ENET'COM
المدرسة الوطنية للإلكترونيك و الإتصالات بصفاقس
ECOLE NATIONALE D'ELECTRONIQUE ET DES TÉLÉCOMMUNICATIONS DE SFAX

DÉCEMBRE 2022

PROJET MICROCONTROLEURS ET INTERFACES :

Réalisation d'une Serrure Electronique

Préparé par :

- **Kallebi Cheher** *2ÈME ANNÉE GII 2*
- **Cherif Cyrine** *2ÈME ANNÉE GII 2*

Encadreé par :

- **M. Kharrat Wajdi**

Réalisation d'une Serrure Electronique

I. Introduction

Nous souhaitons réaliser une serrure électronique. La clef est fixée par un code à 4 chiffres que doit taper l'utilisateur. Si le code utilisateur est correct alors on déclenche l'ouverture de la serrure. Ce type de serrure est utilisé par exemple dans les hôtels pour de petits coffres forts permettant aux clients de laisser des objets précieux dans leur chambre. La serrure électronique est connectée à un clavier 16 touches. Un bus 16 bits ***BusExt*** relie le clavier à la serrure électronique. Chaque touche est connectée à la partie électronique de la serrure grâce à un fils du bus ***BusExt***. La connexion est montrée sur la figure 1.

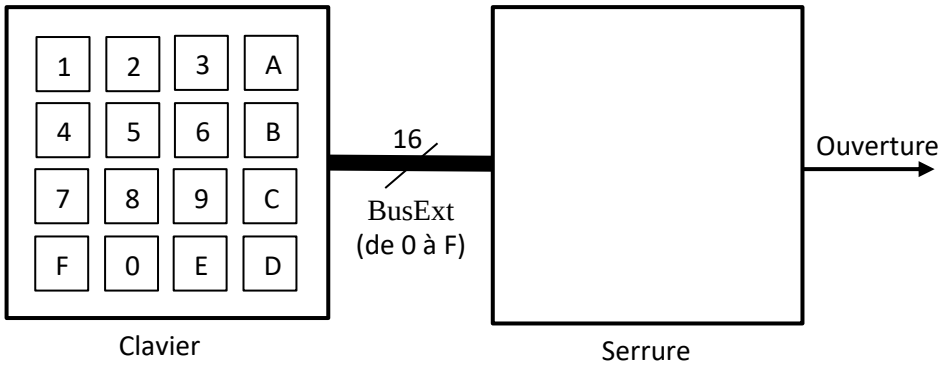


Figure 1 : Interface clavier-serrure

Tant qu'aucune touche n'est enfoncée tous les fils du bus sont au niveau haut. L'appui sur une des touches met le fil qui la relie au codeur au niveau bas tandis que les autres fils restent au niveau haut. La table suivante montre l'état de *BusExt* lors de l'appui sur les différentes touches du clavier. Vous remarquerez la position du bit au niveau bas en fonction de la position de la touche enfoncée sur le clavier.

Touche du clavier enfoncée	**Etat du bus BusExt**
0	1101 1111 1111 1111
1	1111 1111 1111 1110
2	1111 1111 1111 1101
3	1111 1111 1111 1011
4	1111 1111 1110 1111
5	1111 1111 1101 1111
6	1111 1111 1011 1111
7	1111 1110 1111 1111
8	1111 1101 1111 1111
9	1111 1011 1111 1111
A	1111 1111 1111 0111
B	1111 1111 0111 1111
C	1111 0111 1111 1111
D	0111 1111 1111 1111
E	1011 1111 1111 1111
F	1110 1111 1111 1111

Table 1- Table des états de BusExt en fonction de la touche enfoncée

II. Partie 1 : Implémentation de la partie électronique d'une serrure en technologie FPGA

L'architecture globale du système est présentée par la figure 2.

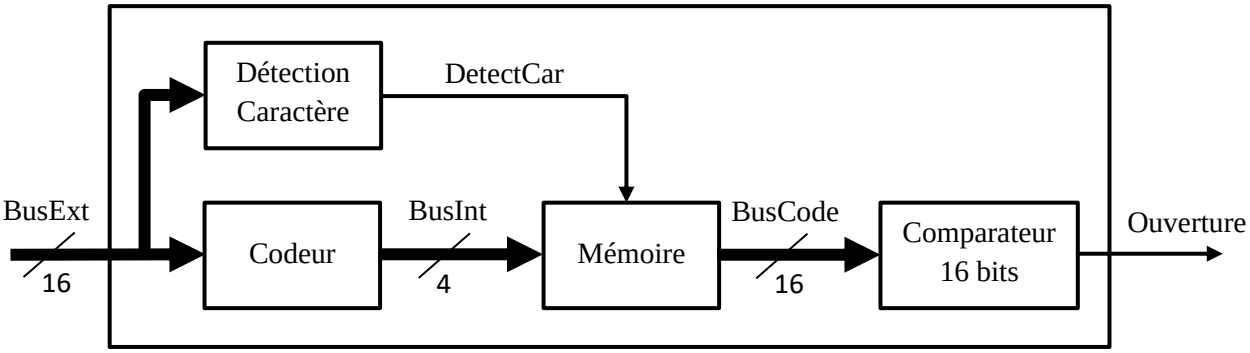


Figure 2 : Architecture globale du système

Dans cette partie, vous allez créer chaque bloc constituant la serrure. Vous allez réaliser leur description structurelle en VHDL et validerez, à chaque fois, leur fonctionnement en effectuant une simulation.

1- Codeur

Un codeur traduisant le code émis par le clavier (cf. table 1) en code binaire qui sera utilisé dans la serrure. Le bus ***BusExt*** provient du clavier et est en entrée du codeur. Le résultat du codage sur 4 bits sera mis sur le bus ***BusInt.*** Le codage retenu est indiqué dans la Table 2.

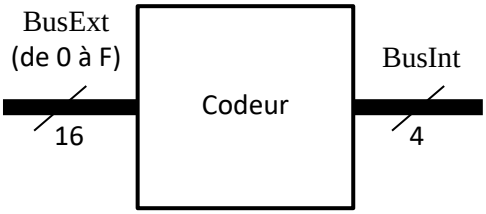


Figure 3 - Codeur

	Etat du bus BusExt	BusInt	Detect
	1111 1111 1111 1111	Z	0
0	1101 1111 1111 1111	0000	1
1	1111 1111 1111 1110	0001	1
2	1111 1111 1111 1101	0010	1
3	1111 1111 1111 1011	0011	1
4	1111 1111 1110 1111	0100	1
5	1111 1111 1101 1111	0101	1
6	1111 1111 1011 1111	0110	1
7	1111 1110 1111 1111	0111	1
8	1111 1101 1111 1111	1000	1
9	1111 1011 1111 1111	1001	1
A	1111 1111 1111 0111	1010	1
B	1111 1111 0111 1111	1011	1
C	1111 0111 1111 1111	1100	1
D	0111 1111 1111 1111	1101	1
E	1011 1111 1111 1111	1110	1
F	1110 1111 1111 1111	1111	1

Table 2 - Codeur

Description VHDL du Codeur :

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

entity Codeur is
    Port (
        BusExt : in  STD_LOGIC_VECTOR (15 downto 0);
        BusInt : out  STD_LOGIC_VECTOR (3 downto 0)
    );
end Codeur;

architecture Behavioral of Codeur is
begin
    process(BusExt)
    begin
        case BusExt is
              when "1101111111111111" => BusInt <= "0000";
              when "1111111111111110" => BusInt <= "0001";
              when "1111111111111101" => BusInt <= "0010";
              when "1111111111111011" => BusInt <= "0011";
              when "1111111111101111" => BusInt <= "0100";
              when "1111111111011111" => BusInt <= "0101";
              when "1111111110111111" => BusInt <= "0110";
              when "1111111011111111" => BusInt <= "0111";
              when "1111110111111111" => BusInt <= "1000";
              when "1111101111111111" => BusInt <= "1001";
              when "1111111111110111" => BusInt <= "1010";
              when "1111111101111111" => BusInt <= "1011";
              when "1111011111111111" => BusInt <= "1100";
              when "0111111111111111" => BusInt <= "1101";
              when "1011111111111111" => BusInt <= "1110";
              when "1110111111111111" => BusInt <= "1111";
              when "1111111111111111" => BusInt <= "Z";
              when others => BusInt <= (others => '0');
        end case;
    end process;
end Behavioral;

```

Ce module possède un port d'entrée « BusExt » de type STD_LOGIC_VECTOR à 16 bits, et un port de sortie « BusInt » de type STD_LOGIC_VECTOR à 4 bits.

À l'intérieur de l'architecture comportementale, un processus est déclenché par le port d'entrée. Dans le processus, une instruction « case » est utilisée pour mapper l'entrée 16 bits à la sortie 4 bits correspondante selon les règles de conversion données. Si l'entrée ne correspond à aucun des cas spécifiés la sortie est 0.

2- Détection

Un détecteur est un composant qui détecte l'appui sur une touche et positionne le signal « ***DetectCar »*** à 1. Si aucune pression sur une touche n'a été détectée (voir table 2), le signal de sortie est mis à 0.

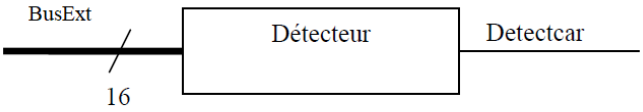


Figure 4- Détecteur

Description VHDL du Détecteur :

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

entity Detecteur is
    Port (
        BusExt : in  STD_LOGIC_VECTOR (15 downto 0);
        DetectCar : out  STD_LOGIC
    );
end Detecteur;

architecture Behavioral of Detecteur is
begin
    process(BusExt)
    begin
        if BusExt = "1111111111111111" then
            DetectCar <= '0';
        else
            DetectCar <= '1';
        end if;
    end process;
end Behavioral;

```

Ce module possède un port d'entrée « BusExt » de type STD_LOGIC_VECTOR à 16 bits, et un port de sortie « Detectar » de type STD_LOGIC.

À l'intérieur de l'architecture comportementale, un processus est déclenché par le port d'entrée. Dans le processus, une instruction if est utilisée pour vérifier si l'entrée est égale à "1111111111111111". Si c'est le cas, la sortie « DetectCar » est mise à 0, sinon « DetectCar » est mis à 1.

3- *Bascule D active sur le front montant*

La bascule D est une bascule comportant uniquement une entrée de données : D. La valeur de l'entrée est recopiée sur la sortie à chaque front d'horloge. Cette bascule permet d'assurer un état de sortie stable entre deux fronts d'horloge, et ainsi d'ignorer toute valeur transitoire apparaissant sur son entrée au cours d'un cycle d'horloge. On ajoute parfois un signal reset afin de pouvoir forcer la valeur initiale à la mise sous tension.

Pour mettre une bascule D active sur front montant et possédé un signal reset *rst* qui remettra à zéro de façon asynchrone la sortie Q on doit réaliser le montage ci-dessous :

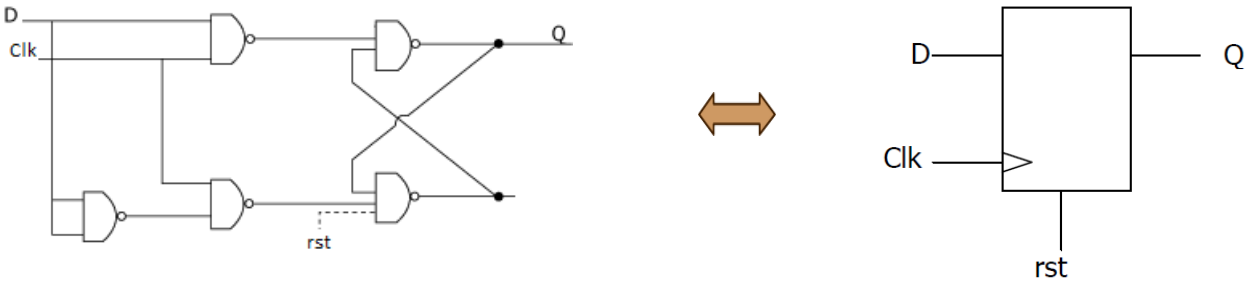


Figure 5 : bascule D active sur niveau

Description VHDL du Bascule D active à front montant avec reset :

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

entity bascule_D is
    port(
        clk : in std_logic;
        reset : in std_logic;
        d : in std_logic;
        q : out std_logic
    );
end entity bascule_D;

architecture behavioral of bascule_D is

    component clock_generator is
        port(
            clk : out std_logic
        );
    end component clock_generator;

begin
    process(clk, reset)
    bas : clock_generator port map (clk);
    begin
        if reset='1' then
            q <= '0';
        elsif rising_edge(clk) then
            q <= d;
        end if;
    end process;
end architecture behavioral;

```

Ce code définit une entité appelée Bascule_D avec quatre ports : clk, reset, d et q. Les entrées clk et reset sont de type std_logic et les entrées d et q sont de type std_logic.

Le comportement de la bascule est implémenté dans l'architecture comportementale à l'aide d'un processus sensible aux entrées clk et reset. Le processus vérifie d'abord la valeur de l'entrée de réinitialisation. Si la réinitialisation est '1', la sortie q est mise à '0'. Si reset vaut '0', le processus attend un front montant sur l'entrée clk, puis règle la sortie q sur la valeur de l'entrée d.

Ce code implémentera une bascule D active sur le front montant de l'horloge et dotée d'une entrée de réinitialisation et d'une entrée D. La sortie Q reflétera la valeur de l'entrée D sur le front montant de l'horloge, sauf si l'entrée de réinitialisation est active, auquel cas la sortie Q sera mise à '0'.

Description VHDL du CLOCK:

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

entity clock_generator is
    port(
        clk : out std_logic
    );
end entity clock_generator;

architecture behavioral of clock_generator is
    constant PERIOD : time := 1 sec;  -- clock period
    constant DUTY_CYCLE : real := 0.5;  -- 50% duty cycle
begin
    process
    begin
        clk <= '0';
        wait for PERIOD * DUTY_CYCLE;
        clk <= '1';
        wait for PERIOD * (1 - DUTY_CYCLE);
    end process;
end architecture behavioral;

```

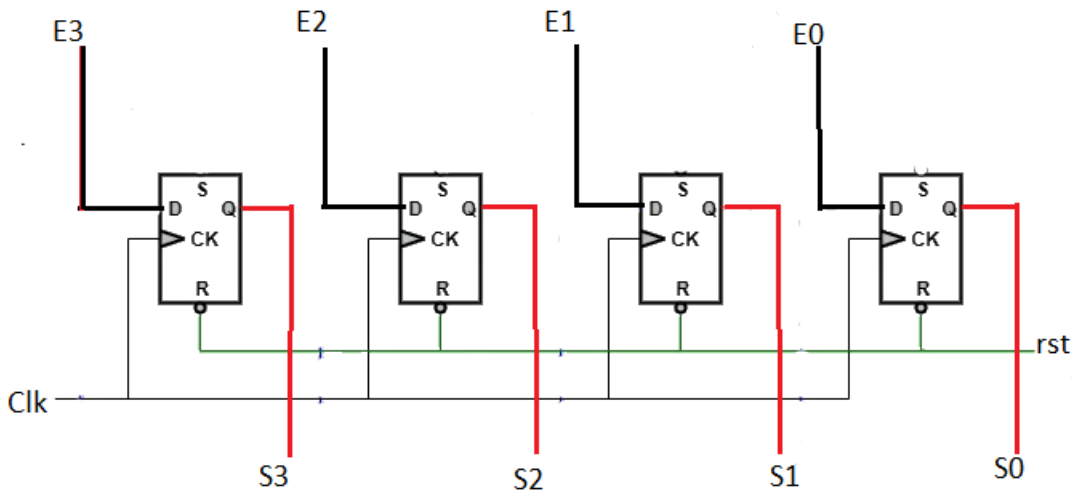
Ce code définit une entité appelée clock_generator avec un seul port de sortie clk de type std_logic. Le comportement du générateur d'horloge est implémenté dans l'architecture comportementale à l'aide d'un processus qui bascule la valeur de la sortie clk à une fréquence de 1 Hz et un rapport cyclique de 50 %.

Pour modifier la fréquence de l'horloge, vous pouvez ajuster la valeur de la constante PERIOD. Par exemple, pour générer une horloge avec une fréquence de 2 Hz, vous pouvez régler PERIOD sur 0,5 sec. Pour modifier le rapport cyclique, vous pouvez ajuster la valeur de la constante DUTY_CYCLE. Par exemple, pour générer une horloge avec un rapport cyclique de 75 %, vous pouvez définir DUTY_CYCLE sur 0,75.

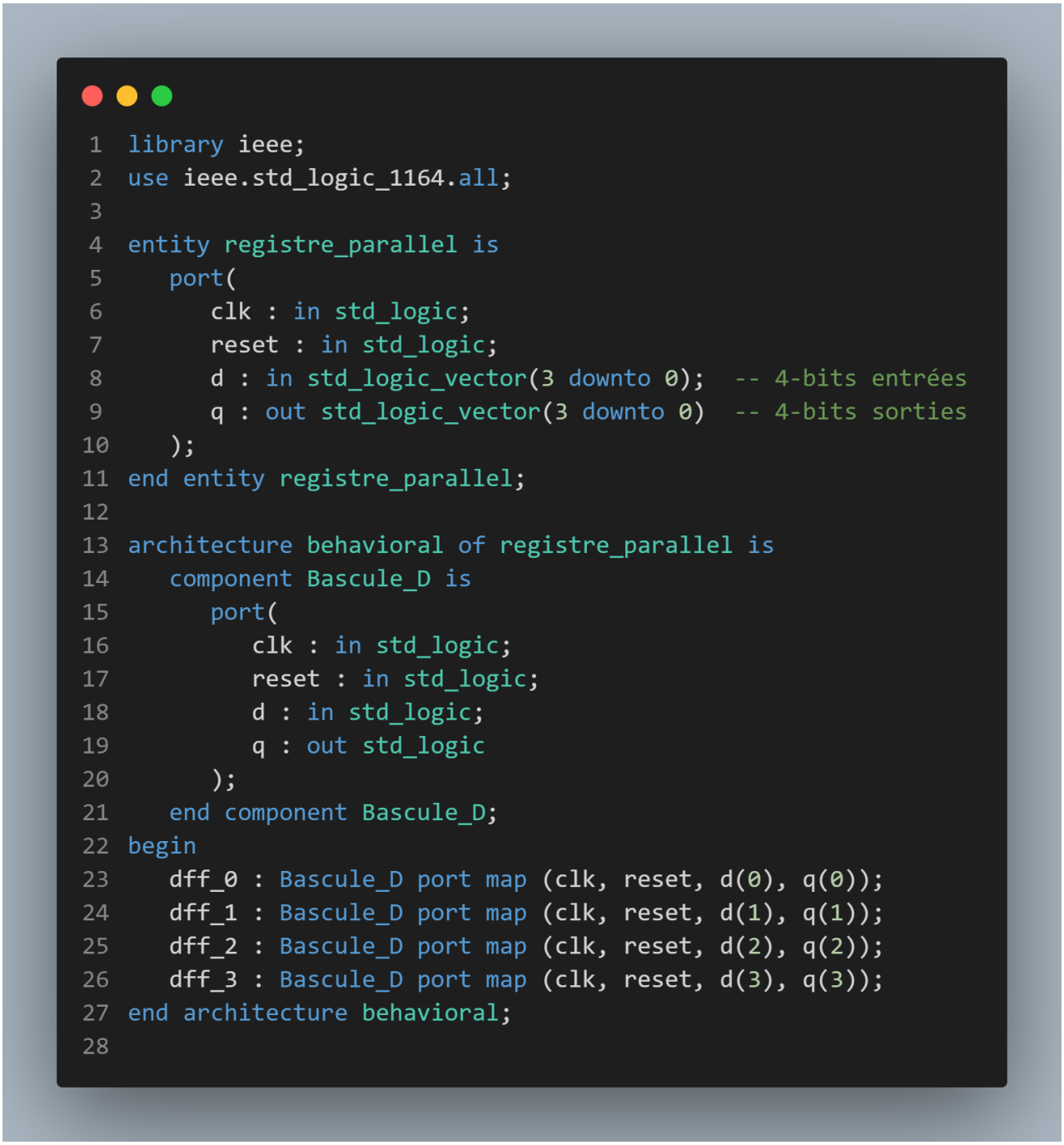
Nous pouvons ensuite utiliser ce signal d'horloge comme entrée clk de la bascule D dans le code précédent

4- Registre à chargement parallèle

Un registre parallèle 4 bits constitué avec des bascules D. Ces bascules possèdent des entrées de forçage au niveau au niveau 0 (R = reset) comme ont étudié dans la section précédente. Ces entrées sont actives quand elles sont au niveau logique 0 et elles ont priorité sur les autres entrées. A chaque front montant de l'horloge (transition du niveau 0 au niveau 1), la valeur du bit présent sur l'entrée D de la bascule est recopié sur sa sortie Q.



Description VHDL d'un registre parallèle avec les bascules D :



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

entity registre_parallel is
   port(
      clk : in std_logic;
      reset : in std_logic;
      d : in std_logic_vector(3 downto 0);  -- 4-bits entrées
      q : out std_logic_vector(3 downto 0)  -- 4-bits sorties
   );
end entity registre_parallel;

architecture behavioral of registre_parallel is
   component Bascule_D is
      port(
         clk : in std_logic;
         reset : in std_logic;
         d : in std_logic;
         q : out std_logic
      );
   end component Bascule_D;
begin
   dff_0 : Bascule_D port map (clk, reset, d(0), q(0));
   dff_1 : Bascule_D port map (clk, reset, d(1), q(1));
   dff_2 : Bascule_D port map (clk, reset, d(2), q(2));
   dff_3 : Bascule_D port map (clk, reset, d(3), q(3));
end architecture behavioral;

```

Ce code définit une entité appelée registre_parllel avec quatre ports d'entrée : clk, reset et d, et quatre ports de sortie : q. Les entrées clk et reset sont de type std_logic, et les entrées d et q sont de type std_logic_vector avec quatre bits (3 à 0).

Le comportement du registre est implémenté dans l'architecture comportementale en utilisant une boucle for pour instancier quatre instances de la bascule D définie dans l'exemple précédent, et en les connectant en série. Les entrées clk et reset sont partagées entre les quatre bascules, et les entrées et sorties d et q sont connectées aux bits correspondants des vecteurs d'entrée et de sortie d et q.

Ce code implémentera un registre parallèle à 4 bits qui décale les valeurs du vecteur d'entrée d dans le vecteur de sortie q sur le front montant de l'horloge, sauf si l'entrée de réinitialisation est active, auquel cas le vecteur de sortie q sera défini sur '0'.

5- Stockage du code

Les caractères saisis par l'utilisateur sont stockés dans une mémoire composée de 4 registres lecture/écriture parallèles 4 bits actifs sur front montant. Quand le signal ***DetectCar*** passe à l'état haut, on capture la valeur du signal présent sur la sortie du codeur ***BusInt*** dans le premier registre et les valeurs précédentes contenues dans les registres sont décalées d'un registre. Le bus ***BusCode*** contient les valeurs des quatre registres. Les 4 premiers bits contiennent la valeur du premier registre (le chiffre de la touche enfoncée en dernier, dernier chiffre du code d'ouverture), les 4 bits suivants du bus contiennent la valeur du second registre et ainsi de suite jusqu'aux 4 bits de poids fort qui contiennent la valeur du 4ème registre (le chiffre de la touche enfoncée en premier, le premier chiffre du code).

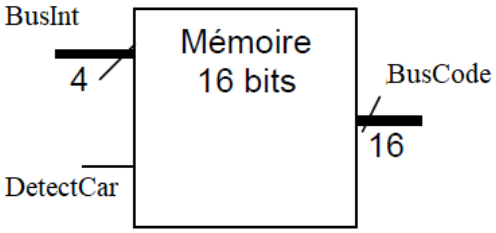


Figure 7 – Stockage

Description VHDL du composant mémoire :

Ce code définit une entité appelée « mémoire » avec quatre ports d'entrée : « clk », « reset », « BusInt » et « DetectCar », et un port de sortie : « BusCode ». Les entrées « clk » et « reset » sont de type std_logic, l'entrée « BusInt » est de type std_logic_vector avec quatre bits (de 3 à 0) et l'entrée « DetectCar » est de type std_logic. La sortie « BusCode » est de type std_logic_vector avec 16 bits (15 jusqu'à 0).

Le comportement du composant « mémoire » est implémenté dans l'architecture comportementale à l'aide de quatre instances du registre parallèle à 4 bits défini dans la section précédente. Les entrées « clk » et « reset » sont partagées entre les quatre registres.

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

entity memoire is
   port(
      clk : in std_logic;
      reset : in std_logic;
      BusInt : in std_logic_vector(3 downto 0);
      DetectCar : in std_logic;
      BusCode : out std_logic_vector(15 downto 0)
   );
end entity memoire;

architecture behavioral of memoire is
   component registre_parallel is
      port(
         clk : in std_logic;
         reset : in std_logic;
         d : in std_logic_vector(3 downto 0);
         q : out std_logic_vector(3 downto 0)
      );
   end component registre_parallel;
begin

   reg_0 : registre_parallel port map
   (clk, reset, BusInt, BusCode(3 downto 0));

   reg_1 : registre_parallel port map
   (clk, reset, BusCode(3 downto 0), BusCode(7 downto 4));

   reg_2 : registre_parallel port map
   (clk, reset, BusCode(7 downto 4), BusCode(11 downto 8));

   reg_3 : registre_parallel port map
   (clk, reset, BusCode(11 downto 8), BusCode(15 downto 12));

   process(clk, reset, DetectCar)
   begin
      if reset='1' then
         BusInt <= (others => '0');
      elsif rising_edge(clk) then
         if DetectCar='1' then
               BusInt <= BusCode(3 downto 0);
               BusCode <= (others => '0');
         end if;
      end if;
   end process;
end architecture behavioral;

```

6- *Comparateur 4 bits*

Un comparateur qui détecte l'égalité de deux valeurs sur 4 bits en entrée. Lorsque les signaux ***Datautil*** et ***Dataamd*** sont égaux alors le signal ***CompOK*** passe à 1, sinon ***CompOK*** est égal à 0.

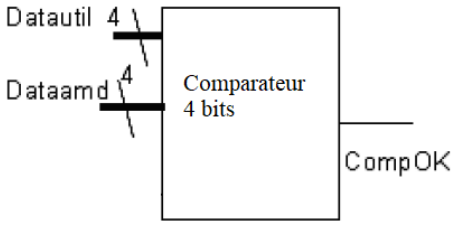


Figure 8 - Comparateur 4 bits

Description VHDL du comparateur :

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

entity comparateur is
   port(
      Datautil : in std_logic_vector(3 downto 0);
      Dataamd : in std_logic_vector(3 downto 0);
      CompOK : out std_logic
   );
end entity comparateur;

architecture behavioral of comparateur is
begin

   process(Datautil, Dataamd)
   begin
      if Datautil = Dataamd then
         CompOK <= '1';
      else
         CompOK <= '0';
      end if;
   end process;
end architecture behavioral;

```

Ce code définit une entité appelée comparateur avec deux ports d'entrée : « Datautil » et « Dataamd », et un port de sortie : « CompOK. » Les entrées « Datautil » et « Dataamd » sont de type std_logic_vector avec quatre bits (de 3 à 0) et la sortie « CompOK » est de type std_logic.

Le comportement du comparateur est implémenté dans l'architecture comportementale à l'aide d'un processus qui compare les entrées « Datautil » et « Dataamd ». Si les entrées sont égales, la sortie « CompOK » est mise à '1'. Si les entrées ne sont pas égales, la sortie « CompOK » est mise à '0'.

7- Comparateur 16 bits

Le comparateur 16 bits est composé de 4 comparateurs 4 bits, vus précédemment. L'entrée ***Datautil*** de chaque comparateur est connectée à 4 bits du bus d'entrée du comparateur 16 bits ***BusCode***. Ce bus est le bus de sortie des registres de la mémoire (cf Figure 2) et contient le code tapé par l'utilisateur. Le bus ***Dataamd*** de chaque comparateur 4 bits contient la valeur d'un des caractères de la clef que vous fixerez à 1234. Lorsque le code entré par l'utilisateur est égal au code de la clef, le signal ***ouverture*** passe à 1, sinon le signal ***ouverture*** reste à 0

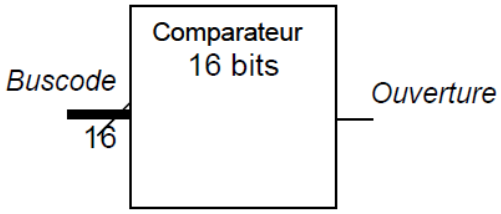


Figure 9 - Comparateur 16 bits

Description VHDL du comparateur à 16 bits :

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

entity comparateur_16 is
   port(
      BusCode : in std_logic_vector(15 downto 0);
      ouverture : out std_logic
   );
end entity comparateur_16;

architecture behavioral of comparateur_16 is
   component comparateur is
      port(
         Datautil : in std_logic_vector(3 downto 0);
         Dataamd : in std_logic_vector(3 downto 0);
         CompOK : out std_logic
      );
   end component comparateur;
begin

   comp_0 : comparateur port map (BusCode(3 downto 0), "0001", ouverture);
   comp_1 : comparateur port map (BusCode(7 downto 4), "0010", ouverture);
   comp_2 : comparateur port map (BusCode(11 downto 8), "0011", ouverture);
   comp_3 : comparateur port map (BusCode(15 downto 12), "0100", ouverture);

end architecture behavioral;

```

8- Assemblage

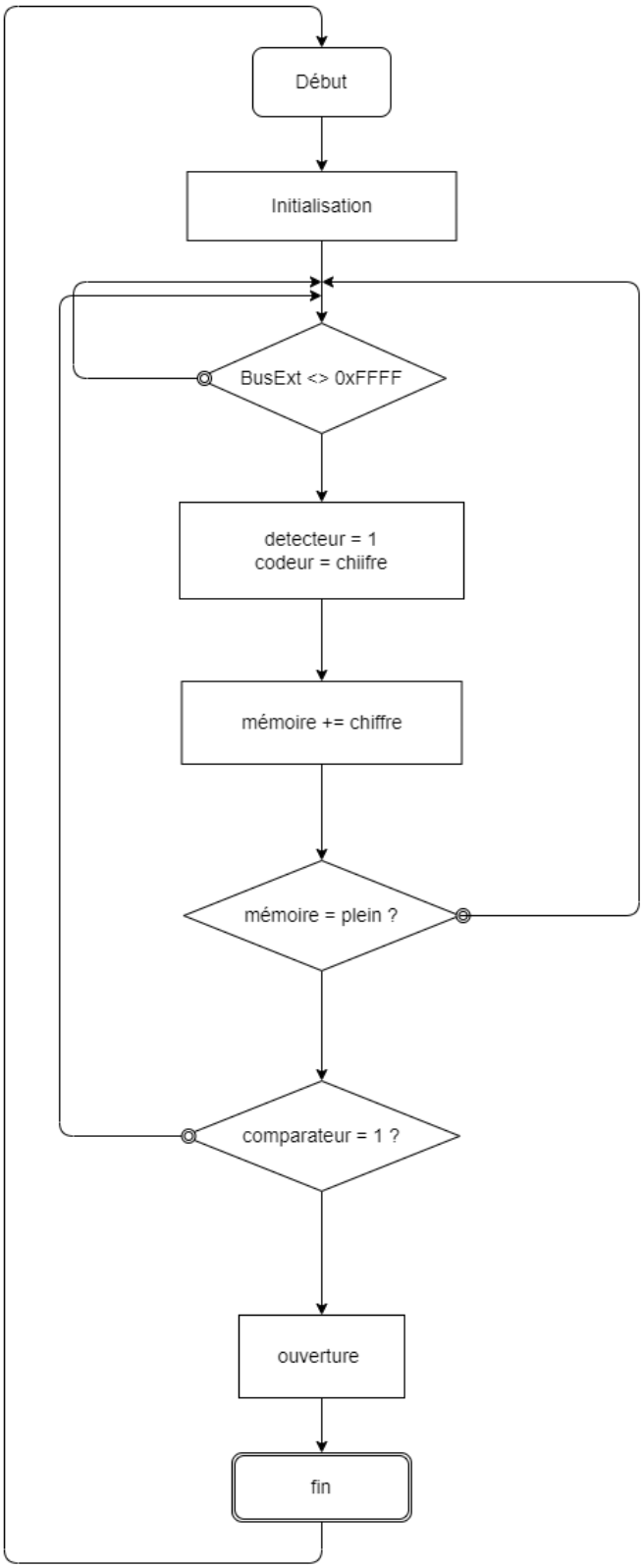
L'assemblage des différents codes précédents assure le fonctionnement de notre serrure électronique, ci-dessous le code de l'application principale en VHDL :

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

entity serrure_electronique is
   port(
      BusExt : in std_logic_vector(15 downto 0);
      ouverture : out std_logic
   );
end entity serrure_electronique;

architecture behavioral of serrure_electronique is
   component Codeur is
      port(
         BusExt : in std_logic_vector(15 downto 0);
         BusInt : out std_logic_vector(3 downto 0)
      );
   end component Codeur;

   component Detecteur is
      port(
         BusExt : in std_logic_vector(15 downto 0);
         DetectCar : out std_logic
      );
   end component Detecteur;

   component memoire is
      port(
         clk : in std_logic;
         reset : in std_logic;
         BusInt : in std_logic_vector(3 downto 0);
         DetectCar : in std_logic;
         BusCode : out std_logic_vector(15 downto 0)
      );
   end component memoire;

   component comparateur_16 is
      port(
         BusCode : in std_logic_vector(15 downto 0);
         ouverture : out std_logic
      );
   end component comparateur_16;

   signal BusInt : out std_logic_vector(3 downto 0);
   signal DetectCar : out std_logic;
   signal BusCode : out std_logic_vector(15 downto 0);

begin

   coder : Codeur port map (BusExt, BusInt);
   detector : Detecteur port map (BusExt, DetectCar);
   memory : memoire port map (clk, reset, BusInt, DetectCar, BusCode);
   compare : comparateur_16 port map (BusCode, ouverture);

end architecture behavioral;

```

III. Partie 2 : Implémentation sur carte STM32 (carte Blue Pill)

Pour l'implémentation du système sur la carte STM32 ,suivre le même principe de fonctionnement cité dans la première partie.

Organigramme :



Ci-joint les codes cpp pour les implémentés dans la carte Blue Pill :

La fonction d’une bascule D en cpp :

```cpp
std::uint8_t bascule_D(std::uint8_t clk, std::uint8_t reset, std::uint8_t D)
{
    std::uint8_t Q = 0;

    static std::uint8_t shiftReg = 0;
    if (clk && !reset)
    {
        shiftReg = D;
    }
    Q = reset ? 0 : shiftReg;

    return Q;
}

```

La fonction du CLOCK en cpp :

```cpp
std::uint8_t bascule_D(std::uint8_t clk, std::uint8_t reset, std::uint8_t D)
{
    std::uint8_t Q = 0;

    static std::uint8_t shiftReg = 0;
    if (clk && !reset)
    {
        shiftReg = D;
    }
    Q = reset ? 0 : shiftReg;

    return Q;
}

```

La fonction du codeur en cpp :

```cpp
#include <array>
#include <string>

std::array<std::uint8_t, 4> Codeur(std::array<std::uint8_t, 16> BusExt)
{
   std::array<std::uint8_t, 4> BusInt{0};

   std::string BusExtStr;
   for (std::uint8_t i : BusExt)
   {
      BusExtStr += std::to_string(i);
   }

   switch (BusExtStr)
   {
      case "1101111111111111":
         BusInt = std::array<std::uint8_t, 4>{0, 0, 0, 0};
         break;
      case "1111111111111110":
         BusInt = std::array<std::uint8_t, 4>{0, 0, 0, 1};
         break;
      case "1111111111111101":
         BusInt = std::array<std::uint8_t, 4>{0, 0, 1, 0};
         break;
      case "1111111111111011":
         BusInt = std::array<std::uint8_t, 4>{0, 0, 1, 1};
         break;
      case "1111111111101111":
         BusInt = std::array<std::uint8_t, 4>{0, 1, 0, 0};
         break;
      case "1111111111011111":
         BusInt = std::array<std::uint8_t, 4>{0, 1, 0, 1};
         break;
      case "1111111110111111":
         BusInt = std::array<std::uint8_t, 4>{0, 1, 1, 0};
         break;
      case "1111111011111111":
         BusInt = std::array<std::uint8_t, 4>{0, 1, 1, 1};
         break;
      case "1111110111111111":
         BusInt = std::array<std::uint8_t, 4>{1, 0, 0, 0};
         break;
      case "1111101111111111":
         BusInt = std::array<std::uint8_t, 4>{1, 0, 0, 1};
         break;
      case "1111111111110111":
         BusInt = std::array<std::uint8_t, 4>{1, 0, 1, 0};
         break;
      case "1111111101111111":
         BusInt = std::array<std::uint8_t, 4>{1, 0, 1, 1};
         break;
      case "1111011111111111":
         BusInt = std::array<std::uint8_t, 4>{1, 1, 0, 0};
         break;
      case "0111111111111111":
         BusInt = std::array<std::uint8_t, 4>{1, 1, 0, 1};
         break;
      case "1011111111111111":
         BusInt = std::array<std::uint8_t, 4>{1, 1, 1, 0};
         break;
      case "1110111111111111":
         BusInt = std::array<std::uint8_t, 4>{1, 1, 1, 1};
         break;
      default: BusInt = std::array<std::uint8_t, 4>{"Z"};
         break;
   }
   return BusInt;
}

```

La fonction du detecteur :

```cpp
#include <array>

std::uint8_t Detecteur(std::array<std::uint8_t, 16> BusExt)
{
    bool allOnes = true;
    for (std::uint8_t i : BusExt)
    {
        if (i != 1)
        {
            allOnes = false;
            break;
        }
    }

    return allOnes ? 1 : 0;
}

```

La fonction du comparateur à 4 bits:

```cpp
bool comparateur(std::array<std::uint8_t, 4> Datautil, std::array<std::uint8_t, 4> Dataamd)
{
    static const std::string DatautilStr{reinterpret_cast<char*>(Datautil.data())};
    static const std::string DataamdStr{reinterpret_cast<char*>(Dataamd.data())};
    return DatautilStr == DataamdStr;
}


```

La fonction du comparateur à 16 bits :

```cpp
bool comparateur_16(std::array<std::uint8_t, 16> BusCode)
{
    std::array<std::array<std::uint8_t, 4>, 4> data;
    std::copy(BusCode.begin(), BusCode.begin() + 4, data[0].begin());
    std::copy(BusCode.begin() + 4, BusCode.begin() + 8, data[1].begin());
    std::copy(BusCode.begin() + 8, BusCode.begin() + 12, data[2].begin());
    std::copy(BusCode.begin() + 12, BusCode.end(), data[3].begin());

    static const std::array<std::array<std::uint8_t, 4>, 4> Dataamd{{0, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 1, 1}, {0, 1, 0, 0}} ;
    return comparateur(data[0], Dataamd) && comparateur(data[1], Dataamd) && comparateur(data[2], Dataamd) && comparateur(data[3], Dataamd);
}
```

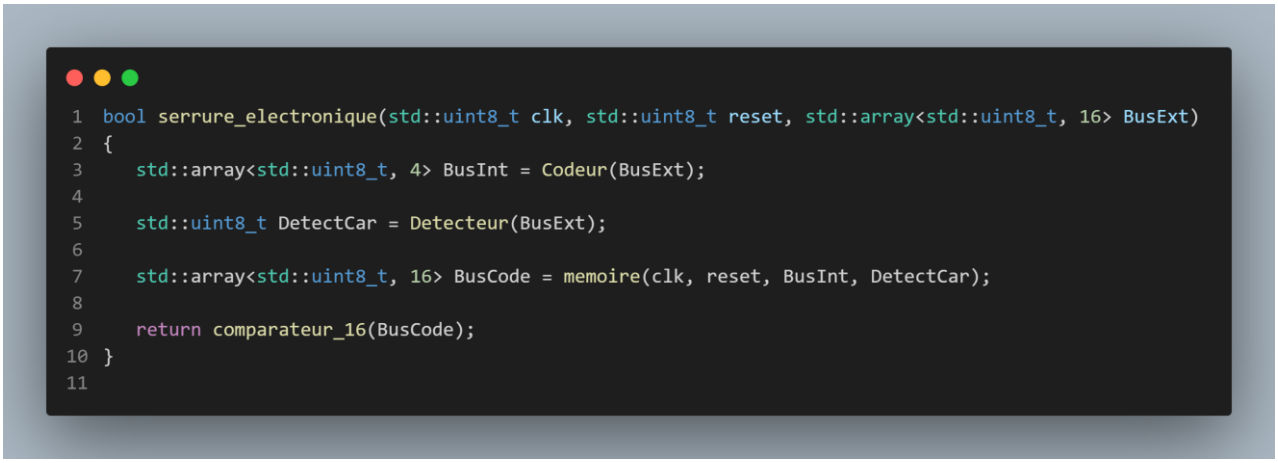
La fonction du registre parallèle :

```cpp
std::array<std::uint8_t, 4> register_parallel(std::uint8_t clk, std::uint8_t reset, std::array<std::uint8_t, 4> Din)
{
    std::array<std::uint8_t, 4> Dout{0};

    static std::array<std::uint8_t, 4> shiftReg{0};
    if (clk && !reset)
    {
        shiftReg = Din;
    }

    if (!clk && !reset)
    {
        Dout = shiftReg;
    }

    return Dout;
}

```

La fonction du mémoire :

```cpp
std::array<std::uint8_t, 16> memoire(std::uint8_t clk, std::uint8_t reset, std::array<std::uint8_t, 4> BusInt, std::uint8_t DetectCar)
{
    std::array<std::uint8_t, 16> BusCode{0};

    static std::array<std::array<std::uint8_t, 4>, 4> shiftReg{{{0}}};
    if (DetectCar)
    {
        shiftReg[3] = shiftReg[2];
        shiftReg[2] = shiftReg[1];
        shiftReg[1] = shiftReg[0];
        shiftReg[0] = BusInt;
    }

    std::copy(shiftReg[3].begin(), shiftReg[3].end(), BusCode.begin());
    std::copy(shiftReg[2].begin(), shiftReg[2].end(), BusCode.begin() + 4);
    std::copy(shiftReg[1].begin(), shiftReg[1].end(), BusCode.begin() + 8);
    std::copy(shiftReg[0].begin(), shiftReg[0].end(), BusCode.begin() + 12);

    return BusCode;
}

```

La fonction principale de la serrure électronique :



```cpp
bool serrure_electronique(std::uint8_t clk, std::uint8_t reset, std::array<std::uint8_t, 16> BusExt)
{
    std::array<std::uint8_t, 4> BusInt = Codeur(BusExt);

    std::uint8_t DetectCar = Detecteur(BusExt);

    std::array<std::uint8_t, 16> BusCode = memoire(clk, reset, BusInt, DetectCar);

    return comparateur_16(BusCode);
}

```

Conclusion

Il est important de signaler que ce mini projet a présenté l'occasion de consolider notre formation par la pratique de nos connaissances en système embarqué.

Ce travail, nous a permis d'avoir une idée générale sur le formalisme ainsi que le système de ventilation d'un local, à travers les étapes suivantes : On commence par la description de miniprojet.

Par la suite, nous avons entamé une étude théorique de projet où nous avons présenté les différents outils de travail. Et finissons par le développement du projet.

Cette démarche nous a permis de bien étudier le besoin et de proposer un programme de à travers le simulateur mbed.

Finalement ce projet présente un profit à l'échelle personnelle puisqu'il nous a permis de consolider nos connaissances en microcontrôleur et interface.