Contents:

➢ *Introduction*

- The project aims to implement Edge Detection for an image using Sobel Operator on a Field Programmable Gate Array(FPGA) device.
- The Edge Detection is an Image Processing Technique for finding the boundaries of objects within image.
- It works by detecting discontinuities in brightness.
- Edge Information for a particular pixel is obtained by exploring the brightness of pixels in the neighbourhood of that pixel.
- Edge Detection is used for Image Segmentation and Data Extraction in areas such as Image Processing, Computer Vision and Machine Vision.

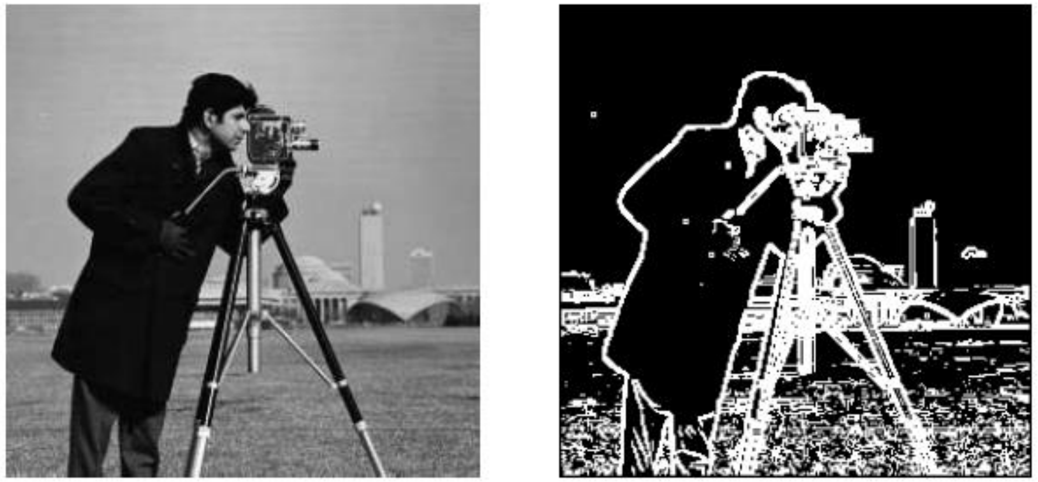


Fig. Edge Detection

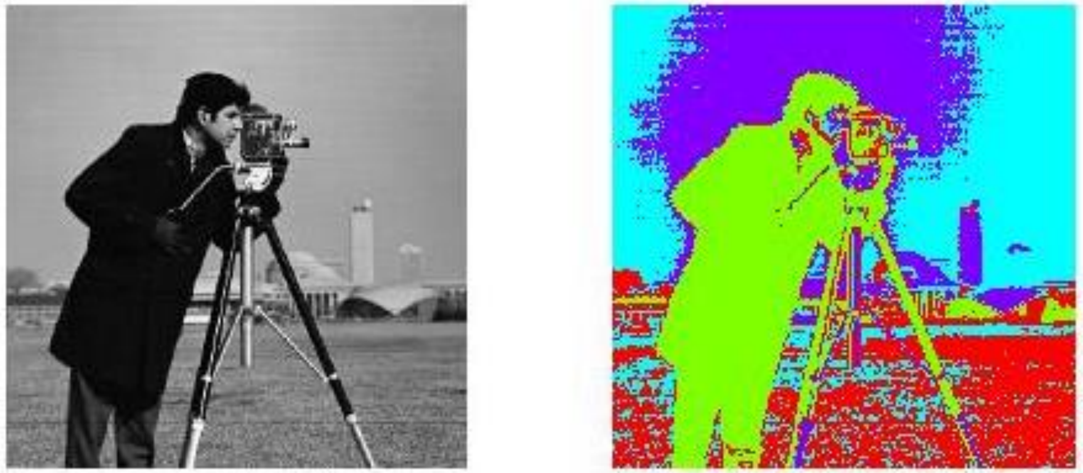


Fig. Image Segmentation

➢ Edge Detection Theory

- An edge can be defined as an abrupt change in brightness as we move from one pixel to its neighbour. In digital image processing, each image is quantized into pixels.
- With grayscale images, each pixel indicates the level of brightness of image in a particular spot: 0->black , 255->white with 8-bit pixels.
- If, for a particular pixel, all the pixels in neighbourhood have almost same brightness, probably there is no edge at that point.
- Detecting edges is an important task in boundary detection, motion estimation, texture analysis and object identification.

➢ Sobel Edge Detection Operator

- Sobel Operator, as it is commonly known, is used particularly within edge detection algorithms where it creates an image emphasising edges.
- It is named after Irwin Sobel.
- It uses two 3×3 kernels(or masks) which are convolved with the original image to calculate approximations of its derivative-one for horizontal and one for vertical.

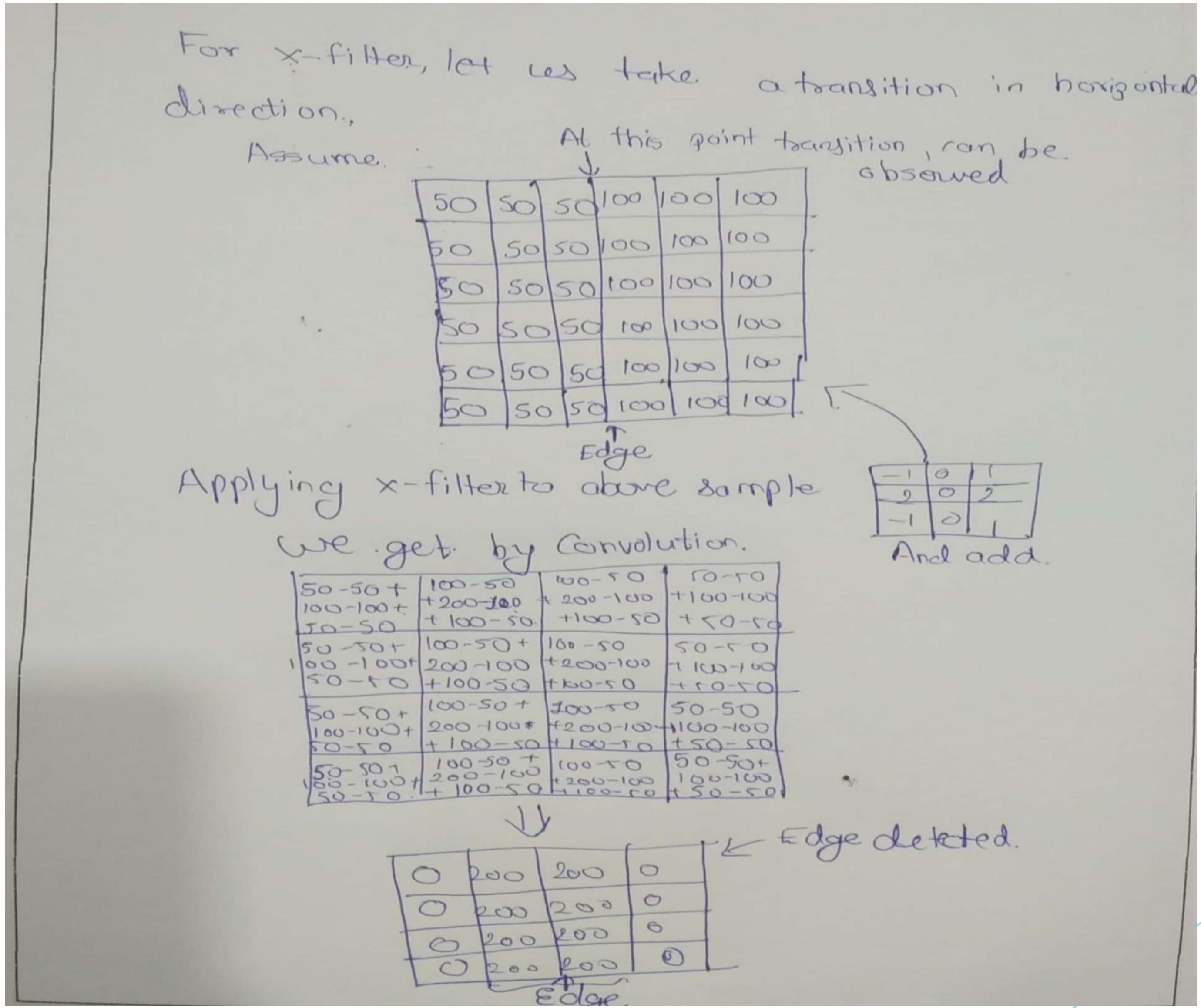
-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter

- An example can be demonstrated as following.



For x-filter, let us take a transition in horizontal direction.,

Assume.

At this point transition, can be observed

50	50	50	100	100	100
50	50	50	100	100	100
50	50	50	100	100	100
50	50	50	100	100	100
50	50	50	100	100	100
50	50	50	100	100	100

Edge

Applying x-filter to above sample

-1	0	1
-2	0	2
-1	0	1

And add.

we get by convolution.

50-50+ 100-100+ 50-50	100-50 +200-100 +100-50.	100-50 +200-100 +100-50	50-50 +100-100 +50-50
50-50+ 100-100+ 50-50	100-50+ 200-100 +100-50	100-50 +200-100 +100-50	50-50 +100-100 +50-50
50-50+ 100-100+ 50-50	100-50+ 200-100+ +100-50	100-50 +200-100 +100-50	50-50 +100-100 +50-50
50-50+ 100-100+ 50-50.	100-50+ 200-100 +100-50	100-50 +200-100 +100-50	50-50+ 100-100 +50-50

⇓

0	200	200	0
0	200	200	0
0	200	200	0
0	200	200	0

Edge detected.

Edge

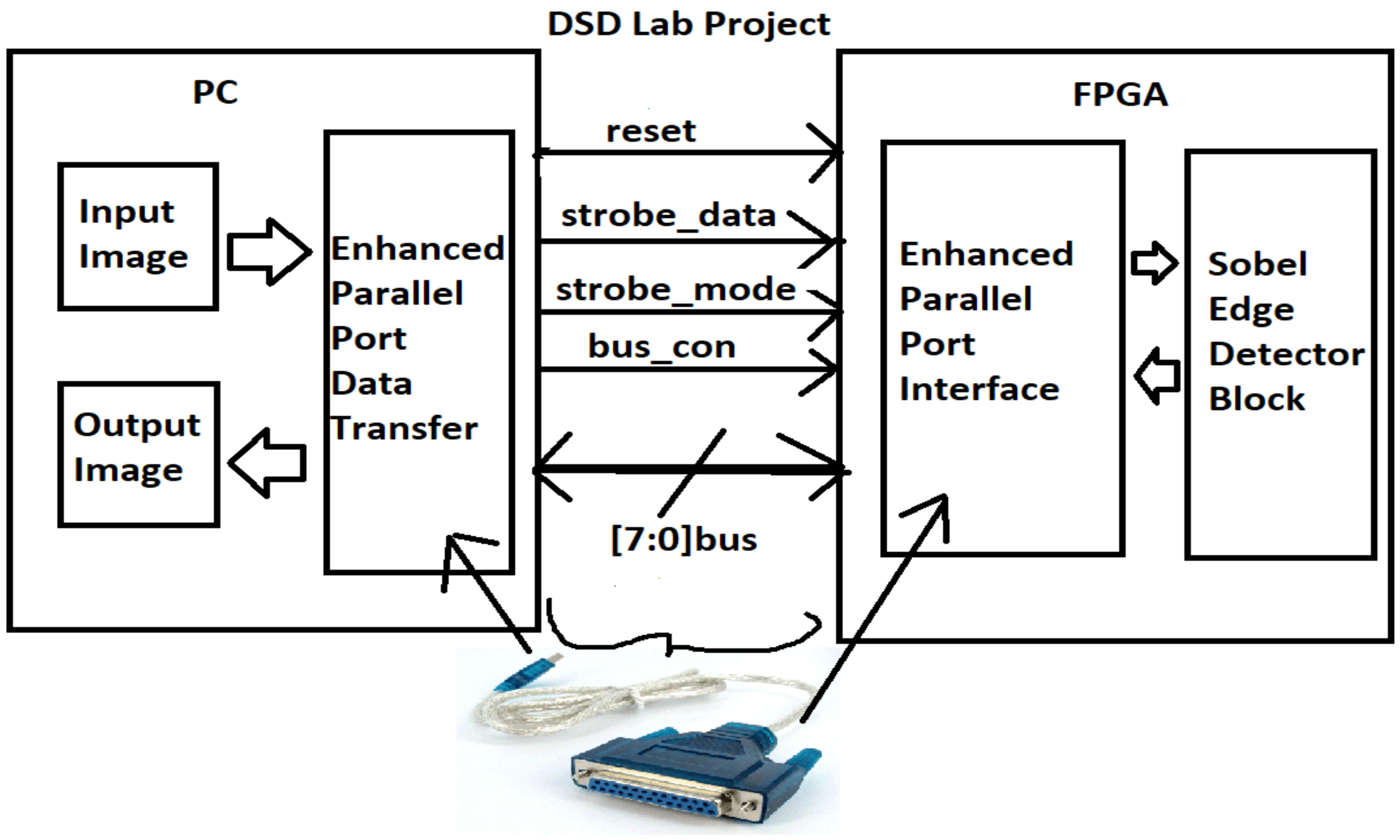
➢ Relevance of Hardware Implementation

- The implementation of image edge detection on hardware is important for increasing processing speed of image processing systems.
- Edge detection is a very complex process affected by noise. A number of operators are defined to solve the problem of edge detection.
- They are effective for certain images, but not suitable for others. Since, it is a very crucial step in Image Processing, we need a more faster, efficient solution.
- Software Implementation can be slower due to limited processor speed, so a dedicated hardware for edge detection can be helpful in Efficient Implementation

➢ PC Interface

- The first step involves in converting the image to a processable data format i.e. in pixels for further actions.
- A pixel data file has to be generated which contains data about image in a form where we use 8-bits to represent a pixel.
- This file is now suitable for providing data for our digital system and enable the further processing steps.
- A parallel port or similar communication ports help in transfer of this data to the FPGA interface. But, one major problem can be mostly with operating voltages at parallel ports (generally, 5V) and for FPGAs(Spartan 3 Series,3.3.V).
- So, generally to comply with the uninterrupted two-way communication, a transceiver such as IC 74LS641 (20 pin DIP IC, two data buses) can be used at the interface.

➢ Overview of System


DSD Lab Project
PC
Input Image
Output Image
Enhanced Parallel Port Data Transfer
reset
strobe_data
strobe_mode
bus_con
[7:0]bus
FPGA
Enhanced Parallel Port Interface
Sobel Edge Detector Block

➢ Software Implementation

- The software implementation is the complete execution of the project undertaken based solely on Xilinx Vivado Tool along with some supported files which enable us to see the results of the implementation
- This section can be classified into a few steps based on the process undertaken sequentially for implementation
  - Verilog Design Code
  - Creating Testbench for a given Input Image
  - Simulation and Fetching Output

- We already know about the evaluation of edges by operator
- In the Verilog Design module ,the 8 pixel values are the neighbouring values for a single pixel with values ranging from 0-255 (8-bit) and ‘edges(8-bit output)’ is the final value of evaluation
- The x & y are the mathematical calculation based (in pixels term)

  where x= (upper right – upper left) + 2*(middle right – middle left)+

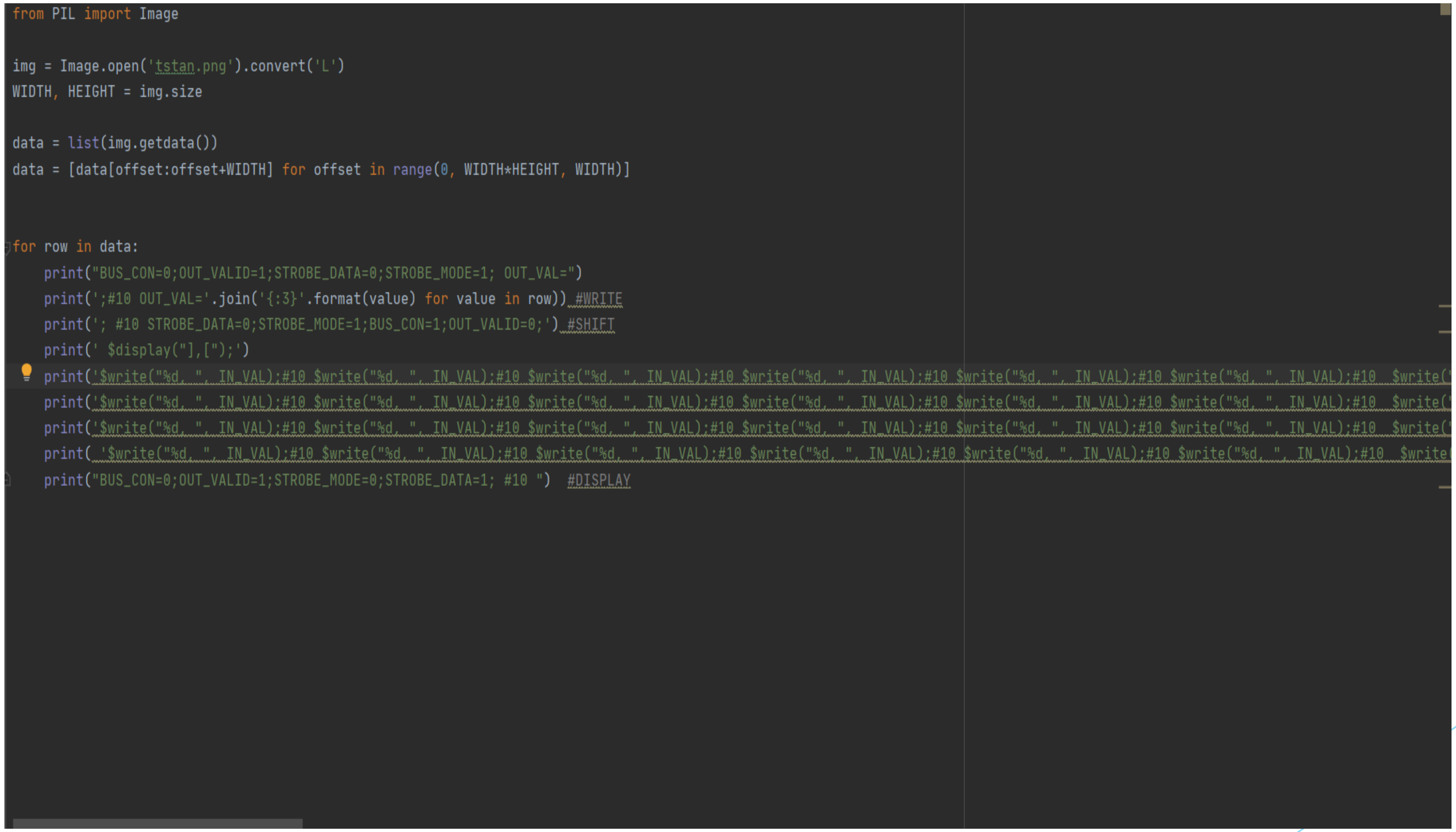
  (lower right – lower left)

  y= (upper left – lower left)+ 2*(upper centre – lower centre) +

  (upper right – lower right)
- Now , the edges can be in any direction i.e. transition from left to right can be either lower->higher pixel value or vice versa and similarly for y-direction

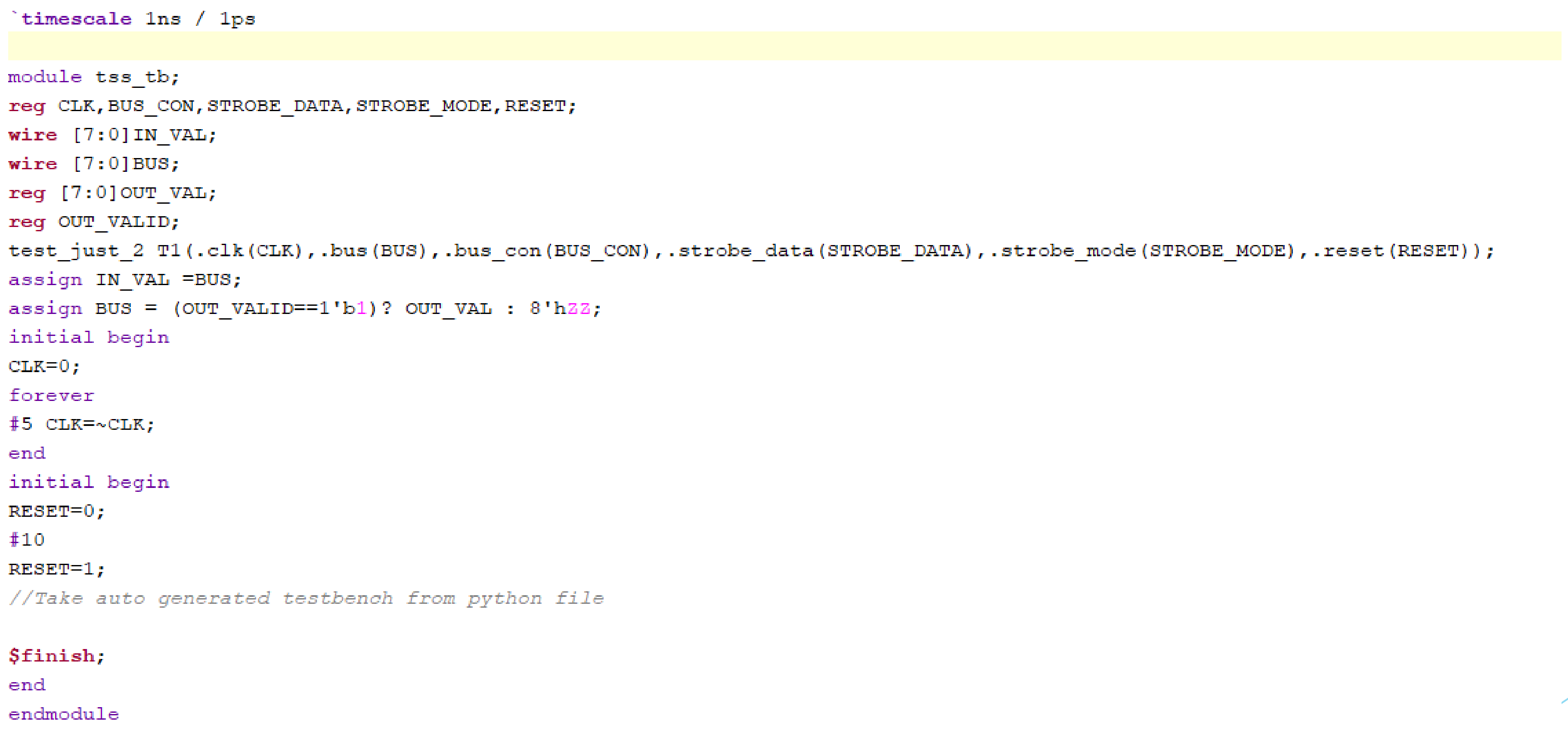
  Hence, mod_x and mod_y are required to treat any transition as edge
- Next ,we will take the resultant of both.
- Finally ,the resultant value is obtained but when back conversion to image will take place, the maximum pixel value should not exceed 255, therefore any value greater than 255 is scaled down to 255 to avoid any error.

Creating Testbench for a given Input Image



```
from PIL import Image

img = Image.open('tstan.png').convert('L')
WIDTH, HEIGHT = img.size

data = list(img.getdata())
data = [data[offset:offset+WIDTH] for offset in range(0, WIDTH*HEIGHT, WIDTH)]


for row in data:
    print("BUS_CON=0;OUT_VALID=1;STROBE_DATA=0;STROBE_MODE=1; OUT_VAL=")
    print(';#10 OUT_VAL='.join('{:3}'.format(value) for value in row)) #WRITE
    print('; #10 STROBE_DATA=0;STROBE_MODE=1;BUS_CON=1;OUT_VALID=0;') #SHIFT
    print(' $display("],[");')
    print('$write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10  $write("
    print('$write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10  $write("
    print('$write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10  $write("
    print( '$write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10 $write("%d, ", IN_VAL);#10  $write(
    print("BUS_CON=0;OUT_VALID=1;STROBE_MODE=0;STROBE_DATA=1; #10 ")  #DISPLAY
```

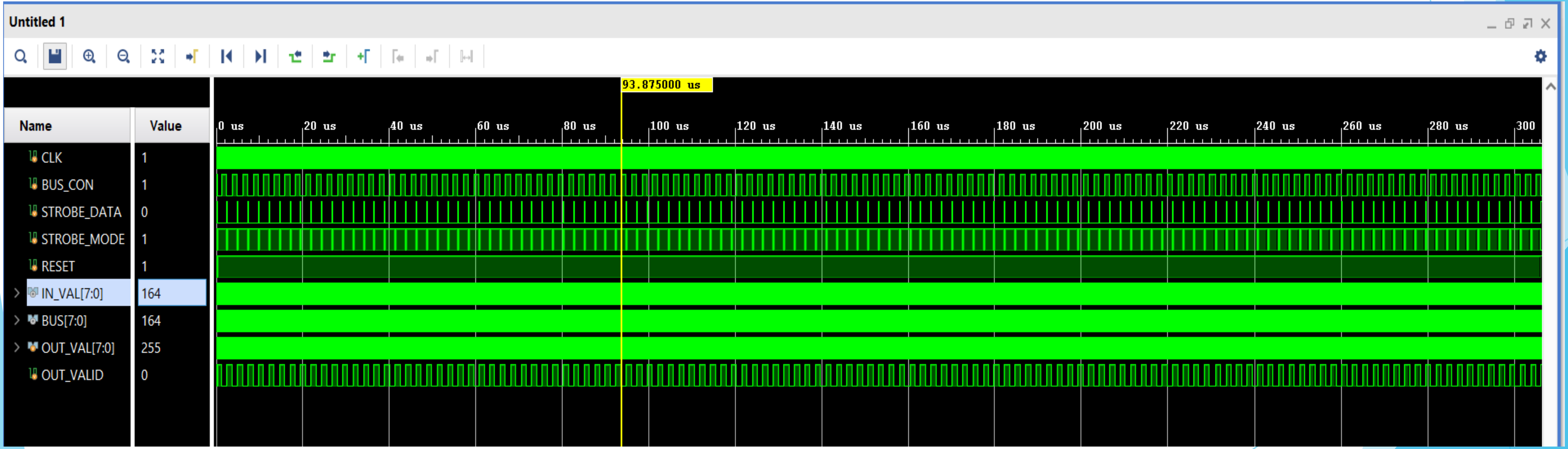
- The python code will help us generate testbench for the Verilog code with respective write , shift and display conditions.
- The testbench can be copied and pasted inside an initial block as given below and observe the output.



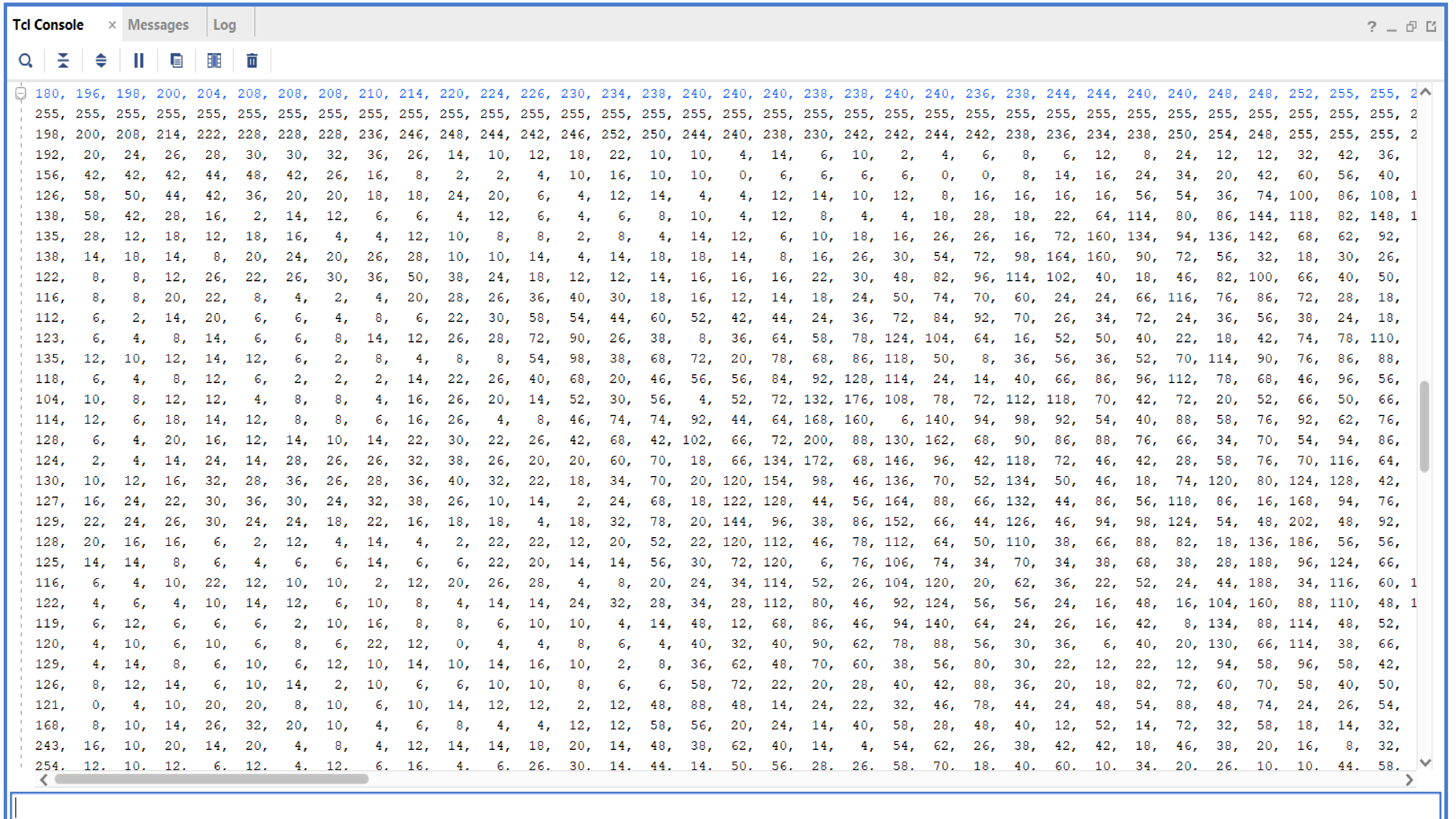
```
`timescale 1ns / 1ps

module tss_tb;
reg CLK,BUS_CON,STROBE_DATA,STROBE_MODE,RESET;
wire [7:0]IN_VAL;
wire [7:0]BUS;
reg [7:0]OUT_VAL;
reg OUT_VALID;
test_just_2 T1(.clk(CLK),.bus(BUS),.bus_con(BUS_CON),.strobe_data(STROBE_DATA),.strobe_mode(STROBE_MODE),.reset(RESET));
assign IN_VAL =BUS;
assign BUS = (OUT_VALID==1'b1)? OUT_VAL : 8'hZZ;
initial begin
CLK=0;
forever
#5 CLK=~CLK;
end
initial begin
RESET=0;
#10
RESET=1;
//Take auto generated testbench from python file

$finish;
end
endmodule
```

Simulation and Fetching Output

- The simulation can be done and we can see the pixel outputs on our Tcl Console
- We’ve already designed the output such that we will get the pixel s for edge-detected image.

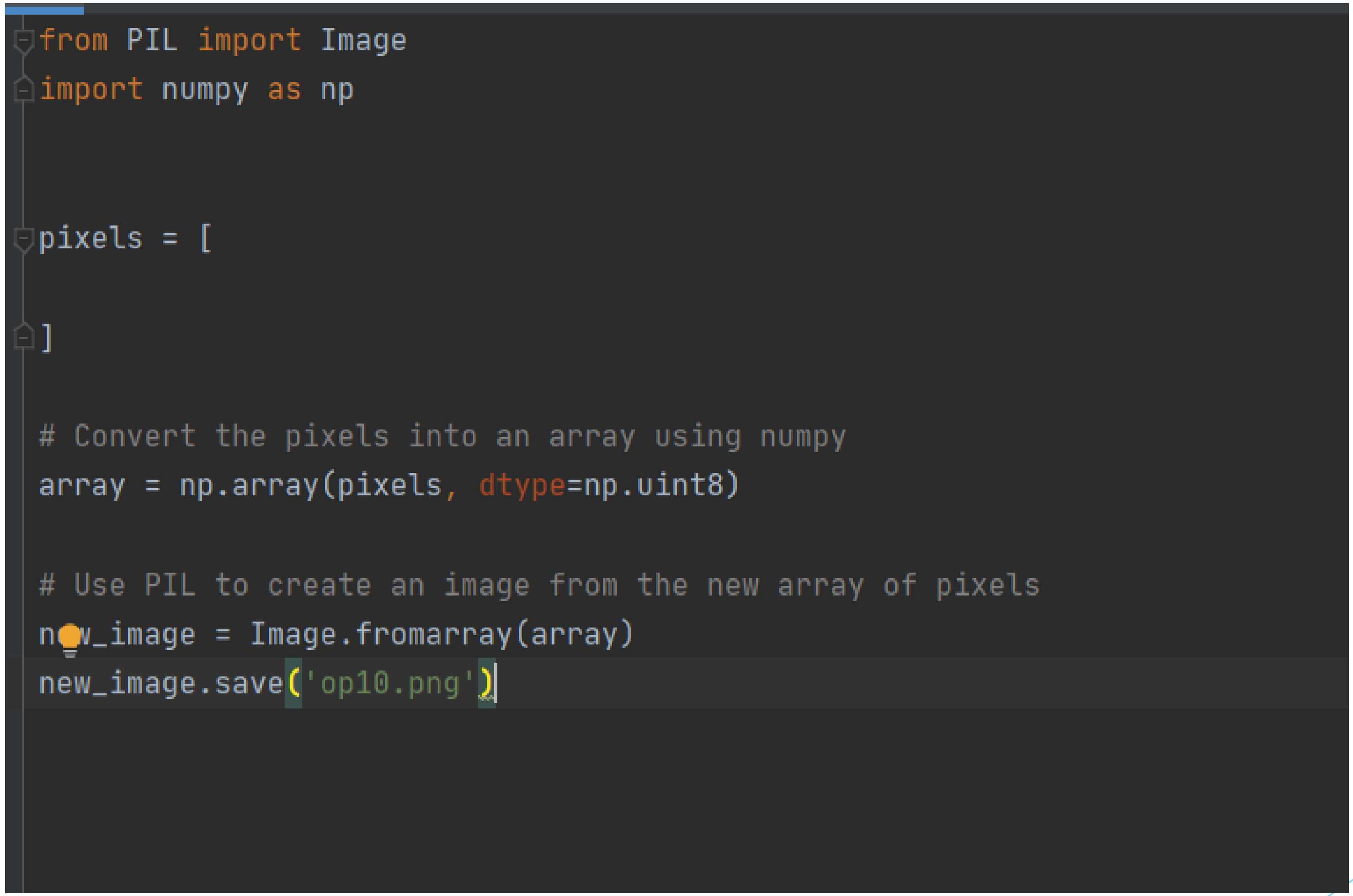

Untitled 1

Name	Value
CLK	1
BUS_CON	1
STROBE_DATA	0
STROBE_MODE	1
RESET	1
IN_VAL[7:0]	164
BUS[7:0]	164
OUT_VAL[7:0]	255
OUT_VALID	0



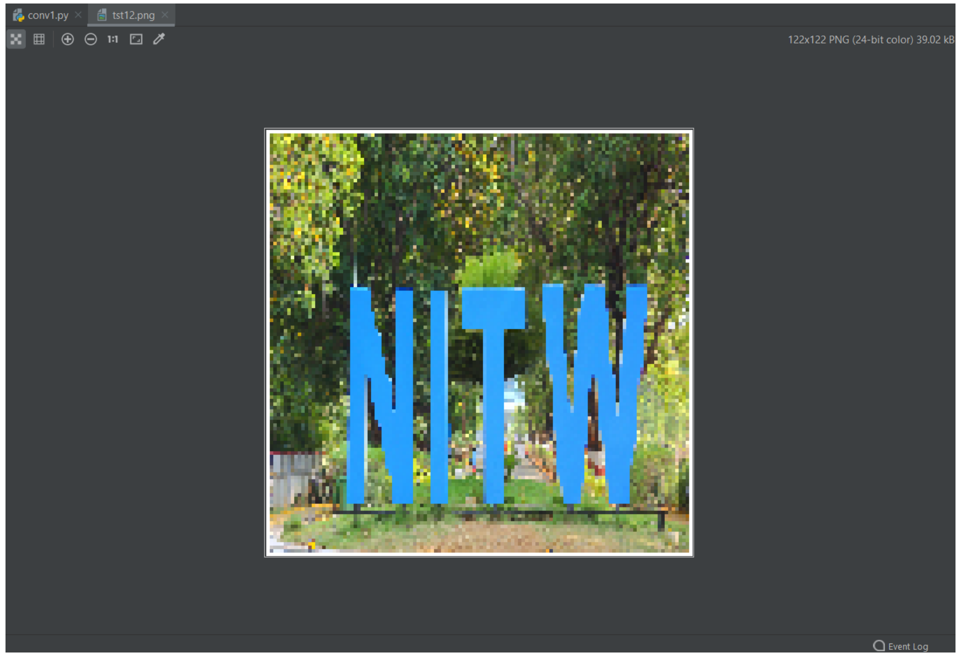
```
180, 196, 198, 200, 204, 208, 208, 208, 210, 214, 220, 224, 226, 230, 234, 238, 240, 240, 240, 238, 238, 240, 240, 236, 238, 244, 244, 240, 240, 248, 248, 252, 255, 255, 2
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 2
198, 200, 208, 214, 222, 228, 228, 228, 236, 246, 248, 244, 242, 246, 252, 250, 244, 240, 238, 230, 242, 242, 244, 242, 238, 236, 234, 238, 250, 254, 248, 255, 255, 255, 2
192,  20,  24,  26,  28,  30,  30,  32,  36,  26,  14,  10,  12,  18,  22,  10,  10,   4,  14,   6,  10,   2,   4,   6,   8,   6,  12,   8,  24,  12,  12,  32,  42,  36,
156,  42,  42,  42,  44,  48,  42,  26,  16,   8,   2,   2,   4,  10,  16,  10,  10,   0,   6,   6,   6,   6,   0,   0,   8,  14,  16,  24,  34,  20,  42,  60,  56,  40,
126,  58,  50,  44,  42,  36,  20,  20,  18,  18,  24,  20,   6,   4,  12,  14,   4,   4,  12,  14,  10,  12,   8,  16,  16,  16,  16,  56,  54,  36,  74, 100,  86, 108, 1
138,  58,  42,  28,  16,   2,  14,  12,   6,   6,   4,  12,   6,   4,   6,   8,  10,   4,  12,   8,   4,   4,  18,  28,  18,  22,  64, 114,  80,  86, 144, 118,  82, 148, 1
135,  28,  12,  18,  12,  18,  16,   4,   4,  12,  10,   8,   8,   2,   8,   4,  14,  12,   6,  10,  18,  16,  26,  26,  16,  72, 160, 134,  94, 136, 142,  68,  62,  92,
138,  14,  18,  14,   8,  20,  24,  20,  26,  28,  10,  10,  14,   4,  14,  18,  18,  14,   8,  16,  26,  30,  54,  72,  98, 164, 160,  90,  72,  56,  32,  18,  30,  26,
122,   8,   8,  12,  26,  22,  26,  30,  36,  50,  38,  24,  18,  12,  12,  14,  16,  16,  16,  22,  30,  48,  82,  96, 114, 102,  40,  18,  46,  82, 100,  66,  40,  50,
116,   8,   8,  20,  22,   8,   4,   2,   4,  20,  28,  26,  36,  40,  30,  18,  16,  12,  14,  18,  24,  50,  74,  70,  60,  24,  24,  66, 116,  76,  86,  72,  28,  18,
112,   6,   2,  14,  20,   6,   6,   4,   8,   6,  22,  30,  58,  54,  44,  60,  52,  42,  44,  24,  36,  72,  84,  92,  70,  26,  34,  72,  24,  36,  56,  38,  24,  18,
123,   6,   4,   8,  14,   6,   6,   8,  14,  12,  26,  28,  72,  90,  26,  38,   8,  36,  64,  58,  78, 124, 104,  64,  16,  52,  50,  40,  22,  18,  42,  74,  78, 110,
135,  12,  10,  12,  14,  12,   6,   2,   8,   4,   8,   8,  54,  98,  38,  68,  72,  20,  78,  68,  86, 118,  50,   8,  36,  56,  36,  52,  70, 114,  90,  76,  86,  88,
118,   6,   4,   8,  12,   6,   2,   2,   2,  14,  22,  26,  40,  68,  20,  46,  56,  56,  84,  92, 128, 114,  24,  14,  40,  66,  86,  96, 112,  78,  68,  46,  96,  56,
104,  10,   8,  12,  12,   4,   8,   8,   4,  16,  26,  20,  14,  52,  30,  56,   4,  52,  72, 132, 176, 108,  78,  72, 112, 118,  70,  42,  72,  20,  52,  66,  50,  66,
114,  12,   6,  18,  14,  12,   8,   8,   6,  16,  26,   4,   8,  46,  74,  74,  92,  44,  64, 168, 160,   6, 140,  94,  98,  92,  54,  40,  88,  58,  76,  92,  62,  76,
128,   6,   4,  20,  16,  12,  14,  10,  14,  22,  30,  22,  26,  42,  68,  42, 102,  66,  72, 200,  88, 130, 162,  68,  90,  86,  88,  76,  66,  34,  70,  54,  94,  86,
124,   2,   4,  14,  24,  14,  28,  26,  26,  32,  38,  26,  20,  20,  60,  70,  18,  66, 134, 172,  68, 146,  96,  42, 118,  72,  46,  42,  28,  58,  76,  70, 116,  64,
130,  10,  12,  16,  32,  28,  36,  26,  28,  36,  40,  32,  22,  18,  34,  70,  20, 120, 154,  98,  46, 136,  70,  52, 134,  50,  46,  18,  74, 120,  80, 124, 128,  42,
127,  16,  24,  22,  30,  36,  30,  24,  32,  38,  26,  10,  14,   2,  24,  68,  18, 122, 128,  44,  56, 164,  88,  66, 132,  44,  86,  56, 118,  86,  16, 168,  94,  76,
129,  22,  24,  26,  30,  24,  24,  18,  22,  16,  18,  18,   4,  18,  32,  78,  20, 144,  96,  38,  86, 152,  66,  44, 126,  46,  94,  98, 124,  54,  48, 202,  48,  92,
128,  20,  16,  16,   6,   2,  12,   4,  14,   4,   2,  22,  22,  12,  20,  52,  22, 120, 112,  46,  78, 112,  64,  50, 110,  38,  66,  88,  82,  18, 136, 186,  56,  56,
125,  14,  14,   8,   6,   4,   6,   6,  14,   6,   6,  22,  20,  14,  14,  56,  30,  72, 120,   6,  76, 106,  74,  34,  70,  34,  38,  68,  38,  28, 188,  96, 124,  66,
116,   6,   4,  10,  22,  12,  10,  10,   2,  12,  20,  26,  28,   4,   8,  20,  24,  34, 114,  52,  26, 104, 120,  20,  62,  36,  22,  52,  24,  44, 188,  34, 116,  60, 1
122,   4,   6,   4,  10,  14,  12,   6,  10,   8,   4,  14,  14,  24,  32,  28,  34,  28, 112,  80,  46,  92, 124,  56,  56,  24,  16,  48,  16, 104, 160,  88, 110,  48, 1
119,   6,  12,   6,   6,   6,   2,  10,  16,   8,   8,   6,  10,  10,   4,  14,  48,  12,  68,  86,  46,  94, 140,  64,  24,  26,  16,  42,   8, 134,  88, 114,  48,  52,
120,   4,  10,   6,  10,   6,   8,   6,  22,  12,   0,   4,   4,   8,   6,   4,  40,  32,  40,  90,  62,  78,  88,  56,  30,  36,   6,  40,  20, 130,  66, 114,  38,  66,
129,   4,  14,   8,   6,  10,   6,  12,  10,  14,  10,  14,  16,  10,   2,   8,  36,  62,  48,  70,  60,  38,  56,  80,  30,  22,  12,  22,  12,  94,  58,  96,  58,  42,
126,   8,  12,  14,   6,  10,  14,   2,  10,   6,   6,  10,  10,   8,   6,   6,  58,  72,  22,  20,  28,  40,  42,  88,  36,  20,  18,  82,  72,  60,  70,  58,  40,  50,
121,   0,   4,  10,  20,  20,   8,  10,   6,  10,  14,  12,  12,   2,  12,  48,  88,  48,  14,  24,  22,  32,  46,  78,  44,  24,  48,  54,  88,  48,  74,  24,  26,  54,
168,   8,  10,  14,  26,  32,  20,  10,   4,   6,   8,   4,   4,  12,  12,  58,  56,  20,  24,  14,  40,  58,  28,  48,  40,  12,  52,  14,  72,  32,  58,  18,  14,  32,
243,  16,  10,  20,  14,  20,   4,   8,   4,  12,  14,  14,  18,  20,  14,  48,  38,  62,  40,  14,   4,  54,  62,  26,  38,  42,  42,  18,  46,  38,  20,  16,   8,  32,
254,  12,  10,  12,   6,  12,   4,  12,   6,  16,   4,   6,  26,  30,  14,  44,  14,  50,  56,  28,  26,  58,  70,  18,  40,  60,  10,  34,  20,  26,  10,  10,  44,  58,
```

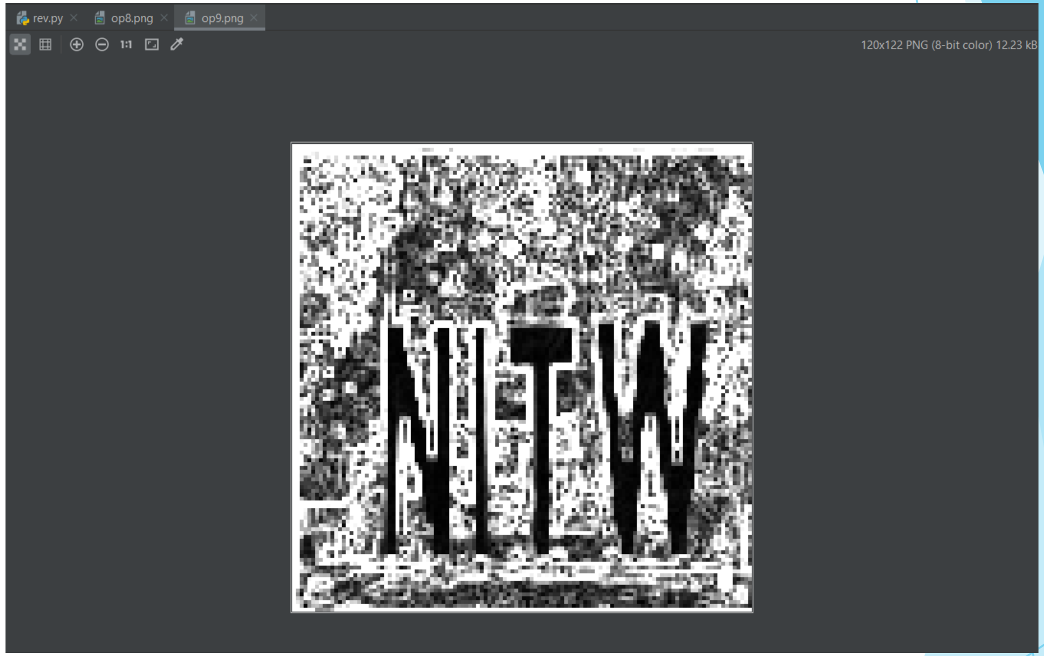
- Now we can copy the contents of Tcl Console to another python file to obtain the image back from pixel data.
- We will paste the contents inside the pixels=[ ] array in the given python file

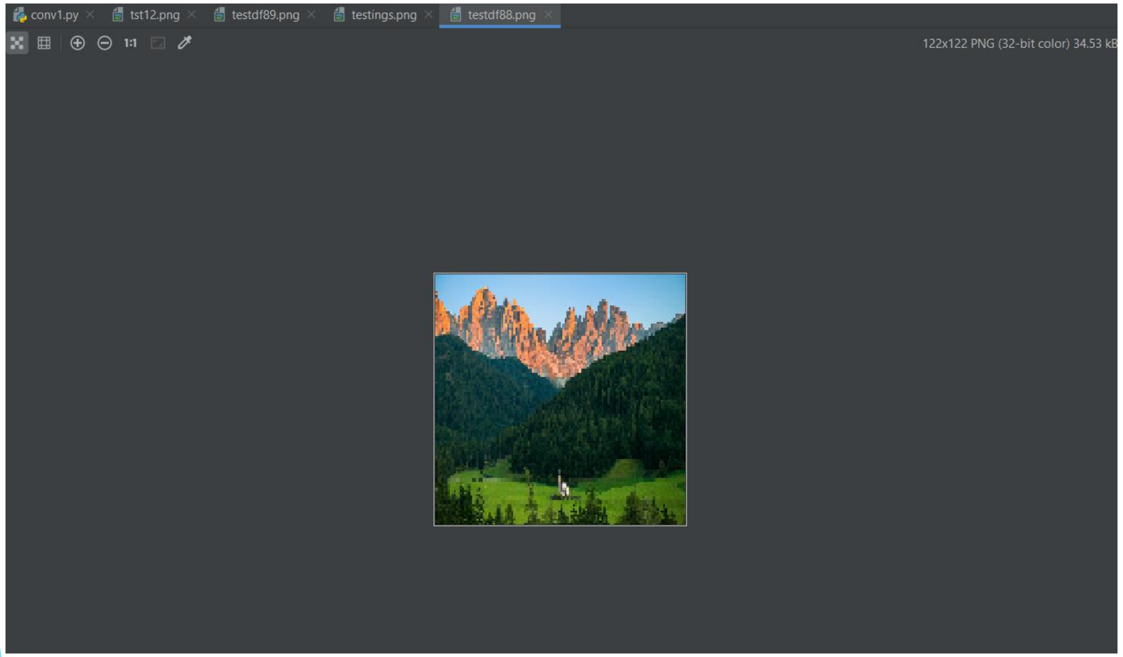


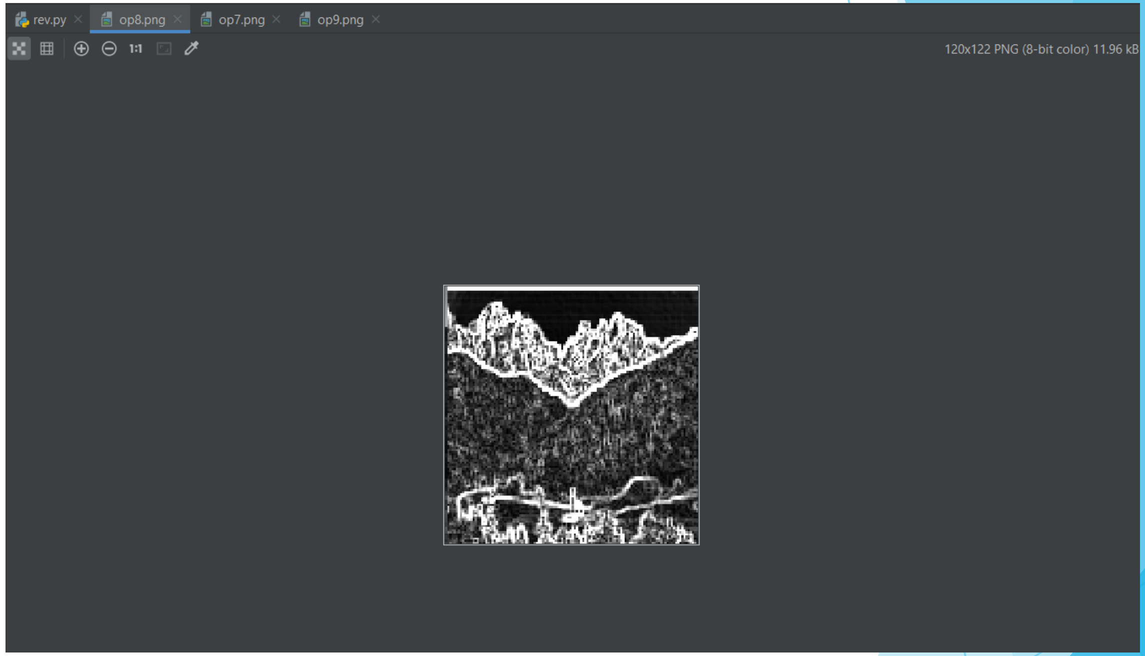
```
from PIL import Image
import numpy as np


pixels = [

]

# Convert the pixels into an array using numpy
array = np.array(pixels, dtype=np.uint8)

# Use PIL to create an image from the new array of pixels
new_image = Image.fromarray(array)
new_image.save('op10.png')
```

- Now we got the output from the given input and we’ve completed the software implementation
- Some Examples :






conv1.py
tst12.png
testdf89.png
testings.png
testdf88.png
1:1
122x122 PNG (32-bit color) 34.53 kB


rev.py
op8.png
op7.png
op9.png
1:1
120x122 PNG (8-bit color) 11.96 kB

➢ References

- Basic Idea

 https://www.hackster.io/

- Wikipedia https://en.wikipedia.org/wiki/Sobel_operator
- Sobel Operator https://www.youtube.com/watch?v=uihBwtPIBxM

Thank You !