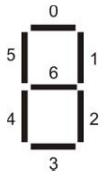
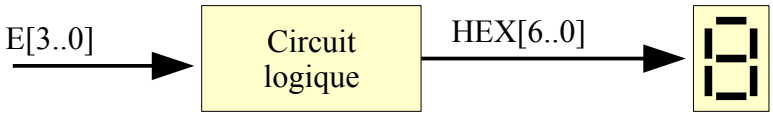
TP 2 – Synthèse de systèmes numériques, VHDL

1. Gestion de l'afficheur : transcodeur binaire/7segments

Les platines de TP disposent d'afficheurs 7 segments. Afin de pouvoir visualiser des valeurs binaires, on souhaite disposer d'un transcodeur permettant d'afficher en les digits décimaux correspondant la valeur d'un mot de 4 bits.
Ci-dessous un afficheur 7 segments standard :



A Chaque segment correspond une led qui peut être allumée ou éteinte. Il s'agit ici d'allumer les leds permettant de représenter graphiquement un nombre décimal en fonction du nombre présent en entrée :



Vous devez ici créez un composant VHDL permettant d'effectuer le transcodage d'une valeur binaire 4 bits en un digit décimal. Si le nombre en entrée est supérieur à 9, l'afficheur devra afficher les digits hexadécimaux A, b, C, d, E, F.
Attention, une led est allumée lorsqu'une valeur logique 0 lui est appliquée.

Ouvrez un nouveau projet sous Quartus II, créez un nouveau fichier d'entrée qui contrairement au TP précédent sera un simple fichier texte. Appeler ce nouveau fichier *afficheur.vhd*.

Dans ce fichier texte donnez la description d'une entity appelée *afficheur*, comprenant un vecteur d'entrée sur 4 bits et un vecteur de sortie sur 7 bits. Il est à noter que l'entity doit avoir le même nom que le fichier. Donner la description de l'architecture dans le même fichier.

Pour tester la syntaxe effectuez une vérification (« processing → Analyse current file »).

Une fois votre description réalisée, vous pouvez la tester sur les platines de TP en affectant l'entrée E aux pattes du FPGA reliées à des Switch (SW) et les sorties HEX aux pattes du FPGA reliées à un des afficheurs.

Sélectionnez le composant correspondant à votre platine, sans oublier de spécifier que les « unused pins » doivent être mises en « input tri-stated » pour ne pas endommager la carte, puis affectez vos entrées/sorties.
Les connexions du FPGA aux composants présents sur la carte utilisée en TP ainsi que les mode sélectriques sont donnés dans le « user manual » de la carte.

Vous pouvez également utiliser le fichier de caractérisation de la carte et le modifier : « DE10_LITE_Golden_Top.qsf » disponible sur moodle.
Dans ce fichier vous n'avez qu'à modifier les noms des 'pins' pour qu'elles correspondent aux noms que vous avez donnés à vos E/S dans votre projet (ou l'inverse,). Ensuite vous importez le fichier : cliquez sur « import assignments » et sélectionnez votre fichier.

Compilez et programmez la carte.
Testez quelques valeurs sur les interrupteurs.

2. Affichage Hexadécimal d'un entier signé sur 8 bits sous forme signe + grandeur

Il s'agit dans cette partie d'afficher sur les afficheurs 7 segments un nombre signé sur 8 bit. Le premier afficheur affichera soit rien si le nombre est positif, soit "–" s'il est négatif, 2 autres afficheurs seront utilisés pour afficher la grandeur sous forme hexadécimale.

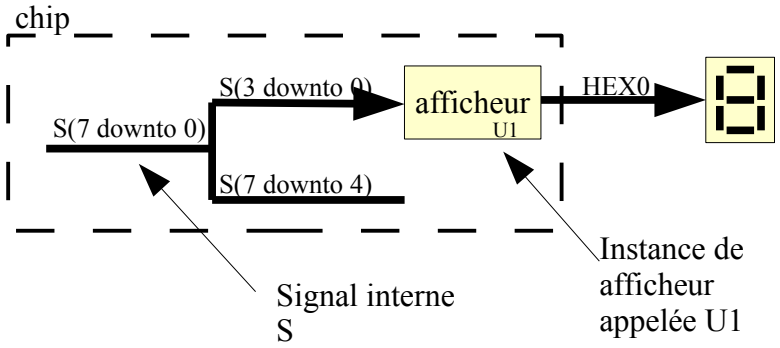
Un bloc combinatoire sera placé en amont des décodeurs/afficheurs afin de transformer le nombre entier 8 bits en un nombre « signe + grandeur ». La description de ce bloc combinatoire peut se faire soit dans une entity/architecture séparée soit directement dans l'architecture du module final.

Pour réutiliser de manière simple des composants déjà écrits (ici la description des décodeurs 7 segments) , il faut que les descriptions entity/architecture soient présentes dans le répertoire de travail et déclarer des « components » (déclaration identique aux entity auxquelles ils se réfèrent).

Une fois le component déclaré, il peut être utilisé dans la description de l'architecture avec un nom d'instance et l'affectation des signaux (mapping des signaux). Un exemple avec le décodeur 7 segments utilisé au 1 est donné ci-après :

```
architecture a of afficheur2 is

signal S : std_logic_vector (7 downto 0);    -- Déclaration des signaux interne

component afficheur is                -- Déclaration des components (identique à l'entity)
port (
       E : in std_logic_vector(3 downto 0);
       HEX : out std_logic_vector(6 downto 0));
end component;
(…)
begin

U1 : afficheur port map (S(3 downto 0), HEX0);
(…)
end a ;
```

Mapping graphique (correspondant à la description partielle ci-dessus) :

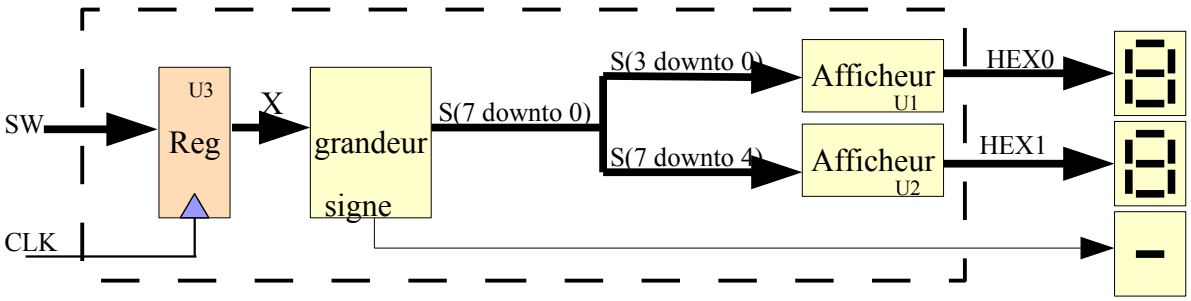


Remarques :

- Il est également possible de créer des paquetages en déclarant tous les « components » dans un fichiers séparés.
- Pour pouvoir utiliser des std_logic_vector comme des objets scalaires (permet de faire des comparaisons de grandeur, addition, ...), il est nécessaire d'utiliser le paquetage ieee.numeric_std.all et d'utiliser des type unsigned ou signed.

3. Ajout d'une mémoire

On souhaite ici ajouter une mémoire afin de mémoriser une valeur 8 bits qui sera affichée. La mémoire sera une simple mémoire mémorisant sur un front montant d'un signal de synchronisation (CLK).



1. Ecrivez l'entity/architecture de la mémoire Reg.
2. Intégrez votre nouveau « component » dans la description précédente pour créer votre nouveau composant
3. Implantez votre description sur les platines de TP et testez, l'horloge sera implémentée par un bouton poussoir (KEY)

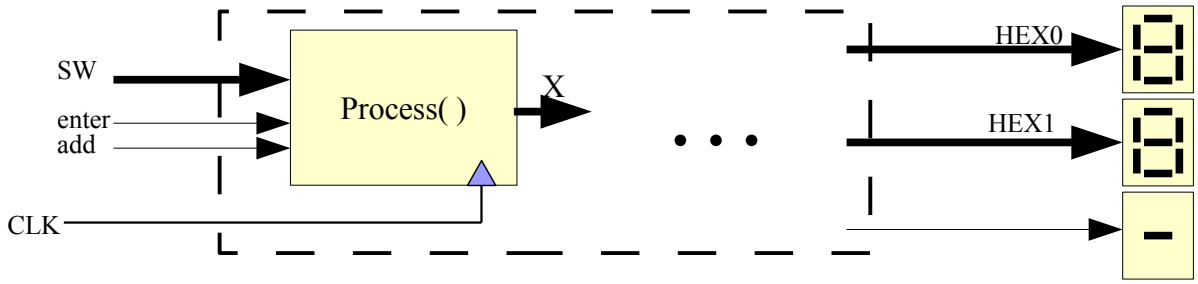
4. Additionneur

Dans cette partie, on souhaite réaliser une unité d'addition. Cette unité utilisera une seule entrée pour les nombres (8 Switchs) et une mémoire pour mémoriser un argument/résultat.
Le cahier des charges est le suivant :

1. une valeur est en mémoire et affichée sur les afficheur
2. positionnement des switches pour entrer une nouvelle
   2 cas possibles
   - appuis sur enter : la valeur présente sur les switches est mémorisée et affichée
   - appuis sur plus : la valeur mémorisée est la somme de la précédente et de la valeur présente sur les switches

   retour au 1.

Pour réaliser cet additionneur, vous devez créer un process synchronisé sur l'horloge CLK qui remplacera le registre. En utilisant des instructions if/elsif/then/else réaliser le cahier des charges.



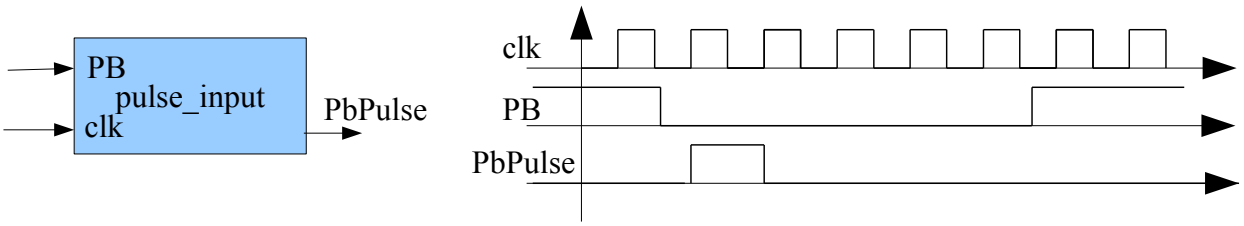
Les entrées enter et add seront matérialisées par 2 switchs et l'horloge par un bouton poussoir.

5. Synchronisation des boutons sur l'horloge

Dans cette partie, on souhaite utiliser une horloge générée par un quartz qui est disponible sur le carte. Cela permet d'éviter de générer les mémorisation à la main.
Les mémorisations sont synchronisées sur l'horloge, un appui sur un bouton poussoir n'est pas synchronisé, de plus un appui peut durer plusieurs cycles d'horloge si l'horloge est de fréquence élevée comme ici.

Pour palier à ce problème, il est possible de réaliser un circuit supplémentaire permettant de générer une impulsion unique lors d'un appui sur un bouton poussoir qui sera de plus synchronisé sur l'horloge.

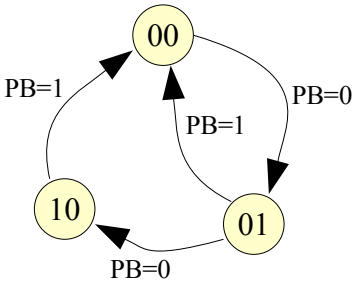
Le principe (avec un bouton qui génère un niveau bas lors de l'appui comme sur les cartes de TP) est donné par le chronogramme ci-après. L'appui qui dure plusieurs cycles d'horloge est transformé en une impulsion unique qui ne dure qu'une période d'horloge.



Pour réaliser la description du circuit, il vous est proposé d'utiliser un signal sur 2 bits permettant de mémoriser l'état du système :

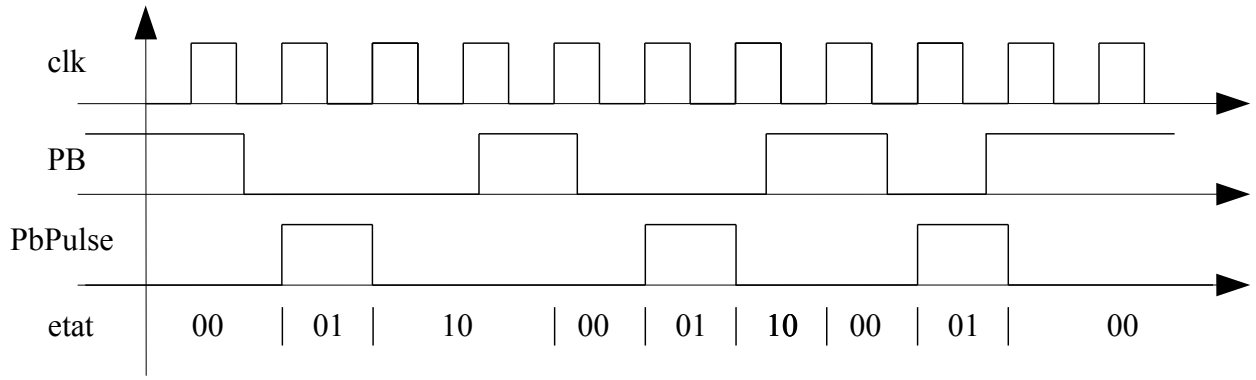
- 00 : circuit au repos, l'entrée PB a été détectée à 1. La sortie Pbpulse est à 0
- 01 : un état bas a été détecté sur le bouton poussoir PB, la sortie PbPulse est à 1
- 10 : état suivant l'état 01 si l'appui sur le bouton PB est maintenu à 0, attente qu'il repasse à 1.
  La sortie PbPulse est à 0.

Le fonctionnement peut être modélisé par le diagramme à états ci-contre :



Les changement d'états s'effectuent en fonction de l'état présent et de la valeur du signal d'entré PB. Les changements ne s'effectuant que sur front d'horloge (machine synchrone).

Le diagramme ci-dessous donne un exemple de l'évolution du signal d'état en fonction des appuis.

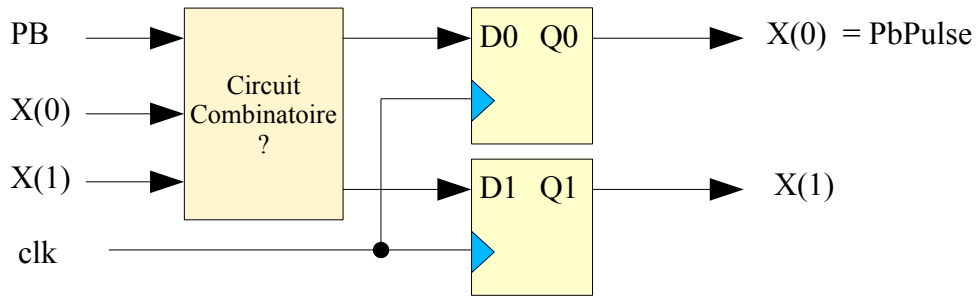


Une trame de description est donnée ci-après :

```
library ieee;
use ieee.std_logic_1164.all;

entity pulse_input is
port(
              pb : in std_logic;
              clk : in std_logic;
              pb_pulse : out std_logic);
end pulse_input;

architecture a of pulse_input is
signal x : std_logic_vector (1 downto 0);
begin

process(clk)
begin
      if rising_edge(clk) then

              (...)

      end if;
end process;

pb_pulse <= x(0);

end a;
```

Complétez la description et intégrez la à votre description globale (pour les boutons add et enter). Testez.

Synthèse logique manuelle (théorique) du circuit « pulse_input » :
Le circuit nécessite une mémoire 2 bits (signal x) qui est réalisé à l'aide de 2 bascules D synchronisées. Il s'agit alors de déterminer les équations d'entrée des mémoires (fonctions logique correspondant à votre description VHDL).



Vous donnerez les équations logiques (D0 et D1) du circuit combinatoire en entrée de la mémoire 2 bits (faire une table de vérité).