Chronomètre sur 100 secondes

Le chronomètre est divisé en 4 éléments principaux : le compteur, la mémoire, les afficheurs, le circuit de commande. A cela s'ajoute un diviseur d'horloge, permettant de passer d'une période d'horloge de 0,02µs (50 Mhz) à 0,1 s.

Une partie des modules ne sont composés que de liaisons entre plusieurs modules de base (ex : chronomètre, compteur par 1000, séquenceur de commande par impulsion...). On se retrouve donc avec des process très courts et simple à comprendre.

Détails des modules

Le code source fourni en pièces jointes étant détaillé et commenté, je ne détaillerais pas chaque fichier ici.

Diviseur d'horloge (div.vhd)

Le diviseur d'horloge permet de passer de l'horloge système de 50MHz à une horloge directement utilisable pour le compteur, de 10Hz.

Module d'affichage (afficheur.vhd)

Le module d'affichage converti un nombre entier (sur 4 bits) en un vecteur de 7 bits, chacun représentant un segment de l'afficheur.

```
process(E)
begin
	if(E = "0000") then
		H <= "0111111";
	elsif(E = "0001") then
		H <= "0000110";
	[...]
	else
		H <= "1111001";
	end if;
end process;
HEX <= not H;
```

Compteur par 10 (compt10.vhd)

```
process(clk)
begin
	if(clk'event and clk = '1') then
		if(nclear = '0') then
			compt <= 0;
		elsif(EN = '1') then
			if(compt = 9) then
				compt <= 0;
			else
				compt <= compt + 1;
			end if;
		end if;
	end if;
end process;
```

Le compteur par 10 stocke dans un signal (*compt*) sa valeur. A chaque coup d'horloge, et si l'entrée *EN* est active (entrée active haut), le signal s'incrémente. Après 9, le compteur repasse à 0.

Il possède une entrée synchrone d'initialisation : *nclear*

Comparateur EN (comparEN.vhd)

Lorsqu'un compteur par 10 passe de 9 à 0, ce module envoie un signal à l'entrée EN du compteur suivant. Cela permet d'activer (brièvement) le compteur des "dizaines" (seconde) lorsque celui des unité (dixièmes de seconde) passe de 9 à 0, et de même avec les centaines avec le passage des dizaines.

Compteur par 1000 (compteur.vhd)

En se basant sur le schéma présent dans le sujet, j'ai créé le module de compteur par 1 000 à partir des modules précédemment créés. Les signaux EN1 et EN2 sont nécessaires afin de passer l'information de la sortie des modules de comparateur à l'entrée du compteur par 10 suivant.

```
signal EN1, EN2 : std_logic;
begin
cpt <= S;
U0 : div port map(clk_in, clock);
U1 : compt10 port map(S(3 downto 0), nclear, EN, clock);
U2 : comparEN port map(EN, S(3 downto 0), EN1);
U3 : compt10 port map(S(7 downto 4), nclear, EN1, clock);
U4 : comparEN port map(EN1, S(7 downto 4), EN2);
U5 : compt10 port map(S(11 downto 8), nclear, EN2, clock);
end arch;
```

Module mémoire (memoire.vhd)

Le module de mémoire stocke sur 12 bits le contenu envoyé en entrée lorsque *nload* est à 0 (entrée synchrone active bas).

Chronomètre (chronometre.vhd)

Le chronomètre ne fait qu'implémenter et relier entre eux les différents modules expliqués précédemment. La mémoire est relié en entrée à la sortie du compteur et en sortie au différents afficheurs. De cette façon, on peut mettre en pause l'affichage tout en continuer d'incrémenter le compteur.

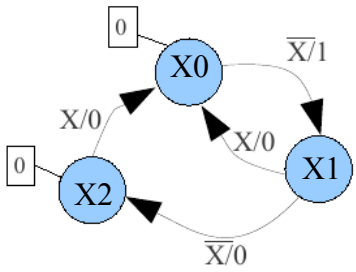
```
U0 : compteur port map(clk_in, nclear, EN, S);
U1 : afficheur port map(mem(3 downto 0), HEX0);
U2 : afficheur port map(mem(7 downto 4), HEX1);
U3 : afficheur port map(mem(11 downto 8), HEX2);
U4 : memoire port map(clk_in, nload, S, mem);
```

Impulsion unique (impulsionUnique.vhd)

Ce module permet de limiter à un seul coup d'horloge un appui sur un bouton. Cela permet d'éviter que le module de commande avance à toute vitesse lorsque l'utilisateur reste appuyé sur le bouton pour plus d'un coup d'horloge (plus de 0,1 seconde).

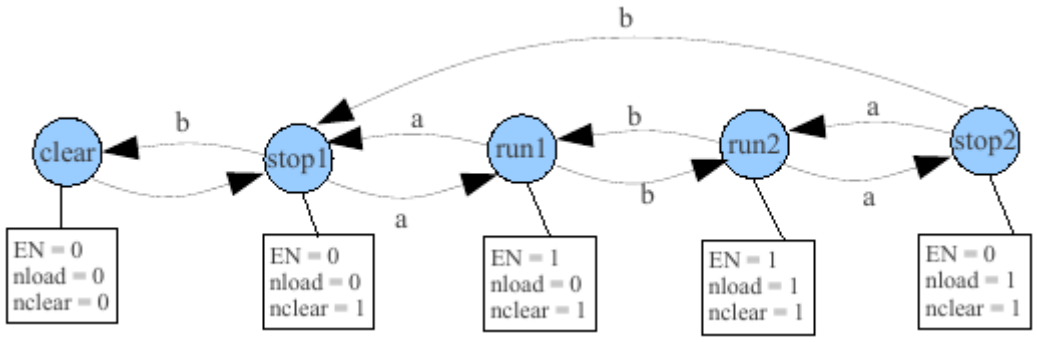
Ci-dessous se trouve l'implémentation de la machine de Mealy sous la forme de 2 process. Le premier modifie les sorties et stocke dans un signal l'état suivant de la machine. Le second s'occupe à chaque coup d'horloge de passer de l'état *courant* à l'état *suivant.*

```
architecture a of impulsionUnique is
type etat is (X0,X1,X2);
signal courant, suivant : etat := X0;
begin
process(E, courant)
begin
	case courant is
		when X0 => if E = '0' then
					S <= '1';
					suivant <= X1;
				else
					S <= '0';
					suivant <= X0;
				end if;
		when X1 =>  S <= '0';
				if E = '1' then
					suivant <= X0;
				else
					suivant <= X2;
				end if;
		when X2 => S <= '0';
				if E = '1' then
					suivant <= X0;
				else
					suivant <= X2;
				end if;
		end case;
end process;
process(clk_in)
begin
	if clk_in'event and clk_in = '1' then
		courant <= suivant;
	end if;
end process;
end a;
```



Circuit de commande (commande.vhd)

Le circuit de commande est également définit par un graphe, mais il s'agit cette fois d'une machine de Moore, où les sorties dépendent de l'état actuel et non plus des transition. Dans ce cas, une implémentation à 3 process m'a semblé plus simple. Le premier état définit l'était suivant en fonction de l'état courant et des variables. Le second passe d'un état au suivant à chaque coup d'horloge. Le 3ème définit les sorties en fonction de l'état courant (voir ci-dessous)




```
process(courant)
begin
	case courant is
		when clear => EN <= '0';
```

```
                nload <= '0';
                nclear <= '0';
            when stop1 => EN <= '0';
                nload <= '0';
                nclear <= '1';
            when run1 => EN <= '1';
                nload <= '0';
                nclear <= '1';
            when run2 => EN <= '1';
                nload <= '1';
                nclear <= '1';
            when stop2 => EN <= '0';
                nload <= '1';
                nclear <= '1';
        end case;
end process;
```

Commande par impulsion (commandeParImpulsion.vhd)

Ce module se contente d'affecter d'appliquer le générateur d'impulsion vu précedemment aux deux entrée (pb1 et pb2) du circuit de commande qu'il implémente.

Module global (resultat_final.vhd)

Le module global relie le chronomètre et le module de commande par impulsion. Un composant diviseur d'horloge à dû être rajouté afin de réduire également le signal d'horloge utilisé pour les impulsion.

```
U0 : chronometre port map(clk_in, nclear, nload, EN, HEX0, HEX1, HEX2);
U1 : commandeParImpulsion port map(clkDiv, pb1, pb2, nclear, EN, nload);
U3 : div port map(clk_in, clkDiv);
```

Difficultés rencontrées

Une bonne partie des erreurs que j'ai rencontrés lors des compilations du projet étaient liées à des erreur de syntaxe. Le fait d'avoir testé presque chaque module avec de simples LED m'a permit d'éviter de grosses étapes de débuggage et de n'avoir à corriger que des petits bouts de code.

Concernant l'afficheur, je n'avais pas remarqué que les segments étaient actifs à l'état bas. C'est pourquoi j'ai rajouté une ligne `HEX <= not H;` dans le module d'afficheur en dehors du process.

Lors de la mise en commun des modules principaux, j'ai été confronté à un problème de synchronisation entre le séquenceur de commande par impulsion et le chronomètre. En effet, je n'avais pas pensé à appliquer le diviseur d'horloge sur l'entrée du circuit de commande, ce qui engendrait des impulsions trop courtes pour êtres captées par le chronomètre. J'ai donc créé une nouvelle instance du diviseur dans le programme général (resultat_final.vhd).

Une optimisation possible aurait été de n'implémenter qu'une seule fois ce diviseur et d'envoyer le signal d'horloge modifié directement au chronomètre. Dans l'état actuel, le diviseur est également présent à l'intérieur du module compteur par 1 000 (compteur.vhd)