Laboratoire 9: Convolution

Département: **TIC**

Unité d'enseignement: **SCF**

Auteur(s):

- **CECCHET Costantino**

Professeur:

- **DASSATTI Alberto**
- **YANN Thomas**

Assistant:

- **JACCARD Anthony I.**

Date:

- **22/05/2024**

[Page de mise en page, laissée vide par intention]

Introduction

Ce laboratoire va nous permettre de développer de A à Z un système complet pour calculer une convolution sur FPGA.

Implémentation

Notre objectif est de réaliser une convolution sur une image en utilisant un noyau de convolution de taille 3x3, en utilisant un FPGA.

Nos test on été effectués avec un image de taille maximale de 348x348, il se peux qu'avec des images plus grandes, le programme ne fonctionne pas correctement. les images doivent être des .png comme pour nos tests.

Pour ce faire, nous allons utiliser la DE1-SoC.

Notre première étape est de réaliser une IP qui va permettre de réaliser la convolution, nous somme partie sur une IP qui utilise des FIFO pour stocker les valeurs de l'image et de registres pour stocker les valeurs du noyau.

Dés que les données sont dans la FIFO d'entrée, on peux commencer les calculs, on va multiplier les valeurs de l'image par les valeurs du noyau et les additionner pour obtenir la valeur de la convolution. Ces operations se font ligne par ligne de 3 pixels.

Nous utilisons des FIFO de 32 bit pour stocker 3 pixels de l'image à la fois, et des registres de 32 bit pour stocker les valeurs du noyau. Ceci simplifie le calcul de la convolution. Chaque ligne calculée est stockée dans une FIFO de sortie et pour finir cette FIFO est lue 3 fois d'affilée pour obtenir les 3 valeurs de la convolution on les additionne et on les stocke dans une FIFO de sortie. Cette dernière FIFO est donc lue par notre utilisateur pour obtenir le valeur de la convolution.

Conception

voici la liste des registres et des FIFO utilisées dans notre IP:

offset	nom	taille	description
0x00	Constant	32 bit	valeur constante 0xBADB100D
0x04	kernel[0-2]	32 bit	valeurs du noyau de convolution
0x08	kernel[3-5]	32 bit	valeurs du noyau de convolution
0x0C	kernel[6-8]	32 bit	valeurs du noyau de convolution
0x10	img[0-2]	32 bit	valeurs de l'image par 3 pixels
0x14	test fifo	32 bit	fifo de test
0x18	fifo sizes	32 bit	taille des FIFO
0x1C	return value	32 bit	valeur de la convolution
0x20	control register	16 bit	registre de contrôle pour la lecture ou écriture des FIFO

déroulement de la convolution

Comment ce déroule la convolution:

1. Avant de commencer une convolution, l'utilisateur doit charger les valeurs du noyau dans les registres correspondants.
2. Une fois le noyau chargé, l'utilisateur peux commencer à charger les valeurs de l'image dans la FIFO d'entrée cest données doivent être chargées par 3 pixels, c-à-dire la première ligne du segment 3x3 que l'on va convoluer, puis la deuxième ligne et enfin la troisième ligne.
3. Une fois 3 pixels chargés, le calcul de la convolution commence, on multiplie les valeurs de l'image par les valeurs du noyau et on additionne le tout pour obtenir la valeur de la convolution. Cette dernière est disponible que lorsque les 3 lignes ont été calculées. Si les données ne sont pas encore prêtes, l'utilisateur peux attendre ou charger de nouvelles données tant que la FIFO d'entrée n'est pas pleine. Avant chaque lecture et écriture, l'utilisateur doit vérifier que la FIFO est prête à être lue ou écrite.

Des registres de contrôle sont utilisés pour vérifier si les FIFO sont prêtes à être lues ou écrites. En lisant a l'offset 0x20, si la FIFO est prête à être lue, le deuxième bit est à 1, si la FIFO est prête à être écrite, le premier bit est à 1. Les 4 bits suivants sont utilisés pour voir le status des FIFO. De la même manière, En lisant a l'offset 0x18, on peut lire la taille actuelle des FIFO.

bit	description
0	fifo d'entrée pas pleine
1	fifo de sortie pas vide
4	fifo de sortie pleine
5	fifo d'entrée vide

bits 31-24	bits 23-16	bits 15-8	bits 7-0
taille fifo de sortie	-	taille fifo d'entrée	taille fifo de test

Ces bits sont utiles si l'on souhaite savoir ou en est notre convolution.

Pseudo code

L'utilisateur doit suivre les étapes suivantes pour réaliser une convolution:

1. Charger les valeurs du noyau dans les registres correspondants.
2. Charger les valeurs de l'image dans la FIFO d'entrée.
3. Attendre que la FIFO de sortie ne soit pas vide.
4. Lire la valeur de la convolution.
5. Répéter les étapes 2 à 4 jusqu'à ce que toutes les valeurs de l'image aient été lues.

VHDL

Nous utilisons des FIFO de Intel pour stocker les valeurs de l'image et de la convolution, voici un exemple de la déclaration d'une FIFO, pour une FIFO de 256x32:

```
scfifo_component : scfifo

    GENERIC MAP (
        add_ram_output_register => "ON",
        intended_device_family => "Cyclone V",
        lpm_numwords => 256,
        lpm_showahead => "ON",
        lpm_type => "scfifo",
        lpm_width => 32,
        lpm_widthu => 8,
        overflow_checking => "ON",
        underflow_checking => "ON",
        use_eab => "ON"
    )
    PORT MAP (
        aclr => aclr,
        clock => clock,
        data => data,
        rdreq => rdreq,
        wrreq => wrreq,
        empty => sub_wire0,
        full => sub_wire1,
        q => sub_wire2,
        usedw => sub_wire3
    );
```

Cette fifo est importée dans notre IP et utilisée pour stocker les valeurs de l'image et de la convolution.

les images sont stockées de la manière suivante:

bits 31-24	bits 23-16	bits 15-8	bits 7-0
pixel 0	pixel 1	pixel 2	-
pixel 3	pixel 4	pixel 5	-
pixel 6	pixel 7	pixel 8	-

Pour les valeurs du noyau, elles sont stockées de la manière suivante:

bits 31-24	bits 23-16	bits 15-8	bits 7-0
valeur 0	valeur 1	valeur 2	-
valeur 3	valeur 4	valeur 5	-
valeur 6	valeur 7	valeur 8	-

Le noyau de base pour la convolution peux être choisit par l'utilisateur, il est possible de le modifier en modifiant le code source de notre IP.

5 noyaux sont disponibles, ils sont stockés dans le programme utilisateur et chargés dans les registres correspondants avant de commencer la convolution.

Ils sont les suivants et peuvent être choisis grace à un argument lors de l'exécution du programme utilisateur:

```
/* Identity 0*/
{
    {0, 0, 0},
    {0, 1, 0},
    {0, 0, 0}},
/* Edge detection 1*/
{
    {0, 1, 0},
    {1, -4, 1},
    {0, 1, 0}},
/* Sharpen 2*/
{
    {0, -1, 0},
    {-1, 5, -1},
    {0, -1, 0}},
/* Box blur 3*/
{
    {1, 1, 1},
    {1, 1, 1},
    {1, 1, 1}},
/* Gaussian blur 4*/
{
    {1, 2, 1},
    {2, 4, 2},
    {1, 2, 1}}};
```

Il est possible de modifier les valeurs des noyaux en modifiant le code source de notre programme utilisateur.

La separation des valeurs du noyau et de l'image en 3 pixels permet de simplifier le calcul de la convolution, on multiplie les valeurs de l'image par les valeurs du noyau et on additionne le tout pour obtenir la valeur de la convolution.

Driver

Un driver est nécessaire pour communiquer avec notre IP, ce driver est écrit en C et permet de charger les valeurs du noyau et de l'image, de lire les valeurs de la convolution et de vérifier le status des FIFO.

Le driver est basé sur celui d'un labo précédent, il a été modifié pour correspondre à notre IP.

Il utilise les read et write pour communiquer avec notre IP et ioctl pour modifier les valeurs aux quelles on souhaite accéder.

Les offsets sont donc les suivants pour lire:

offset	nom	taille	description
0x00	kernel[0-2]	32 bit	valeurs du noyau de convolution
0x01	kernel[3-5]	32 bit	valeurs du noyau de convolution
0x02	kernel[6-8]	32 bit	valeurs du noyau de convolution
0x03	test_fifo	32 bit	fifo de test
0x04	fifo_sizes	32 bit	taille des FIFO
0x05	return value	32 bit	valeur de la convolution
0x06	control register	16 bit	registre de contrôle pour la lecture ou écriture des FIFO
0x07	constant	32 bit	valeur constante 0xBADB100D

Pour écrire:

offset	nom	taille	description
0x00	kernel[0-2]	32 bit	valeurs du noyau de convolution
0x01	kernel[3-5]	32 bit	valeurs du noyau de convolution
0x02	kernel[6-8]	32 bit	valeurs du noyau de convolution
0x03	test_fifo	32 bit	fifo de test
0x04	img[0-2]	32 bit	valeurs de l'image par 3 pixels

Ceci facilitera l'utilisation de notre IP par le biais de notre driver.

Lors de l'écriture ou de la lecture si les offsets ne sont pas corrects, le driver renverra une erreur.

Le driver sera compilé avec la commande suivante depuis le PC hôte:

```
make
```

Puis il sera copié sur la carte SD de la DE1-SoC pour être utilisé.

```
cp convolution.ko <path_SD>
```

Puis une fois sur la carte, il sera chargé avec la commande suivante:

```
insmod convolution.ko
```

Programme utilisateur

Le programme utilisateur à tout d'abord été écrit en utilisant le fichier /dev/mem pour communiquer avec notre IP, puis il a été modifié pour utiliser le driver que nous avons écrit.

Ceci dans un but de simplification de l'utilisation de notre IP et de la phase de debug sans driver initiallement.

Plusieurs define sont à disposition pour choisir le mode d'utilisation avec ou sans driver et avec ou sans messages de debug.

lancer le projet

Pour lancer le programme il faut suivre les étapes suivantes depuis la DE1-SoC:

ceci va lancer la convolution sur une image de test fournie.

```
./conv <kernel_id> <source_image> <destination_image>
```

avec la ligne suivante on peux tester la convolution sur une image de 5x5 pixels fournie avec le programme:

```
./conv <kernel_id>
```

ceci va lancer la convolution sur l'image suivante avec le noyau de convolution choisi par l'utilisateur:

```
int image[IMG_SIZE][IMG_SIZE] = {
    {0, 0, 0, 0, 0},
    {0, 1, 1, 1, 0},
    {0, 1, 0, 1, 0},
    {0, 1, 1, 1, 0},
    {0, 0, 0, 0, 0}};
```

et avec cette dernière ligne on peux tester le fonctionnement de la FIFO de test:

```
./conv
```

Le kernel de convolution est déjà chargé dans le programme, il est possible de le modifier en modifiant le code source du programme. Les make files sont fournis pour le driver et le programme utilisateur.

Les fichiers doivent être copiés sur la carte SD de la DE1-SoC pour être utilisés.

Conclusion

Ce laboratoire nous a permis de réaliser une IP de convolution sur FPGA, nous avons pu voir comment communiquer avec cette IP en utilisant un driver et un programme utilisateur.

Ceci nous a permis de décentraliser une operation sur FPGA et de la rendre accessible à un utilisateur. Nous avons pu voir comment utiliser des FIFO pour stocker des données et comment les lire et les écrire.