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 !

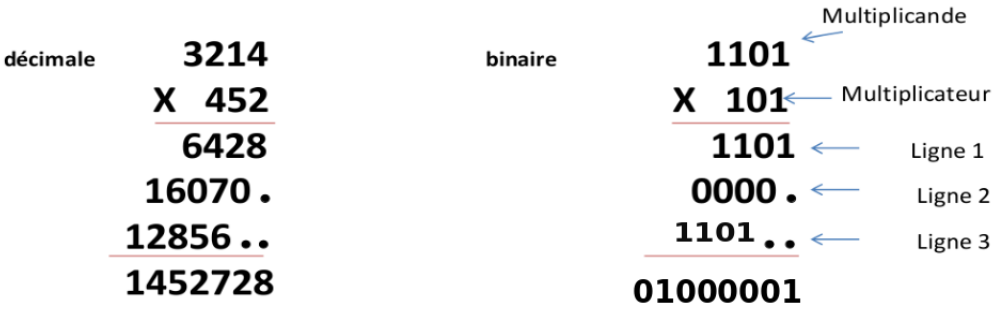

FIGURE 1 – Principe de la multiplication : une séquence de produits partiels et de décalages

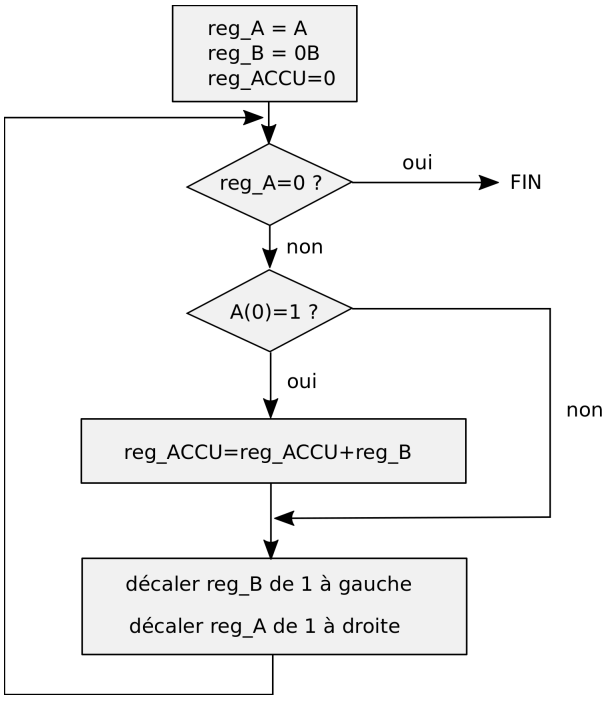

FIGURE 2 – Flowchart de l'algorithme de multiplication séquentielle

1101	<<	11010	<<	110100	<<	1101000	
101	>>	10	>>	1	>>	0	
1101	+	00000	+	110100	+	0000000	

accumulation=1000001

FIGURE 3 – Déroulement 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).

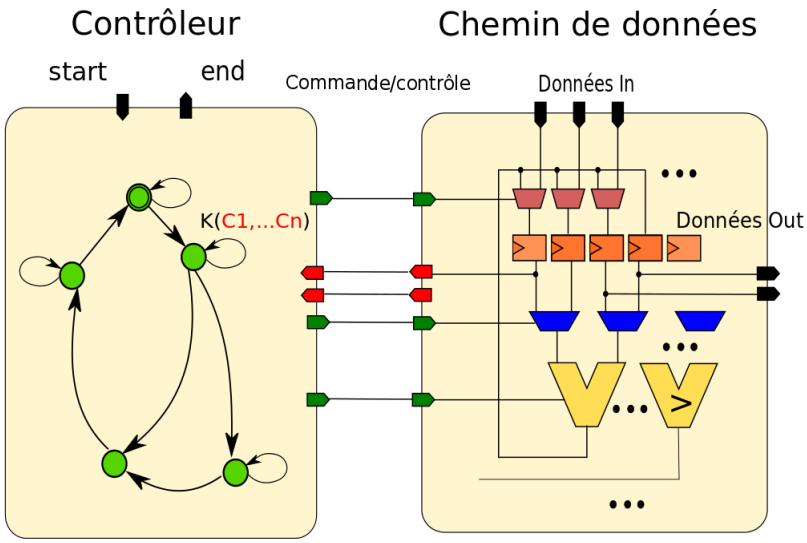

FIGURE 4 – Chemin de données générique

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 $B = 13$.

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

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

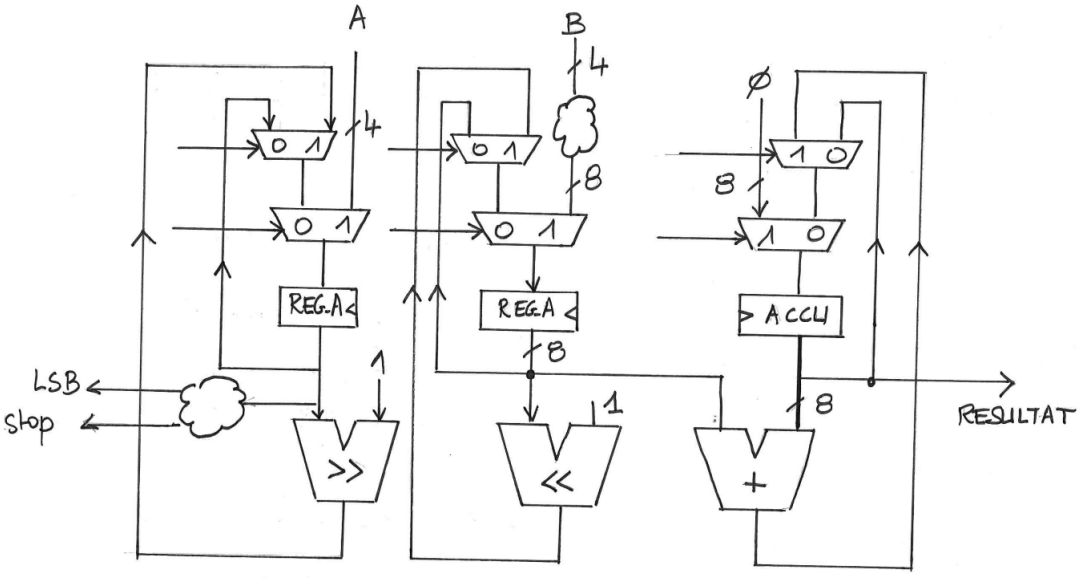

FIGURE 5 – Chemin de données de l'algorithme

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

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" ?

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       reg_a <= "0000";
5     elsif rising_edge(clk) then
6       reg_a <= reg_a_comb;
7     end if;
8   end process;
9
10  reg_a_comb <= unsigned(a) when chRa = '1' else
11                '0' & reg_a(3 downto 1) when shift = '1' else
12                reg_a;
```

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.

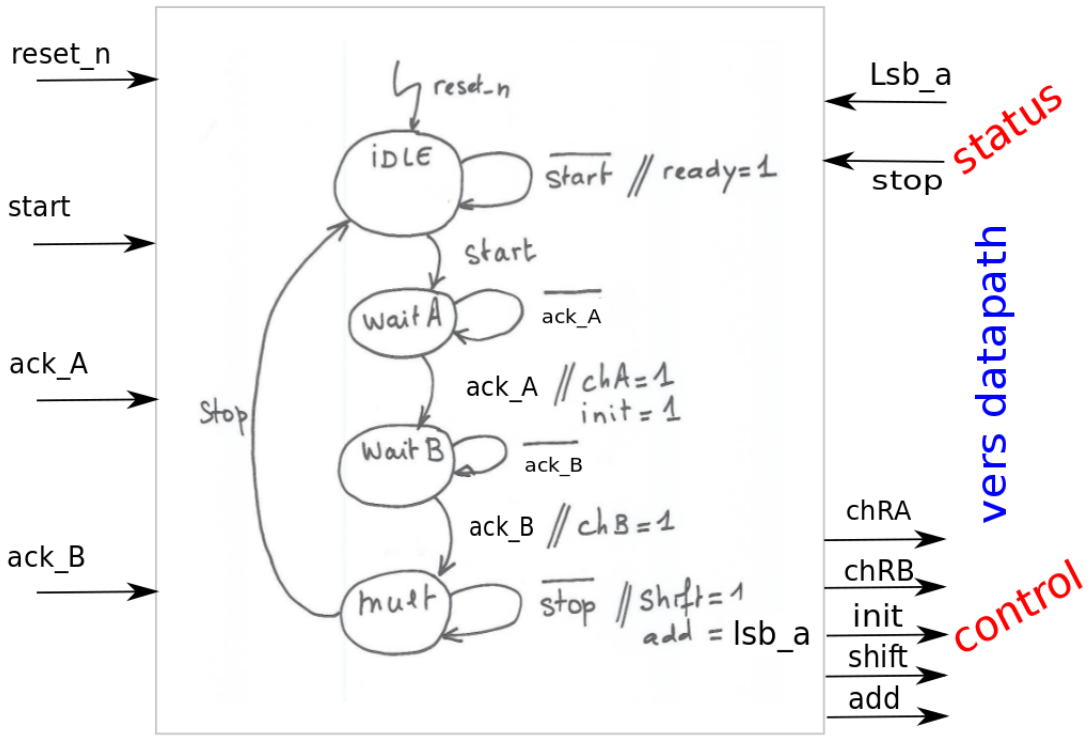

FIGURE 6 – Contrôleur de l'algorithme

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

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.

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.

5 Synthèse FPGA

Lorsque vous avez simulé correctement, consultez votre enseignant pour observer le circuit synthétisé sur un FPGA (FPGA de la société Xilinx, sur carte Nexys 2 de la société Digilent).