Efficient Implementation of Mean Formula for Image Processing using FPGA Device

Aqwam Rosadi Kardian and Sunny Arief Sudiro
STMIK Jakarta STI&K
Jakarta, Indonesia
{aqwam,sunny}@jak-stik.ac.id

Sarifuddin Madenda
Universitas Gunadarma
Jakarta, Indonesia
sarif@staff.gunadarma.ac.id

Abstract—Image processing needs a process based on mathematical formula applied to image data. In software it is easy to do that and accessing data from memory. The other hand for hardware implementation with a lot of constraint. This article propose an implementation of optimum mean formula in FGPA Device. For mean calculation only need one addition component ( in one accumulator) and one division using shift right register, for 8x8 image size need 64 clock cycle to finish the mean calculation.

Keywords—mean, histogram, FPGA, counter, image

I. INTRODUCTION

digital image processing is a proses for manipulating and analysing image with computer. Image processing can be divided into tow kind of activities:

1. Increasing the quality of image for easy interpretation by human eye, this activity known as image enhancement
2. Processing information in the image for object identification automatically

Second application has tight correlation with pattern recognition which usually recognizing the object by extracting important information from the image (image features).

Some image processing uses basic statistic formula such as: histogram, mean, variance, etc. In embedded environment those formula must be implemented in hardware base, like FPGA for example. A research for histogram calculation for 8x8 image need 256 addition, 320 clock cycle and 256 clock for interface with memory 2KB. [1]

Other research for histogram and mean calculation base on FPGA device is proposed in [2]. This article proposed a method that need 512 addition operation, 64 cycles, 1 division (using shift right register), no need for memory but using 256 register 8 bit for 8x8 image size.

Another research for histogram calculation is proposed in [3]. This approach use RAMsubcel and FSM for controlling, this approach use 850 logic element of FPGA and can work at more than 100 MHz of frequency clocks and able to process 260 frame per seconds.

II. BASIC IMAGE PROCESSING FORMULA

A. Image Histogram

Image histogram is a diagram that draw frequency of the appearance every intensity value from the whole image pixel element. [4][5]. The higher value of histogram show the number of pixels with that intensity value is high and vice versa. Histogram can show the brightness and contrast of the image. A lot of use histogram in image texture analysis because of the simplicity of the algorithm. Histogram mathematical formula is:

\[ H(i) = \sum_{n=1}^{N} \sum_{m=1}^{M} 1, \quad if \; i = f(n,m) \] (1)

\( H(i) \) is histogram of the image with NxM size and i is intensity value of the pixel in the image. Fig 1. shows some histogram diagram from several image with their own texture. The images are obtain from http://www.freeimages.com/search/texture

B. Mean Formula

From the histogram value we can calculate the ‘mean’ value from the image. Mathematical formula for ‘mean’ (μ) is: [4]

\[ \mu = \sum_{i=0}^{L-1} i \cdot p(i) \] (2)

In this formula i is grey level of intensity pixel in the image and \( p(i) \) is the probability of occurrence i. L is the higher value of grey level i. This formula will produce an average brightness of objects
III. PROPOSED ALGORITHM

Mean is the first-order statistical analysis methods used for segmentation and feature extraction processes in an image. Mean represents the mean value (mean) of the intensity of the entire pixel in an image. Referring to the equation (2) and (1) where the mean is calculated by using the calculation result histogram, equation 3 is proposed:

\[ \mu = \frac{1}{MN} \sum_{i=1}^{MN} i \cdot p(i) \]

The psudo code of mean calculation based on equation 3 is showed in Pseudo Code 1.

---

Psudo code 1. Mean formula (1st version)

1. \( H = \% \text{ data} \% \)
2. \( h = \text{zeros}(256,1) \)
3. \[ [N,M] = \text{size}(H) \]
4. for \( i = 1:N \)
5. for \( j = 1:M \)
6. \( h(I(i,j)+1) = h(I(i,j)+1)+1; \)
7. end
8. end
9. \( N1 = (N*\text{M}) \)
10. for \( i = 1:256 \)
11. \( p(i) = h(i)/N1 \)
12. end
13. Mean = 0;
14. for \( i = 1:256 \)
15. \( \text{Mean} = \text{Mean} + p(i) \ast (i-1) \)
16. end
17. Mean

Psudo code 2. Mean formula (revised version)

1. \( [M,N,L] = \text{size}(\text{Im}); \% \text{Im is the data} \)
2. \( \text{htsum}=0; \)
3. for \( i=1:M \)
4. for \( j=1:N \)
5. \( \text{htsum} = \text{htsum}+\text{Im}(i,j); \)
6. end
7. end
8. \( \text{optimalmean} = \text{htsum}/(M*N) \)

Psudo code 2 is the revised algoirithm with some efficiency, reducing the loop and mathematical operation.

The results of mean calculation using basic algorithms and optimization algorithms for the same data (Im) are the same, as shown in Figure 2. The values resulting from the evaluation using Matlab will be used for comparison with values that are processed using FPGA components. The time complexity of the algorithm decreases with decreasing the number of arithmetic operations.

\[
\text{Im} = \begin{bmatrix}
6 & 6 & 6 & 6 & 10 & 10 & 8 & 8 \\
6 & 6 & 48 & 48 & 10 & 10 & 8 & 8 \\
6 & 6 & 48 & 48 & 10 & 10 & 8 & 8 \\
6 & 6 & 48 & 48 & 13 & 13 & 10 & 10 \\
6 & 6 & 52 & 52 & 13 & 13 & 10 & 10 \\
5 & 5 & 5 & 5 & 52 & 4 & 4 & 12 & 12 \\
5 & 5 & 5 & 5 & 52 & 4 & 4 & 12 & 12 \\
\end{bmatrix}
\]

![Fig. 1. Example of a histogram diagram comming from several image.](image1)

![Fig. 2. Result form both algorithm with the same data.](image2)
IV. HARDWARE IMPLEMENTATION

A. Component Design

Figure 3 is the first design based on first pseudo code, this design need a selector to select intensity value of pixel from the image between 0 to 255 and send to thier respective accumulator (256 accumulator with 256 addition component). All these accumulator value will be send to 256 input addition (or another 256 addition) so we need 512 addition for this design. And finnally to obtain mean value we divided with the number of pixel wich is 8x8 = 64. We use shift right register 6 bit to do this division operation.

![Component design for first pseudo code](image)

Fig. 3. Component design for first pseudo code[2]

Figure 4 is the new Component design for 2nd pseudo code. Figure 5 and 6 are the new design based on the design in figure 4, to calculate mean value actually we don’t have to calculate histogram, we can directly using one accumualtor to hold the total value of pixel element value (htsum) in the 2nd pseudo code. This approach give us the reduction of mathematical operation (component). This design only need one addition in one accumulator and one shift right register, with the same processing time (64 clock cycles) for 8x8 image size and the same result of mean value.

![Entity Diagram of efficient component](image)

Fig. 5. Entity Diagram of efficient component

![RTL Schematic Diagram of efficient component](image)

Fig. 6. RTL Schematic Diagram of efficient component.

Figure 7 show the behavioral simulation result based on behavioral simulation. This figure show the result for the same data which is 14, at the end for 64 clock cycles, comparing to the Matlab evaluation result wich is 14.5 the difference is 0.5 or an error of 0.005 %. The speed of process is paralel with the comming of data input ( data in).

![Behavioral simulation result for proposed component](image)

Fig. 7. Behavioral simulation result for proposed component

B. Simulation

Figure 6 show the simulation result based on behavioral simulation. This figure show the result for the same data which is 14, at the end for 64 clock cycles, comparing to the Matlab evaluation result which is 14,5 the difference is 0,5 or an error of 0,005 %. The speed of process is parallel with the coming of data input (data in).

V. CONCLUSION

The efficient implementation of mean formula into hardware component using FPGA device has been proposed. This component use one addition and one shift right register with 64 clock cycles to calculate mean value for 8x8 image size. This design needs 14 slices of flip-flops and 14 of 4 input LUTs. The difference (as an error) of mean value comparing to Matlab result is 0,5 or 0,005 % due to floating point problem in FPGA.

Acknowledgment

Thanks to Yayasan Pendidikan Gunadarma and Yayasan Ilmu Komputer for supporting this research as Doctoral research in Doctoral Program Universitas Gunadarma, Jakarta Indonesia.

References


```mermaid
graph TD
    D[Shift Register]
    Data[64] --> E
    E[Selector] --> F[One accumulator] --> G[Shift right register]
    G[Shift right register]
    Data[64] --> H
    H[One accumulator] --> I[Accumulator]
    I[Accumulator] --> J[Adder]
    J[Adder] --> K[Shifter]
    K[Shifter]
    Data[64] --> L
    L[Adder with 256 input] --> M[Shift Register]
    M[Shift Register]
    Data[64] --> N
    N[Adder with 256 input] --> O[Accumulator]
    O[Accumulator] --> P[Adder]
    P[Adder] --> Q[Shifter]
    Q[Shifter]
    Data[64] --> R
    R[Adder with 256 input] --> S[Accumulator]
    S[Accumulator] --> T[Adder]
    T[Adder] --> U[Shifter]
    U[Shifter]
    Data[64] --> V
    V[Adder with 256 input] --> W[Accumulator]
    W[Accumulator] --> X[Adder]
    X[Adder] --> Y[Shifter]
    Y[Shifter]
    Data[64] --> Z
    Z[Adder with 256 input] --> AA[Accumulator]
    AA[Accumulator] --> BB[Adder]
    BB[Adder] --> CC[Shifter]
    CC[Shifter]
    Data[64] --> DD
    DD[Adder with 256 input] --> EE[Accumulator]
    EE[Accumulator] --> FF[Adder]
    FF[Adder] --> GG[Shifter]
    GG[Shifter]
    Data[64] --> HH
    HH[Adder with 256 input] --> II[Accumulator]
    II[Accumulator] --> JJ[Adder]
    JJ[Adder] --> KK[Shifter]
    KK[Shifter]
    Data[64] --> LL
    LL[Adder with 256 input] --> MM[Accumulator]
    MM[Accumulator] --> NN[Adder]
    NN[Adder] --> OO[Shifter]
    OO[Shifter]
    Data[64] --> PP
    PP[Adder with 256 input] --> QQ[Accumulator]
    QQ[Accumulator] --> RR[Adder]
    RR[Adder] --> SS[Shifter]
    SS[Shifter]
    Data[64] --> TT
    TT[Adder with 256 input] --> UU[Accumulator]
    UU[Accumulator] --> VV[Adder]
    VV[Adder] --> WW[Shifter]
    WW[Shifter]
    Data[64] --> XX
    XX[Adder with 256 input] --> YY[Accumulator]
    YY[Accumulator] --> ZZ[Adder]
    ZZ[Adder] --> AAA[Shifter]
    AAA[Shifter]
    Data[64] --> BBB
    BBB[Adder with 256 input] --> CCC[Accumulator]
    CCC[Accumulator] --> DDD[Adder]
    DDD[Adder] --> EEE[Shifter]
    EEE[Shifter]
    Data[64] --> FFF
    FFF[Adder with 256 input] --> GGG[Accumulator]
    GGG[Accumulator] --> HHH[Adder]
    HHH[Adder] --> IEE[Shifter]
    IEE[Shifter]
    Data[64] --> JJJ
    JJJ[Adder with 256 input] --> KKK[Accumulator]
    KKK[Accumulator] --> LLL[Adder]
    LLL[Adder] --> MNN[Shifter]
    MNN[Shifter]
    Data[64] --> NN
    NN[Adder with 256 input] --> PPP[Accumulator]
    PPP[Accumulator] --> QQQ[Adder]
    QQQ[Adder] --> RR
    RR --> SS[Shifter]
    SS[Shifter]
    Data[64] --> TT
    TT[Adder with 256 input] --> UU[Accumulator]
    UU[Accumulator] --> VV[Adder]
    VV[Adder] --> WW[Shifter]
    WW[Shifter]
    Data[64] --> XX
    XX[Adder with 256 input] --> YY[Accumulator]
    YY[Accumulator] --> ZZ[Adder]
    ZZ[Adder] --> AAA[Shifter]
    AAA[Shifter]
    Data[64] --> BBB
    BBB[Adder with 256 input] --> CCC[Accumulator]
    CCC[Accumulator] --> DDD[Adder]
    DDD[Adder] --> EEE[Shifter]
    EEE[Shifter]
```