Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 23025, Pages 1–12
DOI 10.1155/ES/2006/23025

A Reconfigurable FPGA System for Parallel Independent Component Analysis

Hongtao Du and Hairong Qi

Electrical and Computer Engineering Department, The University of Tennessee, Knoxville, TN 37996-2100, USA

Received 13 December 2005; Revised 12 September 2006; Accepted 15 September 2006

Recommended for Publication by Miriam Leeser

A run-time reconfigurable field programmable gate array (FPGA) system is presented for the implementation of the parallel independent component analysis (ICA) algorithm. In this work, we investigate design challenges caused by the capacity constraints of single FPGA. Using the reconfigurability of FPGA, we show how to manipulate the FPGA-based system and execute processes for the parallel ICA (pICA) algorithm. During the implementation procedure, pICA is first partitioned into three temporally independent function blocks, each of which is synthesized by using several ICA-related reconfigurable components (RCs) that are developed for reuse and retargeting purposes. All blocks are then integrated into a design and development environment for performing tasks such as FPGA optimization, placement, and routing. With partitioning and reconfiguration, the proposed reconfigurable FPGA system overcomes the capacity constraints for the pICA implementation on embedded systems. We demonstrate the effectiveness of this implementation on real images with large throughput for dimensionality reduction in hyperspectral image (HSI) analysis.

1. INTRODUCTION

In recent years, independent component analysis (ICA) has played an important role in a variety of signal and image processing applications such as blind source separation (BSS) [1], recognition [2], and hyperspectral image (HSI) analysis [3]. In these applications, the observed signals are generally the linear combinations of the source signals. For example, in the cocktail party problem, the acoustic signal captured from any microphone is a mixture of individual speakers (source signal) speaking at the same time; in the case of hyperspectral image analysis, since each pixel in the hyperspectral image could cover hundreds of square feet area that contains many different materials, unmixing the hyperspectral image (the observed signal or mixed signal) to the pure materials (source signals) is a critical step before any other processing algorithms can be practically applied. ICA is a very effective technique for unsupervised source signal estimations, given only the observations of mixed signals. It searches for a linear or nonlinear transformation to minimize the higher-order statistical dependence between the source signals [4, 5]. Although powerful, ICA is very time consuming in software implementations due to the computation complexities and the slow convergence rate, especially for high-volume or dimensional data set. The field programmable gate arrays (FPGAs) implementation provides a potentially faster and real-time alternative.

Advances in very large-scale integrated circuit (VLSI) technology have allowed designers to implement some complex ICA algorithms on analog CMOS and analog-digital mixed signal VLSI, digital application-specific integrated circuits (ASICs), and FPGAs with millions of transistors. Designs that are developed using analog or analog-digital mixed technologies utilize the silicon in the most efficient manner. For example, analog CMOS chips have been designed to implement a simple ICA-based blind separation of mixed speech signals [6] and infomax theory-based ICA algorithm [7]. Celik et al. [8] used a mixed-signal adaptive parallel VLSI architecture to implement the Herault-Jutten (H-J) ICA algorithm. The coefficients in the unmixing matrix were stored in digital cells of the architecture, which was fabricated on a 3 mm $\times$ 3 mm chip using a 0.5 μm CMOS technology. But the 3×3 chip could only unmix three independent components. The neuromorphic auto-adaptive systems project conducted at Johns Hopkins University [9] used the ICA VLSI processor as a front end of the system integration. The processor separates the mixed analog acoustic inputs and feeds the digital output to Xilinx FPGA for classification purpose.

Although these works could offer possible solutions to some ICA applications, the high cost of the analog or mixed-signal development systems (\$150 K) and the long turnaround period (8–10 weeks) make them suboptimal for most ICA designs [10]. As another branch of VLSI implementation, the digital semicustom group that consists of user programmable FPGAs and non-programmable ASICs presents low-cost substitute solutions.

The general-purpose FPGAs are the best selections for fast design implementations and allow end users to modify and configure their designs for multiple times. Lim et al. [11], respectively, implemented two small 7-neuron independent component neural network (ICNN) prototypes on Xilinx Virtex XCV 812E which contains 0.25 million logic gates. The prototypes are based on mutual information maximization and output divergence minimization. Nordin et al. [12] proposed a pipelined ICA architecture for potential FPGA implementation. Since each block in the 4-stage pipelined FPGA array did not have data dependency with others, all blocks could be implemented and executed in parallel. Sattar and Charayaphan [13] implemented an ICA-based BSS algorithm on Xilinx Virtex E, which contains 0.6 million logic gates. Due to the capacity limit, the maximum iteration number was prelimited to 50 and the buffer size to 2,500 samples. Wei and Charoensak [14] implemented a noniterative algebra ICA algorithm [15] that requires neither iteration nor assumption on Xilinx Virtex E in order to speed up the motion detection operation in image sequences. Although the design only used 90 200 of the 600 000 logic gates, the system could support the unmixing of only two independent components. We see that all these FPGA-based implementations of ICA algorithms are constrained by the limited FPGA resources; hence, they have to either reduce the algorithm complexity or restrict the number of derived independent components.

In order to implement a complex algorithm in VLSI, one common solution is to sacrifice the processing time so as to meet the resource constraints. Although ASICs can obtain better speedup than FPGAs, they are fixed in design and are nonprogrammable. On the other hand, FPGAs have lower circuit density and higher circuit delay which brings capacity limitation to complex algorithm implementations. However, as standard programmable products, FPGAs offer characteristics of reconfigurability and reusable life cycle that allow end users to modify and configure designs for multiple times. The idea of our reconfigurable FPGA system is to use the reconfigurability of FPGA to break its capacity limitation. The proposed approach compromises the processing speed to satisfy the hardware resource constraints so as to provide appropriate solutions to embedded system implementations. In this paper, we first develop and synthesize a parallel ICA (pICA) algorithm based on FastICA [1]. We then investigate design challenges due to the capacity constraints of single FPGA such as Xilinx VIRTEX V1000E. In order to overcome the capacity limitation problem, we present the reconfigurable FPGA system that partitions the whole pICA process into several subprocesses. By utilizing just one FPGA and its reconfigurability feature, the subprocesses can be alternatively configured then executed at run-time.

The rest of this paper is organized as follows. Section 2 briefly describes the ICA, FastICA, and pICA algorithms. Section 3 elaborates the three ICA-related reconfigurable components (RCs) and the corresponding synthesis procedure. Section 4 identifies and investigates design challenges due to the capacity constraints of single FPGA, then presents the reconfigurable FPGA system. Section 5 validates the proposed implementation using a case study for pICA-based dimensionality reduction in HSI analysis. Finally, Section 6 concludes this paper and discusses future work.

2. THE ICA AND PARALLEL ICA ALGORITHMS

Before discussing the hardware implementation, in this section, we first describe the ICA [4], the FastICA [1], and the pICA algorithms. FastICA is one of the fastest ICA software implementations so far, while pICA further speeds up FastICA using single program multiple data (SPMD) parallelism.

2.1. ICA

Let $\mathbf{s}_1, \ldots, \mathbf{s}_m$ be m source signals that are statistically independent and no more than one signal is Gaussian distributed. The ICA unmixing model unmixes the n observed signals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ by an $m \times n$ unmixing matrix or weight matrix $\mathbf{W}$ to the source signals

$$\mathbf{S} = \mathbf{W}\mathbf{X}, \tag{1}$$

where

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_m^T \end{bmatrix}, \qquad \mathbf{w}_i = \begin{bmatrix} w_{i1} \\ \vdots \\ w_{in} \end{bmatrix}. \tag{2}$$

The main work of ICA is to recover the source signal $\mathbf{S}$ from the observation $\mathbf{X}$ by estimating the weight matrix $\mathbf{W}$. Since the source signals $\mathbf{s}_i$ are desired to contain the least Gaussian components, a measure of nongaussianity is the key to estimate the weight matrix, and correspondingly, the independent components. The classical measure of nongaussianity is kurtosis, which is the fourth-order statistics measuring the flatness of the distribution and has zero value for the Gaussian distributions [16]. However, kurtosis is sensitive to outliers. The negentropy is then used as a measure of nongaussianity since Gaussian variable has the largest entropy among all random variables of equal variance [16]. Because it is difficult to calculate negentropy, an approximation is usually given.

2.2. The FastICA algorithm

In order to find $\mathbf{W}$ that maximizes the objective function, Hyvärinen and Oja [1] developed the FastICA algorithm that

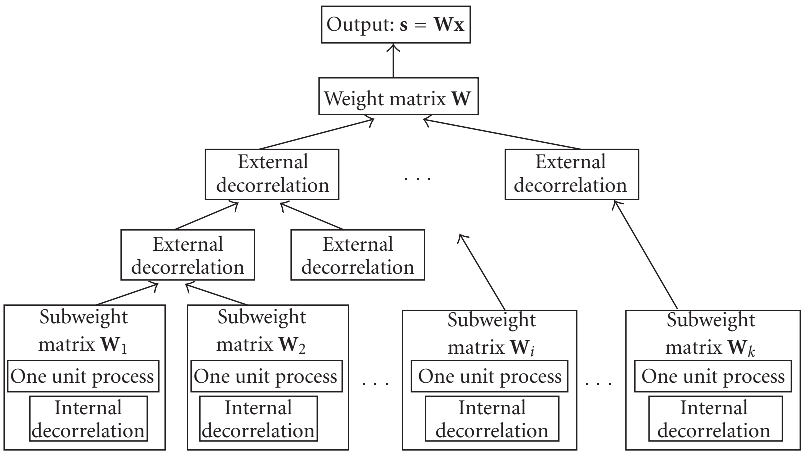


FIGURE 1: Structure of the pICA algorithm.

involves the processes of *one unit estimation* and *decorrelation*. The one unit process estimates the weight vectors $\mathbf{w}_i$ using (3),

$$\mathbf{w}_i^+ = E\{\mathbf{X}g(\mathbf{w}_i^T\mathbf{X})\} - E\{g'(\mathbf{w}_i^T\mathbf{X})\}\mathbf{w}_i, \qquad \mathbf{w}_i = \frac{\mathbf{w}_i^+}{||\mathbf{w}_i^+||}, \tag{3}$$

where g denotes the derivative of the nonquadratic function G in (**??**), and $g(u) = \tanh(au)$.

The decorrelation process keeps different weight vectors from converging to the same maxima. For example, the $(p+1)$th weight vector is decorrelated from the preceding p weight vectors by (4),

$$\mathbf{w}_{p+1}^+ = \mathbf{w}_{p+1} - \sum_{i=1}^{p} \mathbf{w}_{p+1}^T\mathbf{w}_i\mathbf{w}_i, \qquad \mathbf{w}_{p+1} = \frac{\mathbf{w}_{p+1}^+}{||\mathbf{w}_{p+1}^+||}. \tag{4}$$

2.3. The Parallel ICA algorithm

In order to further speed up the FastICA execution, we designed a pICA algorithm that seeks the data parallel solution in SPMD parallelism [17].

PICA divides the process of weight matrix estimation into several subprocesses, where the weight matrix $\mathbf{W}$ is arbitrarily divided into k submatrices, $\mathbf{W} = (\mathbf{W}_1, \ldots, \mathbf{W}_z, \ldots, \mathbf{W}_k)^T$. Each subprocess estimates a submatrix $\mathbf{W}_z$ by the oneunit process and an *internal decorrelation*. The internal decorrelation decorrelates the weight vectors derived within the same submatrix $\mathbf{W}_z$ using (5),

$$\mathbf{w}_{z(p+1)}^+ = \mathbf{w}_{z(p+1)} - \sum_{j=1}^{p,p\le n_z-1} \mathbf{w}_{z(p+1)}^T\mathbf{w}_{zj}\mathbf{w}_{zj}, \qquad \mathbf{w}_{z(p+1)} = \frac{\mathbf{w}_{z(p+1)}^+}{||\mathbf{w}_{z(p+1)}^+||}, \tag{5}$$

where $\mathbf{w}_{z(p+1)}$ denotes the $(p+1)$th weight vector in the zth submatrix, n_z is the amount of weight vectors in $\mathbf{W}_z$, and the total number of weight vectors $n = n_1 + \cdots + n_z + \cdots + n_k$.

The internal decorrelation process only keeps different weight vectors within the same submatrix from converging to the same maxima. But two weight vectors generated from different submatrices could still correlate with each other. Hence, an *external decorrelation* process is needed to decorrelate the weight vectors from different submatrices using (6),

$$\mathbf{w}_{z(q+1)}^+ = \mathbf{w}_{z(q+1)} - \sum_{j=1}^{q,q\le(n-n_z-1)} \mathbf{w}_{z(q+1)}^T\mathbf{w}_j\mathbf{w}_j, \qquad \mathbf{w}_{z(q+1)} = \frac{\mathbf{w}_{z(q+1)}^+}{||\mathbf{w}_{z(q+1)}^+||}, \tag{6}$$

where $\mathbf{w}_{z(q+1)}$ denotes the $(q+1)$th weight vector in the zth submatrix $\mathbf{W}_z$, and $\mathbf{w}_j$ is a weight vector from another submatrix.

The structure of the pICA algorithm is illustrated in Figure 1. With the internal and the external decorrelations, we have decorrelated all weight vectors in all submatrices as if they are decorrelated in the same weight matrix. Hence, the ICA process can be run in a parallel mode, thereby distributing the computation burden from single process to multiple subprocesses in parallel environments. In the pICA algorithm, not only the estimations of submatrices but also the external decorrelation can be carried out in parallel.

3. SYNTHESIS

According to the structure of the pICA algorithm, we design the implementation structure, as illustrated in Figure 2. This design estimates four independent components, that is, $m = 4$. First of all, the weight matrix is divided into the two submatrices, each of which undergoes two oneunit estimations, generates four weight vectors in total using the input observed signal $\mathbf{x}$. Secondly, every pair of weight vectors in the same submatrix executes the internal decorrelation. The

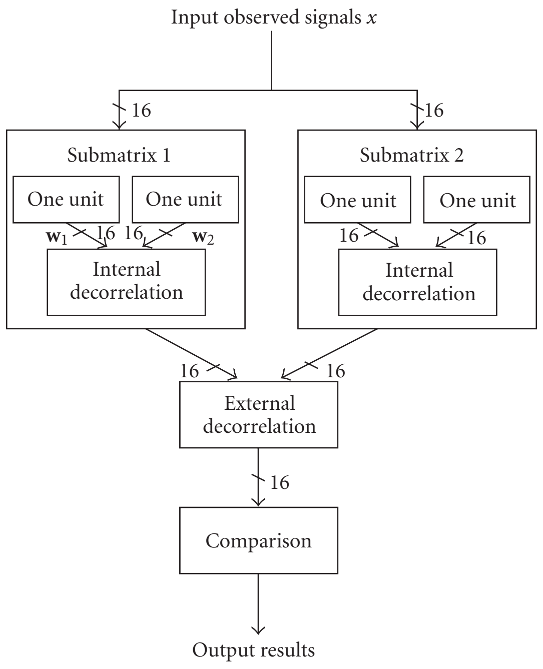


Figure 2: The implementation structure of the pICA algorithm.

four weight vectors then, respectively, undergo the external decorrelation with weight vectors from the other submatrix. So the decorrelated weight vectors generate the weight matrix **W**. Finally, we compare the weights of individual observation channels and select the most important ones. In this work, we set the bit width of both the observed signals and the weight vector to be 16.

Prior to the synthesis process of the pICA algorithm, we first develop three ICA-related RCs for reuse and retargeting purposes. The design and the use of RCs simplify the design process and allow for incremental updates. By using these fundamental RCs, we build up functional blocks according to the structure of the pICA algorithm. These blocks then set up process groups that will be implemented on the single reconfigurable FPGA system.

3.1. ICA-related reconfigurable components

Regarding functionality, the pICA algorithm consists of three main computations: the estimation of weight vectors, the internal and external decorrelations, and other auxiliary processing on the weight matrix. Hence, we develop three RCs for ICA-related implementations, including the one unit process, the decorrelation process, and the comparison process. The comparison process evaluates the importance of individual observation channel. The schematics of these three RCs, as shown in Figure 3, are parameterized using generics to make them highly flexible for future instances. In very high speed integrated circuit hardware description language (VHDL), the use of generics is a mechanism for passing information into a function model, similar to what Verilog provides in the form of parameters.

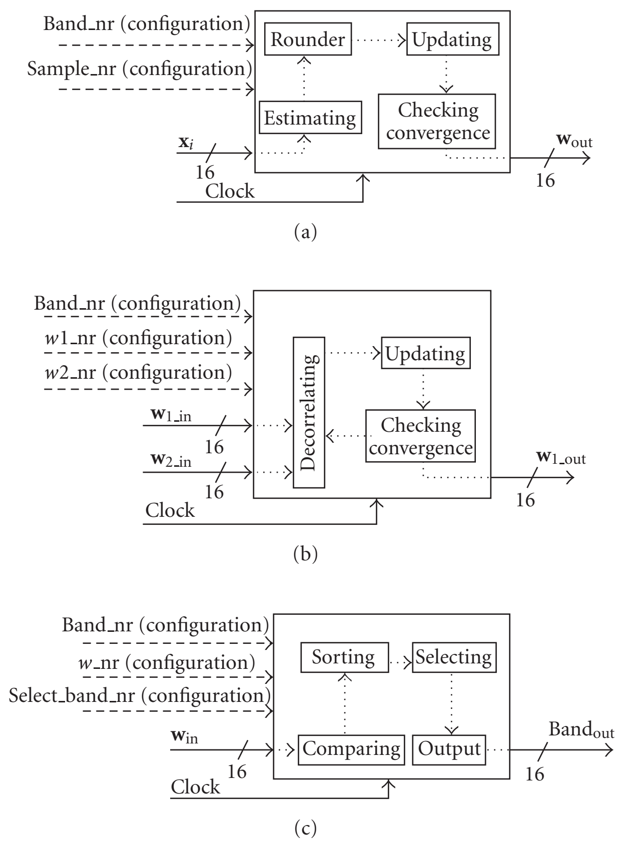
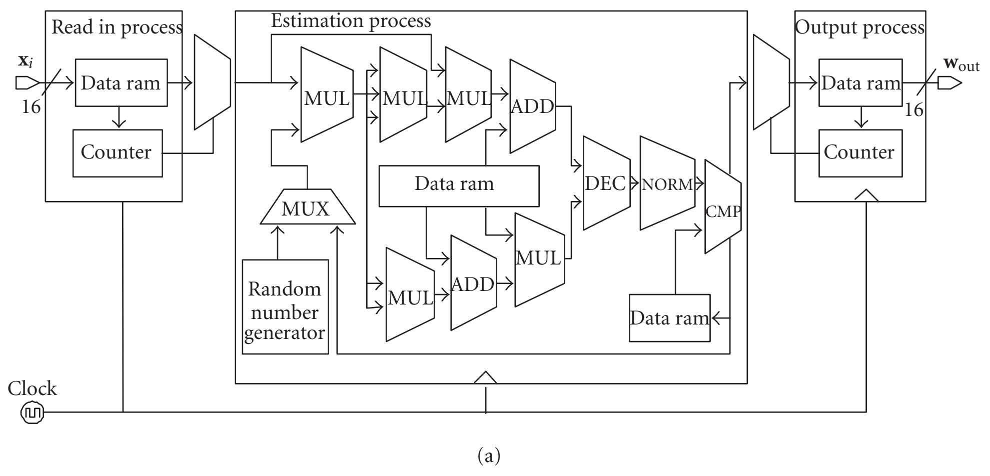
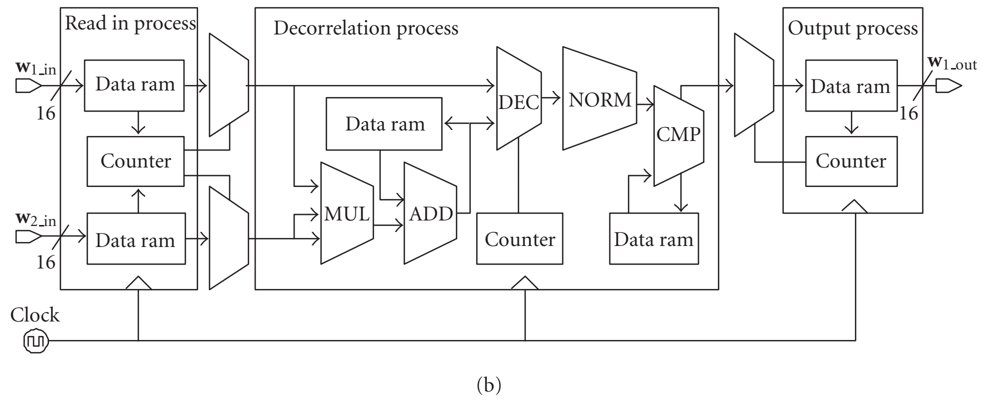


Figure 3: The schematic diagrams of the three RCs for ICA-related processes. (a) One unit estimation. (b) Decorrelation. (c) Comparison.

According to the FastICA and pICA algorithms described in Section 2, the one unit estimation is the fundamental process that estimates an individual weight vector. The input ports of the one unit RC consist of a 16-bit observed signal input ($\mathbf{x}_i$) and a 1-bit clock pulse (*clock*) that synchronizes the interconnected RCs. As we have described in Section 2, the dimensions of the observed signal and the weight vector are the same (n). Both the dimension (*dimension*) and the amount of input observed signals (*sample_nr*) are adjustable for different applications by customizing the reconfigurable generics. The output of the one unit RC ($\mathbf{w}_{\text{out}}$) is the estimated weight vector that needs to be decorrelated with others in the decorrelation process. Inside the one unit component, the 16-bit observed signal is fed to estimate one weight vector. The “rounder” is necessary for avoiding overflow, since it is a 16-bit binary instead of a floating point number used in the estimation. The weight vector is then iteratively updated until convergence, and then sent to the output port. Keeping the observation data and previously estimated weight vectors in the data RAM, Figure 4(a) demonstrates how the input process, the estimate process, and the output process in the one unit RC can be assembled in a pipelined state.

The decorrelation RC is designed for both the internal and the external decorrelations. The schematic diagram is shown in Figure 3(b). The input ports of the decorrelation





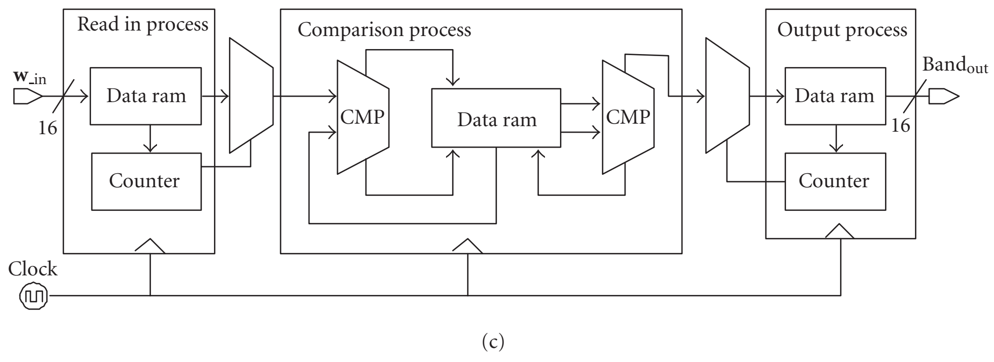


FIGURE 4: RTL schematics of the ICA-related RCs. (a) One unit estimation process. (b) Decorrelation process. (c) Comparison process.

RC include a 1-bit clock pulse (*clock*) and two 16-bit weight vector inputs ($\mathbf{w}_{1\_in}, \mathbf{w}_{2\_in}$), with $\mathbf{w}_{1\_in}$ being the weight vector to be decorrelated, and $\mathbf{w}_{2\_in}$ the sequence of previously decorrelated weight vectors. The generics parameterize the amount ($w1\_nr, w2\_nr$) and the dimension (*dimension*) of the decorrelated weight vectors. The output is a 16-bit decorrelated vector ($\mathbf{w}_{1\_out}$). As the internal diagram shows in Figure 4(b), the decorrelation RC also sets up a pipelined processing flow that includes the input process, the decorrelation process, and the output process.

The comparison RC sorts the weight values within the weight vectors that denote the significance of individual channels in the n observations and selects the most important ones, which are predefined by the end users according to specific applications. As shown in Figure 3(c), the input ports of the comparison RC include a 1-bit clock pulse (*clock*) and a 16-bit weight vector ($\mathbf{w}_{in}$). The generics set the dimension of the weight vector (*dimension*), the length of the weight vector sequence (w_nr), and the number of signal channels to be selected *(select_ band_nr)*. The output port yields the selected

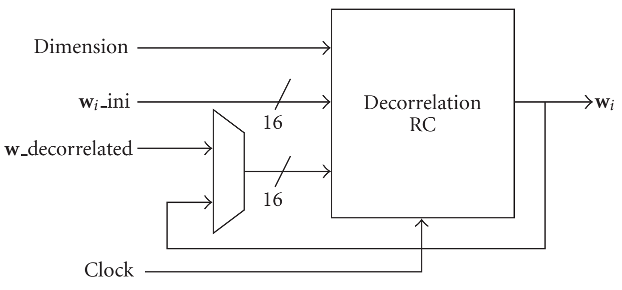


Figure 5: Internal decorrelation with multiple RCs in pipeline.

observation channels *(Band_out)*. Similarly, Figure 4(c) illustrates how the comparison process can be performed in the pipeline state.

The developed RCs are included in a library for the use in the synthesis process. The generics of the RCs are configured according to specific applications. The input and output ports of the RCs are interconnected to build up processes or subprocesses. In addition, the ICA-related RCs can be modified, improved, and extended to new RCs as necessary for other ICA applications. After developing the ICA-related RCs, we add them into a library for the purpose of reuse. During the design procedure, we select and configure appropriate RCs and integrate them to implement specific ICA applications.

3.2. Synthesis procedure

At the beginning of the synthesis work, the whole pICA process is divided into three independent functional blocks: the one unit (weight vectors) estimation, the internal/external decorrelation, and the comparison block. The one unit estimation block consists of several one unit RCs running in parallel, and the number of these RCs is constrained by the capacity limit of single FPGA. Each one unit RC independently estimates one weight vector, which is then collected and decorrelated in the decorrelation block.

The decorrelation block involves both the internal and the external decorrelations. In the internal decorrelation, one initial weight vector is fed to the first 16-bit data port, while the weight vector that does not need to be decorrelated or the previously already decorrelated weight vector sequence is input to the other 16-bit data port. The weight vectors within one submatrix are then iteratively decorrelated. As shown in Figure 5, the output decorrelated weight vector is then combined with the previously decorrelated weight vector sequence using a multiplexer to feed the consequent round as a new decorrelated weight vector sequence.

In the external decorrelation, if we use one decorrelation RC, the process works in virtually the same way as the internal decorrelation. The only difference is that the input decorrelated weight vector sequence is from another weight submatrix without multiplexing the output decorrelated weight vector. In order to speed up the decorrelation process, we can set up parallel processing using multiple decorrelation RCs, as demonstrated in Figure 6. The initial weight vectors from the current weight submatrix are, respectively, input to individual decorrelation RCs, while the decorrelated weight vector sequence from another weight submatrix is concurrently input to all RCs. The clock pulses are uniformly configured by external input for synchronization purpose.

Take a pICA process containing the estimation of four weight vectors as an example, the structure implemented on FPGA is shown in Figure 7. The one unit block of this design consists of four one unit RCs in parallel, the decorrelation block includes three decorrelation RCs, two for the internal decorrelation in parallel and one for the external decorrelation, and the comparison block contains one comparison RC.

A top level block is then designed to configure individual RCs and interconnect collaborative RCs. In addition, the top level block serves as the input/output interface that distributes the input data, synchronizes the clock pulse, and sends out the final results.

When the observed signals are input to the pICA process, the top level block distributes them to the one unit block. The weight vectors are then estimated in parallel and fed to the top level. The top block in turn forwards the estimated weight vectors to the decorrelation block. Finally, the comparison block receives the decorrelated weight vectors from the decorrelation block and compares, and selects the most important signal observation channels. The design is simulated using the ModelSim from Mentor Graphics.

4. FPGA IMPLEMENTATIONS

4.1. Single FPGA and its capacity limit

In general, FPGA/DSP platforms use PCI or PCMCIA slots to exchange data with memory and communicate with CPU. However, the data transfer speed can be extremely slow for applications with large data sets like hyperspectral images. Hence, we select the Pilchard reconfigurable computing platform that uses the DIMM RAM slot as an interface that is compatible with PC133 standard [18], thereby achieving very high data transfer rate. The Pilchard board is embedded with an Xilinx VIRTEX V1000E FPGA. In this work, we implement the pICA algorithm on the Pilchard board that is plugged into a sun workstation equipped with two UltraSPARC processors, as shown in Figure 8. Inside the FPGA, the core is partitioned into the arithmetic block and the dual port RAM (DPRAM) block (Figure 9). The DPRAM, whose capacity is 256×64 bytes, exchanges data between the implemented design and the external memory or cache through a 14-bit address bus and a 64-bit data bus. The Pilchard board with the pICA design therefore communicates directly with the CPU and memory on the 64-bit memory bus at the maximum frequency of 133 MHz.

As the implementation procedure demonstrated in Figure 10, the pICA algorithm shown in Figure 7 is first simulated by ModelSim from Mentor Graphics, then synthesized by Synopsys FPGA Compiler2, and finally placed and routed by Xilinx XVmake. After implementing pICA on Xilinx V1000E embedded on the Pilchard board, we

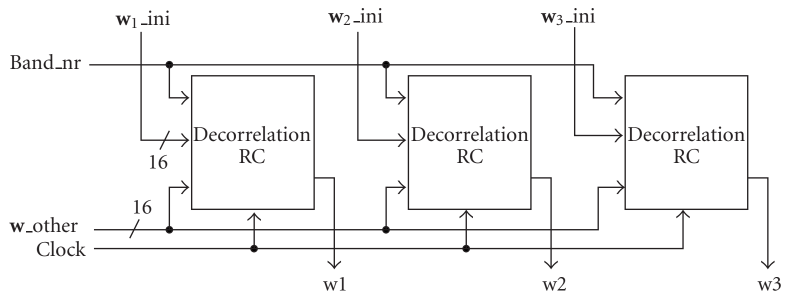


Figure 6: External decorrelation with multiple RCs in parallel.

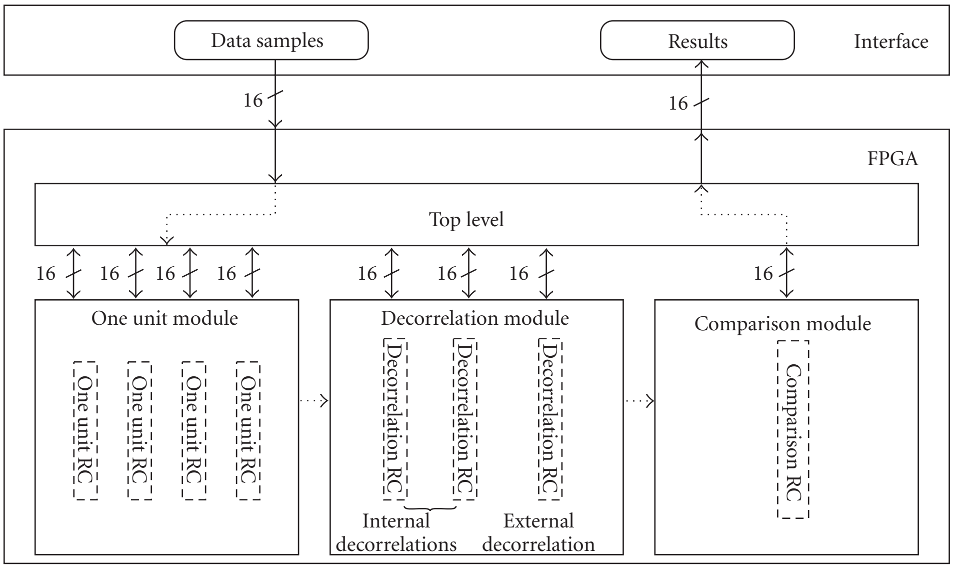


Figure 7: Architectural specification of pICA implemented on FPGA. (Solid lines denote data exchange and configuration. Dotted lines indicate the virtual processing flow.)

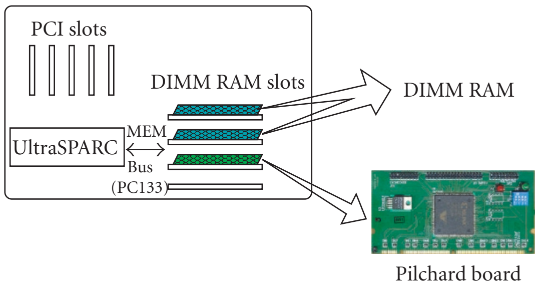


Figure 8: The Pilchard board.

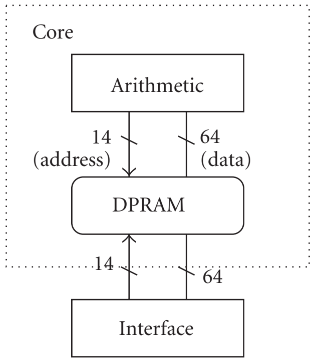


Figure 9: Hierarchy of the FPGA on Pilchard board. The DPRAM exchanges data between arithmetic and an interface written in C.

achieve the maximum frequency of 20.161 MHz (minimum period of 49.600 nanosecond) and the maximum net delay of 13.119 nanosecond. The pICA uses 92% slices of the V1000E. The detailed design and device utilization are listed in Table 1.

In the placement and routing process, however, we observe that several capacity constraints barricade single FPGA from implementing complex algorithms like pICA. Figure 11 shows the relationship between the number of weight vectors in pICA and the capacity utilization of the FPGA Xilinx VIRTEX V1000E. The evaluation metrics we use are the delay and the number of slices, where the delay reflects the design

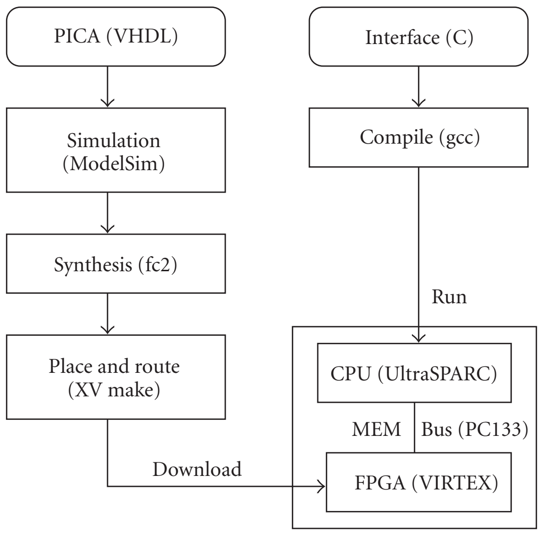
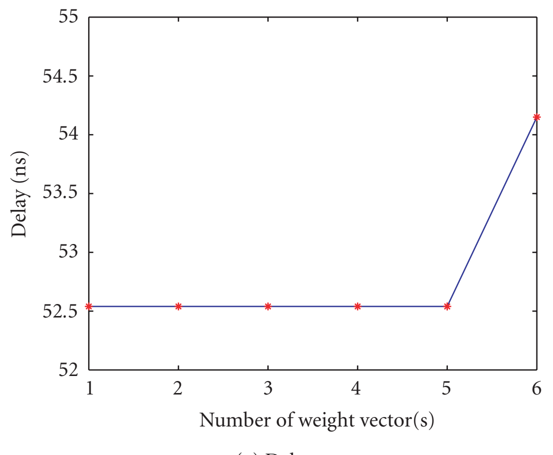


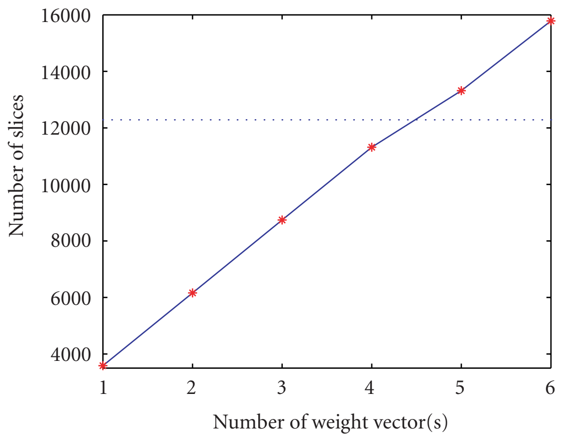
Figure 10: Implementation procedure of the pICA algorithm on Pilchard board.

Table 1: Design and device utilization.

Item	Amount	Percentage
Slices	11 318	92%
Flip-flops	6 061	24%
LUTs	19 114	77%
I/O pins	32	20%
Equivalent gate	229 500	—
After placing and routing		
Paths	129 753 145 344	—
Nets	26 884	—
Connections	73 169	—



(a) Delay



(b) Number of slices

Figure 11: Capacity utilization of Xilinx VIRTEX V1000E for different numbers of weight vectors in pICA. The dotted lines denote the maximum capacity of Xilinx VIRTEX V1000E.

performance and the number of slices puts a constraint on the capacity. In Figure 11(a), the delay that represents the processing speed of designs is estimated by software simulations. We find that the circuit delay significantly increases after the number of weight vectors exceeds five. This is because when the pICA design estimates too many weight vectors, the entire design is too large and the synthesis CAD tools have to run longer paths to connect logic blocks. This problem can be solved by using larger capacity FPGA to shorten the lengths of paths in order to reduce delay. The number of slices, as shown in Figure 11(b), reflects the area utilization of designs, which cannot exceed the available capacity of the target FPGA. We can see that the capacity constraint of Xilinx VIRTEX V1000E in the number of slices is a little more than 12 000. Hence, a single Xilinx VIRTEX V1000E can accommodate a pICA process with, at most, four weight vector estimations that already takes 92% of the maximum capacity. Considering the joint effects of the delay and the capacity constraints, on this FPGA, the pICA process cannot estimate larger number of weight vectors (more than 4) without partitioning or reconfiguration.

4.2. *Reconfigurable FPGA system*

We take the advantage of the reconfigurability feature of FPGA and construct a dynamically reconfigurable FPGA system in which the FPGA capacity limit is overcome by sacrificing the overall processing time.

In a general FPGA platform, all functional blocks are integrated together and synthesized on one FPGA, as shown in Figure 7, which can be executed for multiple times. In the reconfigurable FPGA system, instead of integrating all processes of pICA in one FPGA design, we divide them into three groups: the submatrix, the external decorrelation, and the comparison group. The submatrix group estimates a subweight matrix containing four weight vectors, since our target FPGA VIRTEX 1000E can only accommodate at most four weight vector estimations. So the submatrix group integrates four one unit RCs and two decorrelation RCs for internal decorrelation. In the external decorrelation group, we use four decorrelation RCs and set up a parallel processing

Table 2: Utilization ratios of resources for each group.

Group	Submatrix (4 weight vectors)	External decorrelation	Comparison
Slices	10 501 (85%)	10 683 (86%)	1 274 (10%)
Flip-flops	5 610 (22%)	7 081 (28%)	669 (2%)
LUTs	17 641 (71%)	17 635 (71%)	2 176 (8%)
I/O pins	104 (65%)	104 (65%)	104 (65%)
Maximum frequency	21.829 MHz	21.357 MHz	35.921 MHz

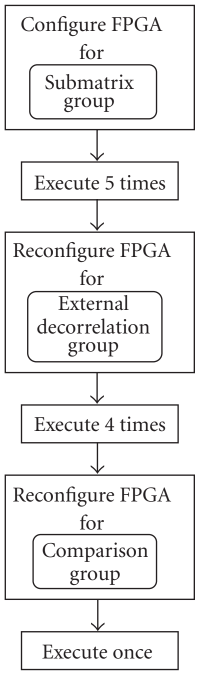


Figure 12: Global run-time reconfiguration flow.

as demonstrated in Figure 6 to decorrelate weight vectors generated from two different submatrices. The comparison group selects the most important observation channel as previously described.

In order to verify the effect of the design, each of these three groups is synthesized by Synopsys FPGA Compiler2 then placed and routed by Xilinx XVmake. Compared to Table 1 that shows synthesis performance of the overall pICA design with the estimation of four weight vectors, Table 2 lists the performance and device utilization ratios for individual groups in the reconfigurable design. Since the submatrix group still includes the internal decorrelation, its performance is similar to that in Table 1. The external decorrelation group includes four decorrelation RCs for parallel processing, thereby taking full use of available FPGA resources. Finally, the *bit* files that are ready to be downloaded to the Xilinx V1000E FPGA are generated by *BitGen* after the placement and routing.

In the reconfiguration process of the reconfigurable FPGA system, as shown in Figure 12, both the execution iteration and the sequence of each group are predefined. We take a reconfigurable FPGA system that estimates twenty weight vectors as an example. In this design, the submatrix group is executed five times, estimating and decorrelating four weight vectors each time. In order to decorrelate these five submatrices, the external decorrelation group needs to be executed hierarchically for four times. The comparison group is executed only once. A shell script file is written to control the reconfiguration flow at run-time, and a clock control block is used to distribute different clock frequencies. Individual groups of consecutive processing are downloaded on FPGA in sequence. The submatrix group is first downloaded to configure the Pilchard FPGA platform. After the submatrix group is executed and the task finished, the external decorrelation group is then downloaded to reconfigure the same FPGA. Since the immediate outputs from the preceding submatrix group are commonly used as inputs of the following configuration of the external decorrelation group, an external memory is used to store these intermediate signals that are originally the internal variables in single FPGA implementation.

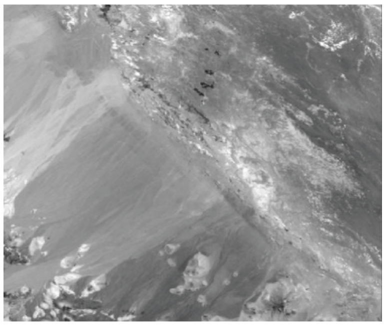
5. CASE STUDY

The validity of the developed reconfigurable FPGA system for the pICA algorithm is tested for the dimensionality reduction application in HSI analysis. Hyperspectral images carry information at hundreds of contiguous spectral bands [19, 20]. Since most materials have specific characteristics only at certain bands, a lot of these information is redundant. The goal of the pICA-based FPGA system is to select the most important spectral bands for the hyperspectral image [21].

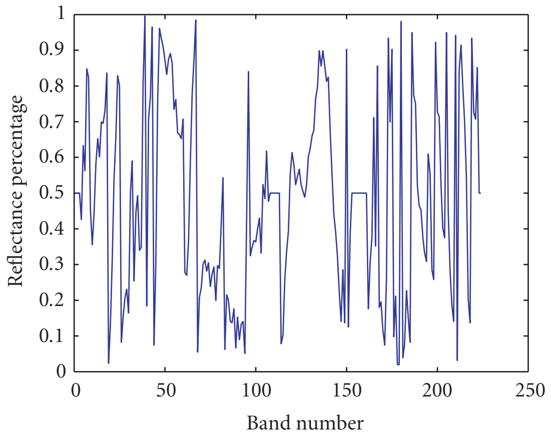
We take the NASA AVIRIS 224-band hyperspectral image (Figure 13(a)) as our testing example [22]. The image was taken over the Lunar Crater Volcanic Field in Northern Nye County at Nevada. The file size of this 614×512 hyperspectral image is 140.8 MB. We use the pICA algorithm to select 50 important spectral bands for this image, thereby reducing the data set by 22.3%.

Figure 14 demonstrates the Pilchard board workflow of the pICA-based dimensionality reduction. For each pixel in the hyperspectral image, the reflectance percentages of spectral bands are represented as 16-bit binaries, and then read in by the interface program written in C language. The interface program checks the execution status, advances these pixels to the pICA-based FPGA system, and obtains the selected spectral bands. As shown in Figure 15(a), the selected 50 bands on the spectral profile contain the most important information that describes the original spectral curve, including the maxima, the minima and the inflection points, thus retaining most spectral information.

The computation time of the pICA process with estimations of twenty weight vectors is compared between the implementations on the reconfigurable FPGA system and on a much faster workstation by C++, where the workstation has a Pentium 42.4 GHz CPU and 1 GB memory. Table 3 lists the percentage of the hyperspectral image processed and the computation time consumed in the respective



(a)



(b)

Figure 13: (a) The AVIRIS hyperspectral image scene [22]. (b) Original 224-band spectrum curve.

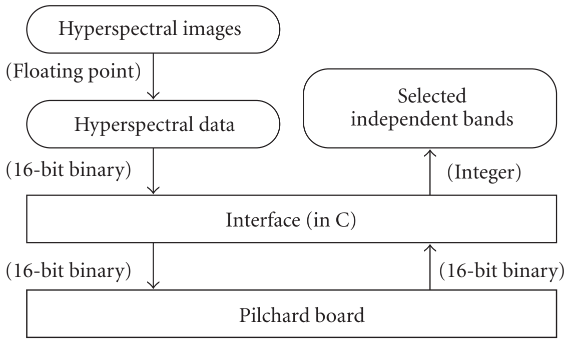
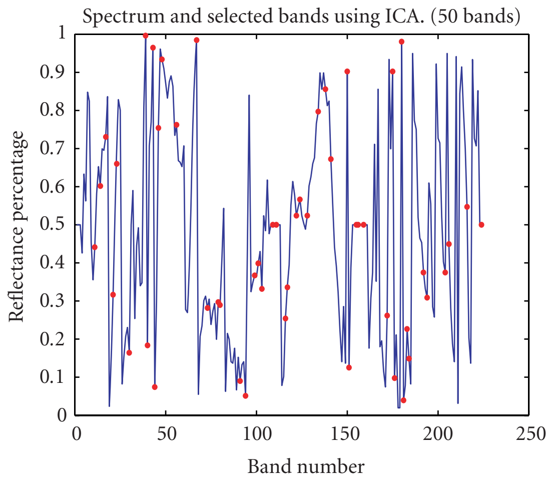
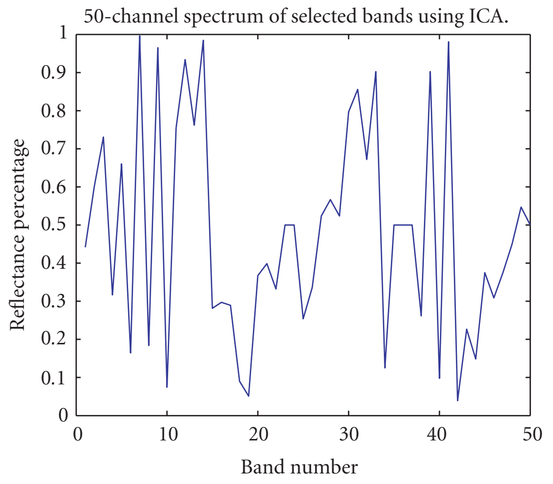


Figure 14: Workflow of pICA-based dimensionality reduction.



(a)



(b)

Figure 15: (a) The selected 50 spectral bands. (b) Spectrum curve plotted by the selected 50 spectral bands.

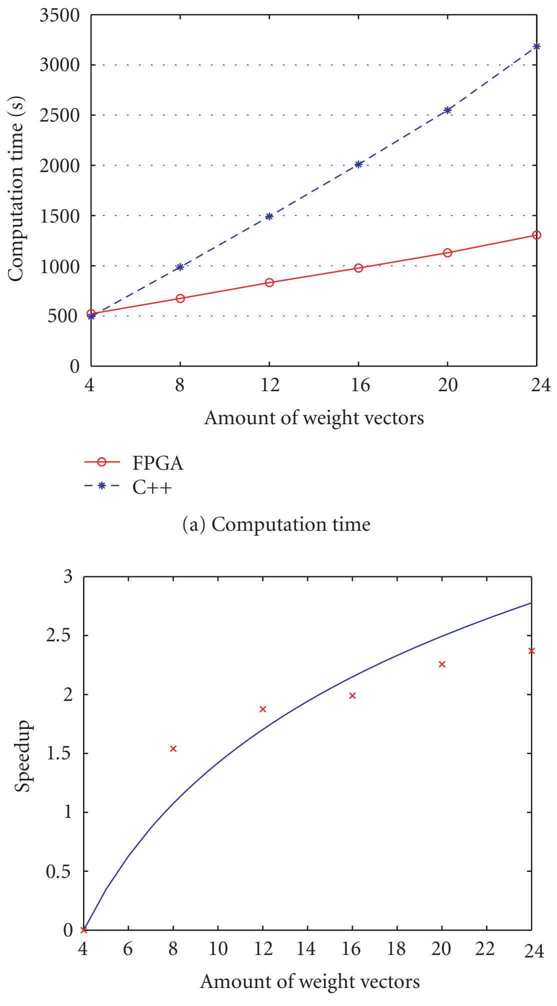
implementations. The configuration and execution time of individual groups are also shown in this table.

Next, we experiment the pICA estimations on the reconfigurable FPGA system using the number of weight vectors ranging from 4 to 24, with a 4-vector interval. Figure 16 elaborates the scalability and the speedup obtained by using the proposed reconfigurable FPGA system. Although the reconfigurable FPGA system consumes overhead time on reconfiguration and data buffering, the speedup compared to the C++ implementation is 2.257 when the amount of weight vectors is twenty.

In this case study, we have demonstrated the effectiveness of the proposed reconfigurable system in terms of providing significant speedup over software implementations while solving the limited capacity problem. We expect better performance of optimizing placement and routing, and implementing the system on modern high-end processors, like the AMD Opteron 64-bit processor. In addition, our current implementation platform, the Pilchard board, contains only one FPGA. If multiple FPGAs are available on one implementation platform, the proposed reconfigurable system can be conducted in the time sharing pattern to reduce the data transfer time, therefore speeding up the overall process.

Table 3: Computation time comparison for the pICA algorithm implementations of twenty weight vectors.

	C++ program	Reconfigurable FPGA system	
Data set used	100%	100%	
Computation time (s)	2548.8	Configuring submatrix group	5.3
		Executing submatrix group	891.7
		Configuring external decorrelation group	4.9
		Executing external decorrelation group	213.7
		Configuring comparison group	4.4
		Executing comparison group	9.5
		Total	1129.5



(a) Computation time

(b) Speedup

Figure 16: Computation time comparison between reconfigurable FPGA system and C++ implementation.

6. CONCLUSION

In this paper, we presented a run-time reconfigurable FPGA system implementation for the pICA algorithm to compensate for the performance limit of single FPGA. The implementation included the development of three reconfigurable components (RCs) based on the principal processes of the FastICA algorithm and the application of dimensionality reduction in HSI analysis. They are highly reusable and can be retargeted to other ICA applications. Based on these RCs, the FPGA implementation was reported, and the resource constraints of single FPGA were investigated in terms of delay and the number of slices. Our analysis concluded that current FPGA could not provide sufficient resource for complex iterative algorithms such as pICA in one design. The proposed reconfigurable FPGA system partitioned the pICA design into submatrix estimation, external decorrelation, and comparison groups. Individual groups were separately synthesized targeting to the Xilinx VIRTEX V1000E FPGA and achieved 85%, 86%, 10% capacity usages and 21.829 MHz, 21.357 MHz, 35.921 MHz maximum frequencies, respectively. The run-time reconfigurable system was executed in sequence on the Pilchard platform that transferred data directly to and from the CPU through the 64-bit memory bus at the maximum frequency of 133 MHz. The experimental results validated the effectiveness of the reconfigurable FPGA system. The speedup, compared to the C++ implementation, is 2.257 when the amount of weight vectors is twenty. The proposed reconfigurable FPGA system inspires an FPGA solution in performing complex algorithms with large throughput. More efficient solutions can be obtained by optimizing different synthesis levels.

ACKNOWLEDGMENT

This work was supported in part by Office of Naval Research under Grant no. N00014-04-1-0797. The authors would like to acknowledge Dr. Donald W. Bouldin and Mr. W. Joel Brooks from the University of Tennessee at Knoxville for their help.

REFERENCES

[1] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation*, vol. 9, no. 7, pp. 1483–1492, 1997.

[2] M. Bartlett and T. Sejnowski, "Viewpoint invariant face recognition using independent component analysis and attractor networks," in *Advances in Neural Information Processing Systems 9*, pp. 817–823, MIT Press, Cambridge, Mass, USA, 1997.

[3] M. Lennon, G. Mercier, M. C. Mouchot, and L. Hubert-Moy, "Independent component analysis as a tool for the

dimensionality reduction and the representation of hyperspectral images," in *Proceedings of IEEE International Geoscience and Remote Sensing Symposium (IGARSS '01)*, vol. 6, pp. 2893–2895, Sydney, NSW, Australia, July 2001.

[4] P. Comon, "Independent component analysis, a new concept?" *Signal Processing*, vol. 36, no. 3, pp. 287–314, 1994, special issue on High-Order Statistics.

[5] T.-W. Lee, M. S. Lewicki, and T. J. Sejnowski, "ICA mixture models for unsupervised classification of non-Gaussian classes and automatic context switching in blind signal separation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 10, pp. 1078–1089, 2000.

[6] M. H. Cohen and A. G. Andreou, "Analog CMOS integration and experimentation with an autoadaptive independent component analyzer," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 2, pp. 65–77, 1995.

[7] K.-S. Cho and S.-Y. Lee, "Implementation of infomax ICA algorithm with analog CMOS circuits," in *Proceedings of the 3rd International Conference on Independent Component Analysis and Blind Signal Separation*, pp. 70–73, San Diego, Calif, USA, December 2001.

[8] A. Celik, M. Stanacevic, and G. Cauwenberghs, "Mixed-signal real-time adaptive blind source separation," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '04)*, vol. 5, pp. 760–763, Vancouver, Canada, May 2004.

[9] G. Cauwenberghs, "Neuromorphic autoadaptive systems and independent component analysis," Tech. Rep. N00014-99-1-0612, Johns Hopkins University, Baltimore, Md, USA, 2003, http://bach.ece.jhu.edu/gert/yip/.

[10] D. Bouldin, "Developments in design reuse," Tech. Rep., University of Tennessee, Knoxville, Tenn, USA, 2001.

[11] A. B. Lim, J. C. Rajapakse, and A. R. Omondi, "Comparative study of implementing ICNNs on FPGAs," in *Proceedings of International Joint Conference on Neural Networks (IJCNN '01)*, vol. 1, pp. 177–182, Washington, DC, USA, July 2001.

[12] A. Nordin, C. Hsu, and H. Szu, "Design of FPGA ICA for hyperspectral imaging processing," in *Wavelet Applications VIII*, vol. 4391 of *Proceedings of SPIE*, pp. 444–454, Orlando, Fla, USA, April 2001.

[13] F. Sattar and C. Charayaphan, "Low-cost design and implementation of an ICA-based blind source separation algorithm," in *Proceedings of the 15th Annual IEEE International ASIC/SOC Conference*, pp. 15–19, Rochester, NY, USA, September 2002.

[14] Y. Wei and C. Charoensak, "FPGA implementation of non-iterative ICA for detecting motion in image sequences," in *Proceedings of the 7th International Conference on Control, Automation, Robotics and Vision (ICARCV '02)*, vol. 3, pp. 1332–1336, Singapore, December 2002.

[15] T. Yamaguchi and K. Itoh, "An algebraic solution to independent component analysis," *Optics Communications*, vol. 178, no. 1, pp. 59–64, 2000.

[16] T. M. Cover and J. A. Thomas, *Element of Information Theory*, John Wiley & Sons, New York, NY, USA, 1991.

[17] H. Du, H. Qi, and G. D. Peterson, "Parallel ICA and its hardware implementation in hyperspectral image analysis," in *Independent Component Analyses, Wavelets, Unsupervised Smart Sensors, and Neural Networks II*, vol. 5439 of *Proceedings of SPIE*, pp. 74–83, Orlando, Fla, USA, April 2004.

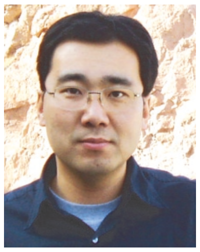
[18] P. H. W. Leong, M. P. Leong, O. Y. H. Cheung, et al., "Pilchard—a reconfigurable compouting platform with memory slot interface," in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*, pp. 170–179, Rohnert Park, Calif, USA, April-May 2001.

[19] S.-S. Chiang, C.-I. Chang, and I. W. Ginsberg, "Unsupervised hyperspectral image analysis using independent component analysis," in *Proceedings of IEEE International Geoscience and Remote Sensing Symposium (IGARSS '00)*, vol. 7, pp. 3136–3138, Honolulu, Hawaii, USA, July 2000.

[20] D. Landgrebe, "Some fundamentals and methods for hyperspectral image data analysis," in *Systems and Technologies for Clinical Diagnostics and Drug Discovery II*, vol. 3603 of *Proceedings of SPIE*, pp. 104–113, San Jose, Calif, USA, January 1999.

[21] H. Du, H. Qi, X. Wang, R. Ramanath, and W. E. Snyder, "Band selection using independent component analysis for hyperspectral image processing," in *Proceedings of the 32nd Applied Imagery Pattern Recognition Workshop (AIPR '03)*, pp. 93–98, Washington, DC, USA, October 2003.

[22] NASA, Jet Propulsion Laboratory, California Institute of Technology, AVIRIS concept, 2001, http://aviris.jpl.nasa.org/html/aviris.concept.html.



Hongtao Du received his M.S. degree in computer engineering from the University of Tennessee in 2003, B.S. and M.S. degrees in electrical engineering from Northeastern University, Shenyang, China, in 1997 and 2000, respectively. He is now working toward his Ph.D. degree in computer engineering at The University of Tennessee, Knoxville. His current research interests include parallel/distributed image and signal processing, task and data partitioning on VLSI, reconfigurable and virtual platform, and high performance computing.

Hairong Qi received her Ph.D. degree in computer engineering from North Carolina State University in 1999, B.S. and M.S. degrees in computer science from Northern JiaoTong University, Beijing, China, in 1992 and 1995, respectively. She is now an Associate Professor in the Department of Electrical and Computer Engineering at The University of Tennessee, Knoxville. Her current research interests are advanced imaging and collaborative processing in sensor networks, hyperspectral image analysis, and bioinformatics. She has published over 80 technical papers in archival journals and refereed conference proceedings, including a coauthored book in machine vision. She is the recipient of the NSF CAREER Award, Chancellor's Award for Professional Promise in Research and Creative Achievement. She serves on the Editorial Board of Sensor Letters and is the Associate Editor for Computers in Biology and Medicine.