UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

TRAVAIL DE SESSION

PRÉSENTÉ À

MESSAOUD AHMED OUAMEUR

COMME EXIGENCE PARTIELLE

DU COURS

CONCEPTION EN VLSI

PAR

ANTHONY PINARD
PINA29049809

LABORATOIRE A3

26 OCTOBRE 2024

Introduction

Dans ce laboratoire, il fallait optimiser le code du filtre FIR-LMS donné à l'aide des directives d'optimisation HLS et valider le bon fonctionnement à l'aide d'un banc de test et de la cosimulation C/RTL.

Filtre FIR-LMS

Le filtre utilisé dans le cadre de ce laboratoire est celui donné avec l'énoncé, auquel quelques modifications furent apportées. Parmi ces modifications, la taille des variables a été élevée à 32 bits pour les entrées, les sorties et les coefficients. Toutes les types de variables comprennent seulement un bit d'entier sauf pour le cas des coefficients qui en comprennent 2. Également, le fichier du banc de test a été modifié afin d'utiliser des tableaux de taille 10 dans lesquels des valeurs prédéfinies ont été assignées. Ces valeurs proviennent des fichiers de sortie du code Matlab fourni.

Une amélioration considérable entre le code écrit pour les laboratoires précédents et celui qui est fourni pour ce laboratoire est l'agglomération des opérations shift_register et accumulation dans une seule boucle de la partie FIR. Également, le code fourni permet de briser le lien de dépendance entre certaines variables en ajoutant des variables tampon dans le but de faciliter l'optimisation.

Optimisation du Code

Le code en C a été optimisé à l'aide des directives HLS de trois façons : optimisation des boucles seulement, optimisation des fonctions seulement et optimisation complète.

<u>Optimisation des boucles</u>

La solution afin d'obtenir les meilleurs résultats en termes de latence et d'intervalle a été d'ajouter les directives Pipeline et Unroll au niveau des boucles Shift_Accum_Loop et Update_Weights_Loop. Ces directives ont permis d'obtenir une latence et un intervalle minimal de 2 cycles et maximale de 4 cycles.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.19	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2	4	2	4	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	68	0	1238
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	25
Register	-	-	1336	-
Total	0	68	1336	1263
Available	650	600	202800	101400
Utilization (%)	0	11	~0	1

Optimisation des fonctions

Il a été également possible d'obtenir des résultats similaires en optimisant les fonctions au lieu des boucles. Le résultat de la synthèse comporte toute fois des différences au niveau des ressources utilisées. Les directives de Pipeline ont été utilisées sur les fonctions FIR et LMS et les tableaux utilisés dans ces mêmes fonctions ont été partitionné. Cette optimisation a permis d'obtenir les mêmes performances en termes de latence et d'intervalle mais, à un cout plus élevé en matière de ressources. Le nombre de Flip-Flops et de Look up Tables utilisés est plus important dans ce scénario d'optimisation. Il est important de souligner que la directive Dataflow a aussi été utilisée mais, celle-ci offrait de moins bonne performance en termes d'intervalle et de latence que la directive Pipeline.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.19	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2	4	2	4	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	57
FIFO	-	-	-	-
Instance	-	68	1335	1213
Memory	-	-	-	-
Multiplexer	-	-	-	25
Register	-	-	391	-
Total	0	68	1726	1295
Available	650	600	202800	101400
Utilization (%)	0	11	~0	1

Optimisation complète

Lors de l'optimisation complète, les résultats de la synthèse sont identiques aux résultats précédents de l'optimisation des fonctions seulement. Encore dans ce cas-ci, l'optimisation a été faite de sorte à obtenir les meilleurs résultats en matière de performance. Les instructions utilisées dans la fonction principale FIR_LMS sont simplement que deux directives d'array partition pour les tableaux input et updated_c. Ensuite, les sous fonctions FIR et LMS ont été pipelinées, leurs tableaux c et shift_reg ont été partitionnés et leur boucle interne ont été également pipelinées et dépliées.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.19	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2	4	2	4	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	57
FIFO	-	-	-	-
Instance	-	68	1335	1213
Memory	-	-	-	-
Multiplexer	-	-	-	25
Register	-	-	391	-
Total	0	68	1726	1295
Available	650	600	202800	101400
Utilization (%)	0	11	~0	1

Vérification RTL

La vérification RTL a permis de vérifier que le code synthétisé en HDL fonctionne bien comme on peut le voir dans la figure qui suit. Les 10 premières valeurs des sorties Matlab ont été écrites dans les tableaux du test bench afin de procéder à la vérification RTL. Comme l'erreur entre les valeurs attendues et les sorties du filtre étaient plus petite que l'epsilon, on conclut que le filtre optimisé et synthétisé est bel et bien fonctionnel.

```
Comparaison des solutions HLS VS Matlab
0.000000    0.000000
0.011697    0.011697
-0.015214    -0.015214
0.016090    0.016090
0.058589    0.058589
-0.098622    -0.098622
0.071088    0.071088
0.127212    0.127212
-0.039312    -0.039312
0.079047    0.079047
*****************************************
Reussite: Les resultats sont acceptables !!!
*****************************************
```

Questions Générales

Justifiez vos choix de directives pour chacune de vos optimisations

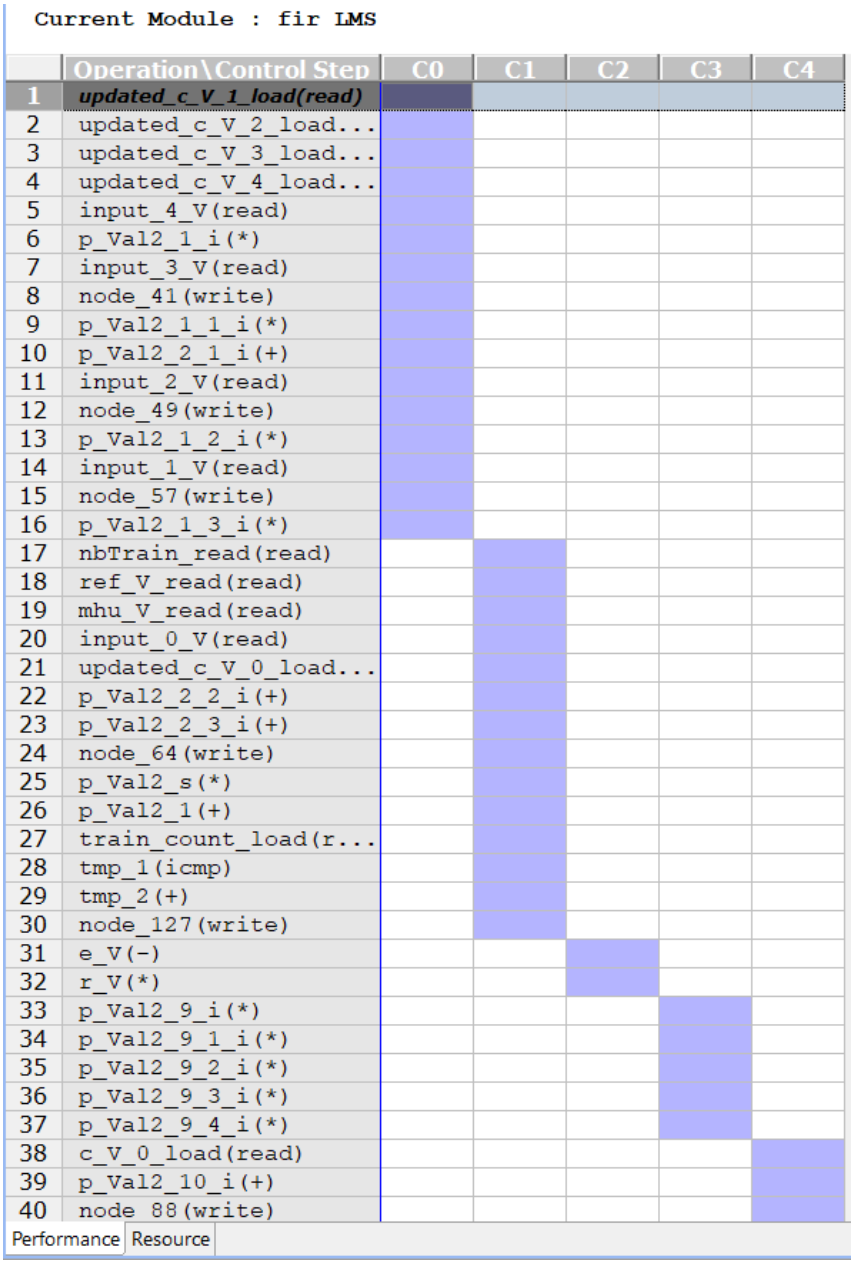
Filtre non optimisé :

Voici le Schedule Viewer du filtre lorsqu'il n'est pas optimisé :

Current Module : fir LMS

	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1	nbTrain_read(read)								
2	ref_V_read(read)								
3	mhu_V_read(read)								
4	data_V_1(read)								
5-15	⊞Shift_Accum_Loop								
16	node_51(write)								
17	updated_c_V_load_1...								
18	node_50(write)								
19	p_Val2_3(*)								
20	p_Val2_4(+)								
21	train_count_load(r...								
22	tmp_1(icmp)								
23	e_V(-)								
24	r_V(*)								
25...	⊞Update_Weights_Loop								
35	tmp_2(+)								
36	node_94(write)								
37	node_97(write)								

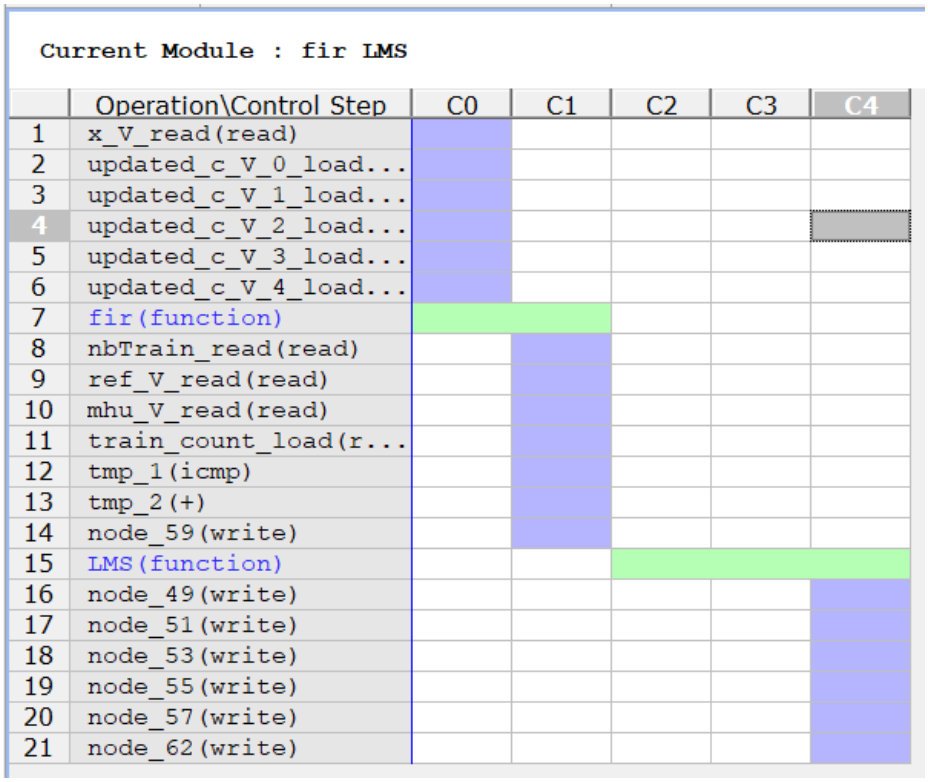
Optimisation des boucles :
On peut remarquer que la boucle Update_Weights_Loop prend plusieurs cycles d'horloge à parcourir. Afin d'optimiser les boucles, une directive de pipeline et de dépliage permettraient de réduire la latence et ainsi améliorer les performances du filtre. C'est pourquoi les directives Unroll et Pipeline ont été choisie. Elles parallélisent la boucle et permettent le démarrage des itérations plus rapidement. Voici le Schedule Viewer une fois les boucles du filtre optimisées :


Current Module : fir LMS

	Operation\Control Step	C0	C1	C2	C3	C4
1	*updated_c_V_1_load(read)*					
2	updated_c_V_2_load...					
3	updated_c_V_3_load...					
4	updated_c_V_4_load...					
5	input_4_V(read)					
6	p_Val2_1_i(*)					
7	input_3_V(read)					
8	node_41(write)					
9	p_Val2_1_1_i(*)					
10	p_Val2_2_1_i(+)					
11	input_2_V(read)					
12	node_49(write)					
13	p_Val2_1_2_i(*)					
14	input_1_V(read)					
15	node_57(write)					
16	p_Val2_1_3_i(*)					
17	nbTrain_read(read)					
18	ref_V_read(read)					
19	mhu_V_read(read)					
20	input_0_V(read)					
21	updated_c_V_0_load...					
22	p_Val2_2_2_i(+)					
23	p_Val2_2_3_i(+)					
24	node_64(write)					
25	p_Val2_s(*)					
26	p_Val2_1(+)					
27	train_count_load(r...					
28	tmp_1(icmp)					
29	tmp_2(+)					
30	node_127(write)					
31	e_V(-)					
32	r_V(*)					
33	p_Val2_9_i(*)					
34	p_Val2_9_1_i(*)					
35	p_Val2_9_2_i(*)					
36	p_Val2_9_3_i(*)					
37	p_Val2_9_4_i(*)					
38	c_V_0_load(read)					
39	p_Val2_10_i(+)					
40	node_88(write)					

Performance | Resource

Optimisation des fonctions :
Comme les boucles se retrouvent dans les fonctions, en optimisant celles-ci on peut réduire le temps d'exécution des boucles. C'est pourquoi la directive array partition a été utilisée afin d'optimiser l'accès aux variables des fonctions et qu'ensuite les fonctions ont eux-mêmes été pipelinées. Avec ces directives les fonctions peuvent parcourir les boucles plus rapidement et efficacement. Voici le Schedule Viewer une fois les fonctions du filtre optimisées :



Optimisation complète :
L'optimisation complète possède déjà toutes les directives des deux optimisations précédentes en plus d'ajouter la directive d'array partition aux tableaux input et updated_c. Comme ces tableaux sont passés aux fonctions FIR et LMS, et que dans ces fonctions les tableaux sont déjà optimisés, aucun effet n'est observable. Une directive de Dataflow a été ajoutée au niveau de la fonction principale mais, a été retirée puisque la latence était augmentée. Le Schedule Viewer après l'optimisation complète est identique au précédent, le voici:

Current Module : fir LMS

	Operation\Control Step	C0	C1	C2	C3	C4
1	x_V_read(read)					
2	updated_c_V_0_load...					
3	updated_c_V_1_load...					
4	updated_c_V_2_load...					
5	updated_c_V_3_load...					
6	updated_c_V_4_load...					
7	fir(function)					
8	nbTrain_read(read)					
9	ref_V_read(read)					
10	mhu_V_read(read)					
11	train_count_load(r...					
12	tmp_1(icmp)					
13	tmp_2(+)					
14	node_59(write)					
15	LMS(function)					
16	node_49(write)					
17	node_51(write)					
18	node_53(write)					
19	node_55(write)					
20	node_57(write)					
21	node_62(write)					

Analysez et comparez la différence en termes de latences, de ressources utilisées et de période minimale pour chacun des types d'optimisations demandés
Les trois optimisations sont équivalentes en termes de latence et de période minimale d'horloge avec 2 à 4 cycles de latence et 8,19ns de période estimée. Les synthèses n'utilisent pas de BRAM et l'utilisation des DSP est pareil dans les trois optimisations. Là où l'optimisation des boucles est plus avantageuse est en termes de ressources. En effet, celle-ci utilise moins de Flip-Flops et de Look up Tables que l'optimisation des fonctions ainsi que l'optimisation complète. L'optimisation des boucles serait alors le meilleur choix des trois.

Discutez l'effet de la directive *inline* sur les ressources utilisées
La directive Inline permet de diminuer la latence au cout de quelques ressources supplémentaires. La latence passe de 12 – 34 sans la directive à 11 – 32 avec la directive. Si la fonction est appelée à plusieurs endroits dans le code, la directive Inline permettrait de réduire la latence du système en répliquant le code de la fonction à l'endroit où elle est appelée.

Dans quelles circonstances une directive de type *inline off* pourrait être utile

La directive Inline off pourrait être utile dans le contexte où les ressources sont précieuses et qu'une fonction est appelée à plusieurs reprise. La directive permettrait de limiter l'utilisation des ressources en s'assurant que l'outil de synthèse ne l'utilise pas automatiquement. La logique de la fonction serait alors présente dans le matériel sous forme d'une seule entité physique.

Montrez un exemple concret du bon fonctionnement de votre filtre

On peut observer le signal nbTrain prendre une valeur de 0x12C, ce qui représente 300. En retournant dans le code du banc de test on peut valider cette valeur et confirmer que le filtre prend bel et bien des valeurs en entrées. Également, on peut observer la dixième sortie y avec la valeur 0x0A1E37C1 qui se traduit par 0.00010100001111000110111111000001 sur 32 bits avec un bit d'entier. En décimal la valeur est de 0.0790471 ce qui correspond à notre dixième sortie de référence. On peut alors en déduire le bon fonctionnement du filtre puisqu'à ce point une erreur dans les calculs serait remarquable.