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 A2

15 OCTOBRE 2023

Introduction

Dans ce laboratoire, il fallait optimiser le filtre FIR-LMS fait dans le cadre du laboratoire précédent en utilisant des variables de type virgule fixe et valider l'implémentation à l'aide d'un banc de test.

Filtre FIR LMS

Le filtre est semblable à celui du laboratoire A1 en termes d'architecture et de calculs, la seule différence étant les types de variables ayant passés de float à virgule fixe. Plusieurs types ont été créés dont un pour les données, un pour les poids, un pour le pas d'adaptation, un pour représenter un Boolean et un pour les entiers.

```
void FIR_LMS(T_FIR_LMS_data *inputEstimate,
             T_FIR_LMS_data *inputNoise,
             T_FIR_LMS_mu mu,
             T_FIR_LMS_data *input,
             T_FIR_LMS_bool isTraining)
```

Type	Bits d'entier	Bits fractionnaire
T_FIR_LMS_data	1	10
T_FIR_LMS_weights	2	12
T_FIR_LMS_mu	2	12
T_FIR_LMS_bool	1	0
T_FIR_LMS_int	4	0

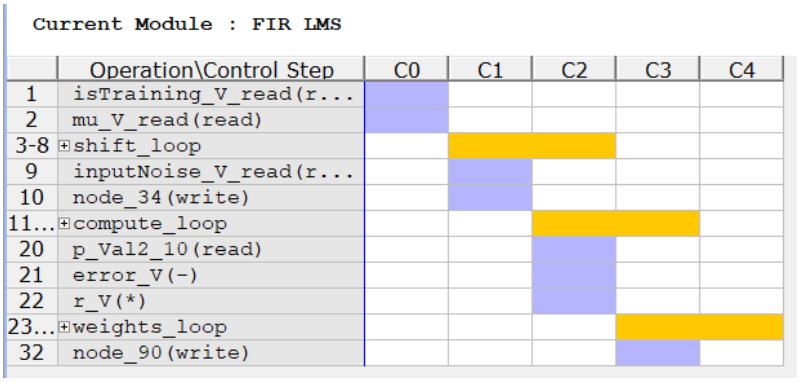
Les dimensions des variables ont été déterminées de façon itérative, en commençant par des variables à 30 bits fractionnaire et en diminuant le nombre de bits significatif à force de correspondre aux exigences de précision du banc de test.

Test Bench

Le banc de test est très similaire à celui du laboratoire A1 à l'exception de l'utilisation des nouveaux types de variables et de la lecture des fichiers .txt en convertissant implicitement les valeurs en point fixe. Également, au lieu de comparer l'erreur accumulé le banc de test compare individuellement chaque sortie du filtre.

Analyse de la synthèse

Dans le schedule viewer, on remarque par les cases en orange qu'il y a trois boucles dans le filtre FIR-LMS. Chaque boucle pourra être pipelinée et parallélisée individuellement mais, les trois boucles pourront également être misent en parallèle afin d'améliorer la rapidité du filtre en


Current Module : FIR LMS

	Operation\Control Step	C0	C1	C2	C3	C4
1	isTraining_V_read(r...					
2	mu_V_read(read)					
3-8	⊞shift_loop					
9	inputNoise_V_read(r...					
10	node_34(write)					
11...	⊞compute_loop					
20	p_Val2_10(read)					
21	error_V(-)					
22	r_V(*)					
23...	⊞weights_loop					
32	node_90(write)					

utilisant les données des boucles précédentes dès qu'elles sont prêtes.

Pour les tableaux de données, le tableau des poids nommé w et le tableau des entrées nommé yn_input pourront être partitionné afin d'améliorer la vitesse d'accès à leurs valeurs et ainsi éviter des goulots d'étranglement.

Questions générales

Différences en termes de ressources et de latence

Les différences en termes de ressources et de latence entre la synthèse en virgule fixe et en virgule flottante est bien présente. En effet, la latence a été réduite à moins de la moitié, passant de 54 – 96 coups d'horloge à 21 – 31 coups d'horloge comme le démontre les figures suivantes (virgule flottante à gauche, virgule fixe à droite).

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.42	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
54	96	54	96	none

Detail

Instance

Loop

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.39	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
21	31	21	31	none

Detail

Instance

Loop

Également, l'utilisation des ressources a grandement diminuée. On peut voir le nombre de DSP48 passer de 5 pour la virgule flottante à 3 pour la virgule fixe et les FF et les LUT passent respectivement de 688 et 699 pour le filtre a virgule flottante à 123 et 208 pour le filtre a virgule fixe. En parallélisant les boucles dans le prochain laboratoire, on pourrait s'attendre a voir le nombre de ressources utilisées augmenté puisque cette technique d'optimisation a pour conséquence d'augmenter l'utilisation des ressources matérielles.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	63
FIFO	-	-	-	-
Instance	-	5	355	349
Memory	0	-	128	6
Multiplexer	-	-	-	281
Register	-	-	205	-
Total	0	5	688	699
Available	650	600	202800	101400
Utilization (%)	0	~0	~0	~0

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	3	-	-
Expression	-	-	0	81
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	50	3
Multiplexer	-	-	-	124
Register	-	-	73	-
Total	0	3	123	208
Available	650	600	202800	101400
Utilization (%)	0	~0	~0	~0

Différence entre #pragma HLS DATAFLOW et #pragma HLS PIPELINE

La directive #pragma HLS PIPELINE a pour but d'optimiser les boucles en parallélisant différentes étapes de la même boucle. Cette directive sert à réduire le nombre de coups d'horloge entre le début et la fin de chaque itération ce qui augmente le throughput. La directive #pragma HLS DATAFLOW a pour but de paralléliser les fonctions ou les blocs de code de sorte à ne pas attendre la fin d'une fonction pour appeler la suivante. Cette directive agit un peu comme un pipeline mais au niveau supérieur entre les fonctions et les boucles.

Différence en termes de ressources utilisées et en termes de latences entre une directive qui pipeline la boucle interne seulement et une directive qui pipeline la boucle externe seulement ?

Lorsqu'on pipeline la boucle externe, la latence du système va grandement diminuer parce qu'on n'attend pas la fin de la dernière boucle externe avant de lancer la suivante. Cela dit, l'utilisation des ressources va grandement augmenter puisqu'on parallélise à la fois la boucle externe et la boucle interne qui y est contenue.

Lorsqu'on pipeline la boucle interne seulement, la latence du système va également diminuer mais, l'impact est moins important que dans le cas précédent puisque la boucle externe reste non optimisée. Cependant, comme seulement la boucle interne est parallélisée, l'utilisation des ressources est moins grande.

Le choix de quelle boucle on veut optimiser sera fait en fonction de la latence désirée du système et des ressources disponibles.