GEI1064 Conception en VLSI
Filtre adaptatif FIR-LMS
Laboratoire A2

Jeremy Poupart et M. Ahmed Ouameur

OBJECTIFS

- ✓ Évaluer votre filtre FIR-LMS avec des variables en virgule fixe et valider le bon fonctionnement à l'aide d'un test Bench
- ✓ Analyser votre filtre FIR-LMS pour déterminer les directives à ajouter pour l'optimisation

TYPES EN VIRGULE FIXE

Pour cette partie, vous devrez recréer un nouveau projet HLS avec des fichiers C++ (.cpp). Vous pouvez copier votre code du premier laboratoire dans votre nouveau projet. Le langage C++ permettra d'utiliser la bibliothèque de Xilinx ap_fixed.h afin de pouvoir utiliser des variables à virgule fixe. Vous pouvez vous référer au chapitre 5 du tutoriel pour plus d'information sur les variables à virgule fixe.

Toutes les variables dans votre code devront être à virgule fixe. C'est vous qui devrez déterminer le nombre de bits d'entiers (avant la virgule) et le nombre de bits fractionnaires (après la virgule) nécessaires pour chaque variable. Pour vous aider, vous pouvez regarder l'ordre de grandeur que prend chaque variable dans le code Matlab fourni. Pour garder les choses simples, il n'est pas nécessaire de se soucier de la plage dynamique des valeurs et il est permis que chaque variable ait le même nombre de bits d'entiers et de bits fractionnaires (pour votre projet de session, vous pouvez voir comment utiliser des variables avec des longueur binaires différentes), en autant que les résultats soient assez précis pour passer le test Bench.

Pour valider votre design, votre test Bench devra comporter les aspects suivants :

- ✓ Le code Matlab fourni qui peut écrire dans des fichiers .txt les valeurs des variables de type double yn, inp et inpest
- ✓ Une section du code C (Test Bench dans HLS) qui pourra récupérer les valeurs de yn, inp et inpest et convertir implicitement les types de yn et inp de double vers virgule fixe
- ✓ Une section du code C (Test Bench dans HLS) qui appliquera les valeurs récupérées à la fonction du filtre FIR-LMS
- ✓ Une section du code C (Test Bench dans HLS) qui comparera les résultats de sortie obtenus avec Matlab et avec votre fonction afin de valider le bon fonctionnement de votre filtre FIR-LMS

Afin de comparer vos résultats avec les résultats de type double de Matlab, vous devrez convertir explicitement vos résultats de type virgule fixe vers double et vérifier que la valeur absolue de l'écart

entre chaque paire de sortie (sortie Matlab et sortie HLS) ne dépasse pas 0.01. La section Annexe peut vous aider pour l'élaboration de votre test Bench en C++.

ANALYSE DE LA SYNTHÈSE

Pour cette partie, vous devrez analyser votre système après la synthèse afin de déterminer les directives nécessaires pour améliorer les performances de votre filtre FIR-LMS. Votre analyse devra comprendre les points suivants :

- ✓ Une inspection détaillée du *Schedule viewer* afin de repérer les endroits de votre code qui pourraient être parallélisés et/ou pipelinés
- ✓ Une analyse de vos tableaux de variables qui pourraient être séparés si nécessaire (*array partitioning*) afin d'éviter les goulots d'étranglement (*bottle neck*)

Cette partie se résume donc à une analyse logique de votre part à l'aide d'images (*Schedule viewer*) et d'explications pour démontrer les points que vous avancez. Vous pouvez tester vos hypothèses en donnant des directives à HLS de type #pragma, mais seule votre analyse sera évaluée pour cette partie. C'est dans le prochain laboratoire que sera évaluée l'efficacité de vos directives.

QUESTIONS GÉNÉRALES

- ✓ Discutez les différences en termes de ressources et de latences entre les synthèses en virgule fixe et en virgule flottante
- ✓ Quelle est la différence entre #pragma HLS DATAFLOW et #pragma HLS PIPELINE ?
- ✓ Supposons qu'une partie de code contient deux boucles imbriquées avec du code seulement dans la boucle la plus interne. Quelle est la 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 ?

ANNEXE

Pour lire dans un fichier .txt et faire la conversion implicite en même temps :

```
#include <cmath>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

in_t input[nbData]; // tableau de type virgule fixe

ifstream Fin;


Fin.open("yn.txt");

for (int i = 0; i < nbData; ++i)
{
    Fin >> input[i]; // conversion implicite de double vers point fixe
}

Fin.close();
```