

QUESTION #1 (4 points) Architecture RISC scalaire et aléas

Soit un pipeline DLX de type M4 à cinq (5) niveaux et soit la boucle suivante (Figure 1.1) qui réalise l'opération vectorielle $A[i] = (A[i] * B[i] / C[i]) + (D[i] / E[i]) + F[i]$ où A, B, C, D, E et F sont des vecteurs de dimension 120 de mots doubles (point flottant). Les éléments de A, B, C, D, E et F sont chargés dans F10, F20, F30, F40, F50 et F60 respectivement, à partir des indices R1, R2, R3, R4, R5 et R6 qui eux sont initialisés respectivement par 960, 1960, 2960, 3840, 4800, 5760 et 6720.


L:	LD	F10, 0(R1)	
	LD	F20, 0(R2)	
	MULTD	F20, F20, F10	
	LD	F30, 0(R3)	
	DIVD	F30, F20, F30	
	LD	F40, 0(R4)	
	LD	F50, 0(R5)	
	DIVD	F40, F40, F50	
	ADDD	F40, F30, F40	
	LD	F60, 0(R6)	
	ADDD	F40, F40, F60	
	SD	0(R1), F40	
	SUBI	R1, R1, #8	 Remplacer par MSUBI R1, R6, #8 pour sauver de la place sur la fig 1.2
	SUBI	R2, R2, #8	
	SUBI	R3, R3, #8	
	SUBI	R5, R5, #8	
	SUBI	R4, R4, #8	
	SUBI	R6, R6, #8	
	BNEQZ	R1, L	

Figure 1.1 Assembleur de l'opération vectorielle

- a) (2 pts) Complétez la figure 1.1 page 3. Vous devez tenir compte des cycles de suspension ou d'attente. Considérez un modèle M4. Finalement, donnez le nombre de cycles pour 120 itérations.

Réponse du nombre de cycles : 28 cycles par itération donc $28 \times 120 = 3360$ au total

- b) (2 pts) À partir du résultat de la figure 1.1, complétez la figure 1.2 (page 4). Pour cela, optimisez le code en a) en déroulant au besoin la boucle et en réorganisant au besoin le code, tout en faisant disparaître du même coup le maximum de suspensions. Expliquez bien votre démarche, calculez l'accélération et comparez cette dernière par rapport à a).

Suggestion : pour minimiser l'écriture, vous pouvez ne mettre que la première et la dernière instruction d'un déroulement. Par exemple, si vous déroulez 8 fois, mettre la première et la huitième avec 6 espaces blancs entre les 2. Mais ne vous trompez pas sur les indices...

*Réponse du nombre de cycles : 62 cycles * 24 = 1488 cycles donc un gain de $3360/1488 = 2.258$*

##	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
1	L: LD F10, 0(R1)	LI	DI	EX	ME	ER																									
2	LD F20, 0(R2)		LI	DI	EX	ME	ER																								
3	MULTD F20, F10, F20			LI	DI	ST	E1	E2	E3	E4	ME	ER																			
4	LD F30, 0(R3)				LI	ST	DI	EX	ME	ER																					
5	DIVD F30, F20, F30						LI	DI	ST	ST	E1	E2	E3	E4	E5	ME	ER														
6	LD F40, 0(R4)							LI	ST	ST	DI	EX	ME	ER																	
7	LD F50, 0(R5)										LI	DI	EX	ME	ER																
8	DIVD F40, F40, F50											LI	DI	ST	E1	E2	E3	E4	E5	ME	ER										
9	ADDD F40, F30, F40												LI	ST	DI	ST	ST	ST	ST	E1	E2	E3	ME	ER							
10	LD F60, 0(R6)														LI						DI	EX	ME	ER							
11	ADDD F40, F40, F60																			LI	DI	ST	E1	E2	E3	ME	ER				
12	SD 0(R1), F40																				LI	ST	DI	EX	SU	SU	ME				
13	MSUBI R1, R7, #8																						LI	DI	SU	SU	EX	ME	ER		
14	BNEQZ R1, L																							LI	SU	SU	DI	ST	ST	EX	
15																											LI	ST	ST	ST	LI
16																															

Figure 1.2 À compléter pour la question 1a)

N.B. On aurait pu aussi supposer qu'à la fin du cycle 25 MSUBI met le résultat dans un accumulateur afin que BNEQZ n'attende pas la fin du cycle 29 avant de connaître le résultat du branchement ou pas. En effet, dans un tel cas BNEQZ pourrait décider dès la fin du cycle 26 et la prochaine itération de boucle démarrerait au cycle 27. C'est d'ailleurs ce que j'ai fait dans le no 1 de h2023 (voir fig. 1.1).

Cycle	Pipeline 1		Pipeline 1 (suite)
1	<i>LD F10, 0(R1)</i>	35	<i>LD F54, -32(R5)</i>
2	<i>LD F11, -8(R1)</i>	36	<i>DIVD F40, F40, F50</i>
3	<i>LD F12, -16(R1)</i>	37	<i>DIVD F41, F41, F51</i>
4	<i>LD F13, -24(R1)</i>	38	<i>DIVD F42, F42, F52</i>
5	<i>LD F14, -32(R1)</i>	39	<i>DIVD F43, F43, F53</i>
6	<i>LD F20, 0(R2)</i>	40	<i>DIVD F44, F44, F54</i>
7	<i>LD F21, -8(R2)</i>	41	<i>ADDD F40, F30, F40</i>
8	<i>LD F22, -16(R2)</i>	42	<i>ADDD F41, F31, F41</i>
9	<i>LD F23, -24(R2)</i>	43	<i>ADDD F42, F32, F52</i>
10	<i>LD F24, -32(R2)</i>	44	<i>ADDD F43, F33, F43</i>
11	<i>MULTD F20, F10, F20</i>	45	<i>ADDD F44, F34, F44</i>
12	<i>MULTD F21, F11, F21</i>	46	<i>LD F60, 0(6)</i>
13	<i>MULTD F22, F12, F22</i>	47	<i>LD F61, -8(R6)</i>
14	<i>MULTD F23, F13, F23</i>	48	<i>LD F32, -16(R6)</i>
15	<i>MULTD F24, F14, F24</i>	49	<i>LD F63, -24(R6)</i>
16	<i>LD F30, 0(R3)</i>	50	<i>LD F64, -32(R6)</i>
17	<i>LD F31, -8(R3)</i>	51	<i>ADDD F40, F40, F60</i>
18	<i>LD F32, -16(R3)</i>	52	<i>ADDD F41, F41, F61</i>
19	<i>LD F33, -24(R3)</i>	53	<i>ADDD F42, F42, F62</i>
20	<i>LD F34, -32(R3)</i>	54	<i>ADDD F43, F43, F63</i>
21	<i>DIVD F30, F20, F30</i>	55	<i>ADDD F44, F44, F64</i>
22	<i>DIVD F31, F21, F31</i>	56	<i>SD 0(R1), F40</i>
23	<i>DIVD F32, F22, F32</i>	57	<i>MSUBI R1, R7, #40</i>
24	<i>DIVD F33, F23, F33</i>	58	<i>SD 32(R1), F41</i>
25	<i>DIVD F34, F24, F34</i>	59	<i>SD 24(R1), F42</i>
26	<i>LD F40, 0(R4)</i>	60	<i>BNEQZ R1, L</i>
27	<i>LD F41, -8(R4)</i>	61	<i>SD 16(R1), F43</i>
28	<i>LD F42, -16(R4)</i>	62	<i>SD 8(R1), F44</i>
29	<i>LD F43, -24(R4)</i>	63	
30	<i>LD F44, -32(R4)</i>	64	
31	<i>LD F50, 0(R5)</i>	65	
32	<i>LD F51, -8(R5)</i>	66	
33	<i>LD F52, -16(R5)</i>	67	
34	<i>LD F53, -24(R5)</i>	68	

Figure 1.3 À compléter pour la question 1b)

QUESTION #2 (3 points) Architecture RISC scalaire et aléas

- a) (1 pt) Que représente le schéma de la figure 2.1 ? Expliquez.
- b) (1 pt) Soit **les 3 types** d'aléas de données
- **LAE** (lecture avant écriture): le conflit survient lorsque j essaie de lire la source avant que i ne l'ait écrite: j obtient alors l'ancienne valeur.
 - **EAL** (écriture avant lecture): le conflit survient lorsque j écrit dans une destination avant qu'elle ne soit lue (complètement) par i . i obtient alors par erreur la nouvelle valeur.
 - **EAE** (écriture avant écriture): le conflit survient lorsque j écrit un opérande avant qu'il ne soit écrit par i . L'écriture se fait alors dans le mauvais ordre, et laisse dans la destination la valeur écrite par i plutôt que celle écrite par j .

Choisissez 2 parmi les 3, indiquez vos choix et pour chaque choix illustrez l'aléas par un exemple de code DLX.

- c) (1 pt) Décrivez ce qu'est une dépendance ou aléas de contrôle ?

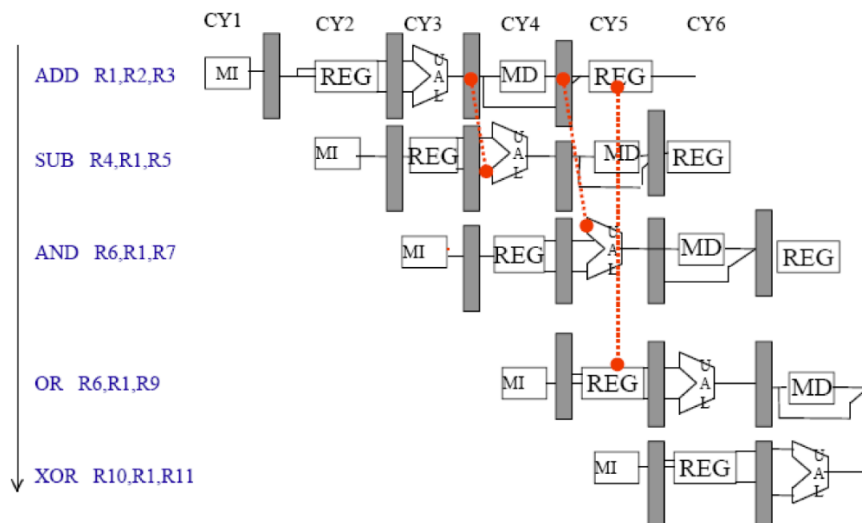


Figure 2.1

- a) Ça été expliqué en classe et présenté dans la solution du no 1 des exercices sur bloc1.
- b) Le classique est LEA avec :

```

LD      F0, 0(R1)
        |
ADDD    F4, F0, F2
        |
SD      0(R1), F4

```

Et EAE :

```

LD      F0, 0(R1)
DIV     F4, F0, F2
        |
SD      0(R1), F4
        |
LD      F0, 0(R1)
        |
ADD     F4, F0, F2
        |
SD      0(R1), F4

```

- c) Vu en classe et de nombreux exercices :

```

...
SUBI    R1, R1, #8
BEQZ    R1, sortie
        ?
        |
LD      F6, 0(R1)
ADDD    F8, F6, F2
SD      0(R1), F8
...
sortie LD      F10, 0(R10)

```

Ici on devra attendre le 3e stage de EX pour faire le bon branchement, ce qu'on nomme un branchement avec délai (2 SU).

QUESTION #3 (4 points) HLS et optimisation

Soit l'expression opération vectorielle $A[i] = (A[i] * B[i] / C[i]) + (D[i] / E[i]) + F[i]$ de la question no 1 que l'on souhaite maintenant implémenter avec Vivado HLS. On utilise donc pour cela le pragma HLS pipeline.

- a) On suppose une contrainte de ressources de 3 additionneurs, 1 multiplieur et 2 diviseurs et 4 BRAM **dual port** disponibles pour mémoriser les données de chaque itération. On demande $II = 1$. Considérez que l'additionneur, le multiplieur et la division prennent 1 cycle.

a.1) Faites le schéma des 6 premiers cycles sur la figure 3.1

a.2) Indiquez si $II = 1$ a été rencontré et donnez aussi la latence.

Pour a) utilisez les symboles suivants :



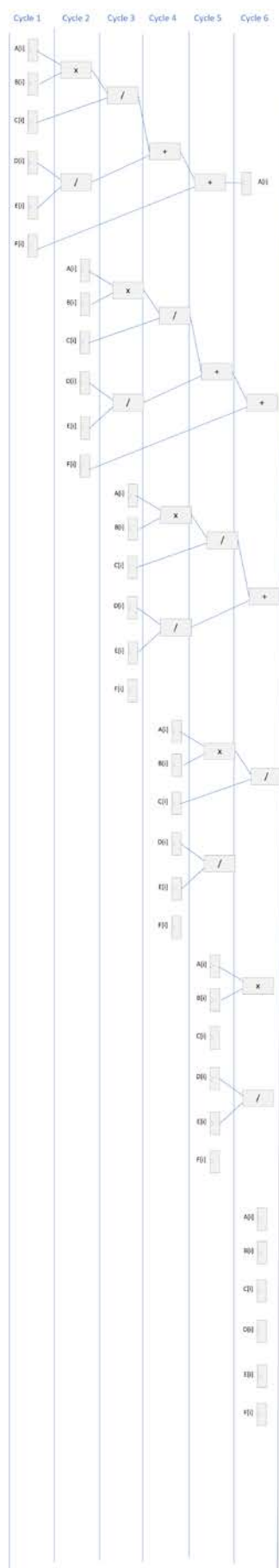
- b) On suppose une contrainte de ressources de 3 additionneurs, 1 multiplieur et 2 diviseurs et 4 BRAM **simple port** disponibles pour mémoriser les données de chaque itération. On demande $II = 1$. Considérez que l'additionneur, le multiplieur et la division prennent 1 cycle.

b.1) Sans faire de schéma, indiquez si $II = 1$ va être rencontré sinon qu'elle sera sa valeur ? Expliquez. Donnez aussi la latence.

- c) On suppose une contrainte de ressources de 3 additionneurs, 1 multiplieur et 2 diviseurs et 4 BRAM dual port disponibles pour mémoriser les données de chaque itération. Cette fois, considérez que l'additionneur prend 3 cycles, le multiplieur 4 cycles et la division 5 cycles mais que ces 3 opérations sont « pipelinable » tout comme pour l'étage EX du modèle M4.

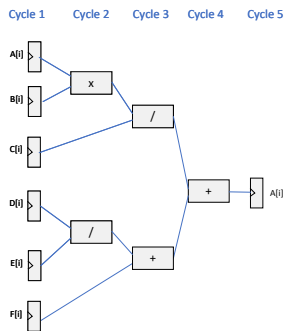
c.1) Sans faire de schéma, indiquez si $II = 1$ va être rencontré, sinon qu'elle sera sa valeur ? Expliquez. Donnez aussi la latence.

c.2) Estimez le temps d'exécution de cette solution pour 120 itérations ~~et donnez l'accélération par rapport au no 1a).~~



Solution figure 3.1 pour no 3a.1)

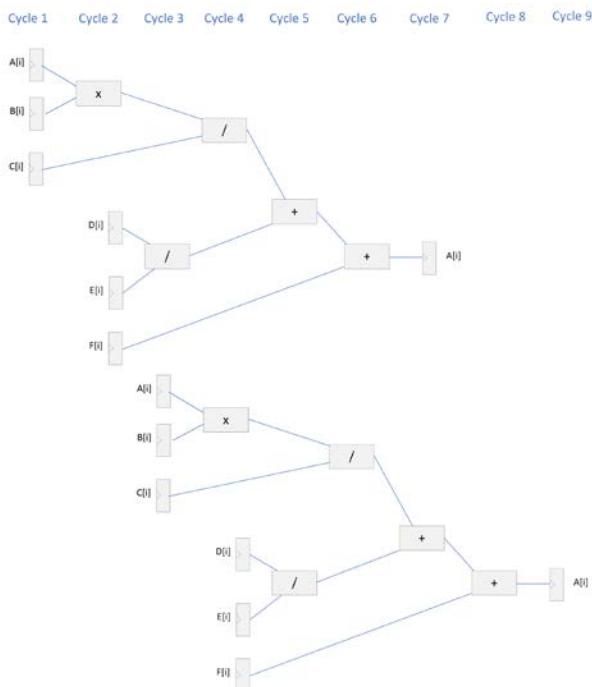
Remarque : Si on n'évalue pas de gauche à droite l'expression, on peut optimiser à 5 cycles (comparer la fig suivante avec fig. 3.1). Les 2 solutions sont équivalentes et acceptables.



À compléter comme à la fig. 3.1...

*a.2) 6 cycles de latences et la cible de $II=1$ est rencontrée (n.b. on a suffisamment de ressources *, 1 et + et on 3 BRAM en entrée au cycle 1 pour 6 entrées et 1 BRAM en sortie pour 1 sortie au cycle 6).*

b.1) Avec seulement 3 ports en entrée au cycle 1, on va fournir 3 entrées à la fois, ça prendra 2 cycles de lecture et on ne peut pas sortir un résultat à tous les cycles, donc $II=2$. La latence passe à 7 cycles. Ce n'était pas demandé, mais j'ai fait un début de schéma :



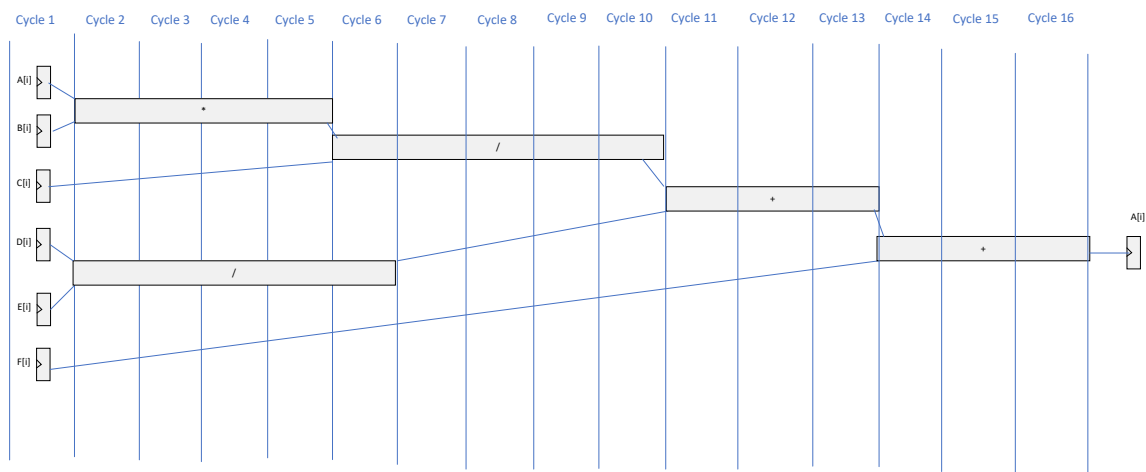
Etc.

Notez que comme on a 2 diviseurs on pourrait fusionner les cycles 3 et 4 et ramener la latence à 6 mais le débit reste à $II=2$ (c'est le paramètre important ici).

c.1) On a $II=1$ car nous avons les mêmes ressources qu'en a). Par contre on a une latence plus grande à cause du pipelining et ce même si le débit est conservé.

*Donc $II=1$ et si on prend la chaîne la plus longue sur la figure 3.1 on aura une latence = 1 cycles pour la lecture + 4 cycles de multiplications + 5 cycles de vision + 3 cycles d'addition * 2 + 1 cycle d'écriture = 17 cycles. Ce n'était pas demandé, mais j'ai dessiné à la page suivante un début de résultat similaire à la fig 3.1.*

*c.2) Pour 120 itérations, on aura $17 + (119 * II) = 136$ cycles ~~alors que pour la solution de la figure 3.1 (no 1a) on a $17 * 120 = 720$ cycles donc $720/136 = 5.3$ fois plus vite avec pipeline.~~*



Etc.

QUESTION #4 (3 points) Interconnexions AXI

- a) (.5 pt) Donnez les avantages et désavantages de l'interconnexion AXI par rapport à un bus standard où le bus standard (par exemple AHB) offre un seul accès à la fois par maître (initiator) et un arbitre fait le choix du maître quand il y a plusieurs maîtres.

Avantages : faible latence et moins de contention

Désavantage : plus de ressources

- b) (1.25 pt) Avec l'information donnée en Annexe, décrivez l'architecture générale d'une interconnexion AXI standard et son protocole.

Voir power point du bloc 2 pp. 73 à 85.

- c) (1.25 pt) Complétez le schéma de la figure 4.1 pour une transaction AXI standard de lecture de 5 mots entre un processeur (*initiator*) et une mémoire (*target*). La mémoire est plus lente que le processeur c'est-à-dire qu'elle prend 1 cycle avant de répondre (en anglais on dit 1 *wait state*). Regroupez ensemble les signaux d'un même canal.

Voir la solution figure 4.1.

Aussi, suite à une question, j'ai ajouté une figure Vivado du lab 2 dans lequel j'ai fait un "expand" des signaux et de l'interconnexion AXI. Vous pouvez voir les connexions entre les 3 signaux du canal Read Channel Address. J'ai mis 3 lignes noir en pointillé pour illustrer la connexion des signaux. Si on demandait à Vivado de générer le VHDL, on pourrait voir un signal entre ces 3 ports (voir figure). Aussi, ici l'interconnexion AXI est de 2:1 c'est-à-dire a 2 maîtres (un premier MM2S du DMA pour lire les 2 matrices et un deuxième S2MM pour le DMA toujours mais pour écrire la matrice résultante après la multiplication) et 1 slave qui est la DDR (dans MM2S la DDR fait une transaction de lecture alors que dans S2MM la DDR fait une transaction d'écriture).

N.B. MM2S = Memory to Stream (de la mémoire DDR au DMA pour lecture)

S2MM = Stream to Memory (du DMA vers la DDR pour écriture)



Figure 4.1 À compléter

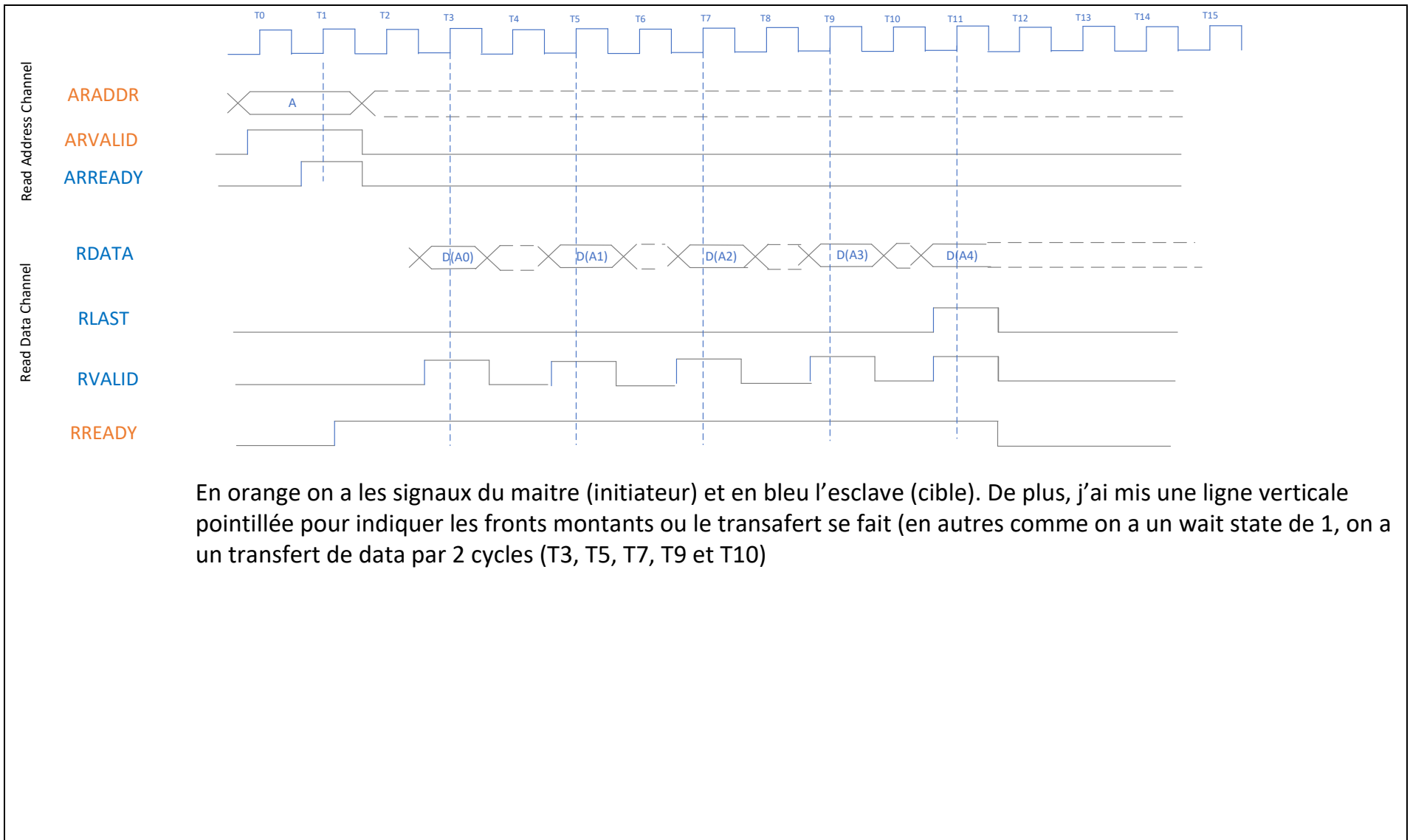
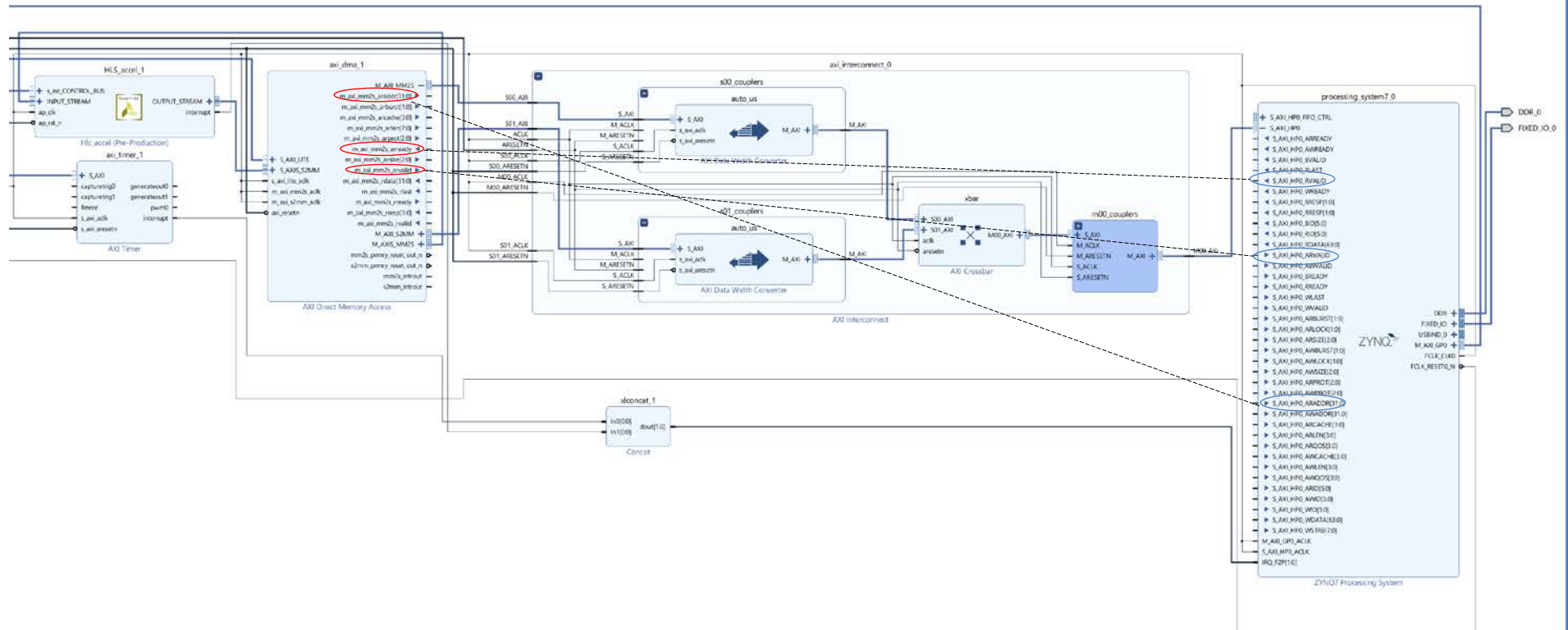


Figure 4.1 Solution



Représentation des paires de signaux (3) connectés entre eux (en pointillés) pour le canal Read Address Channel lors d'une transaction de lecture (MM2S)

QUESTION #5 (3 points) Retour sur l'intra incluant le watchdog

- a) (1 pt) La venue de OS/III a offert une nouvelle option nommé OS_OPT_DEL_ALWAYS est apparue lors de la destruction d'un sémaphore. Plus précisément, en faisant OSDel(&SemFull, OS_OPT_DEL_ALWAYS, &err) uC/OS-III procède à la suppression du sémaphore, que les tâches soient en attente sur le sémaphore ou non. S'il y a des tâches en attente sur le sémaphore, ces tâches deviennent donc prêtes à être exécutées. Ces dernières pourront en être informées par le biais d'un code d'erreur approprié lors de leur retour de l'appel à OSSemPend(&SemFull, , , &err), qui retournera alors `err = OS_ERR_OBJ_DEL`, indiquant que le sémaphore a été détruit.

Quelle est l'utilité de ce mécanisme dans l'implémentation d'un producteur consommateur. Expliquez.

Voir solution intra aut 2023

- b) (1 pt) Dans un test de RMA, quel est en général l'intérêt de doubler la période Task3 c'est-à-dire la faire passer de 210 à 420 tel qu'illustré à la figure 5.1 ? Expliquez.

Tâches	Période P_i	Pire temps d'exécution C_i	Temps d'exécution en moyenne C_i moyen
Task1	100	20	10
Task2	150	30	25
Task3'	420	70	40
Task3''	420	70	40
Task4	400	100	20

Figure 5.1

Voir solution intra aut 2023

- c) (1 pt) Vrai ou faux avec explications : les codes 1 et 2 de la figure 5.2 (page suivante) ne sont pas fonctionnellement équivalents. Considérez que la minuterie du watchdog est 100 fois plus lente que celle de la minuterie principale.

Vrai les 2 permettent d'ordonnancer une tâche de manière périodique et sans tenir compte du temps d'exécution de la tâche.

```

int main (void)
{
    .
    .
    .
    OSSemCreate(&Synchro, "Synchro", 0, &err);

    OSTmrCreate(&watchdog,
                "watchdog",
                0,
                10,
                OS_OPT_TMR_PERIODIC,
                Watchdog_Fct,
                0,
                &err);

    OSTaskCreate(&TaskTCB, "Task", Task, (void *) 0, TASK_PRIO,
                 &TaskStk[0], TASK_STK_SIZE/2, TASK_STK_SIZE, 20, 0, (void *) 0,
                 (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR), &err);

    ⋮
    ⋮
    ⋮
}

void Task (void *data)
{
    OS_ERR err;
    CPU_TS ts;
    OS_TICK actualticks = 0;
    int WAITFORTICKS;

    OSTmrStart(&Scheduler, &err);

    while(1)
    {

        do { WAITFORTICKS = (rand() %500); } while (WAITFORTICKS == 0);
        actualticks = OSTimeGet(&err);
        while(WAITFORTICKS + actualticks > OSTimeGet(&err));
        OSSemPend(&Synchro,0, OS_OPT_PEND_BLOCKING, &ts, &err);

    }
}

void Watchdog_Fct (OS_TMR *p_tmr,
                  void *p_arg)
{
    OS_ERR err;
    OSSemPost(&Synchro, OS_OPT_POST_1, &err);
}

```

Code 1

```

// On suppose simplement une création de tâche Task comme ci-haut.
void Task (void *data)
{
    OS_ERR err;
    OS_TICK actualticks = 0;
    int WAITFORTICKS;
    OSTmrStart(&Scheduler, &err);

    while(1)
    {
        do { WAITFORTICKS = (rand() %500); } while (WAITFORTICKS == 0);
        actualticks = OSTimeGet(&err);
        while(WAITFORTICKS + actualticks > OSTimeGet(&err));
        OSTimeDly(1000, OS_OPT_TIME_PERIODIC, &err);
    }
}

```

Code 2

Figure 5.2**QUESTION #6 (2.5 points) Laboratoire 2**

- a) (1 pt) Quand on part en voyage à l'international comme en Europe, il faut penser d'amener un convertisseur de courant universel pour nos appareils électriques. Ceci m'a amené en classe à faire l'analogie entre ce type de convertisseurs et l'acteur *adapteur* dans la communication AXI. Selon vous, y avait-il un adaptateur dans vos 4 solutions? Expliquez.

Suggestion: vous devriez avoir vu le résultat de synthèse de ce protocole.

A été présenté en classe, il s'agit du protocole entre hls_accel et le DMA composé de 3 boucles. C'est d'ailleurs lui qui fait baisser N de la question b)

- b) (.75 pt) Soit la solution 3.2 (INT16 avec II=2), pour lequel N = 80 (plutôt que 60). Les résultats d'utilisation des ressources sont présentés à la figure 6.1. Cette solution aurait-elle pu être implémentée et exécutée dans SDK comme solution 3.2 (2 copro avec II=2). Expliquez.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	80	-	-
Expression	-	-	0	1026
FIFO	-	-	-	-
Instance	0	-	36	40
Memory	168	-	0	0
Multiplexer	-	-	-	2816
Register	0	-	4030	128
Total	168	80	4066	4010
Available	280	220	106400	53200
Utilization (%)	60	36	3	7

Figure 6.1

Non trop de BRAM si on ajoute l'interface adaptateur de a) et si on souhaite mettre 2 hls_accel sur le FPGA.

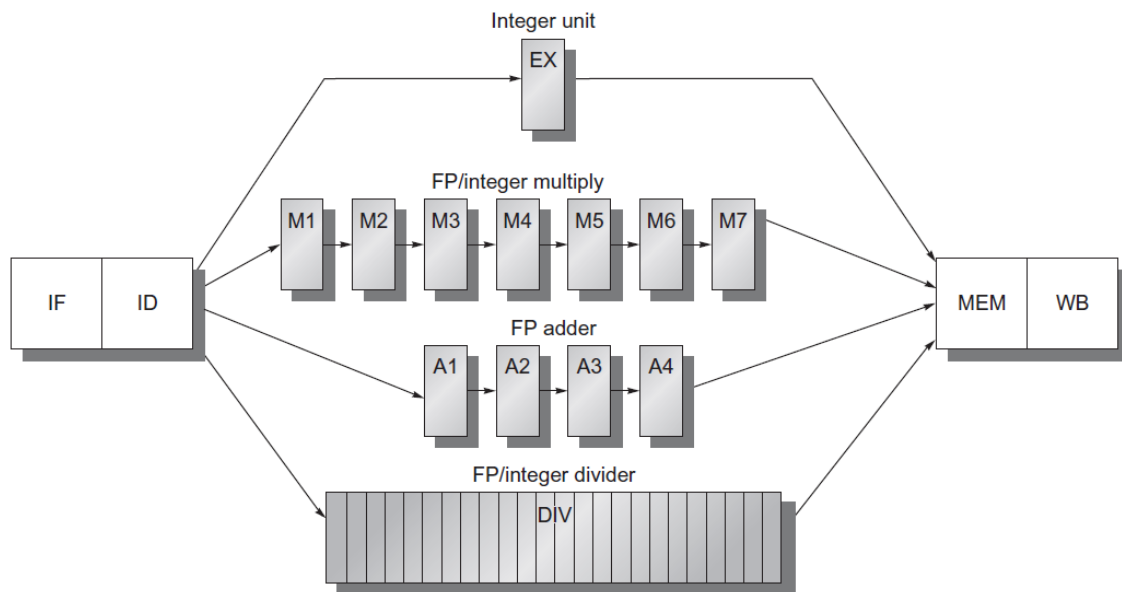
- c) (.75 pt) Pourquoi avoir désactivé la mémoire cache pour le test de II =2 de la solution 1.3 et 2.3 avec `Xil_DCacheInvalidateRange(BASEADDR,16)` où `BASEADDR = 0x3000000`.

Même raison que pour le lab 1partie 3. Si la donnée est modifiée dans la cache L1 de core 0 il que L2 n'est pas mis à jour tout de suite, core 1 ne lira peut être pas la bonne valeur...

Annexe

Détails des instructions pouvant être pipelinées	Nom de l'instruction	Nombre de cycles dans EX	Cycle du pipeline où l'opération termine
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)
ADDD F1, F1, F3	Addition de deux doubles mots : $F1 \rightarrow F1 + F3$	3	ER (le résultat est mis dans l'accumulateur après E3)
MULTD F1, F1, F5	Multiplication de deux doubles mots : $F1 \rightarrow F1 * F5$	4	ER (le résultat est mis dans l'accumulateur après E4)
DIVD F1, F1, F5	Division de deux doubles mots : $F1 \rightarrow F1 / F5$	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
BNEQ R3, etiq	Branch si non nul	1	EX
MSUBI R1, R7, #8 N.B. Ce genre d'instruction n'existe probablement pas, c'est simplement ici pour simplifier le code et ne pas avoir à écrire 7 décréments consécutifs.	Soustraction immédiate multiple pour les registres allant de R1 à R7 : $R1 \rightarrow R1 - 8, \dots, R7 \rightarrow R7 - 8$	1	ER

Tableau 1.1 Détail des instructions du RISC. De plus, considérez qu'il s'agit d'un modèle M4 et que 2 ports de mémoire aux étapes de LI et de ME du pipeline



Exemple de modèle M4

✓ *pragma HLS pipeline*

✓ Description

The PIPELINE pragma reduces the initiation interval (II) for a function or loop by allowing the concurrent execution of operations.

✓ Syntax

Place the pragma in the C source within the body of the function or loop.

```
#pragma HLS pipeline II=<int> enable_flush rewind
```

Where:

- **II=<int>**: Specifies the desired initiation interval for the pipeline. The HLS tool tries to meet this request. Based on data dependencies, the actual result might have a larger initiation interval. The default II is 1.
- **enable_flush**: Optional keyword that implements a pipeline that will flush and empty if the data valid at the input of the pipeline goes inactive.
 - **TIP**: This feature is only supported for pipelined functions; it is not supported for pipelined loops.
- **rewind**: Optional keyword that enables rewinding, or continuous loop pipelining with no pause between one loop iteration ending and the next iteration starting. Rewinding is effective only if there is one single loop (or a perfect loop nest) inside the top-level function. The code segment before the loop:
 - Is considered as initialization.
 - Is executed only once in the pipeline.
 - Cannot contain any conditional operations (if-else).
- **TIP**: This feature is only supported for pipelined loops; it is not supported for pipelined functions.

✓ *pragma HLS array_partition*

✓ Description

Partitions an array into smaller arrays or individual elements and provides the following:

- Results in RTL with multiple small memories or multiple registers instead of one large memory.
- Effectively increases the amount of read and write ports for the storage.
- Potentially improves the throughput of the design.
- Requires more memory instances or registers.

✓ Syntax

Place the pragma in the C source within the boundaries of the function where the array variable is defined.

```
#pragma HLS array_partition variable=<name> \
<type> factor=<int> dim=<int>
```

where

- **variable=<name>**: A required argument that specifies the array variable to be partitioned.
- **<type>**: Optionally specifies the partition type. The default type is **complete**. The following types are supported:
 - **cyclic**: Cyclic partitioning creates smaller arrays by interleaving elements from the original array. The array is partitioned cyclically by putting one element into each new array before coming back to the first array to repeat the cycle until the array is fully partitioned. For example, if **factor=3** is used:
 - Element 0 is assigned to the first new array.
 - Element 1 is assigned to the second new array.
 - Element 2 is assigned to the third new array.
 - Element 3 is assigned to the first new array again.
 - **block**: Block partitioning creates smaller arrays from consecutive blocks of the original array. This effectively splits the array into N equal blocks, where N is the integer defined by the **factor** argument.
 - **complete**: Complete partitioning decomposes the array into individual elements. For a one-dimensional array, this corresponds to resolving a memory into individual registers. This is the default **<type>**.
- **factor=<int>**: Specifies the number of smaller arrays that are to be created.
 - **IMPORTANT**: For complete type partitioning, the factor is not specified. For block and cyclic partitioning the **factor** is required.
- **dim=<int>**: Specifies which dimension of a multi-dimensional array to partition. Specified as an integer from 0 to <N>, for an array with <N> dimensions:
 - If a value of 0 is used, all dimensions of a multi-dimensional array are partitioned with the specified type and factor options.
 - Any non-zero value partitions only the specified dimension. For example, if a value 1 is used, only the first dimension is partitioned.

```
void OSTmrCreate(OS_TMR      *p_tmr,
CPU_CHAR      *p_name,
OS_TICK      dly,
OS_TICK      period,
OS_OPT      opt,
OS_TMR_CALLBACK_PTR p_callback,
void          *p_callback_arg,
OS_ERR      *p_err)
```

Arguments

dly

specifies the initial delay (specified in timer tick units) used by the timer (see drawing above). If the timer is configured for ONE-SHOT mode, this is the timeout used. If the timer is configured for PERIODIC mode, this is the timeout to wait before the timer enters periodic mode. The units of this time depends on how often the user will call OSTmrSignal() (see OSTimeTick()). If OSTmrSignal() is called every 1/10 of a second (i.e., OS_CFG_TMR_TASK_RATE_HZ set to 10), dly specifies the number of 1/10 of a second before the delay expires.

period

specifies the period repeated by the timer if configured for PERIODIC mode. You would set the “period” to 0 when using ONE-SHOT mode. The units of time depend on how often OSTmrSignal() is called. If OSTmrSignal() is called every 1/10 of a second (i.e., OS_CFG_TMR_TASK_RATE_HZ set to 10), the period specifies the number of 1/10 of a second before the timer repeats.

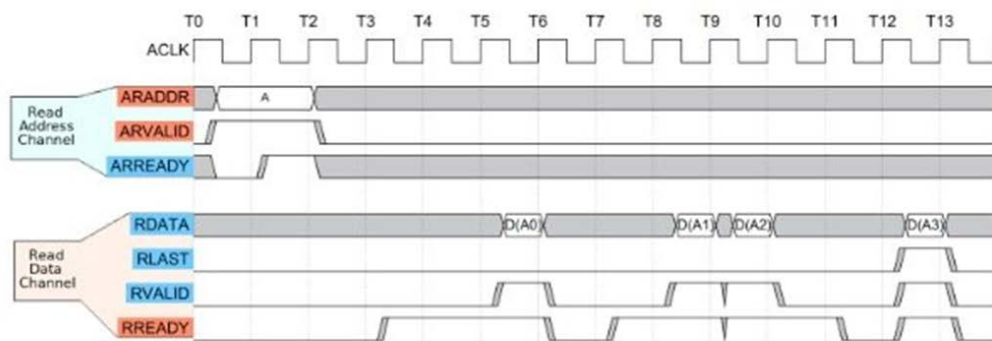
opt

is used to specify whether the timer is to be ONE-SHOT or PERIODIC:

OS_OPT_TMR_ONE_SHOT specifies ONE-SHOT mode

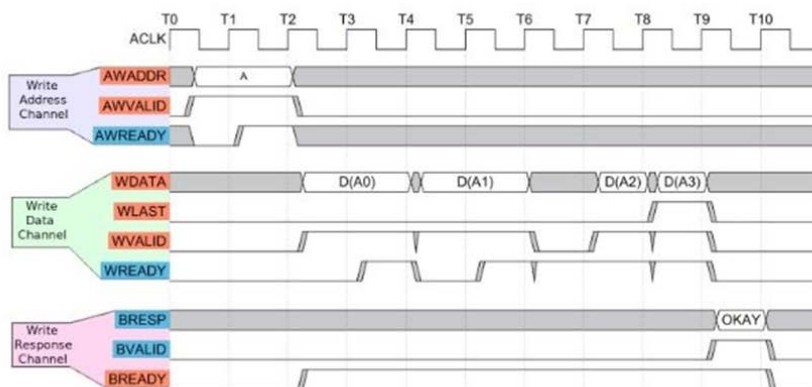
OS_OPT_TMR_PERIODIC specifies PERIODIC mode

Exemples de transaction AXI:



Référence: <https://www.allaboutcircuits.com/technical-articles/what-are-axi-interconnects-tutorial-master-slave-digital-logic/>

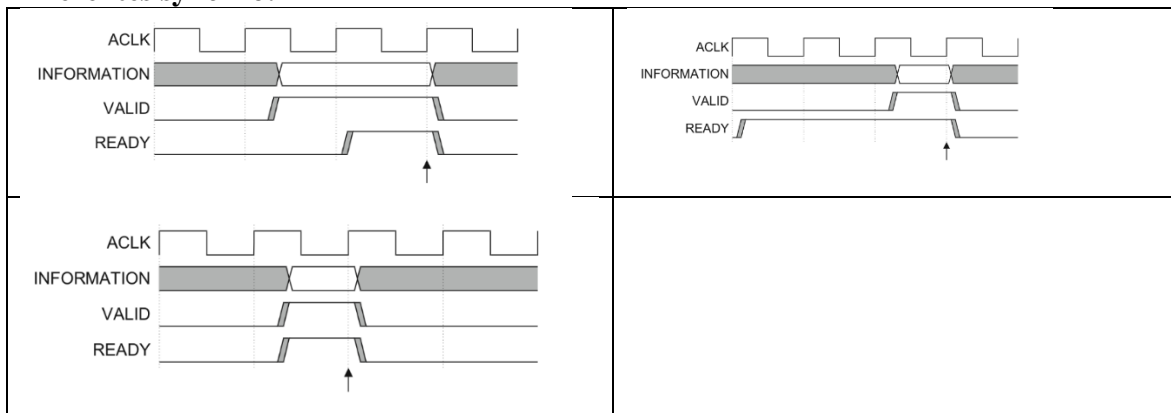
■ :maitre
■ :esclave



Référence: <https://www.allaboutcircuits.com/technical-articles/what-are-axi-interconnects-tutorial-master-slave-digital-logic/>

■ :maitre
■ :esclave

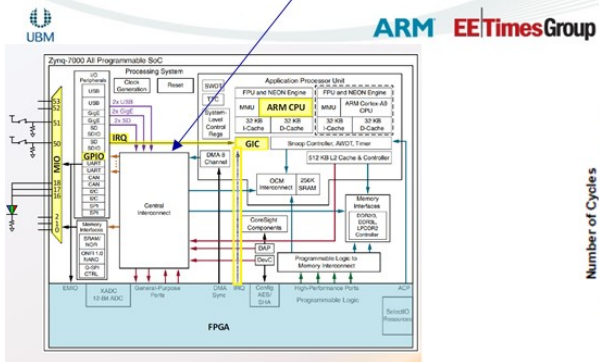
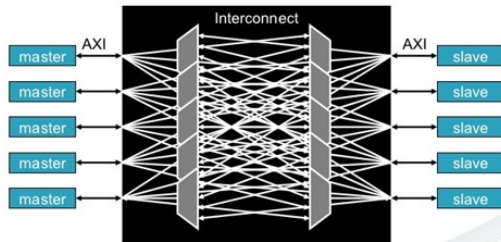
Différentes synchro:



AXI vs AHB

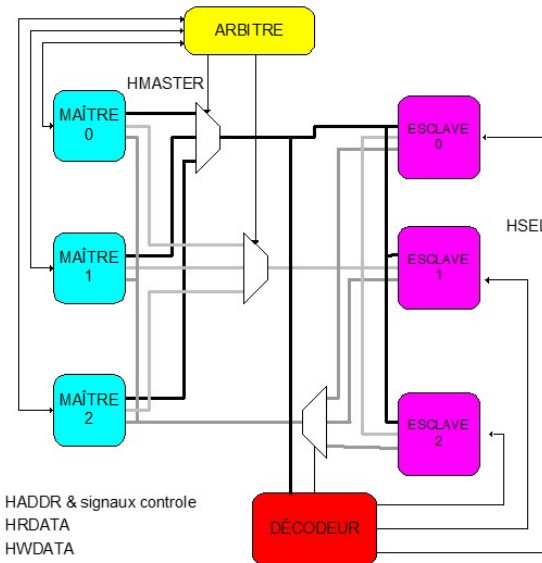
Interconnects logically

The interconnect transports AXI transactions between masters and slaves. The means of transportation are not defined by the AXI spec.

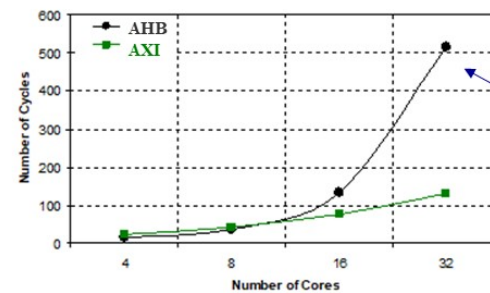


Réf cours no 5

HBUSREQ
or
HGRANT



— HADDR & signaux controle
— HRDATA
— HWDATA



On appelle ça un phénomène
de contention de bus
(bus contention)

INF3610 - Systèmes embarqués

67

