INF3500 : Conception et réalisation de systèmes numériques

Examen final

Lundi 27 avril 2020

Durée: 4h.
Pondération: 50%.
Documentation: Toute documentation permise.
Directives particulières:

- Répondre à toutes les questions, la valeur de chaque question est indiquée.
- Ne communiquez pas avec personne durant l'examen.
- Ne posez pas de question durant l'examen. En cas de doute sur le sens d'une question, énoncez clairement toute supposition que vous faites.
- Après avoir terminé l'examen, téléversez vers Moodle deux fichiers :
  - un fichier zip incluant les codes sources demandés; et,
  - un fichier pdf comportant votre nom, votre matricule, l'énoncé sur l'honneur et vos réponses.

Question 0. (0 points, mais obligatoire)

Dans votre fichier de réponse, ajoutez les informations et la mention suivantes :

Mon nom de famille est : _____

Mon prénom est : _____

Mon matricule est : _____

J'affirme, sur mon honneur, avoir fait cet examen tout/e seul/e sans l'aide d'une autre personne.

Question 1. (4 points)

On désire implémenter le code VHDL suivant en utilisant la PROM de la Figure 1. Avant de ce faire, on vous demande de complétez la ligne ci-après avec votre identifiant personnel. Par exemple, si votre matricule est 1625143, la constante `MON_MATRICULE` vaudra 1625143 (un million six cent vingt-cinq mille cent quarante-trois).

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

entity q1 is
port ( A : in  signed(3 downto 0);
       F : out signed(3 downto 0));
end q1;

architecture arch of q1 is

   constant MON_MATRICULE : positive := 1625143; <= MATRICULE DE L'EXEMPLE

begin
   process( A ) is
      variable k : natural;
   begin
      k := to_integer(A);

      if ( ( ( MON_MATRICULE + k + 1 ) mod 3 ) = 0 ) then

         F <= to_signed(k/2, 4);

      elsif ( ( ( MON_MATRICULE + k + 2 ) mod 5 ) = 0 ) then

         if( k >= 0 ) then
            F <= to_signed( k , 4);
         else
            F <= to_signed( k + 7 , 4);
         end if;

      else

         if( k >= 0 ) then
            F <= to_signed( k/2 , 4);
         else
            F <= to_signed( k + 7 , 4);
         end if;

      end if;
   end process;
end arch;
```

Note : chaque copie a une réponse différente pour laquelle une solution est générée par logiciel. La solution donnée ici correspond à celle obtenue avec le matricule d'exemple.

a) **(2 points)** Identifiez clairement le nom des signaux d'entrée et de sortie.

Connecté à A_3 : `A(3)`	Connecté à D_5 : `-`
Connecté à A_2 : `A(2)`	Connecté à D_4 : `-`
Connecté à A_1 : `A(1)`	Connecté à D_3 : `F(3)`
Connecté à A_0 : `A(0)`	Connecté à D_2 : `F(2)`
Connecté à D_7 : `-`	Connecté à D_1 : `F(1)`
Connecté à D_6 : `-`	Connecté à D_0 : `F(0)`

b) **(2 points)** Implémentez le code VHDL à l'aide de la PROM de la Figure 1 :

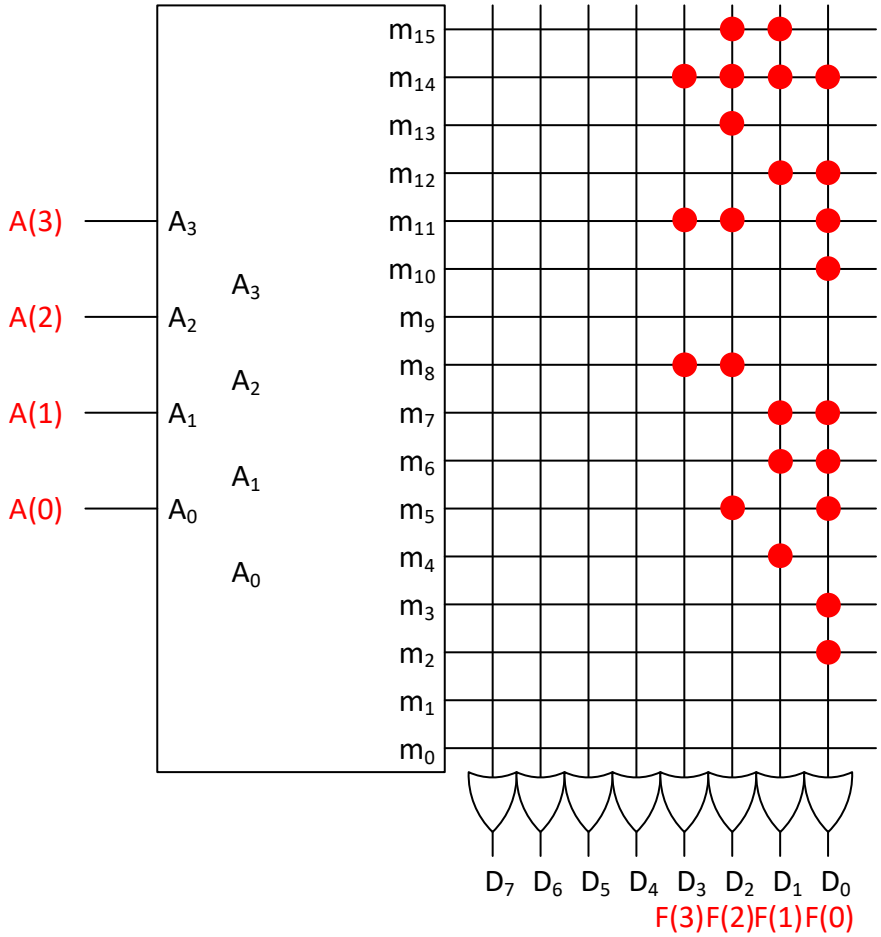


Figure 1 PROM.

Question 2. (4 points)

Nous désirons modifier l'entité `intra2_q1` du contrôle périodique 2. L'entité est décrite dans le fichier `intra2_q1.vhd` et elle est illustrée à la Figure 2 :

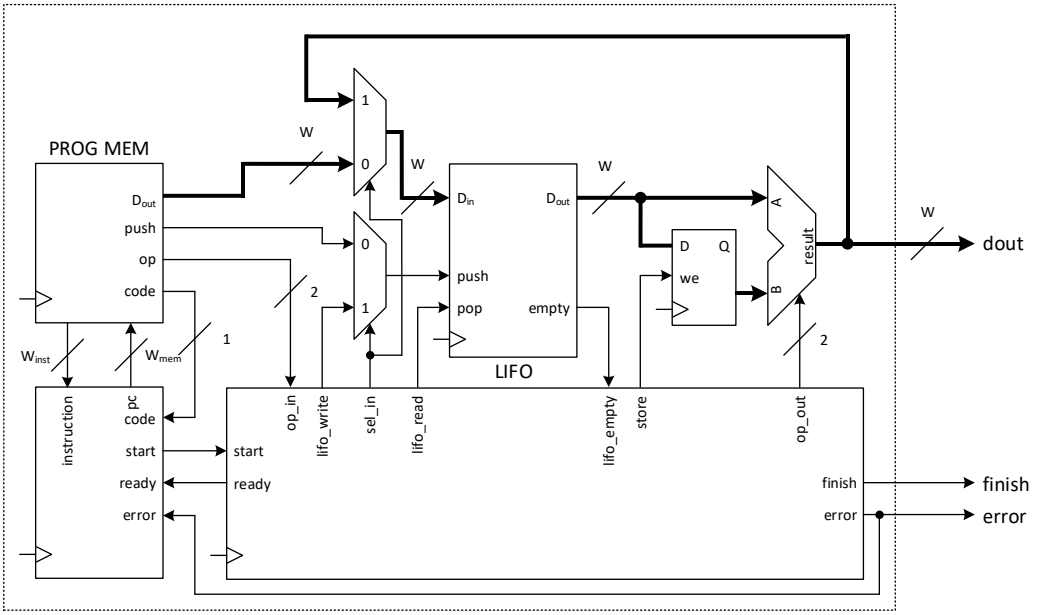


Figure 2 Entité intra2_q1.

L'entité que nous désirons obtenir sera appelée `final_q2`. `final_q2` utilise une nouvelle pile décrite dans `lifo2.vhd`. L'entité `lifo2` peut produire en sortie un (1) ou deux (2) résultats à la fois, selon que `pop1` ou `pop2` soit activé respectivement.

La nouvelle entité `final_q2` est décrite dans le fichier `final_q2.vhd` fourni dans le répertoire `./final_q2` et illustrée à la Figure 3 :

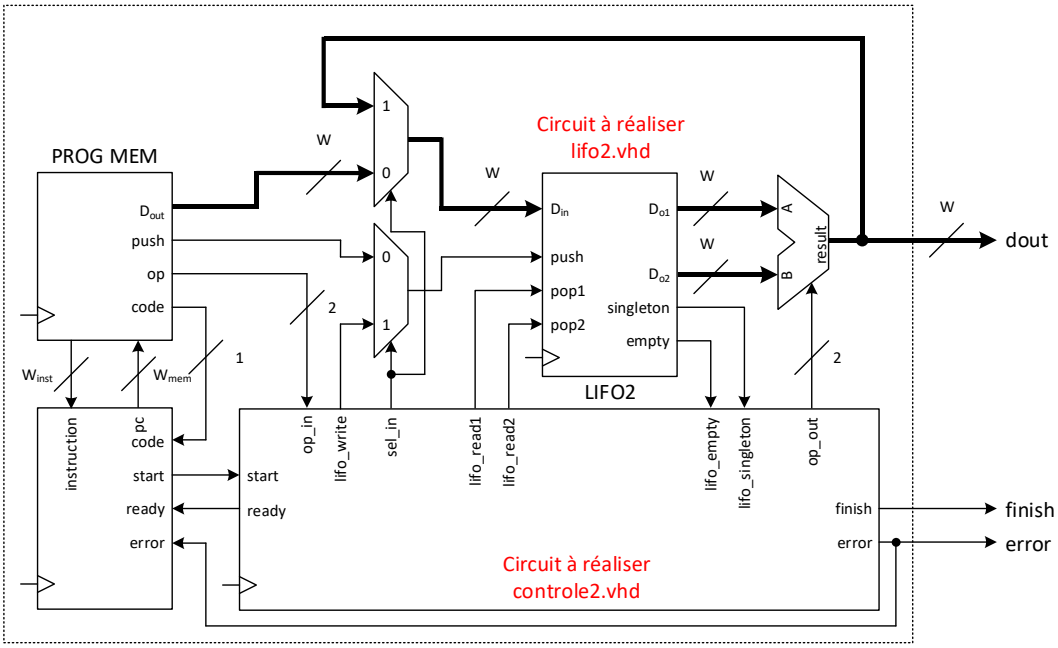


Figure 3 Entité final_q2.

Rappel :

L'entité `intra2_q1` décrit un processeur utilisant une pile (LIFO) pour analyser des expressions arithmétiques données en notation polonaise inverse (post-fixe). Par exemple :

- L'expression (3, 2, +) générera le résultat 5 ;
- L'expression (6, 2, 3, +, ×) générera le résultat 30 ;
- L'expression (6, 2, +, 3, -) générera le résultat 5.

Les arbres arithmétiques de ces trois exemples sont fournis à l'Annexe 1 à titre indicatif.

L'expression arithmétique est contenue sous la forme d'un programme entreposé dans une mémoire (PROG MEM dans les Figures 2 et 3). Le processeur lit une ligne de la mémoire (une instruction) à la fois. Si la ligne de mémoire contient une donnée, la donnée est écrite dans la LIFO. S'il s'agit d'un code d'opération, le circuit de contrôle est sollicité pour l'exécuter. Cette requête est effectuée au moyen du signal `op_in` qui décrit l'opération à effectuer et du signal `start` (actif haut) qui indique que l'exécution doit débuter. Durant l'exécution de l'opération demandée, le signal `ready` passe à 0. La prochaine ligne d'instruction est exécutée lorsque le signal `ready` revient à 1.

Quatre opérations sont possibles :

- `op_in = "00"` (`RETURN`), on lit un opérande de la LIFO, et on le recopie en sortie ;
- `op_in = "01"` (`ADD`), on lit deux opérandes de la LIFO, on les additionne et on écrit le résultat dans la LIFO ;
- `op_in = "10"` (`SUB`), on lit deux opérandes de la LIFO, on les soustrait et on écrit le résultat dans la LIFO ;
- `op_in = "11"` (`MUL`) on lit deux opérandes de la LIFO, on les multiplie et on écrit le résultat dans la LIFO.

Un exemple d'exécution de l'entité `final_q2` est présentée à l'Annexe 2 à titre de référence.

a) **(2 points)** Compléter le fichier `lifo2.vhd` fourni dans le répertoire `./final_q2. lifo2.vhd` qui implémente la pile à deux sorties, utilisée par l'entité `final_q2`.

Précisions :

- La description des ports d'entrée et de sortie est donnée en commentaire dans `lifo2.vhd` ;
- Les ports `do1` et `do2` sont registrés et initialisés à zéro au signal `rst` actif haut ;
- Les ports `do1` et `do2` maintiennent les valeurs lues jusqu'à la prochaine lecture ;
- Le signal singleton indique que la LIFO contient un seul élément ;
- Le signal `pop2` exécute la lecture de deux valeurs dans le même cycle d'horloge ;
- Les trois opérations `push`, `pop1` et `pop2` sont mutuellement exclusives, de sorte qu'il faut qu'un seul des trois signaux soit actif pour qu'une opération s'exécute. Par exemple, si `pop1` et `pop2` sont actifs en même temps, aucune des deux opérations n'est exécutée ;
- Lorsque deux valeurs sont lues de la pile pour effectuer une opération arithmétique, assurer-vous que l'ordre de lecture soit cohérent avec l'opération désirée. Cette remarque est particulièrement importante pour la soustraction. Vous pouvez vous référer au chronogramme de l'Annexe 2 et la Figure 3 ;
- L'entité `final_q2` doit produire les mêmes résultats que `intra2_q1` ;
- Inspirez-vous au besoin de la LIFO décrite dans `lifo.vhd` qui vous est fourni dans le répertoire `./intra2_q1` et dont le comportement est proche.

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

entity lifo2 is
   generic (
      N : positive := 1024; -- la profondeur (le nombre d'éléments) de la pile
      W : positive := 8     -- la largeur (en bits) de la file
   );
   port (
      clk       : in  std_logic;
      rst       : in  std_logic;              -- Synchrone et actif haut
      din       : in  signed(W - 1 downto 0); -- Donnée entrant dans la pile
      do1       : out signed(W - 1 downto 0); -- Première sortie de la pile
      do2       : out signed(W - 1 downto 0); -- Seconde sortie de la pile
      push      : in  std_logic;              -- Si actif, une donnée sera lue de din et
                                              -- empilée au front montant de clk
      pop1      : in  std_logic;              -- Si actif, une donnée sera retirée de la file
                                              -- et placée sur do1 au front montant de clk
      pop2      : in  std_logic;              -- Si actif, deux données seront retirées de la file
                                              -- et placées sur do1 et do2 au front montant de clk
      singleton : out std_logic;              -- Indique que la pile contient un seul élément
      empty     : out std_logic               -- Indique que la pile est vide
   );
end lifo2;

architecture arch of lifo2 is

   type memtype is array( 0 to N-1 ) of signed( W-1 downto 0 );
   signal memory    : memtype;
   signal nb_values : natural range 0 to N := 0;

begin

   process( clk ) is
   begin
      if( rising_edge( clk ) ) then
         if( rst = '1' ) then
            nb_values   <= 0;
            do1 <= ( others => '0' );
            do2 <= ( others => '0' );
         else
            if(    push = '1' and pop1 = '0' and pop2 = '0' and (nb_values < N) ) then
               memory( nb_values ) <= din;
               nb_values <= nb_values + 1;
            elsif( push = '0' and pop1 = '1' and pop2 = '0' and (nb_values > 0) ) then
               do1 <= memory( nb_values - 1 );
               nb_values <= nb_values - 1;
            elsif( push = '0' and pop1 = '0' and pop2 = '1' and (nb_values > 1) ) then
               do1 <= memory( nb_values - 2 );
               do2 <= memory( nb_values - 1 );
               nb_values <= nb_values - 2;
            end if;
         end if;
      end if;
   end process;

   empty     <= '1' when ( nb_values = 0 ) else '0';
   singleton <= '1' when ( nb_values = 1 ) else '0';

end arch;
```

b) (2 points) Compléter le fichier `controle2.vhd` fourni dans le répertoire `./final_q2`. `controle2.vhd` implémente une unité de contrôle de l'entité `final_q2`.

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

entity controle2 is
   port (
      clk            : in  std_logic;
      rst            : in  std_logic;
      start          : in  std_logic;
      lifo_write     : out std_logic;
      sel_in         : out std_logic;
      lifo_read1     : out std_logic;
      lifo_read2     : out std_logic;
      lifo_empty     : in  std_logic;
      lifo_singleton : in  std_logic;
      op_in          : in  std_logic_vector(1 downto 0);
      op_out         : out std_logic_vector(1 downto 0);
      finish         : out std_logic;
      error          : out std_logic;
      ready          : out std_logic
   );
end controle2;

architecture arch of controle2 is

   type state_type is (idle, op, done, err);
   signal state      : state_type := idle;
   signal next_state : state_type;

begin

   process( clk ) is
   begin
      if( rising_edge( clk ) ) then
         if( rst = '1' ) then
            state <= idle;
         else
            state <= next_state;
         end if;
      end if;
   end process;

   lifo_write <= '1'  when state = op    else '0';
   sel_in     <= '0'  when state = idle  else
                 '0'  when state = done  else
                 '0'  when state = err   else
                 '1';
   op_out     <= op_in;
   finish     <= '1'  when state = done  else '0';
   error      <= '1'  when state = err   else '0';
   ready      <= '1'  when state = idle  else
                 '0';

   ---  suite à la prochaine page
```

```
   fonction_transition:
   process( state, start, lifo_empty, op_in) is
   begin

      lifo_read1 <= '0';
      lifo_read2 <= '0';

      case state is
         when idle =>
            if( start = '0' ) then
               next_state <= idle;
            else
               if( op_in = "00" and lifo_empty = '0' ) then
                  lifo_read1 <= '1';
                  next_state <= done;
               elsif( lifo_empty = '0' and lifo_singleton = '0' ) then
                  lifo_read2 <= '1';
                  next_state <= op;
               else
                  next_state <= err;
               end if;
            end if;
         when op =>
            next_state <= idle;
         when done =>
            next_state <= done;
         when err =>
            next_state <= err;
         when others =>
            next_state <= idle;
      end case;
   end process;

end arch;
```

Question 3. (4 points)

Considérez le processeur PolyRISC de la Figure 4. Le port d'entrée `GPIO_in` permet de lire une donnée de l'extérieur dans le bloc des registres. Le port de sortie `GPIO_out` permet d'écrire à l'extérieur le contenu d'un registre spécial dans lequel on peut copier une des valeurs entreposées dans le bloc des registres. Référez-vous à l'Annexe 3 pour les différents codes du processeur.

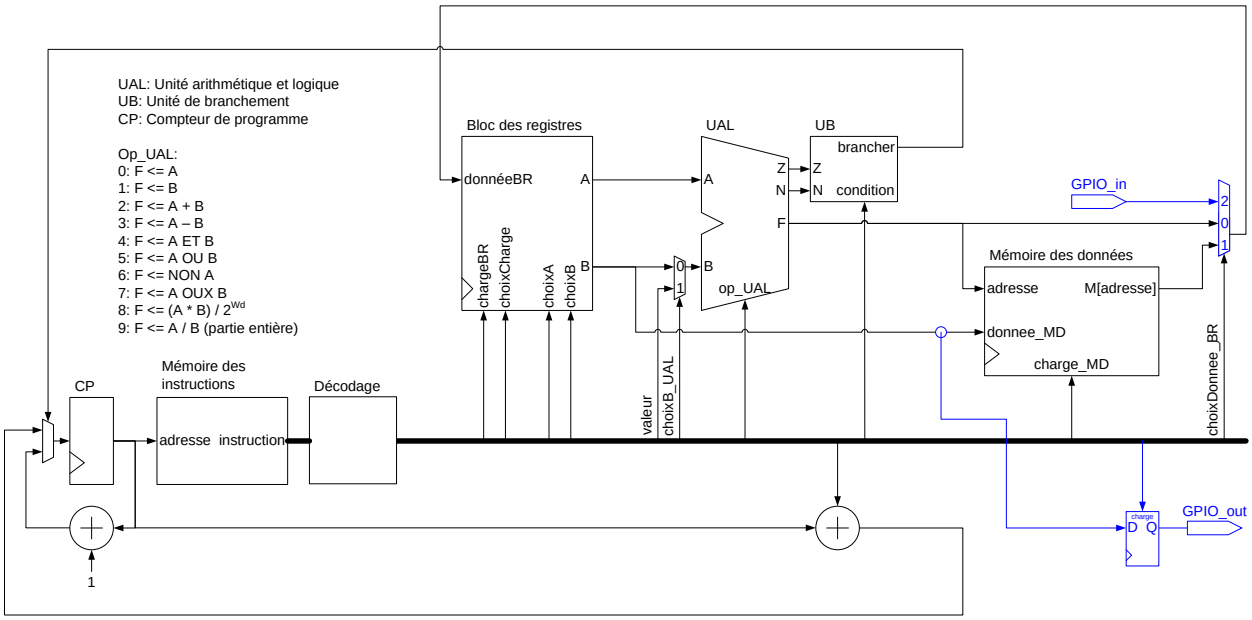


Figure 4 Processeur PolyRISC modifié.

a) **(2 points)** Remplissez le tableau suivant des signaux de contrôle du chemin des données.

opération	chargeBR	choixCharge	choixA	choixB	valeur	ChoixB_UAL	Op_UAL	Charge_MD	choixDon-nee_BR	charge (GPIO_out)
R10 := GPIO_in;	1	10	-	-	-	-	-	0	2	0
GPIO_in := R5;	0	-	-	5	-	-	-	0	-	1
M[4 + R14] := R4;	0	-	14	4	4	1	2	1	-	0
R1 := M[R1 + R2];	1	1	1	2	-	0	2	0	1	0

b) **(2 points)** Complétez le tableau suivant pour l'encodage des instructions du processeur PolyRISC.

Instruction	catégorie 31:30	détails 29:26	25:21	20:16	4:0/ 15:0
R8 := R8 + R4	0	2	8	8	4
MD[R2 + 12] := R12	2	0	9	8	25
Si (R2 < R1) goto PC - 10	2	2	1	2	-10
MD[R1 - 15] := R2	3	1	2	1	-15

Question 4. (3 points)

Vous êtes responsable de la vérification d'un système de contrôle de la vitesse d'une voiture autonome. Le système a quatre entrées :

1. La limite de vitesse (50, 70, 90 ou 100 km/h) ;
2. La période de la journée (jour ou nuit) ;
3. La visibilité (au mètre près) ;
4. L'état de la chaussée (sèche, mouillée ou enneigée).

La sortie du système est la vitesse désirée. La vitesse désirée est la limite de vitesse, moins une marge de sécurité qui dépend de la période de la journée et des conditions atmosphériques et routières.

- Pour une visibilité supérieure à 100 m, il n'y a pas de contrainte ;
- Quand la visibilité est entre 10 m et 100 m, la vitesse désirée doit être réduite de moitié ;
- Quand la visibilité est inférieure à 10 m, la voiture doit s'arrêter ;
- La nuit, la vitesse désirée doit être réduite de 10% ;
- Quand la chaussée est mouillée ou enneigée, la vitesse désirée doit être réduite de 15%.

Par exemple, dans une zone de 90 km/h, la nuit, chaussée mouillée, visibilité de 37 m, la vitesse doit être : $90 \times 0.5 \times 0.9 \times 0.85 = 34.4$ km/h.

a) **(1 point)** Proposez un partitionnement en classes pour chacune des quatre entrées.

1. Limites de vitesse : 4 classes ;
2. Période de la journée : 2 classes ;
3. Visibilité : 3 classes, <10m; ≥ 10m & ≤ 100m; >100m ;
4. État de la chaussée : 2 classes.

b) **(1 point)** Quelle est la taille minimale du vecteur de test pour effectuer un test faible du système de contrôle. Justifiez votre réponse par un calcul.

Min(4, 2, 3, 2) = 4.

c) **(1 point)** Quelle est la taille minimale du vecteur de test pour effectuer un test fort du système de contrôle. Justifiez votre réponse par un calcul.

$4 \times 2 \times 3 \times 2 = 48$

Question 5. (5 points)

Considérez le chemin des données de l'unité `carre` illustrée à la Figure 5. Ce chemin des données permet le calcul du carré d'un entier non signé `A` donné en entrée.

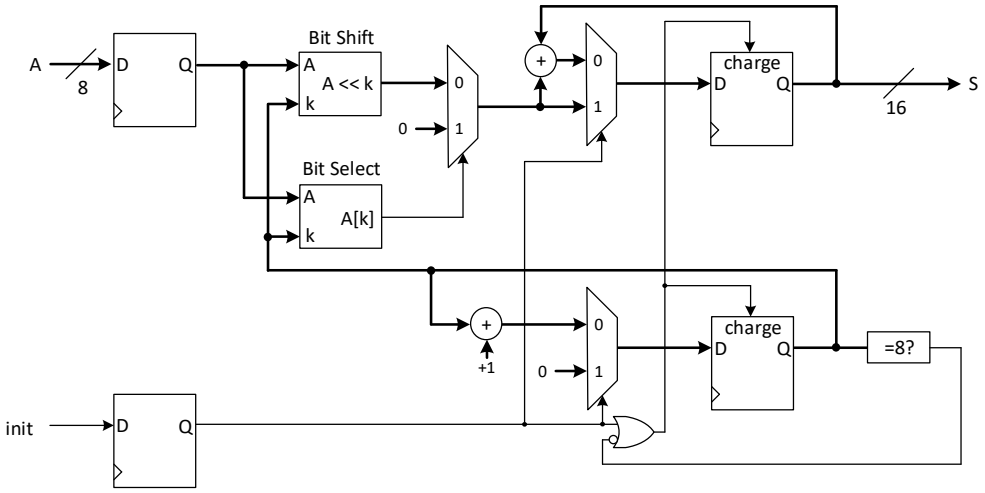


Figure 5 Chemin de données de l'unité carre.

Sachant les composants du circuit sont caractérisés par les délais de l'Annexe 4, répondez aux questions suivantes. Justifier clairement vos réponses par un calcul. Vous pouvez ignorer les délais de propagation.

a) **(2 point)** Identifiez le chemin critique et donnez la fréquence maximale de l'horloge du circuit de la Figure 5.

Bascule k – Bit Shift – Mux – ADD + MUX + Bascule S: 0.3 + 0.5 +1+2+1+0.2 = 5.0 ns ;

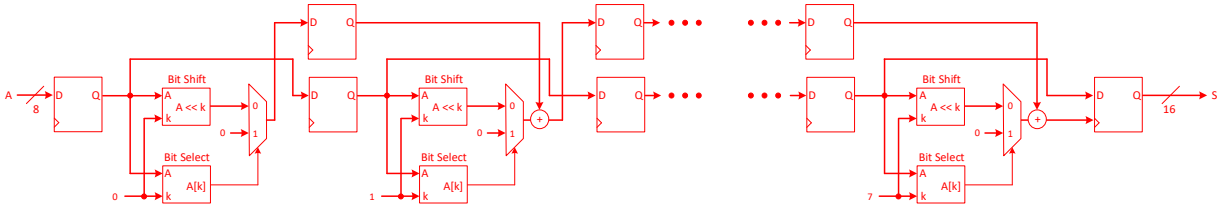
f_{max} = 200 MHz.

b) **(1 point)** Donnez un estimé de la latence en cycles d'horloges de l'unité `carre` ainsi que le débit du circuit à la fréquence maximale en résultats par seconde.

8 cycles d'horloge ;

à 200 MHz, l'unité produit donc $200 \times 10^6 / 8 = 25$ millions de résultats par seconde.

c) **(1 point)** On désire augmenter le débit de l'unité `carre` en étalant les calculs dans l'espace. Montrez comment faire en donnant un nouveau diagramme du circuit. Pipelinez le nouveau chemin de données. Donnez le nouveau débit en résultats par seconde à la fréquence maximale trouvée précédemment. Énoncez toutes vos suppositions.



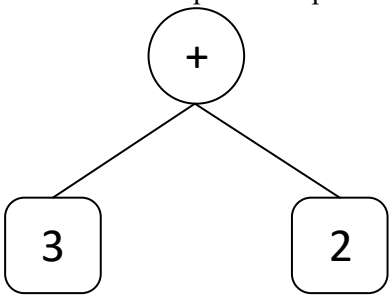
Ce faisant, une fréquence d'horloge de 200 MHz produirait 200 millions de résultats par seconde.

d) **(1 point)** Proposez une approche permettant d'atteindre un débit de 1 milliards de résultats par seconde.

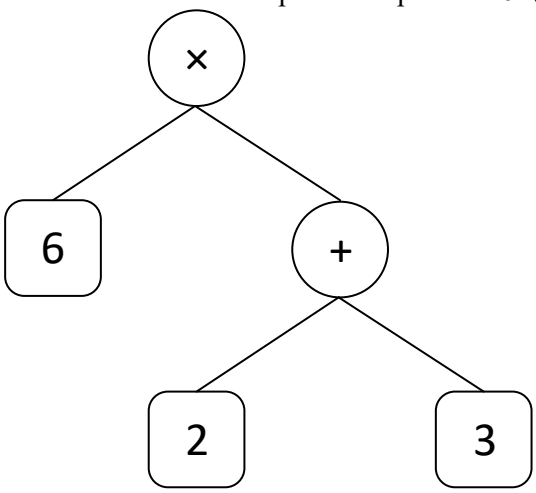
Il suffirait de dupliquer le processeur de la question 5.3 cinq (5) fois : $5 \times (200 \times 10^6) = 10^9$.

Annexe 1

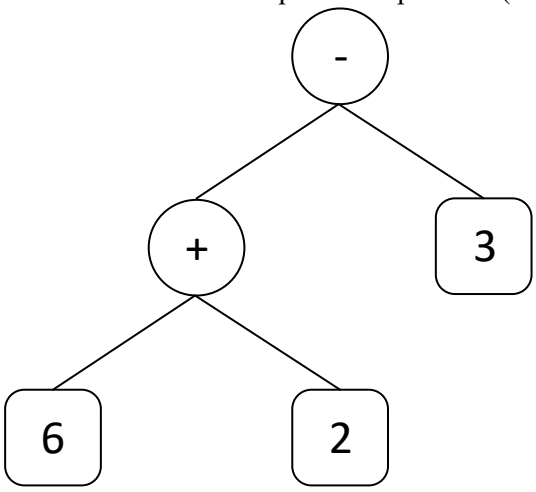
- L'expression (3, 2, +) générera le résultat 5. Elle est obtenue par le parcours post-ordre de l'arbre suivant et correspond à l'opération 3 + 2.



- L'expression (6, 2, 3, +, ×) générera le résultat 30. Elle est obtenue par la parcours post-ordre de l'arbre suivant et correspond à l'opération 6×(2 + 3).



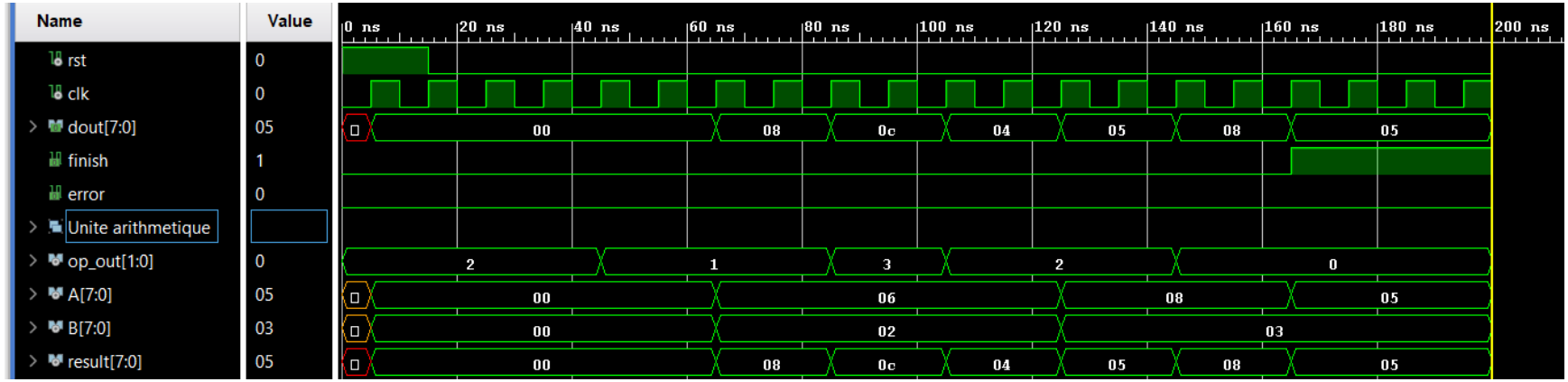
- L'expression (6, 2, +, 3, -) générera le résultat 5. Elle est obtenue par la parcours post-ordre de l'arbre suivant et correspond à l'opération (6 + 2) – 3.



Annexe 2

L'exécution du programme suivant (tiré du code fourni) par l'entité `final_q2` de référence donne le chronogramme ci-après. Il s'agit de l'expression (6, 2, +, 3, -) qui produit le résultat 5

```
type     mem_type is array( 0 to MEM_N - 1 ) of std_logic_vector( W downto 0 );
constant mem : mem_type := (
   '0' & std_logic_vector(to_signed(6, W)),
   '0' & std_logic_vector(to_signed(2, W)),
   ADD,
   '0' & std_logic_vector(to_signed(3, W)),
   SUB,
   RET,
   others => STOP
   );
```

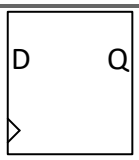
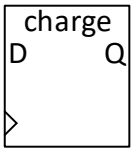
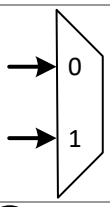
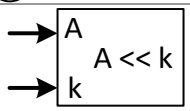
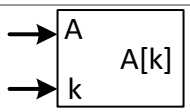
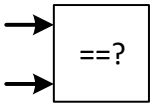
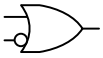


Annexe 3

Op_UAL:	Bits 29:26 (cat. 2)
0: F <= A 1: F <= B 2: F <= A + B 3: F <= A – B 4: F <= A ET B 5: F <= A OU B 6: F <= NON A 7: F <= A OUX B	0: si = 1: si != 2: si < 3: si > 4: si <= 5: si >= 6: toujours 7: jamais

Instruction	**bits 31-30** **catégorie**	**bits 29:26** **détails**	**bits 25:21**	**bits 20:16**	**bits 15:5**	**bits 4:0**
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Annexe 4

Nom	Symbole	Délais
Registre		t_d = 0.2 ns t_{su} = 0.1 ns t_h = 0.05 ns
Registre avec signal de charge		t_d = 0.3 ns t_{su} = 0.2 ns t_h = 0.05 ns
Multiplexeur		t_{comb} = 1 ns
Additionneur		t_{comb} = 2 ns
Décaleur		t_{comb} = 0.5 ns
Sélecteur		t_{comb} = 0.5 ns
Comparateur		t_{comb} = 2 ns
Logique		t_{comb} = 1 ns
Constante		t_{comb} = 0 ns