REPUBLIQUE FRANCAISE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR,
DE LA RECHERCHE ET DE L'INNOVATION

UNIVERSITE DE BOURGOGNE
UFR SCIENCES ET TECHNIQUES
Science pour l'ingénieur
BP 47870 - 21078 DIJON cedex
Tél. 03 80 39 58 87

RAPPORT DE TP FPGA

BOUDZOUMOU NZALANKAZI Jean Marcellino
MASTER 1 TSI
Sous la direction de :
Pr. Jean Miteran & M. TESSOMA Solomon

Année académique 2020-2021

TABLE DES MATIERES

TABLE DES FIGURES

Introduction

Les fpga sont des composants dits à architectures programmables. Le but des manipulations exposées dans ce rapport est de programmer des fpga afin qu'elles réalisent des fonctions bien précises. Nous supposons déjà que vous disposez des bases de la programmation en langage vhdl. Afin de dépasser les consignes du travail demandes, nous utiliserons une approche structurale, en claire nous feront de chaque sous fonctionnalité du circuit une entité à part entière. Ces programmes seront ensuite téléversés dans le fpfa tout en lui associant des contraintes.

Outils

Pour nos manipulations, nous écrirons nos programmes vhdl et compilerons nos programmes via l'environnement de développement Vivado en language vhdl
Par la suite ces programmes pourront être téléversés dans le fpga XC6SLX16 embarqué dans la carte spartan-6 de xilinx

Travail 1 : circuit anti-rebond

Lorsque l'on manipule des composants tel qu'un bouton, le résultat attendu peut être sujet à erreur ou devenir incertain : valeur varie au cours du temps.Pour cela, on procède en électronique par interposition de filtre lequel peut être analogique ou numérique. Dans l'un ou l'autre, l'objectif est de s'assurer qu'une règle de validation au cours du temps est respectée.

I. Le circuit anti rebond

Les grandeurs physiques obtenues à l'aide des capteurs doivent toujours être filtrés. Dans le contexte de L'électronique numérique cela commence par l'échantillonnage pour donner au signal obtenu un sens numérique ou discret

De manière générale, une sortie doit respecter une règle dépendant du temps (fréquence ou période) afin d'être validée. On pourrait parler de seuil en électronique analogique, ou taux de variation minimal pour l'électronique numérique. La règle de validation est que la sortie soit stable. Elle est stable si et seulement si elle ne change pas pendant $2^{16} = 65536$ coup d'horloge. :

Le circuit anti-rebond sera composé de trois blocs : un circuit de lecture de l'entrée, un circuit de comparaison successive des valeurs lues et un circuit de validation.

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

entity debouncing is
    generic(counter_size  :  integer :=20);                                    --counter size (20 bits gives 10.5ms with 100MHz clock)
    port(
        signal clk : in std_logic;                                             --input clock
        signal btn : in std_logic;                                             --input signal to be debounced
        signal res  : out std_logic
        );                                                                     --debounced signal
end debouncing;

architecture debouncing of debouncing is
     signal flipflops   : in std_logic_vector(1 downto 0);                     --input flip flops
     signal counter_set : in std_logic;                                        --sync reset to zero
     signal counter_out : out std_logic_vector(counter_size downto 0) := (others => '0');    --counter output
begin
    counter_set <= flipflops(0) xor flipflops(1);                              --determine when to start/reset counter
    process(clk)
    begin
    if(clk'EVENT and clk = '1') then
        flipflops(0) <= btn;
        flipflops(1) <= flipflops(0);
        if(counter_set = '1') then                                              --reset counter because input is changing
            counter_out <= (others => '0');
        elsif(counter_out(counter_size) = '0') then                             --stable input time is not yet met
            counter_out <= counter_out + 1;
        else                                                                    --stable input time is met
                res <= flipflops(1);
        end if;
        end if;
    end process;
end architecture debouncing;
```

Figure 1 : programme du circuit anti-rebond

1. Le circuit de lecture de l'entrée

Il existe un circuit numérique qui ne met en sa sortie la valeur en entrée : c'est la bascule D. Nous allons donc cadencer deux bascules D qui lirons consécutivement l'entrée.

2. Le comparateur

Il existe un circuit capable numérique d'analyser la différence entre deux bits : c'est la porte « ou exclusif ».

3. Le circuit de validation

Ce circuit est un compteur, il compte le nombre de fois que n'a pas changé. En d'autres termes, il s'incrémente si le résultat du comparateur est vrai, sinon il se remet à zéro

II. Le compteur

```
library iee;
use ieee.std_lgic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
    Port
    (
        clk, counter_in : in  STD_LOGIC;
        res : out  STD_LOGIC_VECTOR (7 downto 0)
    );
end counter;

architecture counter of counter is
signal click_counter : std_logic_vector(7 downto 0):="00000000";
    begin
        process(clk, counter_in)
        begin
            if clk'event and clk='1' then
                if counter_in='1' then
                    click_counter <= click_counter + '1';
                end if;
            end if;
        end process;
        res <= click_counter;
end counter;
```

Figure 2 : Programme du compteur

III. Application

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

entity test is
end entity test;

architecture test of test is
    signal clk : std_logic :='0';
    signal btn : std_logic;
    signal res : std_logic_vector(7 downto 0);
    component click_counter is
        port
        (
            signal clk : in std_logic;
            signal btn : in std_logic;
            signal res : out std_logic_vector(7 downto 0)
        );
    end component click_counter;

    begin
    clk <= not(clk) after 10 ns;
    debounce : click_counter port map(clk=>clk,btn => btn, res=>res);
    process
    begin
        btn <='1';
        btn <= '0' after 20 ns;
        btn <='1';
        wait for 30 ns;
    end process;
end architecture test;

configuration config of test is
    for test
        for debounce : click_counter use entity work.click_counter(click_counter); end for;
    end for;
end configuration config;
```

Figure 3 : Exécution du travail 1

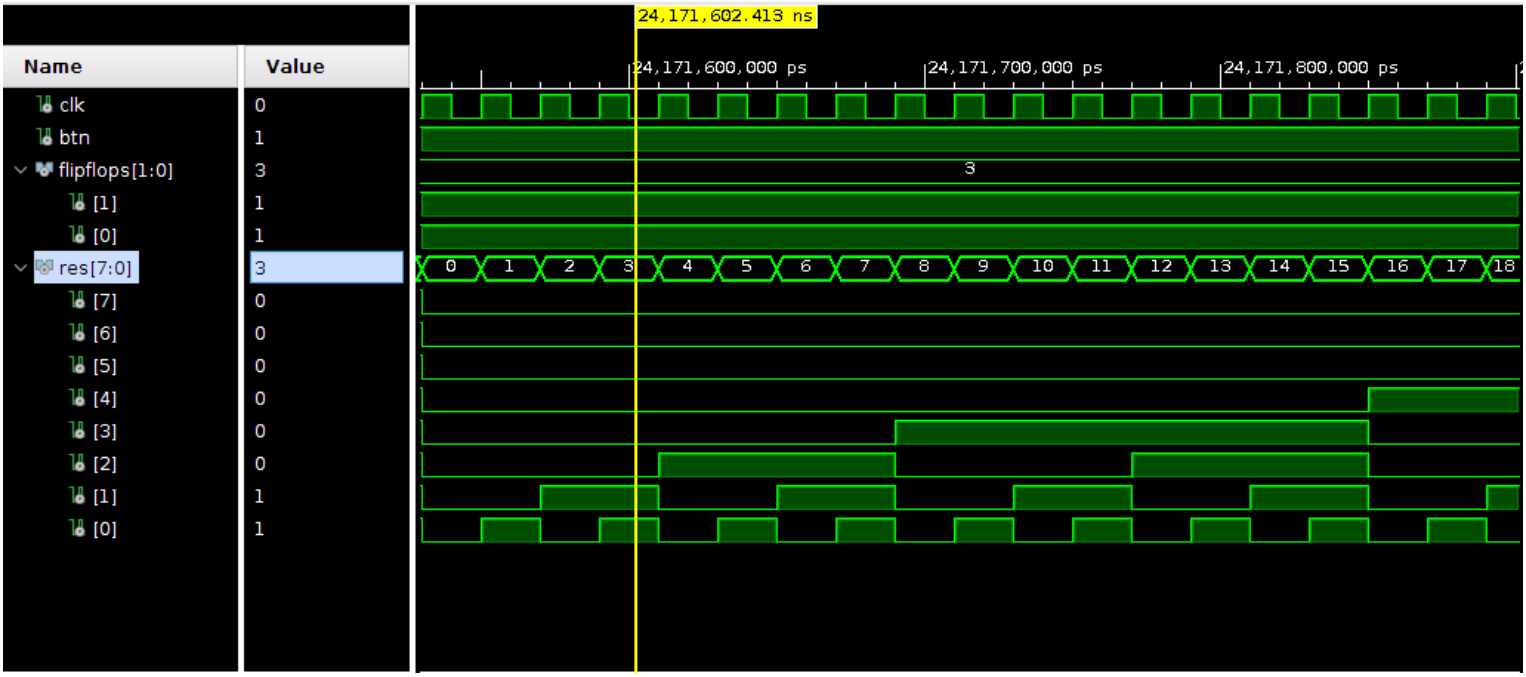


Figure 4 : Résultat de la simulation de la première manipulation

Travail 2 : Compteur automatique et affichage

Le circuit à concevoir dans cette manipulation est constitué des entités suivantes : un compteur et un codeur bcd.

I. Le compteur

Il a comme sortie un tableau de bit qui correspond en binaire au chiffre décimal compté. Pour pouvoir compter, il incrémente d'un bit le mot binaire stocké dans sa sortie à une fréquence donnée.

```
entity counter is
    generic
    (
        second :  std_logic_vector:=x"1f50ff";                          -- N equivalent binaire du nombre de pulsation pour atteindre la seconde par rapport a l'horloge externe
        for_second : integer := 26                                      -- nombre de bits pour compter jusqu'a la seconde
    );

    Port
    (
        clk : in  std_logic;                                            -- signal d'horloge externe
        rst : in  std_logic;                                            -- remise a zero
        binary_number : out  std_logic_vector (3 downto 0)
    );
end counter;

architecture counter of counter is
    signal one_sec_counter: std_logic_vector(for_second downto 0):=(others=>'0'); -- compteur de front montant de l'horloge externe
    signal one_sec_en: std_logic;                                       -- Bit de detection de la seconde
    signal number: std_logic_vector(3 downto 0) :="0000";
begin
    process(clk,rst)                                                    -- Processus de detection de la seconde
    begin
        if rst='1' then
            one_sec_counter <= (others=>'0');
        elsif rising_edge(clk) then
            if one_sec_counter=second then
                one_sec_counter <= (others=>'0');                       -- for each rising edge when  counter in 1 seconde is max assign 0 to himself
            else
                one_sec_counter <= one_sec_counter +'1';                -- else add 1 bit
            end if;
        end if;
    end process;
    one_sec_en <= '1' when one_sec_counter=second else '0';

    process(clk,rst)                                                    -- Process about bcd behavioral on n the time and detecting second
        begin
            if rst='1' then
                number<=(others=>'0');                                      -- bcd code set 0 on rst pressing
            elsif rising_edge(clk) then
                if one_sec_en='1' then                                  -- when 1 second accomplished
                    if number="1001" then                                   -- if bcd =1001 then 10
                        number<=(others=>'0');                              -- set it at 0
                        else
                        number<=number +'1';                                    -- else add 1 bit to bcd
                    end if;
                end if;
            end if;
            binary_number<=number;
    end process;
end counter;
```

Figure 5 : Programme du compteur

II. Le codeur

Le codeur quant à lui est destiné à recevoir le mot binaire en sortie du compteur pour le coder de sorte à obtenir un code bcd pour l'afficheur.

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

entity coder is
    port
    (
        signal binary_number : in std_logic_vector(3 downto 0);
        signal bcd : out std_logic_vector(6 downto 0);
        signal com : out std_logic :='0'
    );
end entity coder;

architecture coder of coder is
begin
    with binary_number(3 downto 0) select
    bcd(6 downto 0)<=
       --abcdefg
        "0000001" when "0000",                      --0
        "1001111" when "0001",                      --1
        "0010010" when "0010" ,                     --2
        "0000110" when "0011",                      --3
        "1001100" when "0100" ,                     --4
        "0100100" when "0101",                      --5
        "0100000" when "0110" ,                     --6
        "0001111" when "0111",                      --7
        "0000000" when "1000" ,                     --8
        "0000100" when "1001",                      --9
        "0001000" when "1010" ,                     --A
        "1100000" when "1011",                      --B
        "0110001" when "1100",                      --C
        "1000010" when "1101" ,                     --D
        "0110000" when "1110",                      --E
        "0111000" when others;                      --F
        --bcd(6) <= '1';
end architecture coder;
```

Figure 6 : Programme du codeur

III. Le circuit bcd_counter

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


entity bcd_counter is
    Port
    (
        signal clk : in std_logic;
        signal rst : in std_logic;
        signal com : out std_logic;
        signal bcd : out std_logic_vector(6 downto 0)
    );
end bcd_counter;

architecture bcd_counter of bcd_counter is
    component counter is
        port
        (
            signal clk : in std_logic;
            signal rst : in std_logic;
            signal binary_number : out std_logic_vector(3 downto 0)
        );
    end component counter;

    component coder is
        port
        (
            signal binary_number : in std_logic_vector(3 downto 0);
            signal com : out std_logic;
            signal bcd : out std_logic_vector(6 downto 0)
        );
    end component coder;

    signal number : std_logic_vector(3 downto 0);
begin
    compteur : counter port map(clk=>clk,rst=>rst, binary_number=>number);
    codeur : coder port map(binary_number =>number,com=>com, bcd=>bcd);

end architecture bcd_counter;

configuration conf of bcd_counter is
    for bcd_counter
        for compteur : counter use entity work.counter(counter); end for;
        for codeur : coder use entity work.coder(coder); end for;
    end for;
end configuration conf;
```

Figure 5 : Programme du circuit bcd_counter

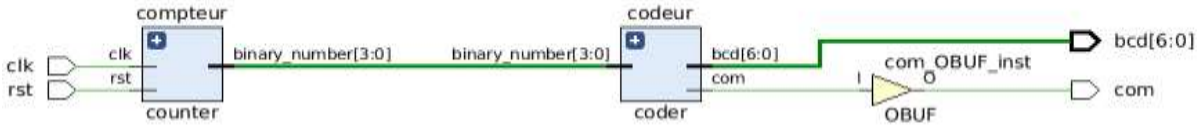


Figure 6 : vue du bcd_counter et ses différentes entités

IV. Application

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

entity test is
end test;

architecture test of test is

    signal clk : std_logic :='0';
    signal rst : std_logic;


    component bcd_counter is
    Port
    (
        signal clk : in std_logic;
        signal rst :in std_logic;
        signal com : out std_logic;
        signal bcd : out std_logic_vector(6 downto 0)
    );
    end component bcd_counter;

    begin
    disp : bcd_counter port map(clk=>clk, rst=>rst);

    process
    begin
    clk <= not(clk);
    wait for 0.01 us;
    end process;
end architecture test;

configuration config of test is
    for test
    for disp :bcd_counter use entity work.bcd_counter(bcd_counter); end for;
    end for;
end configuration config;
```

Figure 7 : Programme pour la simulation du bcd_counter

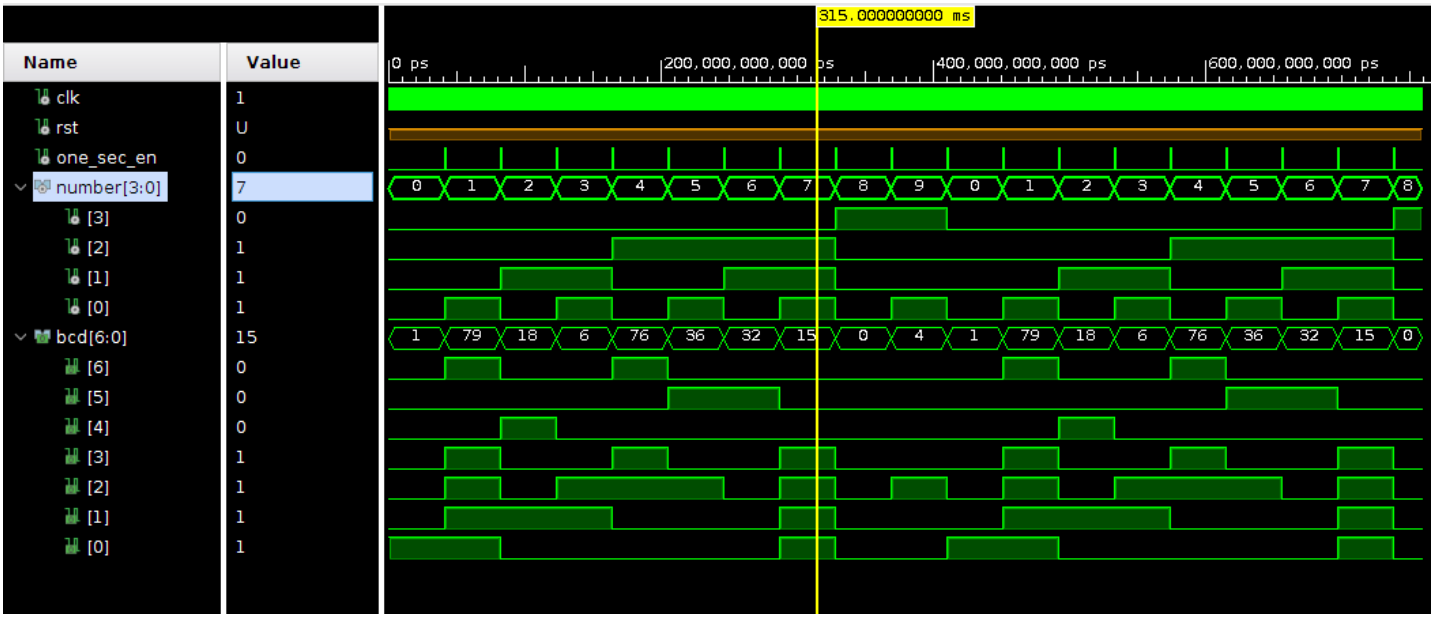


Figure 8 : Résultats de la simulation de bcd_counter

Travail 3 : Compteur automatique et afficheur sur 4 digits

Cette troisième manipulation consiste à réaliser un compteur autonome dont le résultat est à afficher. Cette manipulation comprends trois grandes parties : les compteurs, le multiplexeur, l'afficheur, le codeur

I. les compteurs

II. le multiplexeur

Le multiplexeur ne dépend que de l'horloge. Le même multiplexeur est sera utiliser dans la prochaine manipulation

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity mux is
    generic
    (
        N : integer :=22;
        stamp : std_logic_vector :="11110100001001000000"
    );
    Port
    (
        signal clk : in std_logic;
        signal in0 : in std_logic_vector(3 downto 0);
        signal in1 : in std_logic_vector(3 downto 0);
        signal in2 : in std_logic_vector(3 downto 0);
        signal in3 : in std_logic_vector(3 downto 0);
        signal dp_in: in std_logic_vector(3 downto 0);
        signal sortie : out std_logic_vector(3 downto 0);
        signal dp_out : out std_logic;
        signal com : out std_logic_vector(3 downto 0)
    );
end entity mux;
architecture mux of mux is
     signal count_stamp:std_logic_vector(N-1 downto 0):=(others=>'0');
     signal count :std_logic;
     signal sel: std_logic_vector(1 downto 0);
     begin
        process(clk)
        begin
            if(rising_edge(clk)) then
                if(count_stamp = stamp) then
                    count_stamp <= (others=>'0');
                else
                    count_stamp<=count_stamp +'1';
                end if;
            end if;
        end process;
        sel <= std_logic_vector(count_stamp(N-1 downto N-2));
        process(sel, in0, in1, in2, in3)
        begin
            case sel is
                when "00" => sortie <= in0; com<="1110"; dp_out <= dp_in(0);
                when "01" => sortie <= in1; com<="1101"; dp_out <= dp_in(1);
                when "10" => sortie <= in2; com<="1011"; dp_out <= dp_in(2);
                when others => sortie <= in3; com<="0111"; dp_out <= dp_in(3);
            end case;
        end process;
end architecture mux;
```

Figure 9 : Programme du multiplexeur

III. Le codeur

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

entity coder is
    port
    (
        signal binary_number : in std_logic_vector(3 downto 0);
        signal dp : in std_logic;
        signal sseg : out std_logic_vector(7 downto 0);
        signal com : out std_logic :='0'
    );
end entity coder;

architecture coder of coder is
begin
    com <= '1';
    with binary_number select
    sseg(6 downto 0)<=
       --abcdefg
        "0000001" when "0000",                  --0
        "1001111" when "0001",                  --1
        "0010010" when "0010" ,                 --2
        "0000110" when "0011",                  --3
        "1001100" when "0100" ,                 --4
        "0100100" when "0101",                  --5
        "0100000" when "0110" ,                 --6
        "0001111" when "0111",                  --7
        "0000000" when "1000" ,                 --8
        "0000100" when "1001",                  --9
        "0001000" when "1010" ,                 --A
        "1100000" when "1011",                  --B
        "0110001" when "1100",                  --C
        "1000010" when "1101" ,                 --D
        "0110000" when "1110",                  --E
        "0111000" when others;                  --F
        sseg(7) <= '1';
end architecture coder;
```

Figure 10 : Programme du codeur bcd

La figure ci-après illustre le schéma du circuit ainsi conçu :

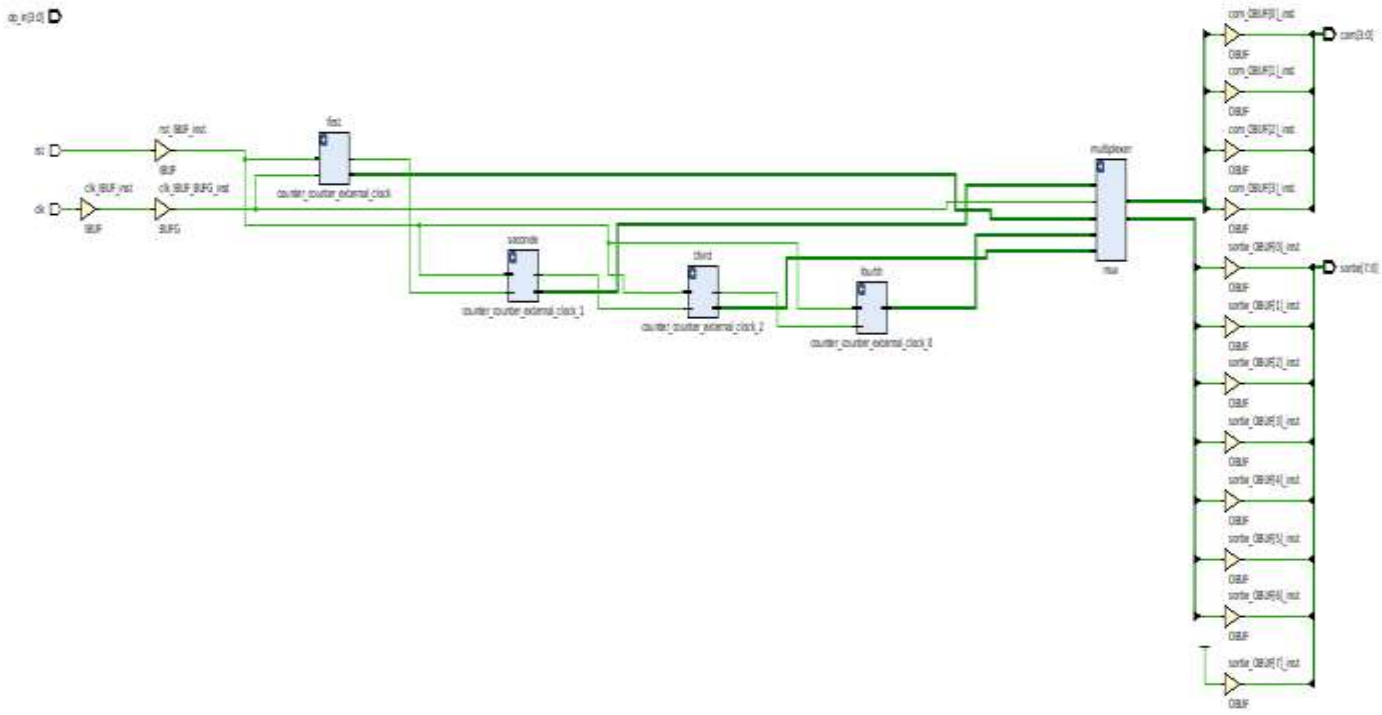


Figure 11 : Architecture du compteur compteur automatique et affichage sur quatre digits

IV. Application

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

entity test is
-- Port ( );
end test;

architecture test of test is
    component displayer
    Port
    (
        signal clk : in std_logic ;
        signal rst : in std_logic ;
        signal dp_in : in std_logic_vector(3 downto 0);
        signal sortie : out std_logic_vector(3 downto 0);
        signal com : out std_logic_vector (3 downto 0)
     );
     end component displayer;
     signal clock : std_logic:='0';
     signal rst : std_logic:='0';
     signal dp : std_logic_vector(3 downto 0) :=(others=>'0');
begin
    display9999 : displayer port map(clk=>clock, rst=>rst, dp_in=>dp);
    process
    begin
       clock <= not(clock);
       dp <=dp +'1';
       wait for 10 ns;
    end process;
end architecture test;

configuration config of test is
    for test
        for display9999 : displayer use entity work.displayer(displayer); end for;
    end for;
end configuration config;
```

Figure 12 : Programme de simulation du compteur

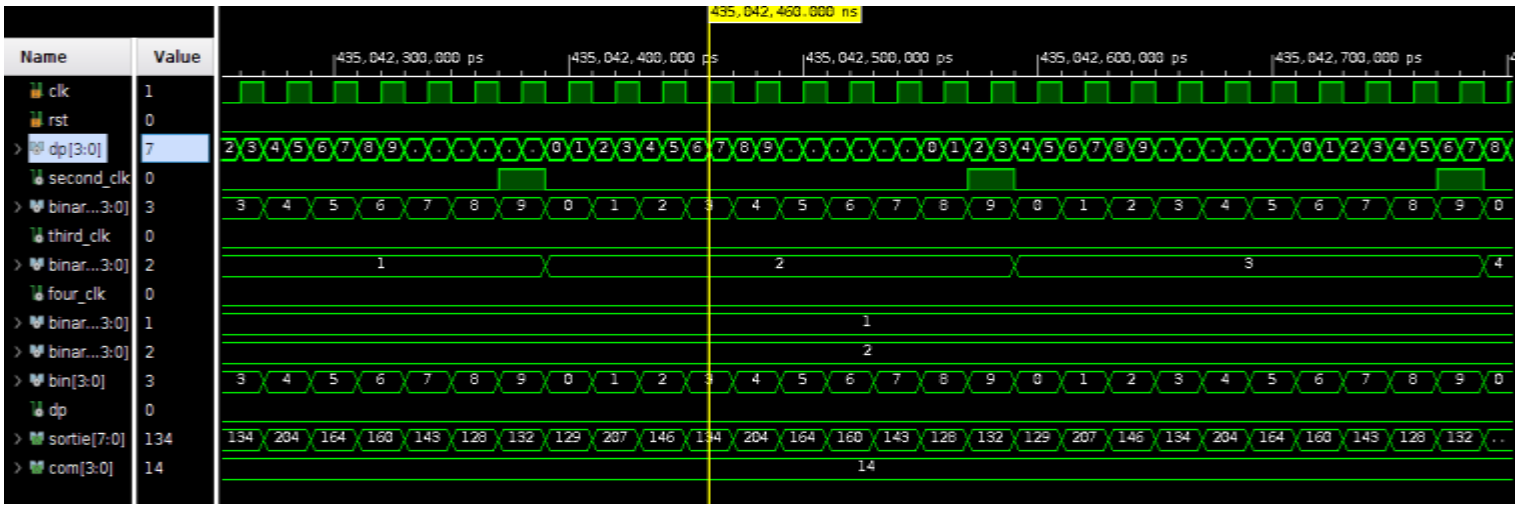


Figure 13 : Chronogramme de la simulation du compteur

Travail 3 : le circuit additionneur

Pour résumer ce travail, il s'agit de lire huit entrées dans un ordre donné, en faire deux nombre sur quatre bites. On additionnera ces deux nombres. Le résultat sera lui aussi sur deux chiffres de quatre bites chacun. On affichera simultanément les nombres entrés et le résultat de la somme sur un afficheur 4 digits.

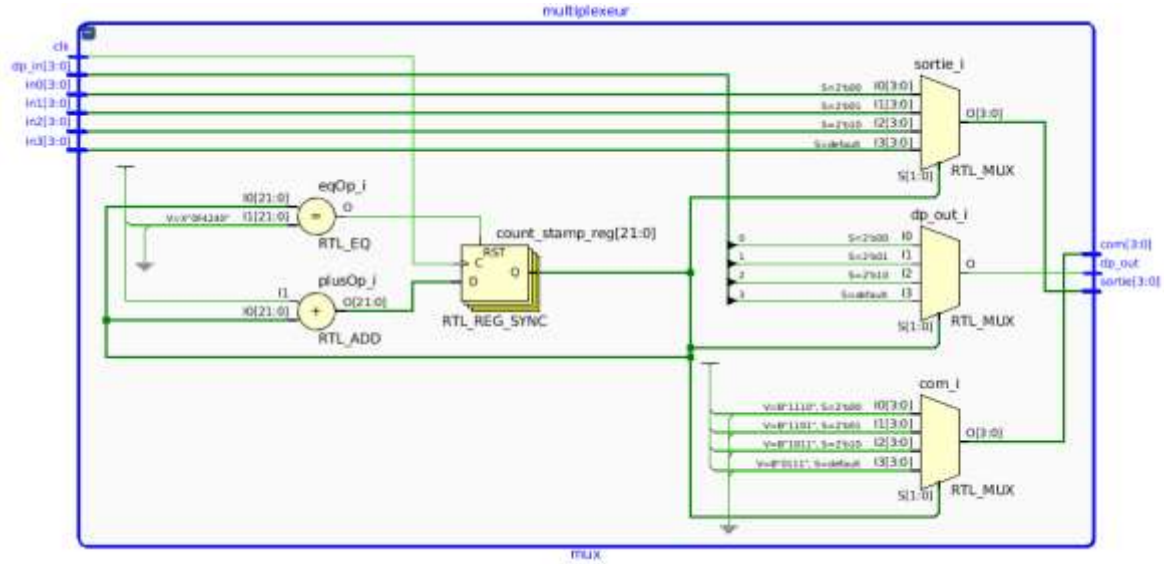
Pour ce faire nous le circuit a été décomposé comme suite : le lecteur, le multiplexeur et le codeur

I. Le lecteur

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
entity lecteur is
  Port
  (
        signal sw : in std_logic_vector(7 downto 0);
        signal hex0: out std_logic_vector(3 downto 0);
        signal hex1: out std_logic_vector(3 downto 0);
        signal hex2: out std_logic_vector(3 downto 0);
        signal hex3: out std_logic_vector(3 downto 0)
   );
end entity lecteur;

architecture lecteur of lecteur is
    signal sum : std_logic_vector (7 downto 0);
begin
    hex0 <= std_logic_vector(sw(3 downto 0));
    hex1 <= std_logic_vector(sw(7 downto 4));
   sum <= std_logic_vector(unsigned("0000"&sw(7 downto 4)) + unsigned("0000"&sw(3 downto 0)));
   hex2 <= std_logic_vector(sum(3 downto 0));
   hex3 <= std_logic_vector(sum(7 downto 4));
end lecteur;
```

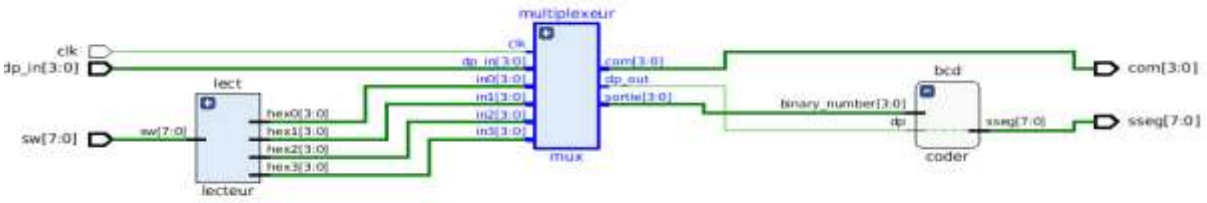
II. Le multiplexeur : Voir travail précèdent



A partir du schéma ci-dessus, on comprend mieux le fonctionnement de ce multiplexeur.

III. le codeur : Voir travail précédent

Voici une représentation du circuit conçu :



IV. La simulation du circuit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity test is
end test;

architecture test of test is
component operation is
    port(
      signal clk: in std_logic;
      signal sw: in std_logic_vector(7 downto 0);
      signal dp_in: in std_logic_vector(3 downto 0);
      signal sseg: out std_logic_vector(7 downto 0);
      signal com : out std_logic_vector(3 downto 0)
      );
end component operation;
signal clk : std_logic:='0';
signal sw :std_logic_vector(7 downto 0):=(others=>'0');
signal dp :std_logic_vector(3 downto 0):=(others=>'0');
begin
op :operation port map(clk=>clk, sw=>sw, dp_in =>dp);
process
begin
    clk<=not(clk);
    sw<=sw+'1';
    dp<=dp +'1';
    wait for 10ns;
end process;
end architecture test;
configuration config of test is
    for test
        for op :operation use entity work.operation(operation);end for;
    end for;
end configuration config;
```

Figure 14 : Programme de test de l'additionneur.

Le chronogramme ci-après décrit le fonctionnement du circuit

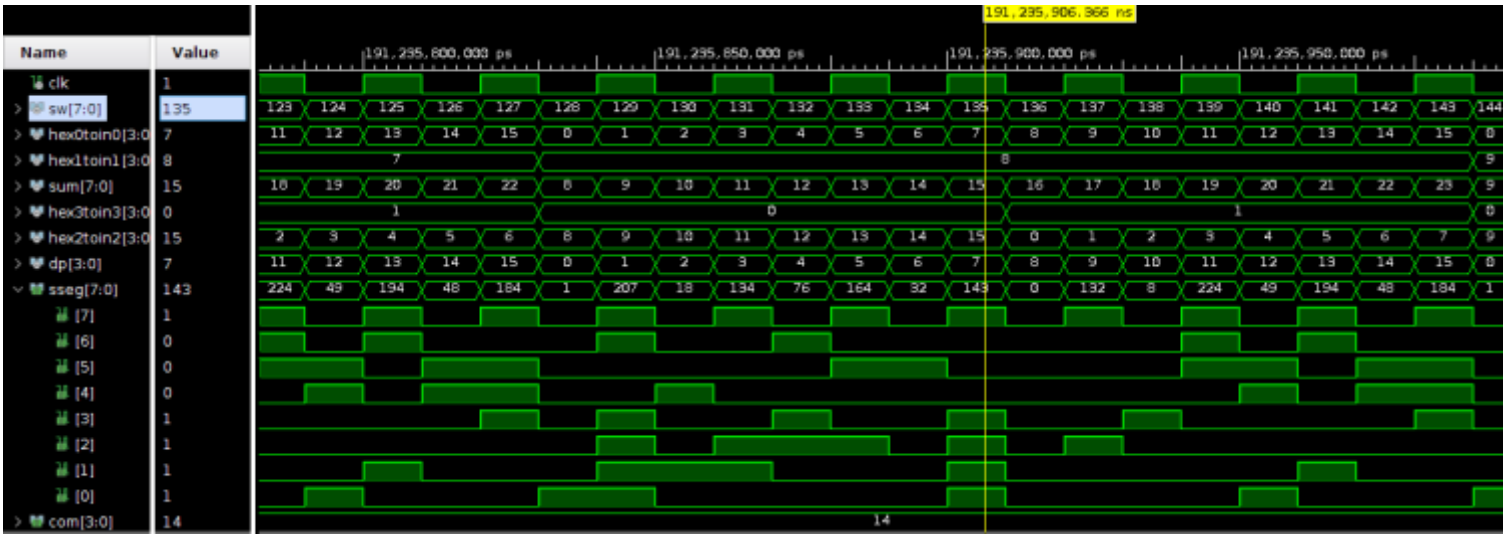


Figure 15 : résultat de la simulation du circuit additionneur

On y voit bien que la somme de hex0 avec hex1 fait belle et bien 15. Et que hex2 affiche 15tandis que hex3 affiche 0.

Travail 4 : La machine à états

Pour cette manipulation nous programmons le circuit qui permet de reconnaitre les pièces d'un de deux et de six dollars pour une machine de distribution automatique. Le schéma ci-dessous montre l'architecture programmée.

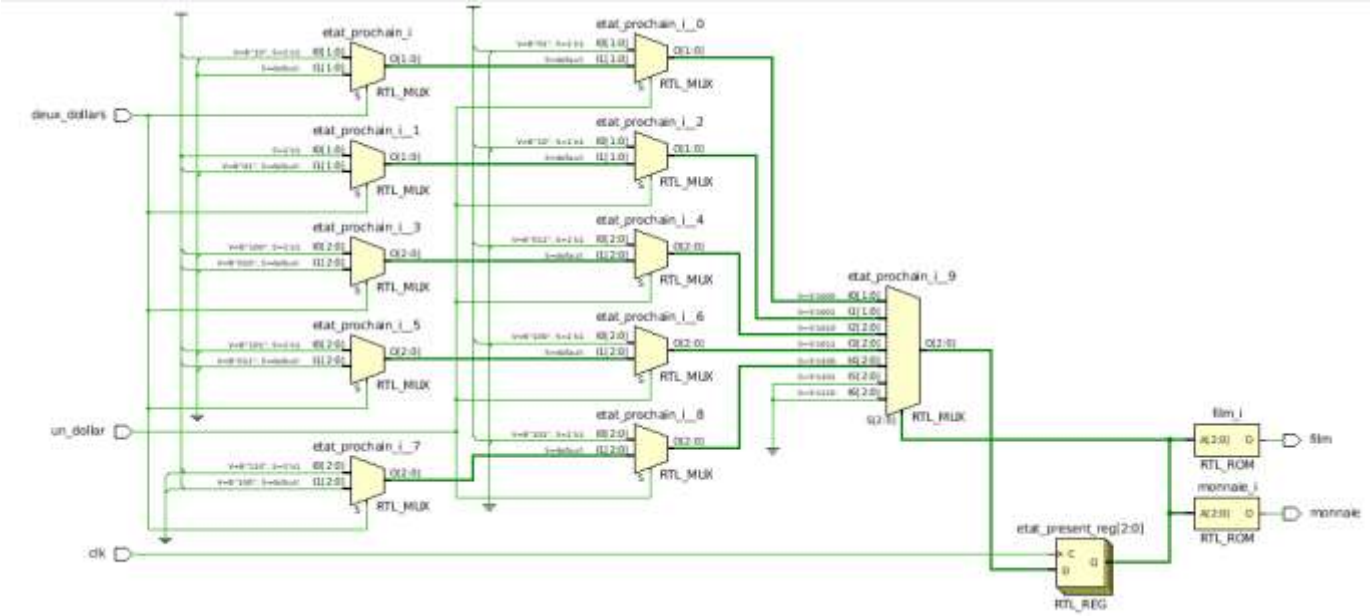


Figure 16 : Circuit de la machine à état

Procédons directement à l'écriture du code du test. Ci-après un extrait du programme de la simulation :

```
begin
   case etat_present IS
       when etat_0 => film <= '0'; monnaie <= '0';
           if un_dollar = '1' then
              etat_prochain <= etat_1;
           elsif deux_dollars = '1' then
              etat_prochain <= etat_2;
           else
              etat_prochain <= etat_0;
           end if;
      when etat_1 => film <= '0'; monnaie <= '0';
      if un_dollar = '1' then
         etat_prochain <= etat_2;
    elsif deux_dollars = '1' then
        etat_prochain <= etat_3;
    else
        etat_prochain <= etat_1;
    end if;
    when etat_2 => film <= '0'; monnaie <= '0';
    if un_dollar = '1' then
        etat_prochain <= etat_3;
    elsif deux_dollars = '1' then
        etat_prochain <= etat_4;
        else
    etat_prochain <= etat_2;
    end if;
    when etat_3 => film <= '0';  monnaie <= '0';
    if un_dollar = '1' then
        etat_prochain <= etat_4;
    elsif deux_dollars = '1' then
     etat_prochain <= etat_5;
    else
        etat_prochain <= etat_3;
    end if;
    when etat_4 =>film <= '0'; monnaie <= '0';
    if un_dollar = '1' then
        etat_prochain <= etat_5;
    elsif deux_dollars = '1' then
        etat_prochain <= etat_6;
    else
        etat_prochain <= etat_4;
    end if;
    when etat_5 => film <= '1'; monnaie <= '0'; etat_prochain <= etat_0;
    when etat_6 => film <= '1'; monnaie <= '1'; etat_prochain <= etat_0;
    end case;
end process;
Pro_2:process (clk)
begin
    if clk'event and clk = '1' then
        etat_present <= etat_prochain;
    end if;
 end process;
```

Figure 17 : Programme de la simulation du circuit de la machine à états

Conclusion

Le contexte de déroulement du cours n'a pas permis de télécharger les programmes des différents travaux dans le fpga et donc nous n'avons pas aussi procéder à l'écriture des contraintes. Cependant tous les travaux ont été faits et ceux de manières claire avec une approche structurelle.