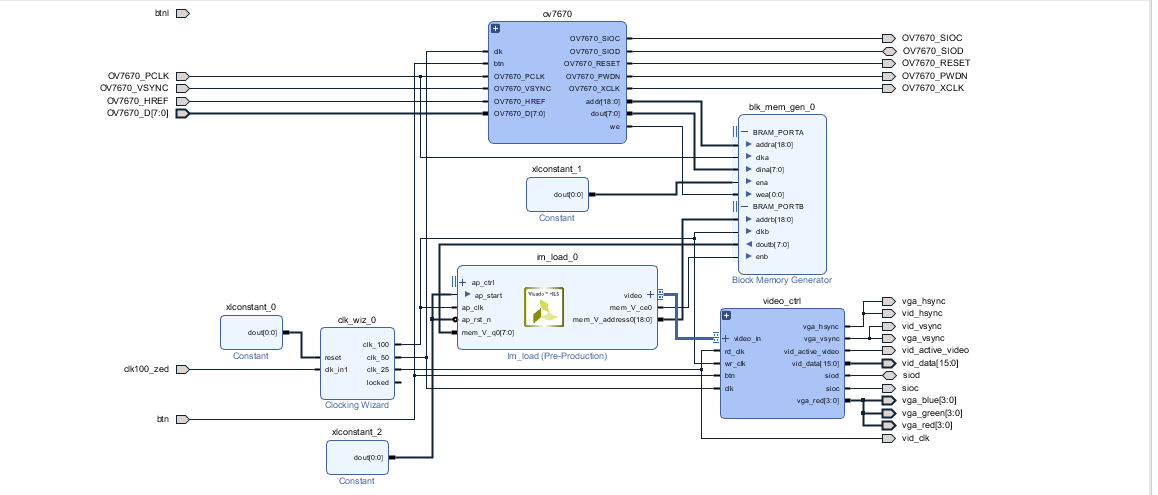
**Chaîne de traitement Vidéo sous Vivado HLS.**

**Q2 : Observer le schéma et décrire le rôle des différents modules. A quoi sert le bloc de mémoire RAM**

Réponse : En ce qui concerne le bloc de mémoire RAM, son rôle est de stocker des données qui peuvent être rapidement écrites et lues par le système. Dans le contexte d'un système de traitement d'image ou de vidéo, il est souvent utilisé pour stocker des images ou des cadres vidéo en cours de traitement ou en attente d'être affichés. La RAM à blocs est particulièrement utile dans les applications FPGA en raison de sa vitesse élevée et de son accès aléatoire aux données

Lien du code hls créer : C:/Users/re623573/AppData/Roaming/Xilinx/Vivado/c\_grav

**Q3 : Créer cette IP sous vivado avec les fichiers suivants** :

Remarque : pour afficher la première valeur il suffit de remplacer le nb = 0, par nb =1



Premier resultat de image au dessus :

Code du proff :

C\_grave.cpp

#include"ap\_int.h"

#include "hls\_video.h"

volatile long cgx\_r=320;

volatile long cgy\_r=240;

typedef hls::stream<ap\_axiu<8,1,1,1> > AXI\_STREAM;

void c\_grav( AXI\_STREAM& s\_axis\_video,AXI\_STREAM& m\_axis\_video, int hsize\_in, int vsize\_in) {

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

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

unsigned nb=0 ; // il faut remplacer le 0 par 1

long cgx=0;

long cgy=0;

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

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

#pragma HLS PIPELINE

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

s\_axis\_video >> video; m\_axis\_video << video;

}

}

cgx\_r=cgx/nb;

cgy\_r=cgy/nb;

}

fichier source testbench CPP c\_grav\_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** **c\_grav**(AXI\_STREAM& s\_axis\_video,AXI\_STREAM& m\_axis\_video, **int** hsize\_in, **int** vsize\_in);

**int** **main** (**int** argc, **char**\*\* argv) {

// Load data in OpenCV image format

IplImage\* src = **cvLoadImage**("..\\..\\..\\triangle.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,dst\_axi2;

// Convert OpenCV format to AXI4 Stream format

IplImage2AXIvideo(src, src\_axi);

// Call the function to be synthesized

c\_grav(src\_axi, dst\_axi,size\_in.width,size\_in.height);

IplImage2AXIvideo(src, src\_axi);

c\_grav(src\_axi, dst\_axi2,size\_in.width,size\_in.height);

// Convert the AXI4 Stream data to OpenCV format

AXIvideo2IplImage(dst\_axi2, dst);

// Standard OpenCV image functions

**cvSaveImage**("..\\..\\..\\out.png", dst);

**cvReleaseImage**(&src);

**cvReleaseImage**(&dst);

**return** 0;

}

**Q3 : Modifier le programme de l’IP pour calculer le CG de l’image. On affichera un carré blanc (10x10) à la position calculée. Tester en simulation.**

Pour cette partie tout les modification sont faite dans le source et pas dans le test bench les test bench on le touche pas et voici le code et dans la partie ci-dessous :

Code Modifiier :

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

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

**volatile** **long** cgx\_r = 320;

**volatile** **long** cgy\_r = 240;

**typedef** hls::stream<ap\_axiu<8,1,1,1> > AXI\_STREAM;

**void** **c\_grav**(AXI\_STREAM& s\_axis\_video, AXI\_STREAM& m\_axis\_video, **int** hsize\_in, **int** vsize\_in)

{

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

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

**unsigned** nb = 0;

**long** cgx = 0;

**long** cgy = 0;

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

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

**#pragma** HLS PIPELINE

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

s\_axis\_video >> video;

// Vérification si le pixel actuel est dans la zone du carré blanc

**if** ((i >= cgy\_r - 5) && (i < cgy\_r + 5) && (j >= cgx\_r - 5) && (j < cgx\_r + 5)) {

video.data = 255; // Pixel blanc

}

**if** (video.data< 125) {

cgx += j;

cgy += i;

nb++;

}

m\_axis\_video << video;

}

}

// Mise à jour des coordonnées du CG

**if** (nb > 0) {

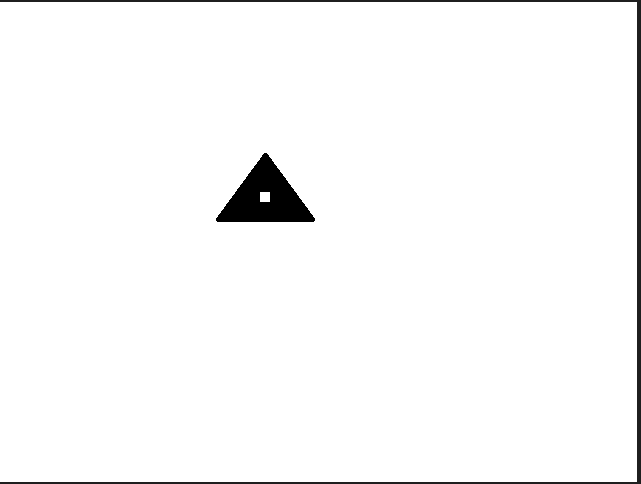
cgx\_r = cgx / nb;

cgy\_r = cgy / nb;

}

}

Teste de la simulations :



Schema :

