Rapport architecture embarquée

Buot Adrien - Coville Benoit - Guicharnaud Léo

Ce cours a été découpé en 3 TP + un projet : ces TP nous ont permis de prendre en main les 3 aspects du software/hardware co-design C++-VHDL. Donc de la programmation qui fait appel à un microcontrôleur ou microprocesseur (dans notre cas un microcontrôleur) qui est interfacé avec un hardware accelerator VHDL.
Ce type de programmation se découpe en 3 grandes parties :

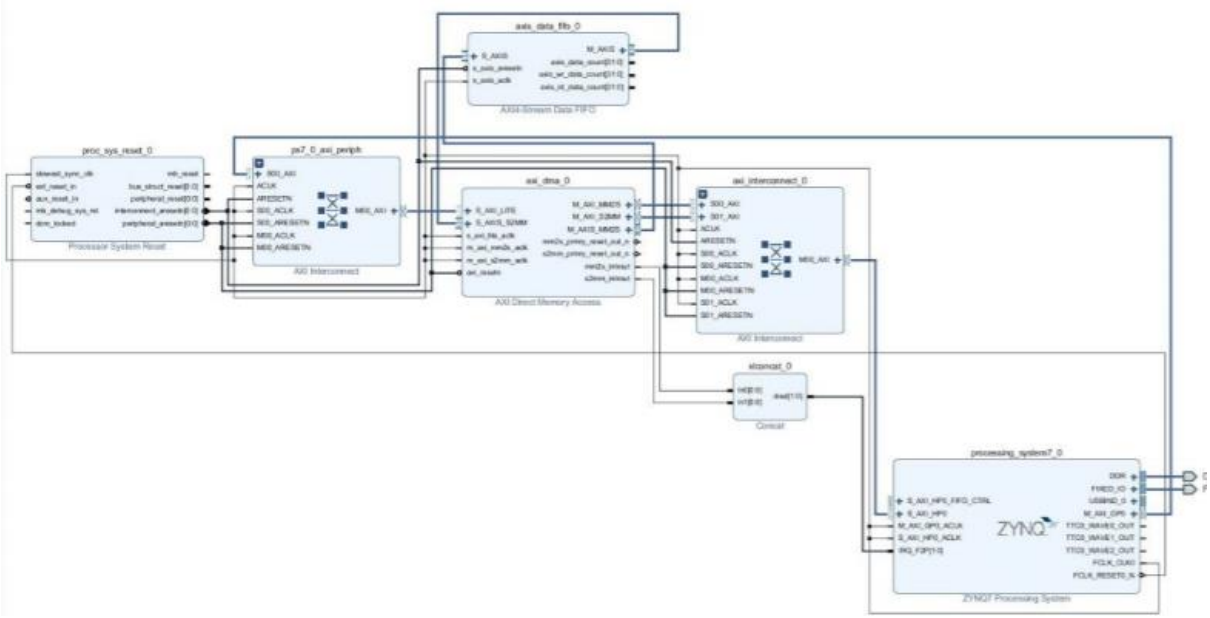
- l'acquisition de l'environnement de programmation
- la programmation du code VHDL
- la programmation du driver qui va envoyer et recevoir les informations de l'hardware accelerator.

Le TP1 nous a permis de prendre en main ce premier aspect de ce type de programmation. Le TP2 nous à permis de prendre en main la programmation du driver, et le TP3 la programmation du code VHDL. Le projet quant à lui nous a permis de mettre à l'épreuve nos nouvelles compétences en appliquant ces 3 domaines de développement.

Buot Adrien
Coville Benoit
Guicharnaud Léo

Lab 1 : Hardware design

Ce TP était le premier tp dans cette matière : le but de ce TP était de prendre en main l'acquisition de l'environnement de travail via le logiciel VIVADO. Ce logiciel nous à permis de créer, compléter et implémenter un design hardware pour la carte ZedBoard Zynq.
Avec ce logiciel, nous avons sélectionné, placé et créé les liaisons entre les différents composants de notre design.



Avec ce logiciel, nous avons eu la possibilité de programmer les adresses master, slave et taille d'adresse pour notre DMA.
Une fois que ce design a été fini, nous l'avons exporté sous forme de bitstream. Ce bitstream peut être exécuté par le linux embarqué sur la carte SD fournie. Ce bitstream permet au linux d'avoir accès aux composants de la ZedBoard.
Quand nous avons booté, la carte nous a demandé un identifiant ainsi qu'un mot de passe. Nous avons saisi "root" pour les deux, et nous avons eu accès à un shell linux. Nous nous sommes déplacés dans l'explorateur de fichier avec les commandes « ls » , « cd »… et nous avons exécuté le programme hello_word avec la commande « ./hello_word ». L'exécution de ce programme nous permet de savoir si les configurations faites sur vivado sont correctes.

Lab 2 : Software design

Ce TP nous a permis de prendre en main la programmation du driver qui va envoyer et recevoir les informations de l'hardware accelerator.

Pour cela, le bitstream nous a été fourni ainsi que ses entrées et sorties. Nous avons donc écrit un fichier DMA.cpp (en langage C++) capable d'envoyer, de réceptionner et d'afficher le résultat de l'opération effectuée par l'hardware accelerator. Nous avons dû cross-compiler le fichier afin qu'il puisse être interprété par le linux embarqué sur la carte SD fournie : la fonction de cross compilation que nous avons utilisé était : `arm-linux-gnueabi-g++-8 DMA.cpp -o prog`.

Une fois le code cross-compilé et ajouté sur la bonne partition de la carte, nous avons pu le tester et nous avons remarqué que,nous envoyons « hello », et nous recevons « rovvy ».

```
Status: running
Status: running idle IOC_Irq
Status: running idle IOC_Irq
0000: 00 0a 05 00 68 65 6c 6c 6f                    ....hello
0000: 72 6f 76 76 79                                rovvy
#
```

Après analyse de la trame envoyée, nous avons observé qu'en faisant varier le deuxième bit de la trame, le programme fonctionne toujours, cependant, le message reçu change : en remplaçant 10 par 2, le message reçu n'est plus « rovvy » mais « jgnnq ».

```
# ./prog
Status: running
Status: running idle IOC_Irq
Status: running idle IOC_Irq
0000: 00 02 05 00 68 65 6c 6c 6f                    ....hello
0000: 6a 67 6e 6e 71                                jgnnq
#
```

On en déduit donc que l'hardware accelerator fais chiffrement par un code de césar. Le code César est un chiffrement basé sur un décalage de l'alphabet (déplacement des lettres plus loin dans l'alphabet), il s'agit d'une substitution monoalphabétique, c'est-à-dire qu'une même lettre n'est remplacée que par une seule autre (toujours identique pour un même message).

Voilà une capture d'écran du code que nous avons utilisé pour le fichier DMA.cpp

```cpp
#include "DMA.hpp"
#include <unistd.h>
#include <string.h>
#include <iostream>
using namespace std;

#define DMAREGISTER 0x40400000
#define MM2S 0x0e000000
#define S2MM 0x0f000000

int main()
{
    DirectMemoryAccess DMA(DMAREGISTER,MM2S,S2MM);
    unsigned int status=0;

    //On stoppe le DMA
    DMA.reset();
    DMA.halt();

    //Ecriture des 4 bits pour le hardware accelerator
    DMA.writeSourceByte(0); //0
    DMA.writeSourceByte(2); //10
    DMA.writeSourceByte(5); //5
    DMA.writeSourceByte(0); //0
    DMA.writeSourceString("hello");

    //Configuration de l'interruption
    DMA.setInterrupt(true,true,0xFF);
    DMA.ready();

    //Configuration des adresses pour envoyer et écrire les données sur la dram
    DMA.setSourceAddress(MM2S);
    DMA.setDestinationAddress(S2MM);
    DMA.setDestinationLength(5);
    DMA.setSourceLength(9);

    //SOURCE
    do{
        status = DMA.getMM2SStatus();
        DMA.dumpStatus(status);
    }while(!(status & 1 << 1) && !(status & 1 << 12));
    //DEST
    do{
        status = DMA.getS2MMStatus();
        DMA.dumpStatus(status);
    }while(!(status & 1 << 1) && !(status & 1 << 12));

    DMA.hexdumpSource(9);
    DMA.hexdumpDestination(5);
    return 0;
}
```

Buot Adrien
Coville Benoit
Guicharnaud Léo

Lab 3: Hardware / Software Co-Design (Accelerator)

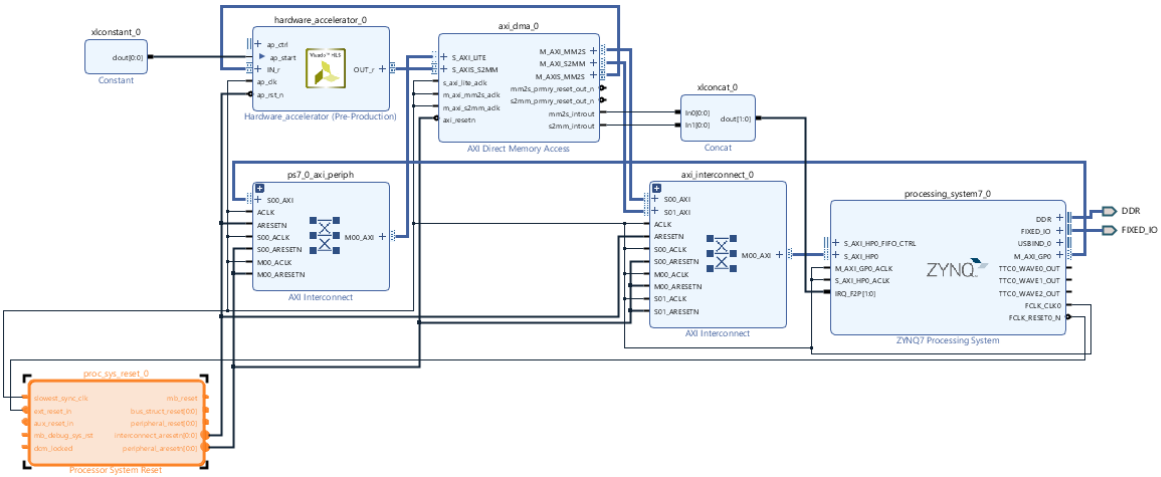
Ce TP nous a permis de prendre en main la programmation du code VHDL. Ainsi que le Co-design hardware et software

Pour cela nous avons utilisé les logiciels VIVADO HLS et VIVADO. Nous devions coder en C++ le code de l'hardware accelerator sur VIVADO HLS. Ensuite nous devions appuyer sur le bouton qui synthétise le code pour le transformer en code VHDL utilisable par la ZedBoard, puis cliquer sur le bouton pour l'exporter sous un format qui sera ensuite utilisable par le logiciel VIVADO.

Ensuite sur le logiciel VIVADO, nous avons ajouté un composant HARDWARE ACCELERATOR qui est capable d'intégrer au design le code VHLD précédemment exporté du logiciel VIVADO HLS. Comme dans le TP1, nous avons ensuite exporté le bitstream qui s'interface avec le nouveau fichier DMA.cpp que nous avons écrit.

Nous avons éprouvé des difficultés pour cette partie. Car lors des tests, nous n'arrivions pas à booter le bon fpga.bit. Après de nombreux tests, nous avons remarqué que la corbeille de la carte (le dossier trash) contenait un fichier fpga.bit et que c'était celui-là qui bootait, nous ne bootions donc pas sur un fichier FPGA.bit qui embarquait notre hardware accelerator. En le supprimant, nous avons réussi à résoudre nos problèmes.

Une fois ce problème résolu, nous avons en effet observé que notre code utilisant le hardware accelerator nous donnait bien un décalage d'un octet (comme indiqué précédemment dans le sujet).



Voilà une capture d'écran du code que nous avons utilisé pour le fichier DMA.cpp

```cpp
#include "DMA.hpp"
#include <unistd.h>
#include <string.h>
#include <iostream>
using namespace std;

#define DMAREGISTER 0x40400000
#define MM2S 0x0e000000
#define S2MM 0x0f000000

int main()
{
    DirectMemoryAccess DMA(DMAREGISTER,MM2S,S2MM);
    unsigned int status=0;

    //On stoppe le DMA
    DMA.reset();
    DMA.halt();

    //Ecriture des 4 bits pour le hardware accelerator

    // DMA.writeSourceByte(0); //0
    // DMA.writeSourceByte(10); //10
    // DMA.writeSourceByte(5); //5
    // DMA.writeSourceByte(0); //0
    // DMA.writeSourceString("hello");
    for(int i = 0; i < 10; i++){
        DMA.writeSourceInteger(i);
    }

    //Configuration de l'interruption
    DMA.setInterrupt(true,true,0xFF);
    DMA.ready();

    //Configuration des adresses pour envoyer et écrire les données sur la dram
    DMA.setSourceAddress(MM2S);
    DMA.setDestinationAddress(S2MM);
    DMA.setDestinationLength(40);
    DMA.setSourceLength(40);

    //SOURCE
    for(int i = 0; i < 10; i++){
        DMA.dumpStatus(i);
    }
    do{
        status = DMA.getMM2SStatus();
        DMA.dumpStatus(status);
    }while(!(status & 1 << 1) && !(status & 1 << 12));
    //DEST
    do{
        status = DMA.getS2MMStatus();
        DMA.dumpStatus(status);
    }while(!(status & 1 << 1) && !(status & 1 << 12));

    DMA.hexdumpSource(40);
    DMA.hexdumpDestination(40);
    return 0;
}
```

PROJET

Pour ce projet, nous avons décidé de faire un hardware accelerator d'image processing. Il prend en entrée 2 images et renvoie en sortie une seule image qui est le mélange des deux entrées.

Pour se faire, nous avons décidé d'envoyer "simultanément" les pixels des images, c'est à dire que nous envoyons le pixel 1 de l'image 1, puis le pixel 1 de l'image 2, ensuite le pixel 2 de l'image 1 et le pixel 2 de l'image 2… Cette méthode nous permet de calculer la valeur moyenne de chaque pixel et d'utiliser au mieux le hardware. En effet, la lecture de data en entrée se fait en séquentiel, une fois l'entrée lue, on ne peut plus y avoir accès, il est donc préférable de calculer les pixels 1 à 1.

Au début du programme, nous récupérons la valeur d'entrée afin qu'elle ne soit pas perdue au fil de nos calculs.

```
input = IN[i];
```

Nous faisons la moyenne de 2 pixels, le calcul est donc simple : $\text{sortie} = \frac{P_{image1}+P_{image2}}{2}$,mais il faut aussi prendre en compte que nous recevons des données sur 32bits soit 4 octets, chaque pixel étant composé d'un seul octet, nous en recevons donc 4 à la fois, et il faut procéder à un masque, ainsi qu'un décalage, pour obtenir des pixels sur 8bits (ci-dessous, le code utilisé pour y arriver)

```
val1 = (input.data & 0xFF000000) >> 24;
val2 = (input.data & 0xFF0000) >> 16;
val3 = (input.data & 0xFF00) >> 8;
val4 = input.data & 0xFF;
```

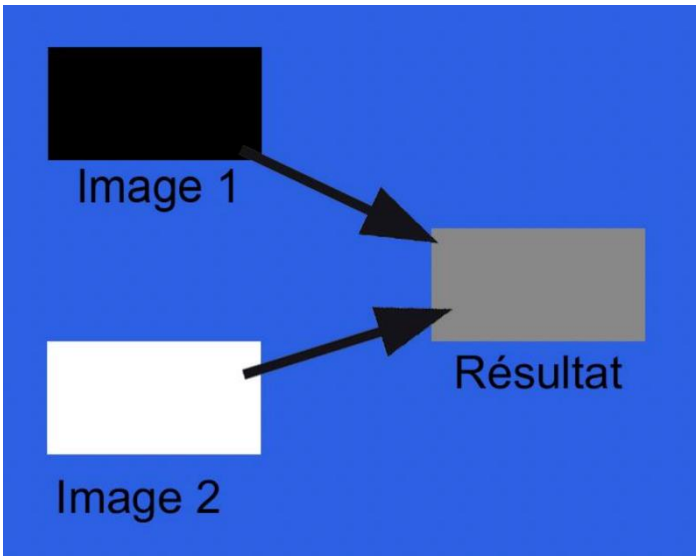
Ensuite, nous faisons le calcul de nos pixels de sortie.

```
value1 = ( val1 + val2 )/2;
value2 = ( val3 + val4 )/2;
```

Nous recevons 4 pixels, et ne pouvons en envoyer que 2 au lieu de 4, c'est pourquoi, nous stockons la valeur des deux premiers pixels, et au tour de boucle suivant, nous recevons 2 pixels de plus, nous pouvons alors tout envoyer. Nous faisons bien entendu des décalages afin de positionner chaque octet correctement.

```
if(i % 2 == 0){
    //We set the first 2 new pixels calculated
    value = (ap_uint<32>) ( value1 << 8 | value2 );
//If we calculate the 2 last pixels of the line, we send it to the output
}else{
    // Add a value to the initial output.data and send it back to the DMA
    // We send 4 new pixels
    output.data = (ap_uint<32>) (value << 16 | value1 << 8 | value2 );
    // Always copy keep signal from input so you
    // do not have to manage it
    output.keep = input.keep;
    // Always copy last signal from input so you
    // do not have to manage it
    output.last = input.last;
    //We send the output
    OUT[j] = output;
    j++;
}
```

Ainsi, avec ce programme accelerator, nous pouvons fusionner deux images en une seule, comme le montre l'image ci-dessous



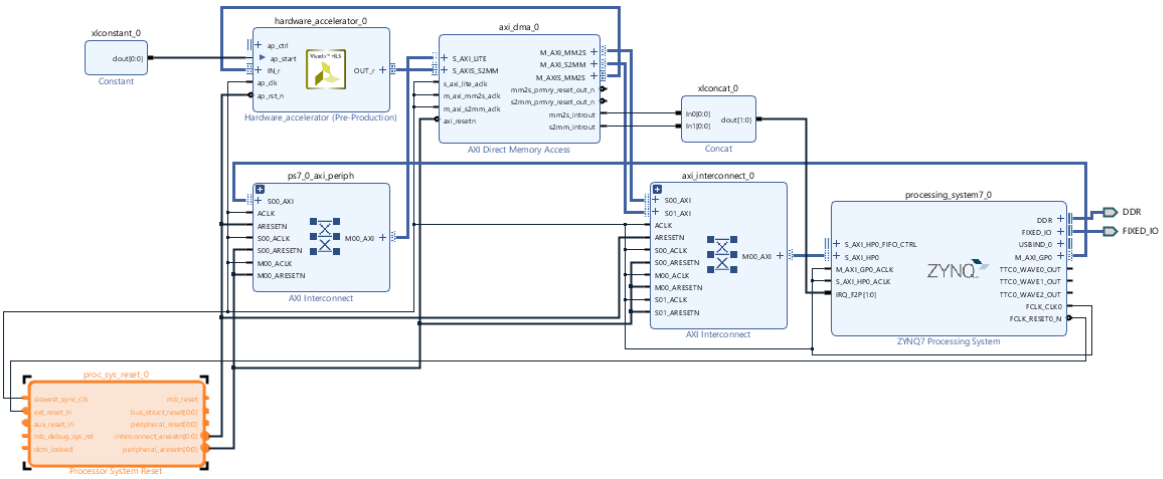
Et nous obtenons en résultat :

```
hello_world_dma  lost+found       prog
# ./prog
Status: running idle IOC_Irq
Status: running idle IOC_Irq
0000: fe 00 fe 00 fe 00 fe 00 fe 00 fe 00 fe 00 fe 00  ................
0010: fe 00 fe 00 fe 00 fe 00 fe 00 fe 00 fe 00 fe 00  ................
0000: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f  ................
#
```

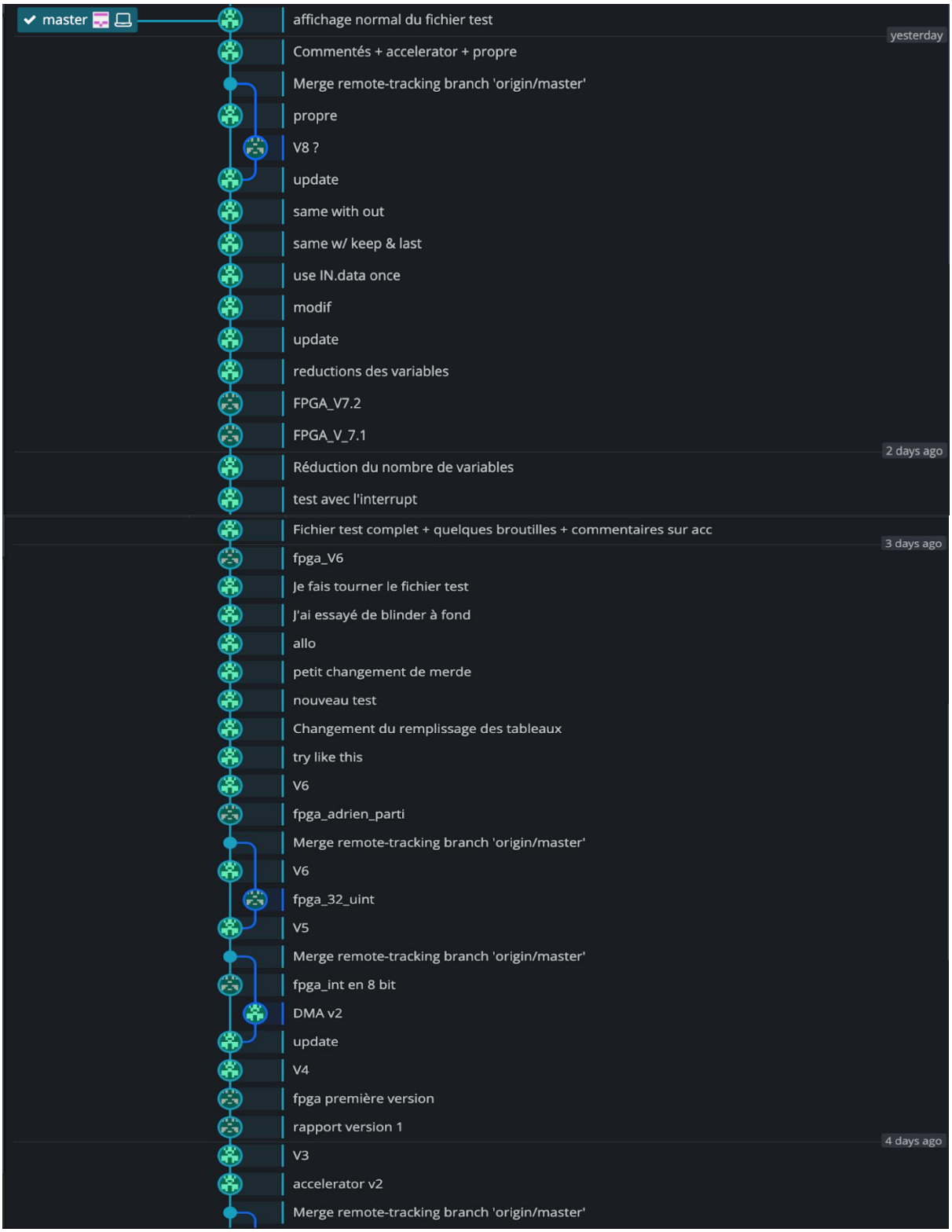
Nous avons ensuite modifié les images d'entrées pour que l'image de sortie soit moins régulière et nous obtenons comme résultat :

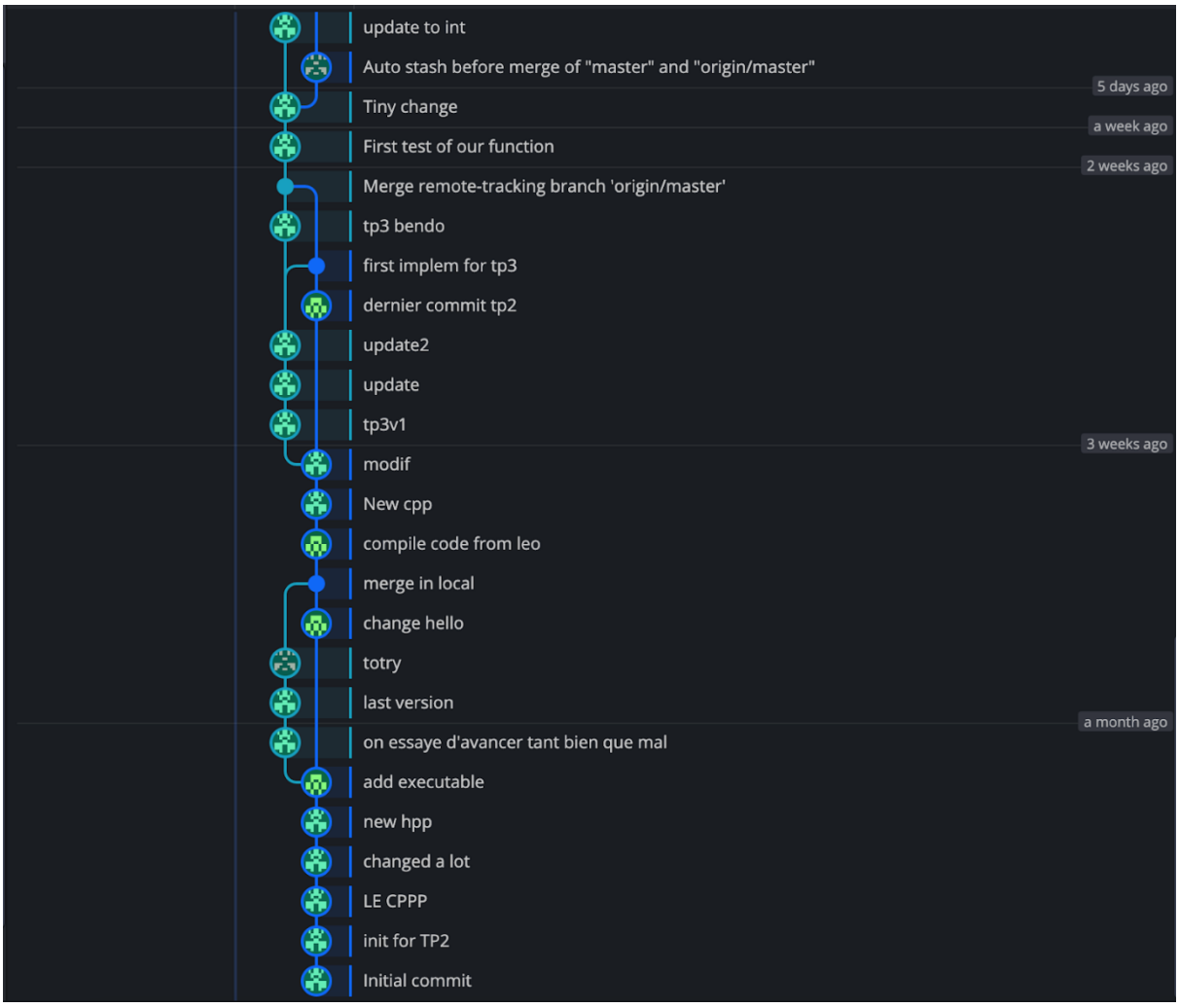
```
zynq login: root
Password:
# ls
hello_world_dma  lost+found       prog
# ./prog
Status: running idle IOC_Irq
Status: running idle IOC_Irq
0000: 00 ff 00 ff 00 ff 00 ff 00 ff 00 ff 00 ff 00 ff  ................
0010: 0a ff 0a ff 0a ff 0a ff 0a ff 0a ff 0a ff 0a ff  ................
0000: 7f 7f 7f 7f 7f 7f 7f 7f 84 84 84 84 84 84 84 84  ................
#
```

Voilà une capture d'écran du design hardware que nous avons utilisé



Voilà une capture d'écran du git que nous avons utilisé pendant et labs et le projet : on peut voir que nous avons éprouvé des difficultées pour générer un fichier fpga.bit correct. Cela est due au fait que nous lisions les données du ficher d'entrée 2 par deux. Etant donnée que le tableau d'entrée est de type FIFO, nous avons longtemps eu des problèmes lors des test avent de comprendre d'où venait l'erreur.





Pour tester la compatibilité et le bon fonctionnement de l'interaction entre l'accélérateur et le reste du design, nous avons écrit un fichier test découpé en 2 programmes : un main qui simule l'environnement général et un sous-programme appelé 'accelerator' qui simule l'accélérateur. Voilà une capture d'écran de ce fichier de test :

```c
#include <stdlib.h>
#include <stdio.h>
typedef struct ap_axi {
    uint32_t data; // Represents data you wrote on the DRAM
    uint8_t keep; // Used when words are longer than 32 bits
    uint8_t last;  // Assert the last data of the burst
} ap_axi;

void accelerator(ap_axi IN[8], ap_axi OUT[4]);
int main(){
    unsigned int blackImg[4][4] = {0x0,0x0,0x0,0x0,
                                   0x0,0x0,0x0,0x0,
                                   0x0,0x0,0x0,0x0,
                                   0x0,0x0,0x0,0x0};
    unsigned int whiteImg[4][4] = {0xFF,0xFF,0xFF,0xFF,
                                   0xFF,0xFF,0xFF,0xFF,
                                   0xFF,0xFF,0xFF,0xFF,
                                   0xFF,0xFF,0xFF,0xFF};
    unsigned int value, result;
    ap_axi IN[8], OUT[4];
    int k = 0;
    for(unsigned int i = 0; i < 4; i++)
        for(unsigned int j = 1; j < 4; j+=2){
            IN[k].data = (blackImg[i][j-1] << 24 | whiteImg[i][j-1] << 16 | blac
            k++;
        }
    accelerator(IN, OUT);
    for(int i = 0; i < 4; i++)
        printf("0x%x\n", OUT[i].data);
}

void accelerator(ap_axi IN[8], ap_axi OUT[4]){
    uint32_t value, value1, value2;
    uint8_t val1, val2, val3, val4;
    int j = 0;
    for(int i = 0; i < 8; i++){
        val1 = (IN[i].data & 0xFF000000) >> 24;
        val2 = (IN[i].data & 0xFF0000) >> 16;
        val3 = (IN[i].data & 0xFF00) >> 8;
        val4 = IN[i].data & 0xFF;
        value1 = ( val1 + val2 )/2;
        value2 = ( val3 + val4 )/2;
        if(i % 2 == 0){
            value = (uint32_t) ( value1 << 8 | value2 );
        }else{
            OUT[j].data = (uint32_t) (value << 16 | value1 << 8 | value2 );
            j++;
        }
    }
}
```

Ressources Informatiques

Lien vers le git de notre groupe : https://github.com/Bubu781/ArchitectureEmarquee

Informations sur le code de césar : https://www.dcode.fr/chiffre-cesar
Ressources pour comprendre le problème de pile de l'entrée https://forums.xilinx.com/t5/High-Level-Synthesis-HLS/Improper-streaming-accesses-error-with-AXI-DMA-stream/td-p/956147