ENSAM
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

Implémentation de l'Enigma en VHDL

Prepared by :

Ali nechchadi
Omar nechchadi
Abir Werzgan

1. Introduction

Contexte historique de l'Enigma

L'Enigma a été développée dans les années 1920 par l'ingénieur allemand Arthur Scherbius. Son utilisation par les forces armées allemandes a été une composante clé de leurs communications sécurisées pendant la Seconde Guerre mondiale. Cependant, les efforts des cryptanalystes alliés, en particulier à Bletchley Park sous la direction d'Alan Turing, ont réussi à percer le code Enigma, ce qui a eu un impact significatif sur le conflit.

Objectif du projet

- Développer des modules VHDL pour chaque composant de l'Enigma, y compris les rotors, les réflecteurs et l'incrimenteur.
- Assembler les modules développés précédemment pour former une machine Enigma complète, capable de chiffrer et de déchiffrer des messages.
- Utiliser des outils de simulation VHDL pour vérifier que la machine Enigma fonctionne correctement selon les spécifications historiques.

1. Principe de fonctionnement de enigma

Tout d'abord, les lettres du message à chiffrer étaient saisies sur le clavier de l'Enigma. Lorsqu'une touche était enfoncée, un courant électrique traversait une série de rotors, chacun étant un disque avec des fils électriques à l'intérieur. Ces rotors pouvaient être positionnés de différentes manières, offrant ainsi des milliards de configurations possibles.

Le courant électrique passait ensuite à travers le plugboard, une série de câbles qui échangeaient les lettres entrantes avec d'autres lettres. Cette étape offrait encore plus de combinaisons potentielles.

Ensuite, le courant traversait les rotors à nouveau, mais cette fois-ci dans l'ordre inverse. Les rotors fonctionnaient comme des substituts alphabétiques, modifiant la lettre initiale en une autre lettre en fonction de leur configuration.

Le courant atteignait ensuite le réflecteur, qui renvoyait le signal à travers les rotors une dernière fois, inversant à nouveau les substitutions. Ce processus complexe de substitution multiple était ce qui rendait l'Enigma si difficile à décoder.

Finalement, la lettre chiffrée s'affichait sur une lampe, indiquant ainsi la lettre correspondante du message chiffré. Le destinataire devait configurer son Enigma avec les mêmes paramètres pour déchiffrer le message.

En résumé, l'Enigma fonctionnait en prenant une lettre d'entrée, en la passant à travers une série de substitutions complexes et en renvoyant une lettre chiffrée. Ce processus garantissait la sécurité des communications allemandes pendant une grande partie de la guerre, jusqu'à ce que les Alliés réussissent à percer son code, un exploit qui a eu un impact significatif sur le déroulement du conflit.

3. Développement des modules

Modules en vhdl

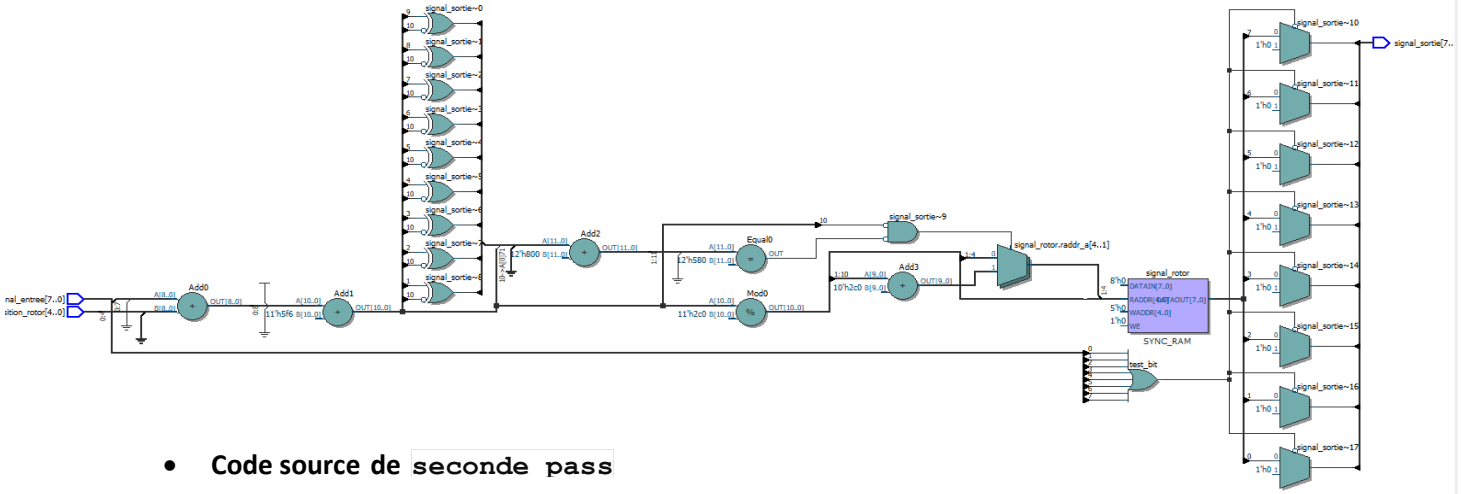
- Rotor : Implémentation du rotor Enigma

À la fin de la création d'un module VHDL pour le rotor, on divise le comportement en deux parties. La première partie décrit le premier passage du rotor, tandis que la deuxième décrit le deuxième passage du rotor. On a nommé la première partie `first_pass` et la deuxième `second_pass`. Voici le code source de chaque partie ainsi que le schéma électrique correspondant.

- **Code source de `first pass`**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std .ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity first_pass is
 port(
          signal_entree   : in  STD_logic_vector(7 downto 0);
          signal_sortie   : out STD_logic_vector(7 downto 0);
          position_rotor  : in  INTEGER range 0 to 31   );
end first_pass ;
architecture code of first_pass is
signal test_bit : STD_LOGIC ;
 type rotor_type is array(0 to 25) of std_logic_vector(7 downto 0);
                signal signal_rotor : rotor_type := (
     "01000001", "01000010", "01000011", "01000100", "01000101",
     "01000110", "01000111", "01001000", "01001001", "01001010",
     "01001011", "01001100", "01001101", "01001110", "01001111",
     "01010000", "01010001", "01010010", "01010011", "01010100",
     "01010101", "01010110", "01010111", "01011000", "01011001",
     "01011010");
begin
  process(signal_entree)
  begin
  -- tester si on a un signal a l'entree
  test_bit <= '0';
  for i in signal_entree'range loop
       if signal_entree(i) = '1' then
       test_bit <= '1';
       exit;
       end if;
       end loop;
  --debut de chiffrement
       if test_bit = '0' then
       signal_sortie <= "00000000" ;
       else
       signal_sortie <= signal_rotor((to_integer(unsigned(signal_entree)) + position_rotor
- 65 ) MOD 26 );
       end if;
       end process ;
end code ;
```

- **le schéma électrique de `first_pass`**

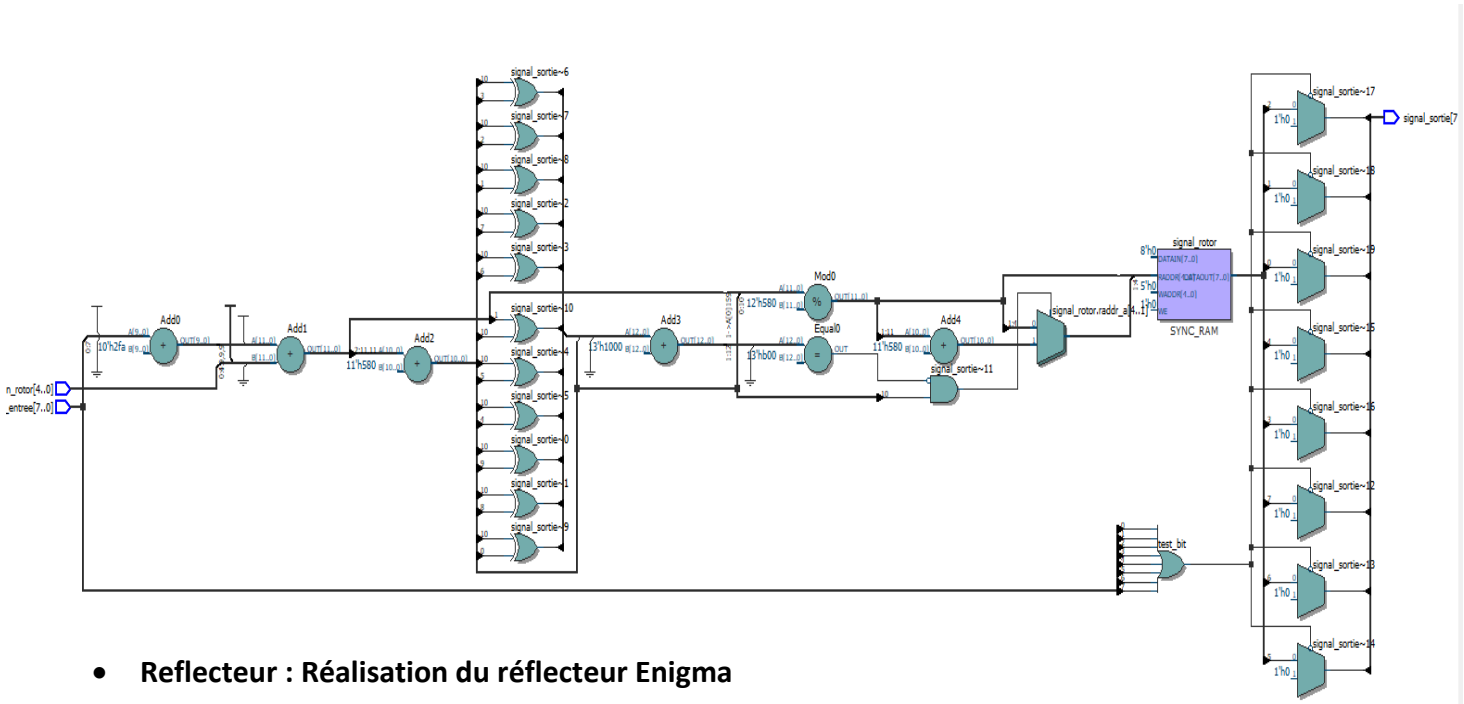


- **Code source de `seconde pass`**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std .ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity second_pass is
port(
                signal_entree  : in  STD_LOGIC_VECTOR(7 downto 0);
                signal_sortie  : out STD_LOGIC_VECTOR(7 downto 0);
                position_rotor : in  INTEGER range 0 to 31
    );
end second_pass;
architecture code of second_pass is
signal test_bit : STD_LOGIC ;
type rotor_type is array(0 to 25) of std_logic_vector(7 downto 0);
signal signal_rotor : rotor_type := (
    "01000001", "01000010", "01000011", "01000100", "01000101",
    "01000110", "01000111", "01001000", "01001001", "01001010",
    "01001011", "01001100", "01001101", "01001110", "01001111",
    "01010000", "01010001", "01010010", "01010011", "01010100",
    "01010101", "01010110", "01010111", "01011000", "01011001",
    "01011010");
begin
  process(signal_entree)
      begin
            -- tester si on a un signal a l'entree
            test_bit <= '0';
    for i in signal_entree'range loop
                  if signal_entree(i) = '1' then
                        test_bit <= '1';
                        exit;
                  end if;
            end loop;
            --debut de chiffrement
            if test_bit = '0' then
                  signal_sortie <= "00000000" ;
            else
```

```
signal_sortie <= signal_rotor((to_integer(unsigned(signal_entree))- 65 - position_rotor
+ 26) MOD 26   );
                end if ;
      end process ;
end code ;
```

- **le schéma électrique de `second_pass`**



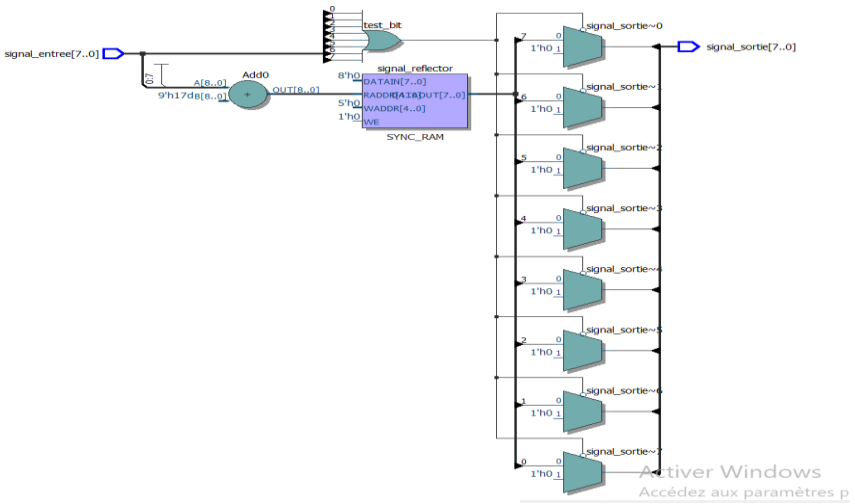
- **Reflecteur : Réalisation du réflecteur Enigma**

  - Code source :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std .ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity reflecteur is
port(
                signal_entree : in  STD_LOGIC_VECTOR(7 downto 0);
                signal_sortie: out STD_LOGIC_VECTOR(7 downto 0)
    );
end reflecteur;
architecture code of reflecteur is
signal test_bit : STD_LOGIC ;
type rotor_type is array(0 to 25) of std_logic_vector(7 downto 0);
signal signal_reflector : rotor_type := (
     "01000101", "01001010", "01001101", "01000100", "01000001",
     "01001100", "01011001", "01011000", "01010110", "01000010",
     "01010111", "01000110", "01000011", "01010010", "01010001",
     "01010101", "01001111", "01010010", "01010100", "01010011",
     "01010000", "01001001", "01001011", "01011000", "01000111",
     "01000100");
begin
  process(signal_entree)
       begin
```

```
for i in signal_entree'range loop
                    if signal_entree(i) = '1' then
                                test_bit <= '1';
                                exit;
                    end if;
            end loop;
            --debut de chiffrement
            if test_bit = '0' then
                    signal_sortie <= "00000000" ;
            else
     signal_sortie <= signal_reflector(to_integer(unsigned(signal_entree)) - 65 );
     end if;
     end process ;
end code ;
```

- le schéma électrique :



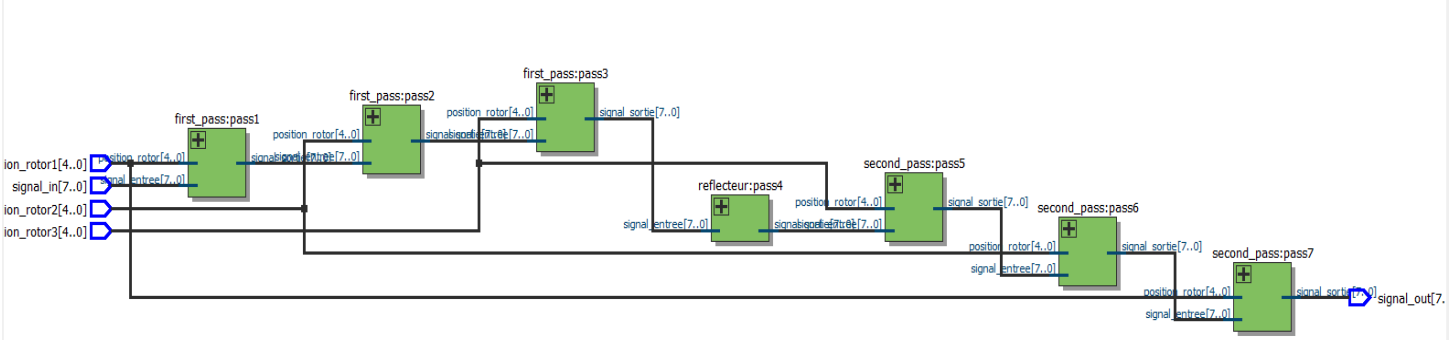
- On regroupe ces composants dans le code suivant :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;
entity Enigma is
    port (
        signal_in           : in  STD_LOGIC_VECTOR(7 downto 0);
        signal_out          : out STD_LOGIC_VECTOR(7 downto 0);
        position_rotor1     : IN INTEGER range 0 to 31;
        position_rotor2     : IN INTEGER range 0 to 31;
        position_rotor3     : IN INTEGER range 0 to 31
    );
end Enigma;
architecture Machine of Enigma is
    component first_pass
        port (
            signal_entree  : in  STD_LOGIC_VECTOR(7 downto 0);
            position_rotor : in  INTEGER range 0 to 31;
            signal_sortie  : out STD_LOGIC_VECTOR(7 downto 0)
        );
    end component;
```

```
component second_pass
      port (
              signal_entree  : in  STD_LOGIC_VECTOR(7 downto 0);
              signal_sortie  : out STD_LOGIC_VECTOR(7 downto 0);
              position_rotor : in  INTEGER range 0 to 31
      );
   end component;
   component reflecteur
      port (
         signal_entree  : in  STD_LOGIC_VECTOR(7 downto 0);
         signal_sortie  : out STD_LOGIC_VECTOR(7 downto 0)
      );
   end component;
   signal incriment              : STD_LOGIC;
   signal test_bit               : STD_LOGIC;
   signal position_rotor_out1  : INTEGER range 0 to 31 ;
   signal position_rotor_out2  : INTEGER range 0 to 31 ;
   signal position_rotor_out3  : INTEGER range 0 to 31 ;
   signal signal_interne1       : STD_LOGIC_VECTOR(7 downto 0);
   signal signal_interne2       : STD_LOGIC_VECTOR(7 downto 0);
   signal signal_interne3       : STD_LOGIC_VECTOR(7 downto 0);
   signal signal_interne4       : STD_LOGIC_VECTOR(7 downto 0);
   signal signal_interne5       : STD_LOGIC_VECTOR(7 downto 0);
   signal signal_interne6       : STD_LOGIC_VECTOR(7 downto 0);
   signal signal_interne7       : STD_LOGIC_VECTOR(7 downto 0);
begin
   pass1 : first_pass  port map (
      signal_entree  => signal_in,
      position_rotor =>   position_rotor1 ,
      signal_sortie  => signal_interne1
   );
   pass2 : first_pass  port map (
      signal_entree  => signal_interne1,
      position_rotor =>  position_rotor2,
      signal_sortie  => signal_interne2
   );
   pass3 : first_pass  port map (
      signal_entree  => signal_interne2,
      position_rotor =>   position_rotor3,
      signal_sortie  => signal_interne3
   );
   pass4 : reflecteur  port map (
      signal_entree  => signal_interne3,
      signal_sortie  => signal_interne4
   );
   pass5 : second_pass port map (
      signal_entree  => signal_interne4,
      position_rotor => position_rotor3,
      signal_sortie  => signal_interne5
   );
```

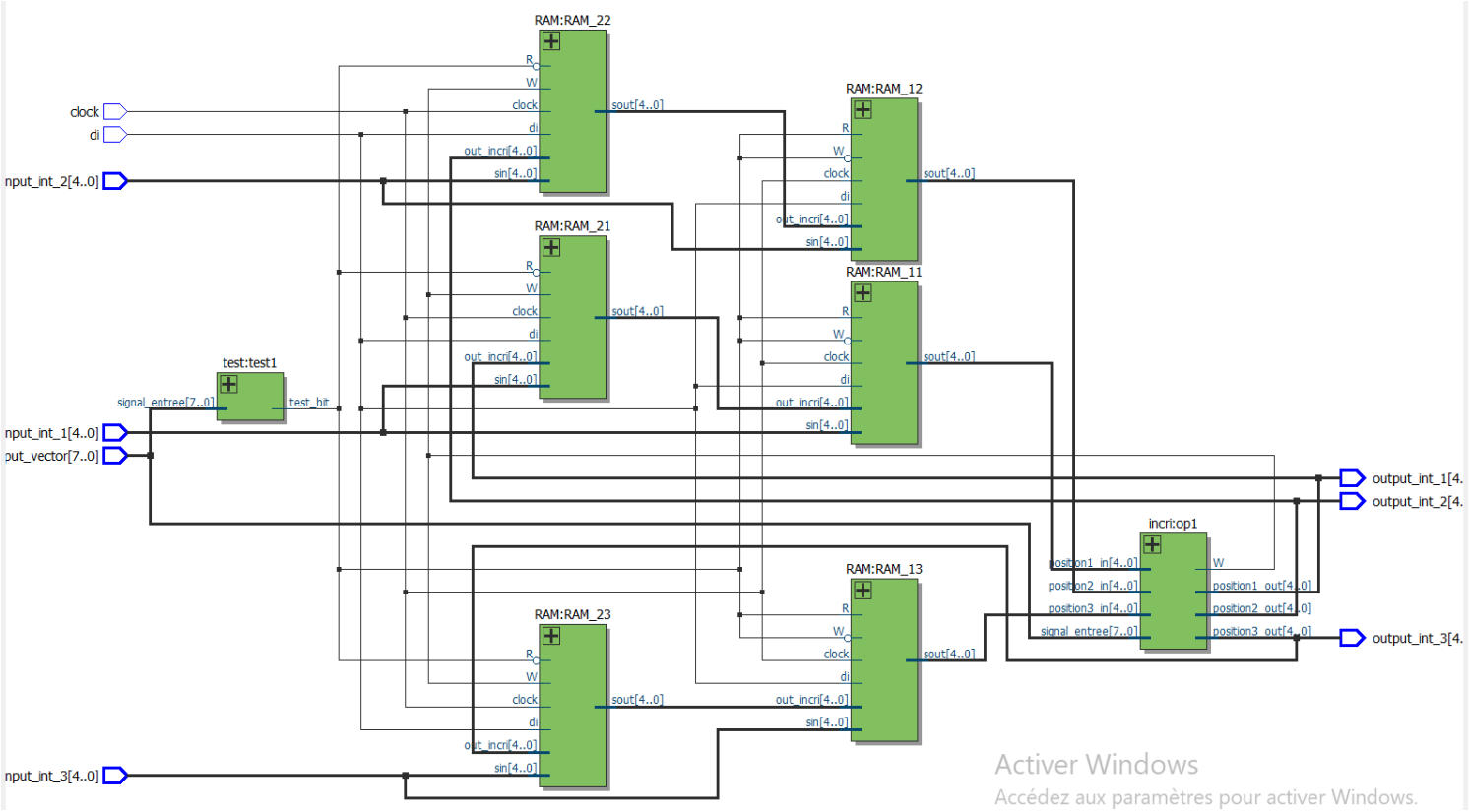
```
    pass6 : second_pass port map (
        signal_entree  => signal_interne5,
        position_rotor => position_rotor2,
        signal_sortie  => signal_interne6
    );
    pass7 : second_pass port map (
        signal_entree  => signal_interne6,
        position_rotor => position_rotor1,
        signal_sortie  => signal_interne7
    );
    signal_out <= signal_interne7;

end Machine;
```

- **Shema electrique Enigma**



- **Incrementeur des position des rotors**

  - Shema electrique d'incrementeur



Le rôle des RAM_11, RAM_12 et RAM_13 est de stocker les positions des rotors insérées par l'utilisateur. Les RAM_21, RAM_22 et RAM_23 ont pour rôle de stocker les positions après l'incrémentation. Le bloc incri a pour fonction d'incrémenter les positions des rotors. Quant au bloc test, il délivre un signal std_logic pour chaque passage du signal à crypter.

- Code de developement du block RAM
  - Code source de la bascule

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

entity bascul is
    Port ( clk, d : in  STD_LOGIC;
           q : out  STD_LOGIC);
end bascul;

architecture code of bascul is
begin
    process(clk)
    begin
        if rising_edge(clk) then
            q <= d;
        end if;
    end process;
end code;
```

  - Code source de la memoire

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

entity memoire  is
    Port ( DIN, CLOCK, W, R : in  STD_LOGIC;
           DOUT : out  STD_LOGIC);
end memoire ;

architecture code of memoire  is
    component bascul
        Port ( clk, d : in  STD_LOGIC;
               q : out  STD_LOGIC);
    end component;

    signal A, D  : STD_LOGIC;
begin
PROCESS (W)
BEGIN

    A = (W and DIN) or ((not W) and D)  ;

END PROCESS;
    U1 bascul port map(clk = CLOCK, d = A, q = D);
process(R)
BEGIN
```

```
        DOUT = D AND R  ;

END PROCESS;
end code ;
```

- Code source de mux_1bit

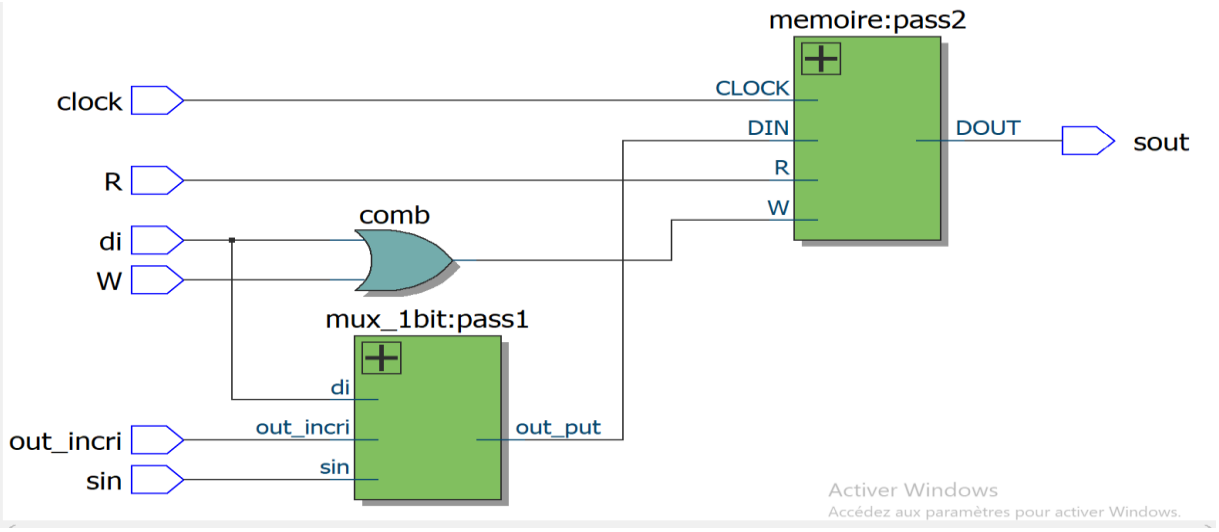
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_1bit is
port (sin ,out_incri,di : in std_logic ;
      out_put           : out std_logic );
end mux_1bit;
architecture code of mux_1bit is
begin
process(sin,out_incri,di)
begin
if di = '1' then
out_put <= sin ;
else
out_put <= out_incri ;
end if;
end process;
end code ;
```

- Code source de RAM_1bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;
entity RAM_1bit is
   port(    di     : in std_logic;
            clock  : in std_logic ;
            R      : in std_logic ;
            W      : in std_logic ;
             sin   : in std_logic ;
        out_incri: in std_logic;
            sout   : out std_logic
   );
end RAM_1bit;
architecture code of RAM_1bit is
signal info : std_logic;
component mux_1bit is
port (sin ,out_incri,di : in std_logic ;
     out_put: out std_logic );
end component;
component memoire is
Port ( DIN, CLOCK, W, R : in  STD_LOGIC;
       DOUT : out  STD_LOGIC);
end component ;
begin
pass1 : mux_1bit port map (
       sin => sin ,
       out_incri => out_incri,
```

```
                di => di,
                out_put => info
);
pass2 : memoire port map (
                DIN     =>  info,
                clock   =>  clock,
                W       =>   W or di,
                R       =>   R,
                DOUT    =>   sout
);
end code;
```

- Shema electrique du RAM_1bit

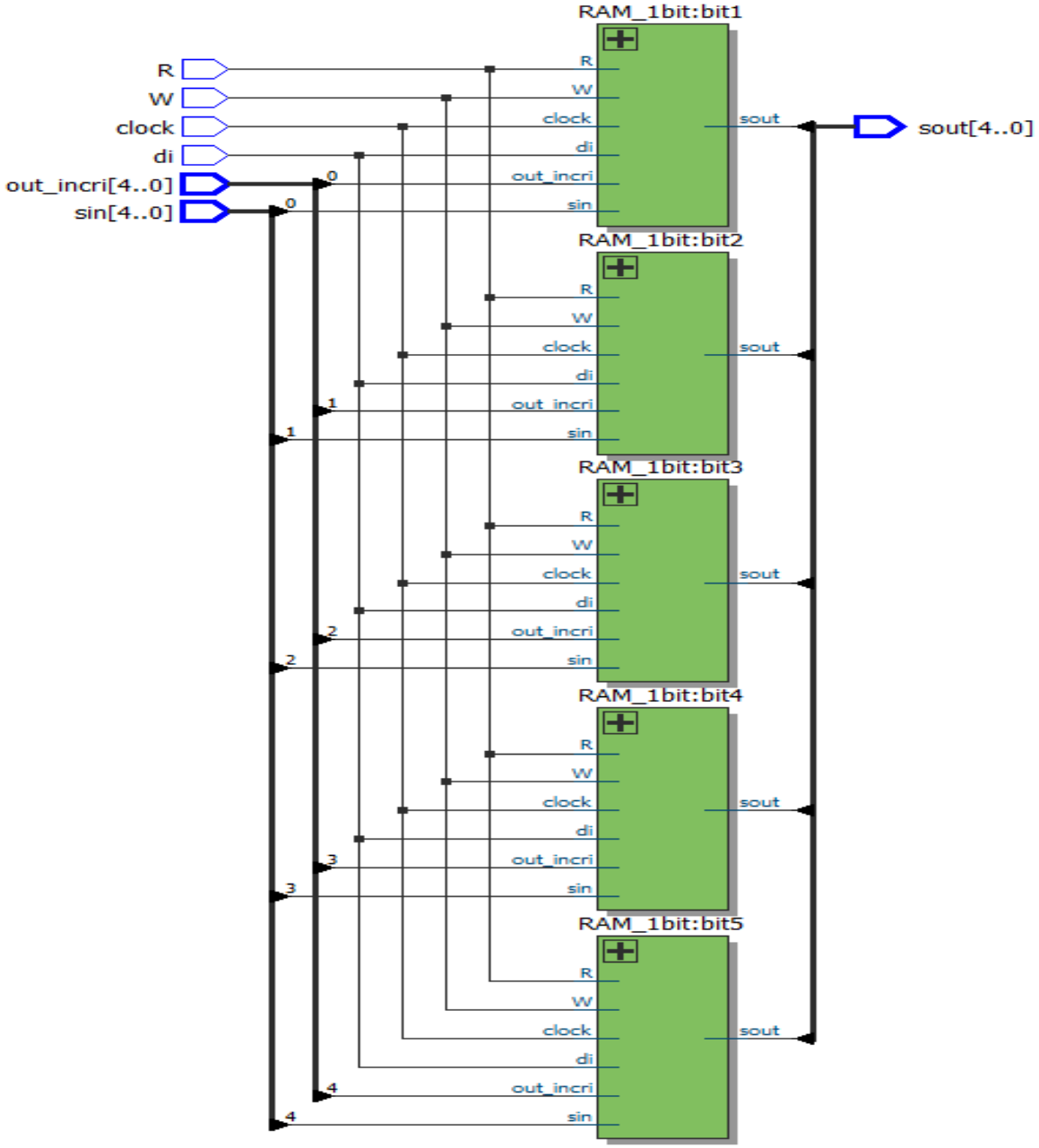


- Code source de RAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;
entity RAM is
   port(  di        : in std_logic;
            clock : in std_logic ;
            R     : in std_logic ;
           W     : in std_logic ;
          sin    : in  std_logic_vector(4 downto 0) ;
       out_incri: in  std_logic_vector(4 downto 0) ;
        sout  : out   std_logic_vector(4 downto 0)
   );
end RAM;
architecture code of RAM is
component RAM_1bit is
  port(  di   : in std_logic;
            clock : in std_logic ;
            R     : in std_logic ;
            W     : in std_logic ;
            sin   : in std_logic ;
            out_incri: in std_logic;
            sout  : out std_logic
   );
end component;
```

```
begin
bit1 : RAM_1bit port map (
      di => di ,
                clock => clock ,
                R     => R ,
                W     => W,
                sin => sin(0),
                out_incri => out_incri(0),
                sout => sout(0)
);
bit2 : RAM_1bit port map (
      di => di ,
                clock => clock ,
                R     => R,
                W     => W,
                sin => sin(1),
                out_incri => out_incri(1),
                sout => sout(1)
);
bit3 : RAM_1bit port map (
      di => di ,
                clock => clock ,
                R     => R,
                W     => W,
                sin => sin(2),
                out_incri => out_incri(2),
                sout => sout(2)
);
bit4 : RAM_1bit port map (
               di => di ,
                clock => clock ,
                R     => R,
                W     => W,
                sin => sin(3),
                out_incri => out_incri(3),
                sout => sout(3)
);
```

- Shema electrique du RAM

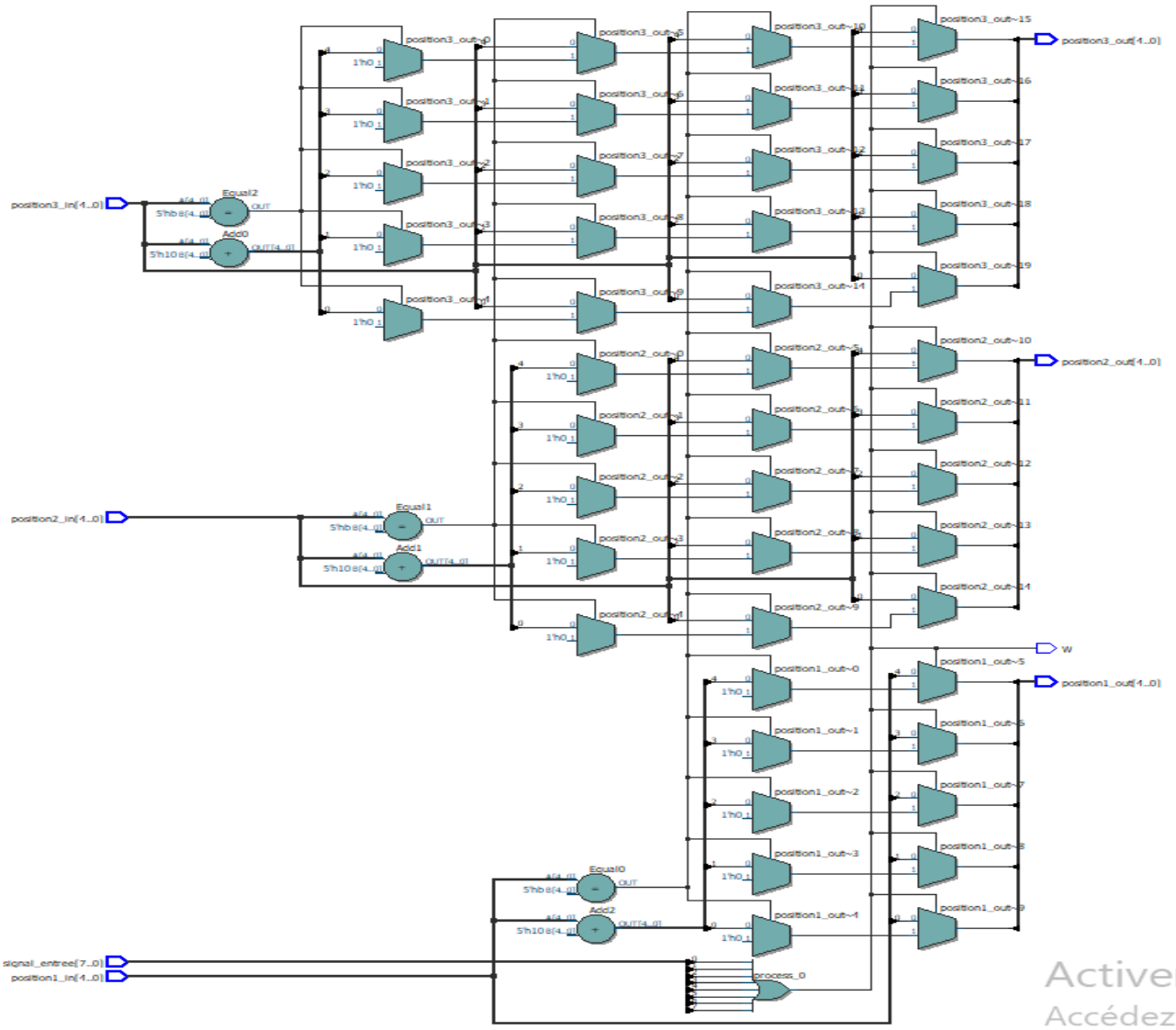


- Code source d'incri

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;
entity incri is
    port(
        signal_entree   : in   STD_LOGIC_VECTOR(7 downto 0);
        position1_in    : in   INTEGER range 0 to 31 ;
        position2_in    : in   INTEGER range 0 to 31 ;
        position3_in    : in   INTEGER range 0 to 31 ;
        position1_out   : out  INTEGER range 0 to 31 ;
        position2_out   : out  INTEGER range 0 to 31 ;
        position3_out   : out  INTEGER range 0 to 31 ;
                  W     : out  std_logic
    );
end incri;
architecture code of incri is
    signal test_bit: std_logic ;
begin
    process(position1_in, position2_in, position3_in, signal_entree)
    begin
                  position1_out <= position1_in;
                  position2_out <= position2_in;
                  position3_out <= position3_in;
        test_bit <= '0';
              for i in signal_entree'range loop
           if signal_entree(i) = '1' then
              test_bit <= '1';
              exit;
            else
              test_bit <= '0';
           end if;
        end loop;
        if test_bit='1'  then
           if position1_in = 26 then
              position1_out <=  0 ;
              if position2_in = 26 then
                 position2_out <=  0 ;
                 if position3_in= 26 then
                    position3_out <=  0 ;
                 else
                position3_out <=  position3_in + 1 ;
                 end if;
              else
                position2_out <=  position2_in + 1 ;
              end if;
           else
                 position1_out <=  position1_in + 1 ;
           end if;
                        W <= '1';
                  else
                        W <= '0';
        end if;
```

```
end process;
end code;
```

- Shema electrique d'incri



- Code source de test

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity test is
port(
test_bit : out  STD_LOGIC ;
signal_entree: in std_logic_vector(7 downto 0 )
);
end test;
architecture code of test is
signal signal_entree_int : integer ;
begin
process (signal_entree)
begin
signal_entree_int <= to_integer(unsigned(signal_entree));
test_bit <= '0';
if  signal_entree_int >= 1 then
     test_bit <= '1';
        else
     test_bit <= '0';
  end if;
end process ;
end code;
```

- On regroupe ces composants dans le code suivant :

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

entity incri01 is
    Port (
            di : in std_logic ;
                clock : in std_logic;
       input_vector : in STD_LOGIC_VECTOR(7 downto 0);
       input_int_1  : in     INTEGER range 0 to 31;
       input_int_2  : in     INTEGER range 0 to 31;
       input_int_3  : in     INTEGER range 0 to 31;
       output_int_1 : buffer INTEGER range 0 to 31;
       output_int_2 : buffer INTEGER range 0 to 31;
       output_int_3 : buffer INTEGER range 0 to 31
    );
end incri01;

architecture code of incri01 is
signal signal_write : std_logic;
signal signal_read  : std_logic;

signal out_RAM_1_std : std_logic_vector(4 downto 0);
signal out_RAM_2_std : std_logic_vector(4 downto 0);
signal out_RAM_3_std : std_logic_vector(4 downto 0);
```

```
signal in_RAM_1_std  : std_logic_vector(4 downto 0);
signal in_RAM_2_std  : std_logic_vector(4 downto 0);
signal in_RAM_3_std  : std_logic_vector(4 downto 0);

signal in_RAM2_1_std  : std_logic_vector(4 downto 0);
signal in_RAM2_2_std  : std_logic_vector(4 downto 0);
signal in_RAM2_3_std  : std_logic_vector(4 downto 0);

signal out_RAM2_1    : std_logic_vector(4 downto 0);
signal out_RAM2_2    : std_logic_vector(4 downto 0);
signal out_RAM2_3    : std_logic_vector(4 downto 0);

signal out_RAM_1_int : INTEGER range 0 to 31;
signal out_RAM_2_int : INTEGER range 0 to 31;
signal out_RAM_3_int : INTEGER range 0 to 31;

signal in_RAM_1_int  : INTEGER range 0 to 31;
signal in_RAM_2_int  : INTEGER range 0 to 31;
signal in_RAM_3_int  : INTEGER range 0 to 31;
component incri is
port(
      signal_entree   : in   STD_LOGIC_VECTOR(7 downto 0);
      position1_in    : in   INTEGER range 0 to 31;
      position2_in    : in   INTEGER range 0 to 31;
      position3_in    : in   INTEGER range 0 to 31;
      position1_out   : out  INTEGER range 0 to 31;
      position2_out   : out  INTEGER range 0 to 31;
      position3_out   : out  INTEGER range 0 to 31;
               W              : out  STD_LOGIC
   );
end component;
component RAM is
port(    di      : in    std_logic ;
            clock    : in    std_logic ;
            R       : in    std_logic ;
                     W       : in    std_logic ;
                     sin     : in    std_logic_vector(4 downto 0) ;
                     out_incri: in    std_logic_vector(4 downto 0) ;
                     sout    : out   std_logic_vector(4 downto 0)
   );
end component;
component test is
port(
test_bit : out  STD_LOGIC ;
signal_entree: in std_logic_vector(7 downto 0 )
);
end component;
begin
test1 : test port map (
      test_bit     => signal_read,
     signal_entree => input_vector

);
```

```
out_RAM_1_int <= to_integer( unsigned(out_RAM_1_std));
                out_RAM_2_int <= to_integer(unsigned(out_RAM_2_std));
                out_RAM_3_int <= to_integer(unsigned(out_RAM_3_std));
                in_RAM2_1_std  <= std_logic_vector(to_unsigned(in_RAM_1_int,5));
                in_RAM2_2_std  <= std_logic_vector(to_unsigned(in_RAM_2_int,5));
                in_RAM2_3_std  <= std_logic_vector(to_unsigned(in_RAM_3_int,5));
                output_int_1  <= in_RAM_1_int;
                output_int_2  <= in_RAM_2_int;
                output_int_3  <= in_RAM_3_int;

op1: incri port map (
                signal_entree => input_vector,
                position1_in  => out_RAM_1_int,
                position2_in  => out_RAM_2_int,
                position3_in  => out_RAM_3_int,
                position1_out => in_RAM_1_int,
                position2_out => in_RAM_2_int,
                position3_out => in_RAM_3_int,
                W             =>  signal_write
);
RAM_11 : RAM PORT MAP (
      di          => di ,
                clock        => clock,
                R            => signal_read,
                W            => not signal_read,
                sin          => std_logic_vector(to_unsigned(input_int_1,5)),
                out_incri     => out_RAM2_1,
                sout         => out_RAM_1_std
);
RAM_12 : RAM PORT MAP (
      di          => di ,
                clock        => clock,
                R            => signal_read,
                W            => not signal_read,
                sin          => std_logic_vector(to_unsigned(input_int_2,5)),
                out_incri     => out_RAM2_2,
                sout         => out_RAM_2_std
);
RAM_13 : RAM PORT MAP (
      di          => di ,
                clock        => clock,
                R            => signal_read,
                W            => not signal_read,
                sin          => std_logic_vector(to_unsigned(input_int_3,5)),
                out_incri     => out_RAM2_3,
                sout         => out_RAM_3_std
);
```

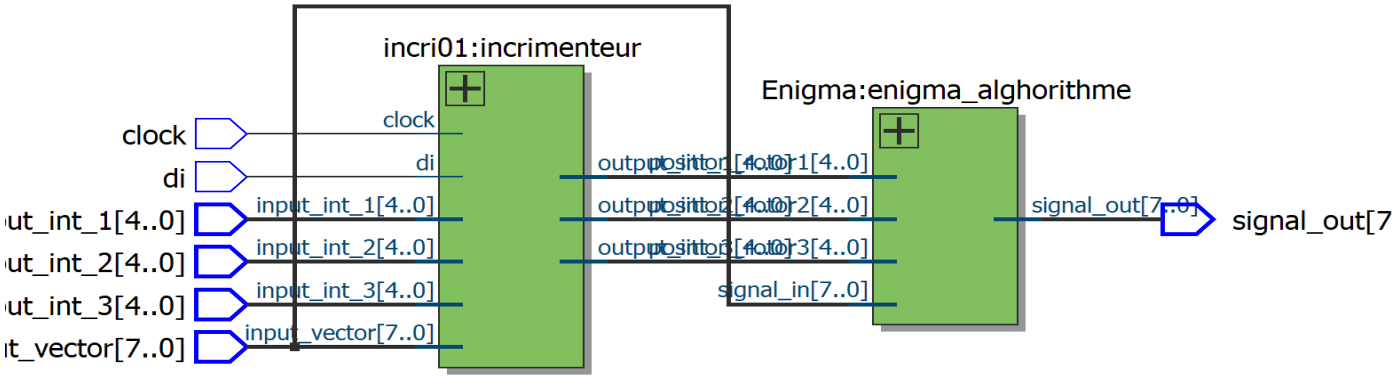
```
RAM_21 : RAM PORT MAP (
      sin         => std_logic_vector(to_unsigned(input_int_1,5)),
      di          => di  ,
                clock        => clock,
                R            => not signal_read,
                W            => signal_write,
                out_incri    => in_RAM2_1_std,
                sout         => out_RAM2_1
);
RAM_22 : RAM PORT MAP (
      sin         => std_logic_vector(to_unsigned(input_int_2,5)),
      di          => di  ,
                clock        => clock,
                R            => not signal_read,
                W            => signal_write,
                out_incri    => in_RAM2_2_std,
                sout         => out_RAM2_2
);
RAM_23 : RAM PORT MAP (
      sin         => std_logic_vector(to_unsigned(input_int_3,5)),
      di          => di  ,
                clock        => clock,
                R            => not signal_read,
                W            => signal_write,
                out_incri    => in_RAM2_3_std,
                sout         => out_RAM2_3
);


end code;
```

- **On regroupe la partie incri01 et la partie Enigma dans le code suivant :**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity electronic_enigma is
port(
    di : in std_logic ;
        clock : in std_logic;
    input_vector : in STD_LOGIC_VECTOR(7 downto 0);
    input_int_1  : in    INTEGER range 0 to 31;
    input_int_2  : in    INTEGER range 0 to 31;
    input_int_3  : in    INTEGER range 0 to 31;
        signal_out : out STD_LOGIC_VECTOR(7 downto 0)
);
end electronic_enigma;
architecture code of electronic_enigma is
signal  interne1 : integer range 0 to 31;
signal  interne2 : integer range 0 to 31;
signal  interne3 : integer range 0 to 31;
```

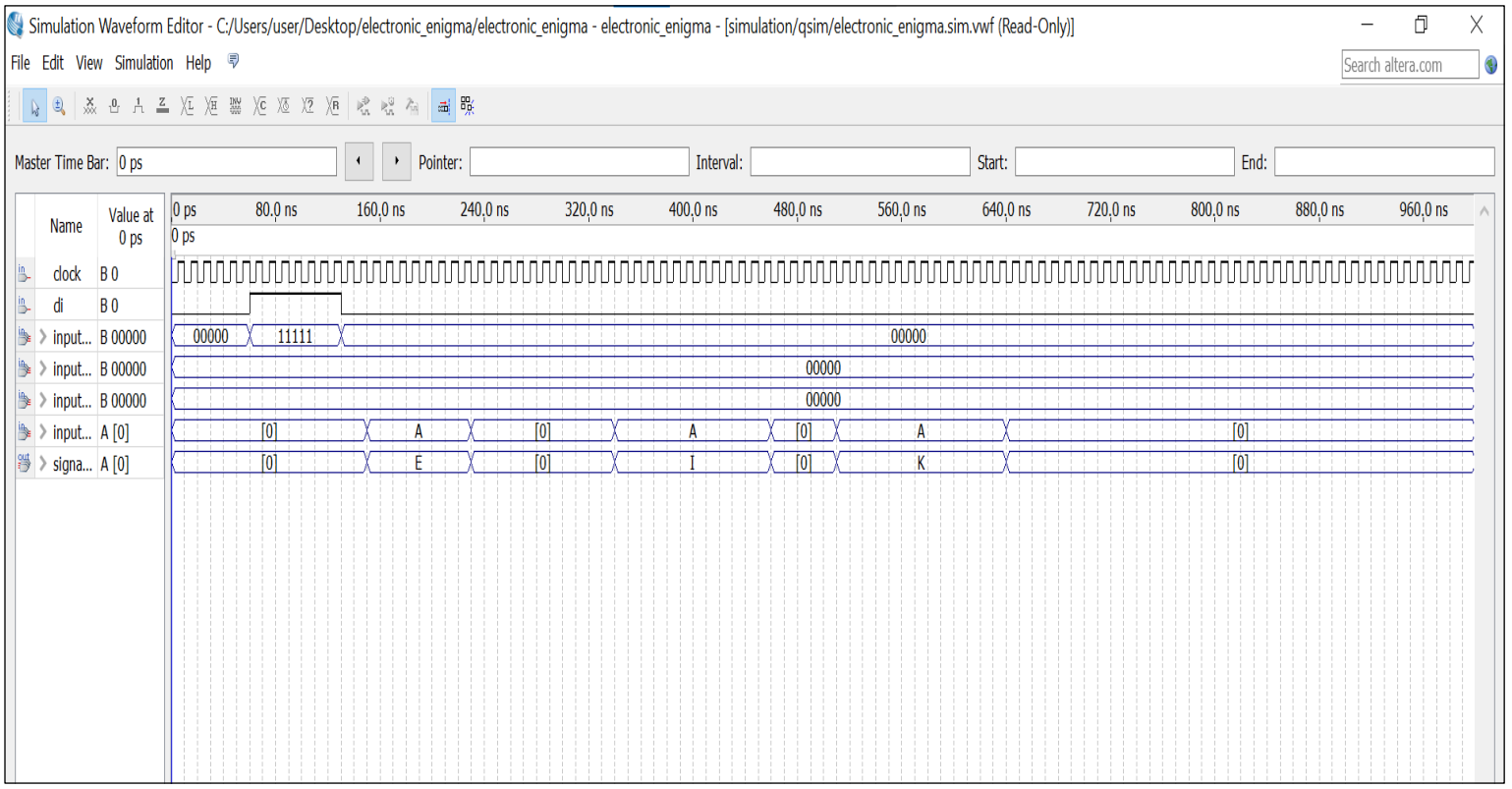
```
component incri01 is
Port (
            di : in std_logic ;
                 clock : in std_logic;
       input_vector : in STD_LOGIC_VECTOR(7 downto 0);
       input_int_1  : in     INTEGER range 0 to 31;
       input_int_2  : in     INTEGER range 0 to 31;
       input_int_3  : in     INTEGER range 0 to 31;
       output_int_1 : buffer INTEGER range 0 to 31;
       output_int_2 : buffer INTEGER range 0 to 31;
       output_int_3 : buffer INTEGER range 0 to 31
   );
end component;
component Enigma is
port (
       signal_in  : in  STD_LOGIC_VECTOR(7 downto 0);
       signal_out : out STD_LOGIC_VECTOR(7 downto 0);
       position_rotor1      : IN INTEGER range 0 to 31 ;
       position_rotor2      : IN INTEGER range 0 to 31;
       position_rotor3      : IN INTEGER range 0 to 31
   );
end component ;
begin
incrimenteur : incri01 port map (
       di   =>  di,
                 clock => clock,
                 input_vector=>input_vector,
                 input_int_1=> input_int_1,
                 input_int_2=> input_int_2,
                 input_int_3=> input_int_3,
                 output_int_1=> interne1,
                 output_int_2=> interne2,
                 output_int_3=> interne3
);
enigma_algorithme : enigma port map (
       signal_in  => input_vector,
                 signal_out => signal_out,
                 position_rotor1=>interne1,
                 position_rotor2=>interne2,
                 position_rotor3=>interne3
);
end code ;
```

- Shema electrique du electronic_enigma



- **Capture de simulation**

CAS de cryptage :



CAS de decryptage :