Rapport de projet d'ASIC

Synthèse d'un générateur de signal programmable

Table des matières

Table des illustrations

Annexes

1.Spécifications du besoin et analyse

1.Objectif

L'objectif du projet est de modéliser et valider un générateur de signaux programmable. Les signaux générés de précisions 1/1000 sont :

- Une sinusoïde de période *Tp* et d'amplitude 5 V avec un *offset* et une excursion de +/- 10 V.
- Un signal carré d'amplitude 5 V de période *Tp*.

Les paramètres *Tp* et *Offset* ont les précisions et gammes suivantes :

- *Offset* varie entre -10 et 10 V avec une précision de 1/1000
- *Tp* varie entre 1ms et 1 s.

2.Interface du générateur programmable

Le générateur programmable de signaux est piloté par un micro-processeur ou micro-contrôleur 8 bits. Une entrée *Addr* permet au composant pilotant le générateur de signaux de définir la donnée envoyée sur le port *Data* à la cadence du signal *Write*.

En sortie, le générateur fournit sur le signal *Vs* de 11 bits la sinusoïde et le signal carré sur *Vc*.

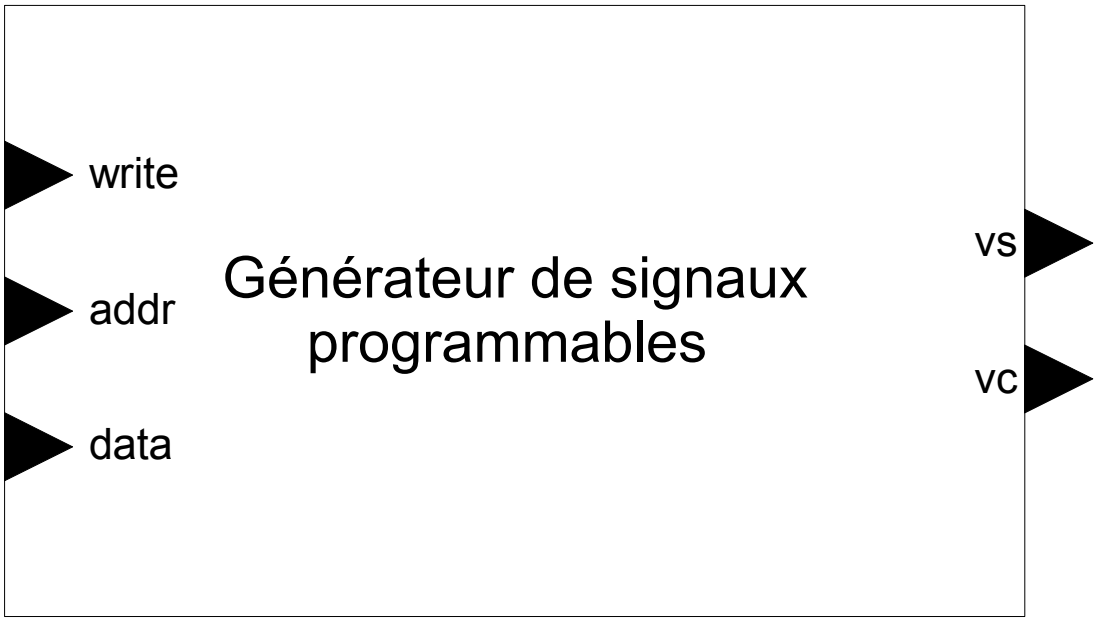

Illustration 1: Interface globale

La structure fonctionnelle la plus adaptée est la suivante :

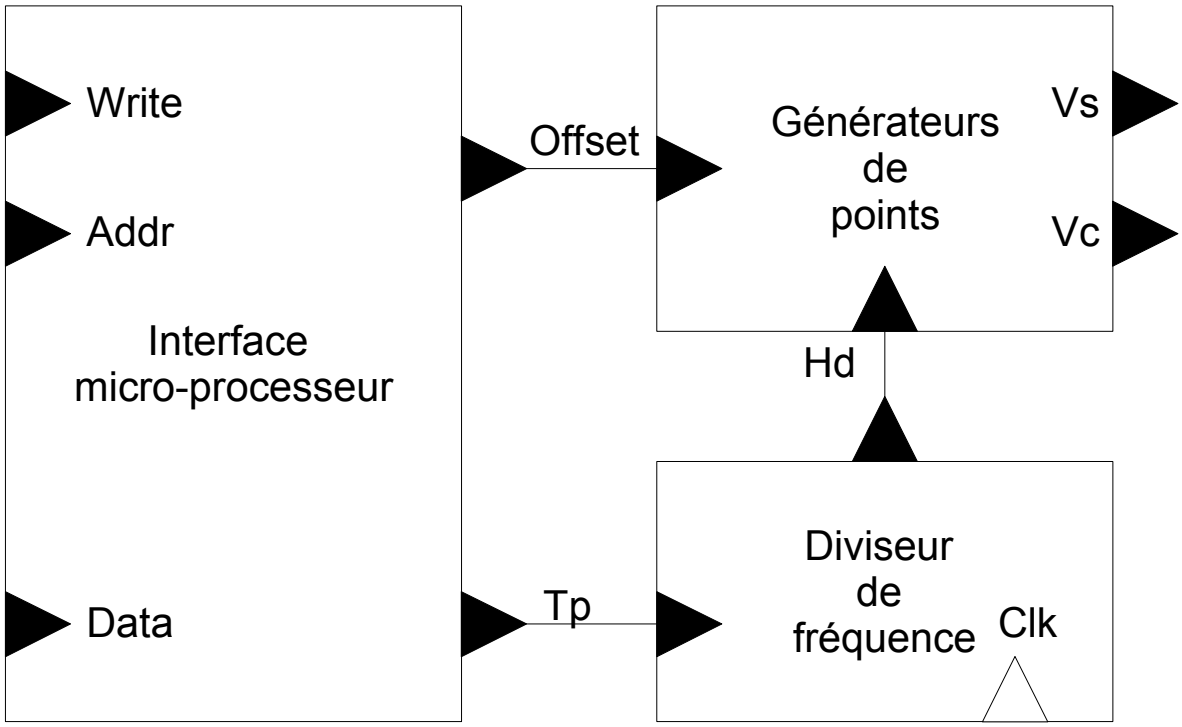

Illustration 2: Structure fonctionnelle

3.Conception au niveau RTL

1. Interface micro-processeur

Le composant pilotant le générateur de signaux programmable possède 8 bits de données. Or pour satisfaire la précision de la valeur moyenne (l'*Offset*), il est nécessaire de fournir 11 bits de données au générateur programmable. Pour satisfaire la précision de *Tp* il faut fournir 10 bits de données.

Le générateur de signaux programmable doit donc être en mesure de gérer la transformation des informations du composant pilotant.

La solution mise en oeuvre dispose donc d'une *Interface micro-processeur* dont le but est de gérer la donnée en entrée pour la transformer en donnée interne (*Offset* ou *Tp*) en fonction des signaux *Addr*, *Write* et *Data*. Dans cette solution, *Write* est le signal de synchronisation (front montant) pour la capture de la donnée sur *Data*.

Deux registres internes à l'interface permettent de recueillir les données en entrée et de les retourner aux autres entités du générateur de signaux une fois les transferts des deux paramètres complétés.

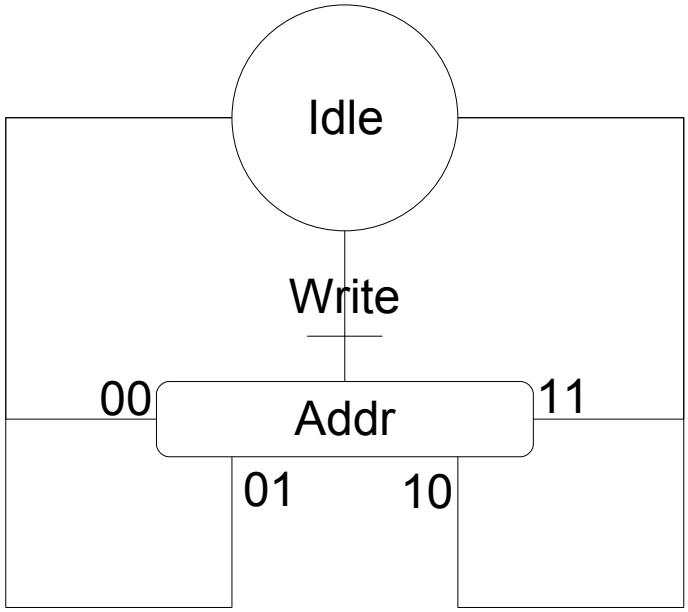

Illustration 3: Fonctionnement de l'interface

La traduction au niveau RTL de l'interface conduit donc au schéma suivant.

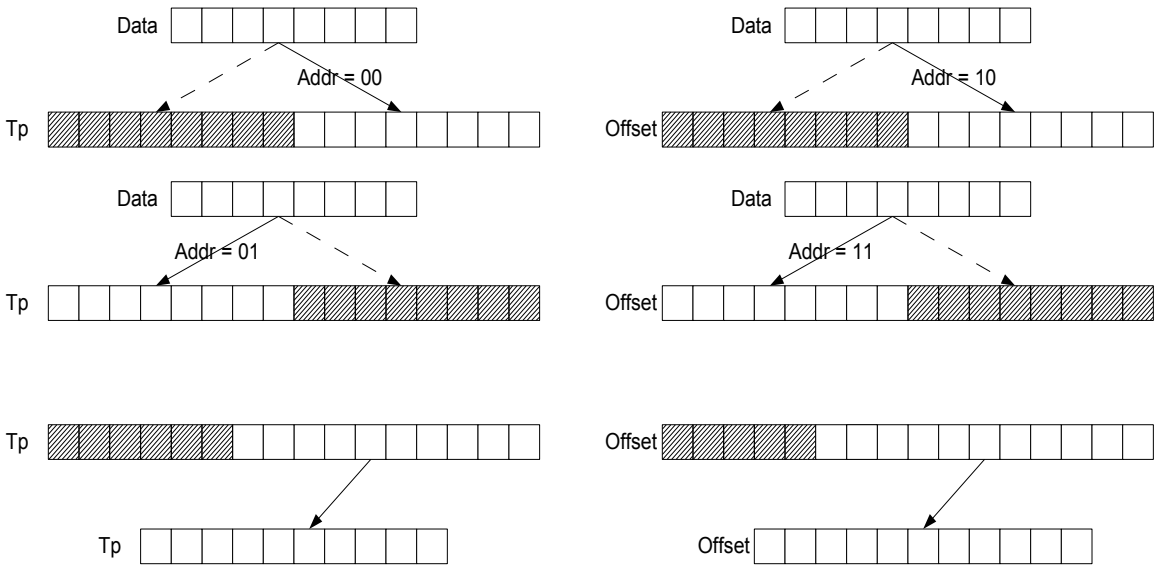

Illustration 4: Opérations de transfert au niveau de l'interface micro processeur

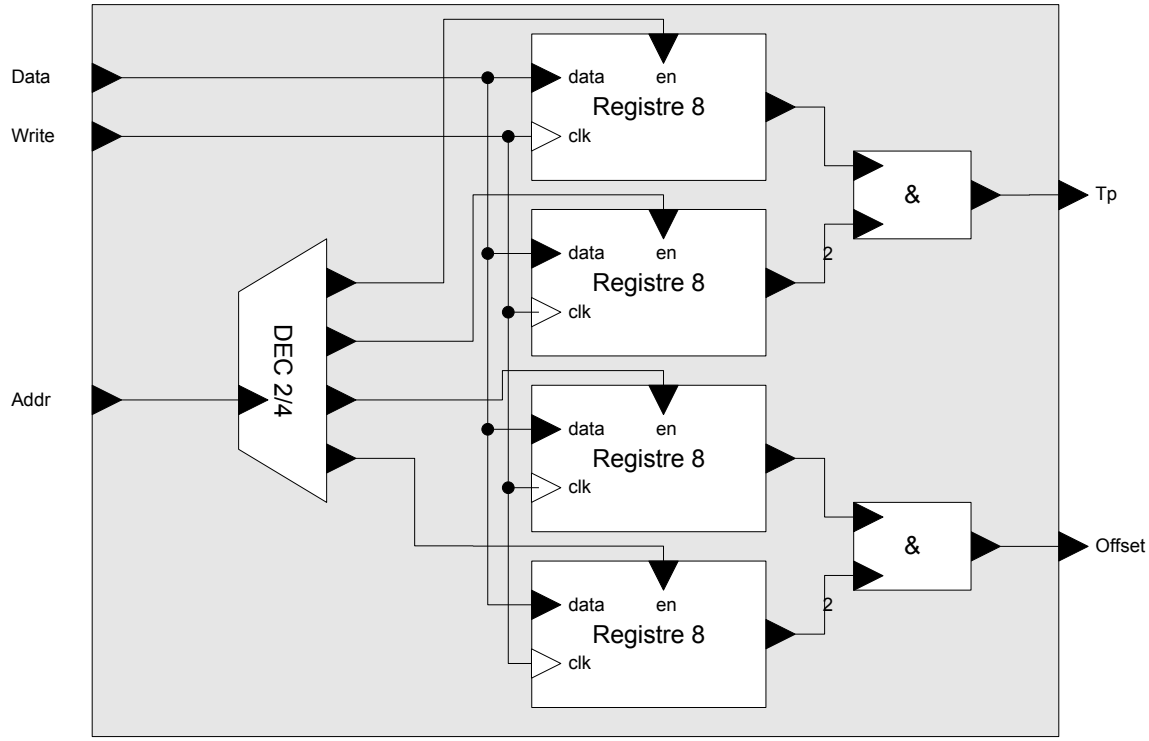

Illustration 5: RTL de l'interface

2. Diviseur de fréquence

Le diviseur de fréquence est l'organe qui module la période des signaux de sortie. Il fournit au générateur de point un signal d'horloge *Hd* fraction de l'horloge générale *clk*. Il est courant de matérialiser cette fonction par un compteur modulo. Dans ce cas le compteur est modulo *Tp*.

3. Générateur de points

Le générateur de points peut être modélisé par une FSMD. Si le chemin de données est constitué d'éléments simples, l'aspect synthétisable de l'ensemble est garanti (les machines à états ont été étudiées l'an passé).

L'idée générale est de stocker les valeurs du premier quadrant d'un sinus dans une ROM et de balayer les valeurs sur ordre de *Hd* avec un compteur et de décider du signe à donner à la valeur.

- Du min au max, puis du max au min (ce qui fournit les valeurs positives de la première demie période).
- Du min au max avec opposition de signe puis de max au min avec opposition de signe (ce qui fournit les valeurs négatives de la seconde demie période).

1. Génération des points du signal sinusoïdal

Pour générer les points du sinus, il est nécessaire de fournir périodiquement (sur signal

Hd) des valeurs caractéristiques de la sinusoïde. Compte tenu d'une précision au 1/1000, il faut donc au minimum 1000 points sur une période minimale (1 ms) du sinus. Or le signal sinusoïdal est impair, donc les valeurs au delà de la demie période corresponde à l'opposé des valeurs de la première demie période. Il est donc suffisant de stocker uniquement les 500 premières valeurs du sinus. Ensuite la demie période d'un sinus est un signal pair, donc il n'est plus nécessaire que de stocker les 250 premiers points.

Il a été décidé de garder 2000 points sur la période minimale d'1ms. Il faut donc une ROM stockant 500 valeurs et une horloge générale de 2 Mhz. Mais une telle période pour l'horloge générale n'assure pas des fractions de périodes entières. Pour cela il est nécessaire de doubler l'horloge générale qui passe à 4 Mhz et revoir légèrement le fonctionnement du diviseur de fréquence.

Les valeurs stockées en ROM sont issues d'un petit programme Java dont le code est donné ci-dessous :

```
public class generateur{
public static void main(String[] args) {
        String ch="";
        int val;
        double val2;
        for(int i=0; i<501 ;i++){
            val2=500*Math.sin(2*Math.PI*0.0005*i);
            val=(int)Math.round(val2);
            ch=Integer.toBinaryString(val);
            for(int j=ch.length();j<9;j++) ch="0"+ch;
            System.out.println("r("+i+"):=\""+ch+"\";");
        }
    }
 }
```

Celui-ci donne directement sur la sortie standard le code VHDL d'un tableau de valeurs binaires. Il ne reste plus qu'à les insérer dans le source VHDL de la ROM.

2. Le chemin de données

La transcription au niveau RTL du fonctionnement du générateur de points conduit au schéma du chemin de données suivant :

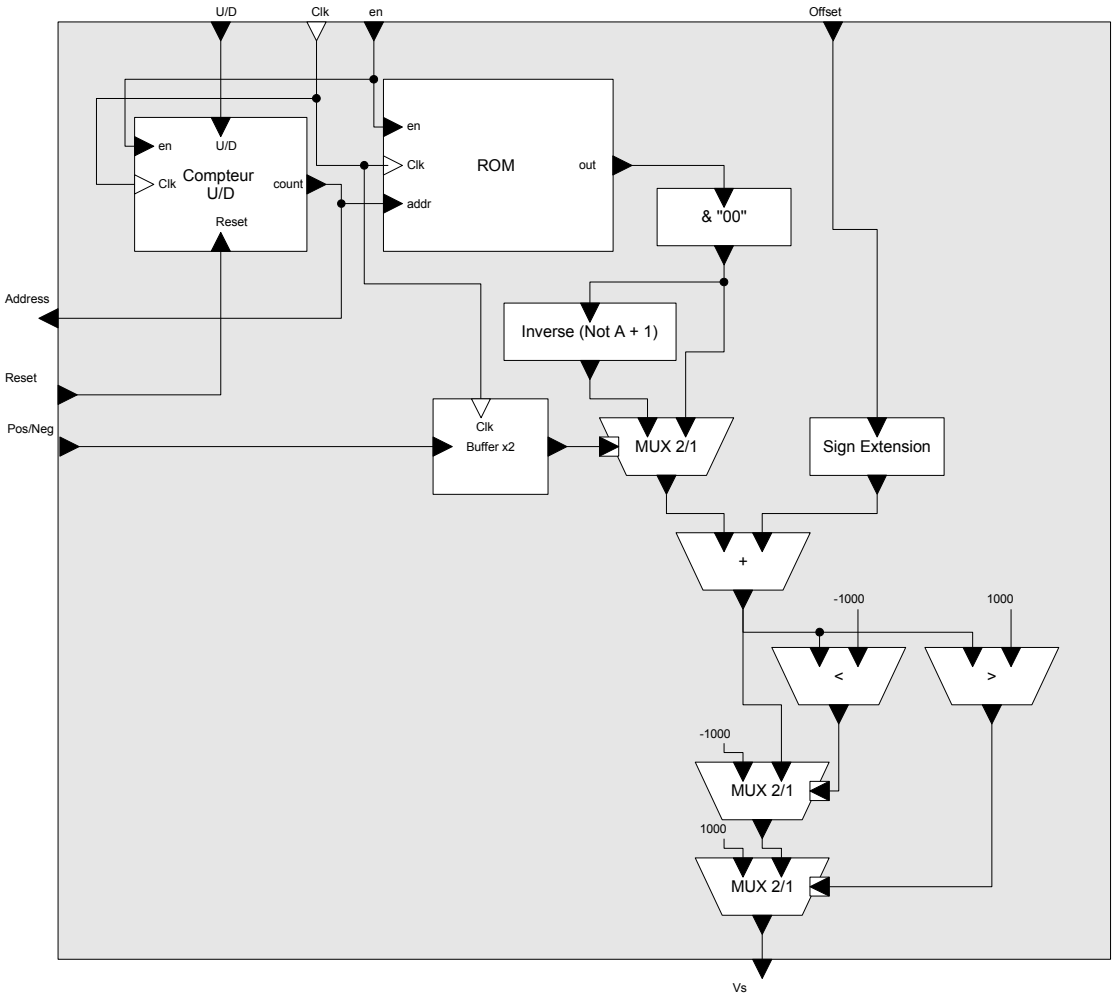

Illustration 6: Chemin de données

Le *buffer* permet de synchroniser sur le même top d'horloge l'arrivée de la sortie de la ROM avec l'aiguillage du choix de signe de la valeur en sortie de ROM.

3. Machine d'états

La machine d'états commande le sens du compteur et le choix du signe de la valeur du sinus en fonction de la valeur courante du compteur (qui donne directement l'adresse de la valeur du sinus stockée dans la ROM).

Quatre états suffisent pour commander le chemin de données précédent :

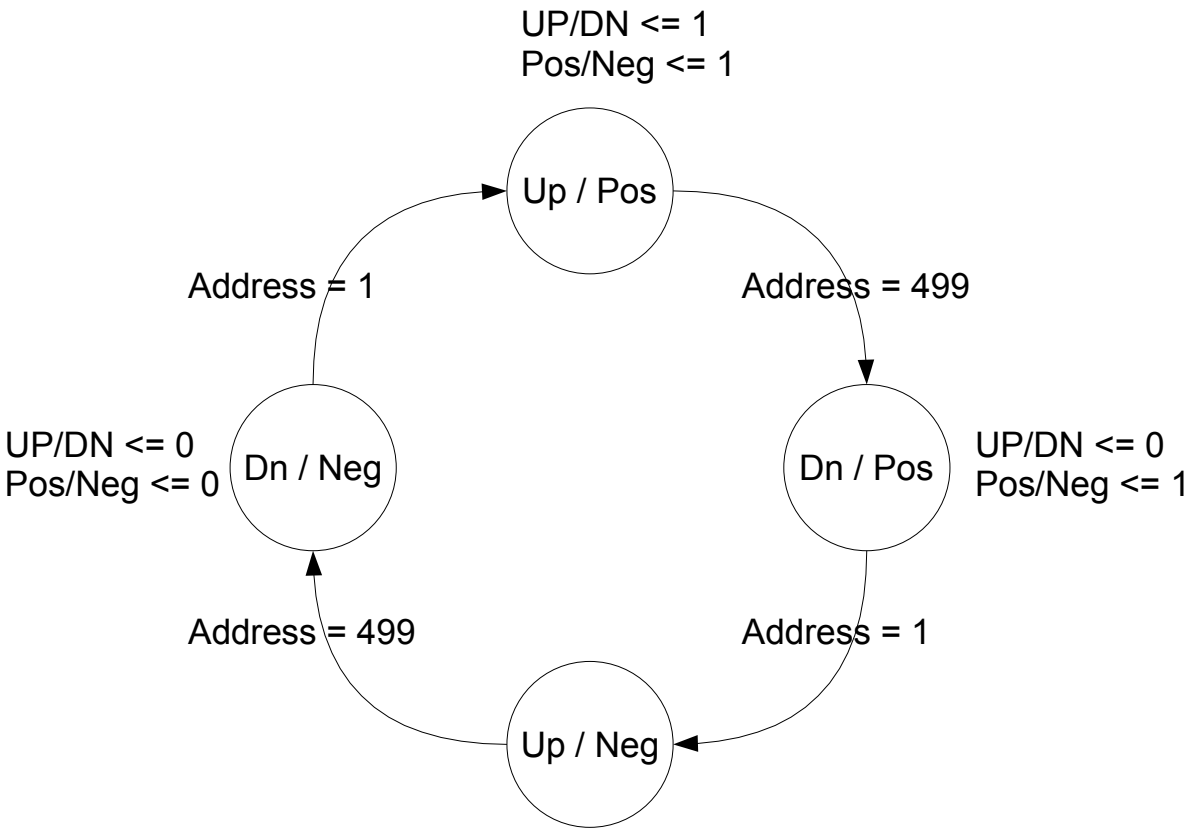

Illustration 7: Machine d'états de Moore pilotant le chemin de données

2.Modélisation sous ModelSim

1.Modélisation de l'interface micro-processeur

1. Entité

L'interface micro-processeur doit assurer le traitement des données en entrée pour les distribuer au générateur de points et au diviseur de fréquence. L'entité est donc la suivante :

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

---------------------------------------------------------------------------
-- Entity : Interface_Microprocesseur
--
---------------------------------------------------------------------------
entity Interface_Microprocesseur is
    port(Data   : in std_logic_vector(7 downto 0);
         Addr   : in std_logic_vector(1 downto 0);
```

```
        Write  : in std_logic;
        Reset  : in std_logic;
        Offset : out std_logic_vector(10 downto 0);
        Tp     : out std_logic_vector(9 downto 0));
end Interface_Microprocesseur;
```

2. Architecture

Pour distribuer les données, l'interface doit d'abord stocker *Tp* et *Offset* dans des registres internes (*Tp_int* et *Offset_int*). Une fois les 2 transferts terminés, les valeurs des registres internes sont poussés vers la sortie.

```
--------------------------------------------------------------------------------
-- Architecture : Interface_Microprocesseur
--
--------------------------------------------------------------------------------
architecture fsm of Interface_Microprocesseur is
begin

   process(write,reset)
   variable Tp_int     : std_logic_vector(15 downto 0);
   variable Tp_out     : std_logic_vector(9 downto 0);
   variable Offset_int : std_logic_vector(15 downto 0);
   variable Offset_out : std_logic_vector(10 downto 0);
   begin
      if(reset ='0') then
         Tp_int := (others => '0');
         Tp_out := (others => '0');
         Offset_int := (others => '0');
         Offset_out := (others => '0');
      elsif(write = '1' and write'event) then
         case Addr is
         when "00" =>
            Tp_int(7 downto 0)  := data(7 downto 0);
         when "01" =>
            Tp_int(15 downto 8) := data(7 downto 0);
         when "10" =>
            Offset_int(7 downto 0) := data(7 downto 0);
         when "11" =>
            Offset_int(15 downto 8) := data(7 downto 0);
            -- Push to ouputs
            Tp_out := Tp_int(9 downto 0);
            Offset_out := Offset_int(10 downto 0);
         when others => null;
         end case;
      end if;
      Tp <= Tp_out;
      Offset <= Offset_out;
   end process;
end fsm;
```

3. Validation

La figure suivante permet de valider le fonctionnement de l'interface micro-processeur.

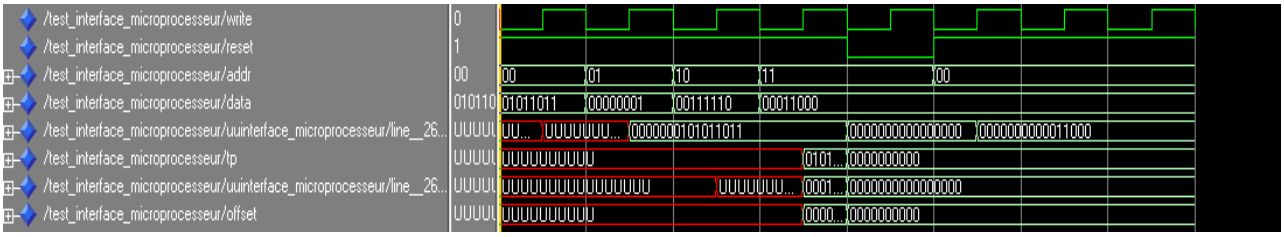

Illustration 8: Validation de l'interface micro processeur

2.Modélisation et validation du diviseur de fréquence

1. Entité

Conformément à la structure fonctionnelle du générateur, l'entité du diviseur de fréquence est la suivante.

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

--------------------------------------------------------------------------------
-- Entity : Diviseur_Fréquence
--
--------------------------------------------------------------------------------
entity Diviseur_Frequence is
    port(clk   : in std_logic;
         Reset : in std_logic;
         Tp    : in std_logic_vector(9 downto 0);
         HD    : out std_logic);
end Diviseur_Frequence;
```

2. Architecture

Le diviseur de fréquence est un compteur avec pour entrée une fréquence d'horloge générale de 4 Mhz. Ceci permet lorsque *Tp* est à 1 de fournir une fréquence de 2 Mhz sur *HD*.

```
--------------------------------------------------------------------------------
-- Architecture : Diviseur_Fréquence
--
--------------------------------------------------------------------------------
architecture beh of Diviseur_Frequence is
begin
    process(clk,Reset)
    variable count : std_logic_vector(10 downto 0):=(others => '0');
    variable HD_int : std_logic :='0';

    begin

        if(clk='1' and clk'event) then
            if(Reset = '0') then
                HD_int := '0';
                count := (others => '0');
            elsif(count >= Tp -1) then
                count := (others => '0');
```

```
            HD_int := not HD_int;
         else
            count := count + 1;
         end if;
      end if;
      HD <= HD_int;
   end process;
end beh;
```

3. Validation

Le chronogramme suivant montre le bon fonctionnement du diviseur de fréquence pour différentes valeurs de *Tp* aussi bien paires qu'impaires.

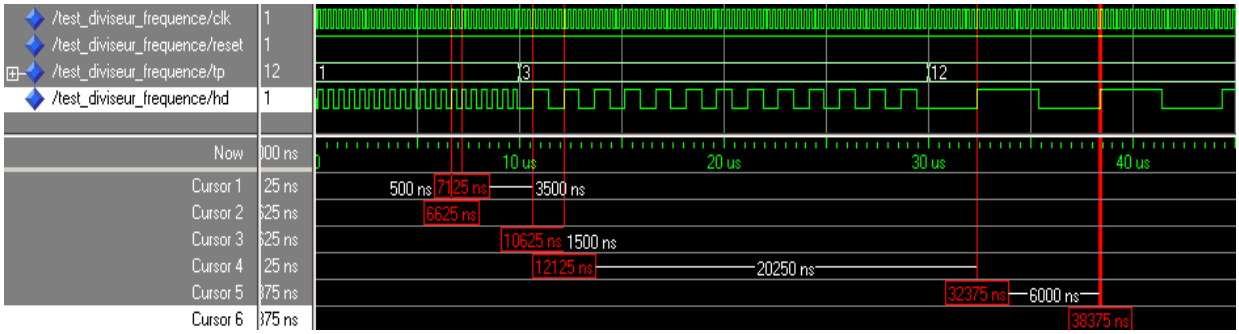

Illustration 9: Validation du diviseur de fréquence

3.Modélisation et validation du générateur de points

1. Entité

Le signal *HD* en entré du générateur de points est équivalent à un signal d'horloge variable.

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

--------------------------------------------------------------------------------
-- Entity : Generateur_Points
--
--------------------------------------------------------------------------------
entity Generateur_Points is
   port(Offset  : in std_logic_vector(10 downto 0);
        Reset   : in std_logic;
        HD      : in std_logic;
        Vs      : out std_logic_vector(10 downto 0);
        Vc      : out std_logic);
end Generateur_Points;
```

2. Architecture

L'architecture est basée sur un chemin de données commandé par une machine d'états (voire spécifications). Le détail des composants majeurs constituants le chemin de

données est détaillé dans la suite.

1. Le compteur

Selon les règles de synthèse, l'entité et l'architecture du compteur sont les suivantes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------------------------------------------------------
-- Entity : Counter
--
--------------------------------------------------------------------------------
entity Counter is
    port(clk   : in std_logic;
         Reset : in std_logic;
         Up_Dn : in std_logic;
         en    : in std_logic;
         cpt   : out std_logic_vector(8 downto 0));
end Counter;

--------------------------------------------------------------------------------
-- Architecture : Counter
--
--------------------------------------------------------------------------------
architecture beh of Counter is
begin
    process(clk,Reset)
    variable count : std_logic_vector(8 downto 0):=(others=> '0');
    begin
        if(clk='1' and clk'event) then
            if(Reset = '0') then
                count := (others => '0');
            elsif(en = '1') then
                if(Up_Dn = '1') then
                    count := count + 1;
                else
                    count := count - 1;
                end if;
            end if;
        end if;
        cpt <= count;
    end process;
end beh;
```

2. Le registre à décalages

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------------------------------------------------------
-- Entity : Shifter
--
--------------------------------------------------------------------------------

entity Shifter is
    port(clk    : in std_logic;
         input  : in std_logic;
         output : out std_logic);
end Shifter;
```

```
--------------------------------------------------------------------------------
-- Architecture : Shifter
--
--------------------------------------------------------------------------------
architecture beh of Shifter is
signal data_int : std_logic:='0';
begin
   process(clk)
   begin
      if(clk='1' and clk'event) then
         data_int<=input;
      end if;
      output <= data_int;
   end process;
end beh;
```

3. La mémoire ROM

La ROM stocke toutes les valeurs du premier quadrant du sinus. Pour qu'elle soit synthétisable, la dimension de la ROM est portée à 512 valeurs, même si seulement 500 valeurs sont nécessaires. D'autre part un *template* avec bufferisation de l'adresse a été choisi.

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

--------------------------------------------------------------------------------
-- Entity : ROM
--
--------------------------------------------------------------------------------
entity ROM is
port (clk : in std_logic;
     en : in std_logic;
     addr : in std_logic_vector(8 downto 0);
     data : out std_logic_vector(8 downto 0));
end ROM;


--------------------------------------------------------------------------------
-- Architecture : ROM
--
--------------------------------------------------------------------------------
architecture Synthesis of ROM is
type rom_type is array (0 to 511) of std_logic_vector (8 downto 0);

function init_ROM(n:natural) return rom_type is
variable r : rom_type;
begin
r(0):="000000000";
r(1):="000000010";
r(2):="000000011";
r(3):="000000101";
r(4):="000000110";
...
r(498):="111110100";
r(499):="111110100";
r(500):="111110100";
r(501):="000000000";
r(502):="000000000";
```

```
r(503):="000000000";
r(504):="000000000";
r(505):="000000000";
r(506):="000000000";
r(507):="000000000";
r(508):="000000000";
r(509):="000000000";
r(510):="000000000";
r(511):="000000000";
return r;
end init_ROM;

signal ROM : rom_type := init_ROM(0);
begin
   process (clk)
   variable read_addr : std_logic_vector(8 downto 0);
   begin
      if (clk'event and clk = '1') then
         if (en = '1') then
            read_addr := addr;
         end if;
      end if;
      data <= ROM(conv_integer(read_addr));
   end process;
end Synthesis;
```

4. Le chemin de données

Le chemin de données reprend les composants précédents et les assemble selon le schéma présenté dans les spécifications.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------------------------------------------------------
-- Entity : Datapath
--
--------------------------------------------------------------------------------

entity Datapath is
   port(clk      : in std_logic;
        Reset    : in std_logic;
        up_dn    : in std_logic;
        offset   : in std_logic_vector(10 downto 0);
        en       : in std_logic;
        pos_neg  : in std_logic;
        vs       : out std_logic_vector(10 downto 0);
        address  : out std_logic_vector(8 downto 0));
end Datapath;
```

```
--------------------------------------------------------------------------------
-- Architecture : Dataflow
--
--------------------------------------------------------------------------------
architecture dataf of Dataflow is
component Counter is
   port(clk   : in std_logic;
        Reset : in std_logic;
        Up_Dn : in std_logic;
        en    : in std_logic;
```

```
        cpt   : out std_logic_vector(8 downto 0));
end component;

component ROM is
port (clk : in std_logic;
      en : in std_logic;
      addr : in std_logic_vector(8 downto 0);
      data : out std_logic_vector(8 downto 0));
end component;

component Shifter is
    port(clk    : in std_logic;
         input  : in std_logic;
         output : out std_logic);
end component;

signal address_int : std_logic_vector(8 downto 0);
signal ROM_out : std_logic_vector(8 downto 0);
signal ROM_out_ext : std_logic_vector(11 downto 0);
signal Offset_ext  : std_logic_vector(11 downto 0);
signal ROM_out_ext_comp2 : std_logic_vector(11 downto 0);
signal sin_value : std_logic_vector(11 downto 0);
signal sum_signals : std_logic_vector(11 downto 0);
signal pos_neg2 : std_logic;
signal vs_int : std_logic_vector(10 downto 0);

begin
    UCounter : Counter port map(clk,Reset,up_dn,en,address_int);
    UROM : ROM port map(clk,en,address_int,ROM_out);
    UShifter : Shifter port map(clk,pos_neg,pos_neg2);

    -- Redirection de l'address en sortie
    address <= address_int;

    -- Extension positive de ROM_out
    ROM_out_ext <= "000"&ROM_out;

    -- Extension de signe de Offset
    Offset_ext <= Offset(9)&Offset;

    -- Complément à 2 de ROM_out_ext (sinus impair)
    ROM_out_ext_comp2 <= not(ROM_out_ext)+1;

    -- Sélections de la valeur négative ou positive du sinus
    sin_value <= ROM_out_ext when pos_neg2='0' else ROM_out_ext_comp2 when pos_neg2='1';

    -- Addition de l'offset à la valeur du sinus
    sum_signals <= signed(sin_value) + signed(offset);

    -- Comparaison au range de sortie
    vs_int <=
         conv_std_logic_vector(-1000,11) when signed(sum_signals) <-1000  else
         sum_signals(10 downto 0) when signed(sum_signals) > -1001 and signed(sum_signals) <
1001 else
         conv_std_logic_vector(1000,11);

    vs <= vs_int when clk = '1' and clk'event;

end dataf;
```

5. La machine d'états

Afin de garantir l'aspect synthétisable de la machine de moore, il a été choisi d'utiliser le *template* à deux process.

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

----------------------------------------------------------------------------
-- Entity : Generateur_Points
--
----------------------------------------------------------------------------
entity Generateur_Points is
    port(Offset  : in std_logic_vector(10 downto 0);
         Reset   : in std_logic;
         HD      : in std_logic;
         Vs      : out std_logic_vector(10 downto 0);
         Vc      : out std_logic);
end Generateur_Points;

----------------------------------------------------------------------------
-- Architecture : Generateur_Points
--
----------------------------------------------------------------------------
architecture fsmd of Generateur_Points is
component Dataflow is
    port(clk      : in std_logic;
         Reset    : in std_logic;
         up_dn    : in std_logic;
         offset   : in std_logic_vector(10 downto 0);
         en       : in std_logic;
         pos_neg  : in std_logic;
         vs       : out std_logic_vector(10 downto 0);
         address  : out std_logic_vector(8 downto 0));
end component;

signal up_dn : std_logic;
signal en : std_logic :='1';
signal pos_neg : std_logic;
signal address : std_logic_vector(8 downto 0);

type ETAT is (Up_Pos,Down_Pos,Up_Neg,Down_Neg);
signal state : ETAT :=Up_Pos;

begin
    UDataflow : Dataflow port map(HD,Reset,up_dn,offset,en,pos_neg,vs,address);
    Vc <=pos_neg;

    process1:
    process(HD)
    begin
        if(HD='1' and HD'event) then
            if(reset = '0') then
                state <= Up_Pos;
            else
                case state is
                when Up_Pos =>
                    if(address = 499) then
                        state <= Down_Pos;
                    end if;
                when Down_Pos =>
```

```
            if(address = 1) then
               state <= Up_Neg;
            end if;
         when Up_Neg =>
            if(address = 499) then
               state <= Down_Neg;
            end if;
         when Down_Neg =>
            if(address = 1) then
               state <= Up_Pos;
            end if;
         end case;
      end if;
   end if;
end process process1;

process2 :
process (state)
begin
   case state is
   when Up_Pos =>
      up_dn <= '1';
      pos_neg <= '0';
   when Down_Pos =>
      up_dn <= '0';
      pos_neg <= '0';
   when Up_Neg =>
      up_dn <= '1';
      pos_neg <= '1';
   when Down_Neg =>
      up_dn <= '0';
      pos_neg <= '1';
   end case;
end process process2;
```

6. Validation

Le chronogramme suivant montre le bon fonctionnement du générateur de points pour la fréquence de base de 2 Mhz pour *Hd* ce qui assure bien une période du signal *Vs* d'1 ms.

Puisque le fonctionnement du diviseur de fréquence a été validé auparavant, il n'est pas nécessaire de valider le générateur de points pour plusieurs fréquence de *Hd*.

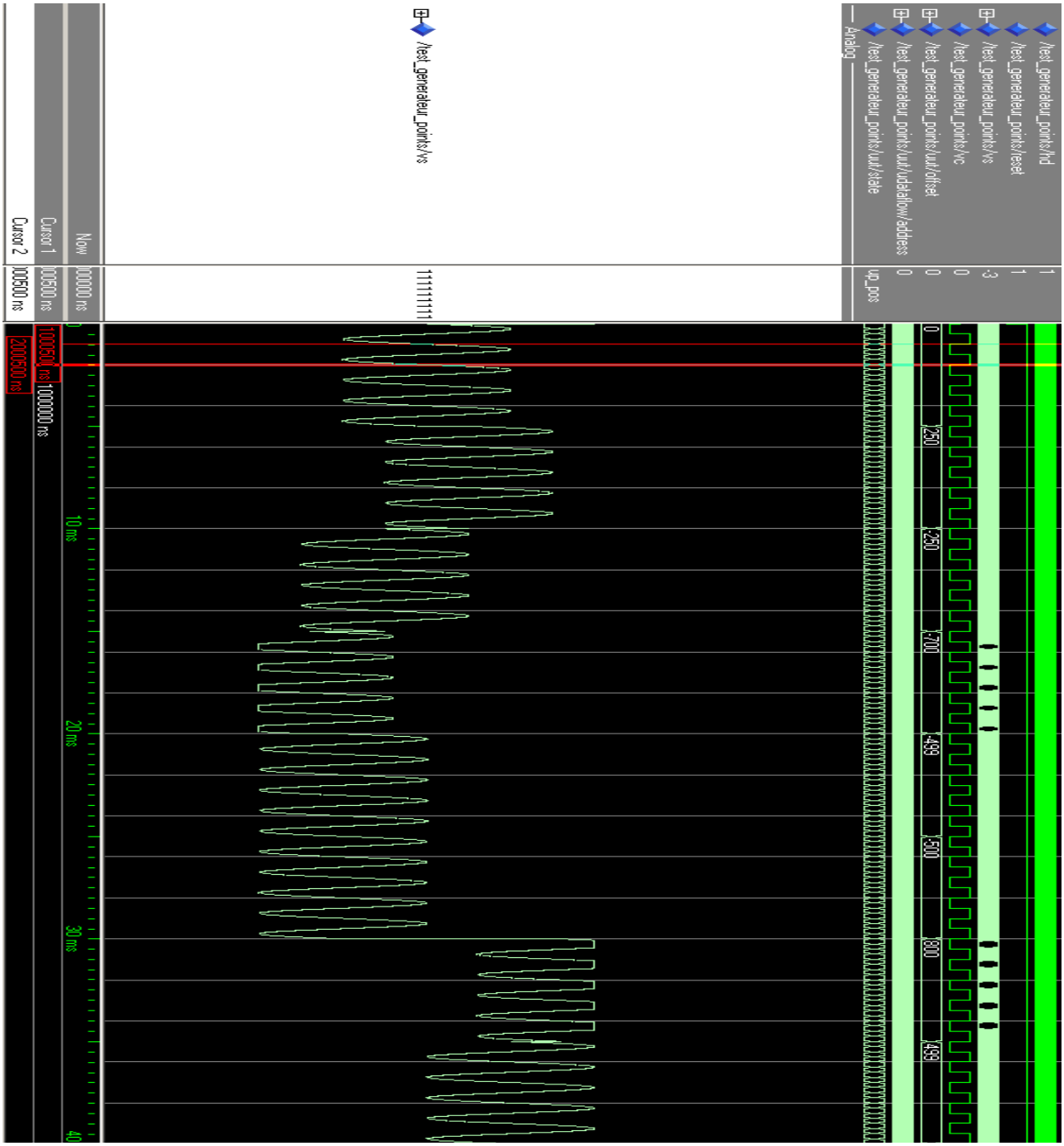

Illustration 10: Validation du générateurs de points

4. Validation du Générateur de signaux

Les trois composants sont regroupés dans le générateur de signaux.

1. Entité

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

--------------------------------------------------------------------------------
-- Entity : Generateur_Programmable
--
--------------------------------------------------------------------------------
entity Generateur_Programmable is
    port( Clk     : in std_logic;
          Reset   : in std_logic;
          Data    : in std_logic_vector(7 downto 0);
          Addr    : in std_logic_vector(1 downto 0);
          Write   : in std_logic;
          Vs      : out std_logic_vector(10 downto 0);
          Vc      : out std_logic);
end Generateur_Programmable;
```

2. Architecture

L'architecture est du type structurel.

```
--------------------------------------------------------------------------------
-- Architecture : Generateur_Programmable
--
--------------------------------------------------------------------------------
architecture struct of Generateur_Programmable is
component Interface_Microprocesseur is
    port(Data   : in std_logic_vector(7 downto 0);
         Addr   : in std_logic_vector(1 downto 0);
         Write  : in std_logic;
         Reset  : in std_logic;
         Offset : out std_logic_vector(9 downto 0);
         Tp     : out std_logic_vector(9 downto 0));
end component;

component Diviseur_Frequence is
    port(clk   : in std_logic;
         Reset : in std_logic;
         Tp    : in std_logic_vector(9 downto 0);
         HD    : out std_logic);
end component;

component Generateur_Points is
    port(Offset  : in std_logic_vector(9 downto 0);
         Reset   : in std_logic;
         HD      : in std_logic;
         Vs      : out std_logic_vector(10 downto 0);
         Vc      : out std_logic);
end component;

signal Tp_int : std_logic_vector(9 downto 0);
signal Offset_int : std_logic_vector(9 downto 0);
signal HD_int : std_logic;

begin
    UInterface_Microprocesseur : Interface_Microprocesseur port
map(Data,Addr,Write,Reset,Offset_int,Tp_int);
    UDiviseur_Frequence : Diviseur_Frequence port map(Clk,Reset,Tp_int,HD_int);
```

```
    Ugenerateur_Points : Generateur_Points port map(Offset_int,Reset,HD_int,Vs,Vc);
end struct;
```

3. Validation

Le chronogramme suivant montre le bon fonctionnement du générateur de signaux programmable avec deux exemples différents (cf Annexe pour le testbench).

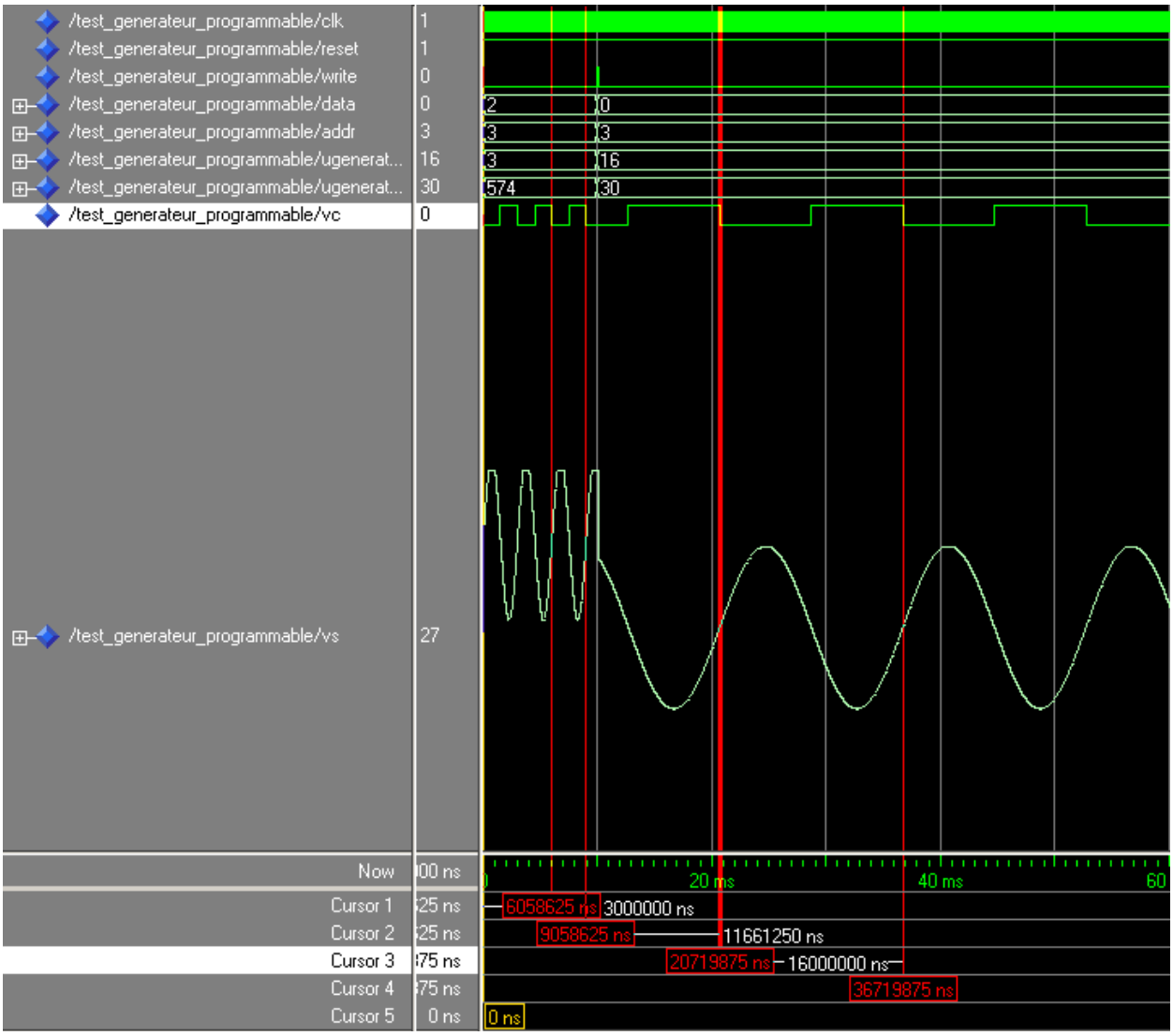

Illustration 11: Validation du générateur de signaux programmable

3.Synthèse sous Leonardo Spectrum

1.Configuration

1. Technologie

La technologie utilisé est le circuit FLEX10K70RC240 avec une vitesse de -4. C'est donc cette librairie qu'il faut charger pour la synthèse.

2. Fichiers d'entrée

Le dossier de travail sera spécifié de la manière suivante :

```
set_working_dir "D:/Mes documents/ELEC3/ASIC/Leonardo/MaxPlus"
```

Les fichiers nécessaires sont appelés ainsi :

```
read -technology "flex10"  {
"D:/Mes documents/ELEC3/ASIC/ROM.vhd"
"D:/Mes documents/ELEC3/ASIC/Shifter.vhd"
"D:/Mes documents/ELEC3/ASIC/Counter.vhd"
"D:/Mes documents/ELEC3/ASIC/Datapath.vhd"
"D:/Mes documents/ELEC3/ASIC/Diviseur_Frequence.vhd"
"D:/Mes documents/ELEC3/ASIC/Generateur_Points.vhd"
"D:/Mes documents/ELEC3/ASIC/Generateur_Programmable.vhd"
"D:/Mes documents/ELEC3/ASIC/Interface_Microprocesseur.vhd" }
```

3. Pré-optimisation

Il est laissé à Leonardo le choix du codage des états pour la machine d'états du générateur de points.

```
pre_optimize -common_logic -unused_logic -boundary -xor_comparator_optimize
pre_optimize -extract
```

Le résultat pour le codage des états est le suivant :

```
Encodings for ETAT values
        value    ETAT[1-0]
=========================
         Up_Pos  00
       Down_Pos  01
         Up_Neg  10
       Down_Neg  11
```

Ce codage est assez cohérent si on se réfère au diagramme d'états puisqu'il n'y a qu'une seule transition par état (schéma dit carré).

2.Analyse du schéma niveau RTL

Le schéma RTL au plus haut niveau est le suivant et est bien cohérent avec celui désigné dans les spécifications

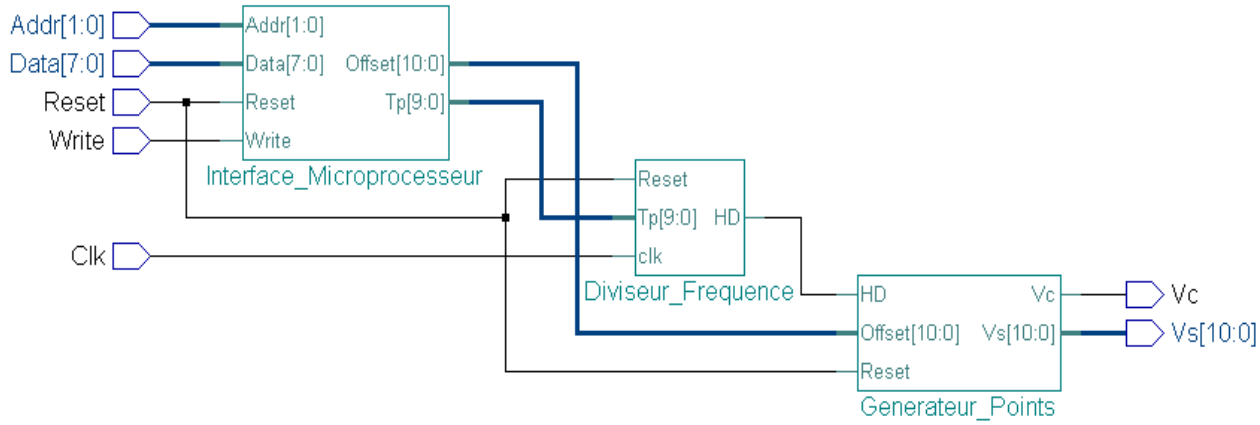

Illustration 12: Schéma RTL haut niveau

1. RTL de l'interface micro-processeur

Celle-ci a été optimisée puisque dans le code VHDL, la taille des registres de stockage interne était de 16 pour *Offset* et *Tp*. Ceci était inutile car seulement 10 bits sont utilisés pour *Tp* et 11 pour *Offset*. De plus le transfert des 3 derniers bits d'*Offset* ont été directement poussés en sortie, ce qui permet de gagner 3 registres. D'où 39 registres sur le schéma suivant et non 42.

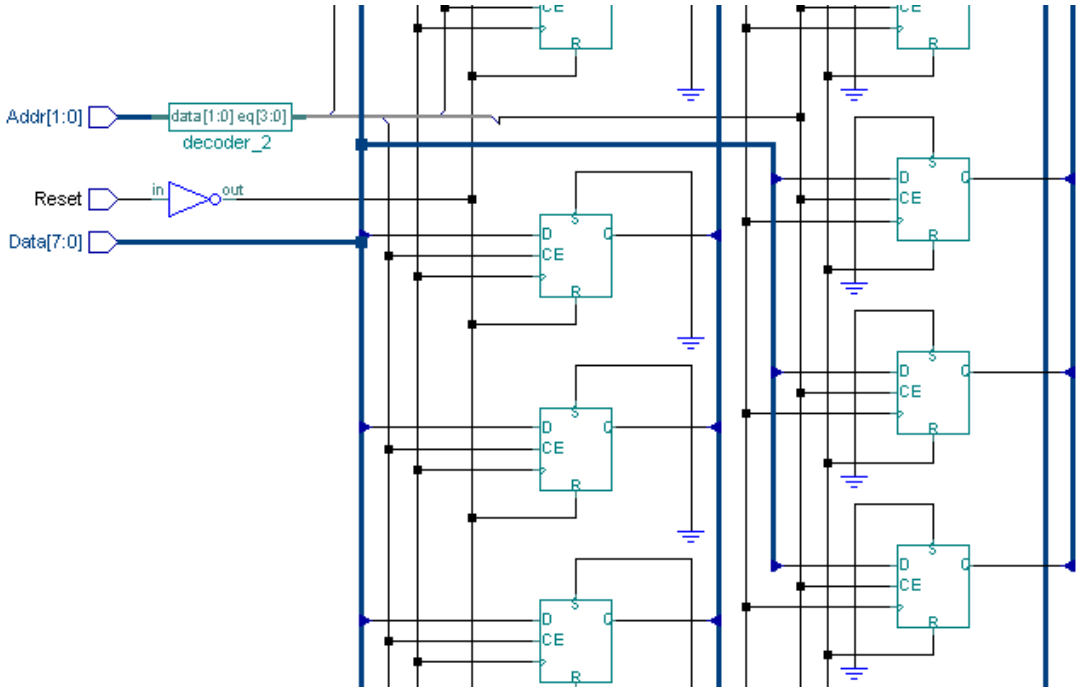

Illustration 13: Transfert direct des 3 derniers bits d'offset vers les registres de sortie

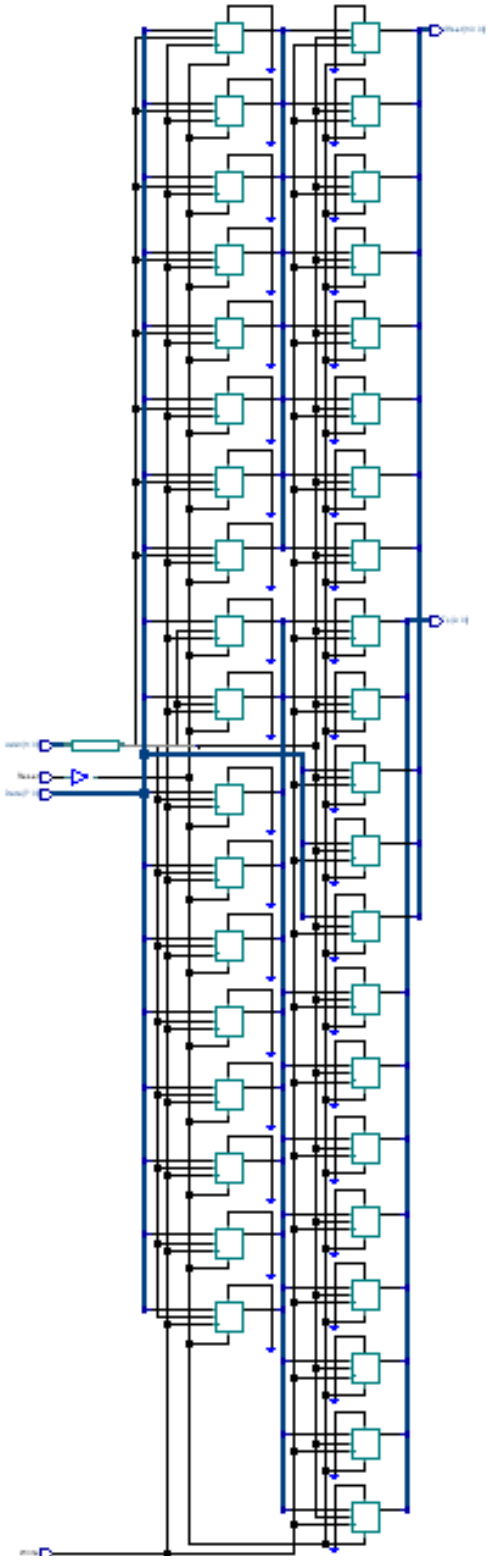

Illustration 14: Schéma RTL de l'interface micro processeur

2. RTL du diviseur de fréquence

Le schéma RTL montre l'utilisation d'un compteur avec *clear* et d'un registre de sortie pour le signal *Hd*.

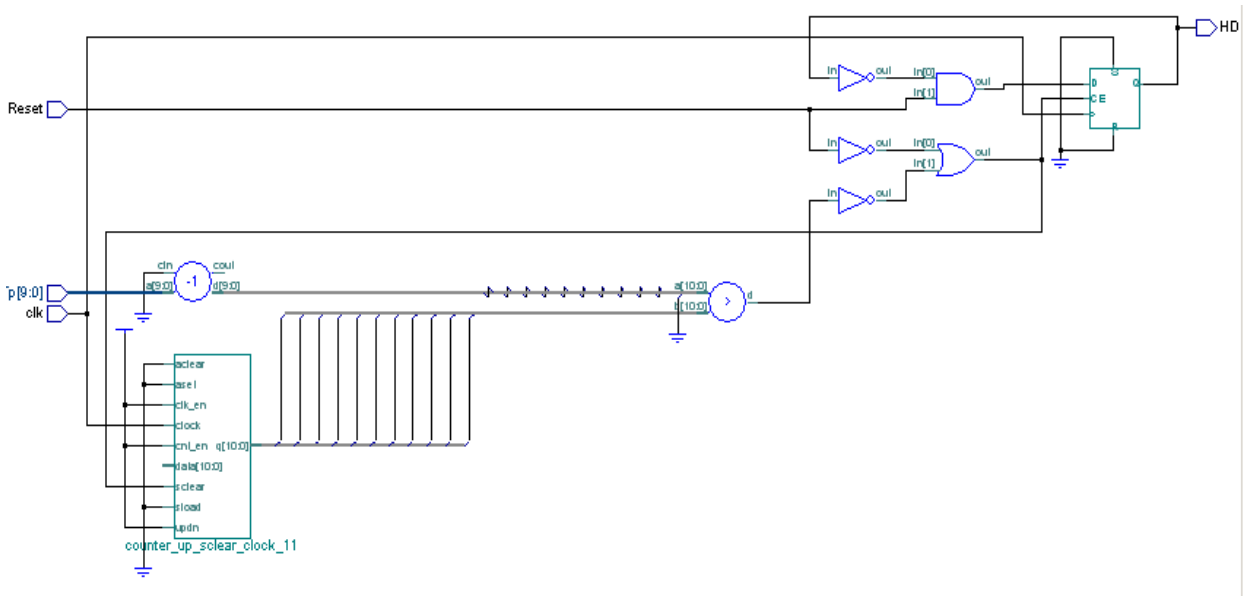

Illustration 15: Schéma RTL du diviseur de fréquence

Le comportement du compteur est le suivant. Un additionneur +1 et des registres pour stocker la valeur courante du compteur. Au +1 est associé un AND qui permet la remise à zéro du compteur de manière synchrone.

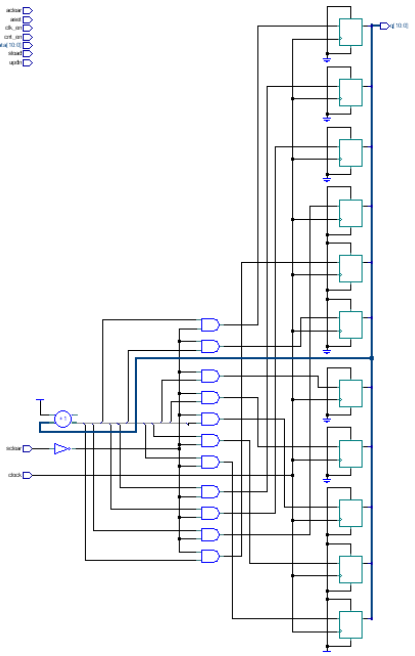

Illustration 16: Compteur avec remise à zéro synchrone

3. RTL du générateur de points

Sur le schéma suivant il est possible de discerner la machine d'états (les 2 registres) et la logique de décision (des sorties et d'états). Le chemin de données est présenté comme un composant conformément aux spécifications.

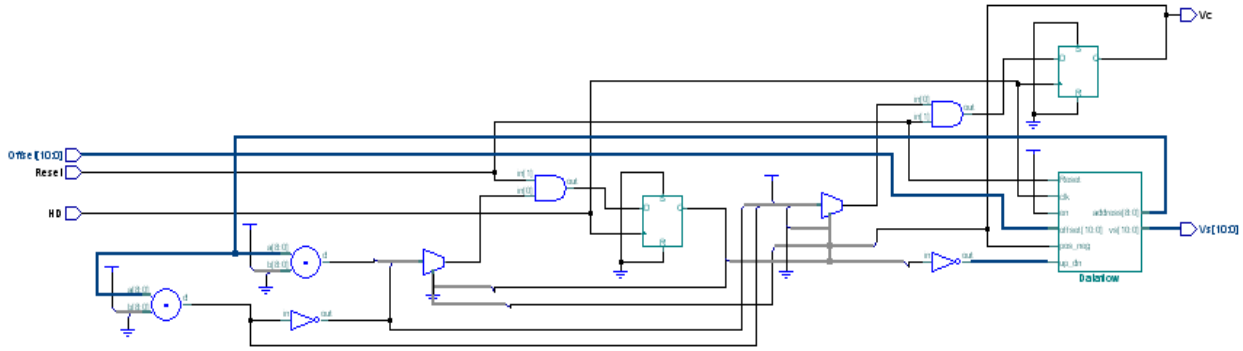

Illustration 17: Schéma RTL du générateur de points

4. RTL du chemin de données

Dans ce schéma on retrouve tous les composants essentiels : le registre à décalages, la ROM et le compteur.

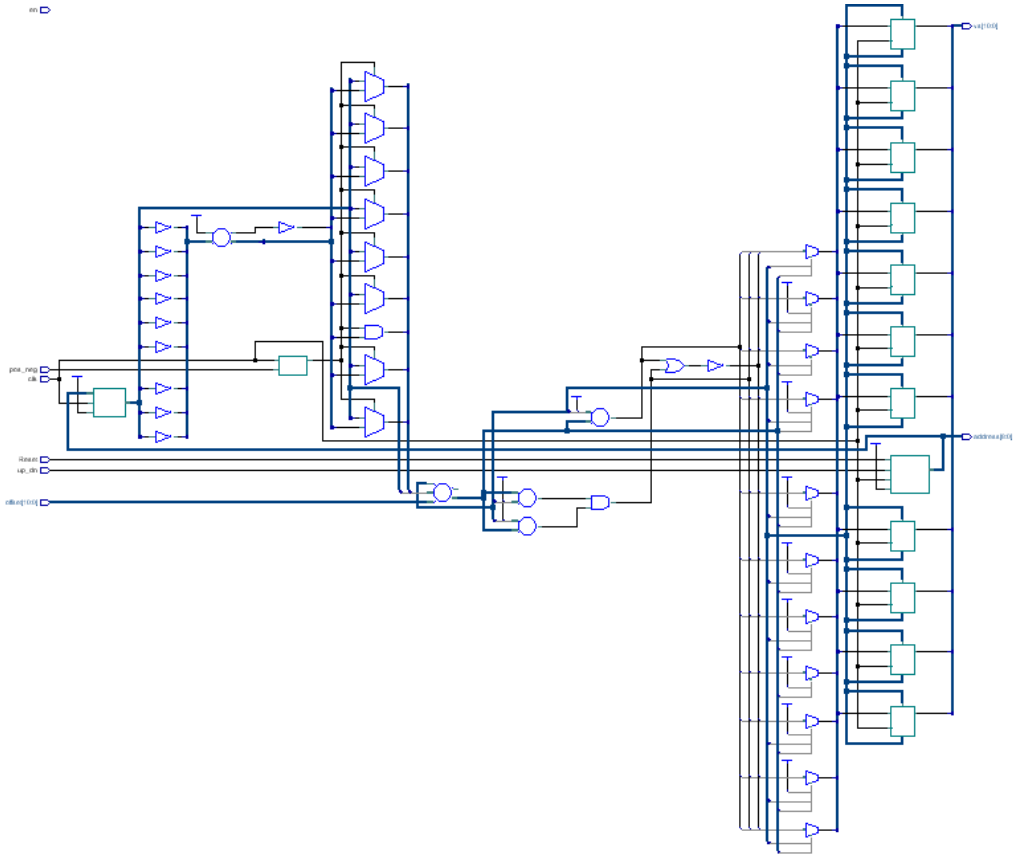

Illustration 18: Schéma RTL du chemin de données

La sortie *Vs* est bufferisée grâce à 11 registres.

1. RTL de la ROM

Le RTL de la ROM montre une bufferisation du signal d'adresse conformément au VHDL. La ROM en elle même est constituée des valeurs chargées dans des LUTs.

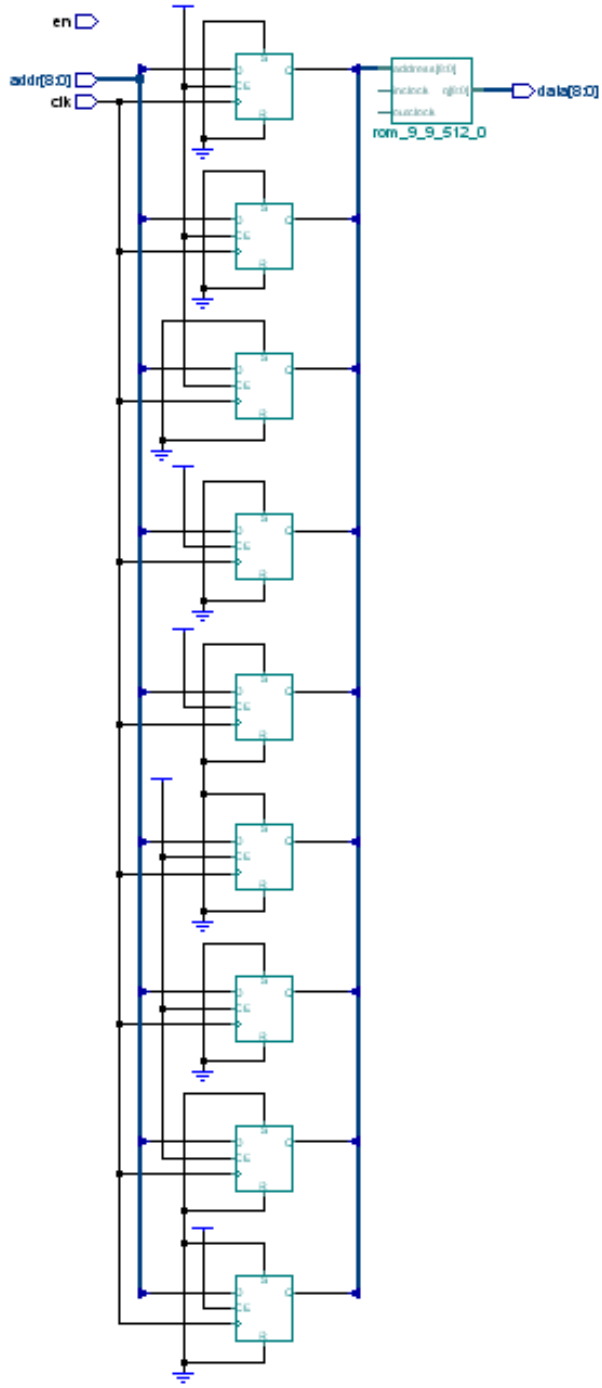

Illustration 19: Schéma RTL de la mémoire

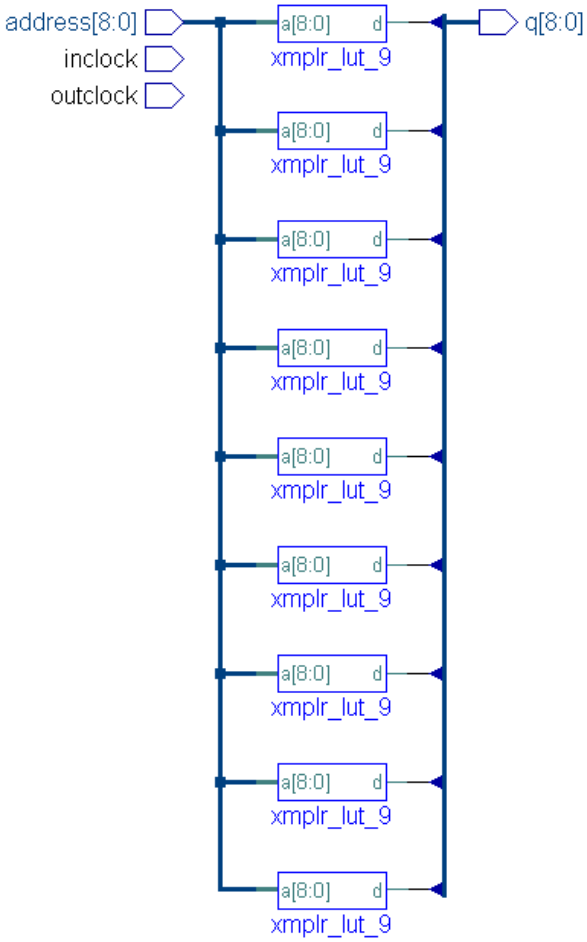

Illustration 20: Utilisation des LUTs pour la mémoire

2. RTL du registre à décalages

Puisque le registre à décalage sert juste de tampon pour un top d'horloge, il s'agit juste d'un simple registre.

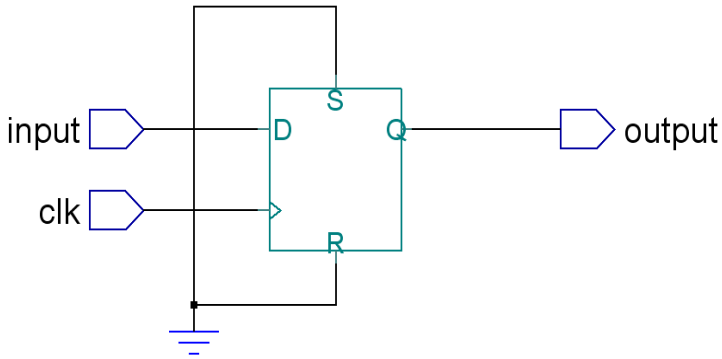

Illustration 21: Schéma RTL du registre à décalages

3. RTL du compteur d'adresse

La synthèse a bien conduit à un compteur dé-compteur avec remise à zéro synchrone.

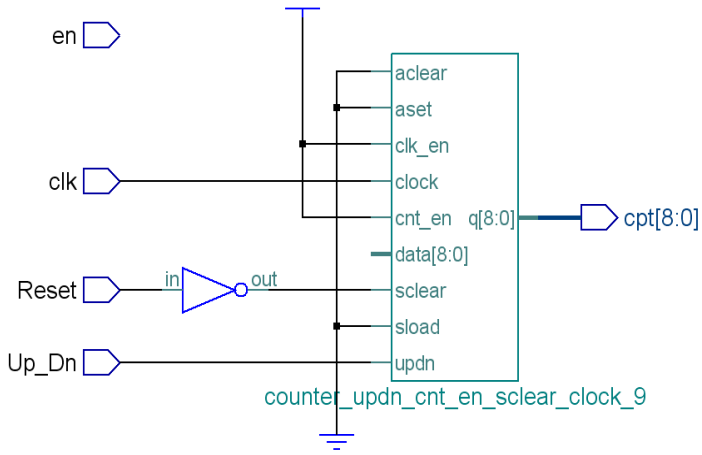

Illustration 22: Inférence du compteur dé-compteur par l'outil de synthèse

3.Optimisation et ajout des contraintes

1. Contraintes d'horloge

Les contraintes suivantes sont définies pour les signaux d'horloge :

```
set_clock -name .work.Generateur_Programmable.struct.Clk -clock_cycle "250.000000"
set_clock -name .work.Generateur_Programmable.struct.Clk -pulse_width "125.000000"
```

Illustration 23: Signal d'horloge

2. Optimisations

Il est demandé au logiciel de synthèse d'effectuer une optimisation avec l'effort maximal (4 passes nécessaires).

```
set part EPF10K10LC84
set process 3
optimize .work.Generateur_Programmable.struct -target flex10 -chip -auto -effort standard -hierarchy auto
optimize_timing .work.Generateur_Programmable.struct
```

4.Analyse du schéma technologique et des rapports

1. Schéma technologique

Le schéma technologique montre la transformation des composants du niveau RTL en LUTs ou en Carry. Sur le schéma suivant, la partie de gauche correspond à la transformation technologique de l'interface micro-processeur, la partie de droite correspond au chemin de données. Il est difficile de retrouver la partie correspondant au diviseur de fréquence.

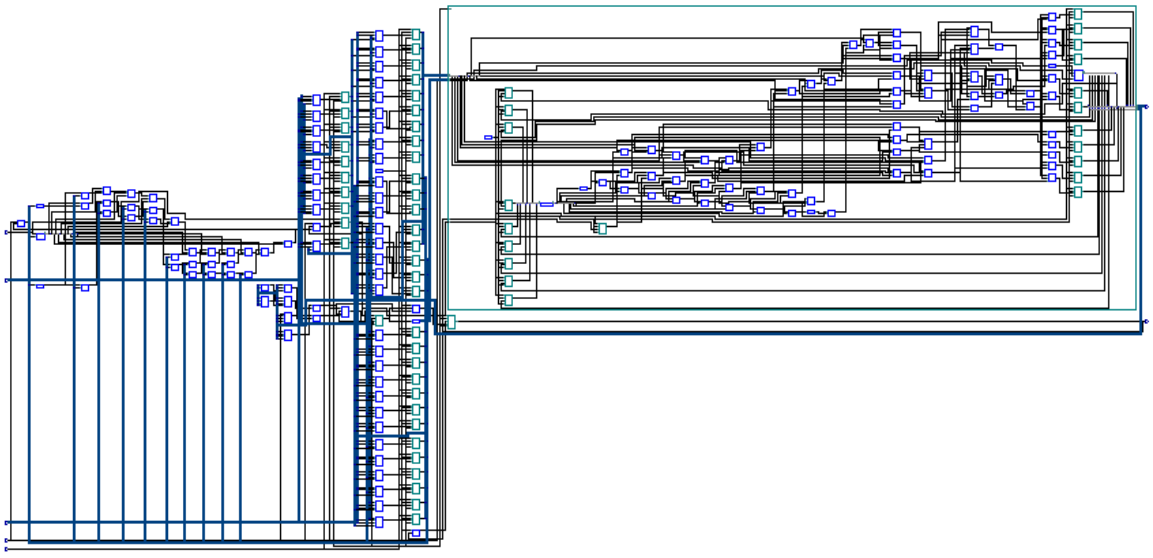

Illustration 24: Schéma technologique du générateur programmable

2. Chemin critique

Le chemin critique se trouve entre le transfert de *Tp* depuis l'interface micro-processeur et la génération du signal *HD* en sortie du diviseur de fréquence. Le chemin critique est d'environ 16.8 ns.

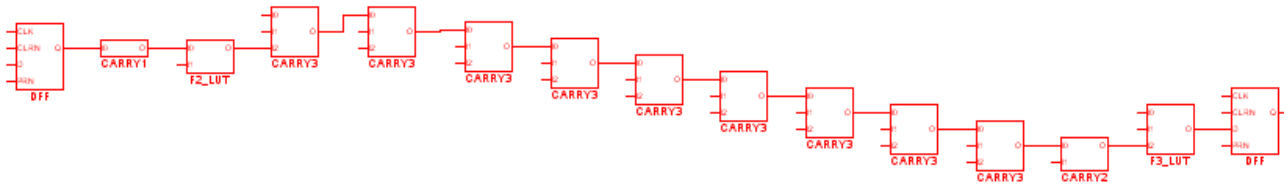

Illustration 25: Chemin critique de Tp jusqu'à Hd

3. Rapport de surface

```
*********************************************************

Cell: Generateur_Programmable    View: struct    Library: work

*********************************************************


 Cell                  Library  References     Total Area

CARRY1                 flex10        1 x      1      1 CARRYs
CARRY2                 flex10       10 x      1     10 CARRYs
CARRY3                 flex10        9 x      1      9 CARRYs
CASCADE2               flex10        2 x      1      2 CASCADEs
                                               1      2 LCs
DFF                    flex10       41 x      1     41 DFFs
                                               1     41 LCs
DFFE                   flex10        1 x      1      1 DFFs
                                               1      1 LCs
Dataflow               work          1 x      7      7 CASCADEs
                                               1      1 VCC
                                              21     21 DFFs
                                            4608   4608 Memory Bits
                                              29     29 CARRYs
                                              62     62 LCs
F1_LUT                 flex10        2 x      1      2 F1_LUT
F2_LUT                 flex10       12 x      1     12 LCs
F3_LUT                 flex10        3 x      1      3 LCs
F4_CAS                 flex10        2 x      1      2 CASCADEs
                                               1      2 LCs
F4_LUT                 flex10       42 x      1     42 LCs
VCC                    flex10        1 x      1      1 VCC
lpm_counter_11_292_0   OPERATORS     1 x     11     11 CARRYs
                                              11     11 LCs
```

```
 Number of ports :                       25
 Number of nets :                       170
 Number of instances :                  128
 Number of references to this view :      0

Total accumulated area :
 Number of CARRYs :                      60
 Number of CASCADEs :                    11
 Number of DFFs :                        63
 Number of F1_LUT :                       1
 Number of LCs :                        142
 Number of Memory Bits :               4608
 Number of VCC :                          2
 Number of accumulated instances :      218
************************************************
Device Utilization for EPF10K70RC240
************************************************
Resource                Used    Avail   Utilization
------------------------------------------------
IOs                     25      189      13.23%
LCs                     142     3744      3.79%
DFFs                    63      4096      1.54%
Memory Bits             4608    18432    25.00%
CARRYs                  60      3744      1.60%
CASCADEs                11      3744      0.29%

------------------------------------------------
```

Il est a noté l'utilisation importante de zones mémoire (1/4).

4. Rapport de délai

```
                      Clock Frequency Report

     Clock                : Frequency
       ----------------------------------

     Clk                  : 47.1 MHz
     Write                : 102.9 MHz
     HD_int               : 22.5 MHz


                      Critical Path Report


Critical path #1, (path slack = 205.5):

NAME                                                    GATE              ARRIVAL
```

```
LOAD
  ------------------------------------------------------------------------------------------
-----------------
  Ugenerateur_Points_UDataflow/UROM_ix0_ix47/q(0)                GENERIC_BLACK_BOX   0.00
0.00 up           0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix23/O            CARRY1       0.60   0.60 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix27/O            CARRY2       0.60   1.20 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix31/O            CARRY2       0.60   1.80 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix35/O            CARRY2       0.60   2.40 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix39/O            CARRY2       0.60   3.00 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix43/O            CARRY2       0.60   3.60 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix47/O            CARRY2       0.60   4.20 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix51/O            CARRY2       0.60   4.80 up
0.00
  Ugenerateur_Points_UDataflow/result_inc_30_ix55/O            CARRY2       0.60   5.40 up
0.00
  Ugenerateur_Points_UDataflow/ix80/Y                          LCELL        4.92  10.32 up
0.00
  Ugenerateur_Points_UDataflow/sin_value(9)/O                  F2_LUT       4.92  15.24 up
0.00
  Ugenerateur_Points_UDataflow/modgen_add_31_ix76/O            CARRY3       0.60  15.84 up
0.00
  Ugenerateur_Points_UDataflow/modgen_add_31_ix80/O            CARRY3       0.60  16.44 up
0.00
  Ugenerateur_Points_UDataflow/modgen_add_31_ix82/O            F3_LUT       4.92  21.36 up
0.00
  Ugenerateur_Points_UDataflow/nx38/O                          F4_CAS       6.24  27.60 up
0.00
  Ugenerateur_Points_UDataflow/nx39/O                          CASCADE2     1.32  28.92 up
0.00
  Ugenerateur_Points_UDataflow/nx45/O                          F4_LUT       4.92  33.84 up
0.00
  Ugenerateur_Points_UDataflow/NOT_a(1)/O                      CASCADE2     1.32  35.16 up
0.00
  Ugenerateur_Points_UDataflow/nx47/O                          F3_LUT       4.92  40.08 up
0.00
  Ugenerateur_Points_UDataflow/reg_vs(9)/D                     DFF          0.00  40.08 up
0.00
  data arrival time                                                          40.08


  data required time  (default specified - setup time)                       245.56
  ------------------------------------------------------------------------------------------
-----------------
  data required time                                                         245.56
  data arrival time                                                          40.08
                                                                           ----------
  slack                                                                    205.48
```

Un slack positif garantit le respect des contraintes temporelles. L'horloge maximale du circuit étant de 47 MHz.

5.Analyse du Floorplan

Dans un premier temps il faut générer le fichier edif :

```
auto_write -format EDIF Generateur_Programmable.edf
```

Ensuite il faut lancer le placement routage :

```
place_and_route Generateur_Programmable.edf -target flex10 -exe_path C:/maxplus2 -part
EPF10K70RC240 -speed_grade 4  -max_ta_reg -max_no_timing -max_no_cliques -max_no_pins
-ba_format VHDL
```

Le plan de placement obtenu est le suivant. Au centre on peut distinguer des cellules dédicacées à la ROM (bleu clair). On retrouve aussi cascadés les différents compteurs (adresse sur la gauche au centre, du diviseur de fréquence sur la gauche,...)

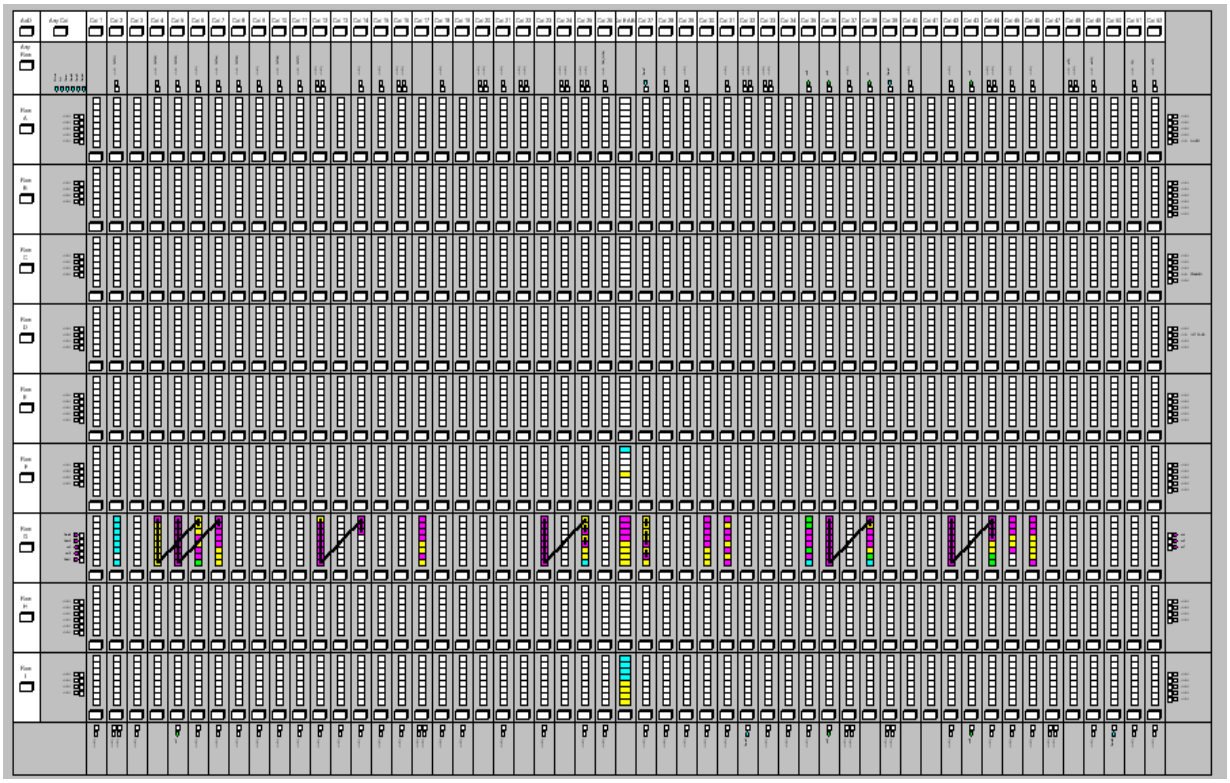

Illustration 26: Floorplan

4.Simulation post-routage

1.Génération du VHO sous MAX + PLUS II et simulation post-routage

Le VHO est récupéré de la compilation du fichier edif sous MAX + PLUS II. La librairie flex10K est créée dans un nouveau projet sous ModelSim regroupant toutes les primitives et les délais de la technologie. Le VHO et un fichier de test sont ajoutés à ce projet. La

simulation aboutit au résultat suivant.

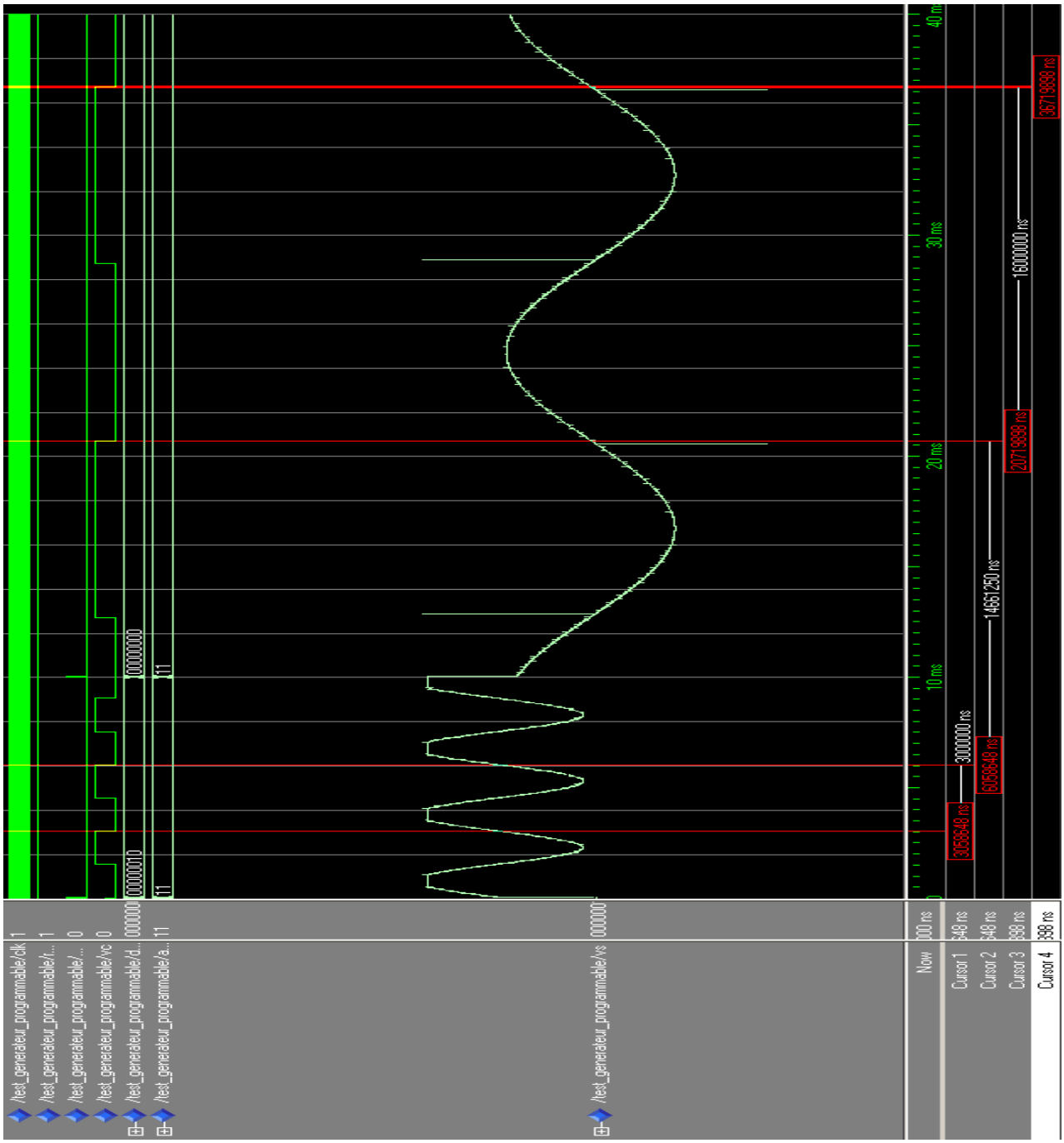

Illustration 27: Chronogramme de la simulation post routage

Il est à noter la présence de *spurious* très importants sur cette simulation. Ceux-ci sont dus à la mise en cascade de plusieurs multiplexeurs pour la génération de *Vs*. Cependant le comportement du composant est bien respecté au niveau des précisions sur les temps et les formes.

Afin de corriger le problème des *spurious*, nous allons utiliser un autre outil de placement routage : Quartus.

2. Génération du VHO sous Quartus et simulation post-routage

Après une correction du testbench utilisé (signal d'horloge initialisé à 1 au lieu de 0), la simulation post routage issu de Quartus donne le résultat suivant remplissant toutes les spécifications demandées.

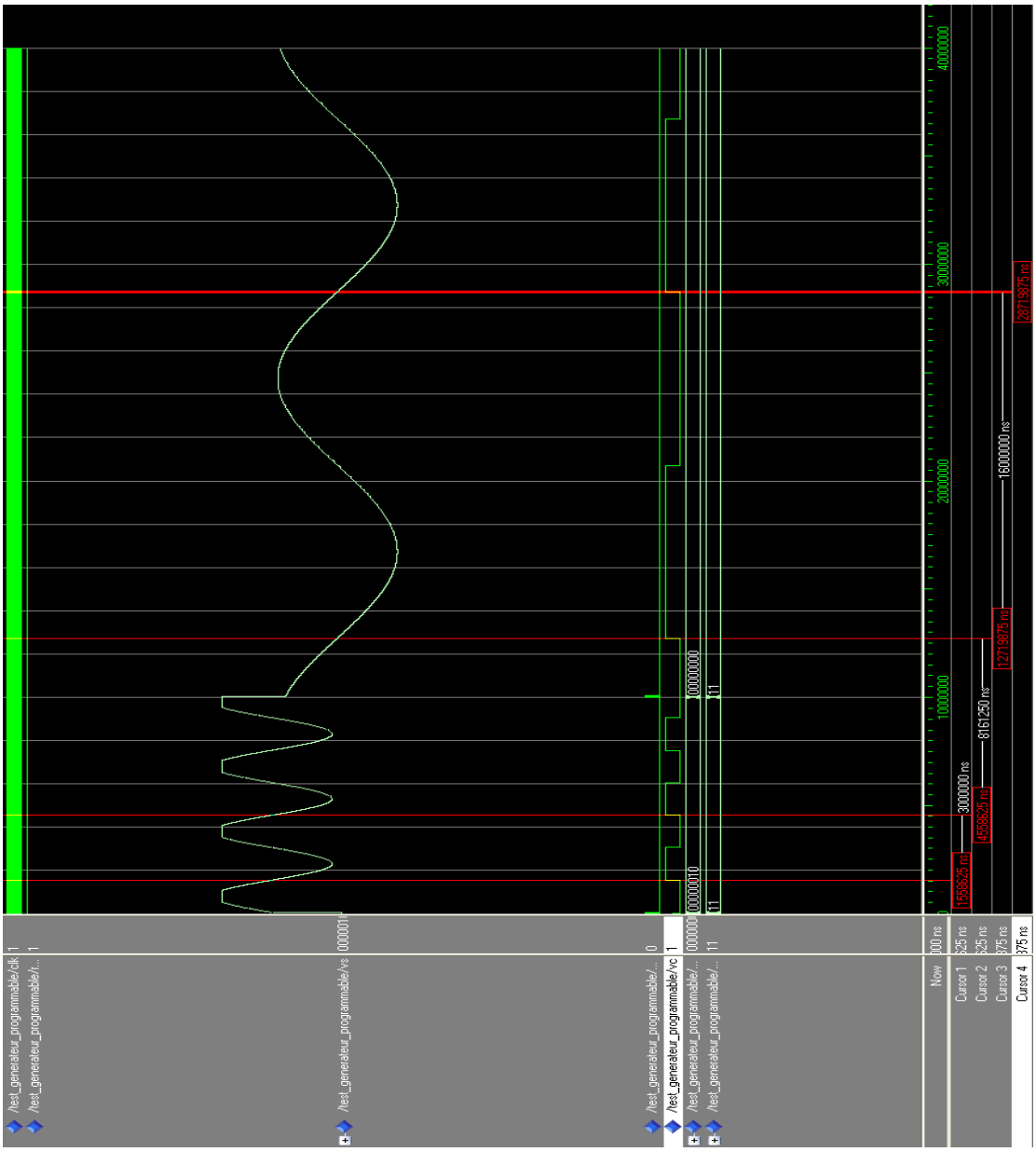

Illustration 28: Chronogramme de la simulation pst routage issu de Quartus

5.Simulation réelle et tests opérationnels

1.Attribution du plan de brochage

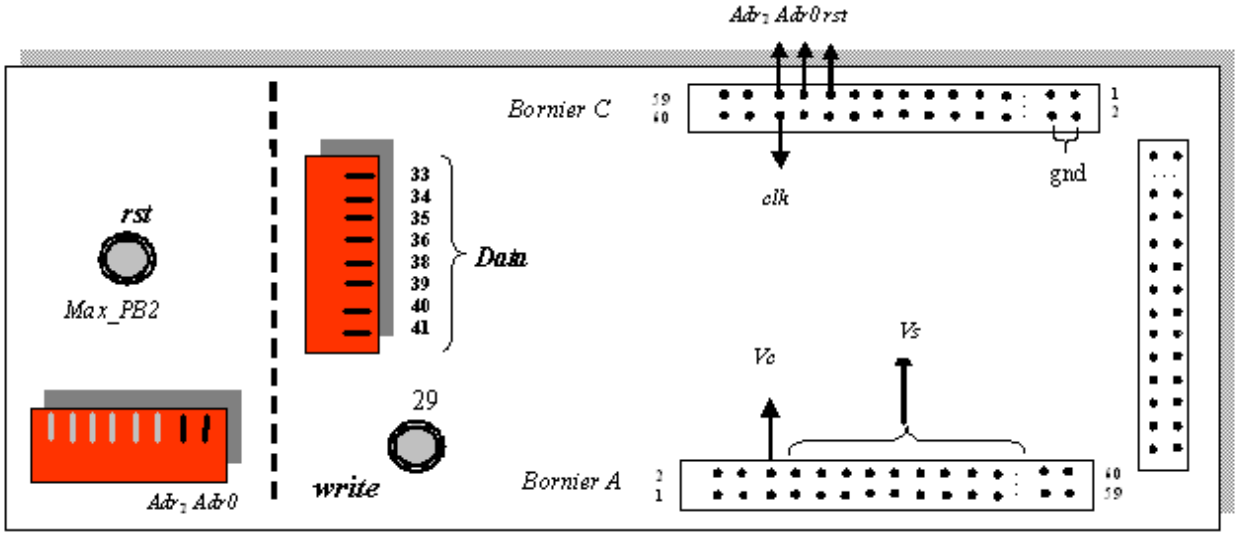

Illustration 29: Plan de brochage sur le circuit de test

Le plan de brochage est inséré dans le script de la manière suivante :

```
set ATOI DE LE REMPLIR MERCI
```

2.Téléchargement de la configuration sur le composant

Quartus est utilisé pour le téléchargement du programme. Une première vérification du fonctionnement consiste à capturer un bit de Vs et regarder si le signal évolu.

3.Simulation réelle

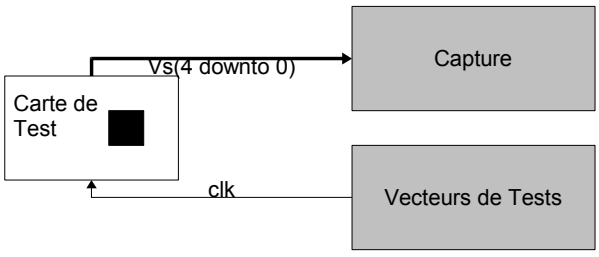

Illustration 30: Environnement de simulation

La reproduction de la première séquence du testbench présent en annexe est effectuée à partir des boutons poussoirs et borniers.

La capture des signaux aboutit aux résultats suivants :

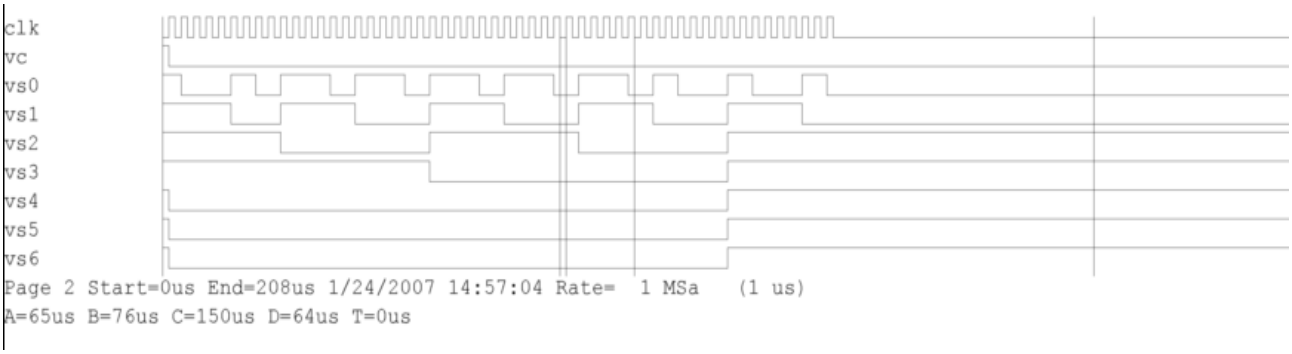

Illustration 31: Capture des signaux

Afin de vérifier le bon fonctionnement il faut rechercher dans la ROM l'évolution de Vs(6 downto 0) décrite sur le schéma. Une seule séquence a été trouvée : le parcours se faisant de r(293) à r(279).

```
r(279):="110000000";
r(280):="110000001";
r(281):="110000010";
r(282):="110000011";
r(283):="110000100";
r(284):="110000101";
r(285):="110000110";
r(286):="110000111";
r(287):="110001000";
r(288):="110001001";
r(289):="110001010";
r(290):="110001011";
r(291):="110001100";
r(292):="110001101";
r(293):="110001110";
```

Enfin la fréquence minimale d'un bit de Vs est de 3 coups d'horloge, ce qui correspond bien à la première configuration du testbench.

6.Conclusion

La conception numérique demande une rigueur et une organisation importante du travail. Le découpage du projet en sous blocs est essentiel. La conception et validation de chacun assure un bon fondement de l'architecture finale dans une approche bottom up.

La conception en suivant des modèles de composants est elle aussi indispensable pour garantir une synthèse efficace. L'agencement de ces composants pour créer un chemin de données propre est très important et rendra d'autant plus simple la machine d'états le pilotant.

Ce projet fut donc une très bonne approche de la conception numérique sur FPGA. Il nous a donné une base utile pour de futurs développements en entreprise.

Annexes

Annexe 1 : Testbench de l'interface microprocesseur

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

--------------------------------------------------------------------------------
-- Entity : test_Interface_Microprocesseur
--
--------------------------------------------------------------------------------
entity test_Interface_Microprocesseur is
end test_Interface_Microprocesseur;

--------------------------------------------------------------------------------
-- Architecture : test_Interface_Microprocesseur
--
--------------------------------------------------------------------------------
architecture testbench of test_Interface_Microprocesseur is
component Interface_Microprocesseur is
    port(Data   : in std_logic_vector(7 downto 0);
         Addr   : in std_logic_vector(1 downto 0);
         Write  : in std_logic;
         Reset  : in std_logic;
         Offset : out std_logic_vector(9 downto 0);
         Tp     : out std_logic_vector(9 downto 0));
end component;

signal Data   : std_logic_vector(7 downto 0) := "01011011";
signal Addr   : std_logic_vector(1 downto 0) := "00";
signal Write  : std_logic := '1';
signal Reset  : std_logic := '1';
signal Offset : std_logic_vector(9 downto 0);
signal Tp     : std_logic_vector(9 downto 0);

begin
    UUInterface_Microprocesseur : Interface_Microprocesseur port map(
      Data, Addr, Write, Reset, Offset, Tp);

    process
```

```
    begin
        write <= not write;
        wait for 1 ns;
    end process;

    Data <= "00000001" after 2 ns, "00111110" after 4 ns, "00011000" after 6 ns;
    Addr <= "01" after 2 ns, "10" after 4 ns, "11" after 6 ns, "00" after 10 ns;
    Reset <= '0' after 8 ns, '1' after 10 ns;

end testbench;
```

Annexe 2 : Testbench du diviseur de fréquence

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

--------------------------------------------------------------------------------
-- Entity : test_Diviseur_Frequence
--
--------------------------------------------------------------------------------
entity test_Diviseur_Frequence is
end test_Diviseur_Frequence;

--------------------------------------------------------------------------------
-- Architecture : test_Diviseur_Frequence
--
--------------------------------------------------------------------------------
architecture testbench of test_Diviseur_Frequence is
component Diviseur_Frequence is
    port(clk    : in std_logic;
         Reset : in std_logic;
         Tp     : in std_logic_vector(9 downto 0);
         HD     : out std_logic);
end component;

signal clk    : std_logic := '1';
signal reset : std_logic := '1';
signal Tp     : std_logic_vector(9 downto 0) :="0000000001";
signal HD     : std_logic;

begin
    UUDiviseur_Frequence : Diviseur_Frequence port map(
      clk,reset,Tp,HD);

    process
    begin
        clk <= not clk;
        wait for 0.125 us;
    end process;

    Tp <= "0000000011" after 10 us, "0000001100" after 30 us, "0000001111" after 80 us;

end testbench;
```

Annexe 3 : Testbench du générateur de points

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

----------------------------------------------------------------------------
-- Entity : Test_Generateur_Points
--
----------------------------------------------------------------------------
entity Test_Generateur_Points is
end Test_Generateur_Points;

----------------------------------------------------------------------------
-- Architecture : Test_Generateur_Points
--
----------------------------------------------------------------------------
architecture testbench of Test_Generateur_Points is
component Generateur_Points is
    port(Offset  : in std_logic_vector(10 downto 0);
         Reset   : in std_logic;
         HD      : in std_logic;
         Vs      : out std_logic_vector(10 downto 0);
         Vc      : out std_logic);
end component;

signal Offset  : std_logic_vector(10 downto 0) := conv_std_logic_vector(0,11);
signal Reset   : std_logic:='0';
signal HD      : std_logic :='0';
signal Vs      : std_logic_vector(10 downto 0);
signal Vc      :std_logic;

begin
    UUT : Generateur_Points port map(Offset,Reset,HD,Vs,Vc);
    process
    begin
        HD <= not HD;
        wait for 0.25 us;
    end process;

    Reset <= '1' after 1 us;
    Offset <= conv_std_logic_vector(250,11) after 5 ms,
              conv_std_logic_vector(-250,11) after 10 ms,
```

```
            conv_std_logic_vector(-700,11) after 15 ms,
            conv_std_logic_vector(-499,11) after 20 ms,
            conv_std_logic_vector(-500,11) after 25 ms,
            conv_std_logic_vector(800,11) after 30 ms,
            conv_std_logic_vector(499,11) after 35 ms,
            conv_std_logic_vector(500,11) after 40 ms;
end testbench;
```

Annexe 4 : Testbench du générateur programmable

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

----------------------------------------------------------------------------
-- Entity : test_Generateur_Programmable
--
----------------------------------------------------------------------------
entity test_Generateur_Programmable is
end test_Generateur_Programmable;

----------------------------------------------------------------------------
-- Architecture : test_Generateur_Programmable
--
----------------------------------------------------------------------------
architecture testbench of test_Generateur_Programmable is
component Generateur_Programmable is
    port( Clk     : in std_logic;
          Reset   : in std_logic;
          Data    : in std_logic_vector(7 downto 0);
          Addr    : in std_logic_vector(1 downto 0);
          Write   : in std_logic;
          Vs      : out std_logic_vector(10 downto 0);
          Vc      : out std_logic);
end component;
signal clk : std_logic :='0';
signal Reset : std_logic :='1';
signal Vs :std_logic_vector(10 downto 0);
signal Write : std_logic:='0';
signal Vc : std_logic;
signal Data : std_logic_vector(7 downto 0):="00011111";
signal Addr : std_logic_vector(1 downto 0) :="00";

--for UGenerateur_Programmable : Generateur_Programmable use entity work.generateur;

begin
        UGenerateur_Programmable : Generateur_Programmable port
map(Clk,Reset,Data,Addr,Write,Vs,Vc);
    process
    begin
        clk <= '0';
```

```
        wait for 0.125 us;
        clk <= '1';
        wait for 0.125 us;
    end process;
    Write <= '1' after 1 us, '0' after 10 us,
             '1' after 20 us, '0' after 30 us,
             '1' after 40 us, '0' after 50 us,
             '1' after 60 us, '0' after 70 us,

             '1' after 10.001 ms, '0' after 10.01 ms,
             '1' after 10.02 ms, '0' after 10.03 ms,
             '1' after 10.04 ms, '0' after 10.05 ms,
             '1' after 10.06 ms, '0' after 10.07 ms;

    Addr  <= "00" after 0.5 us,
             "01" after 12.5 us,
             "10" after 34.5 us,
             "11" after 56.5 us,

             "00" after 10.0005 ms,
             "01" after 10.0125 ms,
             "10" after 10.0345 ms,
             "11" after 10.0565 ms;

    Data  <= "00000011" after 0.5 us,
             "00000000" after 12.5 us,
             "00111110" after 34.5 us,
             "00000010" after 56.5 us,

             "00010000" after 10.0005 ms,
             "00000000" after 10.0125 ms,
             "00011110" after 10.0345 ms,
             "00000000" after 10.0565 ms;

end testbench;
```