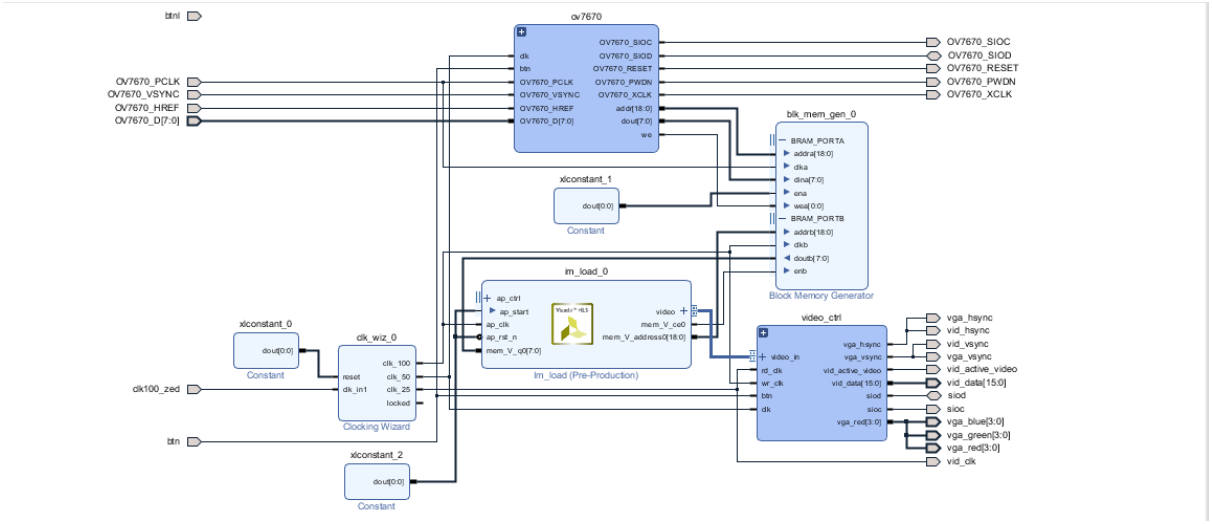
Chaîne de traitement Vidéo sous Vivado HLS.

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

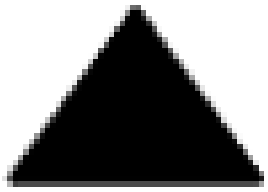


Réponse : En ce qui concerne le bloc de mémoire RAM, son rôle est de stocker des données qui peuvent être rapidement écrites et lues par le système. Dans le contexte d'un système de traitement d'image ou de vidéo, il est souvent utilisé pour stocker des images ou des cadres vidéo en cours de traitement 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

```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;
}
```

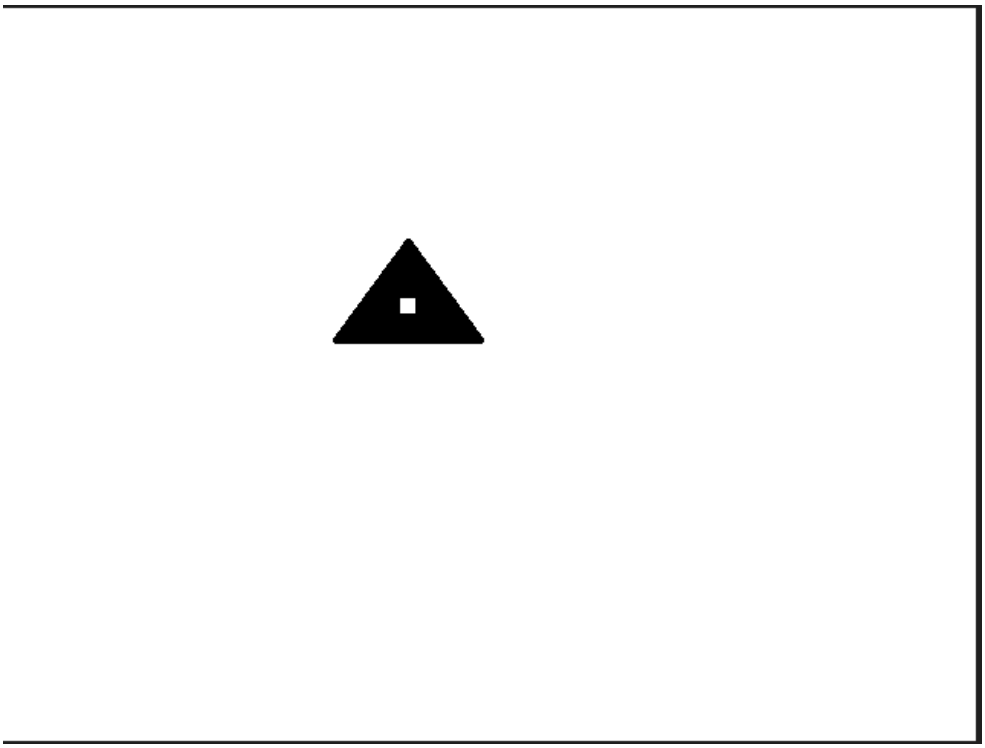
Q3 : Modifier le programme de l'IP pour calculer le CG de l'image. On affichera un carré blanc (10x10) à la position calculée. Tester en simulation.

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 :

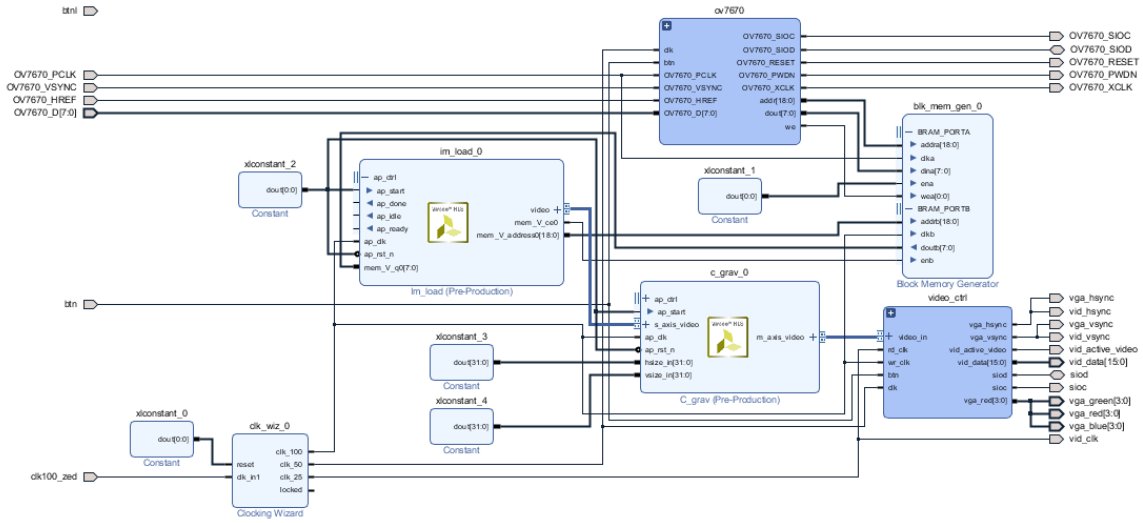
Code Modifiier :

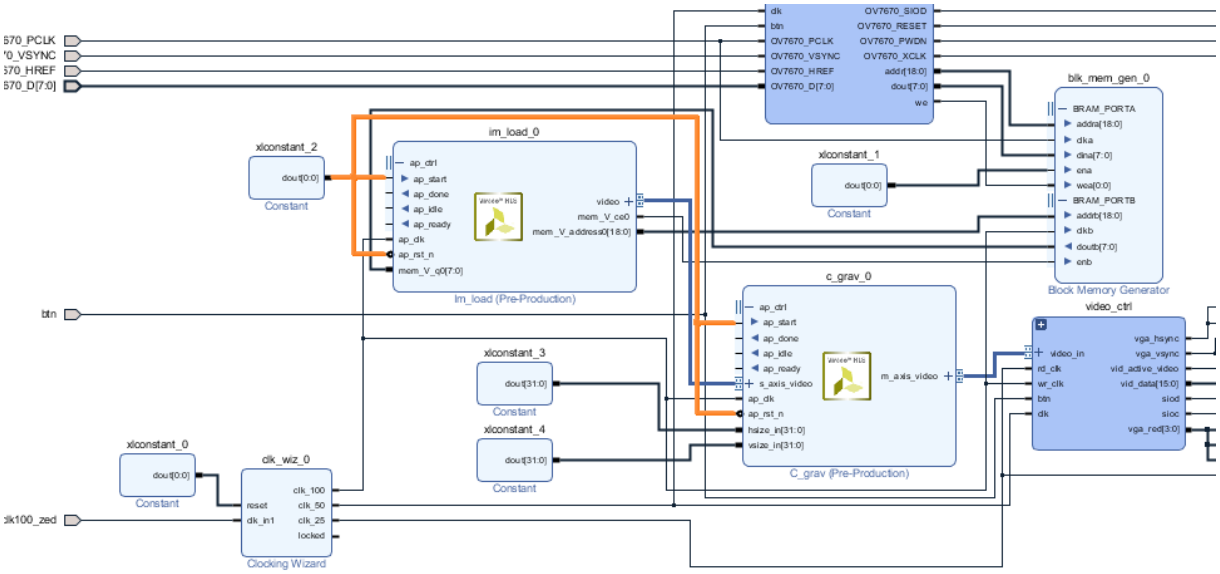
```
#include "ap_int.h"
#include "hls_video.h"

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;
    }
}
```

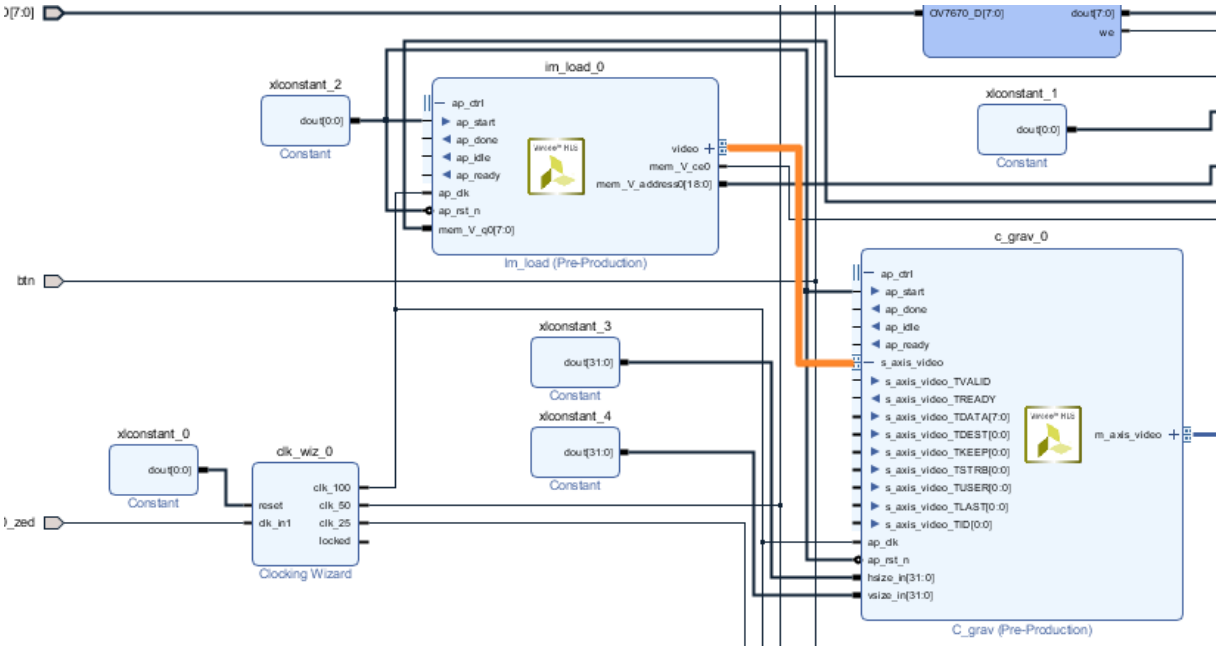
Teste de la simulations :

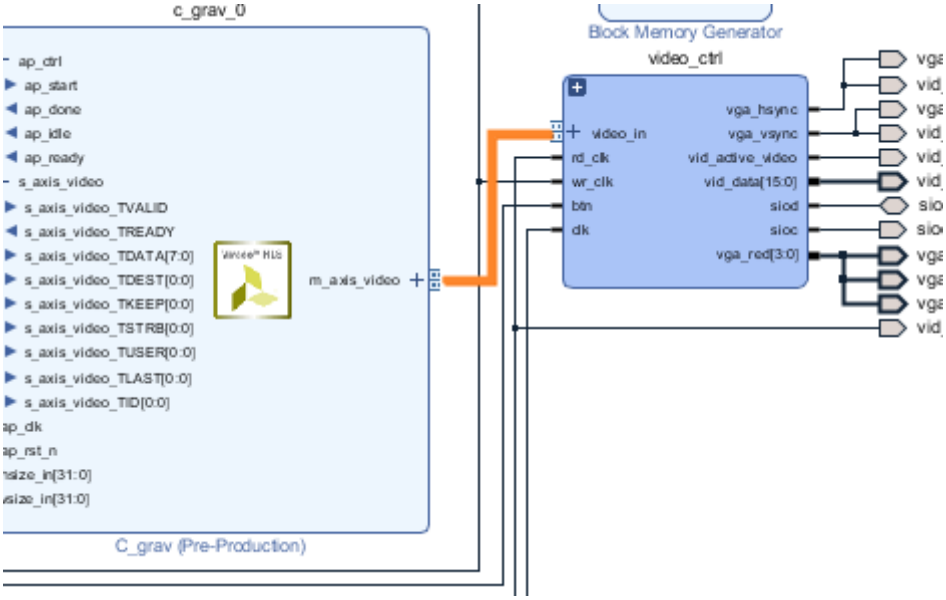


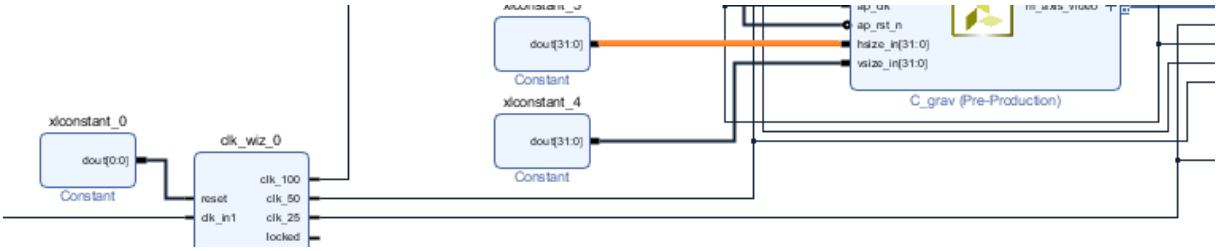
Schema :

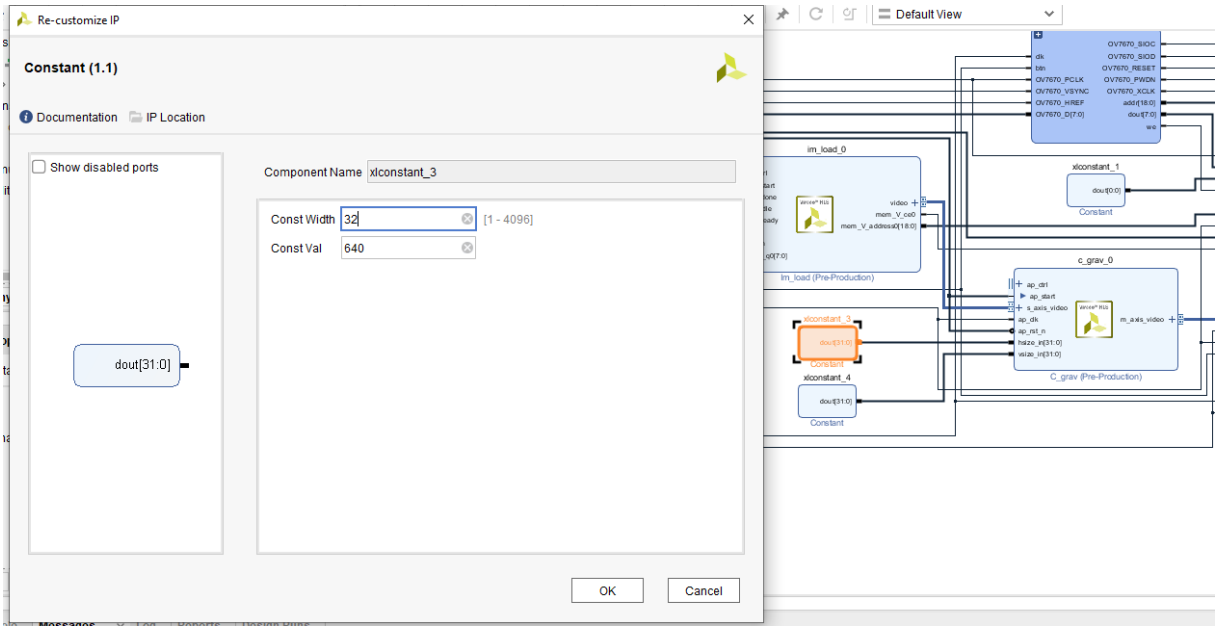


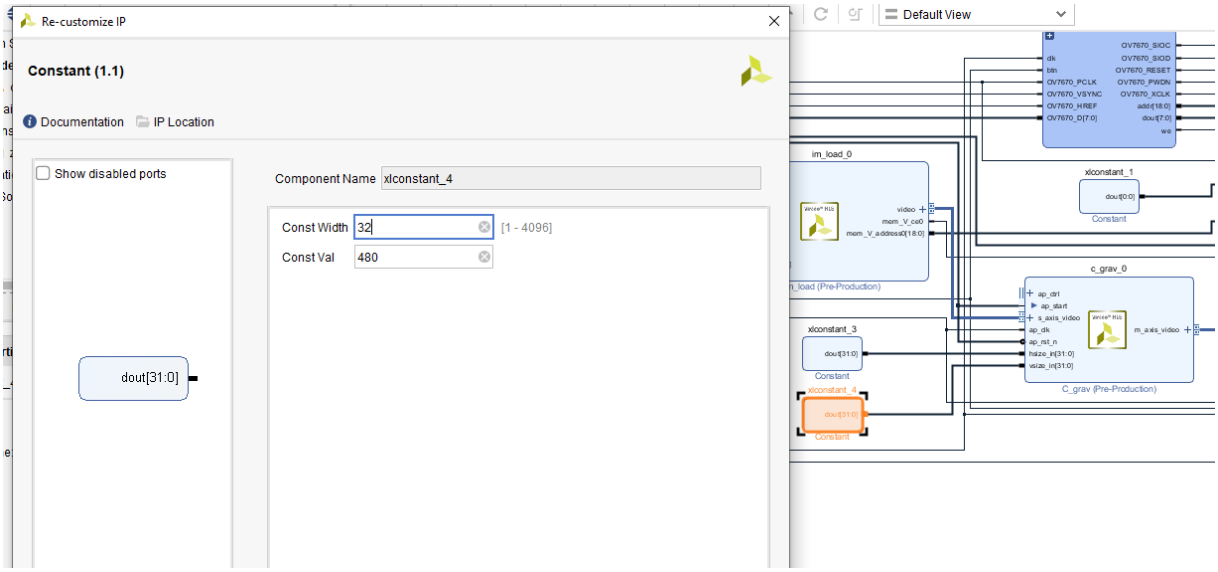

570_PCLK
'0_VSYNC
570_HREF
570_D[7:0]
clk
btn
OV7670_PCLK
OV7670_VSYNC
OV7670_HREF
OV7670_D[7:0]
OV7670_SIOD
OV7670_RESET
OV7670_PWDN
OV7670_XCLK
addr[18:0]
dout[7:0]
we
blk_mem_gen_0
BRAM_PORTA
addra[18:0]
dina[7:0]
ena
wea[0:0]
BRAM_PORTB
addrb[18:0]
dinb
doutb[7:0]
enb
Block Memory Generator
xlconstant_2
dout[0:0]
Constant
im_load_0
ap_ctrl
ap_start
ap_done
ap_idle
ap_ready
ap_clk
ap_rst_n
mem_V_q0[7:0]
video
mem_V_ce0
mem_V_address0[18:0]
Im_load (Pre-Production)
xlconstant_1
dout[0:0]
Constant
c_grav_0
s_axis_video
m_axis_video
hsize_in[31:0]
vsize_in[31:0]
C_grav (Pre-Production)
video_ctrl
video_in
rd_clk
wr_clk
btn
clk
vga_hsync
vga_vsync
vid_active_video
vid_data[15:0]
siod
sioc
vga_red[3:0]
btn
xlconstant_3
dout[31:0]
Constant
xlconstant_4
dout[31:0]
Constant
xlconstant_0
dout[0:0]
Constant
clk_wiz_0
clk_100
reset
clk_50
clk_in1
clk_25
locked
Clocking Wizard
clk100_zed


)[7:0]
OV7670_D[7:0]
dout[7:0]
we
im_load_0
xlconstant_2
dout[0:0]
Constant
ap_ctrl
ap_start
ap_done
ap_idle
ap_ready
ap_clk
ap_rst_n
mem_V_q0[7:0]
video
mem_V_ce0
mem_V_address0[18:0]
Im_load (Pre-Production)
xlconstant_1
dout[0:0]
Constant
c_grav_0
btn
xlconstant_3
dout[31:0]
Constant
xlconstant_4
dout[31:0]
Constant
s_axis_video
s_axis_video_TVALID
s_axis_video_TREADY
s_axis_video_TDATA[7:0]
s_axis_video_TDEST[0:0]
s_axis_video_TKEEP[0:0]
s_axis_video_TSTRB[0:0]
s_axis_video_TUSER[0:0]
s_axis_video_TLAST[0:0]
s_axis_video_TID[0:0]
m_axis_video
ap_clk
ap_rst_n
hsize_in[31:0]
vsize_in[31:0]
C_grav (Pre-Production)
xlconstant_0
dout[0:0]
Constant
clk_wiz_0
clk_100
reset
clk_50
clk_in1
clk_25
locked
Clocking Wizard
)_zed


c_grav_0
ap_ctrl
ap_start
ap_done
ap_idle
ap_ready
s_axis_video
s_axis_video_TVALID
s_axis_video_TREADY
s_axis_video_TDATA[7:0]
s_axis_video_TDEST[0:0]
s_axis_video_TKEEP[0:0]
s_axis_video_TSTRB[0:0]
s_axis_video_TUSER[0:0]
s_axis_video_TLAST[0:0]
s_axis_video_TID[0:0]
ap_clk
ap_rst_n
vsize_in[31:0]
hsize_in[31:0]
m_axis_video
C_grav (Pre-Production)
Block Memory Generator
video_ctrl
video_in
rd_clk
wr_clk
btn
clk
vga_hsync
vga_vsync
vid_active_video
vid_data[15:0]
siod
sioc
vga_red[3:0]


xlconstant_0
dout[0:0]
Constant
clk_wiz_0
reset
clk_in1
clk_100
clk_50
clk_25
locked
dout[31:0]
Constant
xlconstant_4
dout[31:0]
Constant
ap_rst_n
hsize_in[31:0]
vsize_in[31:0]
C_grav (Pre-Production)


Re-customize IP
Constant (1.1)
Documentation
IP Location
Show disabled ports
dout[31:0]
Component Name
xlconstant_3
Const Width
32
[1 - 4096]
Const Val
640
OK
Cancel
Default View
im_load_0
im_load (Pre-Production)
xlconstant_1
Constant
c_grav_0
C_grav (Pre-Production)
xlconstant_3
xlconstant_4


Re-customize IP
Constant (1.1)
Documentation
IP Location
Show disabled ports
dout[31:0]
Component Name
xlconstant_4
Const Width
32
[1 - 4096]
Const Val
480
Default View
OV7670_SIOC
OV7670_SIOD
OV7670_RESET
OV7670_PWDN
OV7670_XCLK
OV7670_VSYNC
OV7670_HREF
OV7670_PCLK
OV7670_D[7:0]
im_load_0
xlconstant_1
Constant
c_grav_0
C_grav (Pre-Production)
xlconstant_3
xlconstant_4