IMPLEMENTATION IN FPGAS OF JACOBI METHOD TO SOLVE THE EIGENVALUE AND EIGENVECTOR PROBLEM

Ignacio Bravo, Pedro Jiménez, Manuel Mazo, José Luis Lázaro, Alfredo Gardel*

Department of Electronics. University of Alcala.
Ctra. Madrid – Barcelona km 33.600. 28871 Alcala de Henares (Madrid) – Spain.
http://www.depeca.uah.es / ibravo@depeca.uah.es

ABSTRACT

This work shows a modular architecture based on FPGA's to solve the eigenvalue problem according to the Jacobi method. This method is able to solve the eigenvalues and eigenvectors concurrently. The main contribution of this work is the low execution time compared with other sequential algorithms, and minimal internal FPGA consumed resources, mainly due to the fact of using the CORDIC algorithm. Two CORDIC modules have been designed to solve the trigonometric operations involved. A parallel CORDIC architecture is proposed as it is the best option to compute the eigenvalues with this method. Both CORDIC modules can work in rotation and vector mode. The whole system has been done in VHDL language, attempting to optimize the design.

1. INTRODUCTION

The calculation of eigenvalues and eigenvectors is a problem that appears in many practical applications of different scientific areas (artificial vision, digital signal processing, power electronics, etc.). Concretely this work is applied in the PCA algorithm of artificial vision (Principal Component Analysis). In any case, the proposed solution can be extended to any other application that needs to solve the eigen problem. The only condition that must be fulfilled is that the input matrix must be symmetrical and with elements belong to the real type, which happens in many practical applications. Respect the maximum matrix size in our work is limited to the FPGA resources. The usual maximum matrix size does not go over 20x20 in practical applications using PCA. In any case, the proposed hardware solution presented herein for the calculation of eigenvalues and eigenvectors, allows easily modifying the HW design to matrices of greater size.

The developed solution has the advantage to use distinct HW modules that run in parallel, with a minimum data dependency between the different modules, achieving a high speed computation of the eigenvalues and eigenvectors for an input matrix.

The rest of the paper is divided into the following sections. Section 2 describes the fundamental aspects of the Jacobi method for computation of eigenvalues and eigenvectors and their possible implementation in a Systolic Array. The use of CORDIC structures to compute the Jacobi algorithm is developed in Section 3. Then, Section 4 is devoted to present the HW design. Finally several results and conclusions are briefly discussed in Sections 5 and 6.

* Project SILPAR (Ministerio de Ciencia y Tecnología ref: DPI2003-05067) and "Cátedra de control electrónico en transportes" founded by LOGYTEL and RENFE.

2. METHOD OF JACOBI TO SOLVE THE EIGEN PROBLEM

The classic Jacobi method (1846) is based on the obtaining of the first diagonal (*D*) of an original matrix (*A*) by means of a sequence of rotations or iterations [1].

$$A = U \cdot D \cdot V^T \qquad (1)$$

where *U* and *V* are orthogonal matrices. Diagonal *D* contains the Singular Values Decomposition (SVD) of the original matrix. Working with Hermitian matrices, the singular values are equal to the eigenvalues. Thus, for this case, the expression (1) can be reformulated as:

$$A = U \cdot D \cdot U^T \qquad (2)$$

where the orthogonal matrix *U* comes to be the matrix of eigenvectors.

For each iteration, the matrix values outside the diagonal are diminished. To make null these outer values, the number of necessary iterations would be infinite. The end of the iteration process takes into account a threshold value in order to consider that the matrix has been diagonalized.

If the initial matrix has a size of nxn elements, several authors as in [2] propose a systolic architecture of processors based on the Jacobi method, which will handle matrices of 2x2 elements. Two different HW architectures based in systolic arrays must be implemented, one for eigenvalues and another for eigenvectors.

In the case of eigenvalues, initially each processor is loaded with a 2x2 matrix from the original matrix. There are two types of processors $P_{i,j}$ according to the position that occupies within the systolic array. Thus, one type will be the Diagonal Processor (DP) corresponding to those systolic processors in which *i(rows)=j(columns)* and Non-

Diagonals Processors (NDP) to systolic array processors in which $i \neq j$.

Each processor computes the formula given by (3):

$$P_{ij}^{(k+1)} = R(\alpha_{j,j}^{(k)})^T \cdot P_{ij}^{(k)} \cdot R(\alpha_{i,i}^{(k)}) \qquad (3)$$

where $i,j=1, 2\ldots ,n/2$, $k=0,\ldots, n\log n$ and $R(\alpha)$ is the so-called rotation matrix $\begin{bmatrix} \cos\alpha \sin\alpha \\ -\sin\alpha\cos\alpha \end{bmatrix}$ considering a rotation angle α. Expression (3) has two rotation angles (α_{ii}, α_{jj}) due to considering two processor types, the rotation angles for R will be different. For DP's the rotation angles $\alpha_{i,i}$ and $\alpha_{j,j}$ must eliminate the off-diagonal values. Thus, making equal $\alpha_{i,i}$ to $\alpha_{j,j}$ in (3), the rotation angle $\alpha_{i,i}$ is formulated in (4).

$$tg(2\alpha_{i,i}^{(k)}) = \frac{2a_{2l-1,2m}^{(k)}}{a_{2l,2m}^{(k)} - a_{2l-1,2m-1}^{(k)}} \qquad (4)$$

The rotation angles needed in the NDP processors will be always different. The rotation angles for a NDP processor come from the DP of row (i), $P_{i,i}$, and DP of column (j),$P_{j,j}$. The NDP's transfer its partial results obtained in each iteration to the DP's in order to eliminate the transmitted element.

When each processor has computed the operations of expression (3) with its corresponding angles, the obtained partial results in each processor must be carefully rearranged transferring them between the different processors. After looping $n\log n$ iterations computing the described process, the final result is a diagonal matrix [2].

The matrix of eigenvectors ($B^{(k)}$) associated to the eigenvalues of A, is also based on the decomposition of the matrix in matrices of 2x2 elements. In this case, the input matrix to be subdivided is the identity matrix [2]. Successive iterations will be applied, until finding the eigenvectors associated. In this case there is no difference between DP's and NDP's processors, since once the first rotation is accomplished in the first iteration, no processor will handle symmetrical matrices. The proposal in [2] is to implement a new processor array of size $n/2 \times n/2$, where the equation to compute for each processor (5), is different from the one given for eigenvalues (3).

$$B_{i,j}^{(k+1)} = B_{i,j}^{(k)} \cdot R(\alpha_{j,j}^{(k)}) \qquad (5)$$

where $B_{i,j}$ correspond to one systolic eigenvector processor (VP), which also is loaded with matrices of 2x2 elements. After the same number of iterations that in the eigenvalues proccesors, the resulting matrix is the eigenvectors matrix associated to the eigenvalues calculated by the other stage.

3. USE OF CORDIC TO ACCOMPLISH THE JACOBI METHOD.

The CORDIC (*COordinate Rotation DIgital Computer*) algorithm has been widely used in FPGÁs as a tool to resolve many mathematical operations [3]. In particular, for the Jacobi method, it can be used to make the calculation of the angle in the DP's, (4), the double rotation in the NDP's and DP's (3) and the simple rotation in the VP's as shown in expression (5) [4].

Computing the rotation angle for each DP is solved by means of the arc tangent formula as previously exposed (4). This inverse trigonometric operation can be resolved by means of CORDIC [3] using circular coordinates and vector mode. Really, computing $\alpha_{i,i}^{(k)}$ consists on obtaining the resulting angle of the rotation of a vector.

Considering the double rotation for the DP's and NDP's, and the simple rotation for the VP's, also CORDIC can be a good approximation method to compute these operations using CORDIC modules with rotation mode.

4. IMPLEMENTATION DESIGN

According to the previous section, the Jacobi method managing 2x2 matrices [2], represents a regular design suitable for implementation in reconfigurable HW such as FPGA's [5]. The requirement in area resources of the FPGA is too large if both systolic architectures are implemented (one for eigenvalues and another one for eigenvectors), that are based on processors with CORDIC modules that use a high number of internal resources. For example, a serial CORDIC module of 16 bits, generated with the tool Xilinx Core Generator occupies up to 400 slices and the number of necessary CORDIC modules for an input matrix of 6x6 elements is 57. The main goal of our proposal design must be reduce the maximum number of CORDIC modules needed. This reduction can be applied from two points of view:

- Multiplexing the CORDIC modules while computing double rotations, reducing the number of modules CORDIC but increasing the execution time.
- Taking advantage of the symmetric property of the input matrix, to reduce the number of symmetrical processors. Thus the systolic array would be transformed into a triangular array (only processors in the diagonal and the lower or upper part of the diagonal).

Even with these types of optimization, the resources used in the implementation represent an excessive number of internal resources. Applying both considerations, the number of necessary CORDIC modules for an input matrix of 6x6 elements would descend from 57 to 15 units (in this case supposes a reduction of 73%). The number of slices associated to these CORDIC modules would occupy up to 6000 slices. Other authors [5] presents an architecture with several modifications from the solution given in [2] where the total number of slices for a matrix of 6x6 and with data of 16 bits reaches the 9900 slices. As shown, the large amount of necessary resources makes worth to look for other alternatives.

We will exploit the sequential property of each one of

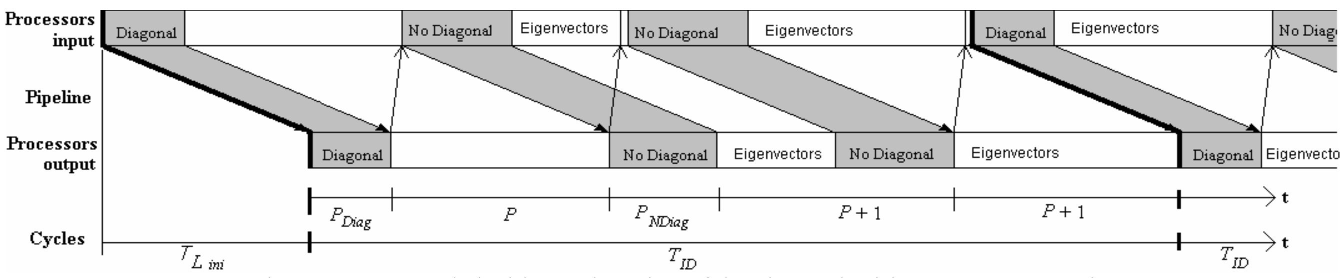

Fig. 1. Operating sequence made inside one iteration of the eigenvalue/eigenvector computing process.

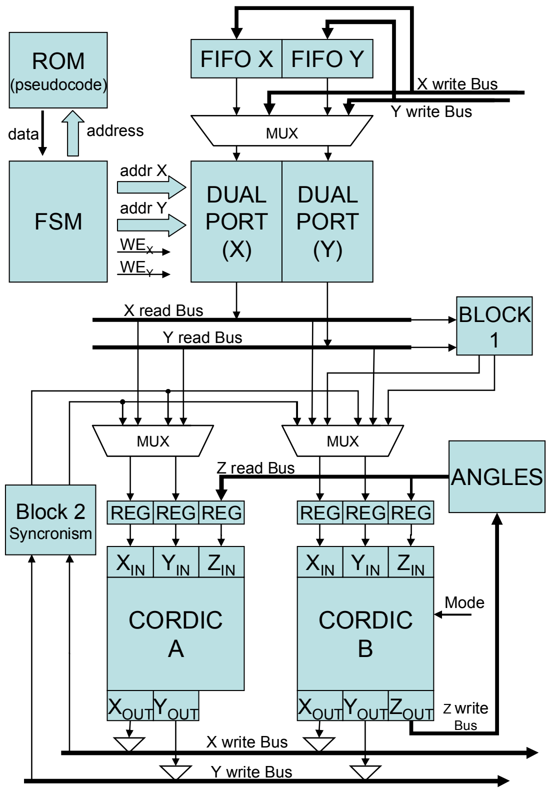

Fig. 2. Developed system to solve eigenvalues and eigenvectors based on the Jacobi method.

the iterations in the Jacobi method (data load, angles computation, double rotation in eigenvalues and simple rotation in eigenvectors, reordering of partial results and data interchange). Our proposal design is shown in Fig. 2. The main modules of the design are the following:

- ROM memory: Based on the processing steps involved by the Jacobi method, an instruction set has been codified, decoded by the FSM module. Each one of the bits of the instruction is associated to a control signal that configures the system in order to accomplish each one of the stages of the Jacobi method.
- Dual Port memory (DPM): It stores the data of the input matrix as well as the variable data of each iteration, with side ports X and Y.
- CORDIC modules: There exist two different types (A/B) both using circular coordinates. The module A will only work in the rotation mode while module B works in both rotation and vector mode. It is enough to compute the rotation angle with only one module while maintaining the maximum computing speed of the whole system. In both cases, the CORDIC architecture used is parallel with m stages, (equal to the size of the input data). The maximum value of m is limited to 18 bits due to the use of embedded FPGA multipliers that will apply the correction factor in the output of the CORDIC module. If one desires to increase the size of the input data, an extension in the block of the correction factor must be made by means of the use of more multipliers.
- FSM: Reads the information from the ROM memory in order to generate the necessary control signals as well as the address data for the Dual-Port Memory (DPM).
- FIFO memories: these memory modules store the eigenvectors temporarily when the data bus of the DPM is busy while storing the eigenvalues.
- Block 1: This module generates the input data for CORDIC B using the data from other blocks computing the rotation angle (numerator and denominator of (4)).
- Block 2: It implements the synchronism block between the feedback data and interchange between the first and second rotation.

In order to obtain the maximum speed, CORDIC modules that execute in all the processing steps have a pipelined architecture with P stages, obtaining a pipelined design.

The operation sequence of one iteration showing the pipeline contents is presented in Fig. 1. The total execution time (T_{Total}) is the sum of the time to load the initial values ($T_{L\ ini}$), the computing time for the eigenvalues and eigenvectors (T_c) and the latency of extracting all the data stored in the pipeline ($T_{L\ extract}$), being $T_{L\ ini}$ y $T_{L\ extract}$ much smaller than the effective computing time (see Fig. 1). $T_{L\ ini}$ is the time spent to load the input matrix for the eigenvalues and the identity matrix for the eigenvectors given by (6).

The computing time T_c is calculated as the number of iterations by the time of each iteration, T_{ID} (7). This time depends on the length of pipeline (P) for the CORDIC processors. In each iteration, according to the described architecture, the first step made is the calculation of the rotation angles. This is made using only module CORDIC

Table 1. Comparation between FPGA and PC execution time.

Matriz Size	**FPGA – 19 Iterations**				**PC – 32 bits**			
	Bits	**Error [%]**	**Slices**	**T_{Total} [us]**	**Bits**	**Itera tions**	**Error [%]**	**T_{Total} [us]**
8x8	18	0.304	1300	**15.06**	64	5	0.204	**25.06**
10x10	18	0.324	1400	**16.53**	64	5	0.124	**36.53**
12x12	18	0.373	1500	**18.25**	64	5	0.273	**48.25**
16x16	18	0.432	1800	**24.15**	64	5	0.332	**64.15**

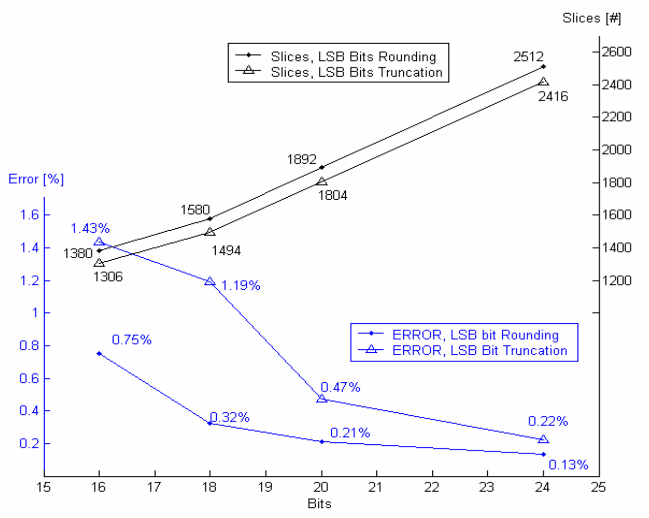

Fig. 3. Area resources and accuracy along the total number of bits and rounded type used for 8x8 input matrix.

B, reason why it requires $P_{ND}=n/2$ operations. Once the rotation angles are obtained the computing of two rotations can be started for all the processors in the eigenvalues stage. Since the eigenvalues matrix is symmetrical there is a total of P_{EIGVAL} (8) rotation processors. After each rotation it is necessary to include a delay cycle to synchronize the modules of the presented architecture.

$$T_{L\,\mathrm{ini}} = \tfrac{1}{2}[n(n+1)]T_{CLK} + n^2 T_{CLK} = \tfrac{1}{2}(3n^2+n)T_{CLK} \qquad (6)$$

$$T_{ID}=(P_{Diag}+P)+(P+1)+(P+P_{NDiag}+1) = \frac{n^2+6n}{8}+3P+2 \qquad (7)$$

$$P_{EIGVAL}=\tfrac{1}{2}[(n/2)2+n/2] \qquad (8)$$

5. RESULTS

The system shown in Fig. 2, has been implemented in a *Virtex II-Pro*, *XC2VP7*, which has 4928 slices and 44 hardware multipliers. The design has been implemented with different word widths, matrix sizes and applying different types of rounded in the multipliers output of the CORDIC correction factor.

The maximum operating frequency reaches 110 MHz. Modifying the number of bits for the input data does not change largely the operating times, except for 2 extra *pipeline* steps for data widths greater than 18 bits.

Respect to the results accuracy, for 8x8 input matrix, Fig. 3 shows the average quadratic error between the obtained eigenvectors using the function *eig* of Matlab (using floating point data) and the results obtained by the implemented design in the FPGA. The eigenvalue error is slightly lower than eigenvector error. In the worst case (truncation error, 16 bits) the error is under 1.4 %. According to Fig. 3 the number of occupied slices is between 1306 for 16 bits, truncating, and 2512 for 24 bits, rounding.

Table 1 shows the processing times using a PC (PIV-2.66 GHz) to obtain a similar accuracy as the one obtained by the FPGA design. The program is written in C, using double data types (floating point with 64 bits). According to the information of that table, we can say that our proposal in the best case, decreases the PC execution time around a 60% with a 0.43% data error.

6. CONCLUSIONS

The presented HW architecture based on the Jacobi method given in [2] solves the calculation of eigenvectors and eigenvalues inside a FPGA, with high accuracy, taking low execution time and using a reduced number of internal resources. Besides, the execution time is lower than an optimal algorithm to solve the eigen problem in a PC. Other important contribution of this work is the low number of internal resources consumed (see Fig. 3).

These features are the main contributions of this paper, improving remarkably other results from similar works [5]. The modularity of the proposed system allows an easy adaptation to different matrix sizes. It can be used in any application, as long as the input matrix is square, real and symmetrical.

7. REFERENCES

[1] Wilkinson J.H. “The algebraic eigenvalue problem”. Oxford Science Publications. 1999. ISBN: 0198534183.

[2] Brent R.P., Luk F.T., Van Loan C. “Computation of the Singular Value Descomposition Using Mesh-Connected Processors”. Journal of VLSI and Computer Systems, Volume I, Number 3. pp 242-270. 1983

[3] Andraka R. “A survey of CORDIC algorithms for FPGA based computers”. Proceed. of the International symposium on Field Programmable Gate Arrays, pp: 191 – 200. 1998.

[4] Cavallaro J.R., Luk, F.T.. “CORDIC arithmetic for an SVD processor”. Journal of Parallel and Distributed Computing, no5, pp: 271-290. 1998.

[5] Ahmedsaid, A.; Amira, A.; Bouridane, A.; “Accelerating MUSIC method on reconfigurable hardware for source localisation”. Proceed. of the Intern. Symposium on Circuits and Systems (ISCAS). Vol. 3, pp.:III - 369-72, 2004.