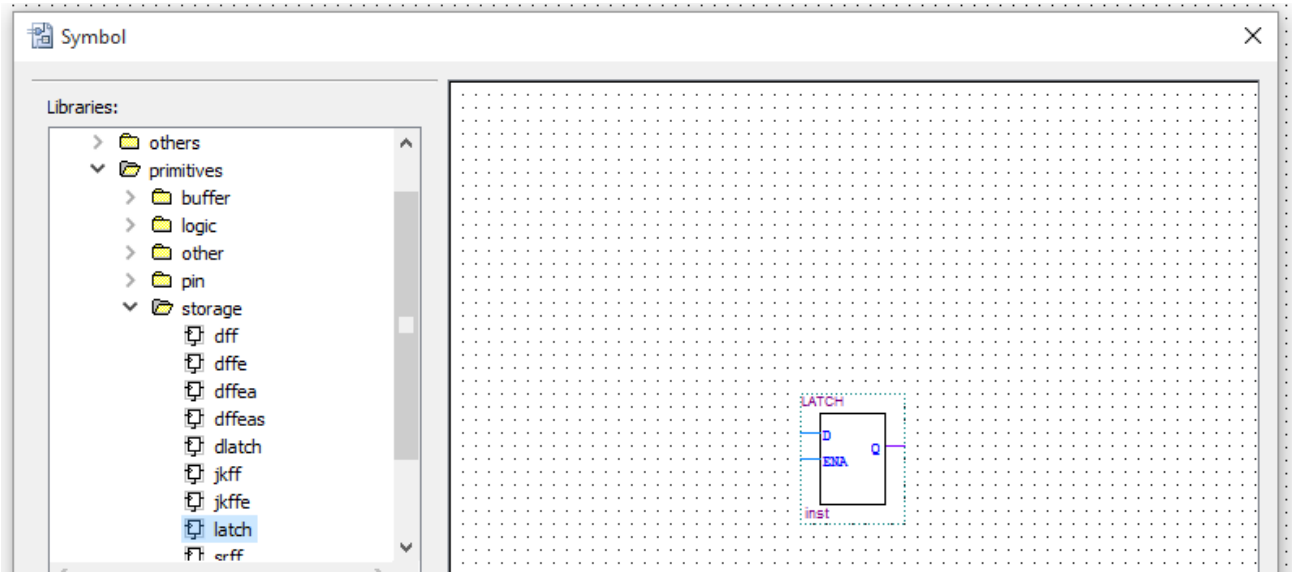
# Partie II : les bascules

Les bascules sont des éléments bistables qui peuvent prendre l'état 0 ou l'état 1, et le conserver. En cela, elles constituent une mémoire élémentaire et offrent ainsi la possibilité de développer des systèmes complexes nécessitant la mise en œuvre de registres et de compteurs.

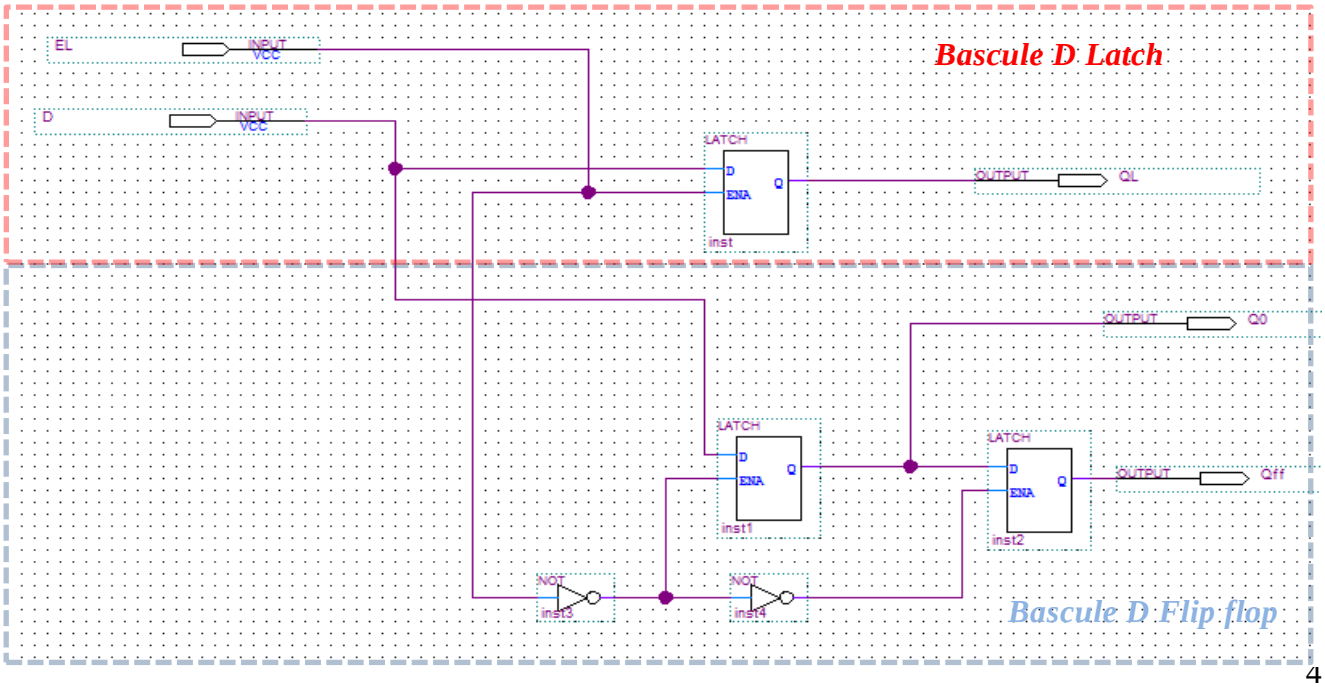
Nous réalisons quelques tests sur les bascules les plus utilisées comme la bascule D-Latch (verrou), les bascules D-flip-flop, T-flip-flop ainsi que la bascule JK.

## 1) Le verrou D-Latch, D flip-flop

Pour tester cette bascule, vous pouvez insérer dans un block diagram Schematic la bascule se trouvant dans : Primitives>storage> Latch




Puis réaliser le schéma suivant :

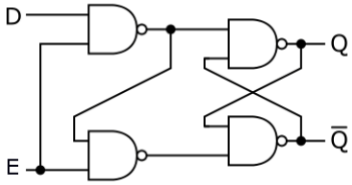


- Simuler ce circuit. Donner la table de vérité de ces deux bascules. Expliquer le fonctionnement de chacune de deux bascules

**Bascule D-latch :**

La bascule D-latch permet de garder une valeur en mémoire ou de recopier l'état en entrée D (Data) en fonction d'un signal de validation E (enable). Grâce à ce signal de validation, cette bascule constitue un premier pas vers une synchronisation des points mémoires.

Le schéma logique de la bascule D-latch est donné ci-après :

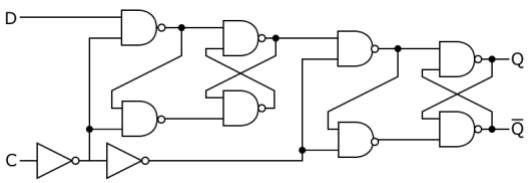



Voici le code VHDL correspondant à cette bascule :

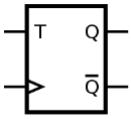
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY test_D_latch IS
        PORT
        (
                D :  IN  STD_LOGIC;
                En :  IN  STD_LOGIC;
                Q :  OUT  STD_LOGIC
        );
END test_D_latch;

ARCHITECTURE ARCH OF test_D_latch IS
BEGIN
PROCESS(En,D)
BEGIN
        IF (En = '1') THEN
        Q <= D;
        END IF;
END PROCESS;
END ARCH;
```

**Bascule D-flip flop**

La bascule D-flip flop recopie l'état courant d'entrée D sur la sortie Q au front montant d'horloge: il s'agit d'une bascule synchrone impliquant un délai entre l'entrée et la sortie du circuit. La bascule D-flip flop se construit par la mise en cascade de deux bascules D (maître et esclave).



## 2) La toggle flip flop



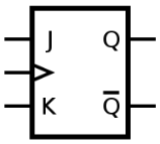

La toggle flip-flop (TFF) est une bascule changeant d'état à chaque coup d'horloge lorsque son entrée T est à l'état haut. Sinon l'état précédent est conservé. Insérer le symbole TFF de la bibliothèque Quartus, simuler son comportement et donner sa table de vérité.

Contrairement à la figure ci-dessus, la bascule TFF de Quartus possède les signaux d'entrées preset (PRN) et reset (CLRN). Etudier l'influence de ces deux signaux d'entrées sur les sorties de la bascule.

Voici le code VHDL correspondant cette bascule :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY test_Tog_flipflop IS
        PORT
        (
                Pst :  IN  STD_LOGIC;
                Rst :  IN  STD_LOGIC;
                T :  IN  STD_LOGIC;
                Clk :  IN  STD_LOGIC;
                Q :  OUT  STD_LOGIC
        );
END test_Tog_flipflop;

ARCHITECTURE Arch OF test_Tog_flipflop IS
BEGIN

PROCESS(Clk,Rst,Pst)
VARIABLE Q_s : STD_LOGIC;
BEGIN
IF (Rst = '0') THEN
        Q_s := '0';
ELSIF (Pst = '0') THEN
        Q_s := '1';
ELSIF (RISING_EDGE(Clk)) THEN
        Q_s := Q_s XOR T;
END IF;
        Q <= Q_s;
END PROCESS;


END Arch;
```

## 3) La JK flip flop




La bascule JK est une version synchrone de la bascule RS. Lorsque J=K, elle peut être assimilée à une bascule TFF. Tout comme pour la D flip-flop et la TFF, l'activité de cette bascule est conditionnée par les fronts montants ou descendants du signal d'horloge. Insérer le symbole JK de la bibliothèque Quartus, simuler son comportement et donner sa table de vérité.

# Fin du TP 3