POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

INF3610

Système embarqué

Lab 2

Travail présenté à :

[redacted]

Soumis par :

[redacted]

6 décembre 2024

Exploration 1

Réalisation 1.1)

Cette solution est la solution de base avec aucune optimisation. Elle minimise l'utilisation des ressources du FPGA (DSP48E dans le cas de multiplication accumulation) mais en contrepartie, le temps de calcul est long. Ceci est du au fait que HLS va par défaut optimiser les ressources si aucune directive lui est donné. En fait, sans pragma il donne une solution séquentiel, sans aucune parallélisation.

- mat_type
- mmult_hw
  - a
  - b
  - out
  - L1
    - L2
      - L3

Summary

Latency		Interval		
min	max	min	max	Type
818581	818581	818581	818581	none

Loop

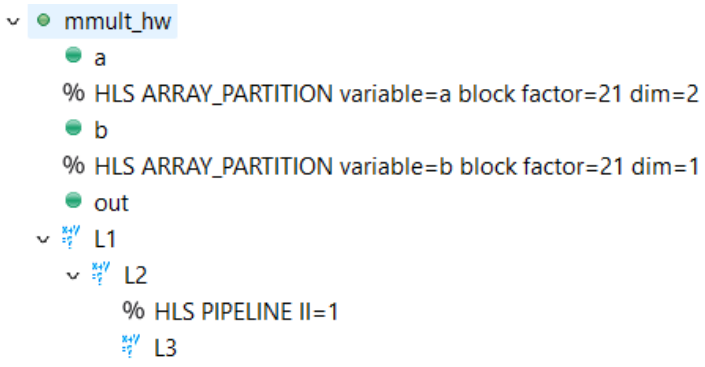
	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	818580	818580	19490	-	-	42	no
+ L2	19488	19488	464	-	-	42	no
++ L3	462	462	11	-	-	42	no

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	143
FIFO	-	-	-	-
Instance	-	5	348	711
Memory	-	-	-	-
Multiplexer	-	-	-	116
Register	-	-	239	-
Total	0	5	587	970
Available	280	220	106400	53200
Utilization (%)	0	2	~0	1

Réalisation 1.2)

Pour cette réalisation, nous avons mis une pipeline au-dessus de L3. Ceci permet de pipeliner les opérations de multiplication et accumulation. Vu que la pipeline demande d'avoir accès à 42 valeurs de a et 42 valeurs de b en même temps (quand le pipeline est plein) il faut partitionner complètement les 2 arrays. Cela peut être fait par 'complète' ou avec un block factor de 21 (21 BRAM à 2 ports chacun). Il faut cependant les partitionner sur 2 dimensions différentes (ligne * colonne donc dim =2 et dim =1 resp.). Avec cette solution, nous arrivons à avoir 1 résultat par cycle après que la latence pour remplir la pipeline est terminé. De plus, cette Solution maximise l'utilisation des DSP48E (95%).



Summary

Latency		Interval		
min	max	min	max	Type
1981	1981	1981	1981	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	1979	1979	217	1	1	1764	yes

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	105
FIFO	-	-	-	-
Instance	-	210	14616	29862
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	0	-	6175	928
Total	0	211	20791	30970
Available	280	220	106400	53200
Utilization (%)	0	95	19	58

Réalisation 1.3)

Vu qu'ici nous cherchons à utiliser autour de 45%. Après réflexion, nous avons réfléchi et avons trouvé que si au lieu d'avoir 1 réponse par cycle, nous avions 1 réponse par 2 cycles, alors nous pouvions diminuer par 2 le nombre de DSP48E nécessaires. Cela peut être fait en augmentant le II de la pipeline à 2. Maintenant, la solution ne demande pas 210 DSP48E, mais plutôt 210/2 = 105 DSP48E (48%). On pourrait avoir moins de 35% si nous passons à II = 3. Remarque : vu que maintenant nous avons besoin de moins de lectures dans les arrays, on pourrait diminuer le partitionnement à factor = 11, car normalement, il faut un factor = 21 (on prend donc plafond de 21/2).

- mmult_hw
  - a
  - % HLS ARRAY_PARTITION variable=a complete dim=2
  - b
  - % HLS ARRAY_PARTITION variable=b complete dim=1
  - out
  - L1
    - L2
      - % HLS PIPELINE II=2
      - L3

Summary

Latency		Interval		
min	max	min	max	Type
3744	3744	3744	3744	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	3742	3742	217	2	2	1764	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	81
FIFO	-	-	-	-
Instance	-	105	7308	14931
Memory	-	-	-	-
Multiplexer	-	-	-	1350
Register	0	-	8358	3335
Total	0	106	15666	19697
Available	280	220	106400	53200
Utilization (%)	0	48	14	37

Réalisation 1.4)
Ici on prend en compte qu'on veut reproduire l'exemple 7 du bloc 2, soit avoir un temps de O(n), avec un n optimal. Pour trouver la meilleure taille, il faudrait prendre en compte 2 limitations de notre FPGA

1er limitation = le nombre de DSP48E est de 220. Vu que 1 multiplication/acc entre 2 float prend 5 DSP48E, alors on peut seulement avoir 220/5 = 44 multiplications qui se font à la fois.

2em limitation = le nombre de lecture que nous pouvons faire dans 1 seul cycle est limité, en effet notre BRAM peut être partitionné en seulement 140 blocs (280 lectures/écritures à la fois)

Si nous supposons que nous voulons partitionner la matrice pour pouvoir calculer 1 ligne par cycle, alors il faudrait calculer combien d'accès sont nécessaires pour ce calcul. Le calcul d'une ligne demande de lire 1 ligne de A, puis toutes les données de B. DIM lecture pour A, DIM *DIM Lecture pour la matrice B, puis DIM Lectures pour AB = 2*DIM + (DIM *DIM). Si on fait un calcul rapide, alors nous arrivons à DIM = 15. Soit 2 * 15 + (15*15) = 255 Lectures à la fois.

Pour le nombre de DSP48E nécessaires, ici on va faire DIM * DIM opérations à la fois pour pouvoir calculer 1 ligne (DIM multiplications pour 1 ligne * DIM colonnes), Donc en gardant en tête que 1 mul/acc prend 5 DSP48E, alors nous arrivons à un maximum de 44 multiplications à la fois, soit un DIM = 6 pour 36 (DIM = 7 donne 49 multiplications à la fois). Cette solution demande seulement 180 LUT, ce qui équivaut à 81% d'utilisation. D'autres solutions sont possibles pour avoir plus de 81% d'utilisations, mais celles-ci demandent d'augmenter les II. Malheureusement, ces solutions ne maximisent la latence.

Réponse : Pour l'implantation de O(n), il faut tout d'abord placer le pipeline au-dessus de L2, ce qui que pendant toute l'exécution, 1 seule boucle s'exécutera : L1. De plus, comme discuté précédemment, il faut donc accéder à 1 ligne de A, pendant que nous accédons à toutes les valeurs de B à la fois. Donc ici notre partitionnement sur A reste le même. Celui sur B, il faudra simplement mettre « dim=0 » pour partitionner la matrice au complet. De plus il faudra aussi partitionner out, car ce dernier demande DIM opérations par cycle, donc nous avons décidé de le partitionner complément. Voici donc la solution qui maximise DIM (6) et maximise les ressources du. FPGA tout en minimisant la latence. De plus, la valeur maximale de SIZE est 6*6 = 36.

- mmult_hw
  - a
  - % HLS ARRAY_PARTITION variable=a complete dim=2
  - b
  - % HLS ARRAY_PARTITION variable=b complete dim=0
  - out
  - % HLS ARRAY_PARTITION variable=out complete dim=0 Attention dim=2 est suffisant
  - L1
    - % HLS PIPELINE II=1
    - L2
      - L3

Summary

Latency		Interval		
min	max	min	max	Type
41	41	41	41	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	39	39	35	1	1	6	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	25
FIFO	-	-	-	-
Instance	-	180	12528	25596
Memory	-	-	-	-
Multiplexer	-	-	-	57
Register	0	-	5360	1032
Total	0	180	17888	26710
Available	280	220	106400	53200
Utilization (%)	0	81	16	50

Questions Exploration 1

1.1)
Le test Bench sert à vérifier le bon fonctionnement de notre solution. En général, un test Bench va comparer les résultats obtenus par notre solution à des résultats attendus prédéfinis dans le test Bench. Dans notre cas spécifique, dans **mmult_sw** il y aura une multiplication logicielle, puis dans le **main** on va partir notre solution pour avoir les résultats. Ensuite on va comparer les résultats des 2 matrices résultats. Si elles sont les mêmes alors les test est 'successful'.

Notez aussi qu'il faut avoir en tête que ce n'est qu'une simulation fonctionnelle! I.e. pas encore de délai...

1.2) 5 DSP
1.3) 211 DSP
1.4) 106 DSP
1.5) 180 DSP

1.6) Ici il faut juste prendre en compte que 1 mult/acc de float utilise 5 DSP (solution 1.1). Ensuite, il faut prendre en compte combien de mult /acc il faut faire à la fois. Si on utilise la solution 1.2 par Example, cette dernière fait le produit de 1 ligne * 1 colonne. Donc il faut 42 multiplications. Donc logiquement il faudrait donc 42 * 5 = 210 DSP, ce qui concorde avec la réponse en 1.3 (1 DSP ajouté pour autres calculs).

1.7)
Solution 1.1 = Ici aucune boucle est déroulé et tout se fais séquentiellement. Comme on peut voir sur l'onglet 'loop', elle fait 3 Loop de 42 Imbriqués, donc on à faire à une complexité de O(n^3).
Solution 1.2 = Ici on fait un pipeline au-dessus de L3, ce qui à pour effet d'éliminer la boucle L3 et la remplacer par un pipelinage qui produit 1 résultat par cycle. On peut donc voir dans 'loop' que nous avons donc 1 seule boucle L1_L2 qui est exécuté 42*42 fois (1764). Donc nous avons donc une complexité de O(n^2).
Solution 1.3 = Ici on a la même solution que 1.2, c'est juste que le II est de 3, donc le temps pour avoir un résultat n'est pas plus long, mais ce changement n'affecte en aucun cas le nombre de boucles à faire. Donc on a une complexité encore de O(n^2).
Solution 1.4 = Comme expliqué plus tôt, vu que la pipeline est au-dessus de L2, alors le calcul donné une ligne résultante à la fois. Comme on peut le voir dans le 'loop' de la solution 1.3, on voit donc que la boucle L1 est la seule à être exécuté 6 fois. Donc nous avons bien une complexité de O(n).

1.8)
Ma solution comporte déjà une valeur qui n'est pas complète. On a fait un bloc-factor = 21, car ce dernier permet tout comme 'complète' d'accéder à toutes les valeurs du arrays en même temps. En effet, la mémoire BRAM étant dual-port, elle peut effectuer deux opérations simultanées (lecture et écriture). En divisant notre tableau de 42

éléments en 21 segments, chaque segment contient 2 éléments. Grâce à la capacité dual-port, ces 2 éléments peuvent être lus simultanément, ce qui équivaut à un accès parallèle à toutes les 42 valeurs du array.

1.9)

Comme déjà discuté précédemment dans la solution 1.3, vu que nous avons augmenté de II à 2, cela nous donne donc 1 résultat par 2 cycles, ce qui diminue donc le nombre de DSP nécessaires. Ce changement fait aussi que nous avons donc besoin de lire moins de valeurs en simultané. Plus précisément 2 fois moins de valeurs à la fois. Donc ici au lieu d'avoir besoin de lire 42 valeurs à la fois dans les arrays nous avons besoin de 21 valeurs à la fois. Donc ici on peut avoir un bloc-facto = 11. 21/11 = 2 lectures à la fois. Donc on peut lire toutes les 21 valeurs nécessaires en 1 cycle puis les 21 autres le cycle suivant.

1.10)

Pour trouver le nombre de lignes, il faut simplement aller dans le fichier VHDL généré par HLS. Ce dernier se trouve dans : solution > syn > vhdl > mmult_hw.vhd. Nous avons donc trouvé pour notre solution 1.2 : 11 637 lignes de code VHDL.

Exploration 2

Réalisation 2.1)

Cette solution est la solution de base avec aucune optimisation. Elle minimise l'utilisation des ressources du FPGA. Ceci est dû au fait que HLS va par défaut optimiser les ressources si aucune directive lui est donné.

Summary

Latency		Interval		
min	max	min	max	Type
818581	818581	818581	818581	none

Detail

Instance

Loop

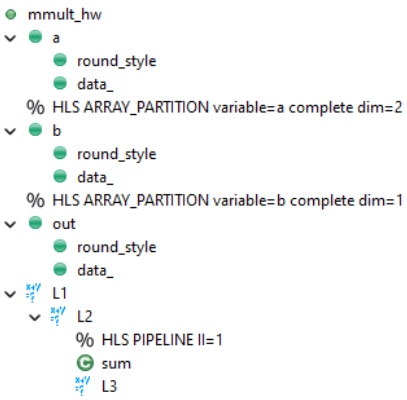
	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	818580	818580	19490	-	-	42	no
+ L2	19488	19488	464	-	-	42	no
++ L3	462	462	11	-	-	42	no

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	143
FIFO	-	-	-	-
Instance	-	4	200	152
Memory	-	-	-	-
Multiplexer	-	-	-	116
Register	-	-	175	-
Total	0	4	375	411
Available	280	220	106400	53200
Utilization (%)	0	1	~0	~0

Réalisation 2.2)

Cette solution est la même que la solution 1.2 en termes de pragma (Ici on assume qu'on voulait garder un II = 1). Pour trouver la valeur optimale nous avons donc nous avons seulement augmenté DIM graduellement jusqu'à trouver cette valeur maximale de ressource. Nous avons trouvé que DIM=54 utilise 98% des ressources et que DIM=55 dépasse le 100%. Donc la valeur maximale de DIM pour avoir un II de 1 est de 54.
Ce qui fais du sens, car 54*4 DSP = 216 et que 55*4 DSP = 220 (HLS ajoute un DSP pour d'autres calculs qui fais déborder à 221 DSP donc non acceptable)



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	82
FIFO	-	-	-	-
Instance	-	216	10800	8208
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	0	-	4105	640
Total	0	217	14905	9005
Available	280	220	106400	53200
Utilization (%)	0	98	14	16

Summary

Latency		Interval		
min	max	min	max	Type
3193	3193	3193	3193	none

Detail

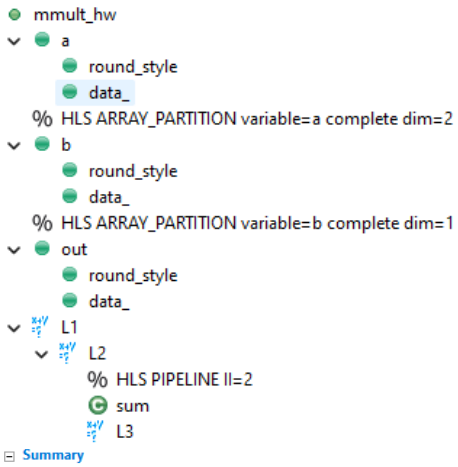
Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	3191	3191	277	1	1	2916	yes

Réalisation 2.3)

Tout comme 1.3, il faut seulement faire varier le II. Si on veut diminuer autour de 45% l'utilisation des DSP. En augmentant le II à 2, nous avons alors besoin de mois de DSP par cycle (une division par 2 du nombre de DSP). Donc au lieu d'en avoir besoin de 216 comme à la solution 2.2, nous en avons maintenant besoin seulement de 216/2 = 108 DSP. Soit 49%.



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	80
FIFO	-	-	-	-
Instance	-	108	5400	4104
Memory	-	-	-	-
Multiplexer	-	-	-	1710
Register	0	-	7238	4876
Total	0	109	12638	10770
Available	280	220	106400	53200
Utilization (%)	0	49	11	20

Summary

Latency		Interval		
min	max	min	max	Type
6108	6108	6108	6108	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	6106	6106	277	2	2	2916	yes

Questions Exploration 2

2.1) 4 DSP
2.2) 217 DSP
2.3) 109 DSP

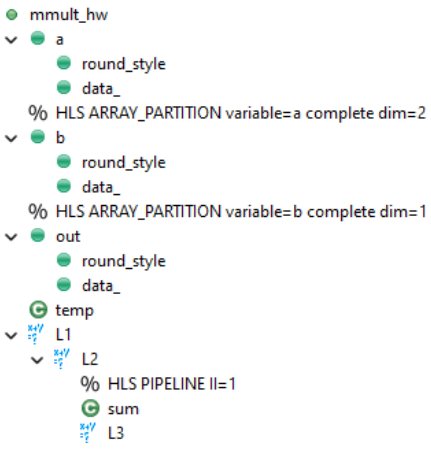
2.4) Premièrement, il faut prendre en compte que 1 multiplication entre 2 half-float utilise 4 DSP. Cela est dû à cause quel la solution 2.1 utilise 4 DSP et que cette dernière fait ses multiplications/accumulation en série. Ensuite pour la

multiplication/accumulation entre 1 ligne *1 colonne il faut donc 54 multiplications. Donc on a besoin de 54 * 4 = 216 DSP. Notre réponse concorde avec la réponse 2.2 (1 DSP ajouté pour d'autres calculs).

Exploration 3

Réalisation 3.1)

Cette solution est la même que la solution 2.2 en termes de pragma. Nous avons aussi utilisé la même DIM que 2.2, soit DIM=54. Celle-ci utilise 98% des ressources comme la solution 2.2.



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	82
FIFO	-	-	-	-
Instance	-	216	10800	8208
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	0	-	4105	640
Total	0	217	14905	9005
Available	280	220	106400	53200
Utilization (%)	0	98	14	16

Summary

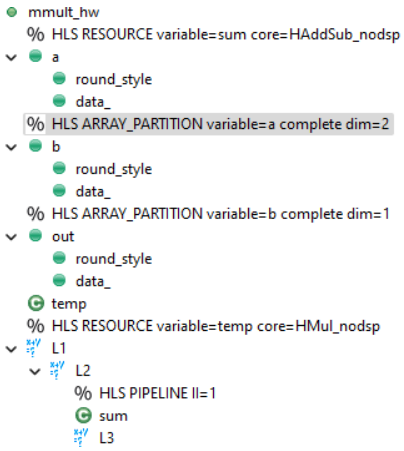
Latency		Interval		
min	max	min	max	Type
3193	3193	3193	3193	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	3191	3191	277	1	1	2916	yes

Réalisation 3.2)

Cette solution ajoute seulement 2 pragmas pour enlever l'utilisation des DSP. Pour trouver trouver le nombre optimal nous avons effectué une recherche binaire entre 0 et 200 (vu que 54 utilisais moins de 50% des ressources). Nous avons trouvé que DIM=140 utilisais 95% des LUT.



Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	95
FIFO	-	-	-	-
Instance	-	0	27440	49840
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	0	-	9914	640
Total	0	1	37354	50650
Available	280	220	106400	53200
Utilization (%)	0	~0	35	95

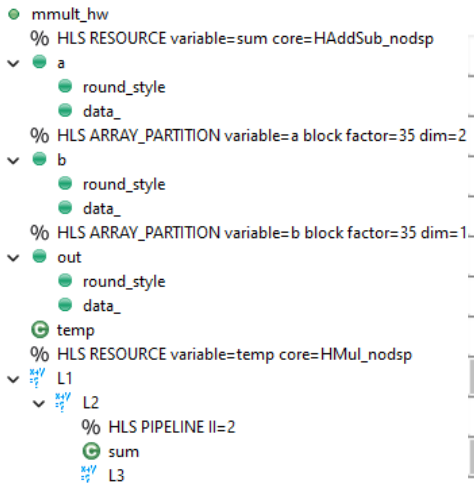
Summary

Latency		Interval		
min	max	min	max	Type
20167	20167	20167	20167	none

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	20165	20165	567	1	1	19600	yes

Réalisation 3.3)

Cette solution utilise le même dim que 3.3, soit DIM=140. Ensuite on change II = 2, puis on change le bloc factor des array A et B. Vu que nous avons seulement besoin de la moitié de la array par cycle, alors nous avons besoin de 70 lectures à la fois, donc 70/2 = 35 est le bon bloc factor à utiliser. Ici on se retrouver avec un peu moins de ressources LUT utilisé, soit 87%. Cette solution sera changée par la suite par une solution avec DIM= 54, donc faudra changer le bloc factor.



Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	166
FIFO	-	-	-	-
Instance	-	0	13720	24920
Memory	-	-	-	-
Multiplexer	-	-	-	6390
Register	0	-	18415	15279
Total	0	1	32135	46755
Available	280	220	106400	53200
Utilization (%)	0	~0	30	87

Summary

Latency		Interval		
min	max	min	max	Type
39767	39767	39767	39767	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	39765	39765	568	2	2	19600	yes

Questions Exploration 3

3.1)

La première directive dit que la variable sum ne pourra pas utiliser de DSP quand elle fera des opérations d'addition ou de soustraction.

```
% HLS RESOURCE variable=sum core=HAddSub_nodsp
```

Pour la deuxième directive, elle restreint la variable temp (s'occupe de garder le résultat de la multiplication a[i][j] * b[i][j]) d'utiliser les DSP durant une opération de multiplication

```
% HLS RESOURCE variable=temp core=HMul_nodsp
```

3.2) Le code diffère car il est obligé d'ajouter une variable « temp » qui va garder le résultat de la multiplication, puis il va faire l'accumulation. Les DSP48E peuvent faire des multiplications/accumulation. Vu que nous ne pouvons pas en utiliser, alors il faut séparer la commande multiplications/accumulation fait initialement par 2 étapes.

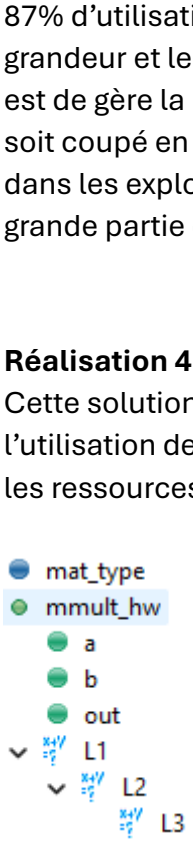
3.3) l'avantage à utiliser aucun DSP est que nous pouvons avoir une dimension beaucoup plus grande. En effet, les DSP nous limitait à 4 DSP par mult/acc. Donc avions un maximum de dim = 54 qui utilise 98% des ressources DSP. Si nous fessons la même chose sans DSP, nous obtenons maintenant une utilisation de LUT à mois de 50%, donc pouvons en conséquence avoir d'autres choses qui sont faite à la fois (comme calculer une autre matrice avec le reste des LUT)

3.4) Non. En effet si nous passons de II = 1 avec une DIM=140, alors 95% des ressources sont utilisés. Maintenant avec II = 2, et la même dimension nous obtenons 87% d'utilisation. Ceci est dû à cause une grande partie des LUT est utilisé pour gérer la grandeur et le flow plutôt que de faire les calculs. Tandis que pour le DSP son seul but est de gère la multiplication/accumulation. Donc ça fait du sens que l'utilisation du DSP soit coupé en 2 si on diminue par 2 le nombre de multiplication/accumulation comme dans les explorations 1 et 2. Tandis que l'utilisations ne diminue pas par 2, car une grande partie des LUT ne s'occupe pas de multiplication/accumulation.

Exploration 4

Réalisation 4.1)

Cette solution est la solution de base avec aucune optimisation. Elle minimise l'utilisation des ressources du FPGA. Ceci est dû au fait que HLS va par défaut optimiser les ressources si aucune directive lui est donné



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	143
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	92
Register	-	-	151	-
Total	0	1	151	235
Available	280	220	106400	53200
Utilization (%)	0	~0	~0	~0

Summary

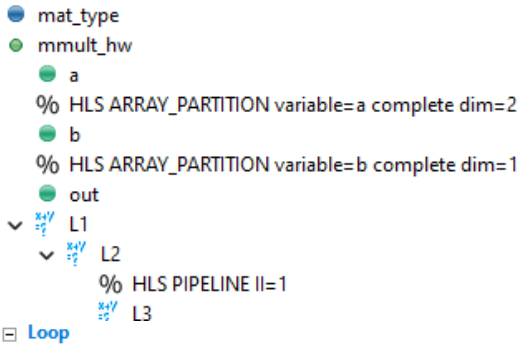
Latency		Interval		
min	max	min	max	Type
225877	225877	225877	225877	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	225876	225876	5378	-	-	42	no
+ L2	5376	5376	128	-	-	42	no
++ L3	126	126	3	-	-	42	no

Réalisation 4.2)

Cette solution fait seulement la partition complète des 2 arrays puis placer un pipeline au-dessus de L3 offrant un temps de O(n2) et un II=1. Cette solution utilise une DIM=216 qui utilise 98% des DSP. Nous avons pu trouver ce DIM avec une simple recherche binaire. Si la solution 4.1 utilise 1 DSP, alors nous pouvons assumer que 1 mult/acc avec short utilise seulement 1 DSP, donc nous pouvons aller chercher directement chercher DIM=216 qui utilise au-dessus de 95% des ressources.



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	217	-	-
Expression	-	-	0	1622
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	0	-	9707	96
Total	0	217	9707	1793
Available	280	220	106400	53200
Utilization (%)	0	98	9	3

Loop

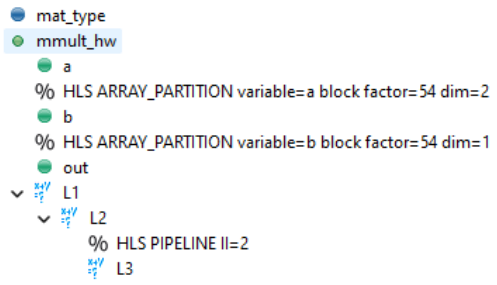
	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	46661	46661	7	1	1	46656	yes

Summary

Latency		Interval		
min	max	min	max	Type
46663	46663	46663	46663	none

Réalisation 4.3)

Il faut seulement ici trouver le bon factor et changer le II =2. Le bon factor serait donc de 216/2 lectures à la fois sur chaque array. Vu que nous avons 2 lectures possibles à cause du dual port alors nous avons besoin de un array_partition = 216/2/2 = 54.



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	217	-	-
Expression	-	-	0	1655
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	3564
Register	0	-	7622	64
Total	0	217	7622	5283
Available	280	220	106400	53200
Utilization (%)	0	98	7	9

Summary

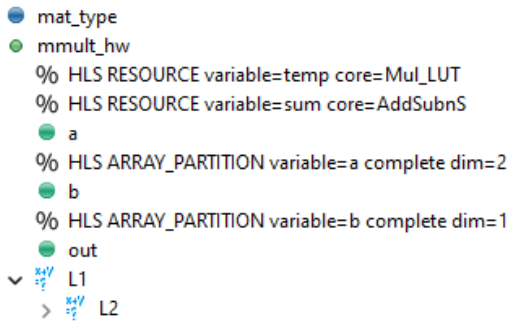
Latency		Interval		
min	max	min	max	Type
93319	93319	93319	93319	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	93317	93317	8	2	2	46656	yes

Réalisation 4.2.1)

Pour trouver la nouvelle valeur de DIM, il faut donc refaire une nouvelle recherche binaire. On trouve finalement que la nouvelle DIM optimale est plus basse que celle qui utilise les DSP. La dim optimale sans DSP est DIM = 164.



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	95
FIFO	-	-	-	-
Instance	-	0	24939	50311
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	0	-	8426	640
Total	0	1	33365	51121
Available	280	220	106400	53200
Utilization (%)	0	~0	31	96

Summary

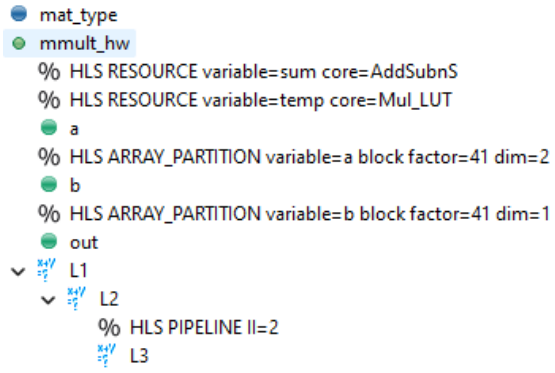
Latency		Interval		
min	max	min	max	Type
27063	27063	27063	27063	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	27061	27061	167	1	1	26896	yes

Réalisation 4.3.1)

Il faut seulement ici trouver le bon factor et changer le II =2. Le bon factor serait donc de 164/2 lectures à la fois sur chaque array. Vu que nous avons 2 lectures possibles à cause du dual port alors nous avons besoin de un array_partition = 164/2/2 = 41.



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	166
FIFO	-	-	-	-
Instance	-	0	24939	28171
Memory	-	-	-	-
Multiplexer	-	-	-	2550
Register	0	-	15847	9408
Total	0	1	40786	40295
Available	280	220	106400	53200
Utilization (%)	0	~0	38	75

Summary

Latency		Interval		
min	max	min	max	Type
53958	53958	53958	53958	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	53956	53956	167	2	2	26896	yes

Questions Exploration 4

4.1) Les pragmas utilisés ont la même utilité que la question 3.1. Le premier fait une limitation des ressources à la variable sum. Il lui dit qu'il ne pourra pas utiliser de DSP pour ses calculs. Ensuite le 2em fais une limitation des ressources sur la variable temp. Il lui dit qu'il peut seulement utiliser les LUT pour faire la multiplication.

4.2) Dans ce cas précis non. Il semble que la solution avec les DSP soit plus avantageuse dans le cas de type short. En effet, les DSP48E ont une meilleur optimisation est calculs permettant d'utiliser moins de ressources que des LUT combinés dans le cas des short.

4.3) Non, ni dans le cas avec DSP ni dans le cas sans DSP. Le système doit utiliser les DSP et les LUT pour d'autres opérations en plus du calcul d'accumulation. Cela inclut des tâches comme la gestion des contrôles, la synchronisation, et les interconnexions. On voit ici le que nombre de DSP ne change pas selon de II. Pour le cas du no DSP, le II = 2 permet de réduite un peu les ressources mais pas les couper en 2.

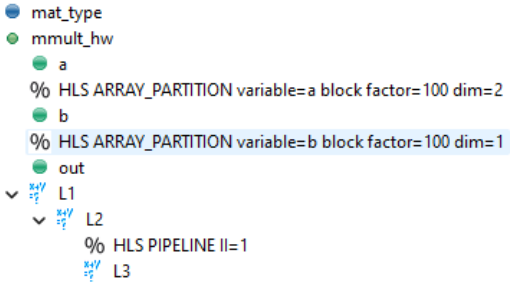
Partie 2 réponses

Test de performance projet_Sol2_2 :

```
Connected to: Serial ( COM6, 115200, 0, 8 )
************************************************************
 __fp16 54 x 54  MATRIX MULT -> AXI DMA -> ARM ACP
 XAPP1170 redesigned with Vivado + HLS + IP Integrator 2015.3
************************************************************
DMA Init done
Loop time for 100 iterations is 137 cycles
Running Matrix Mult in SW

Total run time for SW on Processor is 3833907 cycles over 100 tests.

Cache cleared

Total run time for AXI DMA + HW accelerator is 13982 cycles over 100 tests
Acceleration factor: 274.183

ERROR: results mismatch: 2 times

Temps total avec 1 core: 139 microsecondes pour 1 multiplication de matrices 54 x 54
```

Test de performance projet_Sol4_2 :

Cette solution nous avons donc changé la solution 4.2 pour accommoder une DIM=200 comme demandé. Nous avons seulement changé le bloc factor qui est maintenant de 200/2 = 100.



Summary

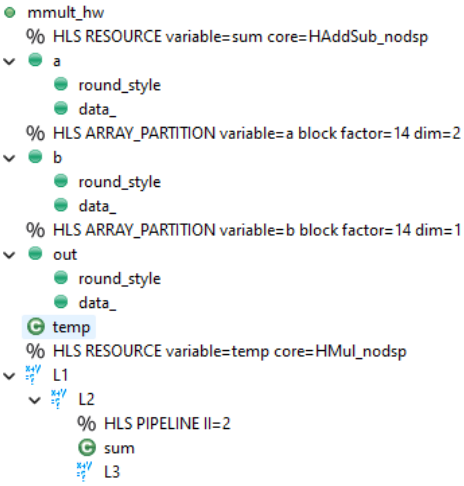
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	201	-	-
Expression	-	-	0	1348
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	0	-	9046	160
Total	0	201	9046	1583
Available	280	220	106400	53200
Utilization (%)	0	91	8	2

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	40006	40006	8	1	1	40000	yes

```
Connected to COM6 at 115200
********************************************************************************
 Short MATRIX MULT II = 1 -> AXI DMA -> ARM AXI HP
 Matrix 200 x 200
 Number de test effectues  50
 Inspire de l'application XAPP1170 de Xilinx
********************************************************************************
DMA Init done
Loop time for 50 iterations is 65 cycles
Running Matrix Mult in SW

Total run time for SW on Processor is 577735 microseconds over 50 tests.
Running Matrix Mult in HW

Cache cleared

Call HW accelerator: 25
End of HW accelerator: 8167248
Total run time for AXI DMA + HW accelerator is 1633 microseconds over 50 tests
Acceleration factor: 353.692

SW and HW results match!

Temps total avec 1 core: 1633 microsecondes pour 1 multiplication de matrices 200 x 200
```

Test de performance projet_Sol3_3 :

Cette solution nous avons donc changé la solution 3.3 pour accommoder une DIM=54 comme demandé. Nous avons seulement changé le bloc factor qui est maintenant de 54/2/2 = 14. Vu que nous avons seulement 31% de LUT, alors cela nous donne assez de place pour instancier 2 accélérateurs.



Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	156
FIFO	-	-	-	-
Instance	-	0	5292	9612
Memory	-	-	-	-
Multiplexer	-	-	-	2490
Register	0	-	7217	4287
Total	0	1	12509	16545
Available	280	220	106400	53200
Utilization (%)	0	~0	11	31

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	6053	6053	224	2	2	2916	yes

Sans ack (core 0) :

```
Already connected to port: COM6********************************************************************************
  __fp16 MATRIX MULT II = 2-> AXI DMA -> ARM AXI HP Core 0
  Matrix 54 x 54
  Number de test effectues  100
  Inspire de l'application XAPP1170 de Xilinx
********************************************************************************
DMA Init done
Core 0: loop time for 100 iterations is 136 cycles
Core 0: running Matrix Mult in SW

Core 0: total run time for SW on Processor is 38456 microseconds over 100 tests.
Core 0: call HW accelerator 26

Cache cleared

Core 0: end of HW accelerator 1948616
Core 0: total run time for AXI DMA + HW accelerator is 194 microseconds over 100 tests
Core 0: acceleration factor: 197.359

Core 0 ERROR: results mismatch: 2 times

Temps total pour core 0 et core 1: 212120 microsecondes pour 2 multiplications de matrices 54 x 54 en parallele
```

Sans ack (core 1) :

```
Connected to COM6 at 115200
********************************************************************************
  __fp16 MATRIX MULT II = 2-> AXI DMA -> ARM AXI HP   Core 1
  Matrix 54 x 54
  Number de test effectues  100
  Inspire de l'application XAPP1170 de Xilinx
********************************************************************************
DMA Init done
Core 1: loop time for 100 iterations is 137 cycles
Core 1: Running Matrix Mult in SW

Core 1: Total run time for SW  is 38299 microseconds over 100 tests.
Core 1: call HW accelerator  25

Cache cleared

Core 1: end of HW accelerator 1957505
Core 1: total run time for AXI DMA + HW accelerator is 195 microseconds over 100 tests
Core 1: acceleration factor: 195.658

Core 1: ERROR results mismatch
```

Synchronisation :

```
**************************************************************************
  __fp16 MATRIX MULT II = 2-> AXI DMA -> ARM AXI HP   Core 1
  Matrix 54 x 54
  Number de test effectues  100
  Inspire de l'application XAPP1170 de Xilinx
**************************************************************************
DMA Init done
Core 1: loop time for 100 iterations is 138 cycles
Core 1: Running Matrix Mult in SW

Core 1: Total run time for SW  is 38459 microseconds over 100 tests.
**************************************************************************
  __fp16 MATRIX MULT II = 2-> AXI DMA -> ARM AXI HP Core 0
  Matrix 54 x 54
  Number de test effectues  100
  Inspire de l'application XAPP1170 de Xilinx
**************************************************************************
DMA Init done
Core 0: loop time for 100 iterations is 137 cycles
Core 0: running Matrix Mult in SW

Core 0: total run time for SW on Processor is 38313 microseconds over 100 tests.
Core 1: call HW accelerator 24


CaCachelearededed

Core 0: end of HW accelerator 1964330
Core 1: end of HW accelerator 1976000
Core 1: total run time for AXI DMA + HW accelerator is 197 microseconds over 100 tests
Core 1: acceleration factor: 194.635

Core 1: ERROR results mismatch

Core 0: total run time for AXI DMA + HW accelerator is 196 microseconds over 100 tests
Core 0: acceleration factor: 195.49

Core 0 ERROR: results mismatch: 2 times

Temps total pour core 0 et core 1: 197 microsecondes pour 2 multiplications de matrices 54 x 54 en parallele
```

Q1S

Concernant le projet_sol2_2 (Synthèse No1 Tableau 1):

a) Que peut-on conclure des 2 erreurs de comparaison entre logiciel et matériel pour le projet_Sol2_2?

Ces mismatch sont dus à la précision limitée du type « float ». En effet, le calcul en logiciel avec le processeur peut utiliser une représentation flottante avec une précision plus petite ou des arrondis différents par rapport au matériel. Dans le cas d'un calcul matériel, des approximations pour gagner en performance, peuvent également introduire des variations dans les résultats. Il y a toujours 2 mismatch, car nous fessons toujours le même calcul de matrice, donc les mêmes erreurs d'approximation et arrondissement vont arriver.

b) L'accélération du matériel par rapport au logiciel est-elle significative? Anticipiez-vous une telle accélération à la suite de la section Exploration architecturale no 2 ? Justifiez.

Bizarre j'ai obtenu plutôt 303 d'accélération

Oui, l'accélération est significative. En effet, On peut voir que la solution 2_2 nous donne une accélération de 274 comparé su SW, ce qui est énorme. Cela est dû à cause des DSP qui sont utilisé à 98% dans cette solution nous permettant d'avoir un temps O(n2) au lieu de O(n3). Nous anticipons une plus petite accélération dans l'exploration 3_3. On anticipe ça car Premieremenet, il n'y a pas de DSP utilisé, vu que ces derniers sont spécialisés dans le calcul de multiplication/accumulation, alors le calcul avec des LUT pourrais prendre un peu plus de temps. De plus, nous avons mis un II=2, ce qui va inévitablement réduire notre accélération. Toutefois, cette solution utilise beaucoup moins de ressources matérielles globales et permet l'exécution parallèle de deux multiplications matricielles. Cela pourrait compenser, dans une certaine mesure, la perte d'accélération.

Q2S

Concernant le projet_sol3_3 (Synthèse No2 Tableau 1):

a) Démarrez Vivado et ouvrez le projet projet_sol3_3. Dans le bas de la colonne de gauche, ouvrez l'onglet IMPLEMENTATION puis Open Implemented Design puis cliquez sur Report Utilization. Vous devriez voir apparaitre la boite de la figure 16 sur laquelle vous cliquez OK. Puis vous verrez apparaitre le tableau de la figure 17 après avoir cliquez sur l'onglet design_1_i. En observant HLS_accel_0 et HLS_accel_1, vous verrez qu'il y a un certain nombre de BRAM. Si vous ouvrez l'onglet HLS_accel_0 y a-t-il une correspondance entre lenombre de BRAM et les pragma HLS ARRAY_PARTITION utilisés à l'Exploration architecturale no 3 (solution 3_3)? Justifiez.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	156
FIFO	-	-	-	-
Instance	-	0	5292	9612
Memory	-	-	-	-
Multiplexer	-	-	-	2490
Register	0	-	7217	4287
Total	0	1	12509	16545
Available	280	220	106400	53200
Utilization (%)	0	~0	11	31

Name	Slice LUTs (53200)	Block RAM Tile (140)	DSPs (220)	Bonded IOPADs (130)	BUFGCTRL (32)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)
design_1_wrapper	31131	42	6	130	1	25813	108	54	11004	27057	4074
design_1_i (design_1)	31131	42	6	0	1		108	54	11004	27057	4074
xlconcat_1 (design_1_xlconcat_1_0)	0	0	0	0	0		0	0	0	0	0
processing_system7_0 (design_1_processing_system7_0_0)	0	0	0	0	1		0	0	0	0	0
proc_sys_reset (design_1_proc_sys_reset_0)	18	0	0	0	0		0	0	12	17	1
axi_timer_2 (design_1_axi_timer_2_0)	293	0	0	0	0		0	0	110	293	0
axi_timer_1 (design_1_axi_timer_1_0)	293	0	0	0	0		0	0	104	293	0
axi_interconnect_2 (design_1_axi_interconnect_2_0)	1044	0	0	0	0		0	0	364	983	61
axi_interconnect_1 (design_1_axi_interconnect_1_0)	641	0	0	0	0		0	0	312	580	61
axi_interconnect_0 (design_1_axi_interconnect_0_0)	1046	0	0	0	0		0	0	351	985	61
axi_dma_2 (design_1_axi_dma_2_0)	1386	5	0	0	0		0	0	596	1273	113
axi_dma_1 (design_1_axi_dma_1_0)	1388	5	0	0	0		0	0	623	1275	113
HLS_accel_1 (design_1_HLS_accel_1_0)	12508	16	3	0	0		54	27	4433	10676	1832
HLS_accel_0 (design_1_HLS_accel_0_0)	12514	16	3	0	0		54	27	4306	10682	1832

Pour cette solution nous avions choisi un bloc-factor = 14 (54/2/2). Ici nous avons donc une bonne correspondance de bloc RAM, soit 16. Si on ouvre plus de détail, on peut voir que 0.5 bloc RAM sont utilisé par bloc-factor instancié. Soit :
a_X_U pour le array A, avec x allant de 0 à 13 (14 blocs) utilisant 0.5 blocs Ram chaque
b_X_U pour le array B, avec x allant de 0 à 13 (14 blocs) utilisant 0.5 blocs Ram chaque
out_U pour la sortie utilisant 2 blocs RAM.
Avec ça nous arrivons donc avec 16 Bloc Ram.

b) Pourquoi ce nombre de BRAM observé dans Vivado n'était pas comptabilisé dans votre rapport de synthèse de Vivado HLS (sous Summary)?

Ici le nombre de BRAM n'est pas connu car HLS ne fais pas de synthèse et ne connais pas le matériel nécessaire. C'est une fois que « ip » faite par hsl pour créer « hls_accel » qu'il pourra nous montrer le nombre de BRAM utilisé. Après avoir créé le IP on peut directement le rouler sur HSL avant même d'aller sur vivado et il nous donnera bien le nombre de BRAM.

c) En observant juste les ressources BRAM, aurait-on pu mettre 3 instances de hls_accel dans le projet_sol3_3? Justifiez.

Si nous fessons un calcul rapide, avec la solution avec 2 instances, nous avons 16 * 2 BRAM utilisé pour les accel et nous avons 5 BRAM utilisé par le 2 DMA. Si nous voulons

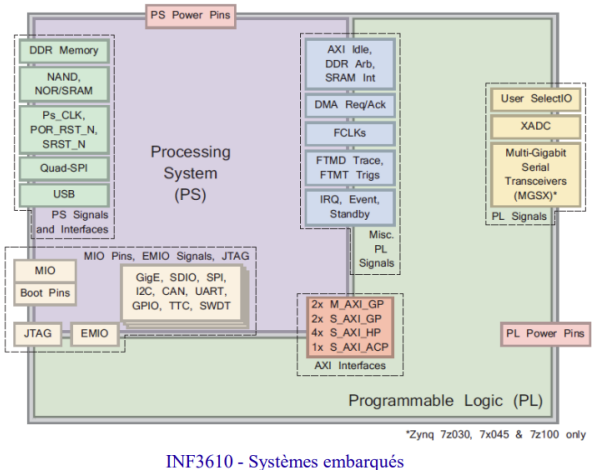
ajouter une 3ème instance, alors il faudrait ajouter un accel + un DMA. Cela nous donne donc 42 + 5 + 16 = 63 bloc RAM. Vu que notre FPGA contient 140 blocs, alors nous en avons bien assez pour faire une 3ème instance (seulement en regardant les ressources BRAM).

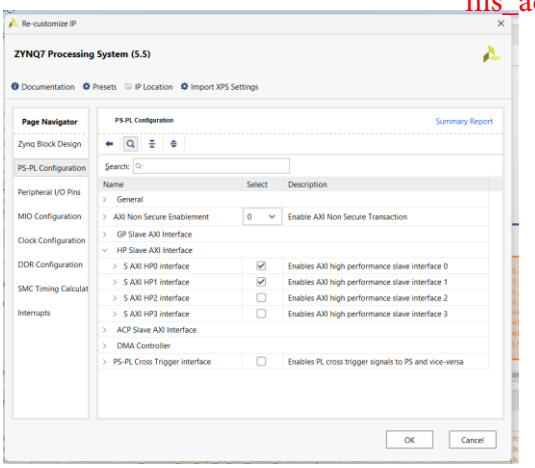
d) En observant juste les ressources de l'ensemble de la puce SoC 7020, aurait-on pu mettre 3 instances de hls_accel dans le projet_sol3_3? Justifiez.

Non on n'aurait pas pu. Ici le nombre de Slice manquerais. Avec 2 accel nous utilisons 11004 slices. Nous remarquons que 1 accel utilise 4433 ou 4306 slices. Donc en ajoutant une 3ème nous avons donc 11004 + 4433 = 15437 Slices. Vu que la carte a au total 13300, alors nous n'aurons pas assez de slice pour instancier un 3ème accel. Ici je n'ai pas pris en compte les ressources nécessaires pour les DMA, car en ajoutant un 3ème DMA il y a aucune ressource manquante.

e) Sachant que j'ai une puce de la série Zynq-7000 SoC (chap. 1, p37) dans laquelle appartient la puce 7020 avec assez de ressources pour mettre plus de 4 instances sur de hls_accel. Qu'elle est alors ma limite en terme du bus AXI. Observez bien votre design (Fig 8). Justifiez.

Premièrement, les ressources de notre FPGA sont limitées à **4 S_AXI_HP (première image provenant du cours)**. Ces bus peuvent donc avoir accès à la mémoire interne DDR donc on va se baser sur cette limiter en termes de bus AXI. Pour le design 3.3, on peut voir qu'il utilise 2 de ces bus HP (deuxième image provenant de vivado). Dans notre cas spécifique, 1 but HP sera utilisé pour transporter nos donnés de la DDR à la nos instances de DMA. Vu que nous avons 2 DMA alors il est logique d'avoir 2 bus HP. Avec ça, on peut donc conclure que pour chaque instance de hls_accel, il nous faudrait donc un bus HP. Donc le maximum de hls_accel qu'on pourrais instancier est de 4. 1 bus HP +1 DMA par hls_accel





f) Finalement, est-ce que l'hypothèse de la page 16 est vérifié? Justifiez.

Ici l'hypothèse est donc vérifiée. En effet, dans le test de 2_2 ou on fait 1 calcul de matrice avec II = 1 et DIM = 54, nous obtenons 139 ms pour le calcul d'une matrice. Pour le test de 3_3 ou on fait 2 calculs de matrices avec II=2 et DIM=54, nous obtenons 197 microsecondes pour 2 matrices. Donc donc si on compare les deux, la solution 2_2 prend 139*2 = 278 ms pour faire 2 matrices et la solution 3_3 prend 196 ms pour calculer 2 matrices, donc on peut dire que la solution 3_3 a une accélération de 1.41 par rapport à 2_2.

Q3S

Concernant le projet_sol4_2 (Synthèse No3 Tableau 1). Si le DIM obtenu lors de Exploration architecturale no 4 solution 4_2 est différent de celui proposé au Tableau 1 (DIM=200), expliquez pourquoi ce choix de 200 qu'on peut voir comme une valeur limite. Suggestion : allez consultez les ressources comme pour la question Q2S a).

Voici les ressources utilisé dans la solution 4_2. Nous avons eu un bloc-factor de 100 pour pouvoir avoir un II=1.

Name	Slice LUTs (53200)	Block RAM Tile (140)	DSPs (220)	Bonded IOPADs (130)	BUFGCTRL (32)	Slice Registers (106400)	F7 Muxes (26600)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)
design_1_wrapper	5018	137	202	130	1	5021	1	2252	4765	253
design_1_i (design_1)	5018	137	202	0	1		1	2252	4765	253
axi_dma_1 (design_1...	1507	5	0	0	0		0	827	1385	122
axi_interconnect_0 (d...	1045	0	0	0	0		0	433	984	61
axi_interconnect_1 (d...	534	0	0	0	0		0	307	473	61
axi_timer_1 (design_1...	294	0	0	0	0		0	128	294	0
HLS_accel_1 (design...	1620	132	202	0	0		1	689	1612	8
proc_sys_reset (desig...	18	0	0	0	0		0	12	17	1
processing_system7_...	0	0	0	0	1		0	0	0	0
xlconcat_1 (design_1...	0	0	0	0	0		0	0	0	0

Ici, 5 bloc RAM sont utilisé pour le DMA et 132 pour le accel. Si nous regardons en détail l'utilisation des bloc RAM dans le accel =
-a_X_U pour le array A, avec X allant de 0 à 99 (50 blocs) utilisant 0.5 blocs Ram chaque
-b_X_U pour le array B, avec X allant de 0 à 99 (50 blocs) utilisant 0.5 blocs Ram chaque
-out_U utilisant 32 blocs RAM

Nous arrivons donc à 137 bloc RAM utilisé au total (98% d'utilisation). Si nous modifions DIM à 201, alors nous allons besoin de 3 blocs supplémentaires (1 pour A, 1 pour B et 1 pour out) ce qui ferais déborder les ressources du FPGA en termes de BRAM.