Electronique numérique

Initiation à VHDL (2/3)

Circuits séquentiels. Automates.

Le TD précédent nous a permis de prendre en main VHDL, à travers des descriptions structurelles (instanciation de composants), ainsi que des équations logiques (assignations concurrentes). Nous poursuivons ici cette découverte du langage : nous allons notamment décrire des bascule D à l'aide du langage, et ainsi construire notre premier circuit séquentiel (un datapath simple). Nous allons également découvrir deux manières de décrire des automates en VHDL :

- En transposant directement les équations logiques des automates.
- Puis en reposant sur les capacités de synthèse du langage à un niveau plus élevé : le niveau RTL.

1 Décrire des bascules D

Nous avons vu précédemment que VHDL nous offre le moyen direct de décrire des équations logiques à l'aide d'assignations concurrentes. La partie droite d'une assignation concurrente peut faire appel à des expressions à deux opérandes (expressions dites "binaires"...) comme le **and**,**or**,**not**,**nor** etc...Ces opérations font partie intégrante du langage : ce sont des mots clés de VHDL. La correspondance avec nos portes logiques est donc immédiate. Malheureusement, ce n'est pas le cas pour la bascule D : il n'existe pas de mot clé pour décrire une telle bascule, de manière directe. Pour décrire les bascules D, il est obligatoire de recourir à des **process**, au sein d'une **architecture**.

```
1 entity a_circuit is
2  port(
3     reset_n : in std_logic;
4     clk : in std_logic;
5     din : in std_logic;
6     qout : out std_logic
7  );
8 end entity;
9
```

```
10 architecture example of a_circuit is
11 begin
12
13   -- ...
14
15   bascule_d: process(reset_n,clk)
16   begin
17     if reset_n='0' then
18       qout <= '0';
19     elsif rising_edge(clk) then
20       qout <= din;
21     end if;
22   end process;
23
24   -- etc
25   -- autres codes...
26
27 end example;
```

L'exemple précédent décrit une (seule) bascule D. On retiendra le gabarit de code qui conduit à créer (inférer est le terme exact) la bascule D. Quelques remarques s'imposent :

- **Toute assignation concurrente sous le contrôle d'un front d'horloge génère une (ou plusieurs) bascule(s) D.**
- Le nommage de l'entrée D et de la sortie Q n'est pas obligatoire : on peut utiliser n'importe quels noms de signaux (pourvu qu'ils soient déclarés). Dans le jargon, on parlera alors de signaux "clockés".
- on peut utiliser n'importe quel *type* de signaux : **std_logic**, **std_logic_vector(63 downto 0)**, **signed(31 downto 0)**, **unsigned(7 downto 0)**, mais également des types créés par l'utilisateur.
- Le caractère prioritaire du reset asynchrone (prioritaire par rapport à l'horloge) est ici manifeste dans le code (if ... elsif...).
- Le front montant se nomme ... **rising_edge**.
- On peut coder plusieurs bascules D, à l'aide d'un seul processus, ou alors créer plusieurs processus. C'est au choix.

2 Descriptions de chemin de données séquentiel

Une relecture de l'énoncé du TD précédent rappelle que les opérateurs arithmétiques traditionnels sont crées automatiquement grâce aux symboles "+","-" etc, dès lors que les données manipulées sont typées en **signed** (par exemple **signed(7 downto 0)**) ou **unsigned(7 downto 0)**.

Questions

1. Coder en VHDL au moins deux architectures numériques qui réalisent un compteur, qui compte de 0 à 255, en respectant l'entité donnée

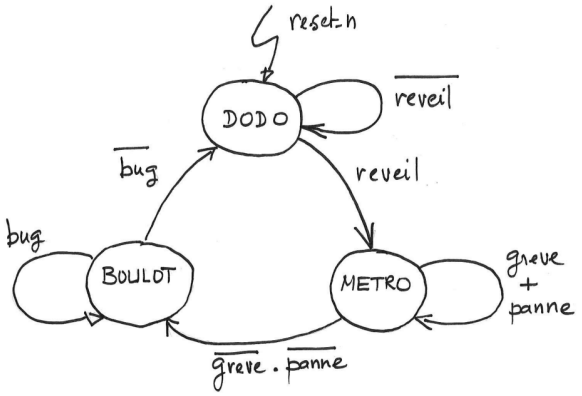


FIGURE 1 – Diagramme états-transitions de l'automate "I love Paris"

ci-dessous. On note que les opérandes sont codés en **unsigned(7 downto 0)** (octets non-signés) ; On prendra soin de dessiner le circuit avant de coder l'architecture.

2. Tester le circuit à l'aide du testbench fourni.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity compteur is
6    port(
7      reset_n : in std_logic;
8      clk : in std_logic;
9      value : out unsigned(7 downto 0)
10   );
11 end compteur;
```

3 Descriptions d'automates au niveau logique

Soit un automate bien connu des informaticiens parisiens, sur la figure 1.

Questions

1. Vérifier que, dans le cas d'un encodage one-hot, les équations d'état suivant sont bien données par :
$$\begin{cases} D_0 &= Q_0.\overline{reveil} + Q_2.\overline{bug} \\ D_1 &= Q_0.reveil + Q_1.(greve + panne) \\ D_2 &= Q_1.\overline{greve}.\overline{panne} + Q_2.bug \end{cases}$$
2. Coder l'automate "I_love_Paris" en VHDL, au niveau logique, en prenant soin de bien coder les bascules D à l'aide de processus. Respectez

l'entité fournie ici. On utilisera une assignation conditionnelle (when) pour coder le signal de sortie "up_and_running".

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity i_love_paris is
6    port(
7      reset_n : in std_logic;
8      clk : in std_logic;
9      reveil : in std_logic;
10     panne : in std_logic;
11     greve : in std_logic;
12     bug : in std_logic;
13     up_and_running : out std_logic
14   );
15 end i_love_paris;
```

3. Utilisez le testbench donné sous Moodle pour tester votre automate. Observez le résultat.

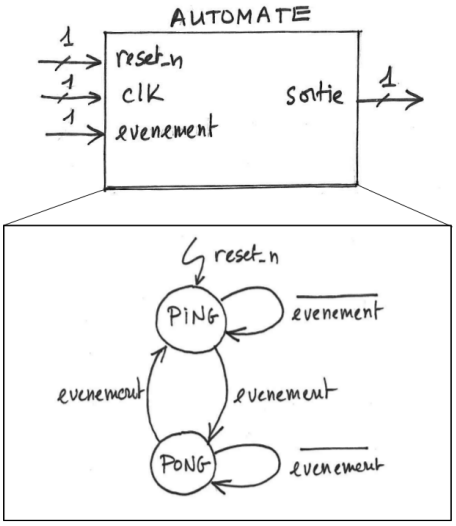
4 Descriptions d'automates au niveau RTL

Ce niveau d'abstraction RTL est le plus couramment utilisé, car il permet de s'affranchir des détails des équations logiques, et décrire des automates (et *micro-architectures*) de manière plus naturelle. Le niveau RTL repose sur la notion d'*inférence matérielle* : l'Electronicien doit connaître certains *motifs de conception*[1], afin de permettre à l'outil de synthèse d'établir automatiquement les équations logiques sous-jacentes. Le cas des automates d'états finis est instructif en ce sens. Voici un exemple de codage VHDL, qui décrit un automate, sans expliciter les équations logiques sous-jacentes, ni l'encodage des états. Pour information, le synthétiseur peut choisir de lui-même un encodage qui maximise la fréquence de fonctionnement du circuit (encodage one-hot généralement), ou tout autre type de contraintes imposées par le concepteur.

Exemple d'automate de niveau RTL Nous donnons ici à titre d'exemple un diagramme états-transitions, ainsi que son code VHDL de niveau RTL. Notez que nous avons séparé le processus ici appelé "next_state_p", qui code la fonction de transition (calcul de l'état suivant), et la sortie de l'automate.

1. "Design patterns", en anglais.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity automate is
6    port(
7      reset_n : in std_logic;
8      clk : in std_logic;
9      evenement : in std_logic;
10     sortie : out std_logic
11   );
12 end automate;
13
14 architecture rtl of automate is
15
16   type state_type is (PING,PONG);
17   signal state, next_state : state_type;
18
19 begin
20   process(reset_n,clk)
21   begin
22     if reset_n='0' then
23       state <= PING;
24     elsif rising_edge(clk) then
25       state <= next_state;
26     end if;
27   end process;
28
29   -- logique d'etat suivant
30   next_state_p: process(state,evenement)
31   begin
32     next_state <= state;--default
33     case state is
34       when PING =>
35        if evenement='1' then
36          next_state <= PONG;
37        end if;
38       when PONG =>
39        if evenement='1' then
40          next_state <= PING;
41        end if;
42       when others =>
43         null;
44     end case;
45   end process;
46
47   -- logique de sortie
48   sortie <='1' when state=PONG else '0';
49
50 end rtl;
```



Questions

1. Prenez le temps de comprendre le codage utilisé ici.
2. Coder l'automate "I_love_Paris" en VHDL, au niveau RTL, en vous inspirant de l'exemple donné.
3. Utilisez le même testbench que précédemment pour tester votre automate. Modifier uniquement le nom de l'architecture associée à l'entité instanciée.