Université de Technologie de Compiègne

MI01

Rapport de TP

2 - VHDL Séquentiel

Automne 2014

Romain PELLERIN - Kyâne PICHOU
Groupe 1
19 octobre 2014

Table des matières

1 Introduction

Le but de ce TP est d'étudier la modélisation en VHDL séquentiel de différents circuits. Nous commencerons par réaliser un simple compteur 2 bits synchrone. Puis nous réaliserons le détecteur de code que nous avons étudié en TD, en détectant les fausses entrées ou non.

En plus de programmer le **FPGA**, nous devrons également réaliser les simulations pour étudier les minimisations et optimisations de synthèse effectuées par le logiciel. En effet, si le code VHDL écrit n'est pas optimal, le logiciel effectuera des optimisations de manière automatique.

2 Exercices

2.1 Exercice 1 - Compteur de 2 bits

Le but de cet exercice est de modéliser un compteur synchrone à 2 bits. Celui-ci utilisera une horloge et un un signal de reset synchrone. Ces deux fonctionnalités seront implémentées respectivement sur **PB_0** et **PB_1** qui sont des boutons poussoirs. L'affichage de sortie se fera sur **LED_10** qui est un ensemble de deux leds.

La modélisation respectera 2 conditions :

— Le front montant de l'horloge est le front actif.

— Le reset est actif au niveau bas (**PB_1 = '1'**)

Ainsi, à chaque pression sur le bouton poissoir **PB_0**, les LEDs afficheront successivement : '00', '01', '10', '11', '00', '01', etc. Si **PB_1** est pressé puis **PB_0**, le compteur sera réinitialisé à 0.

2.1.1 Ports utilisés

Au préalable, nous avons décommenté les lignes du fichier **carte_tp.ucf** correspondant aux ports utilisés :

Fichier : carte_tp.ucf

```
NET "LED_10<0>"   LOC = "P44"    | IOSTANDARD = LVCMOS33;
NET "LED_10<1>"   LOC = "P51"    | IOSTANDARD = LVCMOS33;
NET "PB_0"        LOC = "P128"   | IOSTANDARD = LVCMOS33 | CLOCK_DEDICATED_ROUTE
    = FALSE;
NET "PB_1"        LOC = "P141"   | IOSTANDARD = LVCMOS33 | CLOCK_DEDICATED_ROUTE
    = FALSE;
```

2.1.2 VHDL

Nous proposons une modélisation VHDL de type **comportementale** puisque c'est plus simple à écrire et parfaitement adapté à notre besoin. En effet nous souhaitons utiliser **PROCESS** (instructions séquentielles). Les **signaux déclencheurs** du **PROCESS** seront les deux boutons poussoirs. Les deux LED permettent « d'afficher » 4 valeurs décimales :

— 0 soit 00 en binaire sur 2 bits

— 1 soit 01 en binaire sur 2 bits

— 2 soit 10 en binaire sur 2 bits

— 3 soit 11 en binaire sur 2 bits

Voici l'implémentation :

VHDL

```
entity tp2_1 is
  PORT(LED_10 : out integer range 0 to 3; -- la led peut afficher 4 valeurs sur
     2 bits
       PB_0 : in bit;
```

```
        PB_1 : in bit);
end tp2_1;

architecture Behavioral of tp2_1 is
begin
        process(PB_0,PB_1)
        variable i : integer range 0 to 3 := 0; -- variable qui sert d'alias du
            signal
        begin
                if (PB_1 = '1') then
                        i := 0; -- dès que PB_1 est enfoncé, on reset
                elsif(PB_0'event AND PB_0='1') then
                        i := i + 1; -- on incrémente àchaque pression de PB_0
                end if;
                LED_10 <= i; -- on affecte la valeur de la variable au signal
        end process;
end Behavioral;
```

2.1.3 Synthèse

On synthétise notre circuit :

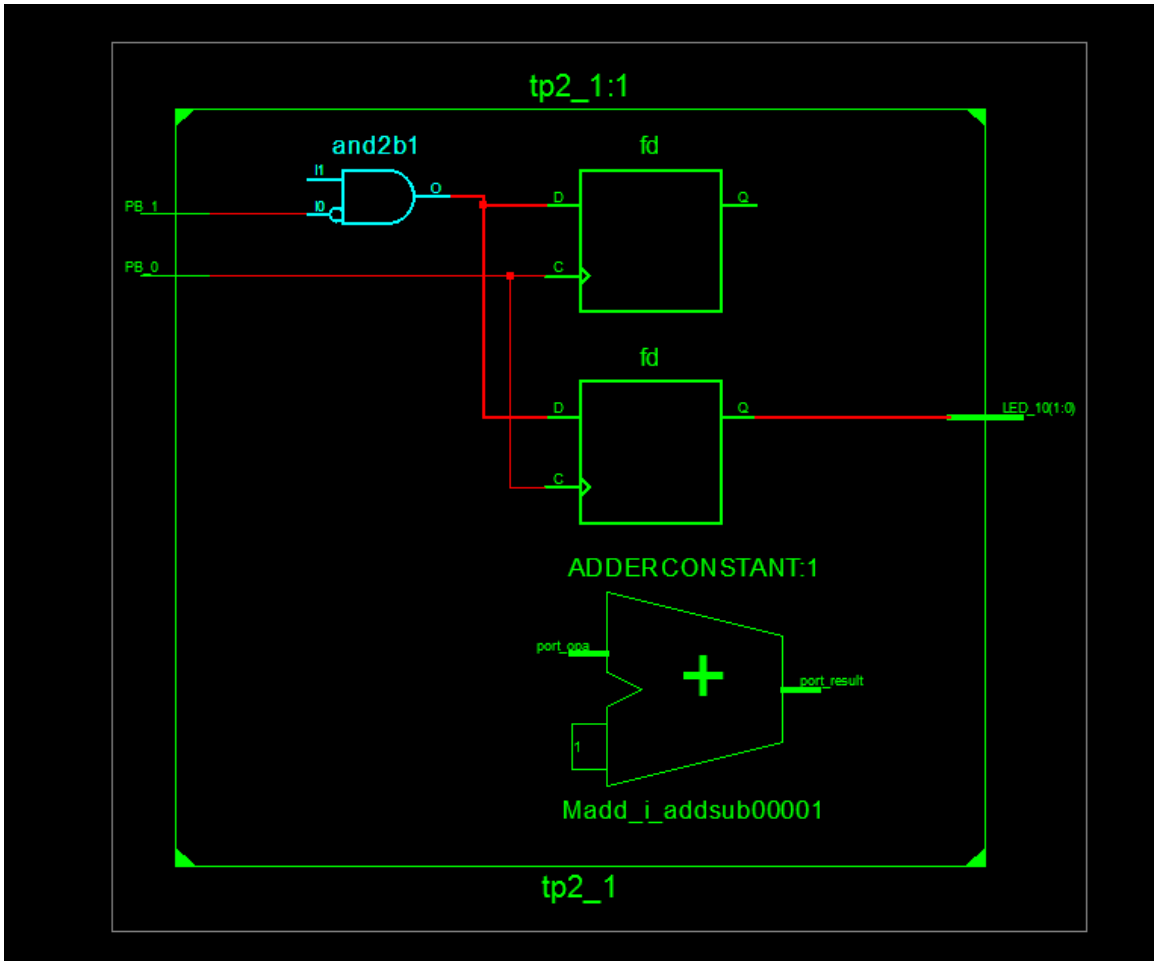

FIGURE 2.1 – Schéma « RTL »

On constate que le synthétiseur a utilisé un circuit additionneur avec peu de portes. C'est

un circuit « pré-optimisé », indépendant du FPGA cible. C'est une représentation logique du VHDL.

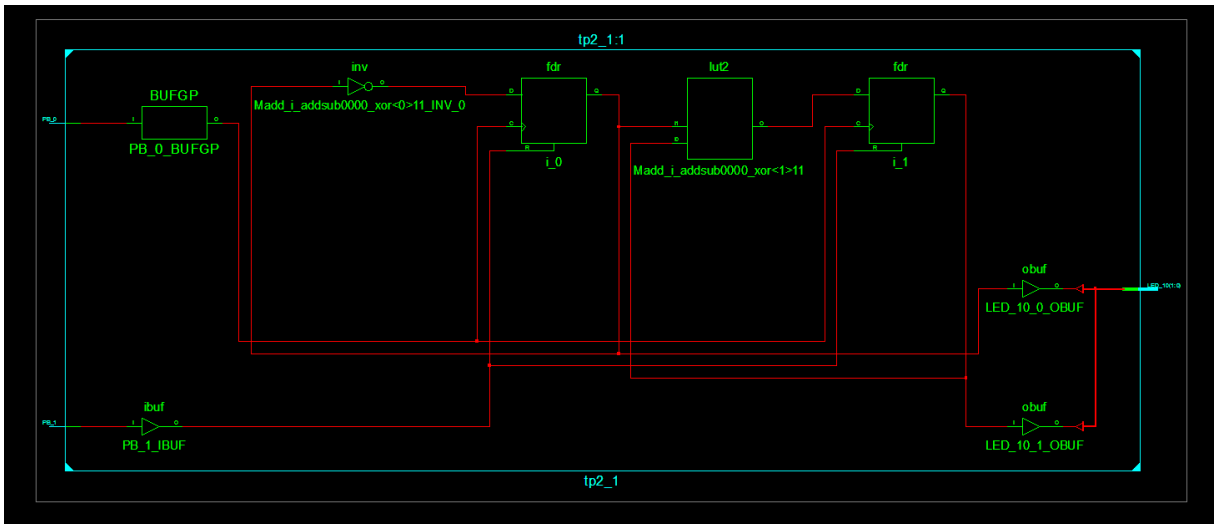

FIGURE 2.2 – Schéma « Technology »

Le schéma « Technology » présente plus d'éléments que le schéma précédent (*inv*, *fdr*, etc) et est par conséquent beaucoup plus complexe. Ce schéma est généré après l'optimisation, qui est faite en fonction du FPGA cible. Les composants sont choisis selon ce FPGA cible. C'est une représentation technologique du VHDL optimisé.

2.1.4 Simulation

Après la synthèse, on simule notre circuit pour vérifier qu'il respecte le cahier des charges :

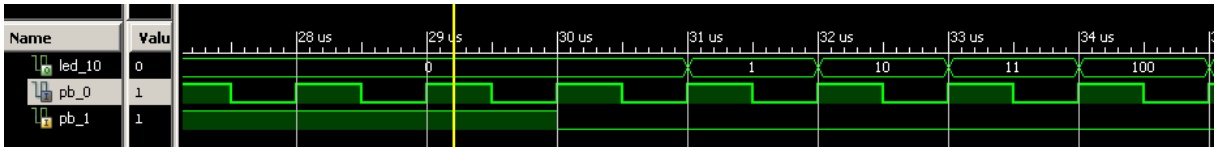

FIGURE 2.3 – Simulation

La simulation est correcte, nous avons bien un compteur incrémenté suivant l'horloge, qui est réinitialisé par le signal de reset (de façon synchrone avec le signal horloge [1]). En effet, au début du schéma, on constate que tant que **PB_1** est « enfoncé » (équivalent à 1), la valeur des LED est 0. Puis, lorsque le bouton poussoir est relâché (à partir de 30μs), la valeur affichée par les LED s'incrémente (01, 10, 11, 00, 01, etc).

Pour terminer on programme le FPGA avec notre code pour tester en pratique le bon fonctionnement. Après tests, notre programme est fonctionnel. On constatera que le reset est ici à nouveau synchrone.

1. Pour reset notre compteur, il faut que le bouton reset soit actif (**PB_1 = '1'**) lors du front d'horloge. En pratique, l'horloge est simulée à l'aide d'un bouton poussoir, de ce fait il fait garder le reset enfoncé (**PB_1**) puis appuyer sur le bouton d'horloge (**PB_0**)

2.2 Exercice 2 - Détecteur de code

Dans cet exercice nous alons modéliser le détecteur de code étudié en TD. Celui-ci lit sur une ligne de transmission série (de 1 bit) par une technique séquentielle synchrone. Un signal **Alarme** est mis à 1 dès que le code "11010" est détecté. On utilise différents signaux :

— BP_0 pour l'horloge (bouton poussoir)

— SW_0 pour la ligne de transmission série (switch)

— LED_0 pour le signal Alarme

2.2.1 Ports utilisés

Au préalable, nous avons décommenté les lignes du fichier **carte_tp.ucf** correspondant aux ports utilisés :

Fichier : carte_tp.ucf

```
NET "LED_0" LOC = "P44" | IOSTANDARD = LVCMOS33;
NET "SW_0" LOC = "P130" | IOSTANDARD = LVCMOS33;
NET "PB_0" LOC = "P128" | IOSTANDARD = LVCMOS33 | CLOCK_DEDICATED_ROUTE = FALSE;
```

2.2.2 VHDL

On propose la modélisation VHDL **comportementale** ci-dessous. Chaque fausse entrée nécessite de re-entrer l'intégralité du code. Il s'agit finalement de modéliser une **machine à états**.

VHDL

```
entity tp2_2 is
  port(PB_0,SW_0 : in bit;
       LED_0 : out bit);
end tp2_2;

architecture Behavioral of tp2_2 is
begin
       process(PB_0) -- Synchronisé sur chaque pression du bouton poussoir
       variable nbt : integer range 0 to 5; -- Un état pour chaque élément du
           code détecté + l'état où rien n'est détecté
       variable allume : bit := '0'; -- alias du signal de la LED
       begin
       if(PB_0'event and PB_0='1') then
              case nbt is
                     -- A chaque pression du bouton poussoir, on vérifie que l'
                        état du switch (1 ou 0) correspond a l'élément du code
                        attendu (selon l'état actuel de la machine), sinon on
                        retombe dans l'état 0 (aucun élément du code détecté)
                     when 0 =>
                            if(SW_0='1') then
                                   nbt:=1;
                            end if;
                     when 1 =>
                            if(SW_0='1') then
                                   nbt:=2;
                            else
```

```
                                nbt:=0;
                        end if;
                when 2 =>
                        if(SW_0='0') then
                                nbt:=3;
                        else
                                nbt:=0;
                        end if;
                when 3 =>
                        if (SW_0 ='1') then
                                nbt:=4;
                        else
                                nbt:=0;
                        end if;
                when 4 =>
                        nbt:=0;
                        if(SW_0='0') then -- Le code entier a été détecté
                                nbt:=5;
                                allume := '1';
                        else
                                nbt:=0;
                        end if;
                when 5 => -- Une fois le code détecté (état = 5), on ré
                    initialise tout peu importe l'état du switch
                        nbt:=0;
                        allume:='0';
                end case;
                LED_0<=allume; -- Affectation réelle
        end if;
    end process;
end Behavioral;
```

2.2.3 Synthèse

On remarque que notre état (nbt) est codé sur 5 bits. Pourtant dans notre code VHDL, c'est un entier de 0 à 5, 3 bits auraient donc, à priori, été suffisants. Au lieu d'utiliser le codage d'un entier en binaire, le synthétiseur utilise 5 bits qui activent où non un multiplexeur. Chaque bit correspond à un état.

2.2.4 Simulations

On simule notre circuit synthétisé dans ISim pour constater son bon fonctionnement.

On fait évoluer l'entrée d'horloge à une fréquence donnée et on force les valeurs correspondant à notre code sur l'entrée ligne. On constate bien que, dès que le code est détecté, la LED s'allume (passage à 1). Elle repasse à 0 au coup d'horloge suivant, prête à détecter à nouveau le code.

On effectue une seconde simulation pour **vérifier que lorsqu'il y a une mauvaise entrée, il y a bien réinitialisation de la machine à état**. Pour cela on compose le code avec une erreur au milieu. Si le code n'est pas détecté, c'est que l'erreur a bien été prise en compte :

Ici, le code est correct à l'exception du 3ème '1' (à 2 μs) qui vient se glisser pour créer une erreur. On remarque que le code n'est donc pas détecté.

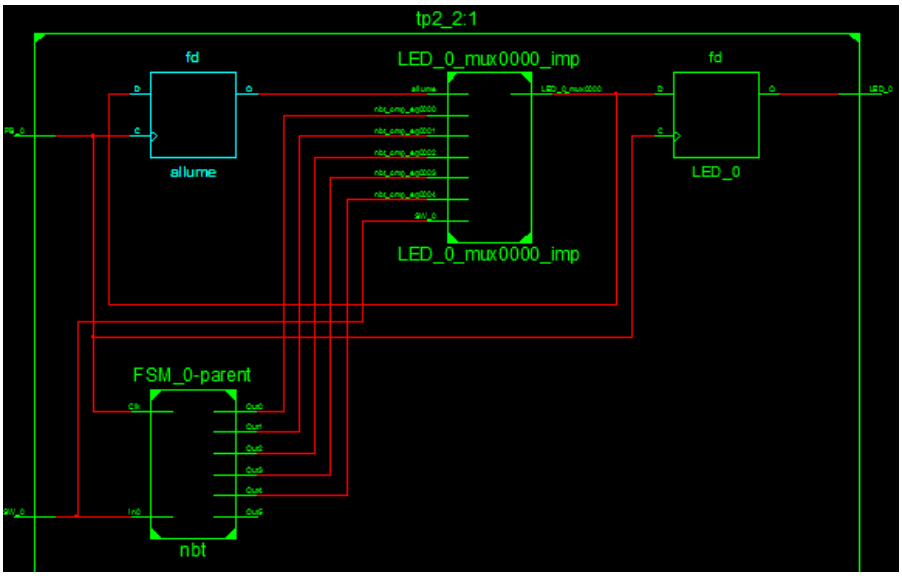

FIGURE 2.4 – Schéma RTL

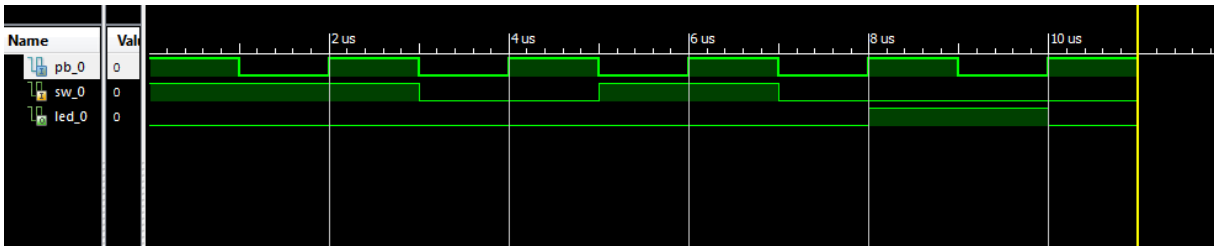

FIGURE 2.5 – Simulation

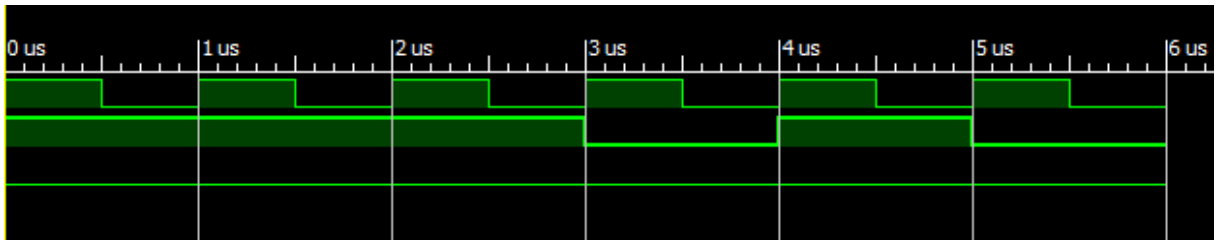

FIGURE 2.6 – Simulation

Après programmation du FPGA, on constate le bon fonctionnement pratique de notre système.

2.3 Exercice 3 - Détecteur de code - Fausse entrée négligée

Il s'agit de répéter l'exercice 2 (section 2.2) mais en négligeant les fausses entrées.

2.3.1 VHDL

On propose la modélisation VHDL **comportementale** ci-dessous. Chaque fausse entrée est négligée. Il s'agit à nouveau de modéliser une **machine à états**.

VHDL

```
entity tp2_3 is
  port(PB_0,SW_0 : in bit;
       LED_0 : out bit);
end tp2_3;

architecture Behavioral of tp2_3 is
begin
  process(PB_0)
  variable nbt : integer range 0 to 5;
  variable allume : bit := '0';
  begin
  if(PB_0'event and PB_0='1') then
    case nbt is
      when 0 =>
        if(SW_0='1') then
          nbt:=1;
        end if;
      when 1 =>
        if(SW_0='1') then
          nbt:=2;
        end if;
      when 2 =>
        if(SW_0='0') then
          nbt:=3;
        end if;
      when 3 =>
        if (SW_0 ='1') then
          nbt:=4;
        end if;
      when 4 =>
        nbt:=0;
        if(SW_0='0') then
          nbt:=5;
          allume := '1';
        end if;
      when 5 =>
        nbt:=0;
        allume:='0';
      end case;
      LED_0<=allume;
    end if;
  end process;
```

```
end Behavioral;
```

Le code proposé ci-dessus permet d'ignorer les fausses entrées. Lorsqu'il y a une erreur elle n'est pas prise en compte et la machine à états reste inchangée.

2.3.2 Synthèse

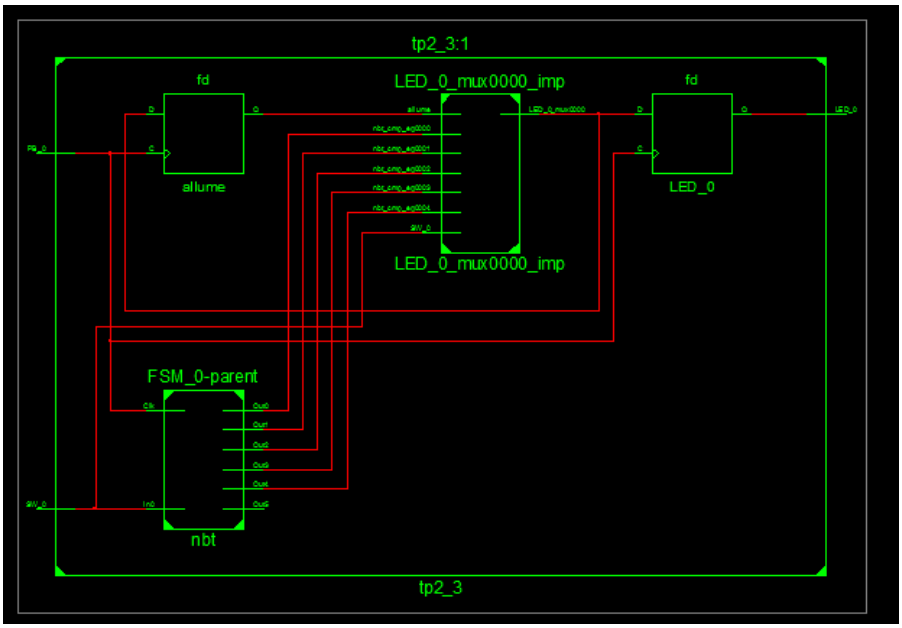

FIGURE 2.7 – Schéma RTL

Le nombre de bits d'état est étrangement le même que dans l'exercice 2.

2.3.3 Différences par rapport à l'exercice 2

On simule notre code à la même manière que dans l'exercice 2. On entre le bon code mais on glisse une erreur au milieu. Cette fois-ci l'erreur ne devrait pas être prise en compte et le code devrait être détecté :

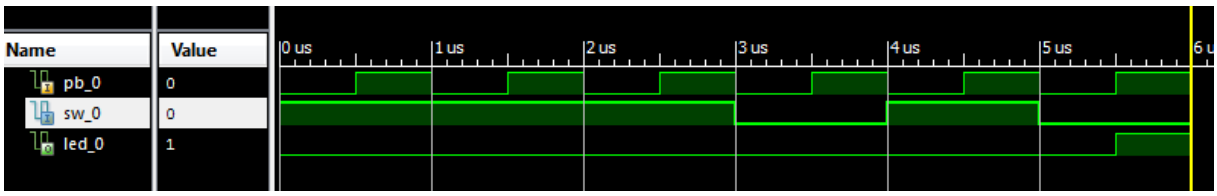

FIGURE 2.8 – Simulation

Notre simulation est concluante. Le troisième '1' était une erreur mais le code a tout de même été détecté. La fausse entrée a donc été négligée.