Rapport de Projet VHDL :

Réception du signal numérique à photo-détection

Effectué par : CHANAA Othmane, EL KASSARI Zakariae, BENLEMLIH Abdellah

Encadré par : Mr. CHERKAOUI Abdeljabbar

Niveau : GSEA2

Année universitaire : 2019/2020

Table des matières

I. Introduction :

La transmission à fibre optique devient de plus en plus fréquente dans la société moderne. Autrefois, elle s'est limitée au but gouvernemental ou commercial, mais aujourd'hui, le système de la transmission à fibre optique peut être utilisé dans la famille personnelle en raison de la demande croissante sur les connexions à haute bande passante pour les diverses applications industrielles et résidentielles.

Un système de communication à fibre optique se compose de trois composants : l'émetteur optique, la jarretière optique et le récepteur optique. L'émetteur optique convertit le signal électrique en signal optique ; la jarretière optique transmet le signal optique de l'émetteur optique au récepteur optique ; et le récepteur optique reconvertit le signal optique en signal électrique. La plupart des fibres optiques sont fabriquées de silice ou de sable, les matières premières sont abondantes par rapport au cuivre. Les fibres optiques de longueur d'environ 43 km peuvent être produites avec seulement quelques livres de verre. Les fibres optiques peuvent servir d'intermédiaire pour la télécommunication et les réseaux, car elles sont flexibles et peuvent être regroupées en câbles. Elles sont particulièrement favorables aux communications à longue distance, parce qu'il n'y a qu'un peu d'atténuation lors de la transmission par rapport aux câbles électriques en cuivre. La fibre optique est supérieure aux conducteurs métalliques en tant que ligne T/N pour les signaux en raison de sa bande passante élevée, de sa faible atténuation, de son interférence, de ses faibles coûts et de son poids léger. C'est pourquoi elle est utilisée dans le domaine des télécommunications.

Grâce à la fibre optique, la rapidité et la facilité de navigation sur le net sont amplifiées. On verra son utilité à travers les téléchargements de fichiers, la qualité d'image, la visualisation de vidéos, la fluidité. Sa performance se qualifie aussi bien côté TV, notamment par le système HD et la 3D.

La fibre optique n'a aucune limite en matière de débit, il serait donc possible que les échanges de flux numériques multimédias ainsi que la domotique soient favorisés.

La mise en place de l'infrastructure de la fibre optique nécessite un investissement très important pour les FAI (Fournisseurs d'accès à Internet).

L'ADSL permet certes des débits élevés en utilisant une infrastructure déjà existante, cependant, les débits élevés sont réservés aux foyers proches d'un NRA (Noeud de Raccordement d'Abonnés): le débit est asymétrique (il est plus faible en émission -upload- qu'en réception de données –download-) et dépend non seulement de la longueur mais également de la qualité de la ligne téléphonique.

La fibre optique permet de se connecter à plusieurs services simultanément (comme la téléphonie IP, télévision numérique et accès Internet), en conservant un débit très important ainsi qu'une qualité de service optimale.

II. Cahier de charge :

Notre projet consiste à concevoir un récepteur de fibre optique, le processus de réception d'un signal optique se constitue de deux parties :

Partie analogique : Elle s'agit d'un circuit analogique de traitement de signal reçu sur son photodétecteur, il représente une chaine d'acquisition qui sert à recevoir le signal analgique, l'amplifier, le filtrer et le convertir en un signal numérique.

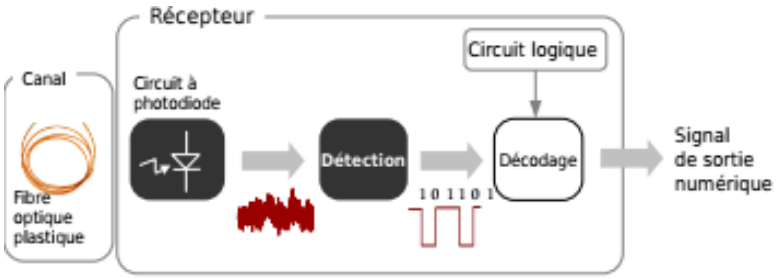


Figure – **Réception numérique**

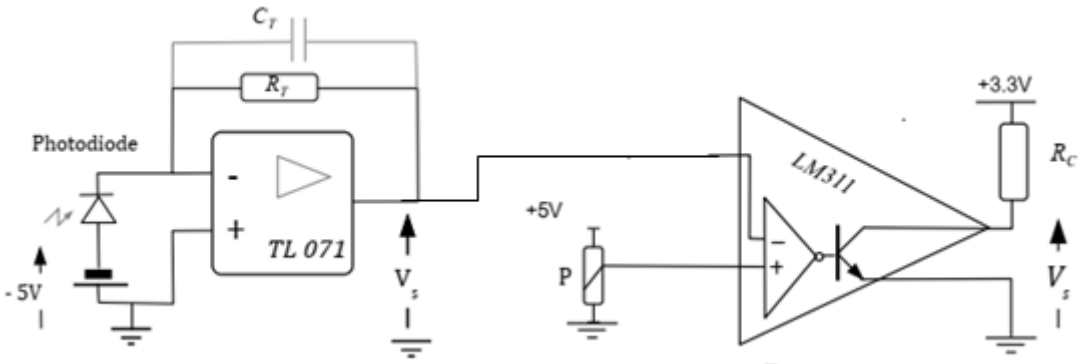


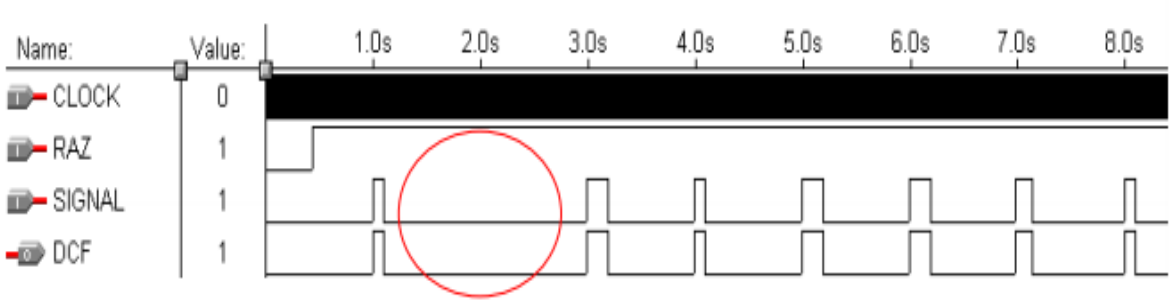
Figure – **Circuit de photo détection**

- TL 071 est un Amplificateur opérationnel lié en parallèle avec une résistance Rt et une capacité Ct, ce petit circuit sert à amplifier et filtrer le signal analogique reçu.
- LM311 est un comparateur lié avec le potentiomètre P qui sert à choisir le seuil de comparaison, Si la valeur de signal analogique est supérieure au seuil, la sortie est égale à 1, sinon la sortie est égale à 0. C'est une méthode de conversion analogique-numérique.

Partie numérique : Cette partie consiste à développer le code VHDL qui sert à faire la détection et comptage de bits de signal numérique, le décodage de données de ce signal et l'affichage de sa valeur décimale sur des 7-segments.

La réalisation de détection des trames de données :

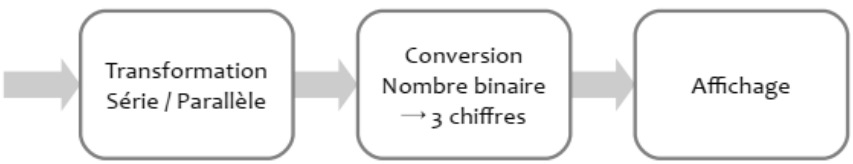
- LES HORLOGES DE SYNCHRONISATION
  L'horloge d'entrée du circuit programmable que nous utilisons sur le Kit de programmation est de grande fréquence, on a besoin de synchroniser la détection et le décodage. Pour cela, il faut effectuer un diviseur de fréquence qui va servir à diviser la fréquence et obtenir la fréquence convenable utilisée dans la synchronisation de toutes les parties à l'aide d'une bascule D. Enfin, on va créer un détecteur des trames de bits en exploitant la fréquence résultante de diviseur de fréquence pour détecter que 8 bits à chaque pulse de fréquence.
  Alors, les étapes de cette partie sont :
  1. Diviseur de fréquence.
  2. Bascule D pour la synchronisation.
  3. Détecteur des trames de bits.
     Le but de bloc « DEBTRAME » est de détecter le bit 9, celui-ci est dit manquant car il ne correspond ni à un niveau '1' ni à un niveau '0'
     Pour réaliser une telle fonction, il faut calculer le temps qui sépare chaque état du signal d'entré, celle-ci se nomme debtrame.vhd, si le temps dépasse 900ms, c'est que le bit 9 est détecté, par mesure de sécurité nous décidons de temporiser à 1200ms.



La réalisation de décodage des données transmises se compose de deux parties :

- La transformation des données arrivant en série en 8 données en parallèle
- La conversion du code binaire sur 8 bits en un nombre entier (compris entre 0 et 255) décomposé en 3 chiffres (compris entre 0 et 9).

Le schéma fonctionnel correspondant est celui de la figure au-dessous :



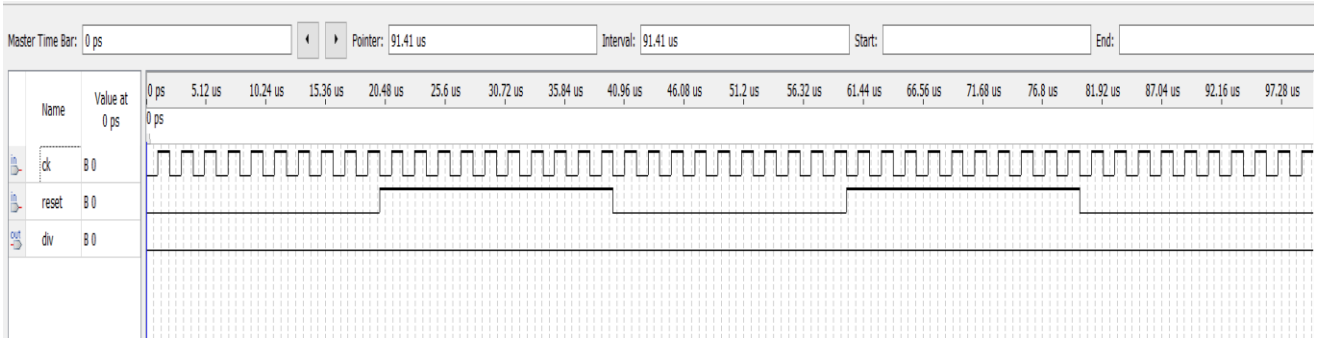
III. Comptage des bits transmis

1. Diviseur de fréquence

a. Code VHDL

```
-- Permet d'obtenir du 1Khz à partir du 25,175Mhz

library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.ALL;

ENTITY divfreq IS
 PORT
        ( ck : IN STD_LOGIC;
          reset : IN STD_LOGIC;
          div: OUT STD_LOGIC );

END divfreq;

ARCHITECTURE numero1 OF divfreq IS
SIGNAL U : INTEGER RANGE 0 TO 25174;
BEGIN
PROCESS(reset,ck)
BEGIN
IF (reset = '0') THEN U <= 0;
ELSIF ck'EVENT AND ck = '1' THEN
      IF (U=25174) THEN U <= 0;
      ELSE U <= U+1;
      END IF;
END IF;
END PROCESS;
div <= '1' WHEN U = 25174 else '0';

 END numero1;
```

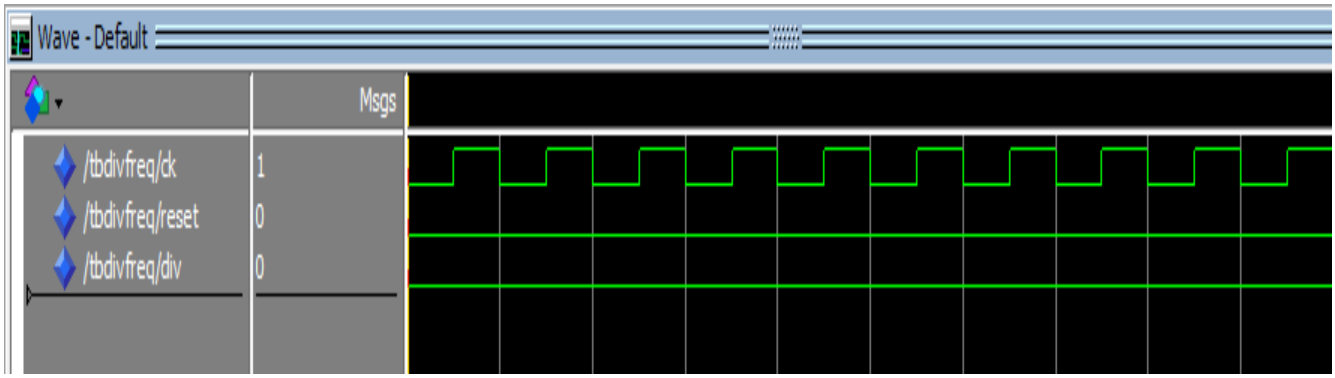
b. Waveform



c. Testbench Code

```
LIBRARY ieee  ;
LIBRARY std  ;
USE ieee.std_logic_1164.all  ;
USE ieee.std_logic_arith.all  ;
USE ieee.std_logic_textio.all  ;
USE ieee.std_logic_unsigned.all  ;
USE std.textio.all  ;
ENTITY tbdivfreq  IS
END ;

ARCHITECTURE tbdivfreq_arch OF tbdivfreq IS
 SIGNAL ck   : STD_LOGIC  ;
 SIGNAL div   : STD_LOGIC  ;
 SIGNAL reset   : STD_LOGIC  ;
 COMPONENT divfreq
  PORT (
   ck  : in STD_LOGIC ;
   div  : out STD_LOGIC ;
   reset  : in STD_LOGIC );
 END COMPONENT ;
 BEGIN
 DUT  : divfreq
  PORT MAP (
   ck   => ck  ,
   div   => div  ,
   reset   => reset   ) ;
-- "Clock Pattern" : dutyCycle = 50
-- Start Time = 0 ns, End Time = 1 us, Period =
100 ns
 Process
          Begin
           ck  <= '0'  ;
          wait for 50 ns ;
-- 50 ns, single loop till start period.
          for Z in 1 to 9
          loop
            ck  <= '1'  ;
           wait for 50 ns ;
            ck  <= '0'  ;
           wait for 50 ns ;

-- 950 ns, repeat pattern in loop.
          end  loop;
           ck  <= '1'  ;
          wait for 50 ns ;
-- dumped values till 1 us
          wait;
 End Process;


-- "Constant Pattern"
-- Start Time = 0 ns, End Time = 1 us, Period = 0
ns
 Process
          Begin
           reset  <= '0'  ;
          wait for 1 us ;
-- dumped values till 1 us
          wait;
 End Process;
END;
```

d. Waveform

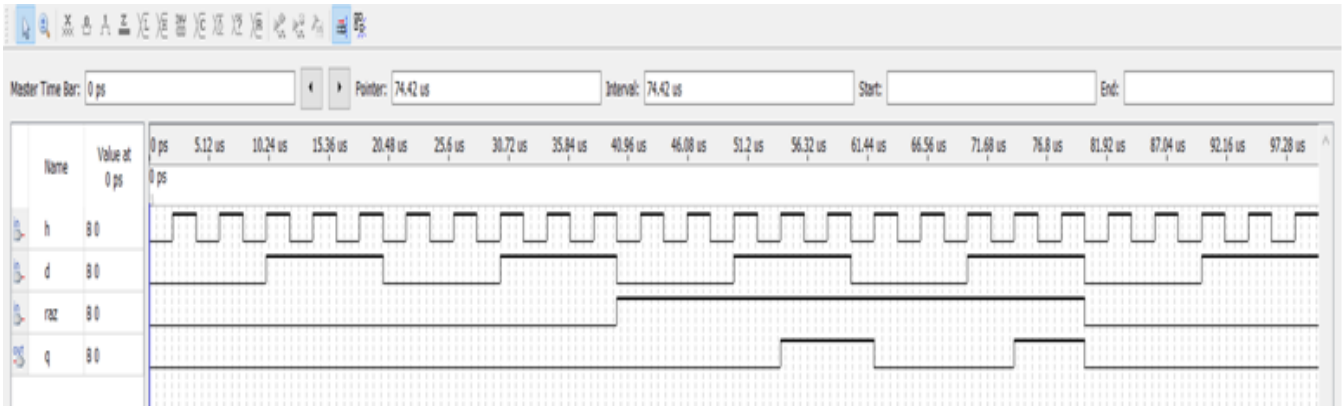


2. Bascule D

a. Code VHDL

```
-- Bascule D,
-- Mise en Forme du Signal , Syncro signal DCF avec h = 1Khz

library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.ALL;

ENTITY basculed IS
 PORT (
 d : IN STD_LOGIC;
 h : IN STD_LOGIC;
 raz : IN STD_LOGIC;
 q : OUT STD_LOGIC
 );

END basculed;

ARCHITECTURE archi1 OF basculed IS
BEGIN
PROCESS (h,raz)
BEGIN
if raz='0' then q<='0';
elsif h'event and h='1' then q<=d;
end If;
END PROCESS;

 END archi1;
```

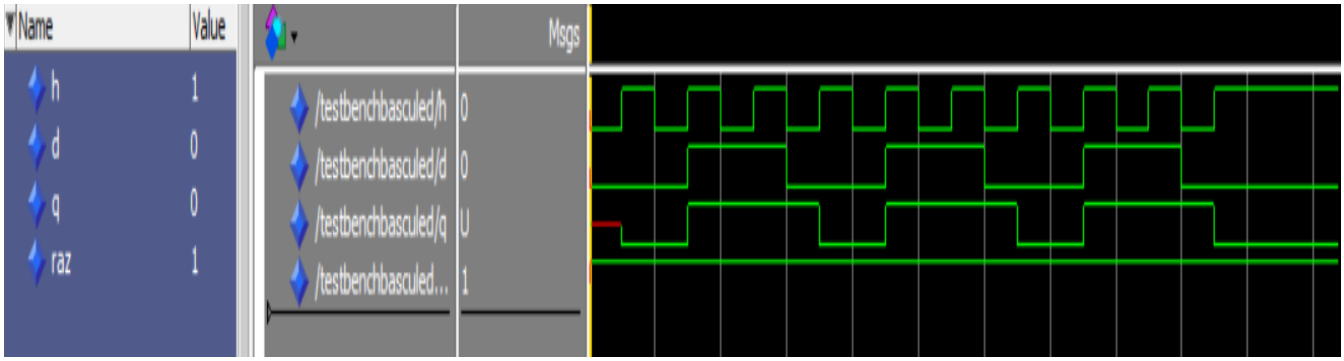
b. Waveform



c. Testbench Code

```
LIBRARY ieee  ;
LIBRARY std  ;
USE ieee.std_logic_1164.all  ;
USE ieee.std_logic_arith.all  ;
USE ieee.std_logic_textio.all  ;
USE ieee.std_logic_unsigned.all  ;
USE std.textio.all  ;
ENTITY tbbasculed  IS
END ;

ARCHITECTURE tbbasculed_arch OF
tbbasculed IS
 SIGNAL h   :  STD_LOGIC  ;
 SIGNAL d   :  STD_LOGIC  ;
 SIGNAL q   :  STD_LOGIC  ;
 SIGNAL raz   :  STD_LOGIC  ;
 COMPONENT basculed
  PORT (
   h  : in STD_LOGIC ;
   d  : in STD_LOGIC ;
   q  : out STD_LOGIC ;
   raz  : in STD_LOGIC );
 END COMPONENT ;
BEGIN
 DUT  : basculed
  PORT MAP (
   h   => h  ,
   d   => d  ,
   q   => q  ,
            raz   => raz   ) ;
-- "Clock Pattern" : dutyCycle = 50
-- Start Time = 0 ns, End Time = 1 us, Period =
100 ns
            Process
            Begin
            h  <= '0'  ;
           wait for 50 ns ;

          -- 50 ns, single loop till start period.
           for Z in 1 to 9
           loop
              h  <= '1'  ;
             wait for 50 ns ;
              h  <= '0'  ;
             wait for 50 ns ;
-- 950 ns, repeat pattern in loop.
            end  loop;
             h  <= '1'  ;
            wait for 50 ns ;
-- dumped values till 1 us
            wait;
 End Process;

-- "Constant Pattern"
-- Start Time = 0 ns, End Time = 1 us, Period = 0
ns
  Process
            Begin
             raz  <= '0'  ;
            wait for 1 us ;
-- dumped values till 1 us
            wait;
 End Process;
-- "Clock Pattern" : dutyCycle = 50
-- Start Time = 0 ns, End Time = 1 us, Period =
50 ns
  Process
            Begin
             d  <= '0'  ;
            wait for 25 ns ;
-- 25 ns, single loop till start period.
            for Z in 1 to 19
            loop
               d  <= '1'  ;
              wait for 25 ns ;
               d  <= '0'  ;
              wait for 25 ns ;
-- 975 ns, repeat pattern in loop.
            end  loop;
             d  <= '1'  ;
            wait for 25 ns ;
-- dumped values till 1 us
            wait;
 End Process;
END;
```

d. Waveform

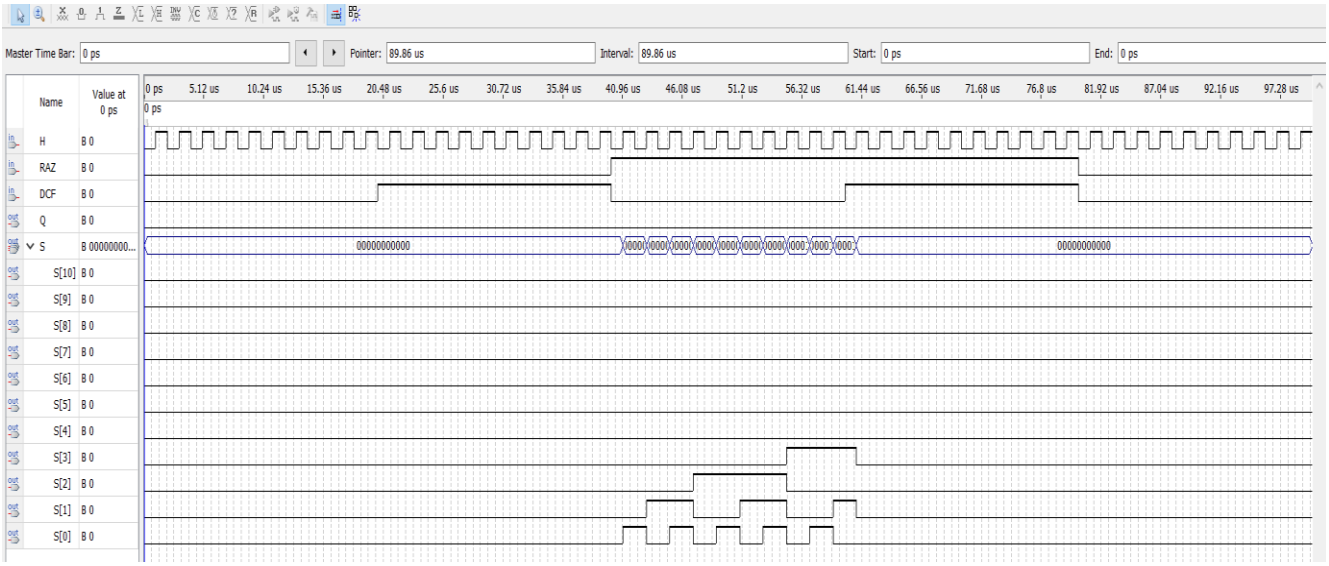


3. Détection de début de trame

a. Code VHDL

```
-- Compteur 1200ms
-- Detection du début de trame => Q= 1
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.ALL;
ENTITY debtrame IS
 PORT (
 H: IN STD_LOGIC;
 DCF: IN STD_LOGIC;
 RAZ: IN STD_LOGIC;
 S: BUFFER INTEGER RANGE 0 TO 1500;
 Q: OUT STD_LOGIC
 );
END debtrame;
ARCHITECTURE archi OF debtrame IS
BEGIN
PROCESS(H, DCF, raz)
 BEGIN
 IF RAZ='0' THEN S<=0; Q<='0';
 ELSIF (H'EVENT AND H='1') Then

 IF (DCF='0') then
 S <= S + 1;
IF S>1200 then Q<='1';
 END IF;
 ELSE S <=0;
 END IF;
 END IF;
END PROCESS;
END archi;
```

b. Waveform



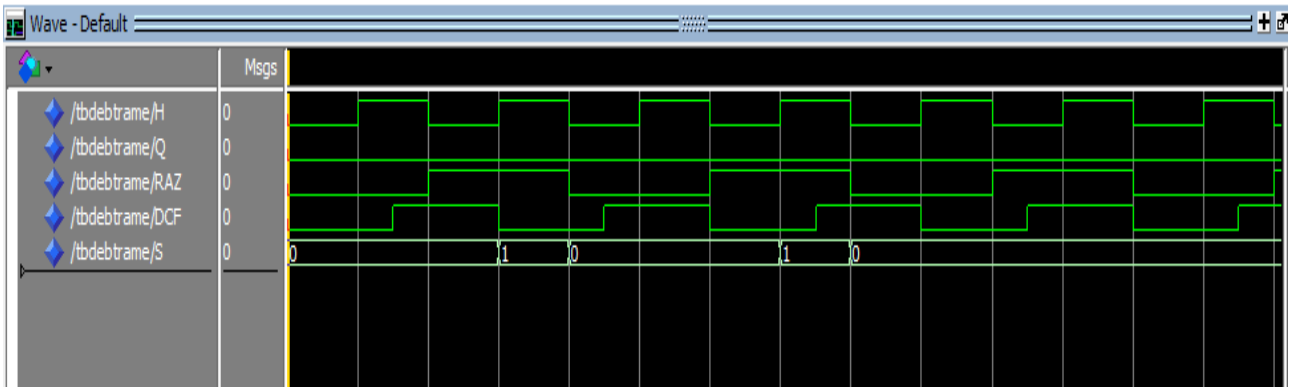
c. Testbench Code

```
LIBRARY ieee  ;
LIBRARY std  ;
USE ieee.std_logic_1164.all  ;
USE ieee.std_logic_arith.all  ;
USE ieee.std_logic_textio.all  ;
USE ieee.std_logic_unsigned.all  ;
USE std.textio.all  ;
ENTITY tbdebtrame  IS
END ;

ARCHITECTURE tbdebtrame_arch OF
tbdebtrame IS
  SIGNAL H   : STD_LOGIC  ;
  SIGNAL Q   : STD_LOGIC  ;
  SIGNAL RAZ   : STD_LOGIC  ;
  SIGNAL DCF   : STD_LOGIC  ;
  SIGNAL S   : INTEGER  ;
  COMPONENT debtrame
   PORT (
    H  : in STD_LOGIC ;
    Q  : out STD_LOGIC ;
    RAZ  : in STD_LOGIC ;
    DCF  : in STD_LOGIC ;
    S  : buffer INTEGER );
  END COMPONENT ;
BEGIN
  DUT  : debtrame
   PORT MAP (
    H   => H  ,
    Q   => Q  ,
    RAZ   => RAZ  ,
    DCF   => DCF  ,
    S   => S   ) ;
```

```
-- "Clock Pattern" : dutyCycle = 50
-- Start Time = 0 ns, End Time = 1 us,
Period = 100 ns
  Process
            Begin
             h  <= '0'  ;
            wait for 50 ns ;
-- 50 ns, single loop till start period.
            for Z in 1 to 9
            loop
              h  <= '1'  ;
             wait for 50 ns ;
              h  <= '0'  ;
             wait for 50 ns ;
-- 950 ns, repeat pattern in loop.
            end  loop;
             h  <= '1'  ;
            wait for 50 ns ;
-- dumped values till 1 us
            wait;
 End Process;


-- "Clock Pattern" : dutyCycle = 50
-- Start Time = 0 ns, End Time = 1 us,
Period = 200 ns
  Process
            Begin
             dcf  <= '0'  ;
            wait for 100 ns ;
-- 100 ns, single loop till start
```

```
period.
            for Z in 1 to 4
            loop
              dcf  <= '1'  ;
             wait for 100 ns ;
              dcf  <= '0'  ;
             wait for 100 ns ;
-- 900 ns, repeat pattern in loop.
            end  loop;
             dcf  <= '1'  ;
            wait for 100 ns ;
-- dumped values till 1 us
            wait;
 End Process;


-- "Constant Pattern"
-- Start Time = 0 ns, End Time = 1 us,
Period = 0 ns
  Process
            Begin
             raz  <= '0'  ;
            wait for 1 us ;
-- dumped values till 1 us
            wait;
 End Process;
END;
```

d. Waveform

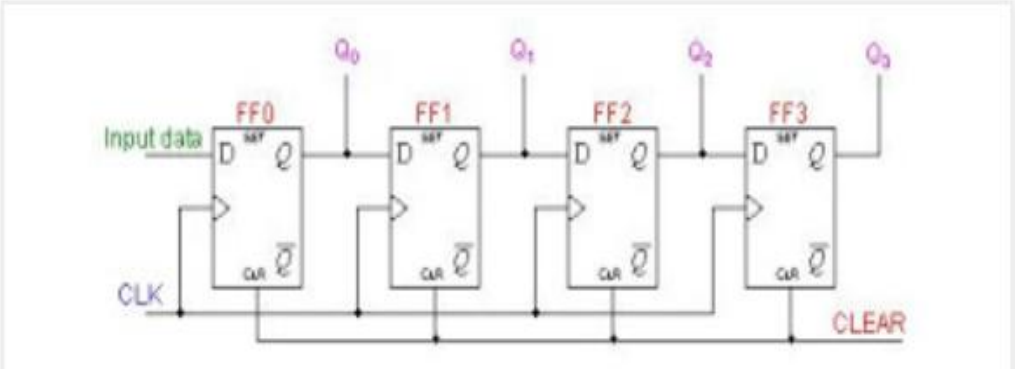


IV. Décodage des données transmises

1. Transformation série/parallèle

Le principe de fonctionnement d'un registre à décalage est basé sur la connexion de la sortie de chaque bascule à l'entrée de la bascule suivante. Ainsi, la donnée introduite dans la première bascule se propage à chaque signal de l'horloge dans les bascules suivantes.

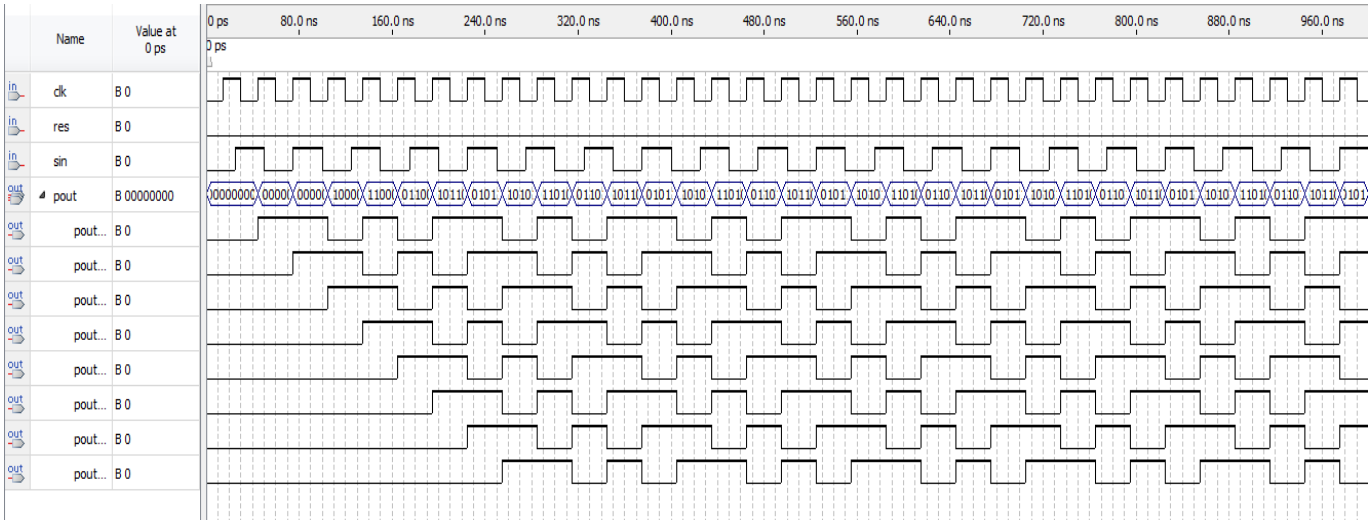
SIPO (Serial In - Parallel Out) : à chaque cycle d'horloge, on ne peut écrire que dans une bascule (lorsque deux bascules ne sont pas reliées entre elles, c'est celle dont l'entrée n'est pas reliée à une autre bascule), mais on peut lire les valeurs de toutes les bascules



a. Code VHDL

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

entity sipo is
port( res: in std_logic;
sin: in std_logic;
clk: in std_logic;
pout: out std_logic_vector(7 downto 0));
end sipo;

architecture beh of sipo is
signal temp: std_logic_vector( 7 downto 0);

begin
process( clk, res)
begin
if(res='1') then
temp<="00000000";
elsif (clk'event and clk ='1')
then
temp(7)<=sin;
temp(6)<=temp(7);
temp(5)<=temp(6);
temp(4)<=temp(5);
temp(3)<=temp(4);
temp(2)<=temp(3);
temp(1)<=temp(2);
temp(0)<=temp(1);
end if;
end process;
pout<=temp;

 end beh;
```

b. Waveform

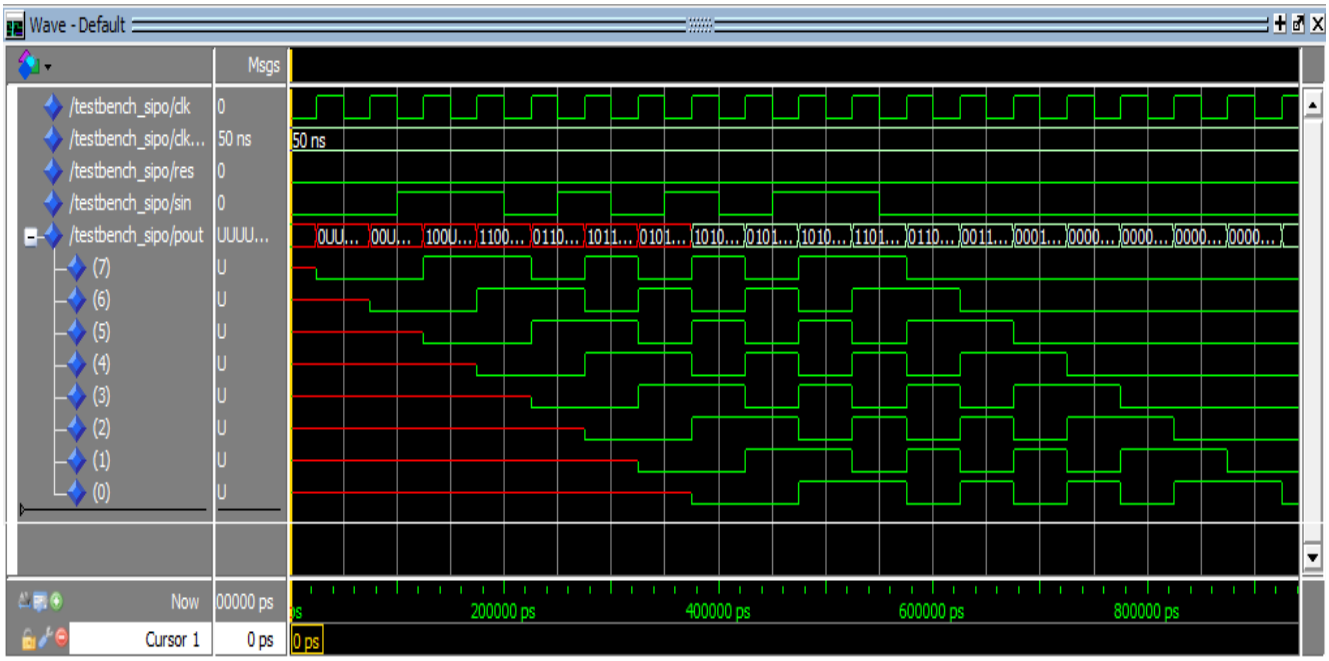


c. Testbench Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY testbench_sipo IS
END testbench_sipo;
ARCHITECTURE behavior OF testbench_sipo IS
   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT sipo
   PORT(
        res : IN  std_logic;
        sin : IN  std_logic;
        clk : IN  std_logic;
        pout : OUT  std_logic_vector(7 downto 0));
   END COMPONENT;
  --Inputs
  signal res : std_logic := '0';
  signal sin : std_logic := '0';
  signal clk : std_logic := '0';
         --Outputs
  signal pout : std_logic_vector(7 downto 0);
  -- Clock period definitions
  constant clk_period : time := 50 ns;
BEGIN
          -- Instantiate the Unit Under Test (UUT)
  uut: sipo PORT MAP (
        res => res,
        sin => sin,
        clk => clk,
        pout => pout       );
-- Clock process definitions
  clk_process :process
  begin
        clk <= '0';
wait for clk_period/2;
 clk <= '1';
 wait for clk_period/2;
  end process;
```

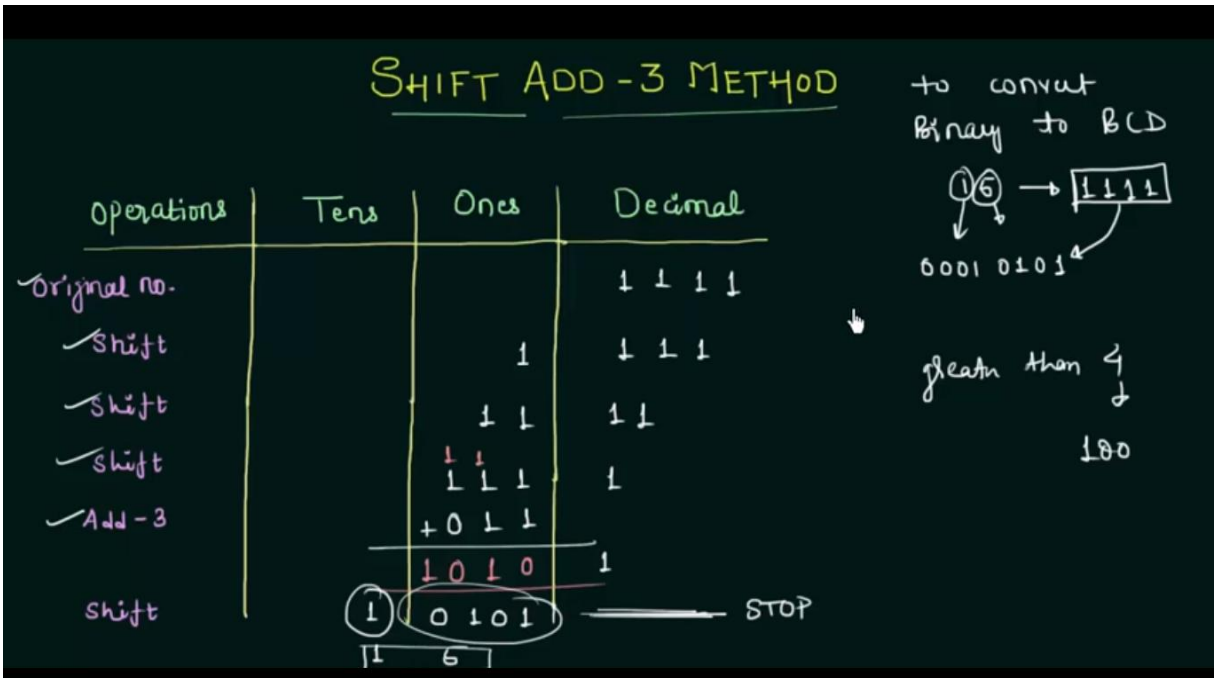
```
  -- Stimulus process
  stim_proc: process
  begin
          sin<='0';
     wait for 50 ns;
sin<='0';
     wait for 50 ns;
sin<='1';
     wait for 50 ns;
sin<='1';
     wait for 50 ns;
sin<='0';
     wait for 50 ns;
sin<='1';
     wait for 50 ns;
sin<='0';
     wait for 50 ns;
sin<='1';
     wait for 50 ns;
sin<='0';
     wait for 50 ns;
                  sin<='1';
     wait for 50 ns;
sin<='1';
     wait for 50 ns;
sin<='0';
     wait for 50 ns;
sin<='0';
     wait for 50 ns;
wait;
     wait;
  end process;
END;
```

d. Waveform



2. Conversion du code binaire au code BCD

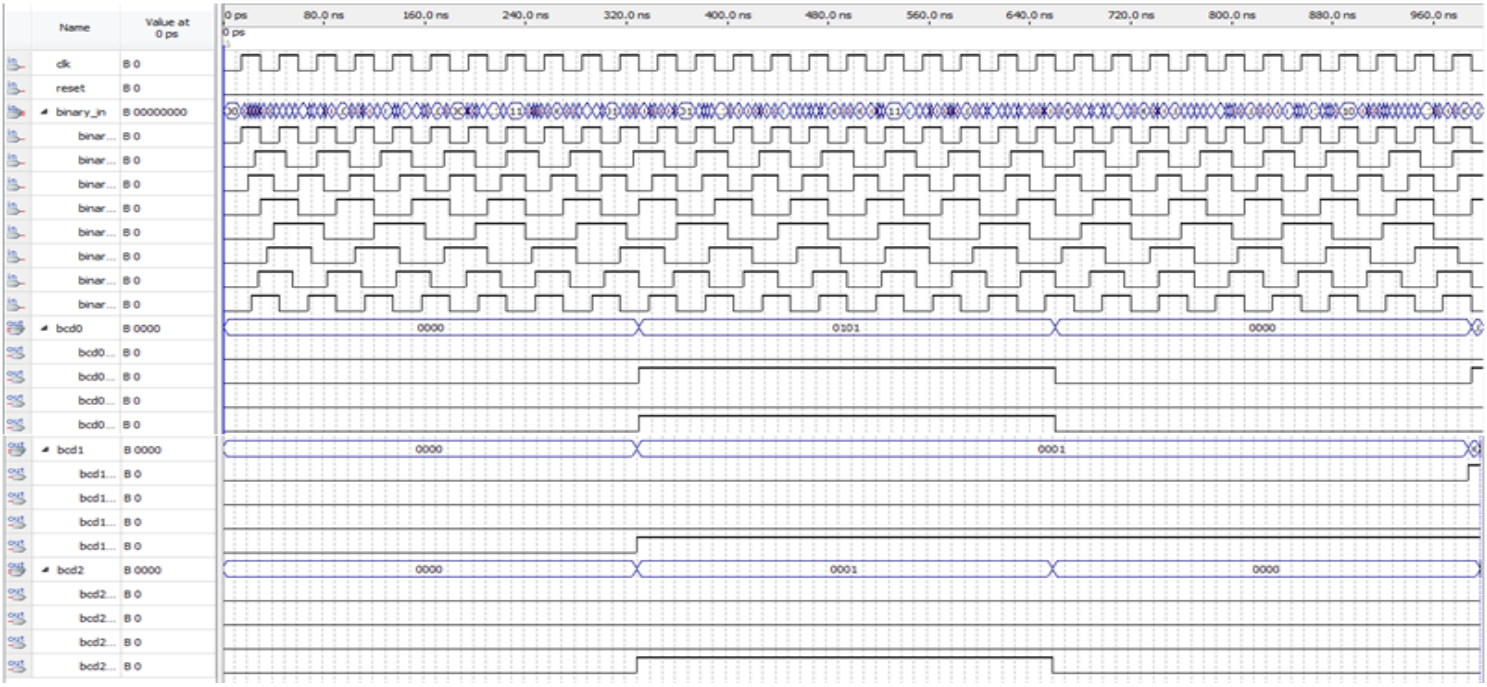
On va utiliser la méthode : Shift ADD-3, par exemple, on convertit le code binaire 1111 en code BCD, alors comme il est montré dans le tableau ci-dessus, on pose le nombre 1111 dans la colonne Decimal, et on le décale à gauche bit par bit, et dés qu'on a un nombre qui est plus grand que 4 (100), on ajoute le nombre 3 (011), et on fait un nouveau décalage à gauche, et finalement on obtient le code BCD 0001 0101.



a. Code VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity binary_bcd is
generic(N: positive := 8);
port(
clk, reset: in std_logic;
binary_in: in std_logic_vector(N-1 downto 0);
bcd0, bcd1, bcd2: out std_logic_vector(3 downto 0)
);
end binary_bcd ;
architecture behaviour of binary_bcd is
type states is (start, shift, done);
signal state, state_next: states;
signal binary, binary_next: std_logic_vector(N-1 downto 0);
signal bcds, bcds_reg, bcds_next: std_logic_vector(19 downto 0);
output register keep output constant during conversion
signal bcds_out_reg, bcds_out_reg_next: std_logic_vector(19 downto 0);
need to keep track of shifts
signal shift_counter, shift_counter_next: natural range 0 to N;
begin
process(clk, reset)
begin
if reset = '1' then
binary <= (others => '0');
bcds <= (others => '0');
state <= start;
bcds_out_reg <= (others => '0');
shift_counter <= 0;
elsif falling_edge(clk) then
binary <= binary_next;
bcds <= bcds_next;
state <= state_next;
bcds_out_reg <= bcds_out_reg_next;
shift_counter <= shift_counter_next;
end if;
end process;
convert:
process(state, binary, binary_in, bcds, bcds_reg, shift_counter)
begin
state_next <= state;
bcds_next <= bcds;
binary_next <= binary;
shift_counter_next <= shift_counter;
case state is
when start =>
state_next <= shift;
binary_next <= binary_in;
bcds_next <= (others => '0');
shift_counter_next <= 0;
when shift =>
if shift_counter = N then
state  next <= done;
binary_next <= binary(N-2 downto 0) & 'L';
bcds_next <= bcds_reg(18 downto 0) &
binary(N-1);
shift_counter_next <= shift_counter + 1;
end if;
when done =>
state_next <= start;
end case;
end process;
bcds_reg(11 downto 8) <= bcds(11 downto 8) +
3 when bcds(11 downto 8) > 4 else
bcds(11 downto 8);
bcds_reg(7 downto 4) <= bcds(7 downto 4) + 3
when bcds(7 downto 4) > 4
else
bcds(7 downto 4);
bcds_reg(3 downto 0) <= bcds(3 downto 0) + 3
when bcds(3 downto 0) > 4 else
bcds(3 downto 0);
bcds_out_reg_next <= bcds when state = done
else
bcds_out_reg;
bcd2 <= bcds_out_reg(11 downto 8);
bcd1 <= bcds_out_reg(7 downto 4);
bcd0 <= bcds_out_reg(3 downto 0);
end behaviour;
```

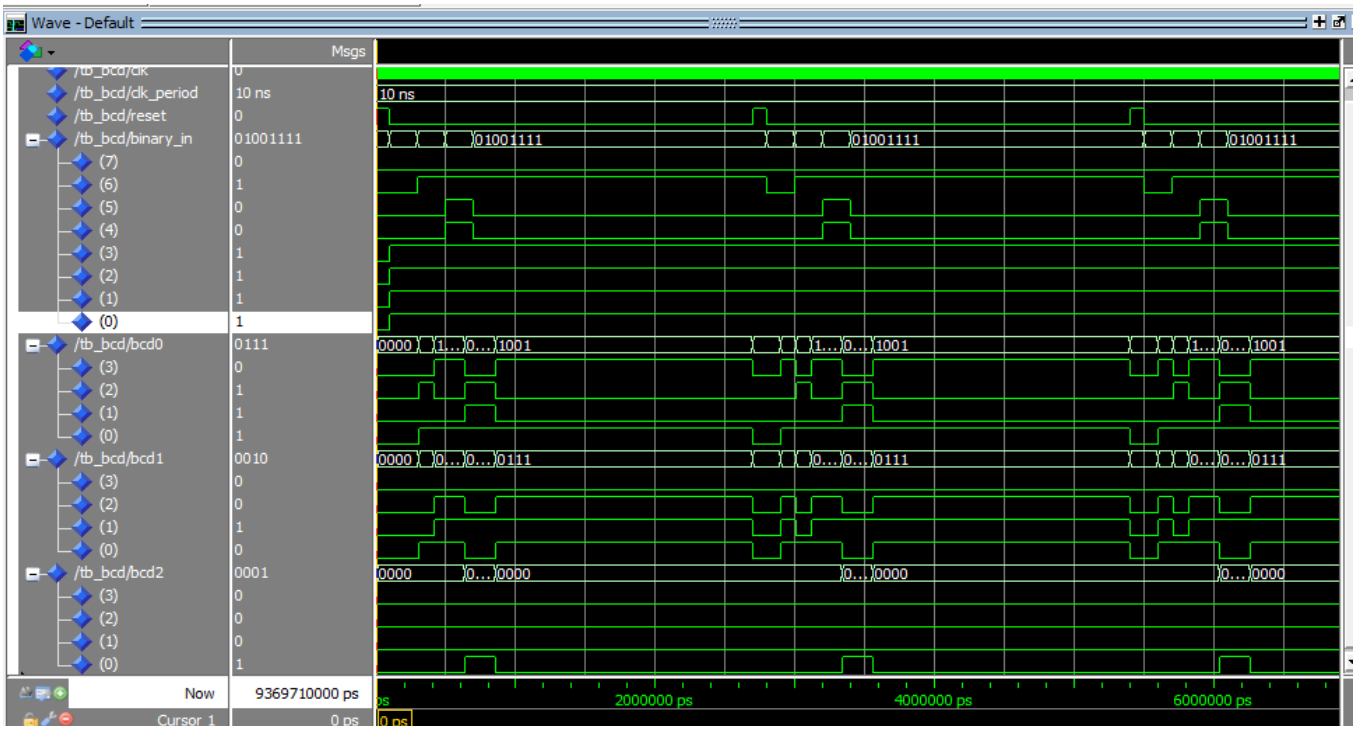
b. Waveform



c. Testbench Code

```
--testbench binary to bcd--
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY tb_bcd IS
END tb_bcd;
 ARCHITECTURE behavior OF tb_bcd IS
   -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT binary_bcd
   PORT(
       clk : IN  std_logic;
       reset : IN  std_logic;
       binary_in : IN  std_logic_vector(7 downto 0);
       bcd0 : OUT  std_logic_vector(3 downto 0);
       bcd1 : OUT  std_logic_vector(3 downto 0);
       bcd2 : OUT  std_logic_vector(3 downto 0)
            );
    END COMPONENT;
   --Inputs
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';
   signal binary_in : std_logic_vector(7 downto 0) := (others => '0');
    --Outputs
   signal bcd0 : std_logic_vector(3 downto 0);
   signal bcd1 : std_logic_vector(3 downto 0);
   signal bcd2 : std_logic_vector(3 downto 0);
-- Clock period definitions
   constant clk_period : time := 10 ns;
 BEGIN
    -- Instantiate the Unit Under Test (UUT)
   uut: binary_bcd PORT MAP (
       clk => clk,
       reset => reset,
       binary_in => binary_in,
       bcd0 => bcd0,
       bcd1 => bcd1,
       bcd2 => bcd2  );
-- Clock process definitions
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;
 -- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
    reset <= '1';
   wait for 100 ns;
    reset <= '0';
     binary_in <= "00001111";
   wait for 200 ns;
     binary_in <= "01001111";
   wait for 200 ns;
     binary_in <= "01111111";
   wait for 200 ns;
    binary_in <= "01001111";
   wait for 2000 ns;
  end process;
END ;
```

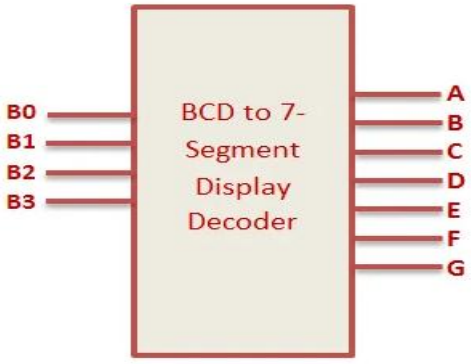
d. Waveform

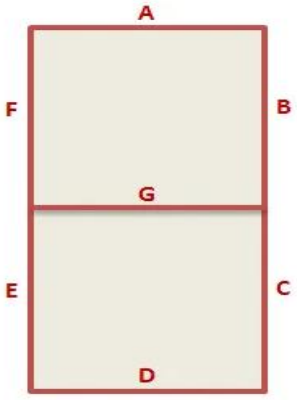


3. Conversion du code BCD au code 7-segment

Le décodeur BCD / 7 segments permet de commander un afficheur à 7 segments.
Il dispose de 7 sorties, notées A,B,C,D,E,F,G correspondant chacune à un des 7 segments de l'afficheur également notés A,B,C,D,E,F,G.

Le segment "A" est évidemment relié à la sortie "A" du décodeur et s'allume ou s'éteint suivant l'état électrique de la sortie (allumé si niveau haut, éteint si niveau bas).
Les entrées sont au minimum de quatre. On notera les quatre entrées principales B3, B2, B1, et B0. Elles représentent le nombre binaire B3 B2 B1 B0 (B3 étant le bit de poids le plus fort et B0 celui de poids le plus faible) à afficher.
L'état des sorties du décodeur dépend du nombre binaire que l'on a en entrée. Ce nombre binaire est affiché en décimal sur l'afficheur à 7 segments.





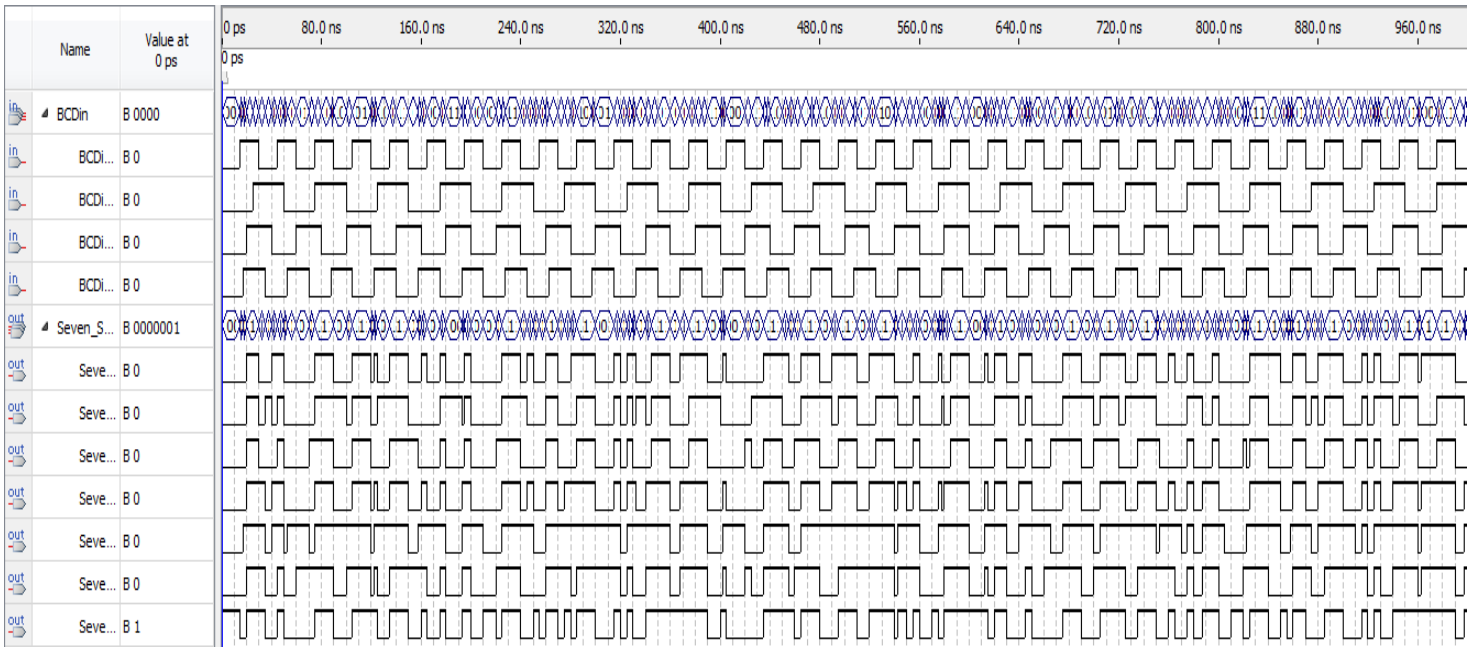
Avec un afficheur à 7 segments, on ne peut afficher que les 10 premiers chiffres de 0 (0000 en binaire) à 9 (1001 en binaire). Si le nombre en entrée du décodeur est supérieur à 9, l'affichage ne représente plus un nombre et dépend du type du décodeur.

B3 B2 B1 B0	A B C D E F G
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0000100

a. Code VHDL

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

entity bcd_7segment is

Port ( BCDin : in STD_LOGIC_VECTOR (3 downto 0);
Seven_Segment : out STD_LOGIC_VECTOR (6 downto 0));
end bcd_7segment;

architecture Behavioral of bcd_7segment is
begin

process(BCDin)

begin

case BCDin is
when "0000" =>
Seven_Segment <= "0000001"; ---0
when "0001" =>
Seven_Segment <= "1001111"; ---1
when "0010" =>
Seven_Segment <= "0010010"; ---2
when "0011" =>
Seven_Segment <= "0000110"; ---3
when "0100" =>
Seven_Segment <= "1001100"; ---4
when "0101" =>
Seven_Segment <= "0100100"; ---5
when "0110" =>
Seven_Segment <= "0100000"; ---6
when "0111" =>
Seven_Segment <= "0001111"; ---7
when "1000" =>
Seven_Segment <= "0000000"; ---8
when "1001" =>
Seven_Segment <= "0000100"; ---9
when others =>
Seven_Segment <= "1111111"; ---null
end case;

end process;

end Behavioral;
```

b. Waveform

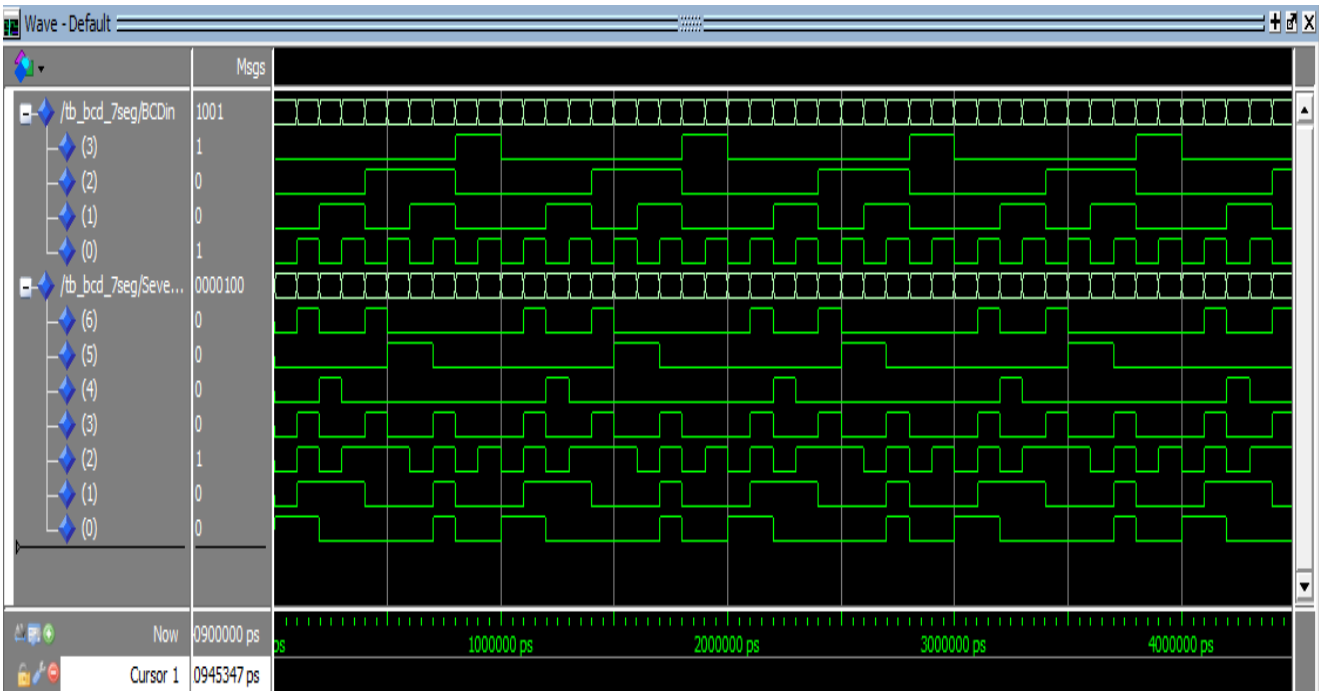


c. Testbench Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_bcd_7seg IS
END tb_bcd_7seg;

ARCHITECTURE behavior OF tb_bcd_7seg IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT bcd_7segment
PORT(
BCDin : IN std_logic_vector(3 downto 0);
Seven_Segment : OUT std_logic_vector(6 downto 0)
);
END COMPONENT;

--Inputs
signal BCDin : std_logic_vector(3 downto 0) := (others => '0');

--Outputs
signal Seven_Segment : std_logic_vector(6 downto 0);

BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: bcd_7segment PORT MAP (
BCDin => BCDin,
Seven_Segment => Seven_Segment );
```

```
-- Stimulus process
stim_proc: process
begin

BCDin <= "0000";
wait for 100 ns;
BCDin <= "0001";
wait for 100 ns;
BCDin <= "0010";
wait for 100 ns;
BCDin <= "0011";
wait for 100 ns;
BCDin <= "0100";
wait for 100 ns;
BCDin <= "0101";
wait for 100 ns;
BCDin <= "0110";
wait for 100 ns;
BCDin <= "0111";
wait for 100 ns;
BCDin <= "1000";
wait for 100 ns;
BCDin <= "1001";
wait for 100 ns;
end process;

END;
```

d. Waveform



V. Conclusion

La souplesse et la flexibilité des langues de description hardware ont eu un impact énorme sur la digitalisation de systèmes électroniques pendant les dernies décennies, pour notre cas, une conception analogique de la partie détection et décodage pourrait être très difficile et compliqué. On aurait besoin de beaucoup de câbles, plusieurs composantes électroniques et le plus important il résulterait un produit très volumineux comparant à un FPGA ou un ASIC.

Le VHDL a facilité la tâche de conception et simulation de fonctionnement de système conçu, d'autre part, l'apparence des ASICs et les FPGA a aussi contribué dans le côté de minimisation de volume de système électroniques. Sans oublier les SOCs qu'ils existent dans 99% de smartphones modernes, On a entré le monde de nanotechnologie grâce à l'évolution de ces circuits intégrés.