System on Chip FPGA

Méthodologie de conception

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Février 2024

1. Méthodologie
2. Systèmes synchrones
3. Machine de Mealy: Exemple
4. Machine de Moore: Exemple
5. Gated clock
6. Reset
7. Méthodologie avancée

Structure d'un projet

- Nom de répertoires :
  - uniquement des lettres, chiffres et souligné _
- Un projet doit être bien structuré:

`<Dir_Projet>`		scripts
	`\src`	fichiers sources VHDL
	`\src_tb`	fichiers sources des bancs de test
	`\lib`	bibliothèques, paquetages
	`\scripts`	fichiers pour la simulation (QuestaSim)
	`\sim`	répertoire depuis lequel la simulation se lance
	`\synth`	fichiers pour la synthèse (Precision)
	`\p_r`	fichiers pour le placement& routage (Quartus)

Conception et Module VHDL

- Décomposer votre projet
  - Une seule fonction par module VHDL
  - Vos descriptions doivent être simples et lisibles
  - Les descriptions doivent être faciles à comprendre pour le synthétiseur

  $\Rightarrow$ permet une meilleure optimisation lors de la synthèse

Mots réservés et conventions REDS pour identificateurs

⚠ Conventions REDS

- Mots réservés :
  - Ils seront écrits en minuscule, affichés en couleur
- Identificateurs :
  - Ils seront écrits en minuscule
  - Les mots sont séparés par un souligné (underscore)
- Exemple : `bus_data`, `entree_serie`, ...
- Il est à noter que le VHDL ne tolère que des lettres, des chiffres et le souligné pour les identificateurs

Conventions REDS pour les identificateurs

⚠ Conventions REDS

Déclaration au top	
`<nom_signal>_i`	entrée (`in`)
`<nom_signal>_o`	sortie (`out`)
`<nom_signal>_io`	entrée/sortie (`inout`)
`n<Nom_signal>_i`	entrée active bas
`n<Nom_signal>_o`	sortie active bas
`n<Nom_signal>_io`	entrée/sortie active bas

Conventions REDS pour les identificateurs

⚠ Conventions REDS

Déclaration dans l'entité	
`<nom_signal>_i`	entrée (`in`)
`<nom_signal>_o`	sortie (`out`)
`<NOM_CONSTANTE>`	constante générique
Déclaration dans l'architecture	
`<nom_signal>_s`	signal interne
`<NOM_CONSTANTE>`	constante
Déclaration dans un processus	
`<nom_variable>_v`	variable

⚠ Le cas des constantes est particulier: En majuscule

Conventions REDS pour les identificateurs

⚠ Conventions REDS

Dans les fichiers de simulation	
`<nom_signal>_sti`	signal de stimulus (entrée)
`<nom_signal>_obs`	signal observé (sortie)
`<nom_signal>_ref`	signal de référence (calculé)

Choisir des noms explicites!

- Noms des composants et des signaux doivent être explicites
  - Auto-documente le projet
  - Facilite la lecture des descriptions, des schémas, ...
- Nom d'un composant :
  - Doit indiquer la fonction
- Noms des signaux dans un composant :
  - Explicite avec la fonction interne
- Noms des signaux dans un schéma :
  - Explicite avec la fonction réalisée dans le schéma

Fichiers VHDL

- Un fichier VHDL ne contient qu'une seule entité et son architecture
  - Potentiellement plusieurs architectures
- Nom du fichier:
  - Identique au nom de l'entité
  - Ne contient que des lettres, chiffres, underscore
- Nom du fichier contenant le banc de test:
  - `<nom_entite>_tb.vhd`
  - Est cohérent avec le nom du banc de test: `<nom_entite>_tb`

Fichiers VHDL: entêtes

Entête de fichier type (en anglais)

```
------------------------------------------------------------
-- Project QCrypt
--
-- File : name_module.vhd
-- Description : Short module description.
--
-- Author : J. Brown
-- Team   : REDS institute
-- Date   : 23.02.22
--
--
--| Modifications |-------------------------------------
-- Ver  Date      Who  Description
-- 1.1  25.02.22  JBR  New generic parameter
--
------------------------------------------------------------
```

Architecture

- Le nom de l'architecture est choisi en fonction du style de description

Style	**Nom d'architecture**
Structurel	`struct`
Comportemental	`behave`
Banc de test	`testbench`
Machine d'état	`fsm`
Flot de données	`dataflow`
Modèle pour simulation	`model`

Choix d'architecture

- Quel type d'architecture choisir?
- Décomposer chaque fois que c'est pertinent:
  - Architecture structurelle
  - Si un composant pourrait être utilisé ailleurs
- Utiliser le comportemental ou data flow lorsque la décomposition n'est plus pertinente
  - Data flow
    - Proche des portes logiques
    - Plus de contrôle sur la logique générée
  - Comportemental
    - Risque de faire de l'algorithmique éloignée de la synthèse
    - Potentiellement plus clair que data flow
    - FSM : est en fait du comportemental

Choix d'architecture

- Ne pas hésiter à réaliser plusieurs processus si une décomposition n'est pas adéquate
- Typique pour les machines d'états (cf. plus loin)
- Responsabilités des processus
- Exemple:
  - Une architecture qui contient une mémoire
  - Soit la mémoire est instanciée
  - Soit elle est décrite dans un processus à part
  - Garanti une synthèse pertinente

Plusieurs architectures

- Il peut être intéressant d'avoir plusieurs architectures pour une même entité
  - Non synthétisable pour tester rapidement le concept
  - Non synthétisable pour valider le banc de test
  - Plusieurs versions synthétisables
    - Timings
    - Usage des ressources

Au top du projet

- Adapter la polarité des signaux au top
  - logique mixte à l'extérieur (polarité positive ou négative)
  - à l'intérieur uniquement en logique positive

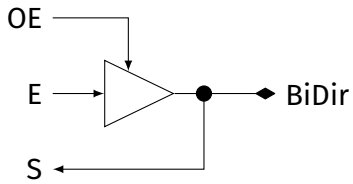
  ⇒ Pas de `nReset` en interne
- Signaux bidirectionnels SEULEMENT au top
  - au top: instanciation des portes 3 états
  - transmettre dans la hiérarchie des signaux unidirectionnels
  - Identifier les signaux en entrée, en sortie ou bidirectionnel
- Type des signaux au top uniquement :
  - `std_logic`, `std_logic_vector`

Au top du projet

- Instanciation des portes bi-directionnelles au top
- Utilisation de la fonction `To_X01`

```
BiDir_io <= E_i when OE_i = '1' else 'Z';
S_o      <= To_X01(BiDir_io);
```

- La fonction `To_X01` permet de convertir l'état 'H' ou 'L' dans un état logique '1' ou '0' respectivement



Polarité des signaux

- Utiliser uniquement des signaux en logique positive à l'intérieur des descriptions.
- Adapter la polarité des signaux au top du projet.
- Indiquer les signaux en logique négative.
  - exemple : `nReset_i`

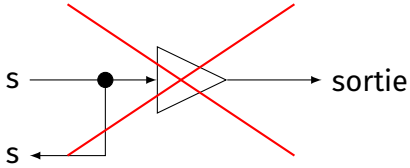
```
signal reset_s      : std_logic;
signal data_valid_s : std_logic;

reset_s       <= not nReset_i;  <-- Adaptation du signal d'entrée

nData_valid_o <= not data_valid_s; <-- Adaptation du signal de sortie
```

Types de ports

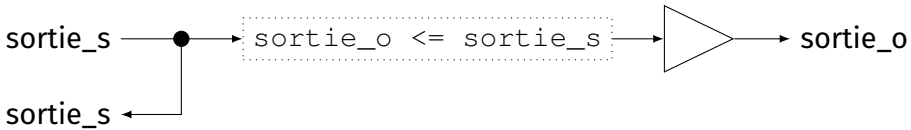
- Utiliser : `in`, `out`, `inout`
- Le type inout sera utilisé uniquement dans le module top
  - Il n'y a pas de ligne trois états à l'intérieur d'un PLD (sauf cas particulier!)
  - possible dans le cas des ASICs
- Le type buffer ne sera jamais utilisé (voir remplacement)

Types de ports

Un port de type `buffer`:



A remplacer par un signal interne et un port `out`:



Déclaration des vecteurs

- Toujours utiliser la déclaration :
  - `n-1 downto 0`
  - Cela correspond à avoir le bit de poids fort à gauche
- Exemple : `signal vecteur : std_logic_vector(31 downto 0);`
- Problème si le vecteur est défini dans l'autre sens
  - `0 to n-1`
- Il y a inversion des bits lors de l'affectation

```
signal vect32 : std_logic_vector(0 to 31);
vecteur <= vect32; --Tous les bits sont croisés!
```

Bibliothèques en synthèse

- Bibliothèque IEEE autorisées :
  - `std_logic_1164`: types et fonctions de base
  - `numeric_std`: opérateurs arithmétiques
- Bibliothèques propres à l'entreprise :
  - A utiliser avec modération (gestion des versions)
  - Fonctions et procédures pour la simulation
  - etc.

Processus combinatoire

- L'écriture d'un processus combinatoire doit suivre 2 règles:
  - Règle 1: Tous les signaux affectés dans le processus doivent se voir assigner une valeur par défaut en début de processus
  - Règle 2: Un signal ne peut être en entrée et sortie du processus (utilisé et affecté)
  - Règle 3: Tous les signaux présents à droite d'une affectation ou dans un test doivent être déclarés dans la liste de sensibilité du processus

Bon exemple

```
process(a,b,c,state) is
begin
    s <= (others=>'0');
    next_state <= state;
    ...
    if (c < 10) then
        s <= a + b;
        next_state <= S2;
    end if;
end process;
```

Mauvais exemple

```
process(a) is
begin

    b <= a + 2;
    ...
    if (c < 10) then
        s <= a + b;
        next_state <= S2;
    end if;
 end process;
```

Processus combinatoire - liste de sensibilité

- En VHDL-2008 il est possible d'écrire ceci:

Bon exemple

```
process(all) is
begin
    s <= (others=>'0');
    next_state <= state;
    ...
    if (c < 10) then
        s <= a + b;
        next_state <= S2;
    end if;
end process;
```

- Il est suggéré d'exploiter `process(all)`
- ⚠ Le code doit être compilé en VHDL-2008

Processus séquentiel

- Un processus séquentiel doit suivre les règles suivantes:
  - La liste de sensibilité ne contient que:
    - L'horloge
    - Le reset, si celui-ci est asynchrone
  - Toutes les affectations sont faites dans le `if (rising_edge(clk_i))`
  - Et dans le `if (reset_i='1')` si le reset est asynchrone
  - Aucune affectation permise en dehors de ces `if`
  - Les valeurs affectées au reset doivent être fixes
    - Ne dépendent pas d'autres signaux que le reset

Processus séquentiel

Bon exemple

```
process(clk_i,reset_i) is
begin
  if ( reset_i = '1' ) then
    a <= '0';
    b <= (others => '0');
  elsif (rising_edge(clk_i)) then
    a <= next_a;
    if (affecte = '1') then
      b <= c + d;
    end if;
  end if;
end process;
```

Mauvais exemple

```
process(clk_i) is
begin
  if ( reset_i = '1' ) then
    a <= '0';
    b <= (others => '0');
  elsif (rising_edge(clk_i)) then
    a <= next_a;
    if (affecte = '1') then
      b <= c + d;
    end if;
  else
      b <= (others=>'0');
  end if;
end process;
```

Labels

- Les processus peuvent posséder un label
- Un label pertinent pour chaque processus permet de mieux retrouver les variables internes (notamment)
- Les `generate` doivent posséder un label
- Le label doit être le plus proche de la fonctionnalité du processus

Exemple

```
clockProc : process is
begin
    clk <= '0';
    wait for CLK_PERIOD/2;
    clk <= '1';
    wait for CLK_PERIOD/2;
end process;
```

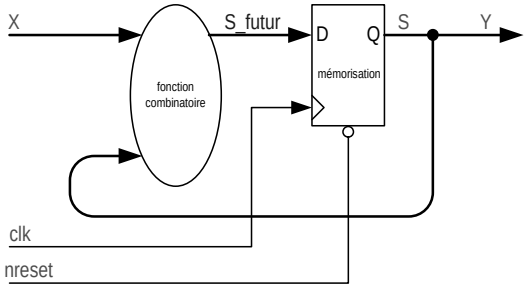
Types de design

- Un design, ou module peut être:
  1. Purement combinatoire
     - Description via des processus implicites ou explicites
  2. Synchrone (ou séquentiel)
     - Combinaison de processus synchrones et combinatoires

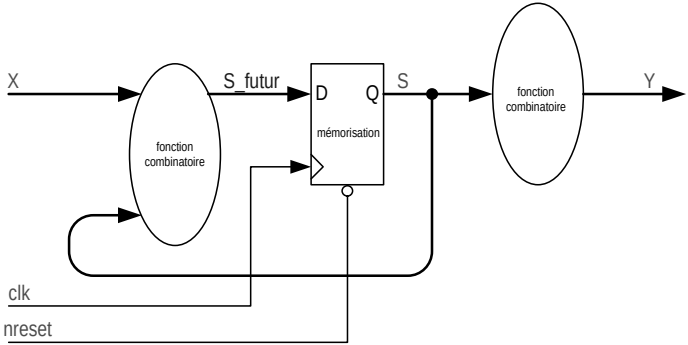
Types de systèmes synchrones

- Posons:
  - X: le vecteur d'entrée
  - S: le vecteur d'état
  - Y: le vecteur de sortie
- On définit les types de machines suivants:
  - Medvedev
    - Le vecteur de sortie Y correspond au vecteur d'état S
    - Y=S
  - Moore
    - Le vecteur de sortie Y est fonction du vecteur d'état S
    - Y=f(S)
  - Mealy
    - Le vecteur de sortie Y est fonction du vecteur d'état S ET du vecteur d'entrée X
    - Y=f(S,X)
- Un système combinatoire correspond à Y=f(X)

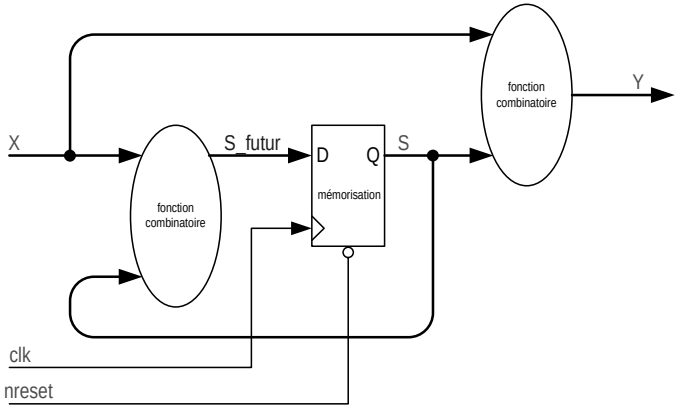
Machine de Medvedev: Y=S



Machine de Moore: Y=f(S)

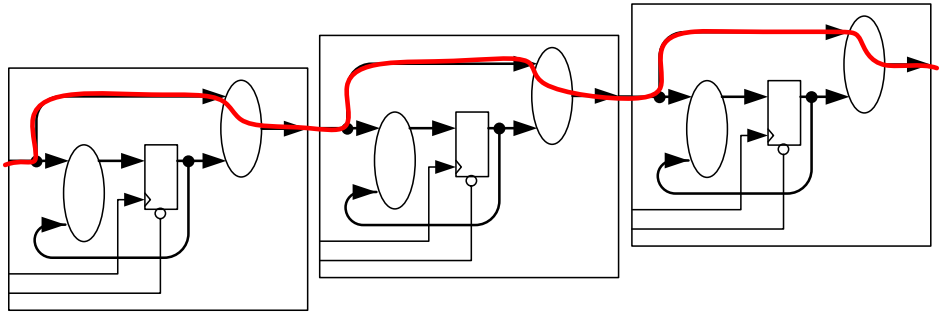


Machine de Mealy: Y=f(X,S)

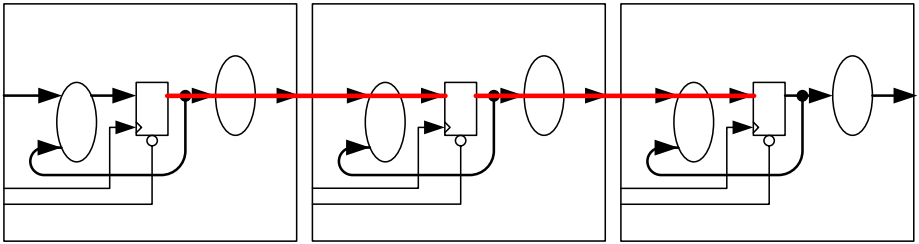


Machine de Mealy: Problème

- En chaînant des machines de Mealy, il faut éviter de propager les signaux combinatoires!!!!!!

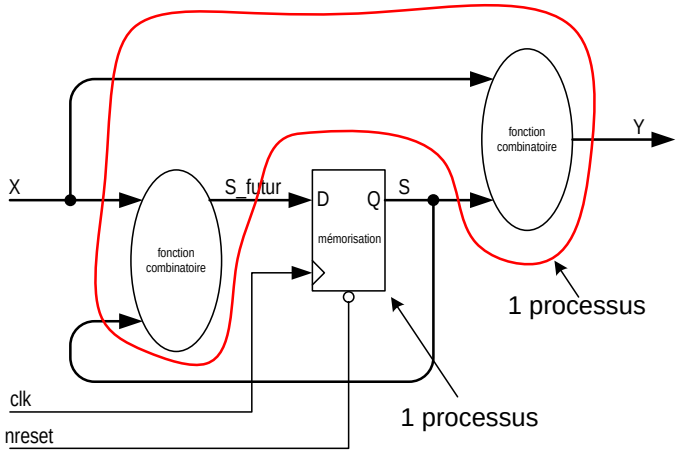


Machine de Moore: Pas de problème, mais...



- Il y a forcément un délai d'un coup d'horloge pour modifier les sorties.

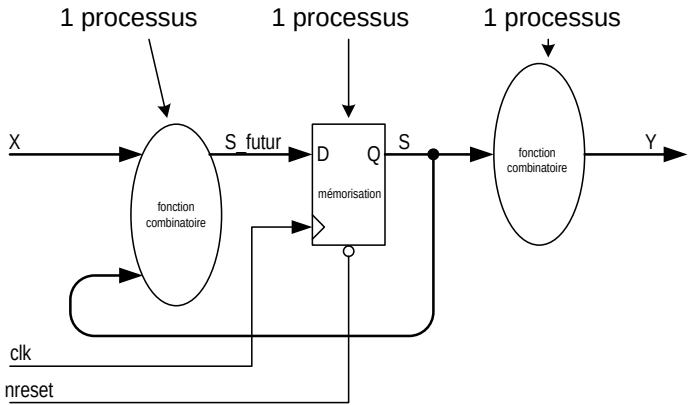
Machine de Mealy: Implémentation



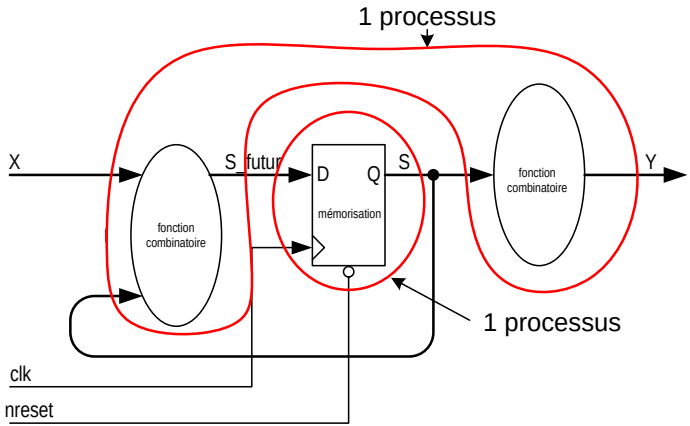
Machine de Mealy: 2 processus

```
sorties_n_etats: process(all)
begin
    -- définition des sorties en fonction de l'état et des entrées
    ...
    -- définition de l'état suivant en fonction de l'état et des entrées
    etat_futur<= ...;
end process;

etats: process(rst_i, clk_i)
begin
    if rst_i='1' then
        etat <= valeurs initiales
    elsif rising_edge(clk_i) then
        etat <= etat_futur;
    end if;
end process;
```

Machine de Moore: Implémentation



Machine de Moore: Implémentation à 2 processus

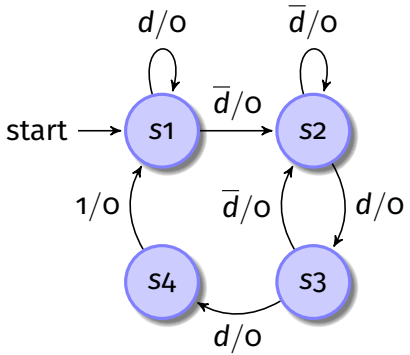


Machine de Moore: Implémentation à 2 processus

```
sorties_etatfutur: process(all) begin
    -- définition des sorties en fonction de l'état
    ...
    -- définition de l'état suivant en fonction de l'état et des entrées
    etat_futur <= ...;
end process;

etats: process(rst_i, clk_i)
begin
    if rst_i='1' then
        etat <= valeurs initiales
    elsif rising_edge(clk_i) then
        -- affectation des registres avec l'état futur
        etat <= etat_futur;
    end if;
end process;
```

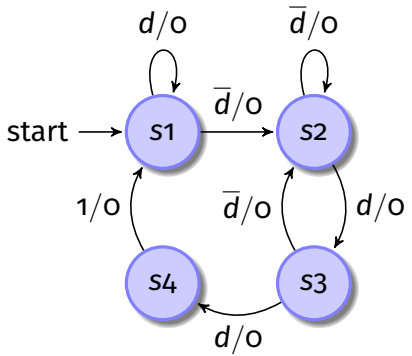
Machine de Mealy: Exemple

- Un détecteur de séquence 0110.
- Le signal de détection passe à '1' de manière asynchrone lorsque la séquence est détectée.



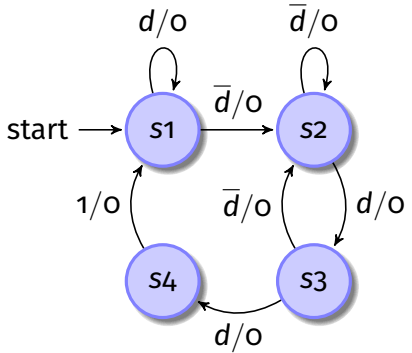
état	!d	d
s1	s2/0	s1/0
s2	s2/0	s3/0
s3	s2/0	s4/0
s4	s1/1	s1/0

Machine de Mealy: Exemple



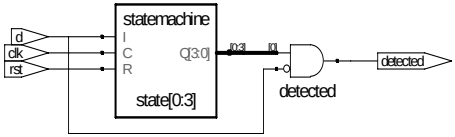

```
library ieee;
use ieee.std_logic_1164.all;

entity seqdetect is
port(clk_i      : in  std_logic;
     rst_i      : in  std_logic;
     d_i        : in  std_logic;
     detected_o : out std_logic);
end seqdetect;

 architecture fsm of seqdetect is

  type state_type is (s1,s2,s3,s4);
  signal state   : state_type;
  signal n_state : state_type;

begin

  reg: process(rst_i, clk_i)
  begin
    if rst_i = '1' then
      state <= s1;
    elsif rising_edge(clk_i) then
      state <= n_state;
    end if;
  end process;
```

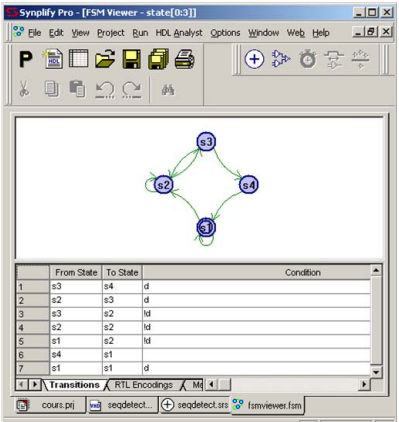
Machine de Mealy: Exemple




```
    next_state: process(all)
    begin
        n_state    <= state;      -- défaut
        detected_o <= '0';        -- défaut
        case state is
        when s1 =>
            if d_i = '0' then
                n_state <= s2;
            end if;
        when s2 =>
            if d_i = '1' then
                n_state <= s3;
            end if;
        when s3 =>
            if d_i = '1' then
                n_state <= s4;
            else
                n_state <= s2;
            end if;
        when s4 =>
            n_state <= s1;
            if d_i = '0' then
                detected_o <= '1';
            end if;
        end case;
    end process;
end fsm;
```

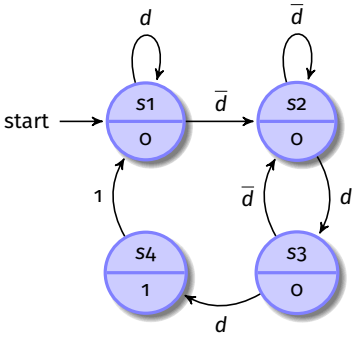
Machine de Mealy: Synthèse





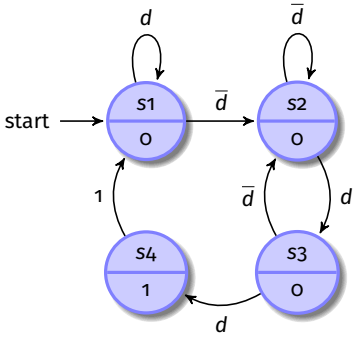
Machine de Moore: Exemple

- Un détecteur de séquence 011.
- Le signal de détection passe à '1' de manière synchrone lorsque la séquence est détectée.



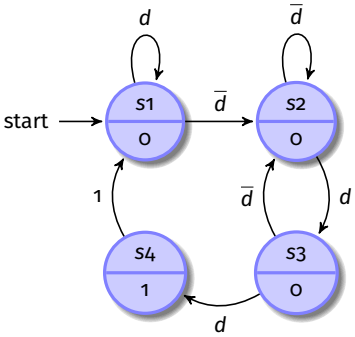
état	!d	d	detected
S1	S2	S1	0
S2	S2	S3	0
S3	S2	S4	0
S4	S2	S1	1

Machine de Moore: Exemple




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

entity seqdetect is
port(clk_i      : in  std_logic;
     rst_i      : in  std_logic;
     d_i        : in  std_logic;
     detected_o : out std_logic);
end seqdetect;

architecture fsm of seqdetect is

  type state_type is (s1, s2, s3, s4);
  signal state   : state_type;
  signal n_state : state_type;

begin

  reg: process(rst_i, clk_i)
  begin
    if rst_i = '1' then
      state <= s1;
    elsif rising_edge(clk_i) then
      state <= n_state;
    end if;
  end process;

  detected_o <= '1' when state=s4 else '0';
```

Machine de Moore: Exemple




```
    next_state: process(all)
    begin
        n_state <= state;          -- défaut
        case state is
        when s1 =>
            if d_i = '0' then
                n_state <= s2;
            end if;
        when s2 =>
            if d_i = '1' then
                n_state <= s3;
            end if;
        when s3 =>
            if d_i = '1' then
                n_state <= s4;
            else
                n_state <= s2;
            end if;
        when s4=>
            if d_i = '0' then
                n_state <= s2;
            else
                n_state <= s1;
            end if;
        end case;
    end process;
end fsm;
```

Gated clock

- Comment gérer un enable sur un registre?
- Exemple: le registre n'est chargé que lorsque `enable` vaut `'1'`

Exemple

```
clk_s <= clk_i when enable_i='1' else '0';

process(reset_i, clk_s) is
begin
    if (reset_i = '1') then
        reg_s <= (others => '0');
    elsif (rising_edge(clk_s)) then
        reg_s <= next_reg_s;
    end if;
end process;
```

- Qu'est-ce que ça donne à la synthèse?

Gated clock

- Les gated clocks sont à éviter
- Les enables doivent être inclus dans la logique combinatoire avant le registre

Exemple

```
process(reset_i, clk_i) is
begin
    if (reset_i = '1') then
        reg_s <= (others => '0');
    elsif (rising_edge(clk_i)) then
        if (enable_s = '1') then
            reg_s <= next_reg_s;
        end if;
    end if;
end process;
```

Reset

- Reset synchrone ou asynchrone?

```
process(rst_i, clk_i) is
begin
    if (rst_i = '1') then
        state <= sInit;
    elsif rising_edge(clk_i) then
        state <= next_state;
    end if;
end process;
```

```
process(clk_i) is
begin
    if rising_edge(clk_i) then
        if (rst_i = '1') then
            state <= sInit;
        else
            state <= next_state;
        end if;
    end if;
end process;
```

- Dépend du projet
- A décider au démarrage
- Peut dépendre de la technologie sous-jacente
- Peut avoir une influence considérable sur la performance

Reset: oui ou non?

- Certains éléments de mémorisations n'ont pas forcément besoin d'un reset
- Reset nécessaire
  - Machine d'états
  - Compteur
- Reset non nécessaire
  - Données dans un pipeline
  - Filtre FIR
  - D'une manière générale, lorsque la donnée stockée n'est pas utilisée avant d'être affectée

Reset: coût

- Ressources de routage
  - Occupe des ressources qui ne sont donc pas disponible pour le reste
  - Augmente le temps nécessaire au routage
- Utilisation des ressources logiques
  - Utilise le reset dédié aux flip-flops
  - Potentiellement ajout de portes logiques avant l'entrée
  - Impacte la taille du design
  - Impacte les performances du système
  - Augmente le temps de placement-routage
- Empêche l'usage des SRL16E (Xilinx)
  - Une SRL16E implémente jusqu'à 16 flip-flops dans chaque LUT
  - Pas de reset supporté par ces éléments
  - Augmentation jusqu'à 16$\times$ en taille
  - L'augmentation risque de réduire les performances du système
  - Augmente le temps de placement-routage

Reset: quand?

- Se poser la question de la pertinence du Reset à chaque fois
- Attention à ne pas mélanger des éléments avec et sans dans un même processus

`a` possède un reset, `b` non

Mauvais exemple

```
process(reset_i, clk_i) is
begin
  if ( reset_i = '1' ) then
    a <= '0';
  elsif (rising_edge(clk_i)) then
    a <= next_a;
    b <= next_b;
  end if;
end process;
```

problème?

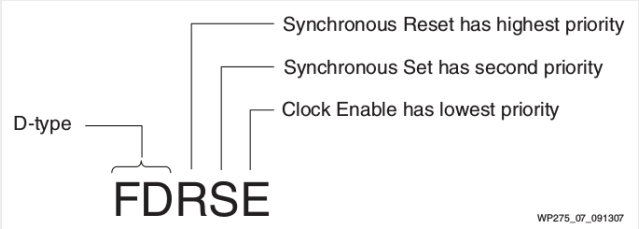
Bon exemple

```
process(reset_i, clk_i) is
begin
  if ( reset_i = '1' ) then
    a <= '0';
  elsif (rising_edge(clk_i)) then
    a <= next_a;
  end if;
end process;

process(clk_i) is
begin
  if (rising_edge(clk_i)) then
    b <= next_b;
  end if;
end process;
```

Processus séquentiel: ordre des opérations

- L'ordre des opérations peut aussi avoir une influence

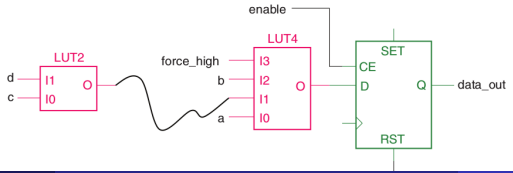
Exemple Xilinx



Processus séquentiel: ordre des opérations

Exemple Xilinx

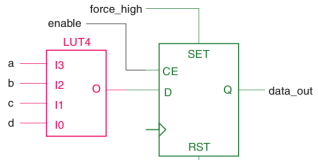
```
process(clk_i)
begin
    if rising_edge(clk_i) then
        if reset_i='1' then
            data_out <= '0';                    <-- reset synchrone
        else
            if enable='1' then                  <-- enable
                if force_high='1' then          <-- set synchrone
                    data_out <= '1';
                else
                    data_out <= a and b and c and d;   <-- logique
                end if;
            end if;
        end if;
    end if;
end process;
```



Processus séquentiel: ordre des opérations

Exemple Xilinx

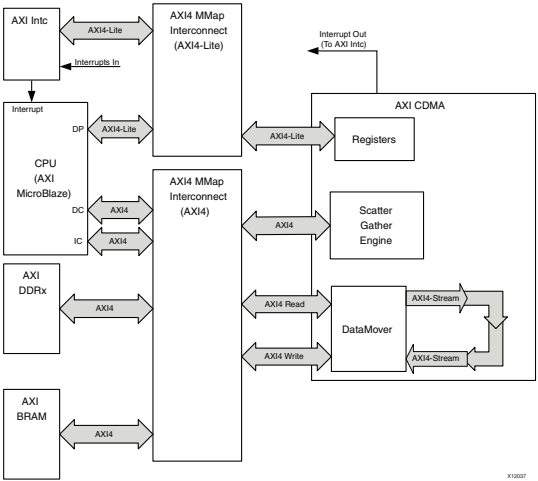
```
process(clk_i)
begin
    if rising_edge(clk_i) then
        if reset='1' then
            data_out <= '0';              <-- reset synchrone
        else
            if force_high='1' then        <-- set synchrone
                data_out <= '1';
            else
                if enable='1' then        <-- enable
                    data_out <= a and b and c and d;   <-- logique
                end if;
            end if;
        end if;
    end if;
end process;
```



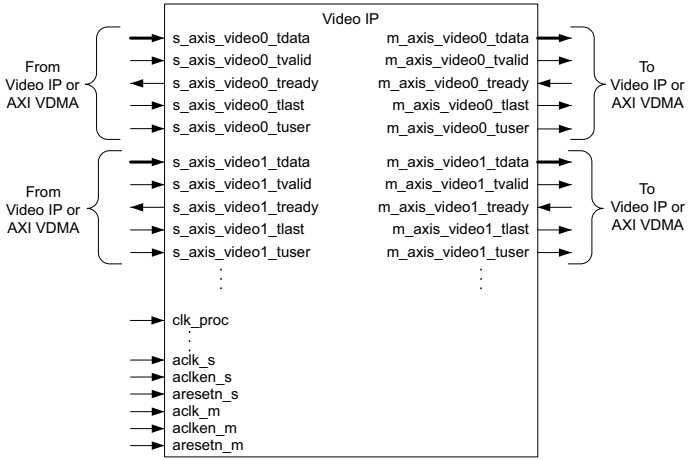
Ports en enregistrements

- Les ports d'entrée/sortie sont en général de type `std_logic` ou `std_logic_vector`
- Pas agréable pour connecter des composants via un bus un peu complexe

Exemple: Bus AXI



Exemple: Bus AXI-Stream



Exemple: Accès mémoire

Exemple

```
type memory_bus_in_t is record
    addr   : std_logic_vector(15 downto 0);
    data   : std_logic_vector(7 downto 0);
    read   : std_logic;
    write  : std_logic;
end record memory_bus_in_t;

type memory_bus_out_t is record
    data : std_logic_vector(7 downto 0);
end record memory_bus_out_t;
```

Types composites: enregistrements

Exemple

```
entity memory is
port (
    bus_in_i  : in  memory_bus_in_t;
    bus_out_o : out memory_bus_out_t
);
end memory;
```

- Les enregistrements comme ports simplifient la connectique
- Typiquement déclarés dans un paquetage
- Très facile d'ajouter un signal