Employing pipelined thinning architecture for real-time fingerprint verifier

P.Y. Hsiao, X.Z. Chen, C.C. Lin, C.H. Hua and C.C. Chang

Abstract: Thinning is a very important operation in the pre-processing stage of fingerprint recognition. With the availability of fast thinning hardware, real-time image processing applications can be achieved. The authors introduce a detailed hardware architecture design of a thinning processor used in an embedded fingerprint recognition system. The proposed thinning algorithm has a parallel-pipelining structure suited to hardware realisation, which is implemented and verified using FPGA. Equipped with a modification unit array, a designated operating schedule, and an address generator based on systolic counter, this thinning processor is able to perform a thinning operation within 0.07 s at 40 MHz for a 512 × 512 picture, which is at least 40 times faster than software execution. Consequently, the proposed thinning processor was successfully integrated into a real-time fingerprint recognition system.

1 Introduction

Many problems, such as fingerprint recognition [1, 2] and remote video surveillance systems [3], have to deal with large volumes of input image data in real time. Usually, the original input data contains more information than is necessary for object tracking, image pattern recognition or personal identification purposes. Several articles have demonstrated different ways of using thinning [1, 2, 4–10], or other means of image pre-processing methods to reduce the amount of input image data for real-time imaging applications. Indeed, the design architecture by Hsiao *et al.* [10] is a preliminary version of the current article.

Fingerprints, along with other characteristics such as iris, palm prints, speech sounds and DNA, are unique to each individual human being and are often used in person identification. Fingerprint recognition is, by far, most common and has a wide range of applications such as entrance control, attendance management and network authorisation as well as encryption. A full fingerprint recognition processing cycle can be roughly divided into four stages: the image acquisition, image pre-processing, feature encoding and feature matching. More specifically, the image pre-processing stage involves transformation, segmentation, binarisation and thinning.

Thinning, or skeletonisation, is a well-known image pre-processing technique used to extract distinctive or significant features from digital patterns. Usually, this is done by iteratively removing black pixels, such that an object without holes that will be eroded to a minimally connected stroke, or an object with holes that will be eroded to a minimally connected ring, until the skeleton, or the stick diagram of the original image is retrieved by preserving its connectivity.

Recently, because of the rapid progress in VLSI design, numerous researches have demonstrated different ways on how to improve the performance and speed of sequential and parallel thinning algorithms. Several enhanced thinning algorithms [4–6] were proposed for real-time image processing. However, the VLSI design [6, 8, 11] issue for thinning operation based on the wide usage and well-known thinning algorithm [9] still remains elusive. For example, the pipelined design [6] was based on a different algorithm from ours. The hardware used by Ranganathan and Doreswamy [6] was very large in comparison with ours. For a 512 × 512 image, the number of processing units (PEs) needed by Ranganathan and Doreswamy was 512 [6]; however, the proposed design requires eight PEs for any size of the processed image. To overcome this problem, and meet the real-time constraint in image applications, this study proposes a faster thinning algorithm and a novel pipelined architecture to realise this algorithm [9].

The pipelined thinning architecture is based on the well-known frame technique and repeatedly uses a 3×3 mask to scan the image. The same operation is performed on each pixel, and the result is, henceforth, a simple function of input pixel and its neighbourhood values [12, 13]. In this study, we modify Zhang's thinning algorithm [9] and use FPGA hardware to implement this externally pipelined and internally parallel thinning algorithm to achieve the skeletonisation of a 512 × 512 binary raw image at high clock rate. Region pixels are assumed to have a value of 1, whereas background pixels have a value of 0. The selected algorithm is superior to other existing parallel thinning algorithms, because it is capable of processing eight pixels at once. It also has the merits of simplicity and regularity to hardware realisation and is able to reuse the same resource to process the image data in different iterations. Another notable benefit is that, unlike many sequential and parallel algorithms proposed in the literature, the selected algorithm does not require random access to image pixel, which helps us to derive an efficient pipelined solution.

IEE Proceedings online no. 20050200

doi:10.1049/ip-cdt:20050200

Paper first received 26th November 2005 and in final revised form 10th June 2006

P.Y. Hsiao is with the Department of Electronic Engineering, Chang Gung University, 259 Wen-Haw 1st Road, Kwei-Shan Tao-Yuan 33, Taiwan, Republic of China

X.Z. Chen is with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

C.C. Lin, C.H. Hua, and C.C. Chang are with the Fingerprint Technology Research Group, Aetex Biometric Corp., Taiwan, Republic of China

E-mail: pyhsiao@mail.cgu.edu.tw

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

Fig. 1 *Neighbourhood arrangement for the thinning algorithm*

2 Parallelism of modified thinning algorithm

To fully promote the performance of the Zhang and Suen's [9] thinning algorithm and migrate the algorithm to a successful hardware realisation, we analysed the original algorithm for optimising hardware parallelisation, and completed its pipelined architectural design.

2.1 Review of Zhang and Suen's thinning algorithm

In Zhang and Suen's original algorithm, the object pixels were defined to have a value of 1 and the background pixels have a value of 0. The algorithm consists of successive passes of two loops to the centre pixel of a given 3×3 region. Consulting the 8-neighbour pixel definition as shown in Fig. 1, the first loop flags a contour pixel P_1 for deletion while the following conditions were satisfied

$$\begin{aligned} &\text{(a)}\ \ 2 \le N(P_1) \le 6 \\ &\text{(b)}\ \ S(P_1) = 1 \\ &\text{(c)}\ \ P_2 \cdot P_4 \cdot P_6 = 0 \\ &\text{(d)}\ \ P_4 \cdot P_6 \cdot P_8 = 0 \end{aligned} \tag{1}$$

where $N(P_1)$ is the number of non-zero neighbours of P_1; that is, $N(P_1) = P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9$ and $S(P_1)$, the number of $0-1$ transitions in the ordered sequence of $P_2, P_3, \ldots, P_9$. In the second loop, conditions (a) and (b) remain the same, whereas conditions (c) and (d) were changed to

$$\begin{aligned} &\text{(c}')\ \ P_2 \cdot P_4 \cdot P_8 = 0 \\ &\text{(d}')\ \ P_2 \cdot P_6 \cdot P_8 = 0 \end{aligned} \tag{2}$$

After applying the first loop operations to all object pixels, those pixels that were flagged for removal are deleted immediately. The second loop operations are then applied to the object pixels of the resulting data in a similar manner. Under repeated iterations of both loops, the final thinned skeleton can be obtained.

For software implementation, before applying the condition checks to all encountered pixel, we need to make sure that the current processing pixel is an object pixel. If it is an object pixel, in the first step we need to perform eight memory fetches and eight load operations to retrieve data for the current 3×3 region, and accumulate the 8-neighbour-pixel values to check whether condition (a) is satisfied. In the second step, we need to further check whether there is only one $0-1$ transition in the 8-neighbour pixel data. The third step is to check if any of the three pixels P_2, P_4 and P_8 is a background pixel, and similarly, the fourth step is to check the other three pixels P_4, P_6 and P_8 for the existence of background pixel. If all the four conditions in the first loop are satisfied, the centre pixel of the current 3×3 region will be flagged for removal. When all pixels are processed, those flagged pixels would be removed and enter the second loop operations, which are performed similar to those in the first loop.

In the original algorithm of Zhang and Suen, the operation starts with checking of the image frame from upper-left to lower-right corner, pixel-by-pixel. For each encountered object pixels, its 8-neighbour pixel must be identified and then used to decide whether the centre pixel is a contour pixel. Therefore processing of a centre pixel in sequential approach requires one plus eight memory fetch cycles and one plus eight operand load cycles. Moreover, six memory fetch cycles and six load cycles are wasted during the transition between processing the current 3×3 region and the next 3×3 region. Therefore complete thinning operation for an image size of 512×512 will take at most $512 \times 512 \times 9$ fetch cycles and $512 \times 512 \times 9$ load cycles.

2.2 Design of our modified thinning algorithm

From the analysis mentioned in the previous subsection, when processing an object pixel, a lot of clock cycles are wasted during the evaluation stage. Yet, even more clock cycles are wasted while fetching its 8-neighbour pixel. It is also quite easy to observe that some of the currently fetched data might have already been fetched by the previous fetch operation, and are ready to be reused by the current evaluation stage, which leads to our improved thinning algorithm.

In order to follow the rule of stable execution cycle in a von Neumann machine, and develop a pipelined dataflow machine to speed up execution rate, we make the following assumptions.

Assume that in our machine, there are:

(a) Three individual on-chip RAM modules grouped as a RAM bank.
(b) Three register sets H, M, L, used to hold the subsequent data of 3×3 image regions.
(c) Each register set has 10 bits and is divided into three subgroups l (1-bit), m (8-bit), and r (1-bit) as shown in Fig. 2. Both subgroups l (1-bit) and m (8-bit) are designed as temporary registers and used to produce the first bit and the last bit of the subgroup register m (8-bit), respectively. In each execution cycle, the register set is allowed to compute and shift out 8-bits in parallel.

Accordingly, we modify the thinning algorithm as follows:

(a) Shift the least significant bit of the m subgroups to l subgroups, and store previous processed data to the temporal register. Also fetch the data from the main memory to the RAM bank.

	l	m							r
H	a,k,0	a,k+1,7	a,k+1,6	a,k+1,5	a,k+1,4	a,k+1,3	a,k+1,2	a,k+1,1	a,k+1,0	a,k+2,7
M	a+1,k,0	a+1,k+1,7	a+1,k+1,6	a+1,k+1,5	a+1,k+1,4	a+1,k+1,3	a+1,k+1,2	a+1,k+1,1	a+1,k+1,0	a+1,k+2,7
L	a+2,k,0	a+2,k+1,7	a+2,k+1,6	a+2,k+1,5	a+2,k+1,4	a+2,k+1,3	a+2,k+1,2	a+2,k+1,1	a+2,k+1,0	a+2,k+2,7

Fig. 2 *Register sets*

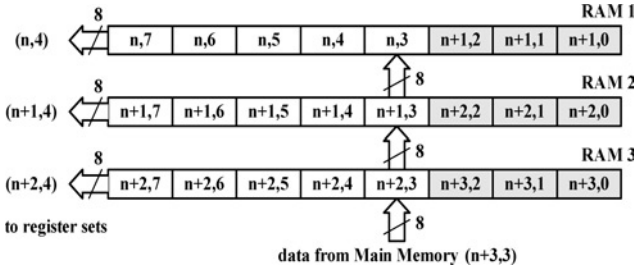

Fig. 3 *Dataflow of the RAM bank*

(b) Feed the data fetched from the three on-chip RAM modules to the three *m* subgroups.
(c) Feed the third RAM module with the main memory data fetched at step (a), while the first and second RAM modules with the data stored on the second and third RAM modules.
(d) While fetching data to the RAM bank, store the processed result in previous execution cycle back to main memory.
(e) Feed the *r* subgroups with the data fetched from the RAM bank directly.
(f) Execute parallel operations in the modification unit array.

The dataflow inside the three RAM modules as well as its relation with the main memory and the three register sets are shown in Fig. 3. The dataflow block diagram based on our modification is shown in Fig. 4. In our modified thinning algorithm, we combine the checking step and the deletion step into one single step. It only takes six clock cycles to complete one parallel pipelined execution cycle.

2.3 Hardware realisation of the proposed pipelined dataflow

The three register sets shown in Fig. 2 are used to hold the current eight image pixels and their eight neighbourhoods. Moreover, each register set has 10-bit data and is divided into three subgroups, *l* (1-bit), *m* (8-bit) and *r* (1-bit), respectively. During every execution cycle, the 8-bit data are processed simultaneously.

In order to implement the pipelining and parallel architecture, to avoid the long latency of the two sequences of eight image pixels, the RAM bank is organised as a pipeline operation to increase the throughput. In Fig. 3, the shaded columns in the RAM bank indicate the overwritten part by the previously fetched image data from the lower part of the RAM bank. The input data of the third RAM module in the RAM bank comes from external memory. The data pair (x, y) in Fig. 3 indicates the xth image line and the yth image pixel.

In Zhang and Suen's thinning algorithm, it takes four steps in sequence to check whether all the four conditions

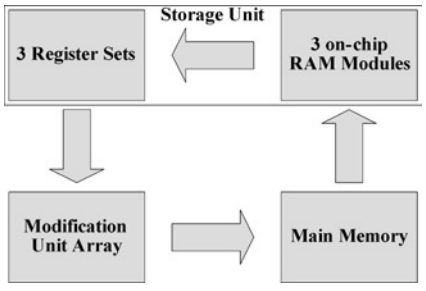

Fig. 4 *Dataflow block diagram*

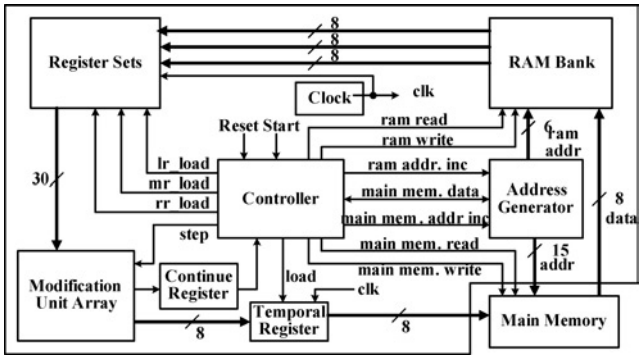

Fig. 5 *Block diagram of the proposed pipelined architecture*

are satisfied, whereas each step may take two or more clock cycles. However, because of the fact that the four check conditions are independent of each other, we propose four block modules in the modification unit. Each module works for one check condition so that our design can execute these four checking steps simultaneously. Therefore the number of execution cycles in our work can be reduced from eight or more clocks to one clock. According to the aforementioned improvement, the proposed modification unit array will be in charge of the arithmetic checking operations for the four conditions. More details about the modification unit array are discussed in the following section.

3 Proposed function units

In this section, we present the proposed modularised hardware scheme for our faster thinning algorithm. The overall proposed pipelined thinning hardware architecture is shown in Fig. 5, which comprises a modification unit array, a storage unit, a controller and an address generator. The continue register produces an enable flag to the controller and the temporal register is used as a buffer between the modification unit array and the main memory. These units are linked together in a pipelined structure. The systolic counter [14] used in the address generator is better than the conventional counter or dynamic counter in terms of delay time, signal sharpness and cost-effectiveness. From Fig. 5, it is not hard to find that the register sets receive their data from the RAM bank. Each register set acts as a queue, while the RAM bank pushes the data to one end of the register set and causes the register line to eject equivalent data from the other end.

In the following subsections, we will introduce each of these function units. Moreover, at the end of this section, we will describe the dataflow and the operating schedule that makes all these units work together in a pipelined manner.

3.1 Storage units

The storage unit comprises three register sets and the on-chip RAM bank organised as three independent memory modules. Because of the fact that the default image size is 512×512, each of the RAM modules has the capability of holding 512 bits for buffering one horizontal image line. We divide each memory module into 64 columns with each column filled with eight adjacent image pixels, as shown in Fig. 6. These column data are mutually exclusive. As mentioned earlier, each register set is divided into three subgroups *l*, *m* and *r*. The detailed scheme of the register sets and their dataflow is illustrated in Fig. 7.

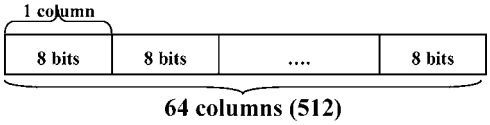

Fig. 6 *RAM module format*

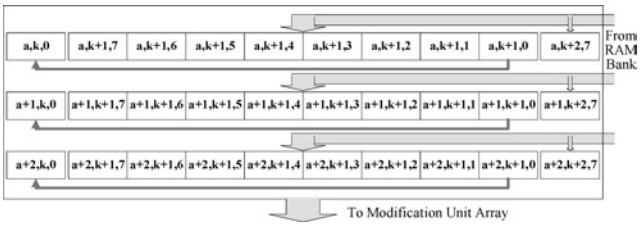

Fig. 7 *Detailed register sets scheme*

3.2 Modification unit array

As far as we know, the four condition checks of Zhang and Suen's thinning algorithm are independent of each other. The modification unit and its operations have been proposed in accordance with the four condition checks, as shown in Fig. 8. As we have to process eight adjacent pixels in parallel, the sequential procedural characteristics used to check whether the current pixel is an object pixel is removed from our design. In order to give the correct status of the centre pixel P_1 of the current 3×3 region, we use the proposed modification unit to perform the corresponding operations in combinational logic circuits. In the original thinning algorithm, we need to perform two loops consecutively, and the different operations employed in these two loops are conditions (c), (d) and (c′), (d′), separately. In our work, we perform conditions (c), (d), and (c′), (d′) simultaneously using the modification unit and apply a multiplexer to select the correct outcome. The signal 'Step' shown in Fig. 8 is designed for selecting conditions (c) and (d), whereas it is low or otherwise for conditions (c′) and (d′).

The modification unit array shown in Fig. 9 consists of eight single modification units. Each modification unit performs arithmetic calculation on a 3×3 region data, and has its own arithmetic logic unit. However, all units share the same three 10-bit input data from the register sets.

3.3 Dataflow

Consider the data flow of our pipelined thinning architecture as illustrated in Fig. 10. The image data flow from the main

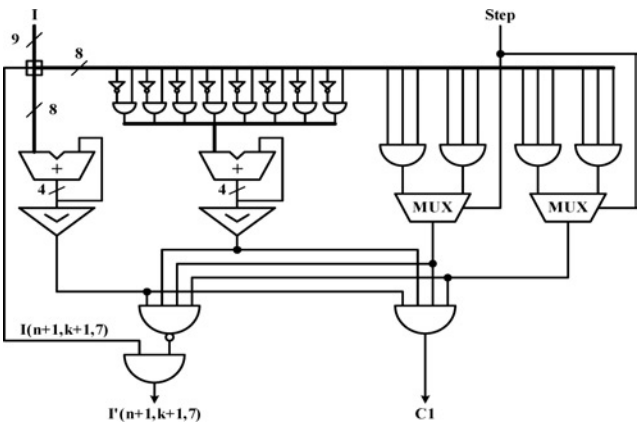

Fig. 8 *Single modification unit*

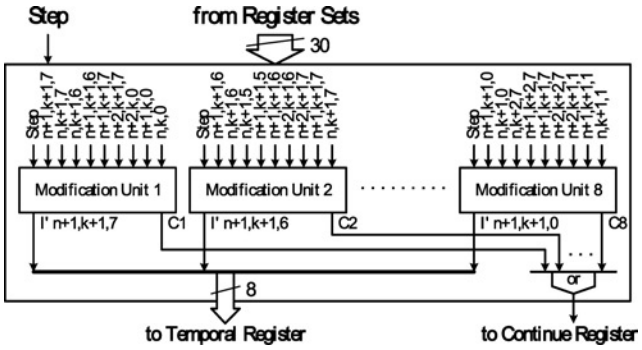

Fig. 9 *Modification unit array*

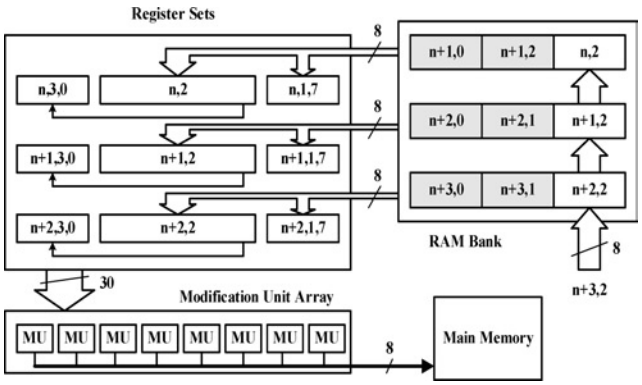

Fig. 10 *Dataflow path of our pipelined architecture*

memory, through the three RAM modules, the three register sets and then the modification unit array. Finally, they are fed back to the main memory.

In the RAM bank, the first RAM module receives data stored in the second RAM module, and the second RAM module receives data stored in the third RAM module. The third RAM module data are sourced from the main memory. Meanwhile, each register set, which works as a queue, gets input data from the corresponding RAM module and ejects output data to the modification unit array, in which we have the four condition checks to be executed in parallel. The processed outcome pixels from the modification unit array are stored back to their original location in the main memory.

3.4 Address generator and operating schedule

In order to generate the required address to fetch the data from the main memory, it is necessary to incorporate an address generator in our design. The conventional ripple counter failed to satisfy the real-time requirement owing to the fact that a long latency exists for the signal to travel from LSB to HSB. We adopted the long binary counter proposed by Pekmestzi and Thanasouras [14]. This counter is based on systolic frequency, which has a pipelined architecture differing from the conventional

Table 1: One parallel pipelined execution cycle

Step	1	2	3	4	5	6
Register sets	Load	Load			Load	Execute
RAM bank	Fetch		Load	Fetch		
Main memory	Fetch	Fetch		Store	Store	

Table 2: Internal execution schedule

Time	T	T + 1	T + 2	T + 3	T + 4	T + 5
Register sets	L(l)	L(m)			L(r)	E
RAM bank	F(n, k)		L(k)	F(n, k + 1)		
Main memory	F(n + 2, k)	F(n + 2, k)		S(n, k − 1)	S(n, k − 1)	
	T + 6	T + 7	T + 8	T + 9	T + 10	T + 11
Register sets	L(l)	L(m)			L(r)	E
RAM bank	F(n, k + 1)		L(k)	F(n, k + 2)		
Main memory	F(n + 2, k + 1)	F(n + 2, k + 1)		S(n, k)	S(n, k)	

Register sets L(l): Load data to left 1-bit register
L(m): Load data to middle 8-bit register
L(r): Load data to right 1-bit register
E: Execute deletion
RAM Bank F(n, k): Fetch the *k*th column data
L(k): Load data to *k*th column
Main memory F(n, k): Fetch the *k*th column data in the *n*th image line
S(n, k): Store the *k*th column in the *n*th image line

Table 3: External pipelined execution schedule

Time	T	T + 1	T + 2	T + 3	T + 4	T + 5
Column k − 1				S(n, k − 1)	S(n, k − 1)	
Column k	F(n + 2, k)	F(n + 2, k)				
Column k + 1						
	T + 6	T + 7	T + 8	T + 9	T + 10	T + 11
Column k − 1						
Column k				S(n, k)	S(n, k)	
Column k + 1	F(n + 2, k + 1)	F(n + 2, k + 1)				

F(n, k): Fetch the *k*th column data of the *n*th image line in the main memory
S(n, k): Restore data to the *k*th column of the *n*th image line in the main memory

binary counter. The main advantages of using such a counter are: (i) simple iterative circuit; (ii) very fast operation because of the local interconnections, the minimum propagation delay time for the count enable signal and the low level fan-in requirements; and (iii) operation frequency is independent of the counter size.

For achieving a high-speed thinning architecture, the proposed functional pipelining is a key to obtain the impressive performance. Before starting to schedule the pipelining, we have to first review the operations of the improved thinning algorithm as mentioned in Section 2. The six clock steps for one pipelined execution cycle of our proposed thinning algorithm are listed in Table 1. In the first clock step, the loaded data are to the left 1-bit register in each register set and the data are fetched from the RAM bank and from the main memory in parallel. In the second step, the data fetched from the RAM bank in the first clock step are loaded to the middle 8-bit register in each register set. The third step is to load main memory data to the RAM bank. 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 in each register set with the data obtained from the RAM bank. Finally, the last step is to execute the deleting operation and feed the RAM bank with the required data.

A pipelined scheme is not allowed to perform, fetch and store operations on the same memory at the same time. Although the main memory and the RAM bank have distinct input and output data buses, there is only one address bus existing for read and write. Thus, our scheme normally is able to fetch data from the main memory and store values to the third RAM module simultaneously. The internal parallel execution schedule is shown in Table 2, and the external pipelined schedule is also provided in Table 3. Note that the internal execution schedule is based on a conventional sequential method with shorter steps in which we use three on-chip RAM modules to simplify the fetch operations.

Generally, in the proposed pipelined operating schedule, the operator fetches instruction for the next execution cycle before the previous execution cycle. The memory fetch instruction is replaced by the operand fetch operation because of being in accordance with the data-flow design. Moreover, the main memory operand fetch operation will not be processed for the successive execution cycle. It should be fetched for the next two image lines.

4 Experimental results

The presented pipelined thinning architecture for embedded real-time fingerprint recognition has been implemented in VHDL using the Xilinx Spartan-II series Logic Cell Array family. The architecture of Xilinx Spartan-II FPGA is an SRAM-based symmetrical array. Table 4 summarises the slices and IOBs hardware utilisation in our design. The delays are estimated from the maximal modular timing delay of the critical path in each functional module. The calculation unit is composed of register sets and the

Table 4: Modular FPGA resource and timing delay

	Controller	Interface	Calculation unit	Systolic counter
Slices	258	58	244	77
IOBs	29	48	41	18
Four input LUTs	176	54	304	98
Delay (critical path) (ns)	18.1	14.8	22.1	23.4

Fig. 11 *Comparison between software and hardware thinning output*

a Original fingerprint pattern
b Result by software
c Result by FPGA

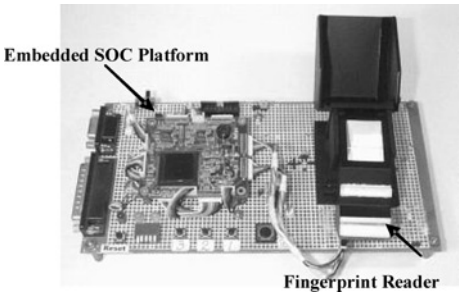

Fig. 12 *Photographic view of the fingerprint recognition system*

modification unit array. The post-layout timing simulation reports a maximum operation frequency of 42.7 MHz and the operation frequency working on the FPGA is 40 MHz. The thinning architecture occupies a total of 637 slices, which is about 53% utilisation of the XC2S100 FPGA chip. The processing time for a 512 × 512 image is about 0.07 s.

Fig. 11 shows the input binary image of a fingerprint pattern, the software thinning output image and the FPGA thinning outcome from our fingerprint verification platform shown in Fig. 12. The results from Fig. 11 illustrate the 100% consistency of our proposed FPGA prototyping with respect to our improved thinning algorithm. The comparative results of execution time based on software in C language and FPGA hardware are listed in Table 5.

Table 5: Comparison of execution time

No. of repeated software executions	Software implementation in C (s)	Proposed hardware in one time execution (s)
5 times	0.877	0.341
10 times	1.725	0.684
15 times	2.577	1.024
20 times	3.462	1.366
50 times	8.612	3.412

Fingerprint image size: 512 × 512 pixels
Platform for software implementation: Pentium-III on 800 MHz
Proposed hardware architecture: operating at 40 MHz

In Table 5, the proposed hardware architecture based on our improved algorithm performs thinning operation at clock rate of 20 times less than the software clock rate. The overall outcome reveals that the hardware is able to achieve thinning operation at least 40 times faster than software execution.

This design has been successfully integrated into our fingerprint recognition system. The system contains a fingerprint reader, and an embedded SOC platform that provides a real-time function of fingerprint verification and the communication modules for system integration. We also developed a monitor software executed in host PC. Fig. 12 shows the prototype photo view of our fingerprint recognition system.

5 Conclusion

This investigation presented an improved thinning algorithm and its real-time FPGA architectural design and implementation. We modified Zhang and Suen's thinning algorithm into fully parallel operations through the proposed modification unit array. Moreover, the proposed pipelined dataflow successfully promoted the efficiency of execution clock rate. This thinning architecture achieved good performance for realistic binary fingerprint patterns, which can be used to establish a real-time compact fingerprint verifier for widespread applications. The experimental results revealed that the hardware scheme completely matched with the software-thinning algorithm, and the hardware operation was able to produce the same results at least 40 times faster than that from software execution. The proposed design has also been realised as an embedded fingerprint recognition system, which provides real-time fingerprint capturing and verification capabilities.

6 Acknowledgments

This work was supported in part by National Science Council, Taiwan, ROC, under grant NSC93-2215-E-182-005 and NSC94-2215-E-182-010.

7 References

1 Hsiao, P.Y., and Hsu, S.F.: 'Method for extracting high-level features for fingerprint recognition'. US Patent 5915035, June 1999
2 Hsu, S.F., Shew, P.W., and Hsiao, P.Y.: 'Method for automatic fingerprint classification'. US Patent 5995642, November 1999
3 Foresti, G.L.: 'Object recognition and tracking for remote video surveillance', *IEEE Trans. Circuits Syst. Video Technol.*, 1999, **9**, (7), pp. 1045–1062
4 Kwon, J.S., Gi, J.W., and Kang, E.K.: 'An enhanced thinning algorithm using parallel processing'. Int. Conf. on Image Processing, 2001, vol. 3, pp. 752–755
5 Espinosa-Duro, V.: 'Mathematical morphology approaches for fingerprint thinning'. 36th Annual 2002 Int. Carnahan Conf. on Security Technology, 2002, pp. 43–45
6 Ranganathan, N., and Doreswamy, K.B.: 'A systolic algorithm and architecture for image thinning'. Fifth Great Lakes Symp. on VLSI: Design Automation of High Performance VLSI Systems, 1995, pp. 138–143
7 Petrosino, A., and Salvi, G.: 'A two-subcycle thinning algorithm and its parallel implementation on SIMD machines', *IEEE Trans. Image Process.*, 2000, **9**, (2), pp. 277–283

8 Majumdar, B., Ramakrishna, V.V., Dey, P.S., and Majumdar, A.K.: 'Design of ASIC chip for skeletonization of gray level digital images', *VLSI Des.*, 1996, **4**, (1), pp. 83–90
9 Zhang, T.Y., and Suen, C.Y.: 'A fast thinning algorithm for thinning digital patterns', *Commun. ACM*, 1984, **27**, (3), pp. 236–239
10 Hsiao, P.Y., Hua, C.H., and Lin, C.C.: 'A novel FPGA architectural implementation of pipelined thinning algorithm'. IEEE Int. Symp. on Circuits and Systems, Vancouver, Canada, May 2004, pp. 593–596
11 Lai, K.L.: 'A K-bit parallel thinning processor for optical character recognition'. Master thesis, National Taiwan University, 1995
12 Kittler, J., and Duff, M.: 'Image processing system architecture' (Research Studies Press, Great Britain, 1985)
13 Onoe, M., Preston, K., and Rosenfeld, A.: 'Real-time/parallel computing' (Plenum Press, NY, 1981)
14 Pekmestzi, K.Z., and Thanasouras, N.: 'Systolic frequency divider/counters', *IEEE Trans. Circuits Syst. II*, 1994, **10**, (11), pp. 775–776