Question 1 (5 points) Architectures scalaires, superscalaires et VLIW

Soit le code de multiplication de 2 matrices carrées à la figure 1.1, très similaire à celui au laboratoire no 2, sauf qu'ici afin de mieux préciser les opération d'accès à un élément de la matrice, on utilise un vecteur plutôt qu'un tableau 2 dimensions. Plus précisément, pour représenter la matrice 4 x 4 suivante :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

On aura le vecteur suivant:

Ligne 1				Ligne 2				Ligne 3				Ligne 4			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

```
1  #define MATRIX_ROWS 4
2  #define MATRIX_COLUMNS 4
3  double op1[MATRIX_ROWS * MATRIX_COLUMNS];
4  double op2[MATRIX_ROWS * MATRIX_COLUMNS];
5  double result[MATRIX_ROWS * MATRIX_COLUMNS];
6
7  void matrix_mult::action() {
8      double temp;
9  L1: for(int i = 0 ; i < MATRIX_ROWS; i++) {
10 L2:     for(int j = 0; j < MATRIX_COLUMNS; j++) {
11             temp = 0;
12 L3:         for(int k = 0; k < MATRIX_COLUMNS; k++) {
13
14 temp += (op1[i*MATRIX_ROWS+k] * op2[k*MATRIX_ROWS+j]);
15             }
16             result[i*MATRIX_COLUMNS+j] = temp;
17         }
18     }
19
```

Figure 1.1

Le code assembleur DLX de la figure 1.1 est donné à la figure 1.2 de la page suivante alors que le détail des instructions est donné en Annexe Fig. A1). Notez qu'il s'agit d'un modèle M4 tel que défini en classe.

```
;  calcul matricielle pour une matrice 4 x 4
; R1 contient la dimension de la matrice n x n (ici n = 4)
; R2 contient l'adresse de départ de op1
; R3 contient l'adresse de départ de op2
; R4 contient l'adresse de départ de result
             MOV R1, R5          ; chargement de l'indice de ligne i
L1:          MOV R1, R6          ; chargement de l'indice de colonne j
L2:          MOV R1, R7          ; chargement de l'indice k
             LD F40,#0           ; initialisation de l'accumulateur
L3:          MULT R8,R5,R1       ; calculs de l'adresse de op1[i*MATRIX_COLUMNS+k]
             ADD R8, R8,R7
             SLLI R8, #3         ; on avance par coup de 8 bytes (double mots)
             ADD R8, R8,R2       ; on ajoute à cela l'adresse de départ de op1
             LD F10, 0(R8)
             MULT R9,R7,R1       ; calculs de l'adresse de op2[k*MATRIX_COLUMNS+j]
             ADD R9, R9, R6
             SLLI R9, #3          ; on avance par coup de 8 bytes (double mots)
             ADD R9, R9, R3       ; on ajoute à cela l'adresse de départ de op2
             LD F20, 0(R9)        ; chargement de la position j
             MULTD F30, F10,F20   ; multiplication et
             ADDD F40, F40, F30   ; accumulation dans temp (F40)
             SUBI R7, R7, #1
             BNEQ R7, L3
             MULT R8, R5,R1       ; calculs de l'adresse de result[i*MATRIX_COLUMNS+j]
             ADD R8, R8, R6
             SD 0(R8), F40        ; on emmagasine le résultat dans i, j
             SUBI R6, R6, #1
             BNEQ R6, L2
             SUBI R5, R5, #1
             BNEQ R5, L1
```

Figure 1.2 Code DLX du calcul matriciel. Au besoin, voir les instructions en Annexe

a) (1.5 pts) Au tableau 1.1, complétez la trace pour l'exécution d'une itération de la boucle L3 pour le pipeline DLX à cinq (5) étages avec les unités points flottants pipelinés. Donnez le nombre de cycles pour 4 itérations.

##		Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	
1	L3	MULT R8,R5,R1	LI	DI	E1	E2	ME	ER																							
2		ADD R8, R8,R7		LI	DI	SU	EX	ME	ER																						
3		SLLI R8, #3			LI	SU	DI	EX	ME	ER																					
4		ADD R8, R8,R2					LI	DI	EX	ME	ER																				
5		LD F10, 0(R8)						LI	DI	EX	ME	ER																			
6		MULT R9,R7,R1							LI	DI	E1	E2	ME	ER																	
7		ADD R9, R9, R6								LI	DI	SU	EX	ME	ER																
8		SLLI R9, #3									LI	SU	DI	EX	ME	ER															
9		ADD R9, R9, R3											LI	DI	EX	ME	ER														
10		LD F20, 0(R9)												LI	DI	EX	ME	ER													
11		MULTD F30, F10,F20													LI	DI	SU	E1	E2	E3	E4										
12		ADDD F40, F40, F30														LI	SU	DI	SU	SU	SU	E1	E2	E3	ME	ER					
13		SUBI R7, R7, #1																LI	SU	SU	SU	DI	EX	ME	ER						
14		BNEQ R7, L3																				LI	DI	EX							
15																									LI						

Tableau 1.1 À compléter pour la question 1a)

Ça fait 22 cycles donc 88 cycles pour 4 itérations

b) (2 pts) Proposez un déroulement de L3 afin d'enlever le maximum de suspensions. Pour cela vous devez :

b.1) déterminer d'abord le nombre optimal de déroulements et le justifier.

On doit dérouler 4 fois pour l'opération de MULTD. Cela coïncide avec la dimension de la matrice donc facilitera la gestion de l'indice k.

b.2) dérouler, regrouper ensemble les mêmes opérations et ordonnancer en complétant le tableau 1.2, et finalement

Voir tableau 1.2

b.3) donner le nombre de cycles et comparer le gain par rapport à a).

Ça fait 36 cycles plutôt que 88 cycles donc une accélération de 2.33

Cycle	Pipeline 1		Pipeline 1 (suite)
1	*MULT R8, R5, R1*	35	SU
2	*MULT R9, R7, R1*	36	*ADDD F40, F40, F33*
3	*ADD R8, R8,R7*	37	
4	*ADD R9, R9, R6*	38	
5	*SLLI R8, #3*	39	
6	*SLLI R9, #3*	40	
7	*ADD R80, R8,R2*	41	
8	*ADD R90, R9,R3*	42	
9	*ADDI R81, R80, #8 // on passe au prochain élément*	43	
10	*ADDI R82, R80, #16*	44	
11	*ADDI R83, R80, #24*	45	
12	*ADDI R91, R90, #40 // ici pour prochain élément on //doit passer à la ligne suivante*	46	
13	*ADDI R92, R90, #80*	47	
14	*ADDI R93, R90, #120*	48	
15	*LD F10, 0(R80)*	49	
16	*LD F11, 0(R81)*	50	
17	*LD F12, 0(R82)*	51	
18	*LD F13, 0(R83)*	52	
19	*LD F20, 0(R90)*	53	
20	*LD F21, 0(R91)*	54	
21	*LD F22, 0(R92)*	55	
22	*LD F23, 0(R93)*	56	
23	*MULTD F30, F10,F20*	57	
24	*MULTD F31, F11,F21*	58	
25	*MULTD F32, F12,F22*	59	
26	*MULTD F33, F13,F23*	60	
27	*ADDD F40, F40, F30*	61	
28	*SU*	62	
29	*SU*	63	
30	*ADDD F40, F40, F31*	64	
31	*SU*	65	
32	*SU*	66	
33	*ADDD F40, F40, F32*	67	
34	*SU*	68	

Tableau 1.2 À compléter pour la question 1b)

c) (1.5 pts) On souhaite faire l'implémentation sur une machine VLIW 192 bits ayant les spécifications suivantes :

- 2 unités de load/store (pour instructions LD, SD),
- 2 unités entières (pour instructions SUBI, SLLI, BNEQ, ADD, MULT, etc.) et
- 1 unité point flottant (pour instructions ADDD, MULTD).

Les instructions sont identiques à celles de l'Annexe.

Proposez une implémentation de la boucle L3 sur cette machine VLIW. Pour cela, complétez le Tableau 1.3 et expliquez bien votre démarche. Comme suggéré en classe, utilisez des flèches pour montrer le respect des dépendances inter pipeline. Aurait-on eu un meilleur résultat avec 2 unités point flottants plutôt que 1? Expliquez.

Le résultat aurait été similaire car : 1) ça nous prendrait 2 unités de transfert additionnel et 2) il y a les dépendances de l'accumulation (celle-ci aurait pu être traité en log N mais ici N=4 n'en vaut pas la peine).

	Unité transfert 1	Unité transfert 2	Unité entière 1	Unité entière 2	Unité flottante 1
			MULT R8, R5, R1	*MULT R9, R7, R1*	
1			*SU*	*SU*	
2			*ADD R8, R8,R7*	*ADD R9, R9, R6*	
3			*SLLI R8, #3*	*SLLI R9, #3*	
4			`ADD R80, R8,R2`	`ADD R90, R9,R3`	
5	`LD F10, 0(R80)`	`LD F20, 0(R90)`	*ADDI R81, R80, #8*	*ADDI R91, R90, #40*	
6	`LD F11, 0(R81)`	`LD F21, 0(R91)`	*ADDI R82, R80, #16*	*ADDI R92, R90, #80*	
7	`LD F12, 0(R82)`	`LD F22, 0(R92)`	*ADDI R83, R80, #24*	*ADDI R93, R90, #120*	`MULTD F30, F10,F20`
8	`LD F13, 0(R83)`	`LD F23, 0(R93)`			`MULTD F31, F11,F21`
9					`MULTD F32, F12,F22`
10					`MULTD F33, F13,F23`
11					`ADDD F40, F40, F30`
12					`SU`
13					`SU`
14					`ADDD F40, F40, F31`
15					`SU`
16					`SU`
17					`ADDD F40, F40, F32`
18					`SU`
19					`SU`
20					`ADDD F40, F40, F33`
21					
22					

Tableau 1.2 À compléter pour la question 1c)

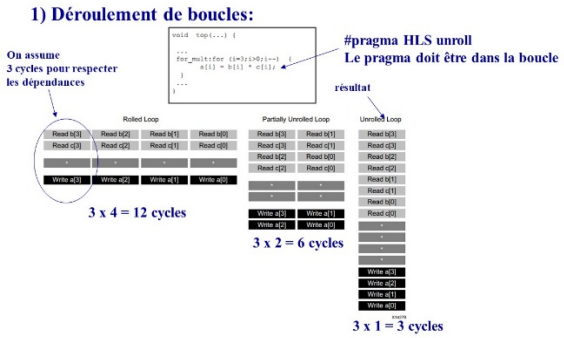
Question 2 (4 points) Synthèse de haut niveau et pragma

a) (1 pt) À partir du code de la figure 2.1, décrivez la différence entre un pragma permettant le déroulement de boucle (#pragma HLS UNROLL) et un pragma permettant de pipelinage d'une boucle (#pragma HLS PIPELINE).

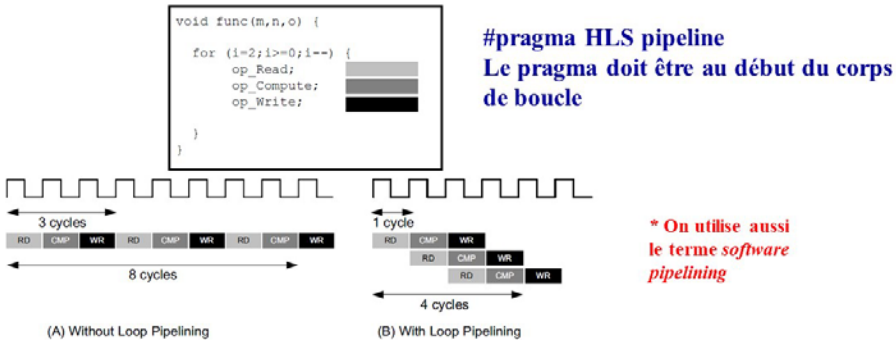
```
void top(...) {

...

for (i=3; i > 0; i--) {
      a[i] = b[i] * c[i];
      }

....

}
```

Figure 2.1

Pragma unroll :



Pragma pipeline de boucle :



b) (1 pt) Décrivez la différence entre un pragma permettant le pipelinage d'une boucle et un pragma permettant le pipelinage d'une fonction. Également, aurait-on pu avoir un pipeline de fonction dans le laboratoire no 2? Expliquez.

Oui si on avait mis le pragma au dessus de L1 on aurait pipeliner un calcul matriciel complet.

c) (1.5 pt) Soit un calcul de multiplication matricielle 72x72 sur des *int32* et pour lequel on a inséré un pragma HLS PIPELINE (#pragma HLS pipeline II=1) entre L2 et L3 (figure 2.2) et qu'on souhaite ne pas modifié au cours de l'exploration architecturale. Indiquez les pragma qui seront requis (dans le style *HLS ARRAY_PARTITION block factor)* pour les différents débits souhaités. Donnez également le nombre de DSP48E alloué (sachant que 1 multiplication/acc = 3 DSP48E):

c.1) Débit de 1 (II=1)
Block factor = 36 et on a besoin de 3 x 72 DSP Unit = 216

c.2) Débit de 2 (II=2)
Block factor = 18 et on a besoin de 3 x 72/2 DSP Unit = 108

c.3) Débit de 4 (II=4)
Block factor = 9 et on a besoin de 3 x 72/4 DSP Unit = 54

d) (.5 pt) Toujours pour un calcul de multiplication de matrice carrée (DIM x DIM), quelle est la dimension limite sur un Zynq SoC 7020 pour avoir II=1? Expliquez.

140 blocs BRAM à 2 ports chacun donc 140x2 = 280 x 280. Du côté DSP, on a 220/3=73 donc min(280, 73) = 73x73…

```
1  #define MATRIX_ROWS 72
2  #define MATRIX_COLUMNS 72
3     int op1[MATRIX_ROWS * MATRIX_COLUMNS];
4     int op2[MATRIX_ROWS * MATRIX_COLUMNS];
5     int result[MATRIX_ROWS * MATRIX_COLUMNS];
6
7  void matrix_mult::action() {
8          int temp;
9  L1: for(int i = 0 ; i < MATRIX_ROWS; i++) {
10 L2:      for(int j = 0; j < MATRIX_COLUMNS; j++) {
11                 temp = 0;
12 L3:            for(int k = 0; k < MATRIX_COLUMNS; k++) {
13
14 temp += (op1[i*MATRIX_ROWS+k] * op2[k*MATRIX_ROWS+j]);
15                 }
16                 result[i*MATRIX_COLUMNS+j] = temp;
17             }
18     }
19
```

Figure 2.2

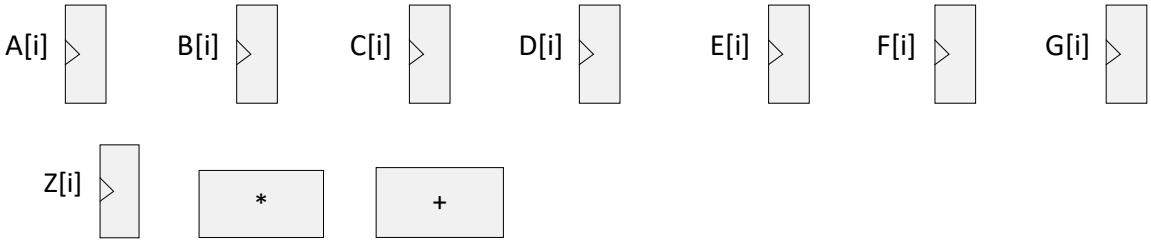
Question 3 (3 points) Synthèse de haut niveau et ordonnancement

Soit l'expression Z[i] = (A[i] + B[i] + C[i]) * (D[i]) + E[i] +F[i]) * G[i] qu'on désire implémenter en matériel dans une boucle for dans laquelle on insère dans un pragma *HLS PIPELINE ainsi que des pragma HLS ARRAY_PARTITION complete* pour chaque variable.

a) En supposant une contrainte de ressources de 2 additionneurs et 1 multiplieur[1], proposez le meilleur ordonnancement en dessinant les 2 premières itérations de boucle sur la figure 3.1. Donnez aussi le débit (II) et la latence (L).

b) En supposant une contrainte de ressource illimités, proposez le meilleur ordonnancement en dessinant les 2 premières itérations de boucle sur la figure 3.2. Donnez aussi le débit (II) et la latence (L).

Utilisez une ou plusieurs instances de blocs suivants pour compléter les figures 3.1 et 3.2 :



Solution dans les slides présentés au cours, je n'ai que changé la formulation.

[1] Bien qu'on n'a pas trouvé en classe quel pragma permettait d'imposer des contraintes de ressources, on suppose que celui-ci existe…

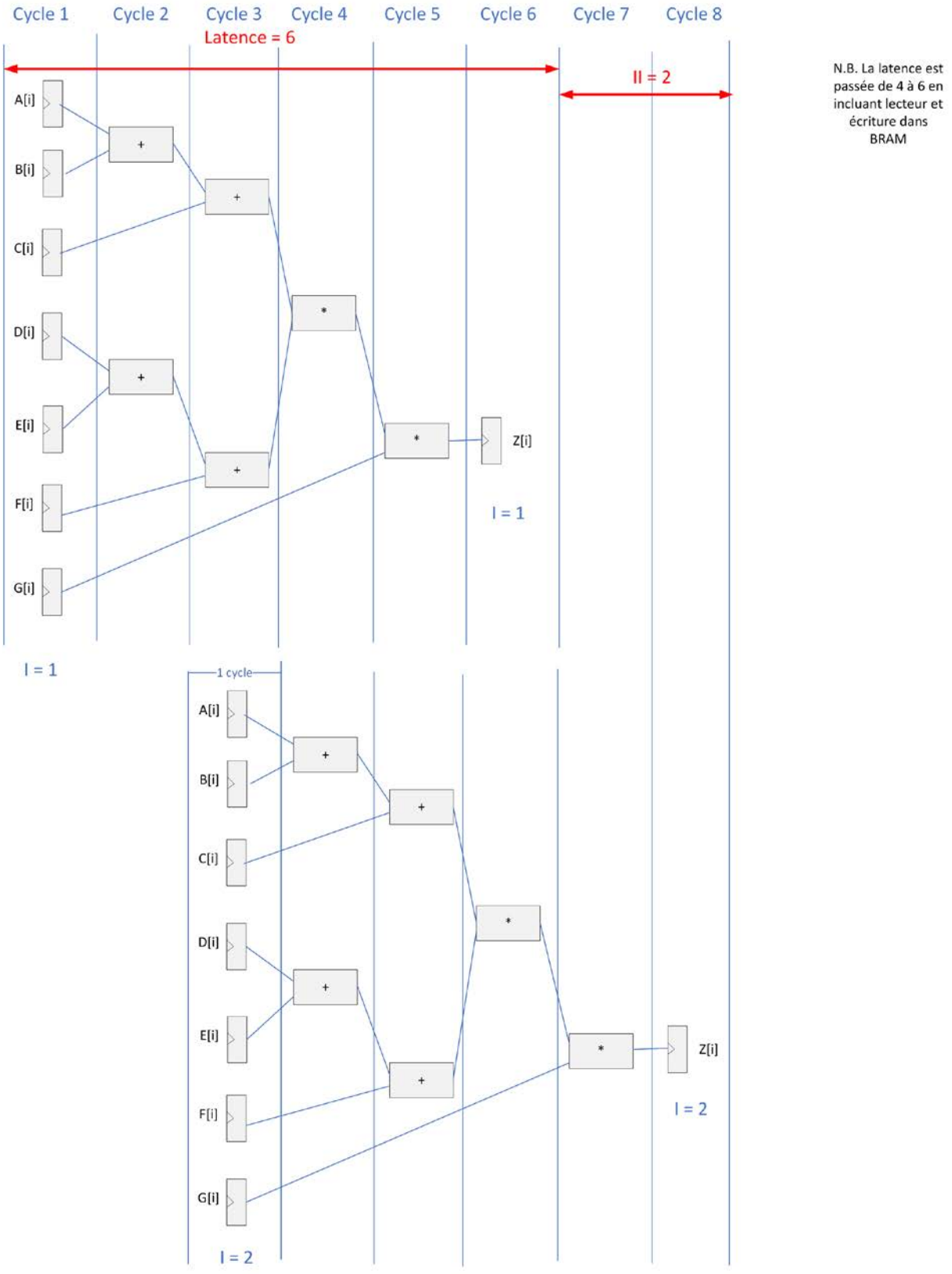

Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Cycle 8
Latence = 6
II = 2
N.B. La latence est passée de 4 à 6 en incluant lecteur et écriture dans BRAM
A[i]
B[i]
C[i]
D[i]
E[i]
F[i]
G[i]
Z[i]
I = 1
I = 1
1 cycle
A[i]
B[i]
C[i]
D[i]
E[i]
F[i]
G[i]
Z[i]
I = 2
I = 2

Figure 3.1

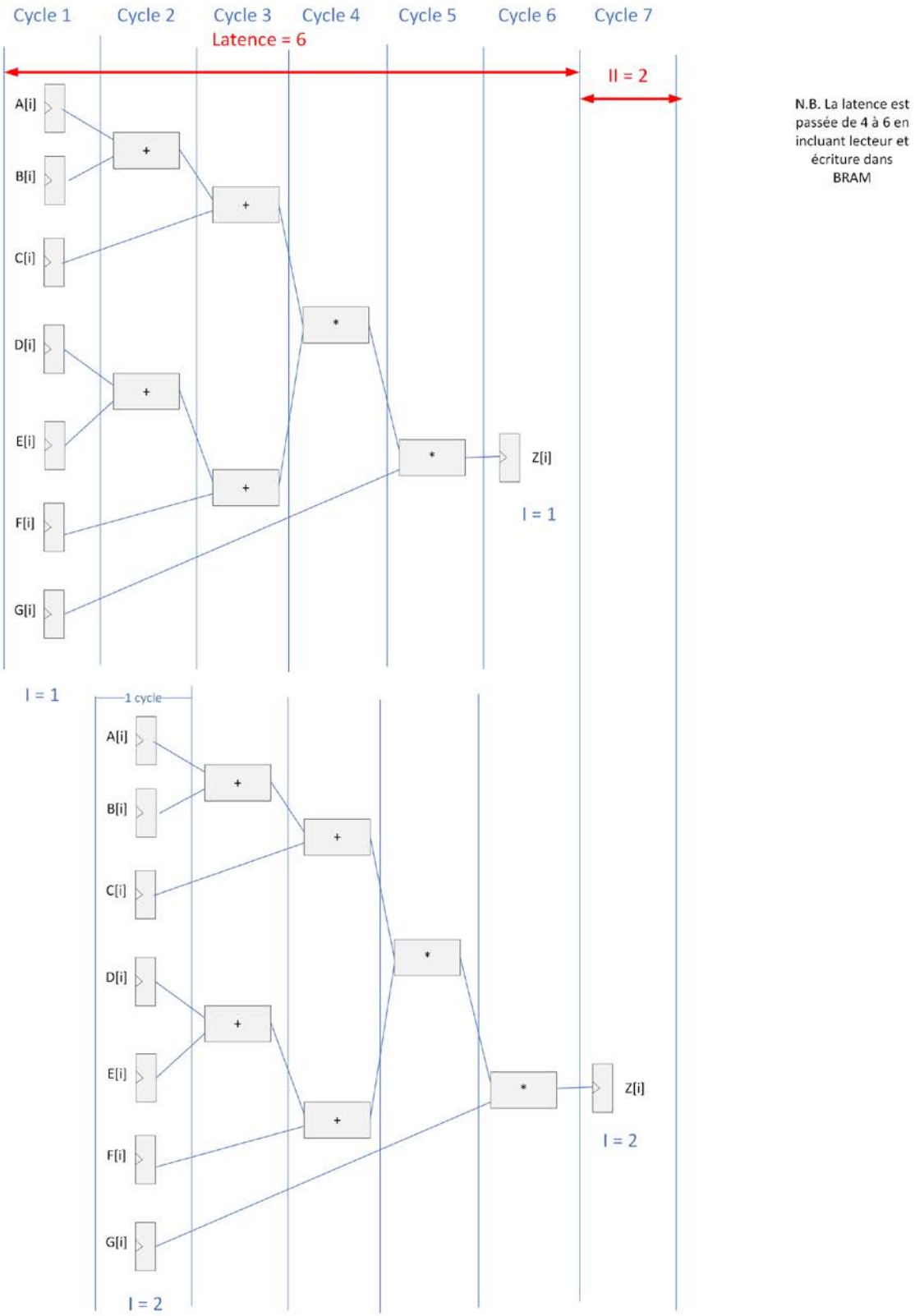

Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Latence = 6
II = 2
A[i]
B[i]
C[i]
D[i]
E[i]
F[i]
G[i]
Z[i]
I = 1
N.B. La latence est passée de 4 à 6 en incluant lecteur et écriture dans BRAM
I = 1
1 cycle
A[i]
B[i]
C[i]
D[i]
E[i]
F[i]
G[i]
Z[i]
I = 2
I = 2

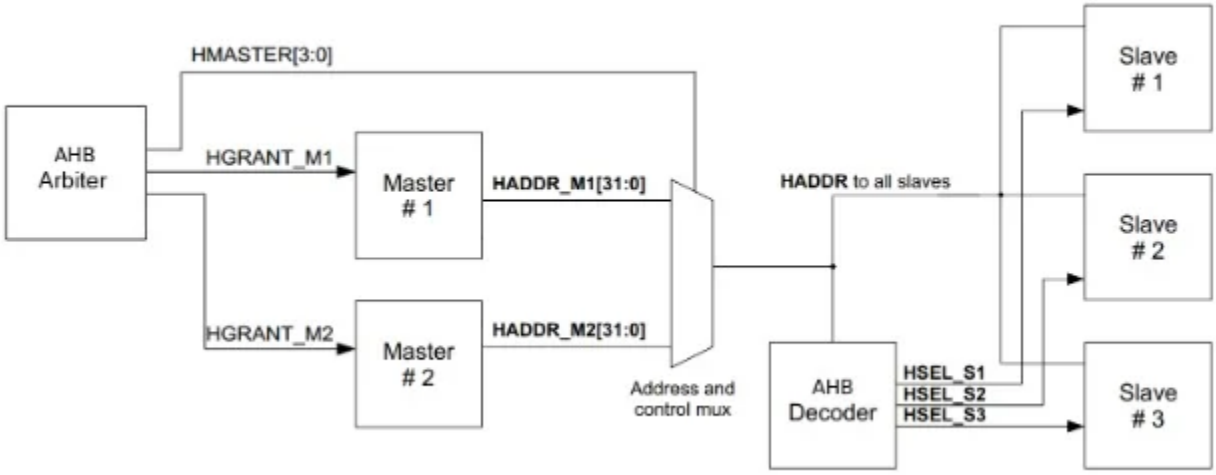
Figure 3.2

Question 4 (4 points) Bus et interconnexions

a) (1 pt) Expliquez la différence entre un bus et une interconnexion en vous référant au protocole AMBA de AXI pour système sur puce. Expliquez également lequel des deux (bus ou interconnexion) offre la meilleure contention, mais à quel prix?

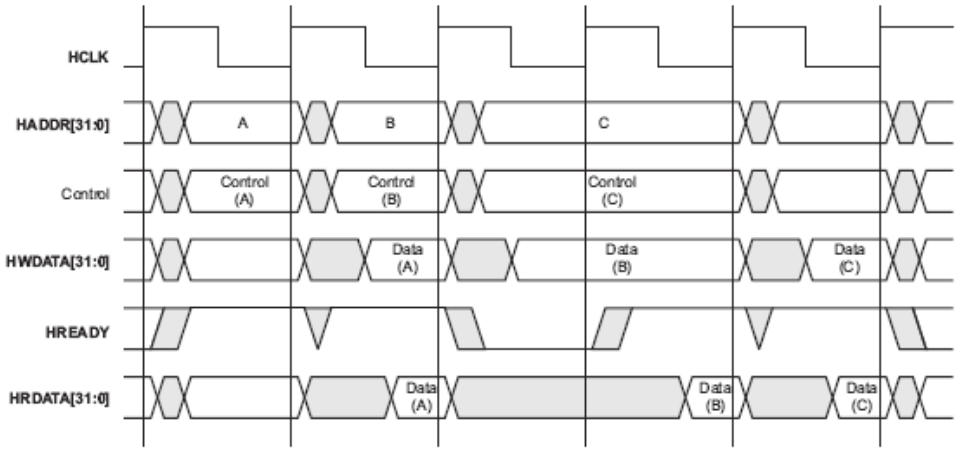
Interconnexion implique des connexions point à point donc nettement mieux qu'un bus. Mais l'interconnexion implique $O(n^2)$ connexions par rapport à O(constante) pour le bus. Mais les fpga étant de plus en plus gros en ressources l'interconnexion est de plus en plus utilisée.

b) (1 pts) Quelle partie d'une transaction AHB est représentée à la figure suivante et quelle partie est manquante (justifiez):



Il s'agit de la demande d'une requête à l'arbitre et de l'envoie d'une transaction (lecture ou écriture avec toutes les infos). Il manque cependant la réponse de l'esclave.

c) (1 pt) Que représente le schéma suivant (décrivez chaque cycle HCLK) :



Il s'agit d'une transaction en pipeline (on lit B pendant qu'on traite A), on voit aussi que C prend 2 wait states.

d) (.5 pt) Décrivez la différence entre AXI (aussi nommé AXI Full) et AXI Stream

5 canaux contre un seul pour le streaming.

e) (.5 pt) Décrivez la différence entre AXI Full et AXI Lite

AXI Full 5 canaux et le AXI Lite est une version très simplifié utilisé simplement pour initié des coprocesseurs comme le DMA. Il est superflu d'avoir un DMA pour faire ce travail.

Question 5 (2 points) Laboratoires

a) (1 point) Expliquez le rôle du *watchdog* dans le laboratoire no 1 et détaillez son mode de fonctionnement.

Le watchdog servait à stopper le système après x secondes d'arrêt sans repartir le système avec un bouton pressoir. Dès qu'on avait plus qu'un certain nombre de paquets erronés après 20 sec d'utilisation le système arrêtait de générer des paquets et le watchdog démarrait alors en mode one shot.

b) (1 point) On veut accélérer 120 fois un calcul de matrices carrées 42x42 sur FPGA avec Vivado HLS/Vivado/SDK. Proposez une architecture qui permettra une telle accélération. Quel SoC FPGA de la série Zynq 7000 est requis pour cela? Justifiez. Considérez que 1 multiplication/acc demande 5 DSP48E.

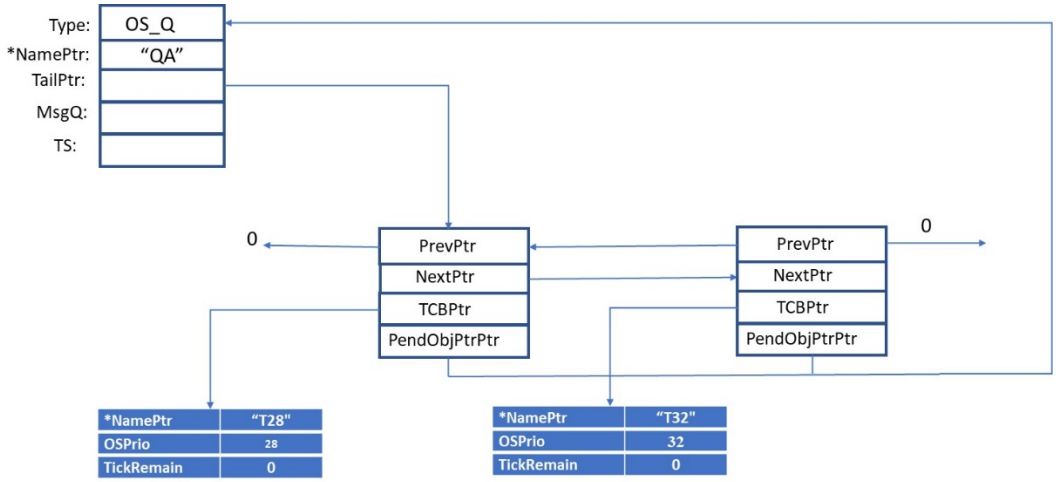
Le lab a montré qu'une accélération de 42 pour des floats. Si on fait 3 instances en parallèle dans Vivado avec 3 DMA et 3 interconnexions HP, on se rapproche de 120 et cela coutera 42x5x3 = 630 DSP donc un 7035 devrait suffire.

Question 6 (2 points) Gestion des évènements

Soit l'état d'un système sous uC/OS-III à la figure 6.1 à la fin du tick no 5.

a) (**1 pt**) On suppose que la tâche en exécution fait appel à *OSQPend(&QA, 2, OS_OPT_PEND_BLOCKING, &msg_size, &ts, &err)* au tick no 6. Donnez l'état du système à la fin du tick no 5, puis énumérez tous les évènements jusqu'à la fin de l'appel à *OSQPend(&QA, 2, OS_OPT_PEND_BLOCKING, &msg_size, &ts, &err).*

b) (**1 pt**) Nous sommes au début du tick no 8 et la tâche déterminée à la fin de l'évènement en a) est toujours celle en exécution. Décrivez les évènements en ce début de 8e tick ? Justifiez.



	0	1	2	3	4	5	6	7
[0]	0	0	0	0	0	0	0	0
[1]	0	0	0	1	0	0	0	0
[2]	0	0	0	0	0	0	0	0
[3]	0	0	0	1	0	1	0	0
[4]	0	0	0	0	0	0	0	0
[5]	0	0	0	0	0	0	0	0
[6]	0	0	0	0	0	0	0	0
[7]	0	0	0	0	0	0	0	1

OSPrioTbl

Figure 6.1 État du système au tick no 5 (les TCB en gris sont évidemment incomplets). Tickremain veut dire que les tâches sont en attente avec un délai infini.

Ce numéro est repris de l'intra (mal réussi) et j'ai changé le délai de l'attente à 2 dans OSQPend. Mais comme j'ai augmenté à 8 le tick (plutôt que 7) la réponse sera la même que dans l'intra).

Annexe

Détails des instructions pouvant être pipelinées	Nom de l'instruction	Nombre de cycles dans EX	Cycle du pipeline où l'opération termine
MOV R1, R5	**Copie R 1 dans R2**	**1**	**ER (le résultat est dans l'accumulateur après EX)**
LD F1, 0(R1)	**À partir de l'adresse contenue dans R1 auquel on additionne 0, chargement du double mot dans F1**	**1**	**ER (le résultat est dans l'accumulateur après MEM)**
ADDI R1, R1, #8	**Addition immédiate : R1 –> R1 + 8**	**1**	**ER (le résultat est mis dans l'accumulateur après EX)**
SLLI R8, #3	**Décalage à gauche de 3**	**1**	**ER (le résultat est dans l'accumulateur après EX)**
ADD R1, R1, R3	**Addition de deux mots : R1 –> R1 + R3**	**1**	**ER (le résultat est mis dans l'accumulateur après EX)**
MULT R8,R5,R1	**Multiplie 2 mots (32 bits) R8 <- R8 X R7**	**2**	**ER (le résultat est dans l'accumulateur après EX)**
ADDD F1, F1, F3	**Addition de deux doubles mots : F1 –> F1 + F3**	**3**	**ER (le résultat est mis dans l'accumulateur après E3)**
MULTD F1, F1, F5	**Multiplication de deux doubles mots : F1 –> F1 * F5**	**4**	**ER (le résultat est mis dans l'accumulateur après E6)**
SD 0(R2), F6	**Rangement d'un mot à partir de F6**	**1**	**MEM**
BNEQ R3, etiq	**Branch si non nul**	**1**	**EX**

Figure A1. Détail des instructions du DLX pour no 1.
Considérez qu'il s'agit d'un modèle M4 et qu'on a un seul port de mémoire aux étapes de LI et de ME du pipeline

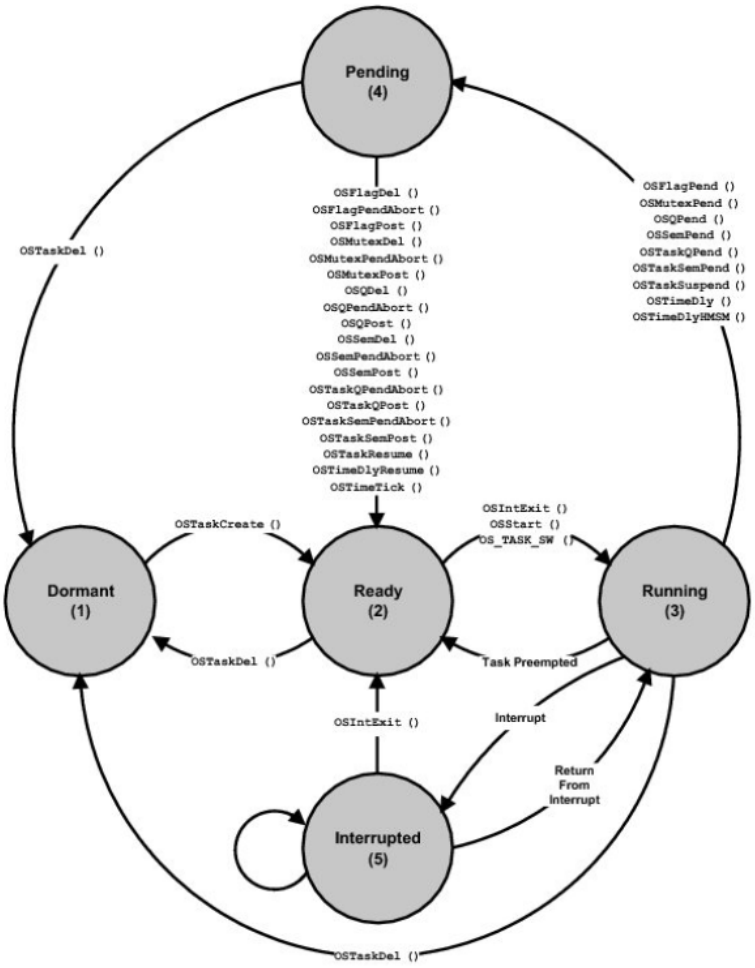

Pending
(4)
OSFlagDel ()
OSFlagPendAbort ()
OSFlagPost ()
OSMutexDel ()
OSMutexPendAbort ()
OSMutexPost ()
OSQDel ()
OSQPendAbort ()
OSQPost ()
OSSemDel ()
OSSemPendAbort ()
OSSemPost ()
OSTaskQPendAbort ()
OSTaskQPost ()
OSTaskSemPendAbort ()
OSTaskSemPost ()
OSTaskResume ()
OSTimeDlyResume ()
OSTimeTick ()
OSTaskDel ()
OSFlagPend ()
OSMutexPend ()
OSQPend ()
OSSemPend ()
OSTaskQPend ()
OSTaskSemPend ()
OSTaskSuspend ()
OSTimeDly ()
OSTimeDlyHMSM ()
OSTaskCreate ()
OSIntExit ()
OSStart ()
OS_TASK_SW ()
Dormant
(1)
Ready
(2)
Running
(3)
OSTaskDel ()
Task Preempted
OSIntExit ()
Interrupt
Return
From
Interrupt
Interrupted
(5)
OSTaskDel ()

Table 1: **Zynq-7000 and Zynq-7000S SoCs *(Cont'd)***

	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix®-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Kintex®-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA
	Programmable Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Block RAM (# 36 Kb Blocks)	1.8 Mb (50)	2.5 Mb (72)	3.8 Mb (107)	2.1 Mb (60)	3.3 Mb (95)	4.9 Mb (140)	9.3 Mb (265)	17.6 Mb (500)	19.2 Mb (545)	26.5 Mb (755)
	DSP Slices (18x25 MACCs)	66	120	170	80	160	220	400	900	900	2,020
	Peak DSP Performance (Symmetric FIR)	73 GMACs	131 GMACs	187 GMACs	100 GMACs	200 GMACs	276 GMACs	593 GMACs	1,334 GMACs	1,334 GMACs	2,622 GMACs
	PCI Express (Root Complex or Endpoint)(3)		Gen2 x4			Gen2 x4		Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security(2)	AES and SHA 256b for Boot Code and Programmable Logic Configuration, Decryption, and Authentication									