KURKLU AHMET **GE4A-GROUPE 1**

TP ESN7

Réalisation d'un périphérique SPI

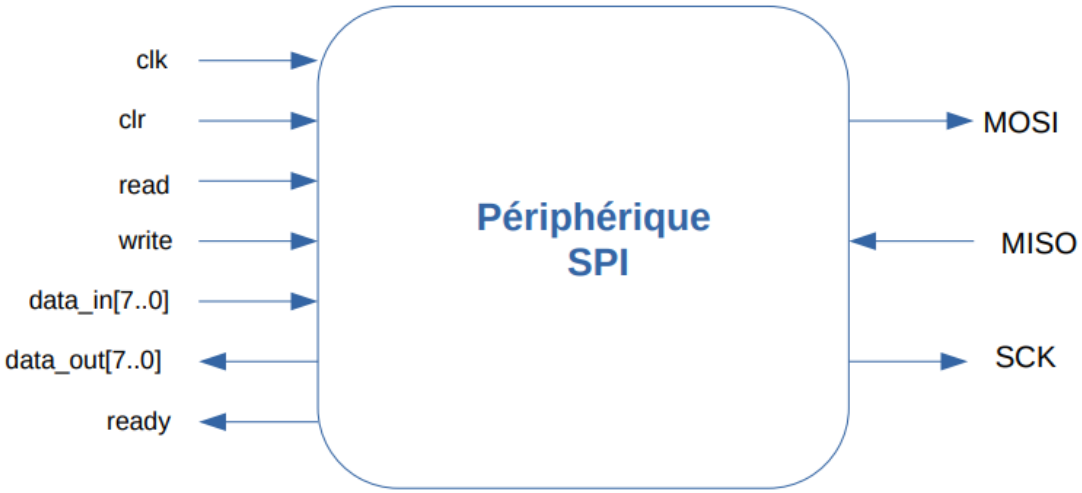
I-Introduction :

L'objectif de ce TP est de développer un périphérique SPI «simplifié » et de vérifier que son fonctionnement est conforme à son cahier des charges. Ce périphérique devra pouvoir être implanté sur un circuit FPGA et par conséquent être décrit sous forme de fichier codé en langage VHDL.

Cahier des charges

Les entrées du périphérique sont :
- **clk** : signal d'horloge qui cadence l'ensemble des opérations internes
- **clr** : signal qui met le périphérique dans l'état initial (état repos)
- **read** : signal qui commande une opération de lecture d'un mot de 8 bits
- **write** : signal qui commande une opération d'écriture d'un mot de 8 bits
- **data_in** : bus de donnée de 8 bits en entrée (utilisé pour les opérations d'écriture)
- **data_out** : bus de donnée de 8 bits en sortie (utilisé pour les opérations de lecture)
- **ready** : signal indiquant si le périphérique est occupé ou disponible pour une nouvelle commande

- **MOSI** : signal SPI (master out slave in), donnée en sortie
- **MISO** : signal SPI (master in slave out), donnée en entrée
- **SCK** : signal SPI (serial clock), horloge cadençant l'échange de donnée entre le maître et l'esclave

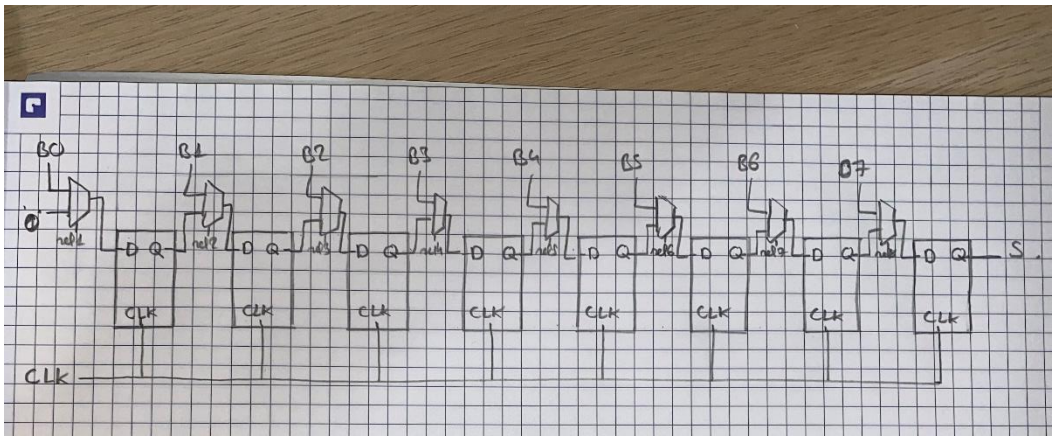


Le mode de fonctionnement souhaité est le suivant :
- Une opération d'écriture et une opération de lecture ne peuvent avoir lieu simultanément (en cas de commande simultanée, la priorité est donnée à une opération de lecture).
- tous les signaux de contrôle/commande sont actifs à l'état haut
- l'activation de la commande « clr » positionne le périphérique à l'état initial (ou de repos) et interrompt toute opération d'écriture ou de lecture le cas échéant.
- A l'état initial (ou de repos), les lignes MOSI et SCK sont à l'état bas
- Une opération d'écriture est lancée quand la ligne wr passe à « 1 » et que le périphérique est prêt (ready = « 1 »)
- Une opération de lecture est lancée quand la ligne rd passe à « 1 » et que le périphérique est prêt (ready = « 1 »)
- le signal ready est à l'état « 0 » lorsque le périphérique n'est pas au repos, autrement dit lorsqu'une opération d'écriture ou de lecture est en cours.

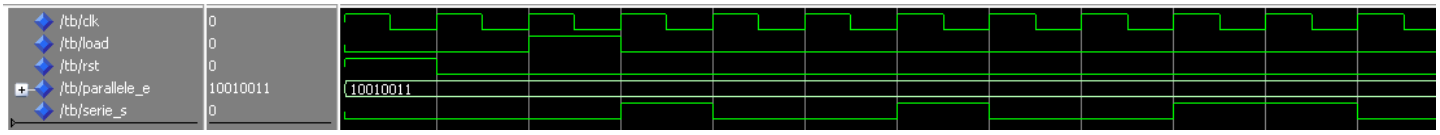
La fréquence d'horloge est égale à 100 MHz et la fréquence de la liaison SPI est 5 MHz (SCK est un signal carré). La configuration SPI sera réglée sur le mode 0.

II-Préparation :

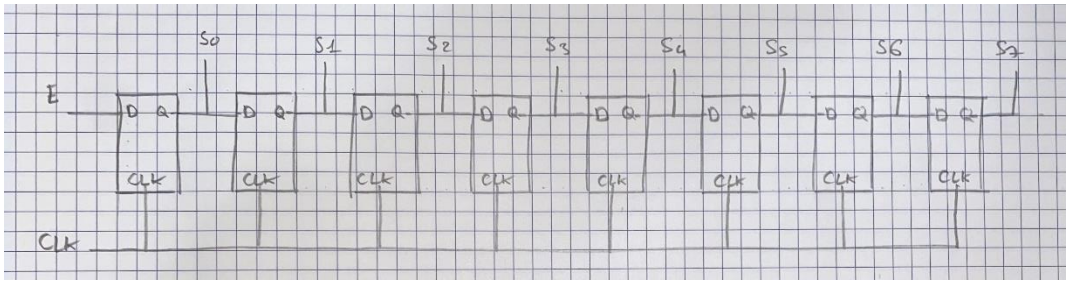
1) Schéma registre 8bits PISO



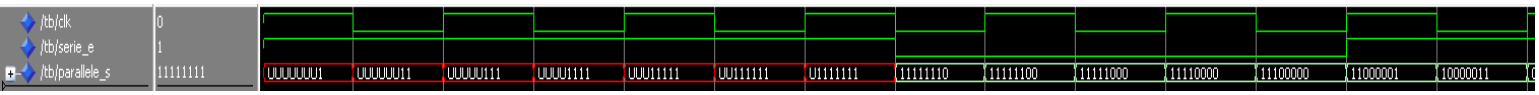
On obtient le chronogramme suivant en simulant le code VHDL de PISO :



2) Schéma registre 8bits SIPO



On obtient le chronogramme suivant en simulant le code VHDL de SIPO :



III-Conception :

Afin de réaliser le périphérique SPI j'ai décidé d'utiliser une machine à état. Elle comporte 5 états qui sont les suivants :

-***Repos*** : l'état d'attente dans lequel se trouve le périphérique quand l'entrée ***clr*** est a '1'. On attend que l'entrée ***rd*** ou ***wr*** passe a '1' pour commencer une action d'écriture ou de lecture. Le passage de ***rd*** a '1' fait passer la sortie ***rdy*** a '0', le compteur ***cpt*** et ***cpt_sck*** sont initialiser a 0 et l'état de la machine passe à l'état ***Read***. Le passage de ***wr*** a '1' fait passer la sortie ***rdy*** a '0' ***wirte_register*** prend le mot sur l'entrée ***data_in*** et l'état passe à ***WaitWrite***. Dans cet état, la sortie ***MISO*** et ***SCK*** sont à l'état bas donc '0'.

-***Read*** : l'état ***Read*** permet de faire la lecture. Dans cet état à chaque front descendant de ***SCK*** un décalage vers la gauche, est fait sur ***shift_register*** avec la valeur sur l'entrée ***MISO***. Au bout de 8 décalages gérer par le compteur ***cpt*** , ***read_register*** prend le mot stocker dans ***shift_register*** et la machine passe à l'état ***WaitRead***.

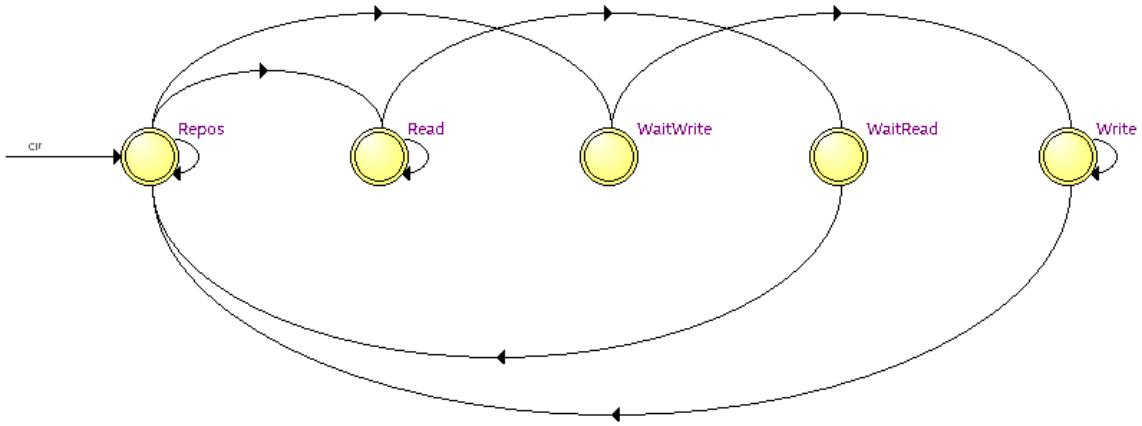
-***WaitRead*** : l'état ***WaitRead*** permet de simuler la copie de ***read_register*** sur ***data_out*** qui prend un coup d'horloge, pour ne pas faire cette copie instantanément. La machine passe ensuite à l'état ***Repos*** .

-***WaitWrite*** : L'état ***WaitWrite*** permet de simuler la copie de ***write_register*** sur ***shift_register*** qui prend un coup d'horloge. Le bit 7 de ***shift_register*** est placé sur la sortie ***MOSI***. ***cpt_sck*** est ***cpt*** sont initialisé à 0 et la machine passe à l'état ***Write***.

-***Write*** : l'état Write permet de faire l'ecriture. Dans cet état à chaque front descendant de SCK un décalage vers la gauche, est fait sur ***shift_register*** avec la valeur sur l'entrée ***MISO*** puis le 7ieme bit de ***shift_register*** est placé sur la sortie ***MOSI***. Au bout de 8 décalages gérer par le compteur ***cpt***, la machine passe à l'état ***Repos***.

Les décalages vers la gauche sur le registre ***shift_register*** dans les états ***Write*** et ***Read*** sont réaliser à l'aide d'une fonction ***d_reg_dec***. Cette fonction prend deux paramètres un std_logic_vector ***X*** et un std_logic ***dec***. La fonction effectue une concaténation entre les bits(6-0) de ***X*** et le bit ***dec***, ce qui correspond à un décalage vers la gauche plus le bit ***dec***.

L'horloge ***clk*** du système est cadencée à une fréquence de 100Mhz et la liaison SPI a 5Mhz ce qui est 20 fois moins rapide. Afin de répondre à ce problème, sachant que ***SCK*** est généré uniquement pendant la lecture et l'écriture, j'utilise un compteur ***cpt_sck*** initialiser à 0 quand on rentre dans un des deux états. Ce compteur doit être incrémenté 20 fois cadencé par des fronts montants de l'horloge ***clk***. De plus on place un signal ***sortie*** à l'état bas ce signal est reporté sur la sortie ***SCK*** a la fin du process. Dans un état ***Read*** ou ***Write*** le compteur ***cpt_sck*** est incrémenté, au bout de 10 incrémentations, on inverse l'état du signal ***sortie*** avec la fonction ***not(sortie).*** Puis au bout de 20 incrémentations, on inverse à nouveau le signal ***sortie*** et on place ***cpt_sck*** a 0. Cette action est répétée 8 fois contrôlée par ***cpt*** pour gérer le nombre de décalage. À la fin, on obtient bien un signal ***SCK*** cadencé a 5Mhz 20 fois moins rapides que le l'horloge ***clk*** de 100Mhz.



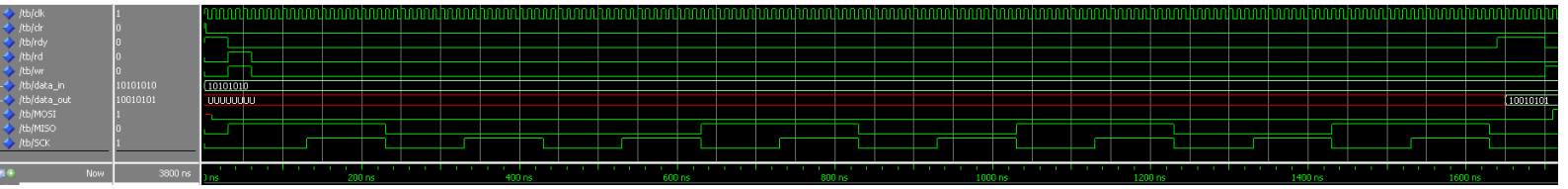
On obtient cette machine à état à l'aide de Quartus.

IV-Vérification:

Pour vérifier que notre périphérique fonctionne correctement, nous allons réaliser des tests afin de valider le mode de fonctionnement souhaité par le cahier des charges qui sont pour le rappeler les suivants:

Le mode de fonctionnement souhaité est le suivant :
- Une opération d'écriture et une opération de lecture ne peuvent avoir lieu simultanément (en cas de commande simultanée, la priorité est donnée à une opération de lecture).
- tous les signaux de contrôle/commande sont actifs à l'état haut
- l'activation de la commande « clr » positionne le périphérique à l'état initial (ou de repos) et interrompt toute opération d'écriture ou de lecture le cas échéant.
- A l'état initial (ou de repos), les lignes MOSI et SCK sont à l'état bas
- Une opération d'écriture est lancée quand la ligne wr passe à « 1 » et que le périphérique est prêt (ready = « 1 »)
- Une opération de lecture est lancée quand la ligne rd passe à « 1 » et que le périphérique est prêt (ready = « 1 »)
- le signal ready est à l'état « 0 » lorsque le périphérique n'est pas au repos, autrement dit lorsqu'une opération d'écriture ou de lecture est en cours.

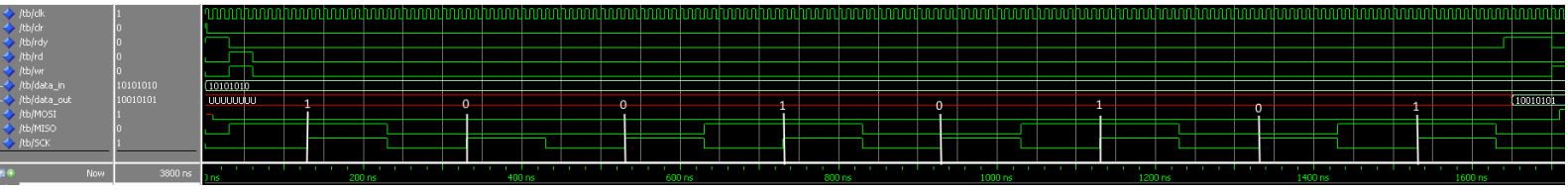
La fréquence d'horloge est égale à 100 MHz et la fréquence de la liaison SPI est 5 MHz (SCK est un signal carré). La configuration SPI sera réglée sur le mode 0.



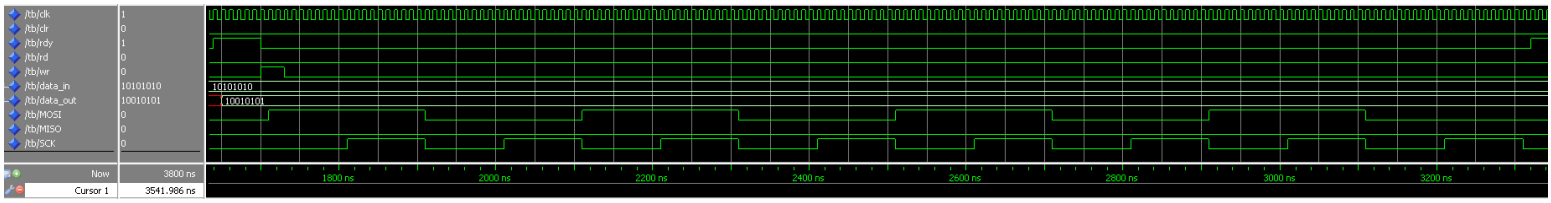
Dans cette première capture d'écran, on remarque le signal ***clr*** a '1' sur le premier front montant cela passe ***rdy*** a '1', la sortie ***MOSI*** et ***SCK*** sont à l'état bas '0', on est dans l'état ***Repos***. À 30 ns le signal ***rd*** et ***wr*** passe a '1' simultanément on remarque alors que l'opération de lecture ce lance car elle est prioritaire, on est dans l'état ***Lecture***. Le signal ***rdy*** passe alors a '0', l'horloge ***SCK*** se lance et à chaque front montant, on lit la valeur de l'entrée ***MISO***. Le système utilise la configuration CPOL = 0 et CPHA = 0 donc le changement de bit ***MISO*** se fait à chaque front descendant de ***SCK*** et la lecture à chaque front montant de ***SCK***. Le signal ***SCK*** a bien 8 fronts montants qui correspond bien au 8 bits d'un octet. A la fin du dernier front

descendant de ***SCK*** la machine, passe dans l' état ***WaitRead*** afin de faire la copie de ***shift_register*** sur ***data_out*** qui est ensuite afficher.

On vérifie le résultat sur ***data_out*** :

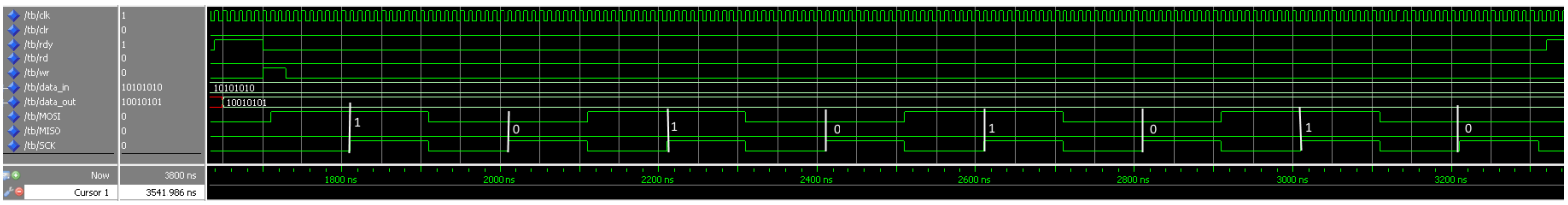


Les fronts montants de ***SCK*** sont marqués par des lignes blanches, le mot à lire est donc le suivant « 10010101 ». On remarque ensuite à la fin de la Lecture que le mot afficher sur ***data_out*** est bien le même mot « 10010101 ». À la fin de la lecture ***SCK***, passe a '0' et ***rdy*** passe a '1', la machine passe à l'état ***Repos***.



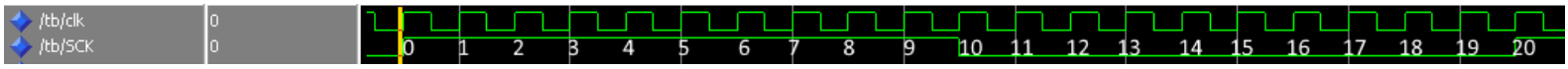
Dans cette deuxième capture d'écran, nous allons tester l'écriture. Le signal ***wr*** passe a '1' à 1700 ns, l'opération d'écriture se lance ***rdy*** passe a '0'. La machine passe d'abord dans l'état ***WaitWrite*** pour faire la copie de ***data_in*** sur ***shift_register*** pendant un coup d'horloge. La machine passe ensuite dans l'état ***Write***. ***SCK*** se lance et à chaque front montant, on place le bit 7 de ***shift_register*** sur la sortie ***MOSI***. Le système utilise la configuration CPOL = 0 et CPHA = 0 donc l'écriture sur ***MOSI*** ce fait à chaque front descendant de ***SCK***. On remarque que l'écriture se fait bien 8 fois pour un octet, car on a 8 fronts montants de ***SCK***. À la fin le signal ***rdy*** passe a '1', ***SCK*** passe a '0' et ***MOSI*** passe a '0'. La machine revient alors sur l'état ***Repos***.

On vérifie le résultat sur ***MOSI*** :



Le mot sur ***data_in*** est « 10101010 », les fronts montant de ***SCK*** sont marqués par des lignes blanches. On remarque que le mot sur ***data_in*** se retrouve bien sur la sortie ***MOSI*** donc l'écriture fonctionne correctement.

Nous allons maintenant vérifier le signal SCK :



On remarque bien qu'une période du signal ***SCK*** dure sur 20 périodes du signal ***clk*** donc le cahier des charges est bien respecté.

V-Conclusion:

Pendant ce TP, nous avons conçu un périphérique SPI simplifié sur le logiciel Quartus en simulant à l'aide de ModelSim. Nous avons vu comment implanter un registre à décalage et comment gérer un système qui fonctionne avec deux horloges. Nous avons aussi vu comment implémenter notre système sur un FPGA

V-Annexe:

-SIPO

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

entity SIPO is
    port(
        clk : in std_logic;
        serie_e : in std_logic;
        parallele_s : out std_logic_vector(7 downto 0));
end entity;

architecture seq of SIPO is
    signal tmp : std_logic_vector(7 downto 0);

begin
    process(clk)
    begin
        tmp<=tmp(6 downto 0)&serie_e;
    end process;
    parallele_s<= tmp;
end architecture;
```

-PISO

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

entity PISO is
    port(
        clk : in std_logic;
        load : in std_logic;
        rst : in std_logic;
        parallele_e : in std_logic_vector(7 downto 0);
        serie_s : out std_logic);
end entity;

architecture seq of PISO is
    type t_state is (Repos, Calcule);
    signal state : t_state;
    signal tmp : std_logic_vector(7 downto 0);

begin

    process(clk,rst)
    begin
        if(rst = '1') then
            serie_s <='0';
            state <= Repos;
        elsif(rising_edge(clk))then
            case state is
            when Repos=>
                if(load = '1')then
                    state<=Calcule;
                    tmp<= parallele_e;
                end if;
            when Calcule=>
                serie_s<=tmp(7);
                tmp<=tmp(6 downto 0)&'0';
            end case;
        end if;
    end process;
end architecture;
```

-SPI

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity SPI is
5    port (
6          clk: in std_logic;
7          clr: in std_logic;
8          rd: in std_logic;
9          wr: in std_logic;
10         data_in: in std_logic_vector(7 downto 0);
11         data_out: out std_logic_vector(7 downto 0);
12         rdy : out std_logic;
13         MOSI : out std_logic;
14         MISO : in std_logic;
15         SCK: out std_logic);
16 end entity;
17
18 architecture seq of SPI is
19
20    --Fonction qui permet de faire le décalage a gauche
21    function f_reg_dec (x: std_logic_vector; dec : std_logic) return std_logic_vector is
22       begin
23          return x(6 downto 0) & dec;
24    end;
25
26    --Déclaration du type pour la machine a état
27    type t_state is (Repos, WaitRead,WaitWrite,Read,Write);
28    signal state : t_state;
29
30    --Compteurs
31    signal cpt : integer range 0 to 8;
32    signal cpt_sck : integer range 0 to 20;
33    --signal intermediare qui va etre appliquer sur SCK
34    signal sortie : std_logic;
35
36    signal write_register : std_logic_vector(7 downto 0);
37    signal read_register : std_logic_vector(7 downto 0);
38
39 begin
40    process(clr,clk)
41    variable shift_register : std_logic_vector(7 downto 0);
42    begin
43       if(clr = '1') then
44          state <= Repos;
45          rdy <= '1';
46          sortie <='0';
47       elsif(rising_edge(clk))then
48          case state is
49          when Repos =>      --Etat Repos dans lequel MOSI et SCK sont
50             rdy <= '1';     -- a l'etat bas
51             MOSI <= '0';
52             sortie <='0';
53             if(rd ='1')then --Quand rd = '1' on passe dans l'etat Read
54                rdy<='0';
55                state<=Read;
56                cpt <= 0;
57                cpt_sck <= 0;
58             elsif(wr = '1')then  --Quand rd = '1' on passe dans l'etat WaitWrite
59                rdy <= '0';       --on copie data_in sur write_register
60                state <= WaitWrite;
61                write_register <= data_in;
62             end if;
63          when WaitWrite => --Etat WaitWrite permet de faire une temporisation de
64             shift_register := write_register; -- un cout d'horloge et le premier bit
65             MOSI <= shift_register(7); --dans shift register est ecrit sur MOSI
66             state <= Write; -- On passe ensuite a l'etat Write
67             cpt_sck <= 0;
68             cpt <= 0;
```

```
69          when Write =>        --Etat Write pour l'ecriture
70             if(cpt < 8)then
71                if(cpt_sck < 19) then   --SCK controler par cpt_sck on inverse ça valeur
72                   cpt_sck <= cpt_sck+1;-- tout les 10 cout d'horloge de clk
73                   if(cpt_sck = 9)then
74                      sortie <= not(sortie);
75                   end if;
76                else
77                   cpt_sck <= 0;     --A chaque fois que SCK est sur une nouvel periode
78                   sortie <= not(sortie);-- on fait un décalage a gauche sur shift_register
79                   cpt <= cpt +1;       --et on ecrit le 7ieme bit sur MOSI
80                   shift_register := f_reg_dec(shift_register,MISO);
81                   MOSI <= shift_register(7);
82                end if;
83             else
84                sortie <= '0'; -- au bout de 8 décalage et ecriture on passe a l'etat Repos
85                rdy <= '1';
86                MOSI <='0';
87                state <= Repos;
88             end if;
89          when Read =>   --Etat Read pour la lecture
90             if(cpt < 8)then
91                if(cpt_sck < 19) then  --SCK gerer de la même façon que l'etat Write
92                   cpt_sck <= cpt_sck+1;
93                   if(cpt_sck = 9)then
94                      sortie <= not(sortie);
95                      --A chaque front montant de SCK on lit la valeur sur MISO
96                      shift_register := f_reg_dec(shift_register,MISO);
97                   end if;
98                else
99                   cpt_sck <= 0;
100                  sortie <= not(sortie);
101                  cpt <= cpt +1;
102               end if;
103            else
104               sortie <= '0'; --au bout de 8 lecture on passe a l'etat WaitRead
105               rdy <= '1';
106               read_register <= shift_register;
107               state <= WaitRead;
108            end if;
109         when WaitRead => --etat Wait read
110            data_out <= read_register;--Perme une temporisation de un cout d'horloge clk
111            state <= Repos;--Pour la copie de read_register sur data_out
112         end case; --On passe ensuite a l'etat Repos
113      end if;
114   end process;
115   SCK <=sortie when state = Read else
116         sortie when state = Write else
117         '0';
118end architecture;
```

-tb

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

entity tb is
end tb;

architecture seq of tb is

  signal clk: std_logic;
  signal clr: std_logic;
  signal rd: std_logic;
  signal wr: std_logic;
  signal data_in: std_logic_vector(7 downto 0);
  signal data_out: std_logic_vector(7 downto 0);
  signal rdy : std_logic;
  signal MOSI : std_logic;
  signal MISO : std_logic;
  signal SCK: std_logic;

begin

  UUT: entity work.SPI port map(clk, clr, rd, wr, data_in, data_out, rdy, MOSI, MISO, SCK);

  CLOCK: process
  begin
    clk<='1';
    wait for 5 ns;
    clk<='0';
    wait for 5 ns;
  end process;

  RESET:process
  begin
    clr <= '1';
    wait for 3 ns;
    clr <= '0';
    wait;
  end process;

  ECRITURE:process
  begin
    wr <= '0';
    wait for 30 ns;
    wr <= '1';
    wait for 30 ns;
    wr <= '0';
    wait for 1640 ns;
    wr<= '1';
    wait for 30 ns;
    wr <='0';
    wait;
  end process;

  LECTURE:process
  begin
    rd <= '0';
    wait for 30 ns;
    rd <= '1';
    wait for 30 ns;
    rd <= '0';
    wait;
  end process;

  ENTREE_DATA:process
  begin
    data_in <= "10101010";
    wait;
  end process;

  ENTREE_SPI:process
  begin
    MISO <= '0';
    wait for 30 ns;
    MISO <= '1';
    wait for 200 ns;
    MISO <= '0';
    wait for 200 ns;
    MISO <= '0';
    wait for 200 ns;
    MISO <= '1';
    wait for 200 ns;
    MISO <= '0';
    wait for 200 ns;
    MISO <= '1';
    wait for 200 ns;
    MISO <= '0';
    wait for 200 ns;
    MISO <= '1';
    wait for 200 ns;
    MISO <= '0';
    wait;
  end process;

end architecture;
```