A NOVEL FPGA ARCHITECTURAL IMPLEMENTATION OF PIPELINED THINNING ALGORITHM

Pei-Yung Hsiao, Chun-Ho Hua and *Chien-Chen Lin
pyhsiao@mail.cgu.edu.tw
Dept. of Electronic Engineering, Chang Gung Univ.,
259 Wen-Hwa 1st Rd., Kwei-Shan, Tao Yuan, Taiwan, 333, ROC
*Dept. of Industrial Education, National Taiwan Normal Univ.,
162 sec.1, Ho Ping E. Rd., Taipei, Taiwan, 106, ROC

ABSTRACT

Thinning is a very important operation in the image preprocessing stage of pattern recognition. This investigation presents an improved thinning algorithm and its FPGA architectural implementation. The improved algorithm based on parallel pipelined design is adapted and formulated such that it is suitable to computing architecture implementation. The FPGA-based architecture extends the applicability of this algorithm in the area of real time image processing. Using the proposed Modification Unit Array, this work performs thinning operation within 0.07 sec at 40MHz for a 512 x 512 picture.

1. INTRODUCTION

Many problems, such as medical image processing, fingerprint recognition [1-2] and vehicle vision system deal with large volumes of input image data. Usually, this data may contain more information than is necessary for pattern recognition or identification purpose. It has been known that reducing the input image data by thinning, edge detection or other image preprocessing is beneficial for certain pattern recognition applications.

Although numerous works have examined how to improve the performance and speed of sequential and parallel thinning algorithms, various VLSI design [3-5] are proposed for implementing real time image processing. However, the VLSI design for thinning operation remains elusive. To overcome this problem, this investigation proposes an improved thinning algorithm and its pipelined architecture being capable of real-time efficiency in comparison with pure software implementation.

The pipelined thinning architecture is based on the well-known frame technique, and uses a 3x3 picture frame to scan the image, repeatedly. The same operation is repeated for each pixel, with the result being a simple function of the input pixel and neighborhood values [6][7]. This work modified and implemented a thinning strategy [8], to provide an external pipelined and internal parallel thinning algorithm, being capable of calculating the skeleton of a raw 512 x 512 binary image with fast clock rate. Region pixels are assumed to have value 1 while background pixels have value 0. The method comprises successive passes of two basic steps applied to the contour pixels of the given region. Moreover, the selected original algorithm is superior to other existing parallel thinning algorithms because it is simple to be hardwarized and uses the same resources to modify image data in each iteration. Another notable benefit of the selected algorithm is that it does not require random access to the image pixels unlike many sequential and parallel algorithms, helping to derive an efficient systolic solution.

The rest of this paper is organized as follows: section 2 describes the parallel processing method of the improved thinning algorithm. Section 3 then presents the design of pipelined data flow for the pipelined thinning architecture, while the function units of the pipelined thinning architecture are designed in detail in Section 4. Subsequently, section 5 presents our experimental results. And finally, section 6 concludes this investigation.

2. PARALLELISM OF THINNING ALGORITHM

In this section, we propose a method to improve the Zhang and Suen's thinning algorithm [8]. In Zhang and Suen's original algorithm, region pixels are assumed to have value 1 and background pixels to have value 0. Their algorithm consists of successive passes of two loops applied to the contour pixels of the given region, where a contour pixel is any pixel with value 1 and having at least one 8-neighbor value 0. With reference to the 8-neighborhood definition shown in Figure 1, the first loop flags a contour pixel p for deletion while the following conditions are satisfied:

(a) \(2 \leq \mathcal{N}(P_j) \leq 6\); 
(b) \(\mathcal{S}(P_j) = 1\);
(c) \( P_2 \cdot P_4 \cdot P_6 = 0 \);
(d) \( P_4 \cdot P_6 \cdot P_8 = 0 \).  \( \quad \) (1)

Where \( N(P1) \) is the number of nonzero neighbors of \( P1 \); that is,
\[ N(P1) = P_2 + P_4 + P_6 + P_7 + P_9 \]
and \( S(P1) \) is the number of 0-1 transition in the ordered sequence of \( P2, P3, \cdots, P9 \). In second loop, conditions (a) and (b) remain the same, but conditions (c) and (d) are changed to
(c') \( P_2 \cdot P_4 \cdot P_8 = 0 \);
(d') \( P_2 \cdot P_6 \cdot P_8 = 0 \).  \( \quad \) (2)

After first loop has been applied to all border pixels, those that were flagged are deleted. Then second loop is applied to the resulting data in exactly the same manner as done in the first loop.

![Figure 1: Neighborhood arrangement for the thinning algorithm](image1.png)

In software implementation, the first step in the first loop of Zhang and Suen's thinning algorithm is to fetch memory data and load them to register, as well as to check whether these data conclude the central point as an object or a background pixel. If it is an object pixel, it has to do eight memory fetches and eight load operations in order to process the pixel's 8-neighbor data. After the 3x3 image frame data are ready, we have to accumulate the 8-neighbor data to check if the condition (a) is satisfied. In the second step, we further check the pixel's 8-neighbor data if there exists only one 0-1 transition according to the condition (b). The third step is to check the three pixels \( P2, P4 \) and \( P8 \) if there is at least one is a background pixel of value 0, and similarly, the fourth step is to check the other three pixels \( P4, P6 \) and \( P8 \). If all the four steps are satisfied, the central pixel will be set to 0. In addition, the operations of the second loop are similar to those of the first loop.

In Zhang and Suen's thinning algorithm, it takes four steps in sequence to check whether all the four conditions are satisfied, where each step may take two or more clock cycles. However, as the above four conditions are independent with each other, our hardware implementation for these algorithmic computations can be executed simultaneously. Therefore, the number of execution cycles in our work can be reduced from eight or more clocks to one cycle.

According to the aforementioned improvement, the proposed Modification Unit as will be stated in Section 4 will be in charge of the arithmetic checking of the four conditions. For finding the correct status of \( P1 \), four arithmetic modules are designed to perform the corresponding operations in combinational logics. In Zhang and Suen's thinning algorithm, we need to perform two loops consecutively, where the differences between these two loops are condition (c), (d) and (c'), (d'), respectively. In our work, we perform the (c), (d), (c') and (d') operations simultaneously through the Modification Unit, furthermore, we use multiplexer to select the correct results. The detailed design of the Modification Unit is proposed as shown in Figure 2.

![Figure 2: Modification Unit](image2.png)

### 3. DESIGN OF PIPELINED DATAFLOW

In the original algorithm of Zhang and Suen, the program begins by checking the image pixels from the upper-left to the lower-right corner pixel-by-pixel. For each pixel encountered that is an objective pixel, whose 8-neighbor must be identified and then used to deciding whether the pixel is a contour pixel. Therefore, processing an objective pixel in sequential approach requires one plus eight memory fetching instruction cycles and one plus eight operand loading instruction cycles. Moreover, six memory fetch cycles and six load cycles are wasted during the transition between processing an objective pixel's 8-neighborhood and the subsequent objective pixel's 8-neighborhood. Processing an image size of 512x512 takes 512x512x9 fetch cycles and 512x512x9 load cycles to grab the objective pixel and complete calculations of its 8-neighborhood pixels. For timing complexity reduction, the proposed architecture reuses the six overlapping pixels to process the subsequent pixel.

In order to implement the pipelining and parallel architecture to speed up execution rate, this investigation uses three individual modules of RAM Bank and the Register Set. To avoid the long latency time between two sequential eight image pixels, a RAM Bank is organized as pipeline architecture to increase the throughput. In figure 3, the shadow columns in the RAM Bank indicate the overwrite part by the previous fetch image data from lower part of the RAM Bank. The third RAM input data in the RAM Bank comes from external memory. The pair data of \((x,y)\) in Figure 3 indicates the x-th image line and the y-th image pixel.

The Register Set as shown in Figure 4 is used to contain the current eight image pixels and their 8-neighborhood. Moreover, each register row in the Register Set has 10-bit data and is divided into three
groups, l (1-bit), m (8-bit), and r (1-bit), respectively, as shown in Figure 4. During every execution cycle, 8-bit data are processed simultaneously.

\[(a, 4) \xrightarrow{8} n_7, n_6, n_5, n_4, n_3, n_2, n_1, n_0, n_1, n_0, n_1, n_0, n_1, n_0 \]

\[(a+1, 4) \xrightarrow{8} n_7, n_6, n_5, n_4, n_3, n_2, n_1, n_0, n_2, n_2, n_2, n_2, n_2, n_2, n_2 \]

\[(a+2, 4) \xrightarrow{8} n_7, n_6, n_5, n_4, n_3, n_2, n_1, n_0, n_3, n_2, n_3, n_3, n_3, n_3, n_3, n_3 \]

to register sets

data from Main Memory (a+3, 3)

Figure 3: Dataflow of the RAM Bank.

<table>
<thead>
<tr>
<th>T</th>
<th>R</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>k</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>k</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>k+1, 3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>k+1, 3</td>
</tr>
</tbody>
</table>

Figure 4: The Register Set

4. FUNCTION UNITS

The pipelined thinning architecture shown in Figure 4 comprises Modification Unit Array, RAM Bank, Main Memory, and Register Set. The Modification Unit Array consists of eight Modification Units, each MU performing the arithmetic operation corresponding to a 3x3 mask data. The structure of the Modification Unit Array is shown in Figure 5.

The main memory, RAM Bank, Register Set, and eight Modification Units thus are linked together in a systolic cycle [10]. The Register Set receive their data from the three related RAM Bank. Each register set functions as a queue, with the RAM Bank pushing data to one end of the Register Set and causing the register line to eject equivalent data from the other end.

Figure 5: Block diagram of pipelined architecture

![Block diagram of pipelined architecture](image)

Each MU shown in Figure 6 has its own arithmetic logic (ALU), however, all of the units share the three 10-bit data from the Register Set. In the RAM Bank, there are three RAM modules, each of which has capacity of 512-bit for buffering one horizontal image line.

Considering the data flow of pipelined thinning architecture illustrated in Figure 7, the data flows from the main memory, through the three RAM modules, the Register Set, and the Modification Unit Array, and finally is fed-back to the main memory. The data of the first RAM module in the RAM Bank is sourced from the main memory. Meanwhile, the third RAM module receives data from the second RAM module and sends the data to the Register Set. The processed pixels from the Modification Unit Array are forwarded to the main memory and stored into their original position.

Figure 6: The Modification Unit Array

Figure 7: Dataflow path of pipelined architecture

![Dataflow path of pipelined architecture](image)

This design attempts to achieve a faster thinning architecture, functional pipelining is the key to the impressive performance of this architecture. Before starting to implement pipelining, this work first checked the operations of the improved thinning algorithm as mentioned in Section 2. The execution steps are scheduled in Table 1 and described in detail as follows: The first step is to fill the left 1-bit register set, fetch the data from on-chip RAM, as well as from main memory. The second step then is to fill the register set with the data fetched from on-chip RAM during the first step. The third step is to load main memory data to the on-chip RAM. Subsequently, the fourth step is to fetch the RAM Bank data and store previous cycle data in the main memory. The fifth step is to feed the right 1-bit register set with the data obtained from the RAM Bank. Finally, the last step is to execute the deletion operation and feed the RAM Bank with the required data.

Table 1: One parallel pipelined execution cycle
5. EXPERIMENTAL RESULTS

The presented design has been implemented by the Spartan-II series of Xilinx Logic Cell Array family. The architecture of Xilinx Spartan-II FPGA is a SRAM based symmetrical array. Table 2 summarizes the used resource of Slices and IOBs from Spartan-II XC2S100 FPGA in our design. The delays are estimated from the maximal modular timing delay of the critical path in each functional module. The calculation unit composes of Register Set and Modification Unit Array. The post layout timing data reports a maximum operation frequency of 42.7MHz. The thinning architecture occupies totally 637 Slices which is about 53% of the designate XC2S100 FPGA chip. The processing time for a 512 x 512 image is about 0.07 seconds.

| Slice s | 228 | 98 | 244 | 77 |
| IOB s | 29 | 48 | 41 | 18 |
| Delay | 176 | 54 | 304 | 98 |

Table 2: The FPGA resource and modular timing delays.

Figure 8 shows an original binary fingerprint image, the software thinning output, and the FPGA thinning result from the proposed pipelined thinning architecture in our system function verification platform. The results from Figure 8 also illustrate 100% the same output for software thinning and the proposed FPGA hardware thinning. The comparative results of execution time based on software in C language and hardware of the proposed architecture are listed in Table3.

<table>
<thead>
<tr>
<th>No. of repeated executions</th>
<th>software implementation in C</th>
<th>Proposed hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 times</td>
<td>0.877 sec</td>
<td>0.541 sec</td>
</tr>
<tr>
<td>10 times</td>
<td>1.725 sec</td>
<td>0.684 sec</td>
</tr>
<tr>
<td>15 times</td>
<td>2.577 sec</td>
<td>1.024 sec</td>
</tr>
<tr>
<td>20 times</td>
<td>3.461 sec</td>
<td>1.568 sec</td>
</tr>
<tr>
<td>50 times</td>
<td>8.612 sec</td>
<td>3.412 sec</td>
</tr>
</tbody>
</table>

Table 3: Comparison of execution time.

6. CONCLUSION

This paper presents an improved thinning algorithm and the design and implementation of a pipelined thinning architecture. We modified the Zhang and Suen's thinning algorithm into fully parallel operation through the Modification Unit Array, moreover, the proposed pipelined dataflow successfully promotes the efficiency of execution rate. This thinning architecture achieving acceptable reliability for most realistic binary image patterns, can be used to establish a compact pattern recognition system that runs in real-time. Besides reducing the execution time to less than 70ms, the current testing results also reveal complete match with software thinning results.

With progress in VLSI technologies, reusable intelligence properties (IP) become a main solution for IC design. Therefore, the result of this work provides further evidence that the reusable thinning IP may achieve acceptable digital image thinning ability with reduced execution time and thus helps create SOCs for the applications to real-time imaging systems.

![Figure 8: Comparison between software and hardware thinning output. (a)Original (b)By software (c)By FPGA.](image)

7. REFERENCES


Acknowledgement

This work was supported in part by the National Science Council of the Republic of China under Contract NSC 92-2218-E-182-007.