Electronique numérique

Décomposition FSMD et Synthèse d'un système numérique

1 But du TE

Le but de ce TE est de "synthétiser" un circuit numérique sur un FPGA. C'est la première expérience concrète que l'on réalise dans l'UV 1.5 !

Nous allons pour cela procéder de la même manière qu'aux TE précédents, en adjoignant une nouvelle étape dans le processus :

- Formuler le problème sous forme de **diagramme à bulles**.
- Etablir les équations logique de notre **contrôleur** [1]
- **Simuler** le circuit virtuel pour s'assurer qu'il fonctionne comme prévu.
- **Synthétiser** le circuit sur FPGA, et le faire fonctionner "pour de vrai".

2 Position du problème : multiplieur séquentiel

Nous savons qu'il est *certes* possible de trouver un circuit purement *combinatoire* qui réalise la multiplication...Nous n'allons pas procéder ainsi.

Le multiplieur proposé ici *séquentiel* : il lui faudra plusieurs cycles pour réaliser son calcul. L'algorithme de multiplication séquentielle reproduit ce que l'on fait sur papier, lorsqu'on multiplie 2 nombres.

Pour multiplier X par Y, la methode consiste à cumuler séquentiellement les produits partiels :

$$P = X \times Y = \sum_i Y_i \times X \times 2^i$$

Il est très important de comprendre la signification de cette équation : selon la valeur de Y_i, le terme $(Y_i \times X) \times 2^i$ de rang i de la somme vaut :

1. (ou "FSM" ou "automate",...)

— 0 si Yi = 0 ou
— X décalé de i positions vers la gauche si $Yi = 1$. Le décallage se fait par 2^i !

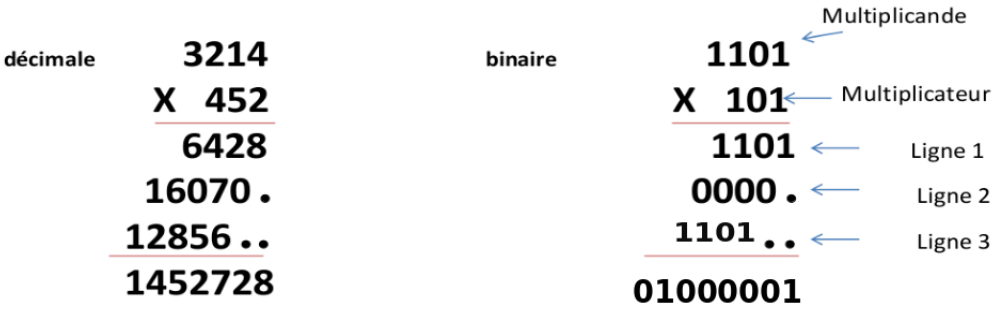

FIGURE 1 – Principe de la multiplication : une séquence de produits partiels et de décalages

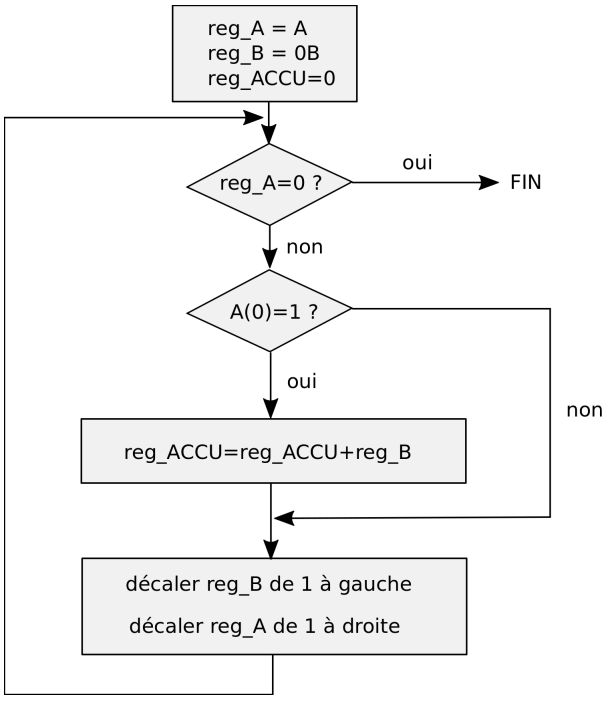

FIGURE 2 – Flowchart de l'algorithme de multiplication séquentielle

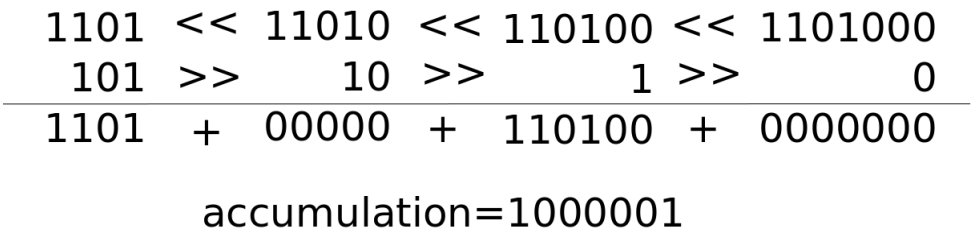

FIGURE 3 – Flowchart de l'algorithme de multiplication séquentielle

3 Décomposition FSMD

3.1 Rappels

La décomposition FSMD (ou contrôleur-chemin de données) est rappelée ici, de manière générique. Le contrôleur réagit aux status du chemin de données, et contrôle ce dernier en conséquence. Le contrôleur réagit également aux ordres extérieurs (dans notre cas un 'start'), en renseigne l'extérieur sur la disponibilité des données produites (dans notre cas un signal 'end' indiquant la disponibilité du calculateur pour un nouveau calcul).

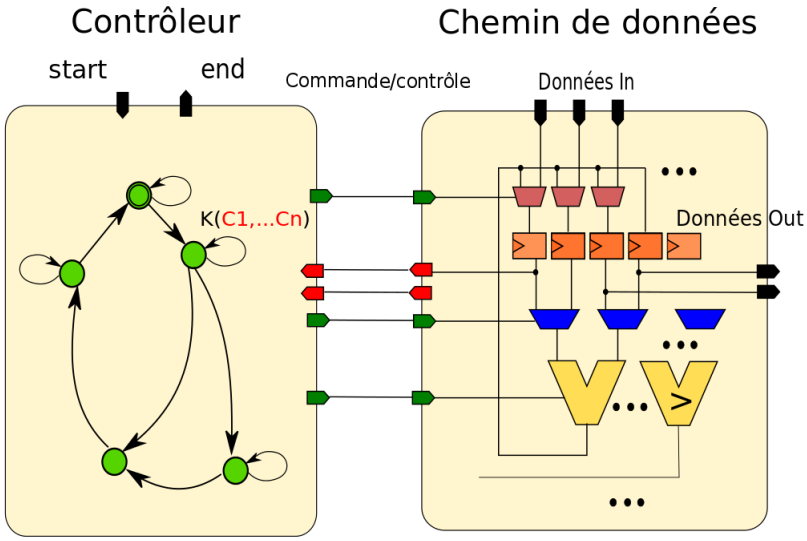

FIGURE 4 – Chemin de données générique

3.2 Chemin de données

Les principes d'additions et de décallages successifs nous invitent à considérer 3 registres : regA, regB et ACCU qui contiendront les valeurs de A,B et de l'addition courante.

Question 1 : donner les différentes valeurs binaires (et décimales) des registres regA, regB et ACCU au cours de la multiplication pour $A = 5$ et $B = 13$.

Corrections : Cela correspond au schéma donné !

regA	regB	ACCU	cycle
0101	00001101	00000000	1
0010	00011010	00001101	2
0001	00110100	00001101	3
0000	01101000	01000001	4

Question 2 : trouver une condition d'arrêt de cet algorithme. Cette condition constituera un élement du *status* renvoyé au contrôleur.

Corrections : lorsqu'il n'existe plus aucun bit à 1 dans reg_A. Pour cela il suffit de constituer le signal :

$$stop = \overline{(reg\_A(3) + reg\_A(2) + reg\_A(1) + reg\_A(0))}$$

Le schéma suivant recense ces opérateurs, les registres ainsi que les routages (multiplexeurs) entre ces élements.

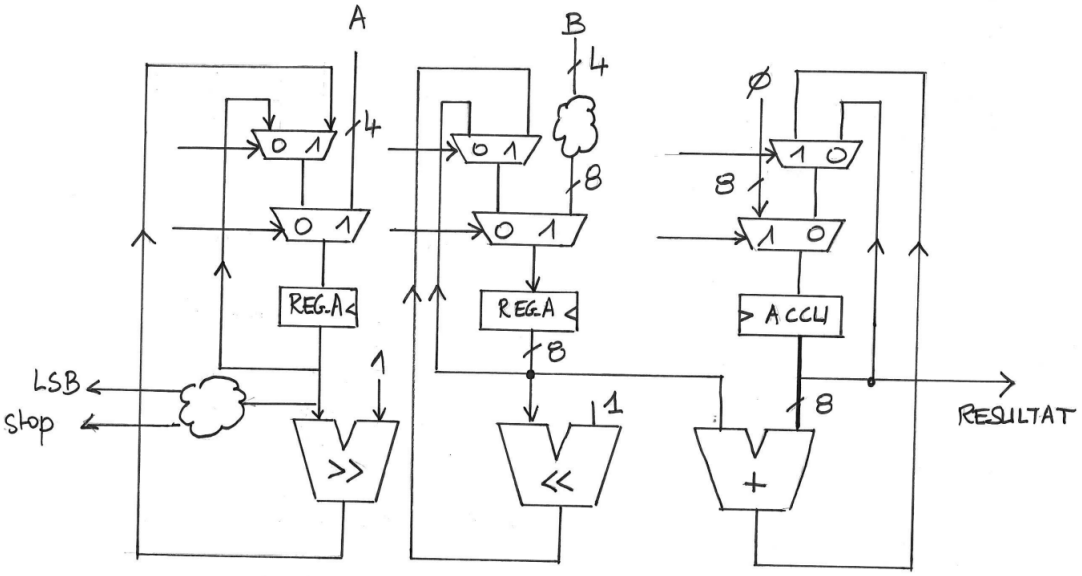

FIGURE 5 – Chemin de données de l'algorithme

Question 3 : tenter de donner un nom significatif au signaux de contrôle des différents multiplexeurs.

Corrections : les multiplexeurs le plus proches des entrées des registres servent à charger la donnée venant de l'extérieur, les autres servent à shifter ou additionner. **Le contrôleur génère ces signaux**.

En respectant la position "géographique" des signaux sur le schéma, voici un nommage possible :

shift	shift	add
chA	chB	init

3.3 Codage VHDL du chemin de données

Sous Moodle on donne un code "à trous" du chemin de données. Le codage VHDL du registre regA est rappelé ici également.

Question 4 : analyser ce code en tentant de faire la correspondance avec le schéma du datapath. En particulier, où se trouve le signal "A_Reg_Comb" ?

Solution : ici, il faut juste tenter de comprendre l'assignation du signal A_Reg_comb en VHDL : il s'agit d'une assignation conditionnelle, utilisant le mot clé "when" (suivi de la condition, précédée du signal ou valeur à affecter). La lecture de l'anglais doit suffire à comprendre cette assignation. Il faut comprendre que cette affectation conditionnelle correspond à un ensemble de multiplexeurs, enchaînés les uns aux autres, afin de router plusieurs donnéés ou valeurs potentielles vers l'entrée du registre (bascule D). Une écriture spécifique VHDL engendrera les circuits correspondants : c'est **l'inférence matérielle**. Le signal "A_Reg_Comb" est donc situé en entrée de la bascule/registre reg_A.

Question 5 : en vous inspirant de ce code, écrire regB, puis celui d'ACCU. Pour ce dernier, vous aurez besoin d'additionner. Le type *unsigned* utilisé ici pour typer les registres permet effectivement d'additionner (ce n'est logiquement pas le cas de vecteurs de bits).

Listing 1 – registre A et logique combinatoire associée

```
1  A_reg_proc : process (clk, reset_n)
2    begin
3      if reset_n = '0' then
4        A_Reg <= "0000";
5      elsif rising_edge(clk) then
6        A_Reg <= A_Reg_comb;
7      end if;
8    end process;
9
10   A_Reg_comb <= unsigned(a) when chRA = '1' else
11                 '0' & reg_a(3 downto 1) when shift = '1' else
12                 reg_a;
```

solution :

Listing 2 – registre B

```
1  ----------------------------------------------------------------
2  --datatpath
3  ----------------------------------------------------------------
4  B_reg_proc : process (clk, reset_n)
5  begin
```

```
6       if reset_n = '0' then
7         reg_b <= "00000000";
8       elsif rising_edge(clk) then
9         reg_b <= reg_b_comb;
10      end if;
11    end process;
12
13    reg_b_comb <= unsigned("0000" & b) when chRb = '1' else
14                  reg_b(6 downto 0) & '0' when shift = '1' else
15                  reg_b;
16    ------------------------------------------------------------------------
17    a_reg_proc : process (clk, reset_n)
18    begin
19      if reset_n = '0' then
20        reg_a <= "0000";
21      elsif rising_edge(clk) then
22        reg_a <= reg_a_comb;
23      end if;
24    end process;
25
26    reg_a_comb <= unsigned(a) when chRa = '1' else
27                  '0' & reg_a(3 downto 1) when shift = '1' else
28                  reg_a;
29    ------------------------------------------------------------------------
30    ACCU : process (clk, reset_n)
31    begin
32      if reset_n = '0' then
33        accu_reg <= unsigned(''00000000'');
34      elsif rising_edge(clk) then
35        accu_reg <= accu_reg_comb;
36      end if;
37    end process;
38
39    accu_reg_comb <= unsigned(''00000000'') when init = '1' else
40                     accu_reg + reg_b when add = '1' else
41                     accu_reg;
42    ------------------------------------------------------------------------
43    res <= std_logic_vector(accu_reg);
44
45    lsb_a <= reg_a(0);
46    stop <= not(reg_a(3) or reg_a(2) or reg_a(1) or reg_a(0));
```

3.4 Conception de l'automate

Le diagramme à bulle de l'automate est donné ici. Les premiers états servent à attendre la fourniture des opérandes A et B : la présence de ces données est signalée respectivement par des signaux externes ack_A et ack_B.

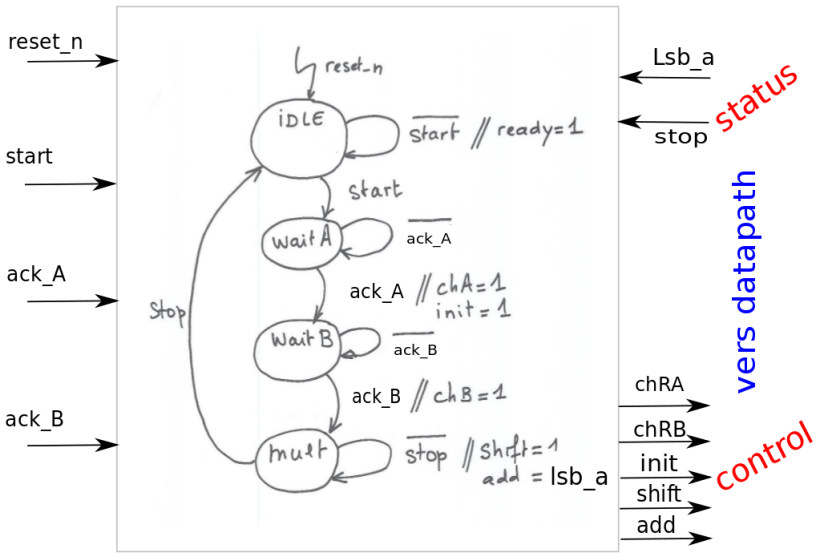

FIGURE 6 – Chemin de données de l'algorithme

Question 6 : établir les équations de cet automate afin de contrôler le chemin de données précédent.

solutions : en choisissant un encodage one-hot.

Version "à la main", avec nom conventionnel des entrées-sorties de bascules D :

```
D0 = Q3.STOP  + Q0./START
D1 = Q0.START + Q1./ACK_A
D2 = Q1.ACK_A + Q2./ACK_B
D3 = Q2.ACK_B + Q3./STOP

SHIFT= Q3
ADD  = Q3.LSB_A
CHRA  = INIT= Q1.ACK_A
CHRB  = Q2.ACK_B
READY= Q0
```

Question 7 : remplir le code à trou concernant ce contrôleur, dans le même fichier, à l'endroit prévu pour cela. En ce qui concerne le codage VHDL, inspirez vous du TE3.

solution :

```
1   ------------------------------------------------------------------------
2   -- Controleur
3   ------------------------------------------------------------------------
4   bascules_etat : process (clk, reset_n)
5   begin
6     if reset_n = '0' then
7       state <= IDLE;
8     elsif rising_edge(clk) then
9       state <= next_state;
10    end if;
11  end process;
12
13  -- logique combinatoire d'etat suivant
14  next_state(0) <= (state(3) and stop) or (state(0) and not(start));
15  next_state(1) <= (state(0) and start) or (state(1) and not(ack_a));
16  next_state(2) <= (state(1) and ack_a) or (state(2) and not(ack_b));
17  next_state(3) <= (state(2) and ack_b) or (state(3) and not(stop));
18
19  -- signaux controleur --> datapath
20  shift <= state(3);
21  add <= state(3) and lsb_a;
22  chrA <= state(1) and ack_a;
23  init <= state(1) and ack_a;
24  chrB <= state(2) and ack_b;
25  -- signal vers l'exterieur (fin traitement)
26  ready <= state(0);
```

4 Simulation

Un banc de test vous est fourni sous Moodle, qui teste le produit de 5 par 15.

Question 8 : avec GHDL, analyser puis simuler votre circuit.

solution : rappels. Tout ceci peut être exécuté en ligne de commande ou mis dans un fichier script sous Linux, afin d'automatiser l'exécution.

```
ghdl -a multSeq.vhd
ghdl -a multSeq_tb.vhd
ghdl -e multSeq_tb
ghdl -r multSeq_tb --wave=multseq.ghw
gtkwave multseq.ghw
```

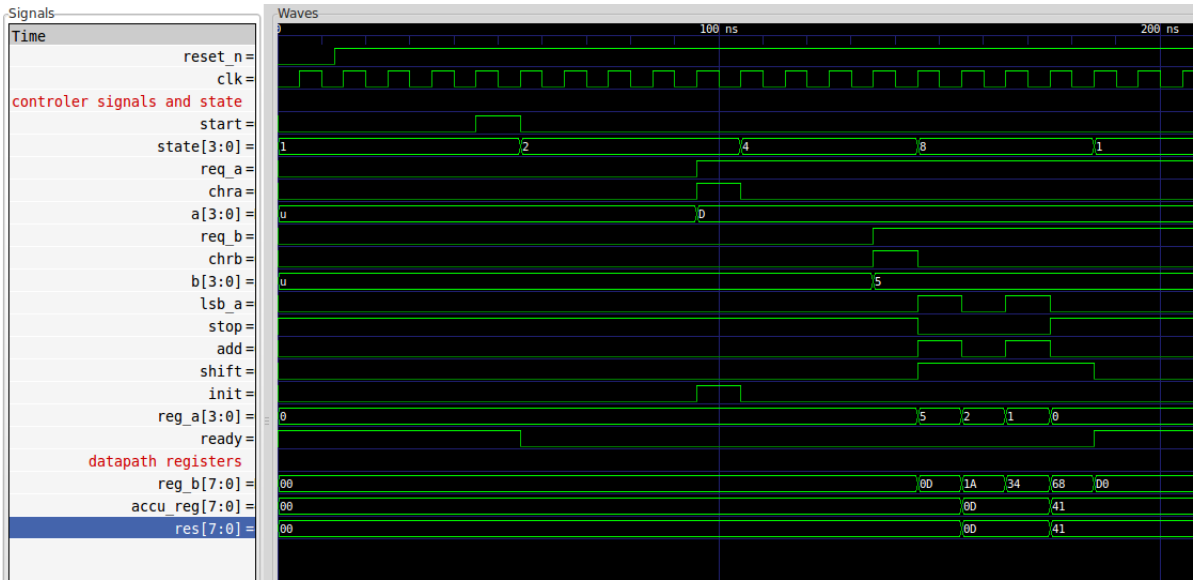

FIGURE 7 – Forme d'onde (waveforms) dans le cas de $A = 5_{10}$ et $B = 13_{10}$

5 Synthèse FPGA

Lorsque vous avez simulé correctement, consulter votre enseignant pour observer le circuit sur plateforme FPGA Cyclone II de la société Altera, sur carte DE2.