Hardware Accelerator for Edge Detection Implementation and Demo

Amish Mittal

1801CS07

CS225 Switching Theory Project

Indian Institute of Technology (IIT) Patna

Mentor – Dr Jimson Mathew

What is Edge Detection?

- Edge detection is an image processing technique for finding the boundaries of objects within images.
- It works by detecting discontinuities in brightness.

Images used for academic reference purposes.

Applications of Edge Detection

- Autonomous cars and drones
- Fingerprint matching
- License Plate detection (traffic speed cams)
- Optical Character Recognition
- Medical diagnosis
- It is one of the most basic and essential task in Computer Vision.

But why Hardware Accelerator?

- Advantages:
  - Ultra low latency as compared to a normal processor (key for 5G devices)
  - Customized secure solution for a particular task
  - Very low power usage – more battery life
  - Memory efficient
  - Full power of parallel processing can be utilized

- Disadvantages:
  - Much more difficult to code and plan than a vanilla Python implementation
  - Complex simulation tools for testing

- These disadvantages can be overcome by High Level Synthesis (HLS). Many libraries to convert C, Python to HDL are now available. Example: LeFlow (open source)

My Implementation for FPGA

Steps:

1. Read image in greyscale format (8-bit: 640x480)– already converted to pixel values.
2. Apply Sobel Convolution mask for both horizontal and vertical direction using Kx and Ky. Our output would now have 2 less pixel rows and columns due to the 3x3 filter. We get Ix and Iy.

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

3. Calculate magnitude of gradient. This would give the result.

$$|G| = \sqrt{I_x^2 + I_y^2}, \qquad \text{or } |\text{Ix}| + |\text{Iy}|$$

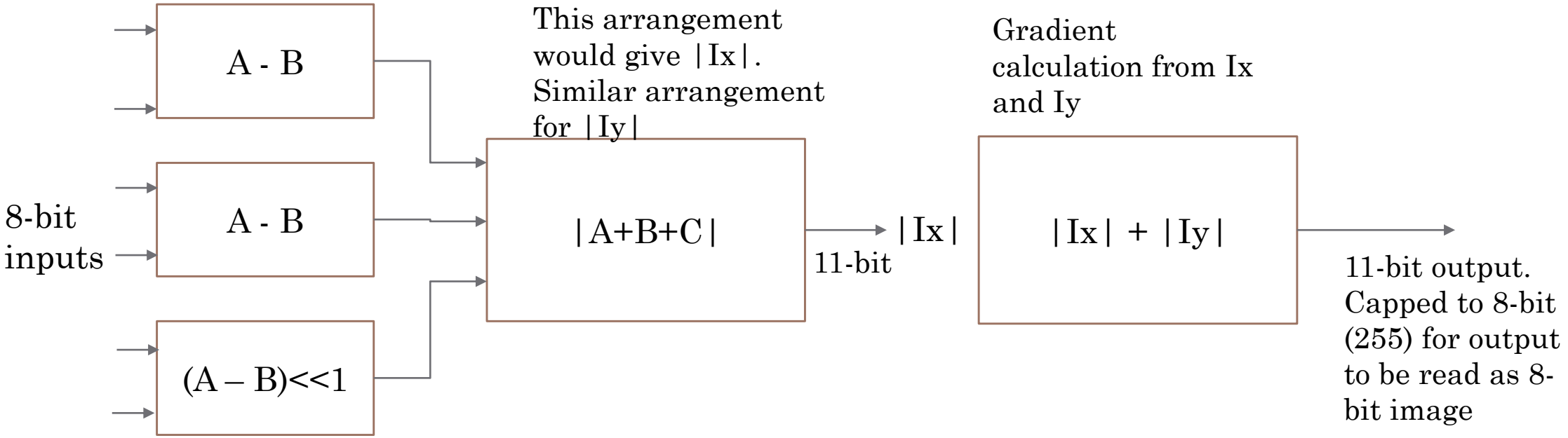
4. View the output using an image processing tool.

My Implementation for FPGA

Now,

$$Ix[i, j] = p[i+1,j-1] + 2*p[i+1, j] + p(i+1,j+1) - \{p[i-1, j-1] + 2*p[i-1, j] + p(i-1, j+1)\}$$

$$Iy[i, j] = p[i-1,j+1] + 2*p[i, j+1] + p(i+1,j+1) - \{p[i-1, j-1] + 2*p[i, j-1] + p(i+1, j-1)\}$$



Sample Results

- My Image

Input
(before conversion to grayscale)

```
164 165 166 166 167 168 169 170 168 169 171 171 170 171 172 173 171 172 172 173 174 175 176 176 175 175 175 176 176
210 211 211 211 211 211 211 211 212 212 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 214 214 215
213 213 212 212 212 212 212 213 213 213 213 212 212 211 210 210 210 210 209 209 208 208 208 208 208 208 208 207 206
167 165 165 166 168 169 169 169 168 170 171 171 171 171 172 173 172 172 173 173 174 175 175 175 175 175 175 176 176
211 211 211 211 212 212 212 212 212 212 213 213 213 213 213 213 213 213 213 213 213 213 213 213 214 213 214 214 215 215
213 213 212 212 212 212 212 213 212 212 212 212 212 211 210 210 210 210 209 209 208 208 208 208 208 208 207 207 206
169 167 165 166 169 171 170 168 169 170 171 171 171 172 173 173 173 173 173 174 174 174 175 175 175 175 175 176 176
212 212 212 212 212 212 212 212 212 212 213 213 213 213 213 213 213 213 213 213 213 214 214 214 214 214 214 215 215
213 213 212 212 212 212 212 212 212 212 212 212 211 211 210 210 211 210 210 209 209 208 208 209 208 208 207 207 206
169 168 167 167 169 170 170 169 170 170 171 171 172 172 173 173 172 173 173 174 175 175 175 176 176 176 176 176 176
211 212 212 212 212 212 212 212 212 212 213 213 213 213 213 213 213 213 213 213 213 214 214 214 214 214 214 215 215
213 212 212 212 212 212 212 212 211 211 211 211 211 211 211 210 211 210 210 209 209 209 209 209 207 207 207 206 206
168 169 169 169 168 168 169 171 170 170 171 171 172 173 173 173 172 172 173 173 174 175 176 176 176 175 175 175 176
211 211 211 211 211 212 212 212 212 212 213 213 213 213 213 213 213 213 213 213 214 214 214 214 214 214 215 215 215
212 212 212 212 212 212 212 211 211 211 211 211 211 211 210 210 210 210 209 209 208 208 208 208 207 207 206 206 206
167 169 170 169 167 167 169 172 171 170 170 171 172 173 172 172 171 172 172 173 174 175 176 176 175 175 175 175 176
211 211 211 211 211 212 212 212 212 212 213 213 213 213 213 213 213 213 213 213 214 214 214 215 214 215 215 215 215
212 212 212 212 212 212 211 211 211 211 211 211 211 210 210 209 210 209 209 208 208 208 208 208 206 206 206 206 205
167 168 169 169 168 168 169 171 171 170 170 170 172 172 172 171 172 172 173 173 174 175 175 176 174 174 175 175 176
211 212 212 212 212 212 212 212 212 212 213 213 213 213 213 213 213 213 213 213 214 214 214 215 215 215 215 215 215
211 212 212 212 212 212 211 211 211 211 211 211 210 210 209 209 209 209 208 208 207 207 207 207 206 206 206 206 205
167 167 167 168 168 169 170 170 171 170 169 170 171 172 172 171 173 173 173 174 174 174 175 175 174 174 174 175 176
212 212 212 212 212 213 213 213 212 212 213 213 213 213 213 213 213 213 213 214 214 215 215 215 215 215 215 215 215
```

After 1st python script to convert to pixels

Sample Results

- My Image

```
255
  0
  0
 14
  2
 12
 20
 14
  4
  6
  4
 14
  6
  2
  4
 10
 10
  4
  8
  8
  8
  8
  6
```

Modelsim Output

Final Output (edges shown)

Sample Results

- My Image

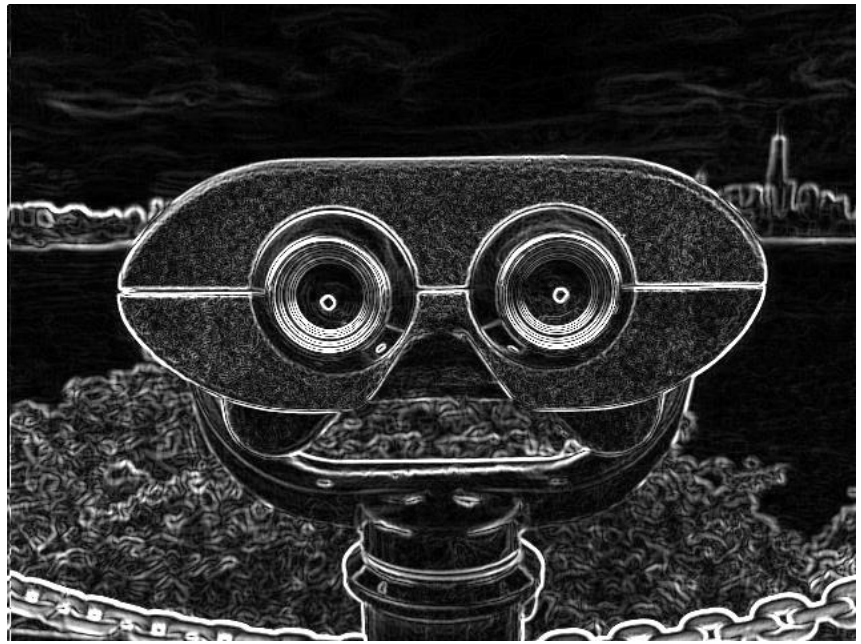
Initial

Final

Sample Results

- Random test image

Initial



Final

Modelsim Simulation

Drive link for code

- The code for the project can be accessed at: https://drive.google.com/file/d/1mWdxIe4YpN-HX4KeydFblvWMNWhKu2fS/view?usp=sharing

Future Work

1. Work on Gaussian Filter before Sobel filter. It is used to reduce the effect of noise.
2. Work on the complete Canny Edge detection algorithm. It gives a much better result for the edges.
3. Try out the code on an FPGA board.

References

- Inspired from multiple Image Processing FPGA implementations available on the internet.