A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing

Abstract:

Binarization plays an important role in document image processing, particularly in degraded document images. Among all local image thresholding algorithms, Sauvola has excellent binarization performance for degraded document images. However, this algorithm is computationally intensive and sensitive to the noises from the internal computational circuits. In this paper, we present a stochastic implementation of Sauvola algorithm. Our experimental results show that the stochastic implementation of Sauvola needs much less time and area and can tolerate more faults, while consuming less power in comparison with its conventional implementation. The proposed architecture of this paper analysis the logic size, area and power consumption using Xilinx 14.2.

Enhancement of the project:

Existing System:

The main step of calculating $t(x, y)$ for each image pixel in Sauvola is to compute $m(x, y)$ and $s(x, y)$. Since we fixed the window size to $9 \times 9$, we have to calculate the mean and the standard deviation of 81 local numbers using

$$\text{mean} = \frac{a_1 + a_2 + a_3 + \cdots + a_{81}}{81}$$

$$\text{standard deviation} = \sqrt{|\text{mean}(x^2) - \text{mean}(x)^2|}.$$

To implement the required square root function used in the standard deviation, we use Newton–Raphson, a method for finding successively better approximations to the root of a real value.
number. Fig. 1 shows the implemented block diagram of the conventional architecture of Sauvola method.

![Diagram](image)

**Fig. 1. Simple view of the conventional process of calculating threshold value for each pixel of an input image using Sauvola method.**

**Disadvantages:**

- High area and power consumption

**Proposed System:**
Stochastic Implementation In stochastic implementation, we have to scale down all pixel intensities from [0, 255] to [0, 1] interval. Therefore, the R constant in the Sauvola equation changes to 1. The new modified Sauvola equation for our stochastic design will be

\[ t(x, y) = m \cdot [1 + 0.5(S - 1)] = m \cdot (s + 1)/2. \]  

(1)

Now, to process all pixels, we need to do three steps:

1) Converting pixel values into stochastic streams;

2) Generating threshold streams;

3) Determining the output binary values.

1) Phase 1 (Generating Stochastic Bit Streams):

To convert pixel intensities from binary format into stochastic streams, we use the stochastic number generator (SNG) presented. In generating pseudorandom numbers required in this SNG, we used different maximum period linear feedback shift registers (LFSRs) corresponding to each different lengths of streams (n-bit LFSR for \(2^n\) stream).

2) Phase 2 (Generating Threshold Bit Streams):

a) First step (averaging local window bit streams): To build an 81-to-1 SMC, we propose to combine nine 16-to-1 SMCs, each one as a 9-to-1 SMC. We use a simple technique to convert the existing 16-to-1 SMC to a new 9-to-1 SMC.

b) Second step (generating standard deviation bit stream): In order to determine the standard deviation of some input numbers, we need to have: 1) the average value of the squares and 2) the square of the average values. The proposed 81-to-1 SMC in Fig. 2 can average 81 input streams. Since this average is in stochastic form, generating its square will only need an AND gate with two uncorrelated versions of the input.
c) Third step (generating threshold bit stream): Considering (1), to generate threshold bit streams, we need to perform two other simple operations. First, we should do a simple scaled addition to convert the s bit stream to \((s + 1)/2\). Second, multiplying \(m\) and \((s +1)/2\) bit streams by a simple AND gate. Fig. 3 shows the process of producing the threshold bit stream for each pixel of image in Sauvola method.
Fig. 3. Simple view of the process of calculating threshold bit stream stochastically for each pixel of an input image using Sauvola method.

3) Phase 3 (Generating Output Binary Values): Now we reach to the final step, the estimation of 0 or 1 output binary values by comparing the produced threshold bit stream with the corresponded pixel intensity bit stream. This function will be the responsibility of a specific circuit, stochastic comparator.
Fig. 4. Our proposed stochastic comparator.

Our proposed comparator (Fig. 4) is based on a simple counter. According to the length of the input streams (2n), we choose an n bit counter. Now, by starting from the first bit of A stream, if it is 1, we increase the counter by one unit, and continue this process for all bits of A. After processing A, we start with the first bit of the second stream B. If this bit is 1 and the counter is not empty, we decrease the counter by one unit, and continue for all B bits. By this simple two stage process, we would know that which one of two input streams has more 1s and so is greater than the other one. In the last step, after processing A and B streams, if the counter was showing zero value, the output of the proposed comparator will be zero, otherwise the output will be a stream with all bits 1.

Advantages:

- Low area and power

Software implementation:

- Modelsim
- Xilinx ISE