Romain COCOGNE
Camille BRUNIN

ELEC4
Groupe 1

Compte rendu TP5

Introduction

Dans ce TP, nous allons voir comment décrire des composants programmables à partir de fichiers.

I) Préparation

A) Le fichier sinus

Tout d'abord, nous avons généré les 512 valeurs d'un sinus sous Excel, à l'aide de la formule suivante : « =DECBIN(SIN(2*PI()*(1/2048)***LIGNE(LC(16383))**)*100;9) ».
Cela signifie que nous convertissons de décimal à binaire (DECBIN), sur 9 bits, les 512 premières valeurs d'un sinus de période 2048, avec un facteur 100 pour avoir des résultats en dizaines de mV. « **LIGNE(LC(…))** » nous permet de récupérer le numéro de la ligne du fichier Excel afin de ne pas à avoir à implémenter à la main les 512 valeurs du sinus.
Après avoir généré toutes ces valeurs, nous les exportons dans un fichier text *fichsinus.txt*.

B) Le package

Afin de pouvoir utiliser ce fichier sinus, nous avons créé un package avec pour entité la suivante :

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5   use std.textio.all;
6   use ieee.std_logic_textio.all;
7
8   package mon_pack is
9           subtype word is std_logic_vector(8 downto 0);
10          type defmem is array(natural range <>) of word;
11          impure function init_mem(nbbits: natural; s:string) return defmem;
12  end mon_pack;
13
```

Dans le corps du package, nous avons défini le corps de la fonction *init_mem,* qui permet d'ouvrir un fichier texte, de le lire, et d'enregistrer toutes ses valeurs, ligne par ligne, dans un tableau *tab*. Ce tableau est ensuite retourné par la fonction.
Il ne faut bien sûr pas oublier d'inclure tous les packages nécessaires.
Le compteur est l'indice du tableau retourné.

```
14  package body mon_pack is
15          impure function init_mem(nbbits: natural; s:string) return defmem is
16                  variable tab: defmem(0 to nbbits-1);
17                  variable li: line; --pour conserver le numero de ligne du fichier
18                  variable v: word;
19                  variable compt: natural := 0;
20                  file fichier : text is s;
21          begin
22                  while not endfile(fichier) loop
23                          readline(fichier,li);
24                          if li'length >0 then
25                          --si ça n'est pas une ligne blanche
26                                  read(li,v); --v=valeur à la ligne i
27                                  tab(compt) := v;
28                          end if;
29                          compt := compt + 1; --incrémentation du compteur
30                  end loop;
31                  return tab;
32          end function;
33  end mon_pack;
```

C) L'entité genfc

Nous avons ensuite créé un nouveau fichier dans lequel nous avons décrit l'interface du circuit à l'aide de l'entité genfc. Pour pouvoir utiliser notre package décrit précédemment, nous l'incluons dans le fichier genfc.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   library work;
6   use work.mon_pack.all; --package include
7
8   entity genfc is
9           port(Hfc,rst: std_logic;
10                m: std_logic;
11                vs: out std_logic_vector(9 downto 0));
12  end entity;
```

D) Description de l'architecture de l'entité à l'aide de 2 processus

a- Processus synchrone à l'horloge

Ce premier processus permet de décrire les transferts réalisés sur les variables *I,sens,signe* et *vs* en fonction des conditions *c1,c2,c3,c4* et *c5*.

```
architecture sin of genfc is
        signal sinus: defmem(0 to 511);
        signal sens, signe, eq0, eqmax, c1, c2, c3, c4, c5: std_logic;
        signal I: natural;
begin
sinus <= init_mem(512, "fichsinus.txt");

transfert: process(Hfc,rst) --hfc pour la syncronisation de la sortie vs
begin
        if rst='0' then
                sens <='0';
                signe <='0';
                I<=0;
        end if;

        if Hfc = '1' and Hfc'event then

                if c4 = '1' then signe <= not signe; end if;
                if c3 = '1' then sens <= not sens; end if;

                if c1 = '1' then I <= I+1;
                elsif c2 = '1' then I <= I-1;
                end if;

                if c5 = '1' then vs <= ('0' & sinus(I));
                else vs <= 0-('0' & sinus(I));
                end if;


        end if;
end process;
```

b- Les instructions d'affectation concurrentes

Nous avons décrit les indicateurs d'état eqmax et eq0 à l'aide d'instructions d'affectation concurrentes :

```
eq0<='1' when I=0 else '0';
eqmax<='1' when I=511 else '0';
```

c- Le processus combinatoire de l'unité de contrôle

Nous avons ensuite créé un deuxième processus, combinatoire, qui décrit l'unité de contrôle à l'aide de la table de vérité donnée. Pour cela, nous avons utilisé un case qui prend en paramètres un mot de 4 bits comprenant : le bit de sens en bit de poids fort, ensuite le bit de signe, puis ceux de eq0 et eqmax.

```
comb: process(m,sens,signe,eq0,eqmax)
begin
        c1<='0'; c2<='0'; c3<='0'; c4<='0'; c5<='0';
        if m= '1' then
        case std_logic_vector'(sens&signe&eq0&eqmax) is
                when "0010" | "0000" =>
                        c1<='1'; c5<='1';
                when "0011" | "0001" =>
                        c3<='1'; c5<='1';
                when "0110" | "0100" =>
                        c1<='1';
                when "0111" | "0101" =>
                        c3<='1';
                when "1001" | "1000" =>
                        c2<='1'; c5<='1';
                when "1011" | "1010" =>
                        c3<='1'; c4<='1'; c5<='1';
                when "1101" | "1100" =>
                        c2<='1';
                when "1111" | "1110" =>
                        c3<='1'; c4<='1';
                when others => null; --car deja initialise a 0
        end case;
        end if;
end process;
end architecture;
```

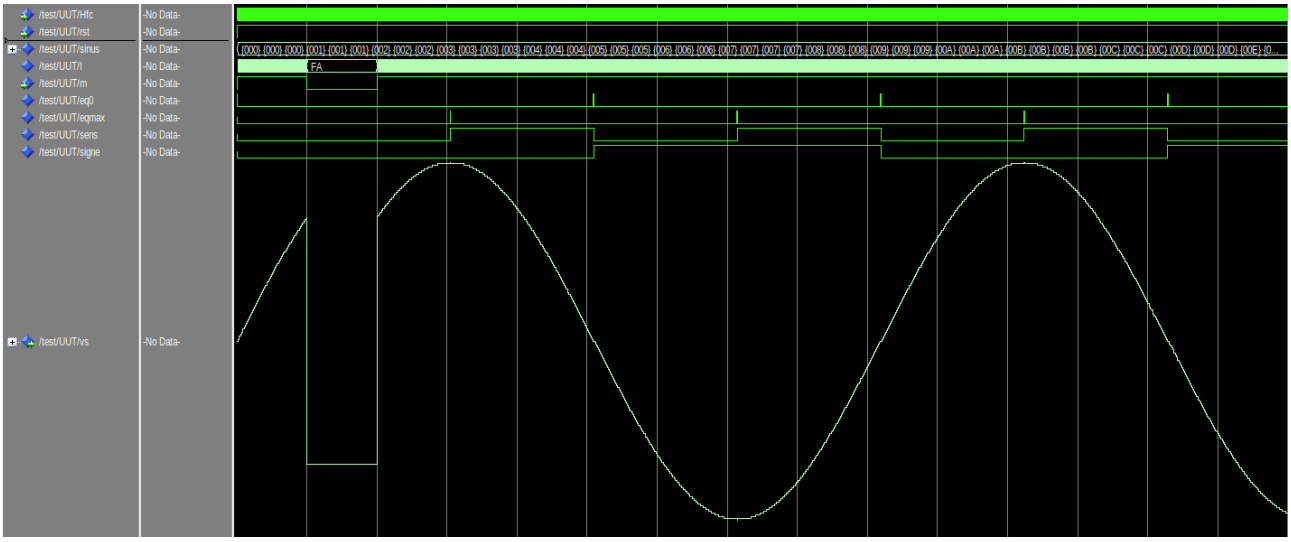
II) Manipulations

A) Test de la sinusoïde

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use std.textio.all;
5   use ieee.std_logic_textio.all;
6
7   library work;
8   use work.mon_pack.all; --package include
9
10  entity test is
11  end entity;
12
13  architecture bench of test is
14  component genfc is
15          port(Hfc,rst: std_logic;
16               m: std_logic;
17               vs: out std_logic_vector(9 downto 0));
18  end component;
19
20  signal m,hfc:   std_logic :='0';
21  signal rst:     std_logic :='1';
22  signal vs:      std_logic_vector (9 downto 0);
23  file resultat:  text open write_mode is "resultat.dat";
24
25  for UUT: genfc use entity work.genfc(sin);
26  begin
27          UUT: genfc port map (m=>m, hfc=>hfc, rst=>rst, vs=>vs);
28
29          hfc     <=      not hfc after 20 ns;
30          rst     <=      '0', '1' after 5 ns;
31          m       <=      '1' after 5 ns, '0' after 10 us, '1' after 20 us;
32  end architecture;
```

Nous avons réalisé le testbench ci-dessus pour tester notre description RTL.

Nous obtenons le chronogramme suivant.

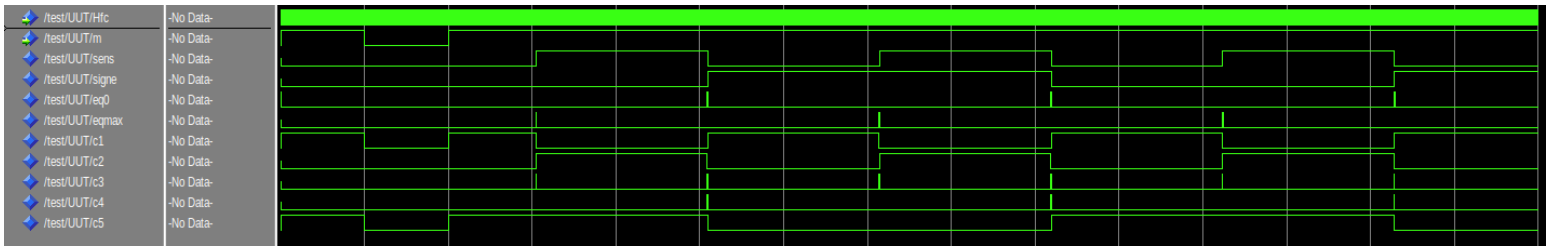


Le fonctionnement de notre description semble être vérifié, en effet :

- Le signal ‘m’ commande bien la mise en marche du sinus
- Les signaux ‘eq0’ et ‘eqmax’ réagissent correctement au compteur ‘I’
- Les signaux ‘sens’ et ‘signe’ changent de valeur au bon moment.
- Le signal ‘vs’ est bien sinusoïdal

On peut observer l’évolution des signaux ‘c1’,’c2’,’c3’,’c4’,’c5’ et confirmer qu’ils correspondent bien à la table de vérité suivante.

M	Sens	signe	eq0	eqmax	c1	c2	c3	c4	c5
0	X	X	X	X	0	0	0	0	0
1	0	0	x	0	1	0	0	0	1
1	0	0	x	1	0	0	1	0	1
1	0	1	x	0	1	0	0	0	0
1	0	1	x	1	0	0	1	0	0
1	1	0	0	x	0	1	0	0	1
1	1	0	1	x	0	0	1	1	1
1	1	1	0	x	0	1	0	0	0
1	1	1	1	x	0	0	1	1	0



Nous pouvons donc valider le fonctionnement de notre description.

Conclusion

Avec le développement de ce générateur de sinus, nous avons pu aborder le concept de fichier en VHDL. Cela pourra avoir de multiples applications, allant de la simulation de mémoire morte à la création de log de débug lors du développement d'un projet.