TP1

Décrire les différents signaux de l’entité mire dans le fichier VHDL (dans impl)

Un signal pour le contrôle de start stop et un signal pour valide que chaque couleur est traitée et un autre pour valide qu’elle est bien arrivée.

Comment est démarré le module mire, quel est son temps d’exécution ?

Il prend 615361 clock time

Pour recup address : C:\Users\ap576391\Documents\Brost\tp2\_p1\hls\incrust\system\impl\misc\drivers\incrust\_v1\_0\sr

0x10

0x18

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 0xdddddddd

POUR JTAG POUR MODIFIER DES VALEURS :

**#include**"ap\_int.h"

**#include** "incrust\_s.h"

**#include** "hls\_video.h"

**#define** size\_x 148

**#define** size\_y 69

**#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,**int** start\_x,**int** start\_y)

{

**#pragma** HLS INTERFACE axis **register** both port=s\_axis\_video

**#pragma** HLS INTERFACE axis **register** both port=m\_axis\_video

**#pragma** HLS INTERFACE s\_axilite port=start\_x

**#pragma** HLS INTERFACE s\_axilite port=start\_y

ap\_axiu<8, 1, 1, 1> video;

**int** pixel = 0;

**for**(**int** i = 0; i < vsize\_in ; i ++)

{

**for**(**int** j = 0; j < hsize\_in ; j ++)

{

s\_axis\_video >> video;

**if**(((j >= start\_x) && (j < (start\_x + size\_x))) && ((i >= start\_y) && (i < (start\_y + size\_y))) ){

video.data = im\_incrust[pixel];

pixel++;

}

m\_axis\_video << video;

}

}

}

-----------------------------------------------------SUJET NUMEROS 2 -------------------------------------------------------

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 :

Formation HLS Lecture Image sous Vivado HLS.

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

Formation HLS Lecture Image sous Vivado HLS.

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 0xdddddddd

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.

Formation HLS Lecture Image sous Vivado HLS.

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 :

Formation HLS Lecture Image sous Vivado HLS.

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.