Projet ETI :

Dossier Technique :

Le synthétiseur musical

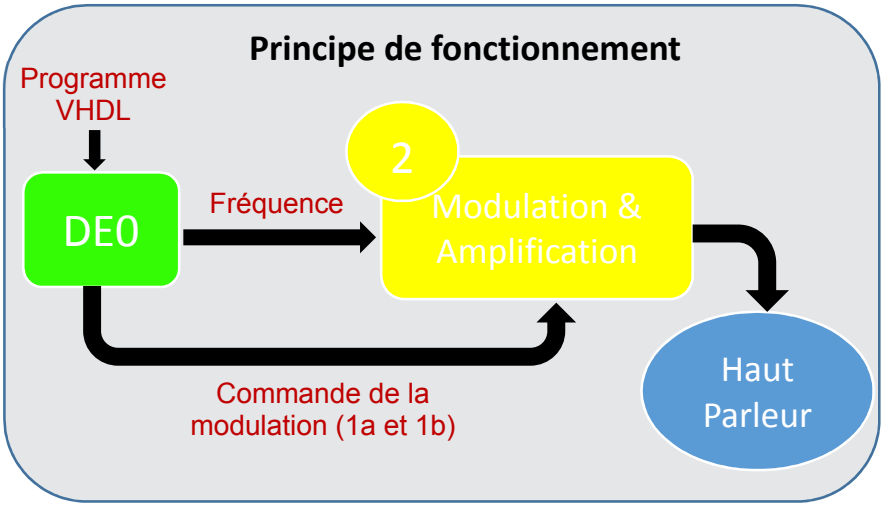
Boust Sylvain
Briosne Fréjaville Clémence
Gr 2

Cahier des charges :

- Pouvoir émettre des sons : 7 notes de la gamme de DO Majeur
- Réaliser une enveloppe en amplitude à l'émission de chacun de ces sons
- Pouvoir jouer une mélodie

Solutions techniques :

- Création des fréquences à l'aide de l'horloge présente sur la carte DEO et d'un compteur : Programmes VHDL *compteur17bitsdo*, *compteur16bitsla*, *compteur17bitsre*…
- Les notes sont jouées grâce aux interrupteurs: création d'un multiplexeur en VHDL *Multiplexer_VHDL*
- Création d'une mélodie : Programme VHDL *mélodie*
- Choisir entre jouer une note avec un interrupteur ou jouer la mélodie : Programme VHDL *choix*
- Programmation des entrées du circuit de modulation en amplitude : Programme VHDL *ampli*



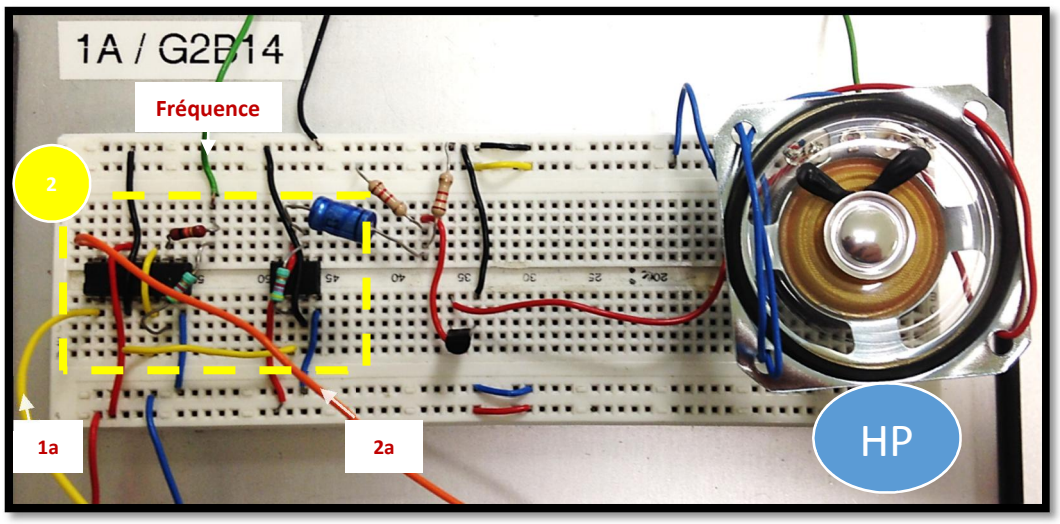


Figure 1 Photo de la maquette

- Circuit de modulation de l'amplitude : DG200a et TL071 (amplificateur opérationnel)

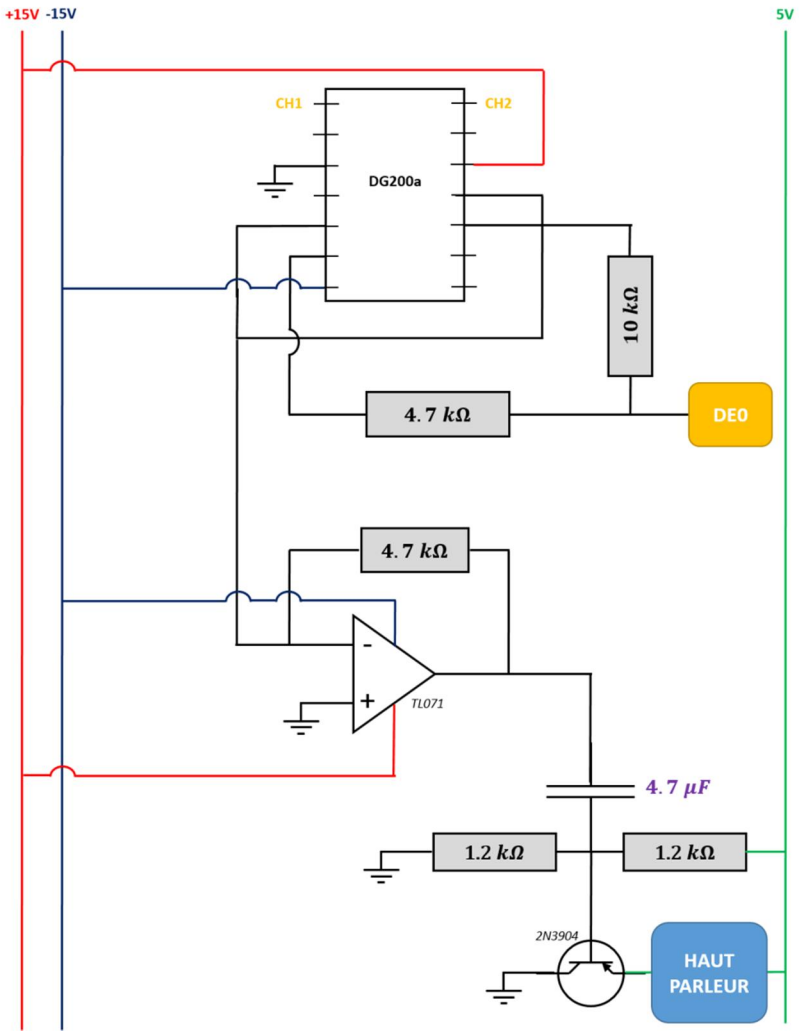


Figure 2 Circuit du synthétiseur

La carte DE0 fournit la fréquence en entrée du circuit qu'on souhaite entendre, le programme VHDL est détaillé à la fin du dossier.

Le composant DG200a est la clé de la modulation, en effet les entrées CH1 et CH2 pilotées par un code VHDL permettent de fournir en sortie différentes tensions de manière précise. Le DG200a est à logique négative, si CH1 et CH2 sont reliées à la masse, alors la tension de sortie est maximale.

Valeur de CH1	**Valeur deCH2**	**Valeur de la sortie par rapport à l'entrée**
Masse	Masse	150 %
Masse	Positif	100%
Positif	Masse	50%
Positif	Positif	0%

Pour pouvoir amplifier jusqu'à 150%, ce circuit de modulation nécessite un amplificateur opérationnel, et le TL071 répond parfaitement aux besoins du circuit.

Avant de finir au hautparleur, le signal est une dernière fois amplifié par un transistor.

Observations sur l'oscilloscope :

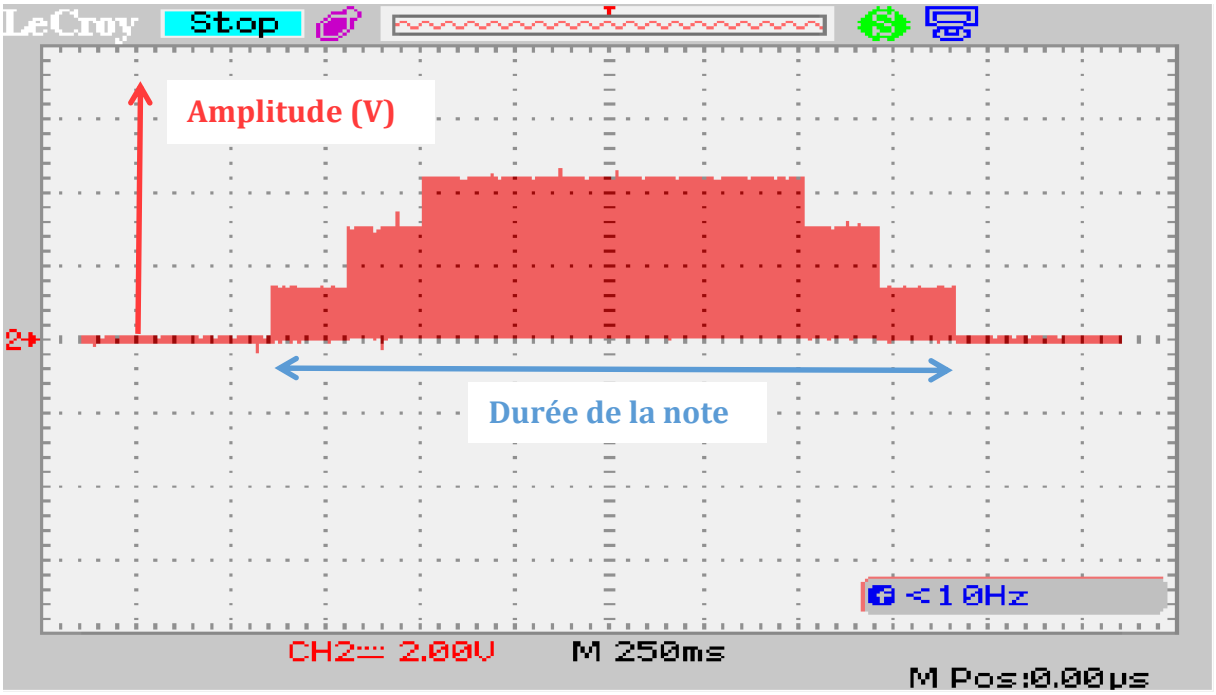


Figure 3 Observation de la variation d'amplitude

On observe bien une variation de l'amplitude du signal qui croit progressivement, puis décroit.

Programmes VHDL

- **Diagramme blocks**

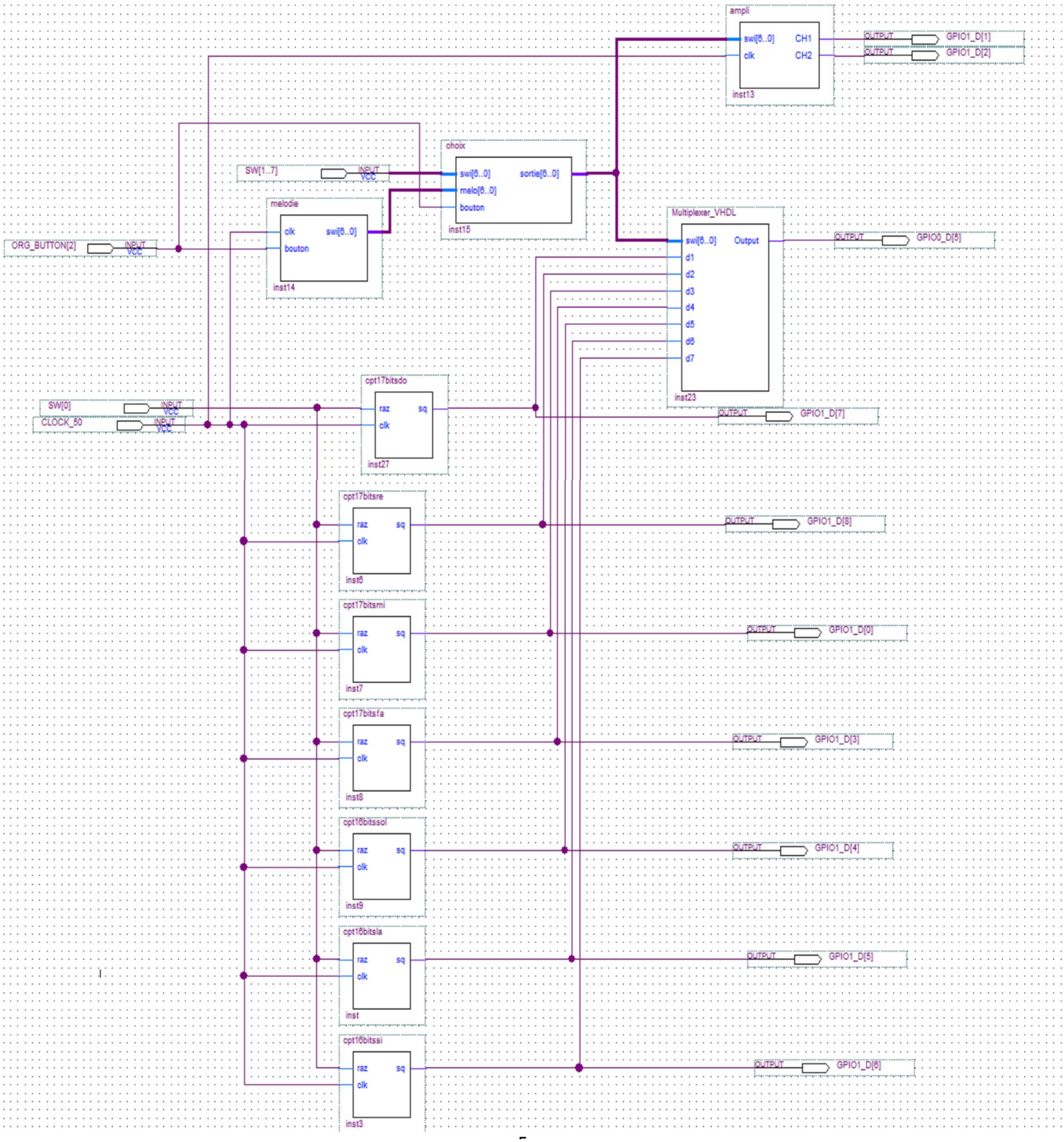


Figure 4 Diagramme en block

Fonctions en VHDL

- Création des fréquences :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity cpt17bitsre is
    port(
        raz, clk : in std_logic;
        sq : out std_logic
    );
    end cpt17bitsre;

    architecture Behavioral of cpt17bitsre is
        signal total: STD_LOGIC_VECTOR(16 DOWNTO 0);
begin
    cpt : process(raz,clk)
    begin
        if raz = '1' then -- description asynchrone
            total <= "00000000000000000";
            elsif ( clk 'event and clk='1' ) then-- description synchrone
        total<=total+1;
        end if;
        if(total="10100110010001011")
    then total <= "00000000000000000";


        end if;
    end process;


    sq <= total(16); --hors processus
    end Behavioral;
```

Figure 5 Code pour générer le ré

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity cpt17bitsdo is
    port(
        raz, clk : in std_logic;
        sq : out std_logic
    );
    end cpt17bitsdo;

    architecture Behavioral of cpt17bitsdo is
        signal total: STD_LOGIC_VECTOR(16 DOWNTO 0);
begin
    cpt : process(raz,clk)
    begin
        if raz = '1' then -- description asynchrone
            total <= "00000000000000000";
            elsif ( clk 'event and clk='1' ) then-- description synchrone
        total<=total+1;
        end if;
        if(total="10111010101000101")
    then total <= "00000000000000000";


        end if;
    end process;

    sq <= total(16); --hors processus
    end Behavioral;
```

Figure 6 Code pour générer le do

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity cpt17bitsfa is
    port(
        raz, clk : in std_logic;
        sq : out std_logic
    );
    end cpt17bitsfa;

    architecture Behavioral of cpt17bitsfa is
        signal total: STD_LOGIC_VECTOR(16 DOWNTO 0);
begin
    cpt : process(raz,clk)
    begin
        if raz = '1' then -- description asynchrone
            total <= "00000000000000000";
            elsif ( clk 'event and clk='1' ) then-- description synchrone

        total<=total+1;
        end if;
        if(total="10001011110100010")
    then total <= "00000000000000000";


        end if;
    end process;



    sq <= total(16); --hors processus
    end Behavioral;
```

Figure 7 Code pour générer le fa

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity cpt17bitsmi is
    port(
        raz, clk : in std_logic;
        sq : out std_logic
    );
    end cpt17bitsmi;

    architecture Behavioral of cpt17bitsmi is
        signal total: STD_LOGIC_VECTOR(16 DOWNTO 0);
begin
    cpt : process(raz,clk)
    begin
        if raz = '1' then -- description asynchrone
            total <= "00000000000000000";
            elsif ( clk 'event and clk='1' ) then-- description synchrone

        total<=total+1;
        end if;
        if(total="10010100001000011")
    then total <= "00000000000000000";


        end if;
    end process;


    sq <= total(16); --hors processus
    end Behavioral;
```

Figure 8 Code pour générer le mi

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity cpt16bitssol is
    port(
        raz, clk : in std_logic;
        sq : out std_logic
    );
    end cpt16bitssol;

    architecture Behavioral of cpt16bitssol is
        signal total: STD_LOGIC_VECTOR(15 DOWNTO 0);
begin
    cpt : process(raz,clk)
    begin
        if raz = '1' then -- description asynchrone
            total <= "0000000000000000";
            elsif ( clk 'event and clk='1' ) then-- description synchrone

        total<=total+1;
        end if;
        if(total="1111100100100000")
    then total <= "0000000000000000";


        end if;
    end process;


    sq <= total(15); --hors processus
    end Behavioral;
```

Figure 9 Code pour générer le sol

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity cpt16bitsla is
    port(
        raz, clk : in std_logic;
        sq : out std_logic
    );
    end cpt16bitsla;

    architecture Behavioral of cpt16bitsla is
        signal total: STD_LOGIC_VECTOR(15 DOWNTO 0);
begin
    cpt : process(raz,clk)
    begin
        if raz = '1' then -- description asynchrone
            total <= "0000000000000000";
            elsif ( clk 'event and clk='1' ) then-- description synchrone
        total<=total+1;
        end if;
        if(total="1101110111110010")
    then total <= "0000000000000000";


        end if;
    end process;


    sq <= total(15); --hors processus
    end Behavioral;
```

Figure 10 Code pour générer le la

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity cpt16bitssi is
    port(
        raz, clk : in std_logic;
        sq : out std_logic
    );
    end cpt16bitssi;

    architecture Behavioral of cpt16bitssi is
        signal total: STD_LOGIC_VECTOR(15 DOWNTO 0);
begin
    cpt : process(raz,clk)
    begin
        if raz = '1' then -- description asynchrone
            total <= "0000000000000000";
            elsif ( clk 'event and clk='1' ) then-- description synchrone

        total<=total+1;
        end if;
        if(total="1100010110111011")
    then total <= "0000000000000000";


        end if;
    end process;


    sq <= total(15); --hors processus
    end Behavioral;
```

Figure 11 Code pour générer le si

- Multiplexeur

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplexer_VHDL is
   port
   (
      swi : in std_logic_vector(6 DOWNTO 0);
      d1,d2,d3,d4,d5,d6,d7 : in std_logic;

      Output: out std_logic

   );
end entity Multiplexer_VHDL;

architecture Behavioral of Multiplexer_VHDL is
begin
   process (d1, d2, d3, d4, d5, d6, d7, swi) is
   begin
      case swi is
         when "0000001"  => Output <= d1;
         when "0000010"  => Output <= d2;
         when "0000100"  => Output <= d3;
         when "0001000"  => Output <= d4;
         when "0010000"  => Output <= d5;
         when "0100000"  => Output <= d6;
         when "1000000"  => Output <= d7;
         when others => Output <= '0';
      end case;
   end process;
end architecture Behavioral;
```

Figure 12 Code du multiplexeur

- Mélodie

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity melodie is
   port
   (

      clk : in std_logic;
      bouton : in std_logic;

      swi : out std_logic_vector(6 DOWNTO 0)

   );
end entity melodie;

architecture Behavioral of melodie is

signal cpt: integer range 0 to 870000000;

begin

   process (clk, bouton) is
   begin

   if bouton='0' then

        if clk'event and clk='1' then
            cpt<=cpt+1;

        if(cpt=1) then --do--
        swi<="0000001";

        elsif(cpt=100000000) then  --sol--
        swi<="0010000";

        elsif(cpt=210000000) then  --la--
        swi<="0100000";

        elsif(cpt=320000000) then  --sol--
        swi<="0010000";

        elsif(cpt=430000000) then  --fa--

        elsif(cpt=540000000) then  --mi--
        swi<="0000100";

        elsif(cpt=650000000) then  --ré--
        swi<="0000010";

        elsif(cpt=760000000) then  --do--
        swi<="0000001";



        end if;
        end if;

   else
        swi<="0000000";

   end if;
   end process;
end architecture Behavioral;
```

Figure 13 Code de la mélodie

- Choix de mode de fonctionnement : notes simples/mélodie

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity choix is
   port
   (
      swi : in std_logic_vector(6 DOWNTO 0);
      melo : in std_logic_vector(6 DOWNTO 0);
      bouton : in std_logic;

      sortie : out std_logic_vector(6 DOWNTO 0)

   );
end entity choix;

architecture Behavioral of choix is

signal cpt: integer range 0 to 110000000;

begin

   process (melo, swi, bouton) is
   begin

   if bouton='0' then
   sortie <= melo;

   else
   sortie <= swi;

   end if;

   end process;
end architecture Behavioral;
```

Figure 14 Code du programme "choix"

- Modification de l'amplitude du signal

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ampli is
   port
   (
      swi : in std_logic_vector(6 DOWNTO 0);
      clk : in std_logic;

      CH1 : out std_logic;
      CH2 : out std_logic

   );
end entity ampli;

architecture Behavioral of ampli is

signal cpt: integer range 0 to 110000000;

begin


   process (clk, swi) is
   begin


   if (swi/="0000000") then

        if clk'event and clk='1' then
            cpt<=cpt+1;

        if(cpt=1) and (swi/="0000000") then --montée premier niveau--
        CH1<='1';
        CH2<='0';

        elsif(cpt=5000000) and (swi/="0000000") then  --montée deuxième niveau--
        CH1<='0';
        CH2<='1';

        elsif(cpt=10000000) and (swi/="0000000") then  --montée au maximum--
        CH1<='0';
        CH2<='0';

        elsif(cpt=80000000) then  --redescente deuxieme niveau--
        CH1<='0';
        CH2<='1';

        elsif(cpt=85000000) then  --redescente premier niveau--
        CH1<='1';
        CH2<='0';

        elsif(cpt=90000000) then  --redescente à 0--
        CH1<='1';
        CH2<='1';

        end if;
        end if;

   else
        CH1<='1';
        CH2<='1';



   end if;
   end process;
end architecture Behavioral;
```

Figure 15 Programme de gestion de l'amplitude