FPGA-based Architecture for Computing Testors

Alejandro Rojas, René Cumplido, J. Ariel Carrasco-Ochoa, Claudia Feregrino, J. Francisco Martínez-Trinidad

Computer Science Department, INAOE, Apdo. Postal 51 &216
Tonantzintla, Puebla, México
roherale, rcumplido, ariel, cferegrino, fmartine @inaoep.mx

Abstract. Irreducible testors (also named typical testors) are a useful tool for feature selection in supervised classification problems with mixed incomplete data. However, the complexity of computing all irreducible testors of a training matrix has an exponential growth with respect to the number of columns in the matrix. For this reason different approaches like heuristic algorithms, parallel and distributed processing, have been developed. In this paper, we present the design and implementation of a custom architecture for BT algorithm, which allows computing testors from a given input matrix. The architectural design is based on a parallel approach that is suitable for high populated input matrixes. The architecture has been designed to deal with parallel processing of all matrix rows, automatic candidate generation, and can be configured for any size of matrix. The architecture is able to evaluate whether a feature subset is a testor of the matrix and to calculate the next candidate to be evaluated, in a single clock cycle. The architecture has been implemented on a Field Programmable Gate Array (FPGA) device. Results show that it provides significant performance improvements over a previously reported hardware implementation. Implementation results are presented and discussed.

Keywords: Feature Selection, Testor Theory, Hardware Architecture, FPGA.

1 Introduction

Although the theoretical aspect of computing irreducible testors is advanced, there are not practical hardware implementations reported previously, excepting a brute force approach [1]. The intensive computational requirements due to the exponential complexity of the algorithms can be met by a combination of technology improvements and efficient hardware architectures based on parallel computational models. Specific parallel architectures can be designed to exploit the parallelism found in the algorithms to speed up the processing. Further optimizations such as incremental processing and the use of multiple processing elements are also possible.

In Pattern Recognition, feature selection is a very important task for supervised classification. A useful way to do this selection is through Testor Theory. The concept of testor for Pattern Recognition was introduced by Zhuravlev [2] in 1966. He defined a testor as a subset of features that allows differentiating objects from different classes. Testors are quite useful, especially when an object description contains both

qualitative and quantitative features, and maybe they are incomplete (mixed incomplete data)[3].

However, the algorithms used to compute all irreducible testors have exponential complexity which seriously limits their practical use. Since software implementations of these algorithms do not provide a reasonable performance for practical problems, an option is to migrate to hardware implementations based on programmable logic to take advantage of the benefits that they offer.

This work is a continuation of the work reported in [1] and reports the development of a configurable hardware architecture for computing testors using the BT algorithm. The architecture is based on a candidate generator that jumps over unnecessary candidates, thus reducing the number of comparisons needed.

The rest of the paper is organized as follows. Section 2 provides the theoretical foundation of testor identification and describes the BT algorithm. Section 3 presents the proposed hardware architecture. In section 4 the FPGA implementation and experimental results are presented. In section 5, the performance improvements are briefly discussed and the obtained results are compared against the brute force approach, and software implementation. Finally, section 6 presents the concluding remarks and directions for further research.

2 Algorithms for computing testors

Let *TM* be a training matrix with *K* objects described through *N* features of any type ($x_1,...,x_N$) and grouped in *r* classes. Let *DM* be a dissimilarity Boolean matrix (0=similar,1=dissimilar), obtained from feature by feature comparisons of every pair of objects from *T* belonging to different classes. *DM* has *N* columns and *M* rows, where $M>>K$.

Testors and Irreducible Testors are defined as follows:

Definition 1. A subset of features *T* is a testor if and only if when all features are eliminated, except those from *T*, there is not any row of *DM* with only 0´s.

Definition 2. A subset of features *T* is an irreducible testor if and only if *T* is a testor and there is not any other testor *T'* such that $T' \subset T$.

In definition 1, if there is not any row of *DM* with only 0´s it means that there is not a pair of objects from different classes that are similar on all the features of *T*, that is, a testor *T* allows differentiating between objects from different classes.

The number of rows in *DM* could be too large, therefore a strategy to reduce this matrix without losing relevant information for computing irreducible testors was introduced [4].

Definition 3. If *t* and *p* are two rows of *DM*, then *p* is a *sub-row* of *t* if and only if:

a) *t* has 1 everywhere *p* has 1

b) there is at least one column such that *t* has 1 and *p* has 0

Definition 4. A row *t* of *DM* is a *basic row* of *DM* if and only if *DM* does not have any other row *t' such* that *t'* is a *sub-row* of *t*.

Definition 5. The matrix that contains only the *basic rows* of *DM* is called *basic matrix* and is denoted by *BM*.

Let *TT(M)* be the set of all irreducible testors of the Boolean matrix *M*, then [4]:

Proposition 1. *TT(DM)=TT(BM)*.

This proposition indicates that the set of all irreducible testors calculated using *DM* or *BM* is the same. However, *BM* is smaller than *DM* and the construction of *BM* from *DM* is a very fast process, for example, the time for obtaining a *BM* matrix with 48 columns and 32 rows from a *DM* matrix with 48 columns and 193,753 rows, is about 0.21 seconds on a PC with an Intel Pentium 4 processor running at 3.0GHz with 2GB of RAM memory.

There are two kinds of algorithms for computing Irreducible Testors: the *internal scale algorithms* and the *external scale algorithms*. The former analyzes the matrix to find out some conditions that guarantee that a subset of features is an irreducible Testor. The latter looks for Irreducible Testors over the whole power set of features; algorithms that search from the empty set to the whole feature set are call *Bottom-Top* algorithms and algorithms that search from the whole feature set to the empty set are call *Top-Bottom* algorithms. The selected algorithm is a *Bottom-Top external scale algorithm*, called BT. In order to review all the search space, BT codifies the feature subsets as binary *N*-tuples where 0 indicates that the associated feature is not included and 1 indicates that the associated feature is included. For computing testors, BT follows the order induced by the binary natural numbers, this is, from the empty set to the whole feature set. The BT algorithm is as follows:

Step 1.- Generate first no null *N*-tuple $\alpha=(\alpha_1,\alpha_2,...\alpha_N)=(0,...,0,1)$.
Step 2.- Determine if the generated *N*-tuple α is a testor of *BM*.
Step 3.- If α is a testor of *BM*, store it and take $\alpha'=\alpha+2^{N-k}$ where *k* is the index of the last 1 in α.
Step 4.- If α is not a testor of *BM*, determine the first row ν of *BM* with only 0´s in the columns where α has 1´s and generate α' as:

$$\alpha'_j=\begin{cases}\alpha_j & if \quad j<k\\ 1 & if \quad j=k\\ 0 & if \quad j>k\end{cases},$$

where *k* is the index of the last 1 in ν.
Step 5.- Take $\alpha=\alpha'$
Step 6.- If α is not after (1,1,...,1,1) then, go to step 3
Step 7.- Eliminate from the stored testors those which are not irreducible testors.

Step 3 jumps over all the supersets that can be constructed from α by adding 1's (features) after the last 1 in α. For example if $n=9$ and $\alpha=(0,1,1,0,0,1,0,0,0)$ then $k=6$ and the following $2^{N-k}-1=2^{9-6}-1=7$ *N*-tuples represent supersets of the feature set represented by α, which is a testor, and therefore these supersets are testors but they are not irreducible testors, as it can be seen as follows:

α	feature set		
011001000	$\{x_2,x_3,x_6\}$	α	
011001001	$\{x_2,x_3,x_6,x_9\}$		} 7 non-irreducible testors
011001010	$\{x_2,x_3,x_6,x_8\}$		
.....	.....		
011001111	$\{x_2,x_3,x_6,x_7,x_8,x_9\}$		
011010000	$\{x_2,x_3,x_5\}$	$\alpha'=\alpha+2^{N-k}$	

Step 4 jumps over all the sets that can not be a testor according to definition 1, because for any combination of 0's and 1's in those *N*-tuples, the row ν has 0's in those positions. For example if α=(011001001) and ν=(100100000) following step 4, the next *N*-tuple to be verified will be α'=(011100000) which has 1 in at least one position where ν has 1 (x_4). Note that all the *N*-tuples between α and the next *N*-tuple to be verified are not testors, because of ν, as it can be seen as follows:

ν	α	feature set	
100100000	011001001	$\{x_2,x_3,x_6,x_9\}$	α
	011001010	$\{x_2,x_3,x_6,x_8\}$	} 31 non-testors
	011001011	$\{x_2,x_3,x_6,x_8,x_9\}$	
	.....	.....	
	011011111	$\{x_2,x_3,x_5,x_6,x_7,x_8,x_9\}$	
	011100000	$\{x_2,x_3,x_4\}$	α'

In step 4, the jump will be bigger if *k* is smaller. For this reason the columns and rows of *BM* are sorted is such way that the columns containing more 0´s are placed on the right and the rows with more 0´s are placed upper. It is important to remark that interchanging columns or rows will not affect the result of computing all irreducible testors, just the information about which column corresponds to which feature must be preserved [5], and that the process for reorganizing the rows and columns of *BM* is very fast.

3 Proposed architecture

The process of deciding if an *N*-tuple is a testor of *BM* involves comparing the candidate against each one of the *BM*'s rows. For software-based implementations, this presents a big disadvantage, in particular for large matrices with many rows. The proposed hardware architecture exploits the parallelism inherent in the BT algorithm and evaluates whether a candidate is a testor or not in a single clock cycle. It is composed by two main modules as seen in Fig. 1. The BM module stores the input matrix and includes logic to decide if an input *N*-tuple is a testor. The candidate generator module produces the candidates to be evaluated by the BM module. To calculate the next candidate according to the BT algorithm, the architecture feedbacks the result of evaluating a candidate to the generator module that will generate the next candidate as specified by the BT algorithm. This process does not introduce latency, thus the architecture is capable of evaluating a candidate in a single clock cycle.

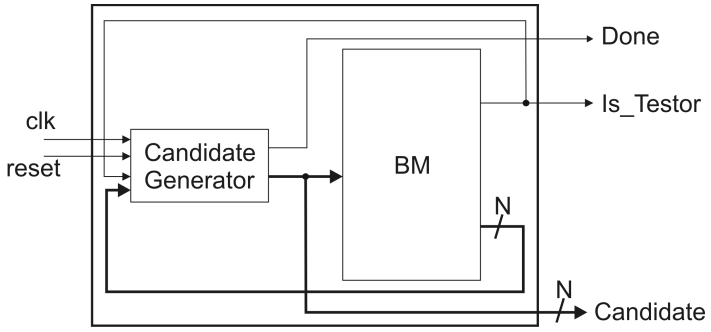

Fig. 1. Top-Level Architecture

The BM module is composed of *M* sub-modules named V_x, as shown in Fig. 2. Each V_x module contains a row (*N* bits) of the *BM* matrix and logic to perform testor evaluation (Fig. 3). To decide if an *N*-tuple is a testor, a bitwise AND operation is performed between the constant stored in each V_x module and the current candidate. If at least one bit of the AND operation result is TRUE, then the output *is_testor* of that particular V_x sub-module will be TRUE, and if the outputs of all V_x sub-modules are TRUE, then the output *is_testor* of the BM module will be TRUE, which means that the candidate is declared a testor of *BM*.

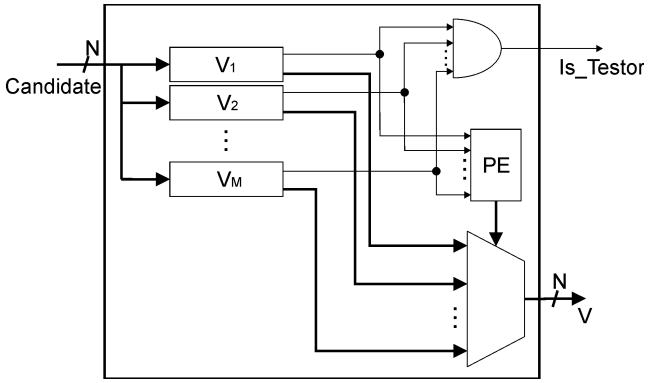

Fig. 2. BM module

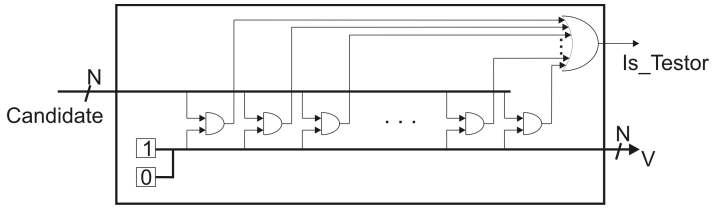

Fig. 3. Vx sub-module

When a candidate fails to be a testor of *BM*, the output *V* of the BM module contains the value of the row closest to the top that caused the test to fail. If the candidate is declared as testor, the output *V* is just ignored. The value of *V* is obtained by using the output of a priority encoder as the select signal of a multiplexer that can select among all the rows of *BM*. This is similar to having a read address in a register file to access the value stored in a particular row.

The candidate generator module uses the feedback from the BM module to calculate the next candidate to be evaluated. As specified by the BT algorithm, there are two ways of generating the next candidate according to the evaluation result of the previous candidate. The candidate generator module consists of two sub-modules, the first sub-module (jump_1) generates the next candidate when the previous candidate is a testor and the second sub-module (jump_2) generates the next candidate when the previous candidate fails to be a testor. The next candidate is selected by a multiplexer according to the evaluation result of the previous candidate (Fig.4).

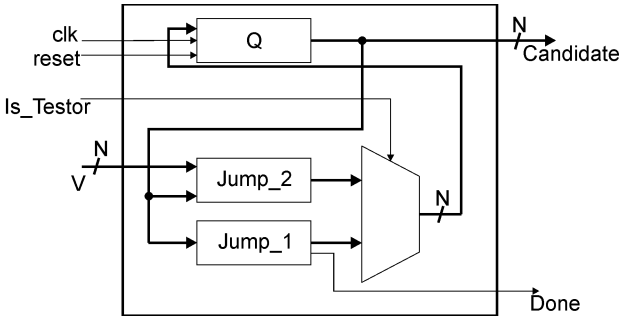

Fig. 4. Candidate generator module

Fig. 5 shows the jump_1 sub-module. It uses a priority encoder to obtain the index of the last '1' in the previous candidate value. The next candidate value is obtained by adding 2^{N-k} to the previous candidate as indicated by the step 3 of the BT algorithm.

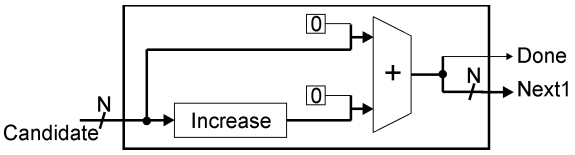

Fig. 5. Jump_1 sub-module

Fig. 6 shows the jump_2 sub-module. Besides the value of the previous candidate, it uses an input *V* that contains the value of the row of *BM* that caused the previous candidate not to be a testor. A priority decoder obtains the index *k* of the last '1' of *V*. By taking the value of the previous candidate, the next candidate is obtained by letting all bits to the left of the k^{th} position unchanged, the bits to the right are changed to '0', and the k^{th} bit is set to '1'. See step 4 of the algorithm.

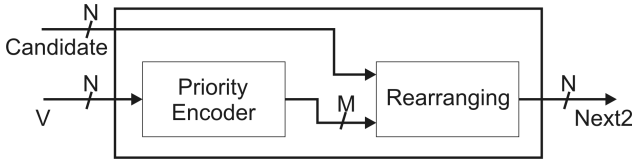

Fig. 6. Jump_2 sub-module

4 FPGA Implementation and Results

The proposed architecture was modeled in the VHDL Hardware Description Language under a structural approach. The VHDL model of the proposed architecture is fully parameterizable in terms of the matrix dimensions (*N*,*M*). The VHDL model was simulated and validated both functional and post-synthesis with ModelSim v6.0. The VHDL model was synthesized with Xilinx ISE v9.0 targeted for a medium size state-of-the-art Virtex-II Pro XC2VP30 FPGA device from Xilinx [6]. The use of the FPGA technology was chosen because it provides a rapid prototyping platform and is specially suited for implementing algorithms based on bit level operations.

The design was implemented on XtremeDSP Development Kit for Virtex-II Pro from Xilinx [7]. This board allows performing hardware-in-the-loop type of simulation using the PCI bus [8]. Although the synthesis results for all test cases show that the architecture can operate in excess of 50MHz, for the purpose of a fair comparison with the work reported in [1], the following results were calculated considering 50MHz as the operating frequency.

In order to show the performance of the proposed architecture, it was compared against the brute force implementation reported in [1], and software implementations of the BT algorithm and CT algorithms [9]. For experimentation purposes, basic matrices from 20 to 30 columns by 100 rows were randomly generated. Figure 7 shows the resulting processing times and Table 1 shows additionally the percentage of candidates tested by the BT algorithm.

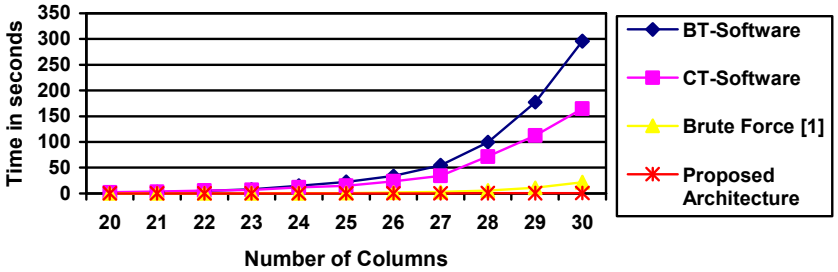

Fig. 7. Processing time in seconds for randomly generated data.

Table 1. Processing times in seconds for different implementations of BT and CT algorithms

Matrix	Software implementations		Hardware implementations		Percentage of candidates tested - BT algorithm
	BT	CT	[1] (Brute Force)	Proposed Architecture	
20x100	1.55	2.06	0.021	0.004	17.79
21x100	2.23	3.16	0.042	0.006	14.59
22x100	4.73	5.11	0.084	0.009	11.38
23x100	7.61	6.72	0.168	0.016	9.54
24x100	14.50	11.55	0.335	0.027	7.94
25x100	22.56	14.78	0.671	0.047	7.08
26x100	33.80	23.43	1.342	0.079	5.86
27x100	54.68	34.14	2.684	0.140	5.20
28x100	99.34	71.75	5.369	0.209	3.88
29x100	177.25	112.33	10.737	0.316	2.94
30x100	295.92	164.84	21.475	0.517	2.41

The processing time t for a specific matrix is given by:

$$t = \left(\frac{2^N}{f}\right)\left(\frac{c}{100}\right)$$

where f is the clock frequency of the architecture and c is the percentage of candidates tested. Note that the value of c is data dependent, i.e. it varies according to each *BM* matrix.

These experiments show that the proposed architecture allows running BT 570 times faster that the software implementation and 318 times faster than software implementation of CT for N=30 and M=100. The improvement against the brute force for this matrix is 40X, this is because the percentage of candidates tested by BT is ~2.4%. The software implementations were executed on a PC with an Intel Pentium 4 processor running at 3.0GHz with 2GB of RAM memory.

The architecture has been designed to process variable sizes of the *BM* matrix. The maximum size of the matrix that can be implemented is only limited by the available resources on the target FPGA. The hardware resources utilization is proportional to the size of the matrix, i.e. the total number of elements NxM. Table 2 summarizes the FPGA resources utilization for large randomly generated matrices of 100 columns by 100 to 300 rows. Note that the number of columns was set to 100 as most of the practical problems that use testor theory will have at most this number of columns.

These results show that the hardware resources required to implement the architecture are proportional to the total number of elements in the *BM* matrix, e.g the 100x200 matrix requires twice as much slices than the 100x100 matrix. Note that even for the larger matrix (30,000 elements), the number of slices used is around 71% of the total available, which means that matrices up to around 40,000 elements can be

processed with this modest size FPGA device. In this approach the matrix is declared as constant, which means that for any new matrix to be processed the architecture has to be resynthesized and the FPGA configured. However, this process takes only a few extra seconds, but as no flip-flops are needed to store the matrix, the FPGA resources utilization is considerably reduced.

Table 2. Hardware resources utilization for large matrices

Matrix	Frec. (MHz)	Slices	Flip-Flops
100x100	87.73	3,391 (24%)	601 (2%)
100x150	82.37	5,064 (36%)	801 (2%)
100x200	80.49	6,567 (47%)	966 (3%)
100x250	75.99	8,060 (58%)	1,288 (4%)
100x300	77.00	9,778 (71%)	1,704 (6%)

5 Discussion

The proposed architecture provides higher processing performance than the previously reported hardware implementation as it now performs the complete BT algorithm. In spite of the added functionality, the architecture still is capable of performing the number of operations needed to test if an *N*-tuple is a testor of *BM* in a single clock cycle. Thus the performance improvement is directly related to the percentage of candidates tested (*c*), which in turn heavily depends on the values in the *BM* matrix. However, for real data this improvement could be significantly higher.

Experiments show that the proposed architecture allows computing testors faster than software implementations of the BT and CT algorithms, with improvements in the range of 2 orders of magnitude. However, for very large real data this improvement could be significantly higher. Additionally, an advantage of the proposed architecture is that it requires only one clock cycle to test each candidate independently of the number of rows, whereas software implementations processing time will significantly increase for matrices with a large number of rows.

It is important to highlight that the proposed architecture computes testors and the decision about which of them are irreductible has to be taken after each testor is found, this applies also to software-based implementations.

6 Conclusions

In this work, an efficient hardware implementation of the BT algorithm for computing testors is presented. The high performance of the proposed architecture is feasible due to the high level of parallelism implicit in the BT algorithm that can be efficiently implemented on a FPGA. The architecture is capable of evaluating a candidate in a single clock cycle for any *BM* matrix, regardless of the number of columns and rows, being the only limitation the size of the used FPGA device. The architecture provides

a good trade-off between performance and hardware resource utilization and it is suitable to be used as a high performance processing module in a hardware-in-the-loop approach.

Even though the proposed architecture offers an improvement compared with a previously reported hardware implementation, further improvements, such as testing two or more candidates per iteration, are still possible. Also, because resource requirements are relatively small, a scheme where the processing core can be replicated will also be explored; this will effectively reduce the processing times proportionally to the number of processing cores that can be accommodated on the FPGA device. The final goal is to build a high performance flexible hardware/software platform for computing testors. On this direction, we are currently exploring the implementation of hardware architectures for more sophisticated algorithms like LEX [10].

References

1. Cumplido, R., Carrasco-Ochoa A., Feregrino, C.: On the Design and Implementation of a High Performance Configurable Architecture for Testor Identification. In: Martínez-Trinidad J. F., Carrasco-Ochoa A., Kittler J. (eds.) CIARP06. LNCS, vol. 4225, pp. 665--673. Springer, Cancún, México (2006)
2. Dmitriev, A.N., Zhuravlev, Y.I., Krendeliev, F.P.: About Mathematical Principles of Objects and Phenomena Classification. Diskretni Analiz, vol. 7, pp. 3--15 (1966) (In Russian)
3. Martínez-Trinidad, J. F., Guzmán-Arenas, A.: The Logical Combinatorial Approach to Pattern Recognition an Overview through Selected Works. Pattern Recognition, vol. 34, No. 4, pp. 741--751 (2001)
4. Lazo-Cortes, M., Ruiz-Shulcloper, J., Alba-Cabrera, E.: An Overview of the Evolution of the Concept of Testor. Pattern Recognition, vol. 34, No. 4, pp. 753--762 (2001)
5. Sánchez Díaz, G., Lazo Cortés, M.: Modifying BT Algorithm for Improving its Runtimes. Revista Ciencias Matemáticas, vol. 20, No. 2, pp. 129--136 (2002) (In Spanish)
6. Virtex-II Pro Data sheet, www.xilinx.com.
7. XtremeDSP Development Kit for Virtex-II Pro, www.xilinx.com.
8. Gomez, M.: Hardware-in-the-Loop Simulation. Embedded Systems Programming, vol. 14, No. 13 (2001)
9. Bravo Martinez, A.: Algorithm CT for Calculating the Typical Testors of k-valued Matrix. Revista Ciencias Matematicas, vol. 4, No. 2, pp.123--144 (1983) (In Spanish).
10. Santiesteban-Alganza Y. and Pons-Porrata A.: LEX: A New Algorithm for Computing Typical Testors. Revista Ciencias Matemáticas, Vol. 21, No. 1, pp. 85--95 (2003) (In Spanish)