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 A1

8 OCTOBRE 2023

Introduction

Le but de ce laboratoire était de conceptualiser un filtre de type FIR-LMS en C à l'aide de l'outil Vivado HLS et de le valider à l'aide d'un test bench et des fichiers générés par Matlab. Il a également été nécessaire de synthétiser le filtre afin d'observer ses performances actuelles dans le but de les améliorer dans les prochains laboratoires.

Filtre FIR-LMS

Le filtre FIR-LMS prends en entrée trois pointeur vers : la sortie (inputEstimate), l'entrée bruitée (inputNoise) et l'entrée de référence(input) ainsi que le pas d'adaptation (mu) et une consigne d'entrainement (isTraining). La consigne d'entrainement remplace le nombre de données utilisées pour l'entrainement.

```
void FIR_LMS(float *inputEstimate,
             float *inputNoise,
             float mu,
             float *input,
             bool isTraining)
```

Dans le filtre, un tableau pour les poids (w) et un tableau pour les 5 dernières entrées (yn_input) sont déclarés comme static afin de garder en mémoire leurs valeurs précédentes. La première boucle nommée shift_loop sert à décaler les valeurs dans le tableau yn_input. On attribue ensuite la nouvelle valeur d'entrée à la position 0 une fois que les valeurs précédentes sont déplacées. On exécute ensuite la boucle nommée compute_loop où on y calcul la valeur de la sortie à l'aide des coefficients de poids calculés et des entrées. L'erreur sur la sortie est ensuite calculée. Avant de sortir de la fonction, on ajuste les coefficients de poids dans la boucle weights_loop si le filtre est en condition d'entrainement (isTraining == true).

```
static float w[CHANNEL_ORDER] = {0};
static float yn_input[CHANNEL_ORDER] = {0};
float error = 0;

int i;

/* shift values inside yn_input[] */
shift_loop: for(i = (CHANNEL_ORDER - 1u); i > 0; i--)
{
    yn_input[i] = yn_input[i - 1u];
}
yn_input[0] = *inputNoise;

/* compute the input's estimate */
compute_loop: for(i = 0; i < CHANNEL_ORDER; i++)
{
    *inputEstimate += w[i] * yn_input[i];
}

/* compute error */
error = *input - *inputEstimate;

if (isTraining)
{
    /* adjust weights */
    weights_loop: for(i = 0; i < CHANNEL_ORDER; i++)
    {
        w[i] = w[i] + (mu * error * yn_input[i]);
    }
}
```

Le filtre est inspiré du code Matlab fourni «FIR_LMS_EQ.m» et n'a pas encore été optimisé. Pour l'instant, seulement la validité du code en C a été vérifié par le test bench et par la synthétisation. On peut observer les estimés de performance dans la figure suivante :

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

Validation du filtre

La validation du filtre s'est faite par le test bench dans Vivado. Ce dernier lit les fichiers générés par Matlab et place les variables dans différents tableaux. Ensuite, une boucle for appelle la fonction FIR_LMS avec les valeurs précédemment lues une par une. Lorsque le nombre d'itérations atteint 19, le pas de convergence est changé de 0,52 à 0,22. Également, à la 200^{e} itération, la variable isTraining est changée à false. Ceci permet de répliquer le comportement du filtre sur Matlab. Une fois que toutes les valeurs sont passées au filtre, le test bench vérifie les erreurs de calculs et affiche les résultats dans la console et dans un fichier de sortie afin de pouvoir comparer les résultats. Un message est ensuite affiché en fonction de l'erreur accumulée. La somme des erreurs absolues entre les sorties Matlab et le filtre est de 0.00052930 pour les 2000 échantillons, ce qui donne une erreur moyenne de 0,00000026. On peut observer le résultat de la simulation dans la console:

Console | Errors | Warnings | DRCs | Debugger Console

Vivado HLS Console

```
1987    0.48307535    0.48307499
1988    0.50473648    0.50473601
1989    -0.48044690    -0.48044699
1990    0.48233977    0.48234001
1991    0.51688087    0.51688099
1992    -0.49378037    -0.49377999
1993    0.49205598    0.49205601
1994    0.49947605    0.49947599
1995    0.49110037    0.49110001
1996    0.50989389    0.50989401
1997    0.48307535    0.48307499
1998    0.50473648    0.50473601
1999    0.50058603    0.50058597
*****************************************
PASS: The output matches the Matlab output!
*****************************************
Accumulated |error|: 0.00052930
Average error:       0.00000026
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

Questions générales

Quelle est la différence entre la synthèse et la simulation ?

La simulation permet de vérifier que le composant se comporte tel qu'on s'attend sans synthétiser le code, elle sert à valider la logique du code. La synthèse, elle, permet de transformer le code en C vers un code de type description matériel.

Quelle peut être l'utilité d'avoir plusieurs solutions dans un même projet ?

Avoir plusieurs solutions dans un même projet permet de garder une trace des optimisations faites. On peut copier les directives d'une solution afin de continuer le processus d'optimisation tout en gardant les itérations précédentes afin de les comparer entre eux. Ceci permet d'explorer plusieurs pistes d'optimisation tout en sauvegardant tous les états intermédiaires de la solution en cours.

Expliquez et discutez les résultats de la synthèse.

L'horloge estimée du système dans le pire scénario est de 8,42 ns, ce qui respecte le délais requis de 10 ns moins l'incertitude de 1,25 ns. La latence du composant est minimalement de 54 coups d'horloge et maximalement de 96 coups d'horloge. Cet écart d'environ 40 coups d'horloge est la différence entre quand le filtre est en entrainement et quand il est en opération normale.

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- shift_loop	8	8	2	-	-	4	no
- compute_loop	40	40	8	-	-	5	no
- weights_loop	40	40	8	-	-	5	no

Dans le résumé des ressources estimées, ont peut voir que le design utilise 5 DSP48E. Cette figure peut sembler importante pour un filtre FIR-LMS. Le design a besoin d'autant de DSP48E parce que les données sont des float qui utilisent 32 bits. Dans le prochain laboratoire, en utilisant des nombres à point fixes, il sera possible de diminuer le nombre de DSP48E.

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

Une fois la synthèse effectuée, quelle est la différence entre «Latency» et «Interval» ?

La Latency est le nombre de coups d'horloge que le composant prend avant de sortir son résultat. L'Interval est le nombre de coups d'horloge avant de pouvoir prendre de nouvelles données en entrée.

Quelle est la différence en termes de ressources utilisées et de fréquence maximale d'horloge lorsque vous synthétisez votre filtre avec des variables de type float et de type int?
Lorsque synthétisé avec des int au lieu des float, la période de l'horloge diminue un peut, passant à 8,15 ns. La latance et l'interval sont également coupé d'un facteur de 2 pour le minimum et de 3 pour le maximum. En termes de ressources, plus de DSP48E sont utilisées mais les FF passent de 688 à 221 et les LUT passent de 699 à 373. L'utilisation de float permet d'utiliser une horloge de 118.76MHz alors que les int permettent une horloge de 122.70MHz.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.15	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
21	36	21	36	none

Detail

Instance

Loop

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	9	0	243
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	128	6
Multiplexer	-	-	-	124
Register	-	-	93	-
Total	0	9	221	373
Available	650	600	202800	101400
Utilization (%)	0	1	~0	~0