Cours VHDL - IV

L3-S6 - Université de Cergy-Pontoise

Laurent Rodriguez – Benoît Miramond

Plan du cours

I – Historique de conception des circuits intégrés
- **HDL**
  - Modèles de conceptions
- **VHDL**
  - Les modèles de conceptions en VHDL
  - Les 5 briques de base

II – VHDL et FPGA
- **VHDL**
  - **Syntaxe et typage**
  - **Retour sur les briques de bases**
  - **Retour sur la conception structurelle et comportementale en VHDL**
    Port map, Equations logiques, Tables de vérités (With ... Select)
- **FPGA**
  - **Qu'est ce qu'un FPGA**
  - **Flot de conception**
  - **Carte de développement et environnement de TP**

Plan du cours

III – Modélisation

- Compléments sur la description Structurelle

- Description comportementale approfondie

Circuits combinatoires

- Styles de description

Flots de données

Instructions concurrentes

Table de vérité

Fonctions

- Circuits standards

- Comparateur
- Multiplexeurs
- Encodeurs

Plan du cours

IV – Processus et gestion du temps
- **Syntaxe et sémantique**
- **Portée - signaux & variables**
- **Annotations temporelles et délais**

V – Simulation
- **TestBench**
- **GHDL & GTK-Waves**
- **Quartus & ModelSIM**

VI – Modèles génériques
- **Paramètres génériques**
- **« Generate »**
- **Boucles**
- **Exemples**

Plan du cours

IV – Processus et gestion du temps
- **Syntaxe et sémantique**
- **Portée - signaux & variables**
- **Annotations temporelles et délais**

V – Simulation
- **TestBench**
- **GHDL & GTK-Waves**
- **Quartus & ModelSIM**

VI – Modèles génériques
- **Paramètres génériques**
- **« Generate »**
- **Boucles**
- **Exemples**

Plan du cours

VIII – Machines à états finis (FSM)

- Définition
- Représentation
  - Formelle
  - Table de transition
  - Diagramme
- Type de machines
- Exemples
  - Multiplieur à machine à états
- Notions importantes
  - Reset
  - Représentation des états

VIII – Machine à états finis

Finite State Machine (FSM)

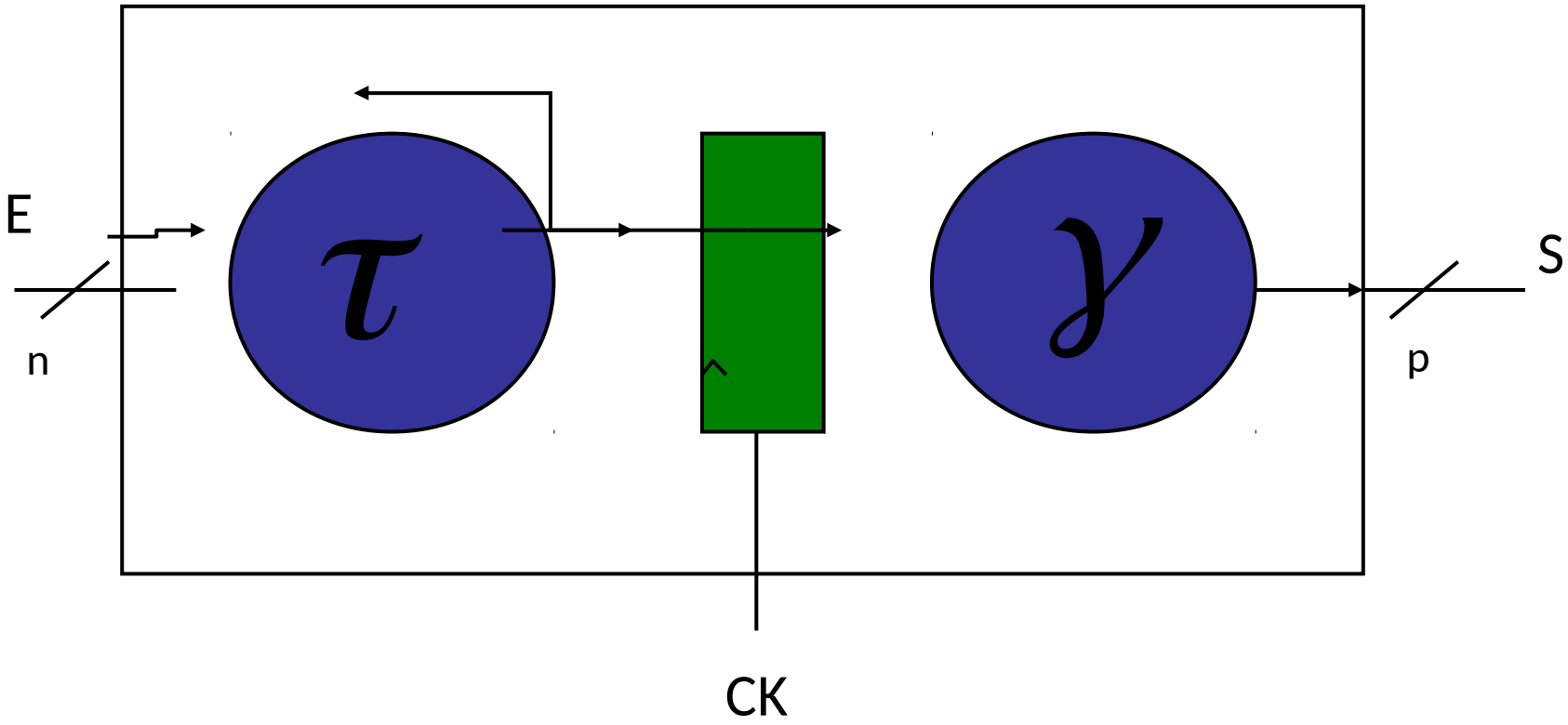
VIII - FSM : Définition

Les Machines à états finis (FSM en anglais) sont utilisées pour décrire des comportements séquentiels liés au contrôle des parties opératives
Cet aspect séquentiel fait intervenir la notion d'état interne implémentés dans les circuits sous forme de registres

Un automate est défini par :

- Un ensemble d'entrées — E = {Ei}
- Un ensemble de sorties — S = {Sj}
- Un ensemble d'états — R = {rk}
- Une fonction de transition — $\tau : R \times E \xrightarrow{\tau} R$
- Une fonction de génération — $\gamma : R \times E \xrightarrow{\gamma} S$

Représentation circuit



Automate de Moore :

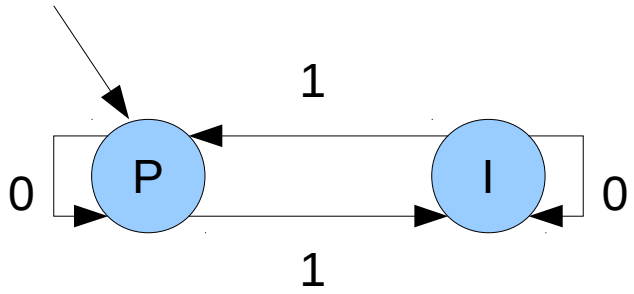
Si la fonction de génération ne dépend que de l'état courant (ne dépend pas des entrées)

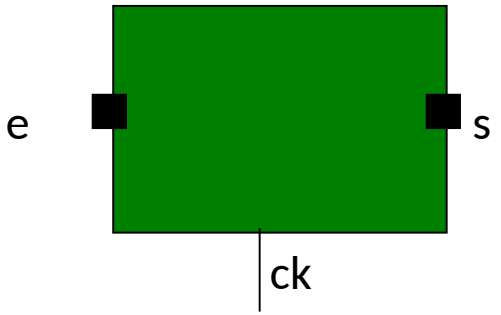
Exemple

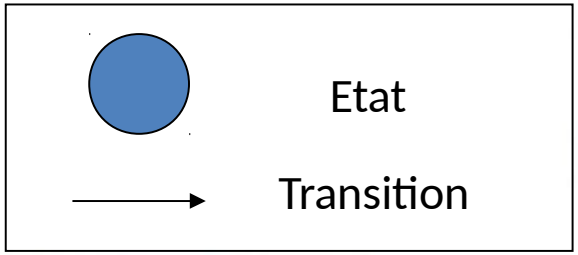
Automate détectant la parité d'une suite binaire

Nombre de 1 pair => 1

Nombre de 1 impair => 0







Propriétés d'un automate

Complet :

Un automate est complet si la somme logique (OR) des expressions booléennes sortant de chaque état vaut 1

$$\sum_{\forall x} \tau_x(E_i) = 1$$

Déterministe :

Si le produit logique des expressions booléennes sortant de chaque état 2 à 2 vaut 0

$$\forall \underset{x \neq y}{x, y} \quad \tau_x(E_i).\tau_y(E_i) = 0$$

2) Représentation d'une FSM

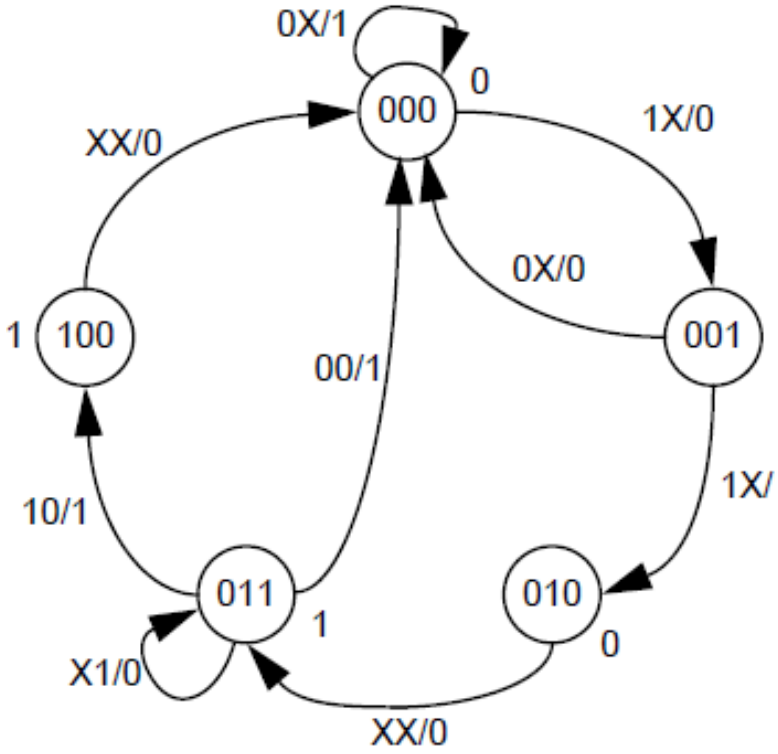
On représente de manière équivalente une FSM

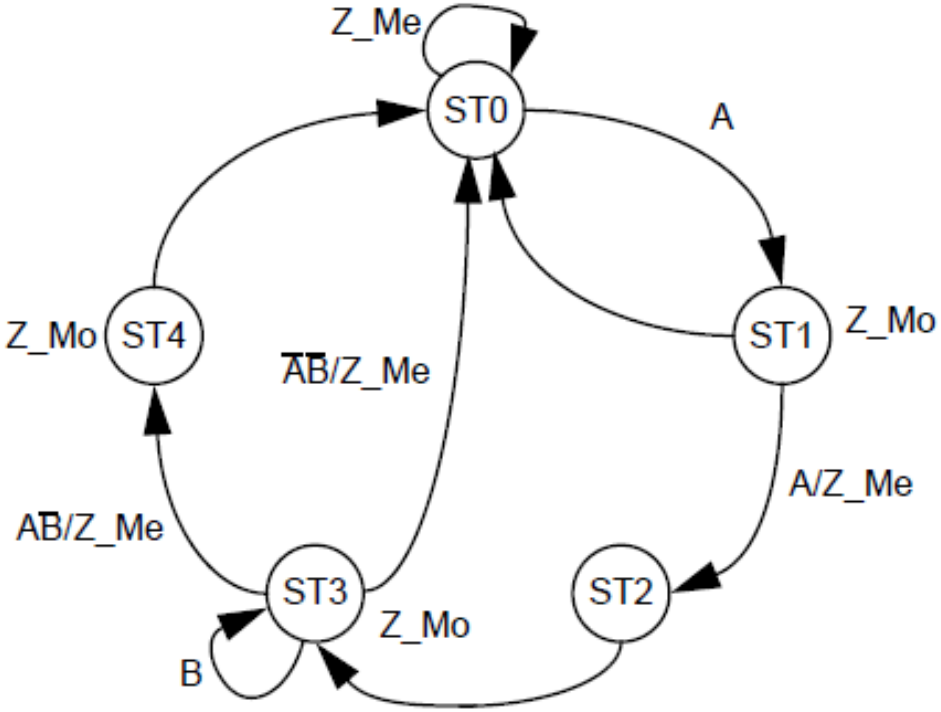
Soit par un diagramme d'états

Soit par une table de vérité correspondant aux fonctions de transitions et de générations

Entrées						Sorties	
A	B	Etat courant		Etat suivant		Z_Me	Z_Mo
0	X	000	(ST0)	000	(ST0)	1	0
1	X	000	(ST0)	001	(ST1)	0	0
0	X	001	(ST1)	000	(ST0)	0	1
1	X	001	(ST1)	010	(ST2)	1	1
X	X	010	(ST2)	011	(ST3)	0	0
X	1	011	(ST3)	011	(ST3)	1	1
0	0	011	(ST3)	000	(ST0)	1	1
1	0	011	(ST3)	100	(ST4)	0	1
X	X	100	(ST4)	000	(ST0)	0	1

Diagrammes correspondants





3) Modélisation des FSM

Utilisation de registres d'états
- Explicite : énumération des états
- Implicite : dépend de la structure du modèle

Introduction d'une méthode d'initialisation
- Reset synchrone
- Ou asynchrone

Encodage des états
- La façon de coder les états à un impact direct sur la complexité de la fonction de transition, donc sur les performances du circuit

Choix Mealy / Moore
- Moore, plus de maîtrise sur le comportement
- Mealy, plus de flexibilité puisque les sorties peuvent changer au sein d'un même état

! Transitions

La transition d'un état à l'autre est synchronisée sur signal d'horloge implicite qui n'est représenté ni dans la tables d'états ni dans les diagrammes d'états

Types de machines

- Machine de Moore : Codage des sortie directement dans les états
- Machine de Mealy : Codage des sorties sur les transitions

Exemple

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FSM is
    port (CLK, RST: in  std_logic;    -- signaux de contrôle
          A, B:     in  std_logic;    -- entrées primaires
          Z_Mo:     out std_logic);   -- sortie de Moore
end FSM;

architecture MooreEnc of FSM is

    constant ST0: std_logic_vector := "000"; -- encodage des états
    constant ST1: std_logic_vector := "001";
    constant ST2: std_logic_vector := "010";
    constant ST3: std_logic_vector := "101";
    constant ST4: std_logic_vector := "111";

    signal current_state, next_state: std_logic_vector(2 downto 0);
```

```
begin
    ------------------------------------------------------------
    NSL: process (current_state, A, B)  -- processus combinatoire
    begin
        next_state <= current_state; -- pour éviter un registre
        case current_state is
            when ST0 => if A = '1' then next_state <= ST1; end if;
            when ST1 => if A = '1' then next_state <= ST2;
                        else            next_state <= ST0; end if;
            when ST2 => next_state <= ST3;
            when ST3 => if B = '1'     then next_state <= ST3;
                        elsif A = '1' then next_state <= ST4;
                        else               next_state <= ST0; end if;
            when ST4 => next_state <= ST0;
            when others => next_state <= current_state;  -- pour couvrir tous les cas possibles
        end case;
    end process NSL;
    ------------------------------------------------------------
    SM: process  -- processus séquentiel avec reset synchrone
    begin
        wait until CLK = '1';
        if RST = '1' then
            current_state <= ST0;
        else
            current_state <= next_state;
        end if;
    end process SM;
    ------------------------------------------------------------
    Z_Mo <= current_state(0);
end MooreEnc;
```

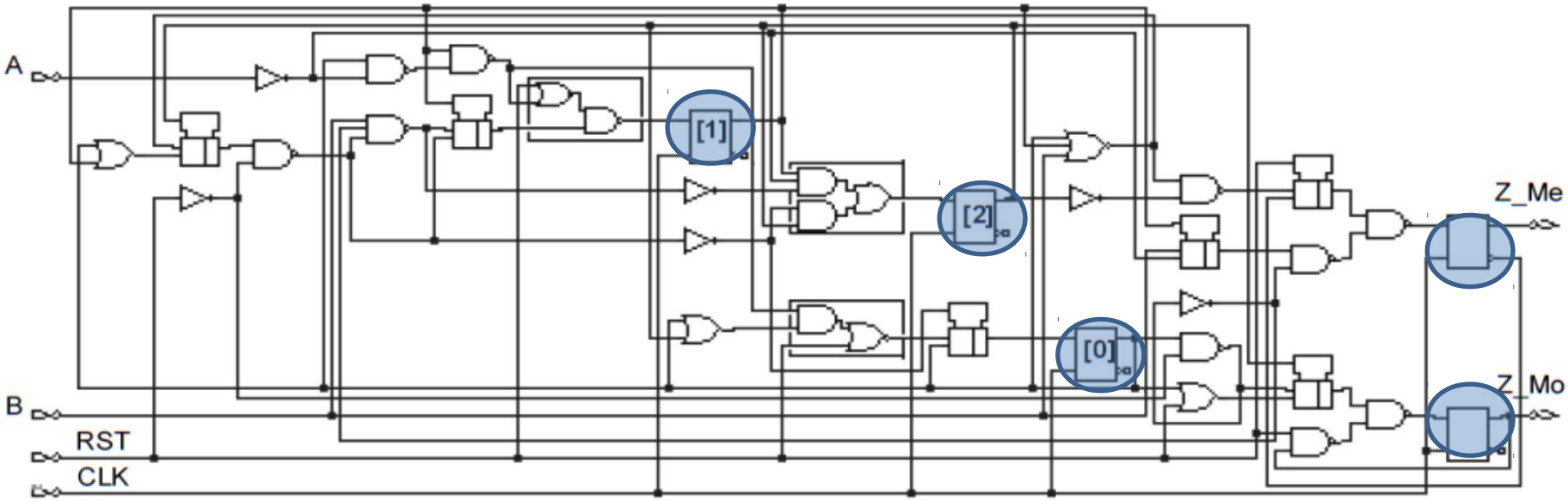
a) Registre explicite et processus

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FSM is
    port (CLK, RST:     in  std_logic;  -- signaux de contrôle
          A, B:         in  std_logic;  -- entrées primaires
          Z_Me, Z_Mo: out std_logic); -- sorties primaires
end FSM;
```

```
architecture oneproc of FSM is
    type fsm_states is (ST0, ST1, ST2, ST3, ST4);
    signal state: fsm_states;
begin
    process
    begin
        wait until CLK = '1';
        if RST = '1' then      -- reset synchrone
            state <= ST0;
        else
            case state is
                when ST0 => ----------------------------------------
                    Z_Mo <= '0';     -- sortie de Moore
                    if A = '0' then
                        Z_Me <= '1';   -- sortie de Mealy
                    else
                        Z_Me <= '0';
                        state <= ST1;
                    end if;
                when ST1 => ----------------------------------------
                    Z_Mo <= '1';
                    if A = '1' then
                        Z_Me <= '1';
                        state <= ST2;
                    else
                        Z_Me <= '0';
                        state <= ST0;
                    end if;
```

```
                when ST2 => ----------------------------------------
                    Z_Mo <= '0';
                    Z_Me <= '0';
                    state <= ST3;
                when ST3 => ----------------------------------------
                    Z_Mo <= '1';
                    if B = '1' then
                        Z_Me <= '0';
                        state <= ST3;
                    else
                        Z_Me <= '1';
                        if A = '1' then
                            state <= ST4;
                        else
                            state <= ST0;
                        end if;
                    end if;
                when ST4 => ----------------------------------------
                    Z_Mo <= '1';
                    Z_Me <= '0';
                    state <= ST0;
            end case;
        end if;
    end process;
end oneproc;
```

Circuit synthétisé



Génération de 5 bascules :
3 pour l'état
2 pour les sorties synchrones

b) Usage de plusieurs processus

L'utilisation de plusieurs processus (trois) permet de simplifier l'écriture en séparant clairement :

- La génération des sorties
- Le calcul des transitions
- La mémorisation de l'état courant

Code à 3 processus

```
architecture threeproc of FSM is
    type fsm_states is (ST0, ST1, ST2, ST3, ST4);
    signal current_state, next_state: fsm_states;
begin
    ------------------------------------------------------------------------------------------------
    NSL: process (current_state, A, B)   -- processus combinatoire
    begin
        next_state <= current_state;  -- pour éviter un registre
        case current_state is
            when ST0 =>
                if A = '1' then
                    next_state <= ST1;
                end if;
            when ST1 =>
                if A = '1' then
                    next_state <= ST2;
                else
                    next_state <= ST0;
                end if;
            when ST2 => next_state <= ST3;
            when ST3 =>
                if B = '1'  then
                    next_state <= ST3;
                elsif A = '1' then
                    next_state <= ST4;
                else
                    next_state <= ST0;
                end if;
            when ST4 => next_state <= ST0;
        end case;
    end process NSL;
```

NSL = Next State Logic

```
    OL: process (current_state, A, B)  -- processus combinatoire
    begin
        Z_Mo <= '0'; Z_Me <= '0';
        case current_state is
            when ST0 =>
                Z_Mo <= '0';
                if A = '0' then
                    Z_Me <= '1';
                else
                    Z_Me <= '0';
                end if;
            when ST1 =>
                    Z_Mo <= '1';
                    if A = '1' then
                        Z_Me <= '1';
                    else
                        Z_Me <= '0';
                    end if;
            when ST2 =>
                    Z_Mo <= '0';
                    Z_Me <= '0';
            when ST3 =>
                    Z_Mo <= '1';
                    if B = '1' then
                        Z_Me <= '0';
                    else
                        Z_Me <= '1';
                    end if;
            when ST4 =>
                    Z_Mo <= '1';
                    Z_Me <= '0';
        end case;
    end process OL;
end threeproc;
```

Sortie logique

Partie générique pour la mémorisation

```
-------------------------------------------------------------------------------
SM: process  -- processus séquentiel avec reset synchrone
begin
    wait until CLK = '1';
    if RST = '1' then
        current_state <= ST0;
    else
        current_state <= next_state;
    end if;
end process SM;
```

4) Exemple

On veut réaliser le contrôleur d'un multiplieur qui utilise
des additionneurs (lancés par le signal ADD) et

des décalages (signal interne SHIFT)

On utilise le signal interne INIT pour le reset du multiplieur complet (adder+shifter)

Et le signal LSB qui indique le bit de poids faible du multiplicande : le registre A

Le signal d'entrée STB démarre le calcul

Le signal de sortie done indique la fin du calcul

On implémente une machine de Moore

Schéma en blocs

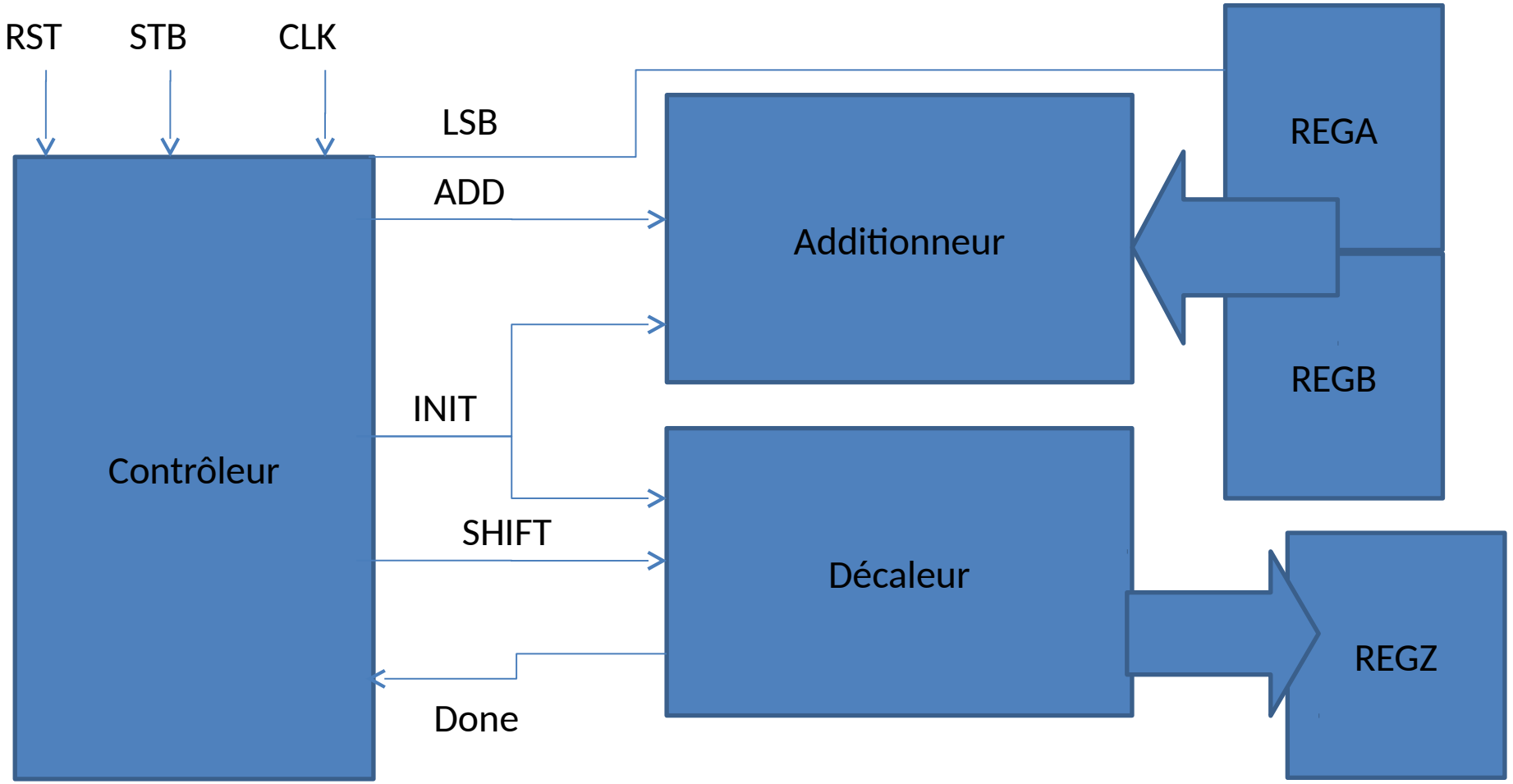
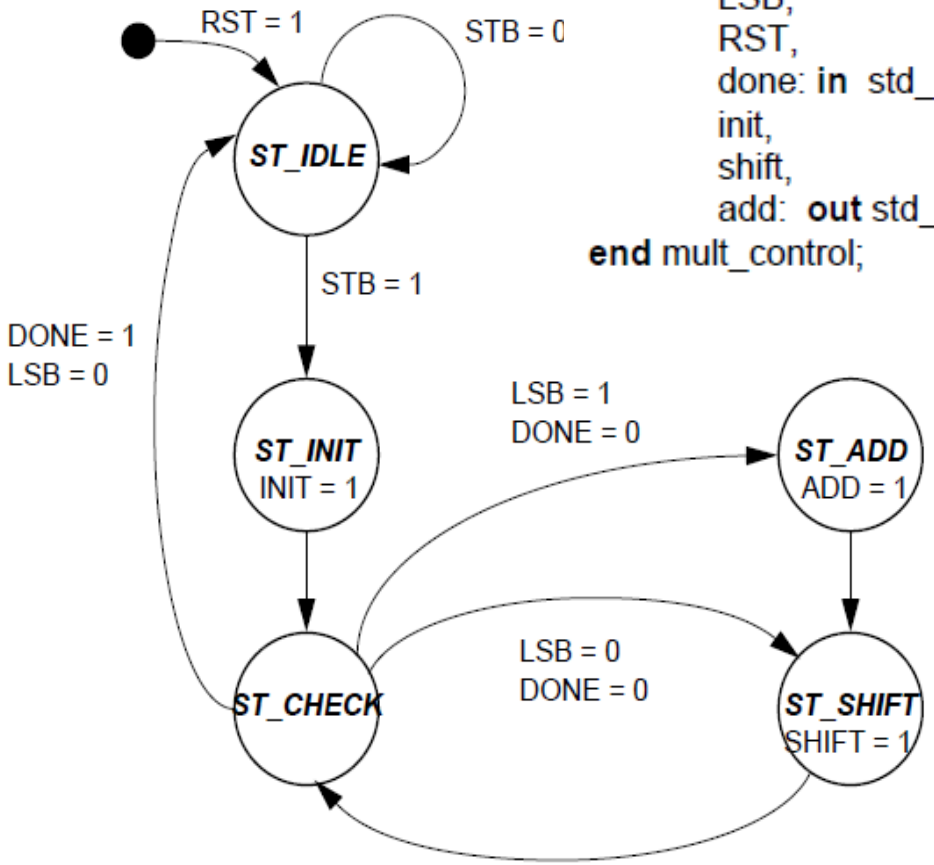


Diagramme du contrôleur




```
library IEEE;
use IEEE.std_logic_1164.all;
entity mult_control is
    port (STB,                  -- commande de démarrage de la multiplication
          CLK,                  -- horloge
          LSB,                  -- bit de poids faible du registre REGA
          RST,                  -- réinitialisation du contrôle
          done: in  std_logic;  -- fin du calcul courant
          init,                 -- initialisation des registres et de l'accumulateur
          shift,                -- décalage des registres
          add:  out std_logic); -- addition avec accumulation
end mult_control;
```

Code du contrôleur

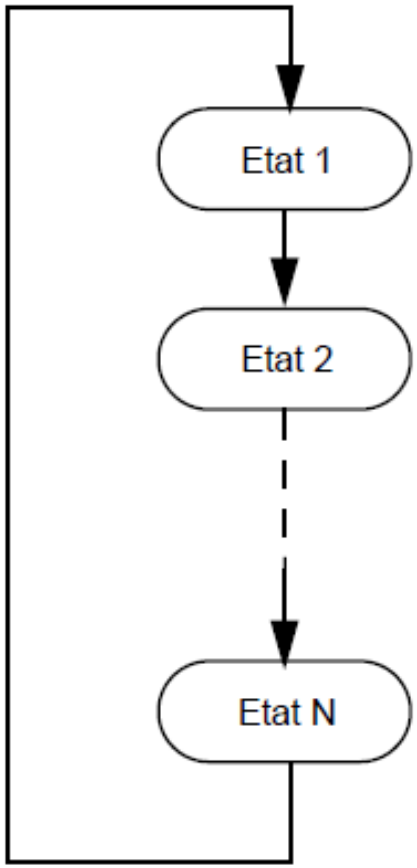
```
architecture fsm of mult_control is
    -- déclaration des états du multiplieur et du signal mémorisant l'état courant
    type mult_states is (ST_IDLE,      -- attente
                         ST_INIT,      -- initialisation
                         ST_CHECK,     -- test de fin d'opération
                         ST_ADD,       -- addition
                         ST_SHIFT);    -- décalage
    signal state: mult_states;
begin

    -- les signaux de contrôle dépendent de l'état courant
    -- les trois instructions concurrentes pourraient être groupées dans un seul processus OL

    init    <= '1' when state = ST_INIT   else '0';
    add     <= '1' when state = ST_ADD    else '0';
    shift   <= '1' when state = ST_SHIFT  else '0';

    -- détermination du prochain état
```

```
    NSLSM: process (CLK, RST)
    begin
        if RST = '1' then  -- reset asynchrone
            state <= ST_IDLE;
        elsif CLK'EVENT and CLK = '1' then
            case state is
                when ST_IDLE =>
                    if STB = '1' then
                        state <= ST_INIT;
                    end if;
                when ST_INIT =>  state <= ST_CHECK;
                when ST_CHECK =>
                    if LSB = '1' then
                        state <= ST_ADD;
                    elsif done = '0' then
                        state <= ST_SHIFT;
                    else
                        state <= ST_IDLE;
                    end if;
                when ST_ADD  => state <= ST_SHIFT;
                when ST_SHIFT => state <= ST_CHECK;
            end case;
        end if;
    end process NSLSM;
end fsm;
```

5) Modélisation sans registre explicite

L'utilisation de séquences d'instruction wait sur le même signal modélise une FSM sans nommage des états :

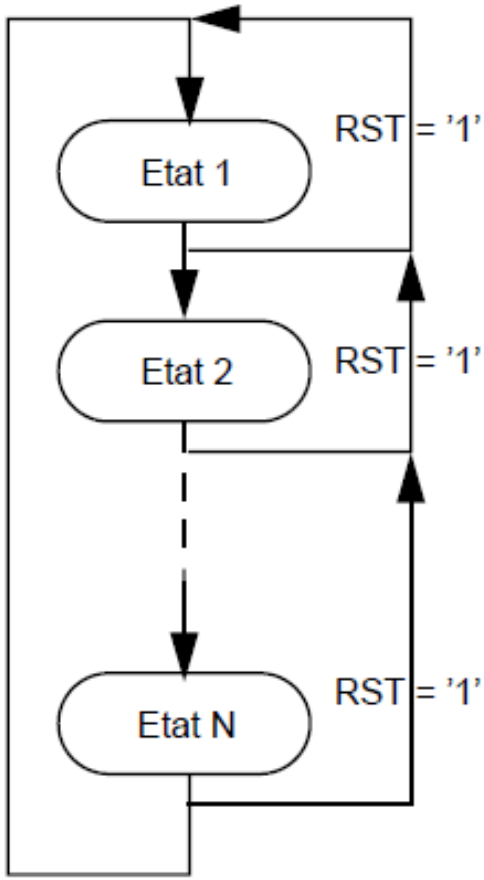
```
process
begin
    wait until CLK = '1';
    -- traitement de l'état 1
    wait until CLK = '1';
    -- traitement de l'état 2
    ...
    wait until CLK = '1';
    -- traitement de l'état N
end process;
```



Cette méthode convient seulement pour les comportements itératifs.

a) Exemple avec reset synchrone

```
process
    procedure etat_1 (...) is
    begin
        -- traitement de l'état 1
    end procedure etat_1;
begin
    loop
        wait until CLK = '1';
        if RST = '1' then
            etat_1(...);
            exit;
        end if;
        -- traitement de l'état 2
        ...
        wait until CLK = '1';
        if RST = '1' then
            etat_1(...);
            exit;
        end if;
        -- traitement de l'état N
        wait until CLK = '1';
        loop
            etat_1(...);
            exit when RST = '0';
        end loop;
    end loop;
end process;
```



b) Exemple avec reset asynchrone

```
process
begin
    RST_LABEL: loop
        if RST = '1' then
            -- initialisation
        end if;
        wait until (CLK'EVENT and CLK = '1') or RST = '1';
        next RST_LABEL when RST = '1';
        -- traitement de l'état 1
        wait until (CLK'EVENT and CLK = '1') or RST = '1';
        next RST_LABEL when RST = '1';
        -- traitement de l'état 2
        ...
        wait until (CLK'EVENT and CLK = '1') or RST = '1';
        next RST_LABEL when RST = '1';
        -- traitement de l'état N
    end loop RST_LABEL;
end process;
```

6) Exemple du calcul de convolution

Le produit de convolution (en traitement de signal) se calcule de la manière suivante :

Où xi représentent les valeurs des échantillons capturés sur l'horloge Clk et ai les coéfficients.

$$C = \sum_{i=0}^{N-1} a_i \cdot x_i$$

a) Entité du convolueur

```
package conv_pkg is
    type integ_arr is array (NATURAL range <>) of INTEGER;
end conv_pkg;

use WORK.conv_pkg.all;
entity convolution is
    generic (coeff: integ_arr;      -- vecteur des coefficients
             Nbi: POSITIVE;         -- nombre de bits du mot d'entrée
             Nbo: POSITIVE);        -- nombre de bits du mot de sortie
    port (CLK, RST: in  BIT;                                          -- horloge, reset
          X       : in  INTEGER range -2**(Nbi-1) to 2**(Nbi-1)-1;    -- signal à échantillonner
          C       : out INTEGER range -2**(Nbo-1) to 2**(Nbo-1)-1;    -- produit
          valid   : out BIT);                                         -- résultat valide
end convolution;
```

b) Architecture FSM

```
architecture fsm of convolution is
begin
    process
        variable CC: INTEGER range -2**(Nbo-1) to 2**(Nbo-1)-1;
    begin
        loop
            wait until CLK = '1';
            valid <= '0'; CC := coeff(0) * X;
            exit when RST = '0';
            for i in 1 to coeff'RANGE loop
                wait until CLK = '1';
                if RST = '1' then
                    valid <= '0';
                    CC := coeff(0) * X;
                    exit;
                else
                    CC := CC + coeff(i) * X;
                end if;
            end loop;
            valid <= '1';
        end loop;
        C <= CC;
    end process;
end fsm;
```

7) Notions importantes

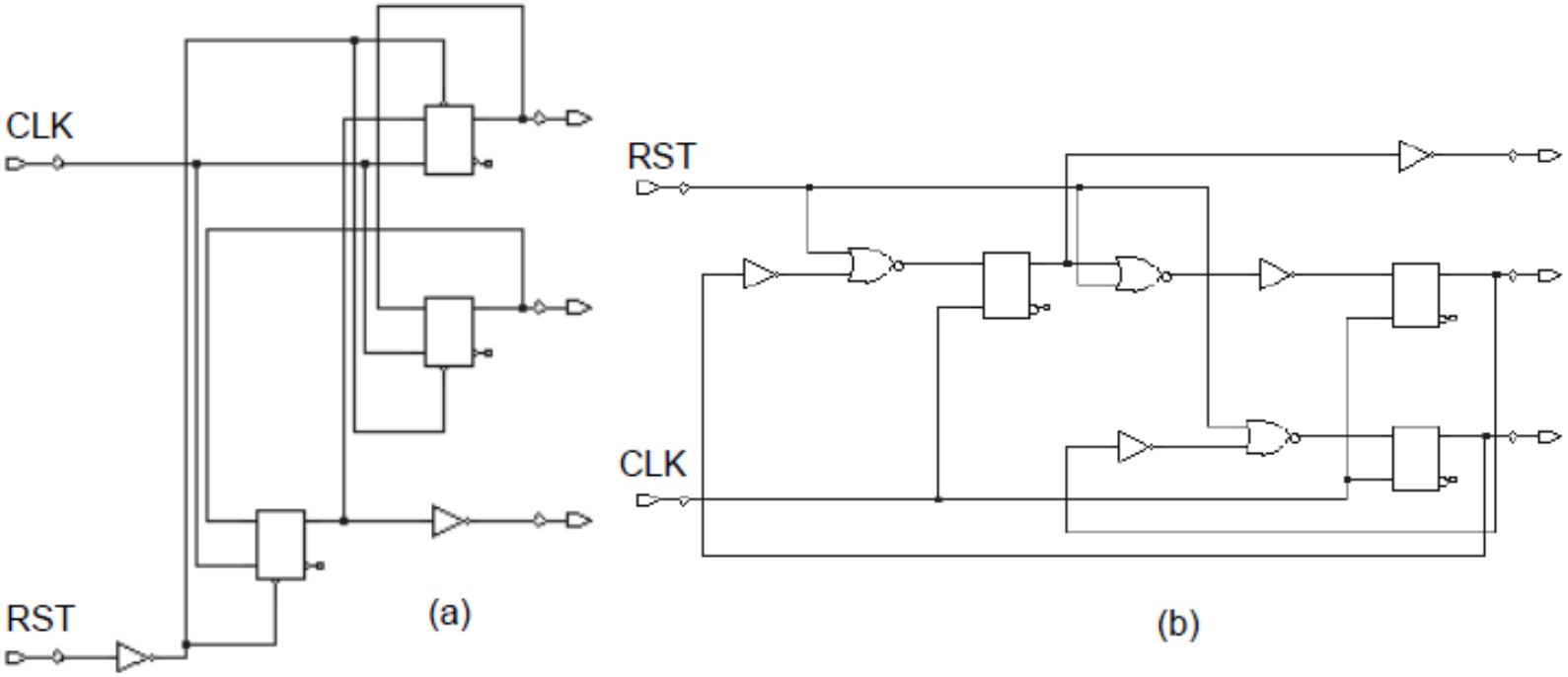
a) Reset

Le reset synchrone requiert plus de logique combinatoire que l'asynchrone car il faut que tous les états possibles soient pris en compte dans le calcul du prochain état.
Ceci est possible en ajoutant par exemple une clause **when others** au processus NSL en assignant le prochain état next_state à un état valide.
Si ce n'est pas fait, la machine d'état peut démarrer dans un état invalide sans qu'elle puisse plus sortir, mê me avec le reset.

b) Encodage des états

Les codages possibles sont :
Séquentiel, avec des valeurs binaires croissantes
De Gray, et de Johnson, qui ont la propriété de ne changer que d'un seul bit d'un état à l'autre
One-hot, utilise un seul flip-flop par état, pour des circuits plus rapides, mais occupant plus de ressources
Codage spécifique (custom)

Comparaison de 2 FSM avec reset asynchrone (a) ou synchrone (b)



Les FSM à RST synchrone occupent en moyenne 15% de ressources en plus

Formats d'encodage

No.	Séquentiel	Gray	Johnson	One-hot
0	0000	0000	00000000	0000000000000001
1	0001	0001	00000001	0000000000000010
2	0010	0011	00000011	0000000000000100
3	0011	0010	00000111	0000000000001000
4	0100	0110	00001111	0000000000010000
5	0101	0111	00011111	0000000000100000
6	0110	0101	00111111	0000000001000000
7	0111	0100	01111111	0000000010000000
8	1000	1100	11111111	0000000100000000
9	1001	1101	11111110	0000001000000000
10	1010	1111	11111100	0000010000000000
11	1011	1110	11111000	0000100000000000
12	1100	1010	11110000	0001000000000000
13	1101	1011	11100000	0010000000000000
14	1110	1001	11000000	0100000000000000
15	1111	1000	10000000	1000000000000000

Biblio

Documents de cours

- Ce cours en ligne à :

Http://perso-etis.ensea.fr/rodriguez/

- Un très bon support de cours

http://hdl.telecom-paristech.fr/index.html

- Le cours de Licence 2 (en particulier pour ceux qui ne l'ont pas suivi):

http://perso-etis.ensea.fr/miramond/Enseignement/L2/circuits_numeriques.html

- Le cours de Licence 3 de 2013 :

http://perso-etis.ensea.fr/miramond/Enseignement/L3/Cours_VHDL_2011.html

Documents de développement

- Quartus

http://quartushelp.altera.com/current/

- Documentation Altera sur les Cyclones II et IV (entre autre...)

http://www.altera.com/literature/lit-index.html