POLYTECHNIQUE MONTRÉAL

Questionnaire examen final

INF3610

Sigle du cours

Identification de l'étudiant(e)		
Nom :	**Prénom :**	
Signature :	**Matricule :**	**Groupe :**

Sigle et titre du cours	*Groupe*	*Trimestre*
INF3610 – Systèmes embarqués	**Tous**	**20203**
Professeur	***Local***	***Téléphone***
Guy Bois	**M-4115**	**5944**

Jour	*Date*	*Durée*	*Heures*
Mercredi	**5 mai 2021**	**2 h 30**	**13h30 à 16h00**

Documentation	*Calculatrice*	
☒ Toute	☐ Aucune	**Les cellulaires, agendas électroniques ou téléavertisseurs sont interdits.**
☐ Aucune	☐ Toutes	
☐ Voir directives particulières	☒ Non programmable (AEP)	

Directives particulières

Important

Cet examen contient **5** questions sur un total de **13** pages **(excluant cette page)**

La pondération de cet examen est de **30** **%**

- Les questions sont dans un fichier .docx et un .pdf qui vous parviendra par e-mail à 13h30
- Vous devez me remettre un .pdf
- Pour les tableaux à compléter, vous pouvez compléter directement avec le logiciel de dessin ou compléter à la main et retourner une numérisation ou photo jpeg.

À partir de 16 h, vous aurez 15 minutes pour transférer les documents sur Moodle (procédure similaire à celle d'une remise de devoir).

L'étudiant doit honorer l'engagement pris lors de la signature du code de conduite

Question 1 (5 points) Architectures RISC scalaire et aleas

Soit le code C++ (fig. 1.1):

```
int const DIM = 8;

int similar_to_L3 (float A[DIM], float B[DIM], float C[DIM]) {
float sum = 0 ;
   for (int i = 0; i < DIM; i=i+2)
      sum +=  (A[i] * (B[i]+C[i]) + A[i+1] * B[i+1]);
   return sum;
}
```

Figure 1.1

Une fois assemblé (fig. 1.2), on obtient le code suivant qui s'exécute sur un pipeline DLX à 5 niveaux :

```
L1 :    LD  F1, 0(R1)
        LD  F2, 0(R2)
        LD  F3, 0(R3)
        LD  F5, 0(R5)
        ADDD F2, F2, F3
        MULTD   F2, F1, F2
        LD  F3, 8(R1)
        LD  F4, 8(R2)
        MULTD   F4, F3, F4
        ADDD F6, F2, F4
        ADDD F5, F5, F6
        ADDI   R1, R1, #16
        ADDI   R2, R2, #16
        ADDI   R3, R3, #16
        ADDI   R4, R4, #2
        SEQI  R5, R4, #7
        BEQZ R5, L1
```

Figure 1.2

où:

- A, B et C sont des vecteurs de dimension 8 en doubles mots (8 octets de 8 bits) dont les adresses de départ sont déjà calculées et se trouvent en R1, R2 et R3, respectivement.
- On itère par incrément de 2 via R4 qui est initialisé à 0.
- F5 joue le rôle de l'accumulateur qui contient la valeur de *sum* et est inialisé à 0.
- On a au maximum 1 accès mémoire pour la lecture d'instruction (LI), ainsi que 1 accès pour la lecture/écriture de données.

Vous devez :

a) (1.5 point) Complétez la table 1.1 de la page suivante. Vous devez tenir compte des cycles de suspension ou d'attente, à cause d'aléas, en les indiquant par SU. Considérez un modèle M4. Finalement, donnez le nombre de cycles que prendra l'exécution complète. Référez-vous à la description des instructions assembleurs donnée en Annexe.

b) (2 points) Toujours en considérant un modèle M4, complétez la table 1.2, en déroulant la boucle en 4 itérations et en réorganisant les opérations pour enlever le maximum d'aléas. Si parfois des suspensions sont requises tentez de les remplacer par du code utile, sinon indiquez-les avec SU. Donnez le nombre de cycles que prendra l'exécution complète et comparez l'accélération par rapport à a). Finalement, est-ce que ce choix de 4 pour le déroulement était un bon choix? Justifiez.

c) (1.5 point) On suppose maintenant un processeur similaire au Cortex A9 superscalaire (M5) tel que vous avez sur le SoC de votre carte Cora Z7 (Processing Subsystem) avec les caractéristiques suivantes :

- 2 chargements et décodage d'instructions à la fois
- 2 unités entières du type ALU (pour instructions MSUBI, SLLI et BNEQ),
- 2 unités entières (pour instructions ADD, MULT),
- 1 unité de load/store de 128 bits de données en parallèle (ou de 4 mots de 32 bits ou de 2 mots de 64 bits) et
- 2 unités point flottantes (FP/Neon) capable de faire 2 opérations similaires (ADDD, MULTD, DIVD, etc.) sur des mots de 64 bits.

Complétez la table 1.3, en assignant chaque opération de b) à un cycle, pour ainsi accélérer encore davantage par rapport à b). Utilisez des flèches pour montrer les dépendances. Si parfois des suspensions sont requises tentez de les remplacer par du code utile, sinon indiquez-les avec SU. Finalement, donnez le nombre de cycles que prendra l'exécution complète et comparez l'accélération par rapport à a).

Remarque : Tout comme en a), supposez que les adresses de A, B et C sont déjà calculés et se trouvent en R1, R2 et R3, respectivement.

##		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	29	30	31
1	L:	LD F1, 0(R1)	*LI*	*DI*	*EX*	*ME*	*ER*																										
2		LD F2, 0(R2)		*LI*	*DI*	*EX*	*ME*	*ER*																										
3		LD F3, 0(R3)			*LI*	*DI*	*EX*	*ME*	*ER*																									
4		ADDD F2, F2, F3				*LI*	*DI*	*SU*	*E1*	*E2*	*E3*	*ME*	*ER*																				
5		MULTD F2, F1, F2					*LI*	*SU*	*DI*	*SU*	*SU*	*E1*	*E2*	*E3*	*E4*	*ME*	*ER*																
6		LD F3, 8(R1)							*LI*	*SU*	*SU*	*DI*	*EX*	*ME*	*ER*																		
7		LD F4, 8(R2)										*LI*	*DI*	*EX*	*ME*	*ER*																	
8		MULTD F4, F3, F4											*LI*	*DI*	*SU*	*E1*	*E2*	*E3*	*E4*	*ME*	*ER*												
9		ADDD F6, F2, F4												*LI*	*SU*	*DI*	*SU*	*SU*	*SU*	*E1*	*E2*	*E3*	*ME*	*ER*									
10		ADDD F5, F5, F6														*LI*	*SU*	*SU*	*SU*	*DI*	*SU*	*SU*	*E1*	*E2*	*E3*	*ME*	*ER*						
11		ADDI R1, R1, #16																		*LI*	*SU*	*SU*	*DI*	*EX*	*ME*	*ER*							
12		ADDI R2, R2, #16																						*LI*	*DI*	*EX*	*ME*	*ER*					
13		ADDI R3, R3, #16																							*LI*	*DI*	*EX*	*ME*	*ER*				
14		ADDI R4, R4, #2																								*LI*	*DI*	*EX*	*ME*	*ER*			
15		SEQI R5, R4, 64																									*LI*	*DI*	*EX*	*ME*	*ER*		
16		BEQ R5, L1																										*LI*	*DI*	*EX*			
17		Prochaine instruction																											*SU*	*SU*	*LI*		

Table 1.1 À compléter pour la question 1a)

28 cycles pour une itération donc 28 x 4 =112 instructions

1	LD F10, 0(R1)	**33**	ADDD F60, F20, F40
2	LD F11,8(R1)	**34**	ADDD F61, F21, F41
3	LD F12, 16(R1)	**35**	ADDD F62, F22, F42
4	LD F13, 24(R1)	**36**	ADDD F63, F23, F43
5	LD F20, 0(R2)	**37**	ADDD F50, F50, F60
6	LD F21, 8(R2)	**38**	ADDD F51, F51, F61
7	LD F22, 16(R2)	**39**	ADDD F52, F52, F62
8	LD F23, 24(R2)	**40**	ADDD F53, F53, F63
9	LD F30, 0(R3)	**41**	ADDD F50, F50, F51
10	LD F31, 8(R3)	**42**	su
11	LD F32, 16(R3)	**43**	ADDD F52, F52, F53
12	LD F33, 24(R3)	**44**	su
13	ADDD F20, F20, F30	**45**	su
14	ADDD F21, F21, F31	**46**	ADDD F50, F50, F52
15	ADDD F22, F22, F32	**47**	
16	ADDD F23, F23, F33	**48**	
17	MULTD F20, F10, F20	**49**	
18	MULTD F21, F11, F21	**50**	
19	MULTD F22, F12, F22	**51**	
20	MULTD F23, F13, F23	**52**	
21	LD F30, 8(R1)	**53**	
22	LD F31, 16(R1)	**54**	
23	LD F32, 24(R1)	**55**	
24	LD F33, 32(R1)	**56**	
25	LD F40, 8(R1)	**57**	
26	LD F41, 16(R1)	**58**	
27	LD F42, 24(R1)	**59**	
28	LD F43, 32(R1)	**60**	
29	MULTD F40, F40, F30	**61**	
30	MULTD F41, F41, F31	**62**	
31	MULTD F42, F42, F32	**63**	
32	MULTD F43, F43, F33	**64**	

Table 1.2 À compléter pour la question 1b)

On accumule dans F50

48 cycles pour une itération de 4 donc 112/46=2.43 d'accélération

Table 1.3 À compléter pour la question 1c)

Cycle	LD/SD 1	LD/SD 2	ALU1	ALU2	Entier1	Entier2	FP1/Neon1	FP1/Neon1
1 L1	LD F10, 0(R1)	LD F12, 16(R1)						
2	LD F11,8(R1)	LD F13, 24(R1)						
3	LD F20, 0(R2)	LD F30, 0(R3)						
4	LD F21, 8(R2)	LD F31, 8(R3)						
5	LD F22, 16(R2)	LD F32, 16(R3)					ADDD F20, F20, F30	
6	LD F23, 24(R2)	LD F33, 24(R3)					ADDD F21, F21, F31	
7	LD F30, 8(R1)	LD F40, 8(R1)					ADDD F22, F22, F32	
8	LD F31, 16(R1)	LD F41, 16(R1)					ADDD F23, F23, F33	MULTD F20, F10,F20
9	LD F32, 24(R1)	LD F42, 24(R1)					MULTD F40, F40, F30	MULTD F21, F11,F21
10	LD F33, 32(R1)	LD F43, 32(R1)					MULTD F41, F41, F31	MULTD F22, F12,F22
11							MULTD F42, F42, F32	MULTD F23, F13,F23
12							MULTD F43, F43, F33	
13								ADDD F60, F20, F40
14								ADDD F61, F21, F41
15								ADDD F62, F22, F42
16							ADDD F50, F50, F60	ADDD F63, F23, F43
17							ADDD F51, F51, F61	
18							ADDD F52, F52, F62	
19							ADDD F53, F53, F63	
20							ADDD F50, F50, F51	
21							SU	
22							ADDD F52, F52, F53	
23							SU	
24							SU	
25							ADDD F50, F50, F52	
26							SU	
27							SU	

27 cycles pour une itération de 4 donc 112/27=4.15 d'accélération

Question 2 (4 points) Synthèse de haut niveau et laboratoire no 2

a) (1 point) Quelles différences existent-ils entre un *pragma HLS unroll* et un *pragma HLS pipeline*. Expliquez les avantages et désavantages de chacun.

Le pragma HLS unroll effectue un déroulage de la boucle, ce qui équivaut à une parallélisation complète des opérations de la boucle. Ainsi, le même traitement est effectué sur toutes les données de façon simultanée, ce qui est très rapide en temps d'exécution (la latence totale équivaut au nombre de cycles requis pour une seule itération). Le désavantage est que le déroulage est très coûteux en ressources, puisque toutes les opérations identiques doivent se faire en même temps requiert d'avoir N ressources pour N itérations déroulées.

Le pragma HLS pipeline, pour sa part, effectue un pipelinage de la boucle, soit de séparer son traitement en plusieurs étapes avec des registres intermédiaires. De cette façon, de nouvelles données peuvent acceptées et traitées à chaque cycle, et un résultat est ainsi produit à chaque cycle une fois le premier résultat obtenu (à cause de la latence initiale d'itération). L'avantage est que le traitement est toujours beaucoup plus efficace que sans pipelinage, mais très peu de ressources additionnelles sont nécessaires, puisque toutes les opérations du pipeline ne sont pas exécutées en même temps. Ainsi, il n'est pas nécessaire d'avoir N ressources pour N itérations, contrairement au déroulage. Le désavantage est que le temps de traitement total est un peu plus long qu'un déroulage complet, mais il reste tout de même très performant.

b) (3 points) Soit le code suivant (fig. 2.1), une variante du laboratoire no 2 et pour lequel la boucle L3 est similaire à celle analysée à la question no1 :

```
int const DIM = 8;

void mmult_hw (float a[DIM][DIM], float b[DIM][DIM],
                   float c[DIM], float out[DIM][DIM])
{
    L1:for (int ia = 0; ia < DIM; ++ia)
       L2:for (int ib = 0; ib < DIM; ++ib)
       {
          float sum = 0;
          L3:for (int i = 0; i < DIM; i=i+2)
                sum += (a[ia][i] * (b[i][ib]+c[i]) + a[ia][i+1] * b[i+1][ib]);
                out[ia][ib] = sum;
       };
}
```

Figure 2.1

On veut pipeliner cette fonction en y ajoutant les pragma suivants pour lequel X, Y, Z, U, V et W sont des variables (fig. 2.2).

```
void mmult_hw (float a[DIM][DIM], float b[DIM][DIM],
                   float c[DIM], float out[DIM][DIM])
{
   #pragma HLS array_partition variable=a block factor=X dim=U
   #pragma HLS array_partition variable=b block factor=Y dim=V
   #pragma HLS array_partition variable=c block factor=Z dim=W
```

```
      L1:for (int ia = 0; ia < DIM; ++ia)
         L2:for (int ib = 0; ib < DIM; ++ib)
         {
            #pragma HLS PIPELINE II=1
            float sum = 0;
            L3:for (int i = 0; i < DIM; i=i+2)
                  sum += (a[ia][i] * (b[i][ib]+c[i]) + a[ia][i+1] * b[i+1][ib]);
                  out[ia][ib] = sum;
         };
}
```

Figure 2.2

b.1) Quelle valeur de X, Y, Z, U, V, W donnera pour L3 un débit de 1 résultat par cycle (II=1)? Justifiez votre réponse.

U=2, V=1, W=0. On veut briser le tableau a en colonnes pour accéder simultanément à plusieurs éléments d'une même ligne selon le débit demandé dans le pipelinage de la boucle L3. On veut briser le tableau b en lignes pour accéder simultanément à plusieurs éléments d'une même ligne selon le débit demandé dans le pipelinage de la boucle L3. Enfin, L3 n'est qu'un vecteur, donc on peut se permettre de le partitionner selon toutes les dimensions (dim=0). N.B. W=1 est également bon.

X=4, Y=4. Il faut donc partitionner en 4 blocs BRAM pour accéder à 8 données en parallèle.

Z=2. Il faut donc partitionner en 2 blocs BRAM pour accéder à 4 données en parallèle.

b.2) Quelle valeur de X, Y, Z, U, V, W donnera pour L3 un débit de 1 résultat par 2 cycles (II=2)? Justifiez votre réponse.

U=2, V=1, W=0. On veut briser le tableau a en colonnes pour accéder simultanément à plusieurs éléments d'une même ligne selon le débit demandé dans le pipelinage de la boucle L3. On veut briser le tableau b en lignes pour accéder simultanément à plusieurs éléments d'une même ligne selon le débit demandé dans le pipelinage de la boucle L3. Enfin, L3 n'est qu'un vecteur, donc on peut se permettre de le partitionner selon toutes les dimensions (dim=0). N.B. W=1 est également bon.

X=2, Y=2. Il faut donc partitionner en 2 blocs BRAM pour accéder à 4 données en parallèle.

Z=1. Il faut donc partitionner en 1 bloc BRAM pour accéder à 2 données en parallèle.

b.3) Quelle valeur de X, Y, Z, U, V, W donnera pour L3 un débit de 1 résultat par 4 cycles (II=4)? Justifiez votre réponse.

U=2, V=1, W=0. On veut briser le tableau a en colonnes pour accéder simultanément à plusieurs éléments d'une même ligne selon le débit demandé dans le pipelinage de la boucle L3. On veut briser le tableau b en lignes pour accéder simultanément à plusieurs éléments d'une même ligne selon le débit demandé dans le pipelinage de la boucle L3. Enfin, L3 n'est qu'un vecteur, donc on peut se permettre de le partitionner selon toutes les dimensions (dim=0). N.B. W=1 est également bon.

X=1, Y=1. Il faut donc partitionner en 1 bloc BRAM pour accéder à 2 données en parallèle.

Z=1. Il faut donc partitionner en 1 bloc BRAM pour accéder à 1 donnée en parallèle.

b.4) Identifiez parmi les 4 solutions de la figure 2.3 laquelle correspond à b.1), b.2) et b.3) et finalement quelle solution ne contient aucune optimisation. Justifiez bien vos choix.

Latency (clock cycles)

		solution1	solution2	solution3	solution4
Latency	min	5521	290	164	101
	max	5521	290	164	101
Interval	min	5521	290	164	101
	max	5521	290	164	101

Utilization Estimates

	solution1	solution2	solution3	solution4
BRAM_18K	0	0	0	0
DSP48E	8	12	24	48
FF	820	2880	3848	5029
LUT	1351	2838	4622	7729

Figure 2.3

Solution 1 = pas d'optimisation
Solution 2 = b.3)
Solution 3 = b.2)
Solution 4 = b.1)

Question 3 (3 points) Laboratoires

a) (1 point) Laboratoire no 1 Partie 1

Dans ce laboratoire, on vous a demandé de mettre une instruction d'attente active dans *TaskComputing()*. Il s'agissait de 3 ticks d'attente active. **La raison de cette dernière était d'émuler du temps d'exécution remplaçant un certain traitement à venir**.

Or, nous sommes au laboratoire no1 **hiver 2022** et on demande aux étudiants de faire un accélérateur matériel sur FPGA avec HLS Vivado **qui va réellement faire le traitement, émuler à l'hiver 2021, en calculant une valeur de checksum ou CRC** (somme de contrôle) sur chaque paquet généré de 64 octets[1]. On suppose que les communications pour transférer un paquet (64 octets à la fois) du processeur vers l'accélérateur sont similaires à ceux que vous avez conçu au laboratoire no 2 (DMA et AXI4).

Estimez une bonne supérieure d'exécution de l'accélérateur et du temps de communication pour un traitement de 64 paquets afin d'assurer que les fifos se vident complètement après 250 ms de délai entre 2 rafales de 0 à 255 paquets générés aléatoirement avec OS_CFG_TICK_RATE_HZ = 1000 Hz.

Également, puisqu'on ne veut pas augmenter la taille des différents fifos (i.e. les 4 fifos à 1024), indiquez quelle(s) tâche(s) devrai(en)t être bloquée(s) et quelle(s) tâche(s) pourrai(en)t poursuivre leur exécution durant l'appel à l'accélérateur checksums.

Expliquez bien votre démarche.

*3 ticks à 1000Hz ça fait 3 * 1/1000 = 3 ms*

On doit bloquer TaskGenerate et TaskComputing mais les autres peuvent poursuivre car elles ne font que vider les fifos.

b) (2 points) Laboratoire no 1 Partie 2

Décrivez les différentes étapes lors d'une interruption de type SPI venant du *fittimer* à partir du moment où vous appuyez sur le bouton et le moment où la tâche TaskStop démarre.

Il s'agissait d'une question du rapport de lab mais au lieu de l'appliquer au push button il s'agit du fittimer, donc même principe sauf que les numéros sur FPGA changent.

[1] Le générateur pouvant ainsi introduire volontairement des erreurs pour ensuite valider le checksum sur l'accélérateur. Aussi, notez que tout comme pour le calcul matriciel, le CRC peut être parallélisé.

Question 4 (4 points) Interconnexion et laboratoire no 2

a) (2 points) Décrire la transaction suivante (fig. 4.1) sur *AXI interconnect* en indiquant le rôle de chaque canal et ce qui se passe à chaque tick d'horloge.

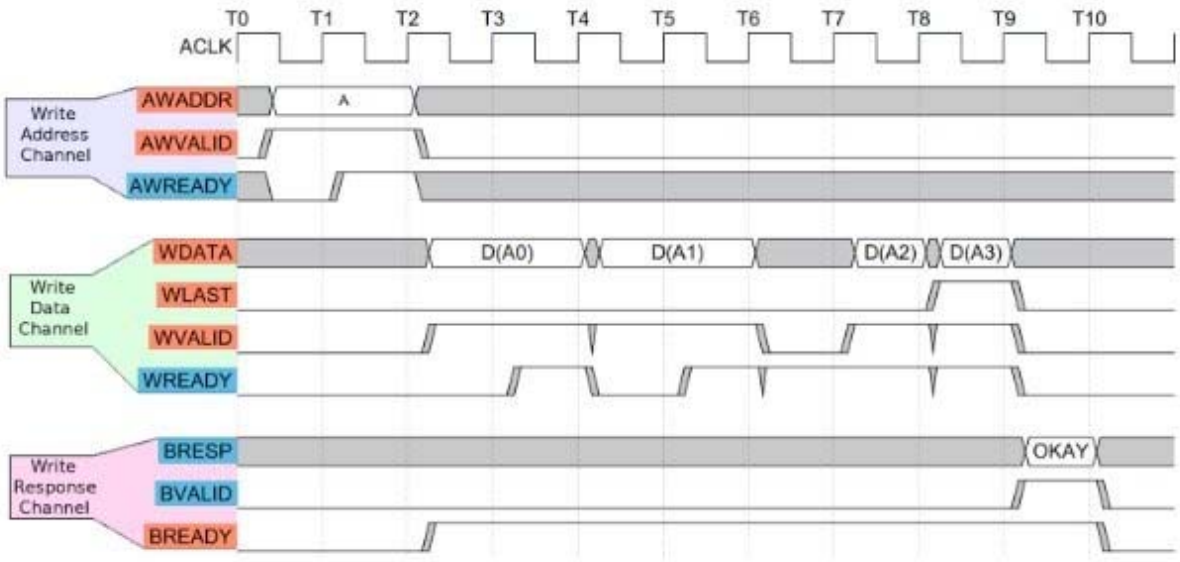


Figure 4.1

Vu au dernier cours il s'agit de l'écriture de 4 données via AXI standard

b) (2 points) On dit que le DMA peut être à la fois maître et esclave. Montrer une situation de ce double emploi en vous servant du schéma Vivado du laboratoire no 2, plus précisément en décrivant :

1) Le maitre impliqué lorsque le DMA est esclave ainsi que le type d'interconnexion (AXI ou AXI Stream ou AXI Lite).

2) Le ou les esclaves lorsque le DMA est maitre ainsi que le(s) type(s) d'interconnexion AXI utilisé(s) (AXI ou AXI Stream ou AXI Lite).

Au besoin, référez-vous aux schémas de la page suivante, soit la figure 4.2 (qui est en fait celle du lab. 2) et la figure 4.3 (DMA tiré des slides du bloc 3).

En tant que slave, le DMA est programmé par le processeur avec AXI lite.

En tant que master, le DMA est maitre et HLS_accel est esclave avec AXI Stream et même chose entre DMA et DDR.

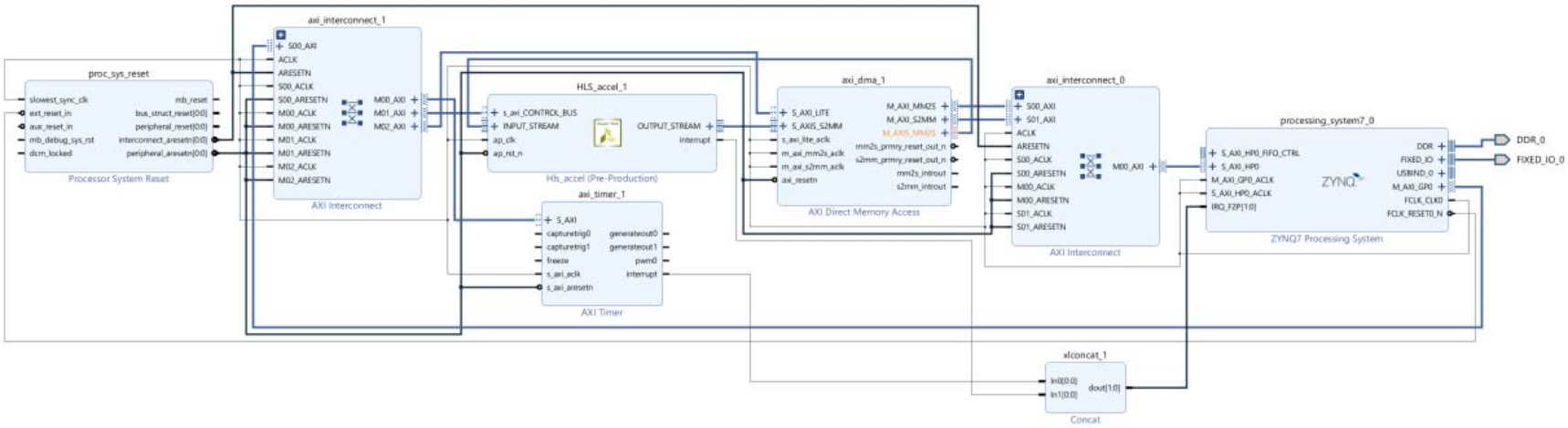


Figure 4.2 (Schéma Vivado de votre laboratoire no 2)

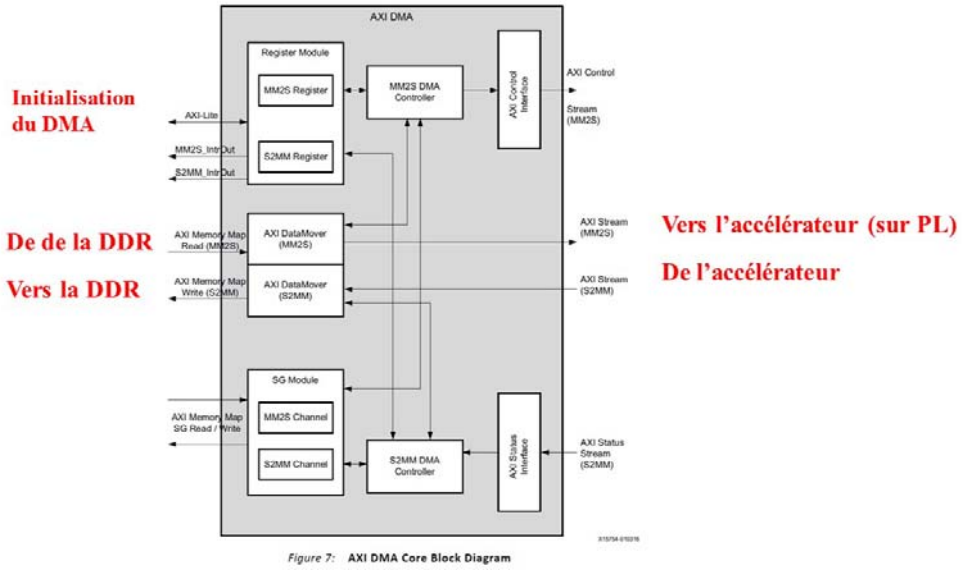
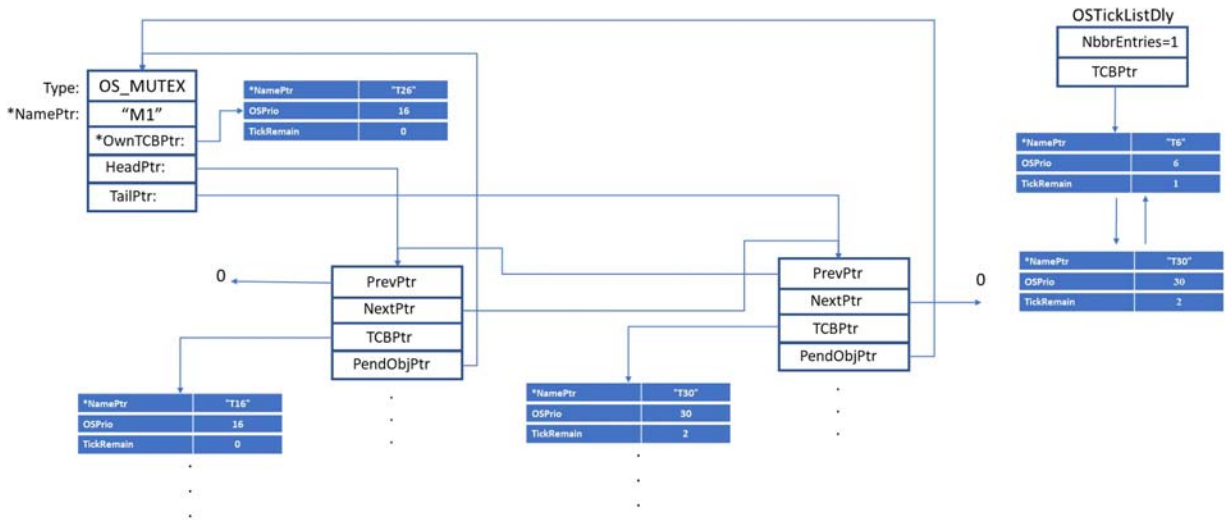


Figure 7: **AXI DMA Core Block Diagram**

Figure 4.3 (slide 87 du bloc 3)

Question 5 (4 points)

Soit un système sous uC/OS-III qui démarre (*OSStart()*) et qui exécute 3 ticks, de sorte que nous sommes à la toute fin du tick no 3, juste avant l'interruption qui va nous amener au tick no 4. Le système comprend T6, T16, T26, T30, T36, T42 et T63. L'état du système à ce moment-là est illustré à la figure 5.1. Notez que T6 et T30 sont en attente d'un délai.



	0	1	2	3	4	5	6	7
[0]	0	0	0	0	0	0	0	0
[1]	0	0	0	0	0	0	0	0
[2]	**1**	0	0	0	0	0	0	0
[3]	0	0	0	0	0	0	0	0
[4]	0	0	0	0	**1**	0	0	0
[5]	0	0	**1**	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 5.1 États d'un système à la toute fin du tick no 3, juste avant l'interruption qui va nous amener au tick no 4.

a) (1 point) Décrivez l'état du système tel que présenté à la Figure 5.1, c'est-à-dire les différentes tâches en cours (sous forme Ti) et leur état à la toute fin du tick no 3, juste avant l'interruption qui va nous amener au tick no 4. Indiquez aussi la tâche en cours d'exécution et comment celle-ci a été déterminée. Les états du RTOS sont présentés à la figure 5.2.

b) (1.5 point) Nous sommes maintenant au début du tick no 4 c'est-à-dire juste après l'interruption de la minuterie. La tâche la plus prioritaire trouvée en a) étant celle qui s'exécutait jusqu'à ce qu'elle soit interrompue au début du tick no 4. Indiquez ce qui se passe au niveau de ce ISR no 4 de la minuterie ainsi que pour la fonction OSIntExit() qui termine ce ISR. Vous devez indiquer :

- S'il y a lieu, les changements des états des tâches.
- Les détails internes de ce qui se passe durant la sous-routine d'interruption
- S'il y a lieu, les changements de contexte (suite à l'appel de l'ordonnanceur), en indiquant la prochaine tâche à être exécuté.

c) (1.5 point) On suppose ensuite, et toujours au tick no 4, que la tâche trouvée en b) fait appel à OSMutexPend(&M1, 2, OS_OPT_PEND_BLOCKING, &ts, &err). Ici indiquez, dans un ordre chronologique, ce qui se passe lors de cet appel i.e. :

- S'il y a lieu, les changements des états des tâches.
- Les détails internes de ce qui se passe dans le noyau lorsque que les fonctions pend et post sont appelées c'est-à-dire décrire ce qui se passe au niveau des structures et des appels système.
- S'il y a lieu, les changements de contexte (à la suite de l'appel de l'ordonnanceur), en indiquant la prochaine tâche à être exécuté (pas besoin ici de me réexpliquer comment la tâche prochaine tâche à exécuter est déterminée puisque c'est demandé en a).

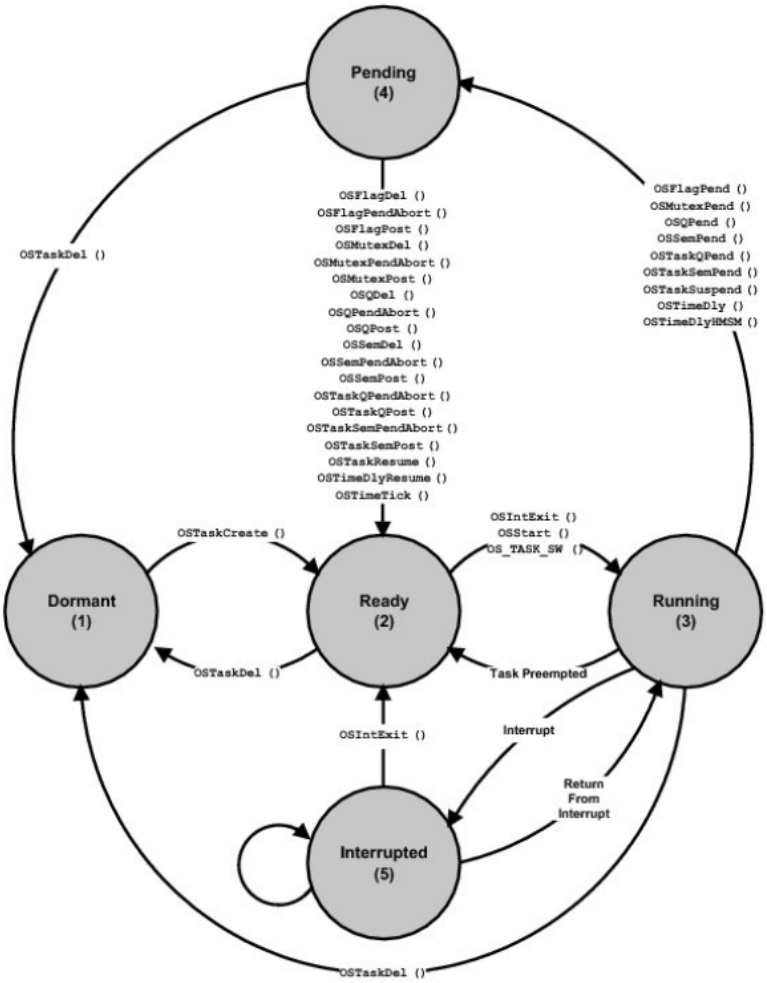


Figure 5.2 5 états d'un système sous uC/OS-III (tirée du cours no 3 et 4).

a)

T26 est en exécution après avoir héritée de T16 qui elle est en attente. T30 est aussi en attente de M1 et T6 est en attente d'un délai. Les autres tâches sont prêtes.

*Pour trouver T26 on a utilisé la fonction OSPrioGetHighest() qui passe les lignes de 0 à 7. Pour chaque ligne, elle regarde si la ligne 0 = 0. Si oui, elle incrémente donc l'indice de 1 et passe à la ligne suivante. Sinon (ligne différente de 0). La fonction CPU_CntLeadZeros va trouver le 1er bit à 1 en partant de la colonne 0. Par conséquent, dans notre exemple, on aura une ligne non nul à la ligne 2 sur laquelle CPU_CntLeadZeros retournera la colonne 0. Ce qui fait (8*2)+0 = 16.* ***La tâche la plus prioritaire 16 qui est en faite la tâche 26.***

b)

L'ISR appelle OSTimeTick().

On retire le contexte de T16

On regarde met à jour OSTickListDly (décrémente tous les délais de 1 tick), et on enlève la tâche 6 car le TickRemain correspondant est rendu à 0.

Via OS_PrioInsert(), T6 est mise à 1 dans OSPrioTbl[0][1], elle devient READY avec une priorité de 6.

Le traitement de OSTimeTick se termine.

On appelle OSIntExit() :

> *Désactive les interruptions, puis décrémente OSIntNestingCtr*
>
> *Puisque OSIntNesting arrive à 0 ça veut dire qu'il n'y plus d'interruption,*
>
> ***On appelle OS_PrioGetHighest(). La tâche la + prioritaire est T6 (prio=6)***
>
> *Tâche 6 passe de READY -> RUNNING*

OSIntCtxSw() est appelée, on prend le contexte de T6 et on le met sur le CPU et elle prend le relai !

c)

On regarde si le mutex est libre, ce qui n'est pas le cas.

Comme il y a un timeout, on insère la tâche T6 dans la tick list avec OS_TickListInsert() et aussi on met T6 dans dans la liste d'attente OS_PEND_OBJ du mutex juste avant T8 car elle est plus prioritaire (c'est une liste triée par ordre de priorité).

Via OS_PrioRemove(), on met T6 à 0 dans OSPrioTbl[1][1].

La tâche T6 passe donc de RUNNING à PENDING

Le owner du mutex (T26 qui a hérité d'une priorité de 16) hérite maintenant de la priorité de T6 qui est 6.

On appelle ensuite OSSched(), qui appelle OS_PrioGetHighest().

La tâche la + prioritaire est T26. T26 passe dès lors de READY -> RUNNING

On appelle ensuite OSCtxSw(), il y'a changement de contexte et T26 devient la tâche courante en exécution.

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**	**1**	**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)**
SUBD 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 E4)**
DIVD F1, F1, F5	**Division de deux doubles mots : F1 –> F1 / F5**	**5**	**ER (le résultat est mis dans l'accumulateur après E5)**
SD 0(R2), F6	**Rangement d'un mot à partir de F6**	**1**	**MEM**
SEQI R5, R4, #val	**si (R4 = #val) alors R5 <= 1 sinon R5 <= 0**	**1**	**ER (le résultat est mis dans l'accumulateur après EX)**
BEQ R3, etiq	**Branch si nul**	**1**	**EX**

Notes aussi que :

- Une seule lecture d'instruction par cycle est permise et une seule lecture/écriture de données par cycle est permise
- On peut avoir plusieurs opérations en parallèle dans EX mais pas dans les autres étages.

Figure A1. Détail des instructions du DLX pour no 1