Nom : _____ Matricule : _____

INF3500 : Conception et réalisation de systèmes numériques
Examen final – Automne 2020

Durée: 2h30
Pondération: 50%

Examen à distance par activité test dans Moodle
Documentation: Toute

Question 1. (Q2/3 dans Moodle) (9 points)

Pendant votre stage de l'été 2021, vous faites partie d'une équipe qui doit concevoir un processeur réseau spécialisé. Les processeurs seront portés par des drones afin de déployer rapidement un réseau cellulaire lors d'un désastre naturel pendant lequel les infrastructures seraient détruites ou endommagées. Vous considérez trois options d'implémentation : un processeur embarqué ARM Cortex-A72 dans un système Raspberry Pi 4, un FPGA et une puce conçue sur mesure en logique fixe.

L'effort de développement pour le ARM serait entièrement en logiciel et prendrait 2 mois à 2 ingénieur/es, alors qu'il faudrait 4 mois à 5 ingénieurs pour le FPGA et 7 mois de travail à 6 ingénieurs pour la logique fixe. Tous les outils sont libres pour le Raspberry Pi. Pour le FGPA, les frais de licences sont de 250 $/poste/mois, alors qu'ils sont le double pour la logique fixe. Pour la logique fixe, les frais de démarrage de la fonderie seraient de 600000 $, puis les puces couteraient 3 $ chacune. Le salaire et les frais d'emploi d'un/e ingénieur/e s'élèvent à 10000 $/mois. Les FPGA coutent 34 $ chacun et les Raspberry Pi coutent 150 $ chacun. Pour concevoir et fabriquer les cartes, il faudrait compter 25 $ pour les FPGA et 14 $ pour la logique fixe, alors que les Raspberry Pi arriveraient déjà tous montés. Les délais de livraison sont les mêmes pour les Raspberry Pi et les FPGA, alors que les processeurs en logique fixe arriveraient 4 mois après la fin de la conception.

Quels seraient les couts pour chaque système pour les quantités suivantes, arrondis au $ près ?

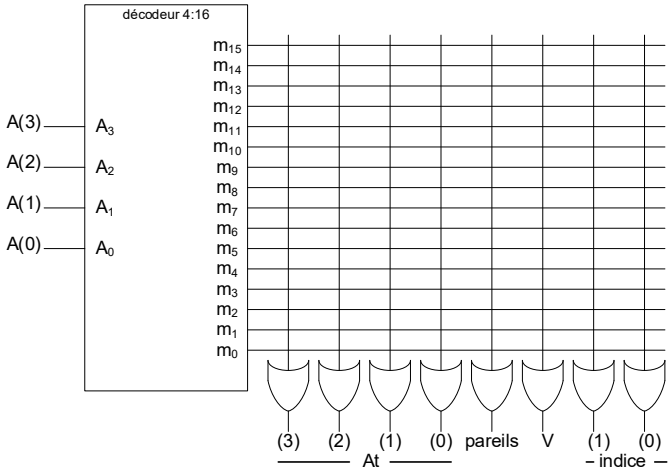
	Raspberry Pi 4	FPGA	Logique fixe
1 système			
100 systèmes			
1000 systèmes			

Question 2. (Q4/5 dans Moodle) (8 points)

Considérez le module VHDL suivant.

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

entity module19 is
    port (
    A : in unsigned(3 downto 0);
    At : out unsigned(3 downto 0);
    pareils : out std_logic;
    V : out std_logic;
    indice : out unsigned(1 downto 0)
    );
end module19;

architecture arch of module19 is
begin

    process(A)
    begin
        V <= '0';
        indice <= (others => '0');
        for k in A'range loop
            if A(k) = '1' then
                indice <= to_unsigned(k, indice'length);
                V <= '1';
                exit;
            end if;
        end loop;
    end process;

    process(A)
    begin
        for k in A'range loop
            At(A'length - k - 1) <= A(k);
        end loop;
    end process;

    pareils <= and(A) or nor(A);

end arch;
```

Considérez le diagramme suivant d'une ROM à 4 entrées et 8 sorties. Le pairage des ports entre le module VHDL et la ROM est indiqué. Montrez comment programmer la ROM pour implémenter le module VHDL.



Question 3. (Q6 dans Moodle) (8 points)

Faites la conception d'un processeur pour effectuer le calcul de la somme des différences absolues (SDA) entre deux blocs de 256 pixels A et B gardés en mémoire. La SDA mesure la similarité entre les deux blocs, permettant par exemple de faire le suivi d'objets dans une séquence vidéo (on imagine le bloc A contenant une image de ballon, et le bloc B contenant une partie de l'image dans laquelle on cherche le ballon). Chaque octet de chaque mémoire contient l'intensité d'un seul pixel sous la forme d'un nombre non signé entre 0 et 255 inclusivement, représentant un ton de gris. La SDA correspond à accumuler la somme de la valeur absolue des différences, pixel à pixel, des blocs A et B. Une version en pseudocode d'un algorithme permettant de calculer la SDA est donné ici.

```
si reset == '1' {
    fini ← '1'
    adresse ← 0
    somme ← 0
} sinon, à chaque coup d'horloge {
    si adresse == 0 {
        fini ← '1'
        SDA ← somme
        si go == '1' {
            somme ← 0
            adresse ← 255
            fini ← '0'
        }
    } sinon {
        somme ← somme + abs(A[adresse] – B[adresse])
        adresse ← adresse - 1
    }
}
```

Les calculs doivent débuter quand le signal d'entrée `go` est activé, on comprend après que les blocs A et B aient été chargés avec les bonnes données. Les ports d'adresse des deux blocs de mémoire sont menés par le signal de sortie `adresse` et leurs ports de données sont les entrées `A` et `B`. Le signal de contrôle `fini` est activé par le processeur quand les calculs sont terminés et que la valeur sur le port de sortie `SDA` est bonne.

a. (4 points) Donnez un diagramme du chemin des données de ce processeur. Indiquez clairement toutes les ressources requises, leur nombre, leur largeur en bits et leurs signaux de contrôle. Remettez votre diagramme dans un fichier SDA.svg, SDA.png, SDA.jpg, SDA.pdf ou un autre format graphique de votre choix.

b. (4 points) Complétez le code VHDL suivant pour modéliser le processeur. Ne modifiez pas les noms de l'entité et de l'architecture, ni les ports d'entrée et de sortie. Remettez votre code dans un fichier SDA.vhd.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity SDA is
  port (
    CLK, reset, go : in std_logic;
    A, B : in unsigned(7 downto 0); -- bus des données de chaque bloc
    adresse : out unsigned(7 downto 0); -- port d'adresse des blocs
    SDA : out unsigned(15 downto 0);
    fini : out std_logic
  );
end SDA;

architecture arch of SDA is
-- votre code ici
begin
-- votre code ici
end arch;
```

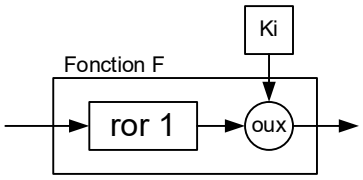
Question 4. (Q7/8 dans Moodle) (5 points)

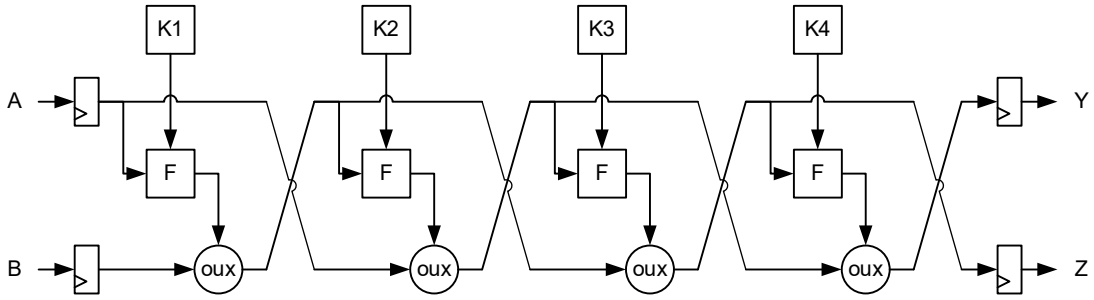
Considérez le code VHDL suivant qui modélise un processeur pour implémenter un réseau de Feistel à quatre étages, montré sur le diagramme.

Estimez le nombre de ressources nécessaires en termes de tables de conversion (LUT), bascules (FF) et tranches DSP48 pour l'implémenter sur un FPGA de la série 7 de Xilinx. Supposez que W = 16 bits.

Justifiez complètement votre analyse.

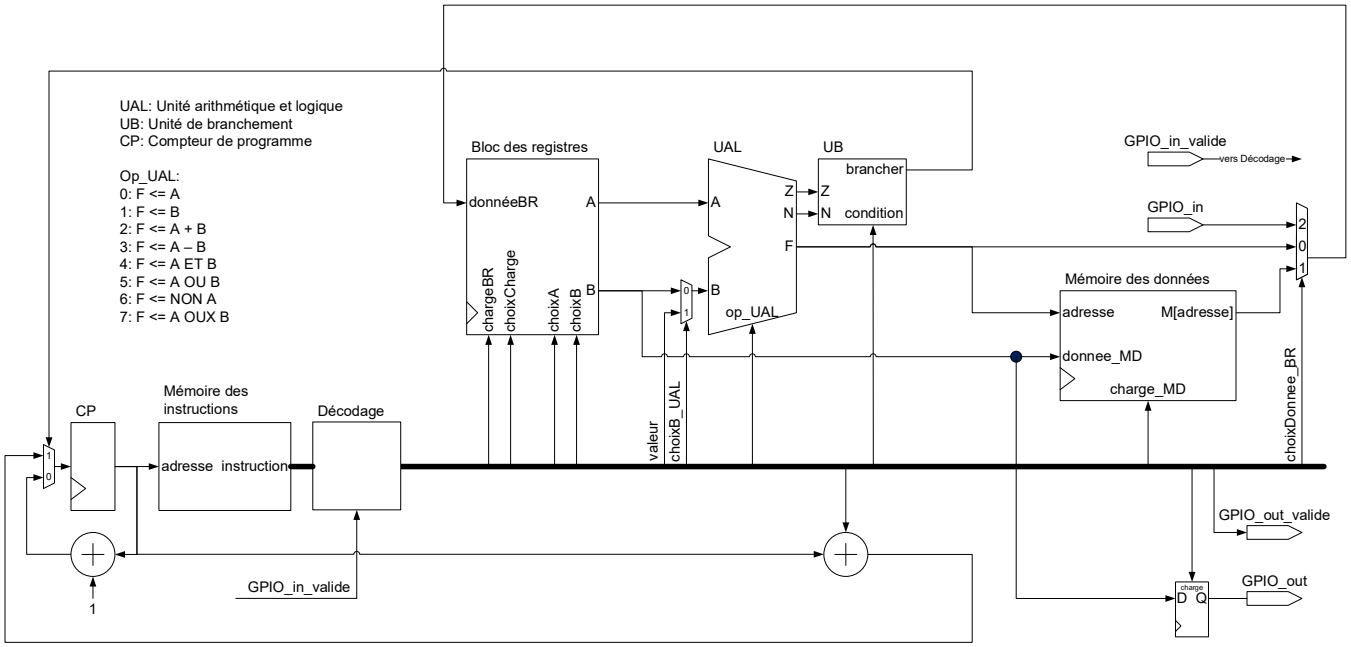
```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity feistel4 is
    generic (
        W : positive := 16
    );
    port (
        clk, reset : in std_logic;
        A, B : in unsigned(W - 1 downto 0);
        Y, Z : out unsigned(W - 1 downto 0)
    );
end feistel4;

architecture arch1 of feistel4 is

type unsigned2D is array(natural range <>) of unsigned(W - 1 downto 0);
constant K : unsigned2D(1 to 4) := (x"1234", x"5678", x"9ABC", x"DEF0");

signal A0, A1, A2, A3, A4, B0, B1, B2, B3, B4 : unsigned(W - 1 downto 0);

begin

    process (clk, reset)
    begin
        if reset = '1' then
            Y <= (others => '0');
            Z <= (others => '0');
            A0 <= (others => '0');
            B0 <= (others => '0');
        elsif rising_edge(clk) then
            A0 <= A;
            B0 <= B;
            Y <= A4;
            Z <= B4;
        end if;
    end process;

    B1 <= A0;
    A1 <= (A0 ror 1) xor K(1) xor B0;
    B2 <= A1;
    A2 <= (A1 ror 1) xor K(2) xor B1;
    B3 <= A2;
    A3 <= (A2 ror 1) xor K(3) xor B2;
    B4 <= A3;
    A4 <= (A3 ror 1) xor K(4) xor B3;

end arch1;
```


Ki
Fonction F
ror 1
oux


K1
K2
K3
K4
A
B
F
F
F
F
oux
oux
oux
oux
Y
Z

Question 5. (Q9/10 dans Moodle) (10 points)

Considérez le diagramme suivant du processeur PolyRISC. Le bloc des registres a 16 registres de 32 bits et le compteur de programme a 16 bits.



a. (5 points) Complétez le tableau suivant des signaux de contrôle du chemin des données pour chaque opération. Pour la colonne valeur, exprimez la quantité en hexadécimal avec quatre chiffres. Pour les valeurs binaires, inscrivez un '0' ou un '1'. Pour les valeurs sans importance, inscrivez un tiret '-'. Si l'opération est impossible, inscrivez un 'F' dans toute la rangée.

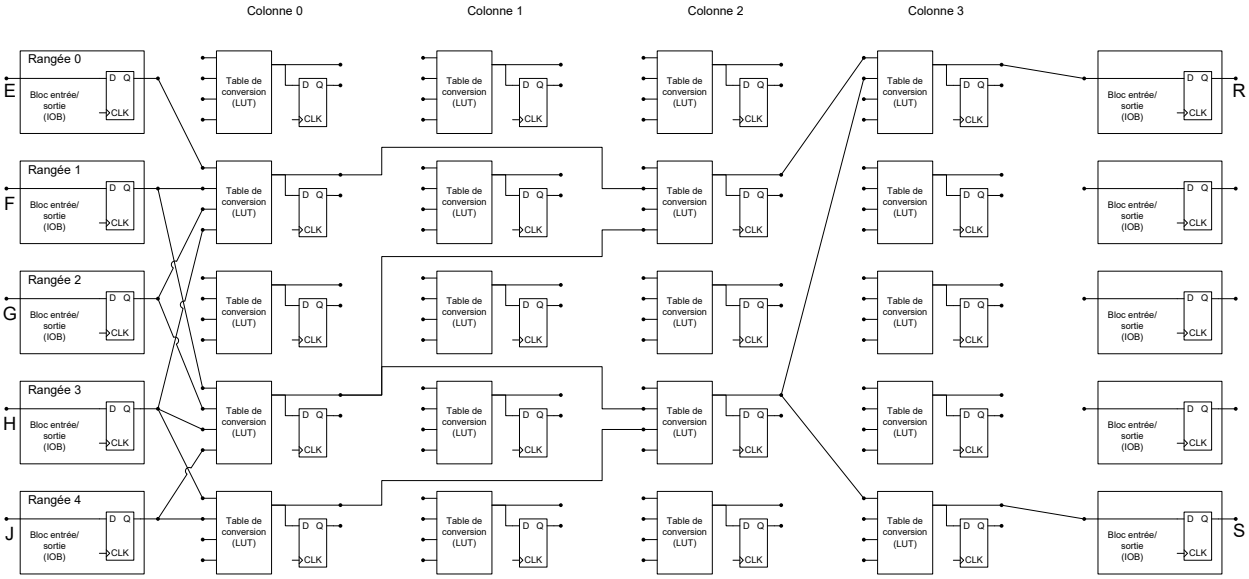
opération	chargeBR	choixCharge	choixA	choixB	valeur	ChoixB_UAL	Op_UAL	Charge_MD	choixDonnee_BR	charge (GPIO_out)
R9 := R7 oux x"FFFF";										
M[R4] := R11 OU R13										
R14 := GPIO_in + R1										
R8 := M[R5 ET R3];										
R6 := M[R12 – R2];										

b. (5 points) En excluant la mémoire des données, identifiez le chemin critique et calculez la fréquence maximale d'horloge du processeur PolyRISC implémenté sur un FPGA bas de gamme. Le bloc des registres a un délai t_D de 3 ns et un temps de préparation t_{su} de 1 ns. L'UAL a un délai de 10 ns. Les multiplexeurs ont tous un délai de 2 ns. Le compteur de programme a un délai de 2 ns et un temps de préparation t_{su} de 1 ns. La mémoire des instructions a un délai de 8 ns et le module de décodage a un délai de 4 ns. Les additionneurs ont des délais de 8 ns. L'unité de branchement a un délai de 1 ns. On néglige les temps de propagation sur les fils. Décrivez clairement le chemin critique et donnez un exemple d'instruction où il serait exercé.

Question 6. (Q11 dans Moodle) (10 points)

Considérez le diagramme suivant montrant un circuit implémenté sur un FPGA simplifié. Les entrées du circuit sont les signaux E, F, G, H et J. Il y a deux sorties R et S. Le FPGA est composé de tranches ayant chacune une table de correspondance (LUT) et une bascule pouvant être utilisée ou non, et de blocs d'entrée/sortie (IOB) dans lesquels les bascules sont toujours utilisées.

Pour ce FPGA, les LUT ont un délai de 0.3 ns. Les bascules ont un délai de 0.4 ns, un temps de préparation t_{su} de 0.2 ns, et un temps de maintien t_h de 0.1 ns. Chaque fil d'interconnexion ajoute un délai de 0.25 ns pour chaque rangée et chaque colonne de distance. Bien que des lignes diagonales soient tracées sur le diagramme, tous les signaux sont routés en lignes horizontales et/ou verticales.



a. (6 points) Pipelinez le circuit pour atteindre un débit de 500×10^6 résultats par seconde, où un résultat est une paire (P, Q). Annotez le diagramme pour montrer les modifications à apporter au circuit. Justifiez complètement votre approche.

b. (4 points) On veut reprogrammer complètement le FPGA avec le module décrit dans le code VHDL suivant. Quelle serait la fréquence maximale d'horloge qu'on puisse atteindre ? Ne pas modifier le code. Justifiez complètement votre réponse.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity inverseur is
  port (
    clk, E : in std_logic;
    R : out std_logic
  );
end inverseur;
```

```
architecture arch of inverseur is
signal Etemp : std_logic;
begin
  process (clk)
  begin
    if rising_edge(clk) then
      Etemp <= E;
      R <= not Etemp;
    end if;
  end process;
end arch;
```

Solutions

#1 options d'implémentation

tableau	Raspberry Pi	FPGA	logique-fixe
1	40 150 $	205 059 $	1 041 017 $
100	55 000 $	210 900 $	1 042 700 $
1000	190 000 $	264 000 $	1 058 000 $

#2 ROM

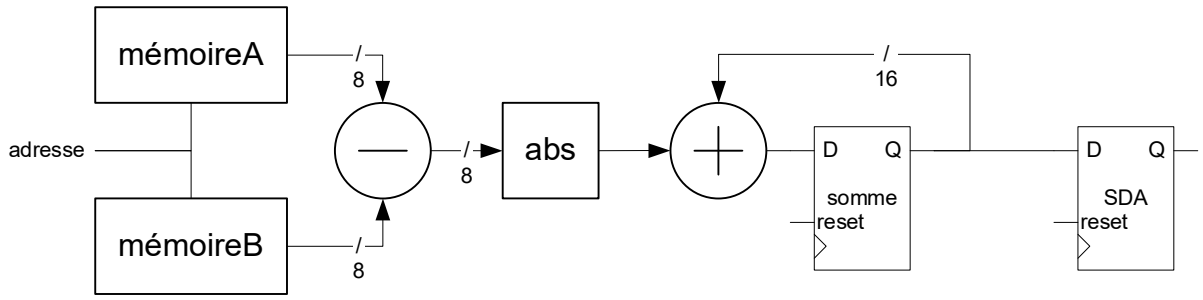
à compléter

#3 SDA

a. ** solution incomplète, chemin des données seulement **

Il manque un registre pour adresse, avec un additionneur pour en décrémenter la valeur et un moyen de l'initialiser à 255.

Il manque une vérification que adresse == 0 pour contrôler le reste du circuit.



b. une réponse possible

```
architecture arch of SDA is
type type_etat is (attente, init, accumule);
signal etat : type_etat;
begin
  process(CLK, reset) is
  variable adresseInt : integer range 0 to 255;
  variable somme : unsigned(15 downto 0);
  begin
    if (rising_edge(CLK)) then
        if (reset = '1') then
            etat <= attente;
     else
       case etat is
         when attente =>
           if (go = '1') then
             etat <= init;
           end if;
         when init =>
           etat <= accumule;
           somme := (others => '0');
           adresseInt := 255;
         when accumule =>
           somme := somme + abs(to_integer(A) - to_integer(B));
           if (adresseInt = 0) then
             etat <= attente;
           else
             adresseInt := adresseInt - 1;
           end if;
         when others =>
           etat <= attente;
       end case;
     end if;
    end if;
    adresse <= to_unsigned(adresseInt, adresse'length);
    SDA <= somme;
  end process;
  fini <= '1' when etat = attente else '0';
end arch;
```

#4 Feistel – coûts

FF : 4 registres de 16 bits, donc 64 bascules

LUT : les fonctions F et oux peuvent être combinées à chaque étage. Le bloc résultant agit sur trois bits : un bit du haut, un bit du bas et une constante. Chaque fonction entre bien dans une seule LUT à 6 entrées. Chaque signal est sur 16 bits, donc il y a 16 LUT par étage, donc 64 LUT au total pour 4 étages.

Il pourrait être possible aussi de combiner deux étages, et de considérer alors que chaque LUT aurait 4 entrées : signal du haut, signal du bas et deux constantes. Chaque signal est sur 16 bits, donc 16 LUT par paire d'étages, total serait alors de 32 LUT.

On peut pousser le raisonnement à l'extrême et considérer qu'il ne faut que 16 LUT à 6 entrées chacune : un bit de A0, un bit de B0, et un bit de K1, K2, K3 et K4.

tranches DSP48 : aucune

#5 PolyRISC

a.

opération	chargeBR	choixCharge	choixA	choixB	valeur	ChoixB_UAL	Op_UAL	Charge_MD	choixDonnee_BR	charge (GPIO_out)
R9 := R7 oux x"5C5C"	1	9	7	-	5C5C	1	7	0	0	0
M[R4] := R11 OU R13	F	F	F	F	F	F	F	F	F	F
R14 := GPIO_in + R1	F	F	F	F	F	F	F	F	F	F
R8 := M[R5 ET R3]	1	8	{5, 3}	{3, 5}	-	0	4	0	1	0
R6 := M[R12 – R2]	1	6	12	2	-	0	3	0	1	0

b. Le chemin critique serait :

CP, mémoire des instructions, décodage, mux, UAL, UB, mux, tsu

2 + 8 + 4 + 2 + 10 + 1 + 2 + 1 = 30 ns, pour une fréquence maximale d'horloge de 33.3 MHz.

Ce chemin serait activé lors d'une instruction de branchement.

#6 délais sur FPGA

a. On veut Tmin ≤ 2 ns pour un débit de 500×10^6 résultats par seconde.

Il faut pipeliner aux colonnes 0, 2 et 3.

Le chemin critique est alors de la colonne 2 à la colonne 3 :

Tmin = 0.4+ 4 × 0.25 + 0.3 + 0.2 = 1.9 ns

b. Le signal Etemp serait placé dans un registre d'un bloc d'entrée, et le signal de sortie R dans un bloc de sortie. Il faut utiliser une table de conversion pour l'inversion. On place tout sur une seule rangée. On a alors Tmin = 0.4 + 0.3 + 5 × 0.25 + 0.2 = 2.15 ns, donc fmax = 465 MHz