Formation HLS **Lecture Image sous Vivado HLS.**

Dans ce TP, on va afficher et modifier une image stockée en BLOCRAM à l'aide d'une fonction HLS :

PARTIE 1

Dézipper le fichier tp2_p1.zip

Q1 : Ouvrir le projet im_load sous Vivado HLS (sous répertoire HLS) et le projet tp2_p1 sous Vivado

Q2 : Lancer la simulation C (***run C simulation***) et vérifier que le fichier image pgm généré est bien la mème image que 480.pgm

Q3 : Ajouter le paramètre `ap_uint<2> mode` à la fonction ***im_load*** et modifier le code de la fonction pour avoir le fonctionnement suivant :

mode = 0 : image normale
mode = 1 : symétrie en x (miroir)
mode = 2 : Symétrie en y
mode = 3 : Symétries en x et y

Vérifier le résultat en simulation C et faire un packaging de l'IP im_load

Q4 : Mettre à jour l'IP im_load sur le schéma du projet tp2_p1 sous Vivado. Ajouter deux switchs SW1 et SW2 comme vu au tp1. Compléter le schéma, faire la synthèse, l'implantation et tester sur la zedboard.

PARTIE 2

On va maintenant ajouter une IP pour incruster une image dans le flux vidéo.
Cette IP comportera une entrée et une sortie Stream.

Q5 : Créer cette IP sous vivado avec les fichiers suivants :

fichier source CPP incrust.cpp

```
#include"ap_int.h"
#include "incrust_s.h"
#include "hls_video.h"
#define size_x 100
#define size_y 67

#define start_x 300
#define start_y 100
#define hsize_in 640
#define vsize_in 480

typedef hls::stream<ap_axiu<8,1,1,1> >   AXI_STREAM;
void incrust(AXI_STREAM& s_axis_video,AXI_STREAM& m_axis_video)
{
#pragma HLS INTERFACE axis register both port=s_axis_video
#pragma HLS INTERFACE axis register both port=m_axis_video

        ap_axiu<8, 1, 1, 1> video;

        for(int i = 0; i < vsize_in ; i ++)
        {
                for(int j = 0; j < hsize_in ; j ++)
                {
                s_axis_video >> video;

                m_axis_video << video;

                }
        }

}
```

fichier source testbench CPP incrust_tb.cpp

```
#include"ap_int.h"
#include "hls_video.h"
#include <hls_opencv.h>

typedef hls::stream<ap_axiu<8,1,1,1> >         AXI_STREAM;
void incrust(AXI_STREAM& s_axis_video,AXI_STREAM& m_axis_video);
int main (int argc, char** argv) {

        // Load data in OpenCV image format
        IplImage* src = cvLoadImage("..\\..\\..\\480.pgm",CV_LOAD_IMAGE_GRAYSCALE);

        //Get input Image size
        CvSize size_in;
        size_in = cvGetSize(src);
        //Set output image size
        CvSize size_out;
        size_out.width = size_in.width;
        size_out.height = size_in.height;
        //Create Destination image
        IplImage* dst = cvCreateImage(size_out, src->depth, 1);
        //Create the AXI4-Stream
        AXI_STREAM src_axi, dst_axi;

        // Convert OpenCV format to AXI4 Stream format
        IplImage2AXIvideo(src, src_axi);


        // Call the function to be synthesized
        incrust(src_axi, dst_axi);

        // Convert the AXI4 Stream data to OpenCV format
        AXIvideo2IplImage(dst_axi, dst);

        // Standard OpenCV image functions
        cvSaveImage("..\\..\\..\\out.png", dst);

        cvReleaseImage(&src);
        cvReleaseImage(&dst);
        return 0;
}
```

Q6: Le fichier incrust_s.h contient les données d'une imagette de 100x67 pixels.
Modifier le source de l'IP pour insérer l'imagette aux coordonnées ***start_x*** et ***start_y***.
Tester en simulation C

Q7 : Exporter l'IP et l'insérer dans le flux vidéo dans le projet tp2_p1 testé en Q4

PARTIE 3

On se propose ensuite de pouvoir modifier pendant le fonctionnement la position d'affichage de l'imagette vue dans la partie 2.

Q8 : Reprendre le projet HLS
Modifier la fonction incrust en ajoutant start_x et start_y

```
void incrust(AXI_STREAM& s_axis_video,AXI_STREAM& m_axis_video,int start_x,int start_y)
```

Ajouter les directives suivantes :

```
#pragma HLS INTERFACE s_axilite  port=start_x
#pragma HLS INTERFACE s_axilite  port=start_y
```

faire quelques tests en simulation.

Q9 : Faire une synthèse et observer le top du module vhdl créé (solution1 → impl → vhdl → incrust.vhd)

Q10 : ouvrir le fichier /incrust/solution1/impl/misc/drivers/incrust_v1_0/src/xincrust_hw.h et relever les adresses (offsets) de start_x et start_y

Q11 : exporter l'IP et mettre à jour celle-ci dans le projet Vivado .
En l'absence d'un processeur dans le design, on va utiliser l'IP fournie dans vivado ***jtag_axi***
Effectuer les connexions et paramétrer l'IP jatg_axi avec un type axilite

Q12 : faire la synthèse et tester sur la zedboard. Pour piloter l'axi_jtag, on effectuera les commandes suivantes :

démarrer l'outil XSCT (c:\xilinx\vitis\2019.2\bin\xsct.bat)
connect
targets 6

Pour lire une adresse : **mrd 0xaaaaaaaa**
Pour écrire à une adresse : **mwr 0xaaaaaaaa 0xddddddddd**

PARTIE 4

Dézipper le fichier tp2_p2.zip. Charger le projet tp2_p2

Q13 : Etudier le schéma et noter les différences dans la façon de gérer le bloc RAM.

Q14 : Intégrer l'IP incrust vue dans la partie 3. Faire la synthèse et tester. Noter les différentes adresses présentes dans le design.

Dans cette configuration de la RAM, on ne peut pas inclure un fichier de type .coe au moment de la synthèse. Pour charger la RAM, il faut utiliser la console TCL. On a trois fichiers (mountain.tcl, 480.tcl et lune.tcl) disponible pour charger la RAM.

Pour augmenter la taille de la mémoire disponible, on va intégrer la mémoire DDR intégrée sur la carte. Pour cela, il est nécessaire d'intégrer l'IP du processeur ZYNQ car il contient le contrôleur de la DDR

Q15 : Placer l'IP Zynq dans le design. Il est nécessaire de configurer l'IP :

- presets zedboard
- PS/PL configuration → HP slave AXI interface HP0

On raccordera HP0 avec l'AXI smart interconnect.

Q16 : Ouvrez l'IP im_load_mm et modifier l'offset de la directive :

#pragma HLS INTERFACE m_axi port=mem_ddr offset=slave

Compléter le schéma et synthétiser et constater l'incidence sur le module VHDL

Pour que le design fonctionne, il faut faire les actions suivantes :

- file → export → export hardware (avec bitstream)
- Dezipper le fichier .xsa généré dans un répertoire
- démarrer xsct (c:\xilinx\vitis\2019.2\bin\xsct.bat)
- se positionner sur le répertoire précédent
- targets -set -nocase -filter {name =~ "ARM Cortex-A9 MPCore #0"}
- source ps7_init.tcl
- ps7_init
- ps7_post_config
- fpga design_2_wrapper.bit

On pourra utiliser les commande ***mrd*** et ***mwr*** vues précédemment

Pour charger des images, on pourra utiliser la commande :

dow -data fic.bin add

On chargera plusieurs images en mémoire (en ddr à partir de 0x10000000) et on basculera l'affichage en modifiant l'adresse du GPIO.