Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Hardware–software platform for computing irreducible testors

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

Computer Science Department, National Institute for Astrophysics, Optics and Electronics, Sta. Ma. Tonanzintla, Puebla 72840, Mexico

ARTICLE INFO

Keywords:
Feature selection
Testor theory
Custom architectures
FPGAs

ABSTRACT

In pattern recognition, feature selection is a very important task for supervised classification. The problem consists in, given a dataset where each object is described by a set of features, finding a subset of the original features such that a classifier that runs on data containing only these features would reach high classification accuracy. A useful way to find this subset of the original features is through testor theory. A testor is defined as a subset of the original features that allows differentiating objects from different classes. Testors are very useful particularly when object descriptions contain both numeric and non-numeric features. Computing testors for feature selection is a very complex problem due to exponential complexity, with respect to the number of features, of algorithms based on testor theory. Hardware implementation of testor computing algorithms helps to improve their performance taking advantage of parallel processing for verifying if a feature subset is a testor in a single clock cycle. This paper introduces an efficient hardware–software platform for computing irreducible testors for feature selection in pattern recognition. Results of implementing the proposed platform using a FPGA-based prototyping board are presented and discussed.

1. Introduction

Reconfigurable computing based on the combination of conventional microprocessors and field programmable gate arrays (FPGAs), has become increasingly popular for implementing special-purpose hardware to accelerate complex tasks. Usually an FPGA-based implementation is embedded in a PC or workstation, which drives its activity and manages the results. Following this trend, we developed an efficient hardware–software platform for computing irreducible testors (Lazo-Cortés, Ruiz-shulcloper, & Alba-cabrera, 2001) for feature selection in pattern recognition (Al-Ani, 2009; Chen, Tseng, & Hong, 2008; Jain & Zongker, 1997; Kwan & Choi, 2002; Liu & Setiono, 1998).

The feature selection problem in pattern recognition consists in, given a dataset where each object is described by a set of features, finding a subset of the original features such that a classifier that runs on data containing only these features would reach higher classification accuracy. This procedure can reduce not only the cost of recognition by reducing the number of features to be collected, but in some cases it can also provide better classification accuracy. For this task, a higher performance with lower computational effort is expected (Kwan & Choi, 2002). Several algorithms have been proposed for feature selection, however, most of them were developed for numeric features (Guyon & Elisseeff, 2003; Jain & Zongker, 1997). We chose BT, an algorithm based on testor theory, which can be applied on datasets described with both numeric and non-numeric features, even when there are missing data. Although the theoretical aspect of computing irreducible testors is advanced (Asaithambi & Valev, 2004; Djukova, 2005; Kudryavtsev, 2006; Martínez-Trinidad & Guzmán-Arenas, 2001; Valev & Sankur, 2004), there are not practical hardware implementations reported previously, excepting our previous works.

In our first work, an architectural design based on a brute force approach for computing testors was proposed (Cumplido, Carrasco, & Feregrino, 2006). In this first approach, each candidate was generated by a counter that incremented its value by 1 on each iteration. The architecture is able to evaluate if a candidate is a testor in a single clock cycle, however, the architecture did not exploit the characteristics of a particular data set that could allow to significantly reduce the number of candidates tested. The next step in our architectural design (Rojas, Cumplido, Carrasco-Ochoa, Feregrino, & Martnez-Trinidad, 2007) was the implementation of BT algorithm for computing testors where a candidate generator that jumps over unnecessary candidates allows reducing the number of comparisons needed in the brute force approach. These two previous works compute the whole set of testors, however for pattern recognition applications where testor theory can be applied, it is important to obtain only testos that are irreducible. Thus, as the next step in our design, this work proposes a hardware–software platform for computing only irreducible testors. This platform consists of the combination of a specialized hardware architecture that is implemented on a commercial FPGA-based prototyping board and a host application running on a PC. The architecture

* Corresponding author.
E-mail address: rcumplido@inaoep.mx (R. Cumplido).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.07.004

implements the BT algorithm, as in Rojas et al. (2007), but it also includes a new module that eliminates most of the testors that are not irreducible before transferring them to the host application for final processing.

The intensive computational requirements due to the exponential complexity, with respect to the number of features, of the testor theory based algorithms can be met by a combination of technological improvements and efficient hardware architectures based on parallel computational models. Specific parallel architectures can be designed to exploit the parallelism found in the irreducible testor computing algorithms. Further optimizations such as incremental processing and the use of multiple processing elements are also possible.

2. Computing irreducible testors

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 Dmitriev, Zhuravlev, and Krendeliev (1966). They defined a testor as a subset of features that allows differentiating objects from different classes. Testors are quite useful, especially when object descriptions contain both numeric and nonnumeric features, and maybe they are incomplete (mixed incomplete data) (Martínez-Trinidad & Guzmán-Arenas, 2001).

Let *TM* be a training matrix with *K* objects described through *N* features of any type ($x_1,\ldots,x_N$) 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 *TM* belonging to different classes. *DM* has *N* columns and *M* rows, where $M \gg K$.

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 from *DM*, except those from *T*, there is not any row of *DM* with only 0s.

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 by Lazo-Cortés et al. (2001).

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

Table 1
N-tuples omitted by step 3.

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

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

This proposition indicates that the set of all irreducible testors calculated using *DM* or *BM* is the same (Lazo-Cortés et al., 2001). 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 s on a PC with an Intel Centrino Duo processor running at 1.6 GHz, with 1024 MB of RAM.

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 called *Bottom–Top* algorithms and algorithms that search from the whole feature set to the empty set are called *Top–Bottom* algorithms. The selected algorithm is a *Bottom–Top external scale algorithm*, called BT (Sánchez-Díaz & Lazo-Cortés, 2002). This algorithm was selected because of its simplicity and inherent parallelism which can be easily exploited in a hardware architecture.

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 (some examples can be seen in Table 1). 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:

1. Generate first no null *N*-tuple
$$\alpha = (\alpha_1,\ldots,\alpha_N) = (0,\ldots,0,1).$$
2. Determine if the generated N-tuple α is a testor of *BM*.
3. If α is a testor of *BM*, store it and take
$$\alpha' = [(\alpha)_b + 2^{N-k}]_{tuple},$$

where *k* is the index of the last 1 in α and $(\alpha)_b$ represents the natural number corresponding to α and $[(\alpha)_b + 2^{N-k}]_{tuple}$ represents the *tuple* associated with the natural number $(\alpha)_b + 2^{N-k}$.

4. If $\alpha$is not a testor of *BM*, determine the first row v of *BM* with only 0's in the columns where α has 1's and generate α' as:
$$\alpha'_j = \begin{cases} \alpha_j & j < k, \\ 1 & j = k, \\ 0 & j > k, \end{cases}$$

where *k* is the index of the last 1 in v.

5. Take $\alpha = \alpha'$.
6. If $\alpha$is not after (1,1,...,1,1) then, go to 3.
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

Table 2
N-tuples omitted by step 4.

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

Table 3
Basic matrix with $N = 5$ columns and $M = 2$ rows.

BM	x_1	x_2	x_3	x_4	x_5
v_1	0	0	1	0	0
v_2	0	1	0	0	0

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 in Table 1.

Step 4 jumps over all the sets (*N*-tuples) 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 v of *BM* has 0's in those positions. For example if $\alpha = (011001001)$ and $v = (100100000)$ following step 4, the next *N*-tuple to be verified will be $\alpha' = (011100000)$ which has 1 in at least one position where v has 1 (in this case x_4). Note that all the *N*-tuples between α and the next *N*-tuple to be verified are not testors, because of v, as it can be seen in Table 2.

For example, consider the basic matrix of Table 3. Although not all *N*-tuples are generated and evaluated, Fig. 1 shows all the search space, in order to illustrate the jumps that BT algorithm would perform.

In Fig. 1, only the *N*-tuples marked with "*" are verified by BT in order to compute the irreducible testors of the BM of Table 1. The shadowed *N*-tuples do not need to be neither generated nor verified. At the end, only 01100 and 11100 are stored, and from them

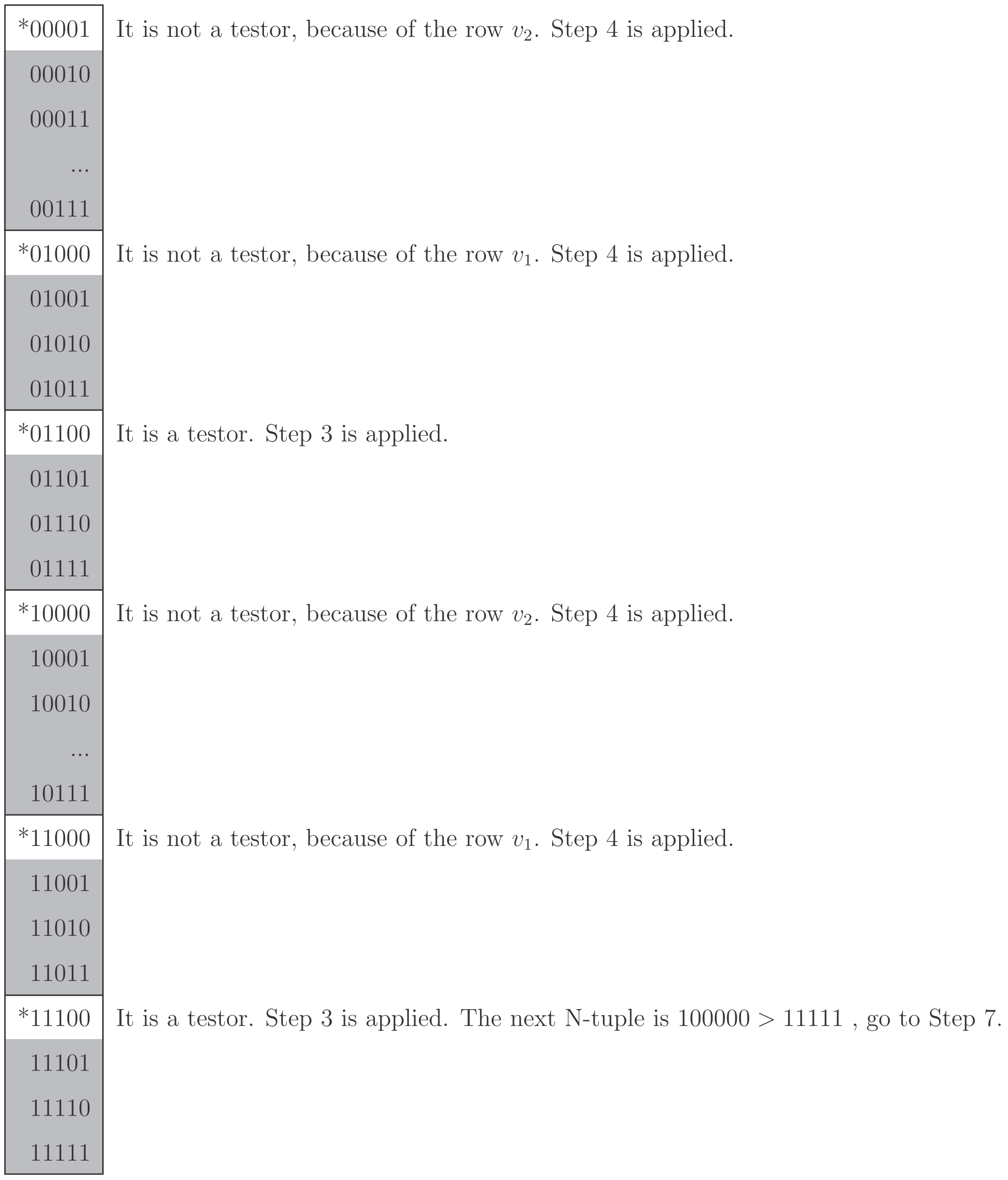

*00001
It is not a testor, because of the row v_2. Step 4 is applied.
00010
00011
...
00111
*01000
It is not a testor, because of the row v_1. Step 4 is applied.
01001
01010
01011
*01100
It is a testor. Step 3 is applied.
01101
01110
01111
*10000
It is not a testor, because of the row v_2. Step 4 is applied.
10001
10010
...
10111
*11000
It is not a testor, because of the row v_1. Step 4 is applied.
11001
11010
11011
*11100
It is a testor. Step 3 is applied. The next N-tuple is 100000 > 11111 , go to Step 7.
11101
11110
11111

Fig. 1. Route of BT through the search space for computing the irreducible testors of the basic matrix of Table 3.

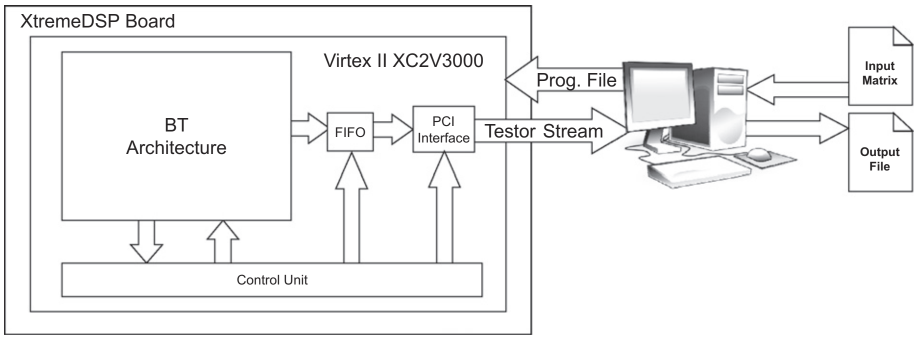


Fig. 2. Proposed hardware–software platform.

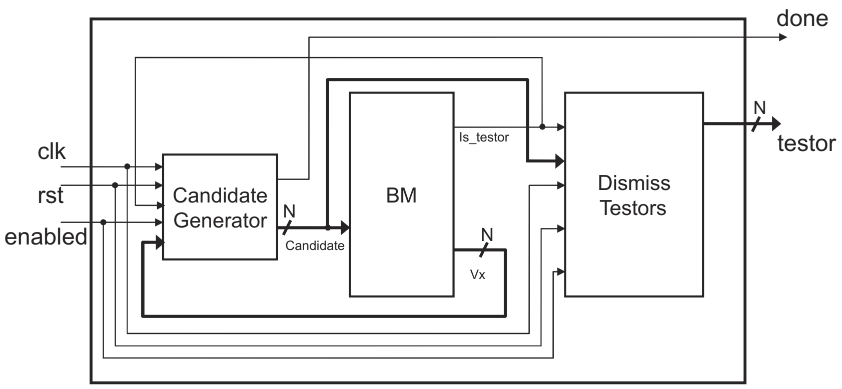


Fig. 3. BT architecture.

only 01100 is an irreducible testor, therefore 11100 is eliminated by step 7. Finally, the set of all irreducible testors, for this example, is $\{\{x_2,x_3\}\}$.

3. Proposed platform

Since algorithms for computing all irreducible testors have exponential complexity, with respect to the number of columns in *BM*, software based implementations do not provide reasonable performance for practical problems. An interesting alternative is to migrate to custom hardware architectures based on programmable logic to take advantage of the parallelism inherent in this type of algorithms. This work is a continuation of our previous work (Rojas et al., 2007) and reports the development of a hardware–software platform for computing irreducible testors using the BT algorithm. The proposed platform is shown in the Fig. 2. A description of all the platform modules is given in the next sections.

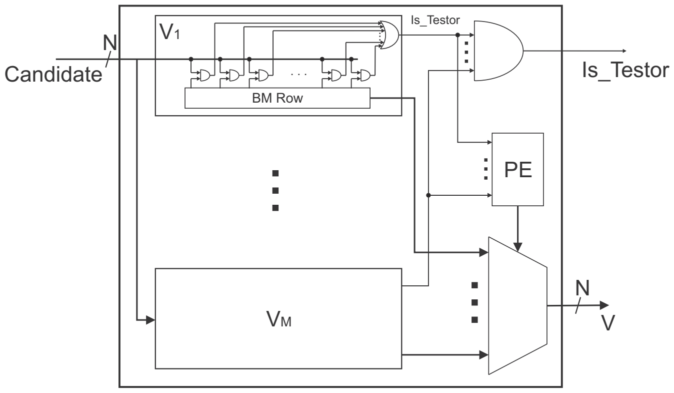


Fig. 4. BM module.

4. Hardware 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-only implementations, this is 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 three main modules as seen in Fig. 3. The BM module stores the input matrix and includes logic to decide if an *N*-tuple is a testor. The candidate generator module produces the candidates (*N*-tuples) to be evaluated by the BM module. In order to calculate the next candidate according to the BT algorithm, the architecture feedbacks the evaluation result of the previous candidate to the generator module, this allows to drastically reduce the number of candidates tested thus the number of iterations needed by the algorithm. At this point, the architecture is able to obtain all testors of *BM*, however since only irreducible testors are of interest, a final hardware processing module eliminates most of the testors that are not irreducible before sending the remaining testors to the software for final processing. The dismiss module exploits the way consecutive testors are obtained. If a testor is a superset of at least one previous testor, it is not an irreducible testor, thus it is eliminated. This final process does not introduce delays, thus the architecture is still capable of evaluating a candidate in a single clock cycle.

The BM module is composed of *M* sub-modules named V_x, as shown in Fig. 4. Each V_x module contains a row (*N* bits) of the

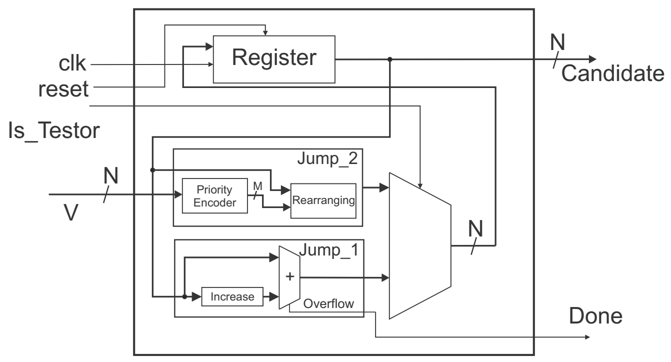


Fig. 5. Candidate generator module.

BM matrix and logic to perform testor evaluation. 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*.

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 failure. 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 one. The candidate generator module (Fig. 5) consists of two sub-modules, the first sub-module (jump_1) generates the next candidate when the previous one is a testor and the second sub-module (jump_2) generates the next candidate when the previous one fails to be a testor. The next candidate is selected by a multiplexer according to the evaluation result of the previous candidate.

The jump_1 sub-module uses a priority encoder to obtain the index, *k*, 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.

Besides the value of the previous candidate, the jump_2 sub-module 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.

The dismiss testors (DT) module (Fig. 6) discards some candidates that were evaluated as testors in the BM module, but they are supersets of other testors, thus they are not irreducible testors. The DT module has a predefined number of REG_DT sub-modules that store equal number of testors. These sub-modules are initialized with the *N*-tuple 1,...,11. A new testor is evaluated in each sub-module. If the input testor is a superset of any of the stored ones, it must be disposed, else, it is sent to the output *testor* and furthermore, it is stored in the first REG_DT sub-module. Each time a new testor is stored, each REG_DT sub-module passes its value to the next one. As the number of registers within the REG_DT sub-modules is limited, a discarding policy must be implemented to delete a testor once all sub-modules are filled. Because the last testor is more likely to be a superset of one or more of the previously generated testors, the selected discarding policy consists in eliminating the oldest testor stored within the DT module.

On each REG_DT sub-module, a bitwise AND operation is performed between the stored value in the register and the input *Candidate*. The *N*-tuple stored is compared again, this time against the result of the previous operation. If both are equal, it means that the input testor is a superset of the stored one. The output of each REG_DT sub-module always takes the value stored in the register, moving out this value to the next register in case of being necessary.

The used prototyping kit allows partitioning the functionality of the application between software and hardware effectively. It provides a set of functions that allows the user to communicate data between the hardware architecture in the FPGA and the host application running on the PC using high level calls. An interface core provided with the kit offers a communication mechanism to build a simplified interface for connecting the user architecture in the FPGA to the PCI bus using a FIFO memory as buffer. This abstracts the complexities of the communication process. In turn, for the host application, a set of Application Program Interfaces (APIs) provides functions for sending and receiving data from across the PCI (XtremeDSP, xxxx).

Table 4 summarizes experiments made for deciding the appropriate number of registers in the dismiss testors module. As results

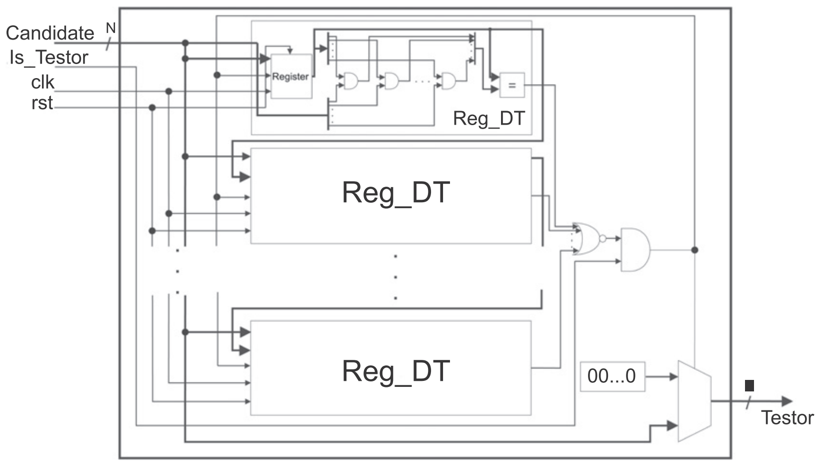
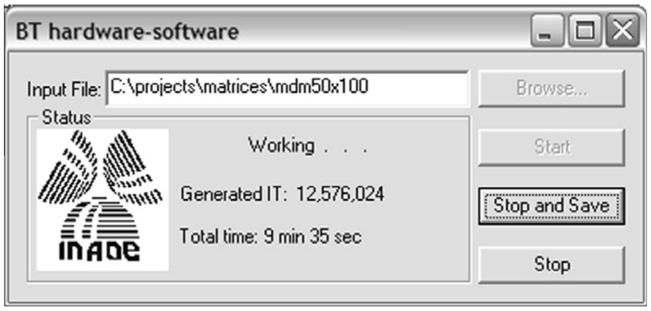


Fig. 6. Dismiss testors module.

Table 4
Remaining testors using the dismiss testors module, with different amount of REG_DT sub-modules.

BM	Received	Amount of registers in the dismiss testors module							
	Testors	8	16	32	64	128	256	512	1024
21	173,957	57,020	36,243	26,014	19,943	15,081	11,934	9,965	7,870
22	280,926	85,624	51,667	36,643	27,302	20,776	16,875	13,978	11,176
23	476,445	134,068	80,296	55,491	39,314	28,679	22,982	18,852	14,800
24	822,454	214,127	121,877	79,440	55,398	40,731	32,869	26,962	21,348
25	1,483,468	352,075	192,316	124,136	84,781	59,736	47,075	37,549	29,356
26	2,506,239	537,179	281,416	176,384	117,335	84,304	67,515	54,513	42,936
27	4,492,643	839,811	443,758	272,190	175,103	123,866	98,442	77,682	61,278
28	6,712,871	1,236,039	637,972	385,418	246,115	170,057	133,463	105,581	84,056
29	9,956,854	1,727,804	892,049	551,305	355,303	247,730	193,879	151,596	120,986


BT hardware-software

Input File: C:\projects\matrices\mdm50x100

Status

Working . . .

Generated IT: 12,576,024

Total time: 9 min 35 sec

Fig. 7. User interface for BT hardware–software platform.

Table 5
Processing time in seconds (broken down for each stage) for 45X100 low, medium and high density matrices.

Stages	Low density		Medium density		High density	
	HW/SW	SW-O	HW/SW	SW-O	HW/SW	SW-O
Load BM	0.031	0.031	0.031	0.031	0.032	0.031
BM arrangement	0.281	0.281	0.281	0.024	0.282	0.01
Files creation	0.032	N/A[1]	0.031	N/A[1]	0.046	N/A[1]
Synthesis	691	N/A[1]	1,032	N/A[1]	930	N/A[1]
IT[2] computing	316	170,830	1,303	51,777	8	0.06
TOTAL	1,008	170,831	2,336	51,778	939	0.11

[1] Not Applicable.
[2] Irreducible testors.

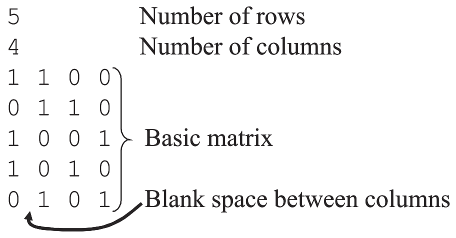

```
5
4
1 1 0 0
0 1 1 0
1 0 0 1
1 0 1 0
0 1 0 1
```

Fig. 8. *BM* input file format.

show, the more REG_DT sub-modules, the higher the percentage of non irreducible testors eliminated. However, as the number of REG_DT sub-modules impacts directly on hardware requirements, we concluded that 16 REG_DT provide a good tradeoff for two reasons: (1) the percentage of non irreducible testors eliminated was always over 79%; and (2) as shown in Table 7 (in Section 6), the hardware requirements are still modest when compared against the rest of the architecture. This number of REG_DT is used for all the experiments.

5. Software description

The software component of the hardware–software platform works as an interface between the user, the computer, and the FPGA board. Fig. 7 shows the user interface, which requests the location of the input file containing the basic matrix, and allows starting and stopping the irreducible testor computing process. The input data file must be in plain text following the format shown in Fig. 8.

The process of deciding if a testor obtained by the architecture is an irreducible testor by the host application is straightforward. It consists in deciding for each new testor, if it is a superset of any previously stored testor, which indicates that the new testor is not an irreducible testor, and it must be discarded. This process starts once the first testor is received by the host application, from that moment until the last testor is received, both the hardware architecture and the host application work alternately. Thus the processing times reported in Table 5 for the irreducible testors processing is mostly the time spent by the hardware architecture to produce the testors, including the time needed for deciding which of the testors obtained by the architecture are indeed irreducible testor. The platform is able to extract all irreducible testors independently of the number of Reg_DTs. The purpose of the DT module is to reduce the number of testors sent to the host application in order to avoid having a bottleneck in the system. The amount of the reduction can be seen in Table 4.

The testor computing process begins when the user clicks the *Start* button. First the basic matrix is read from the input file. Once the matrix is loaded into the computer's memory, the basic matrix is reorganized by swapping rows and columns in such a way the resulting basic matrix is close to be a lower-left-triangular matrix, this with the purpose of optimizing the jumps that BT algorithm performs. Afterward, in order to complete the project files, three VHDL files are generated and used to create a project for ISE (implementation tool from Xilinx), which produces the programming file for the FPGA device.

In the next stage, the interaction between hardware and software is started. First, the board and the FPGA are located, then the device is programmed with the bit-file obtained from the previous stage, and both, the board and the device are initialized. Now, the hardware architecture starts computing testors. Each testor is temporally stored in a FIFO memory, when there are 512 elements in the FIFO they are sent to the software through the PCI bus. The software stores the incoming testor set into a buffer. When the buffer is full, the hardware is paused while a selection process empties the buffer by choosing and storing only irreducible testors. After this, the hardware architecture and the software alternate until the last candidate is verified.

6. Evaluation

In order to show the performance of the proposed hardware–software platform, it was compared against a software-only implementation of the BT algorithm. For experimentation purposes, three kinds of basic matrices were randomly generated. Each type containing different amount of 1's per row:

(1) High density matrices: between 90% and 98%.
(2) Low density matrices: between 4% and 12%.
(3) Medium density matrices: between 47% and 53%.

High density matrices represent datasets with well separated classes, so it is very easy to find out subsets of features that allow differentiating objects from different classes; in consequence it is easy to compute all irreducible testors. On the other hand, low density matrices represent datasets where objects from different classes are very similar, which results in a low amount of 1's; therefore it is difficult to find a feature subset that allows to distinguish between objects belonging to different classes and computing all irreducible testors would be more difficult. Medium density matrices contain a balanced amount of 1's and 0's, and they come from datasets where the classes are not well separated, but there still be enough difference between objects from different classes. These are difficult matrices for computing all irreducible testors, and in practice, most of the interesting pattern recognition problems produce this kind of matrices.

Several basic matrices of different sizes were randomly generated for each density, from 35 to 50 columns and all of them with 100 rows. On the hardware–software platform the runtimes for the following stages: load basic matrix to memory, matrix arrangement, project files creation, synthesis of ISE project, and irreducible testor computing (with a hardware frequency of 40 MHz), were measured. Figs. 9–11 show graphs of the whole processing time for high, medium and low density matrices respectively. In these graphs, it is possible to see that the proposed platform obtains better performance than the software-only implementation of BT for low and medium density matrices. For high density matrices the software-only implementation required less time, because computing irreducible testors is very easy for this kind of matrices, and the synthesis time required for the proposed platform is much higher than the irreducible testor computing time.

The processing time t, of the last stage of the proposed platform, for a specific matrix is given by:

$$t = \left(\frac{2^N}{f}\right)\left(\frac{c}{100}\right), \qquad (1)$$

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 for each basic matrix, *BM*.

It is important to notice that the processing time for computing irreducible testors does not only depend on the size and density of the *BM*, but also on the distribution of 0's and 1's inside the matrix. This assertion can be appreciated in points 43–45 of Fig. 11.

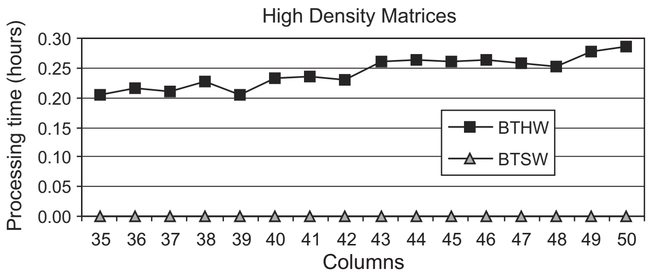


Fig. 9. Whole processing time in hours for high density matrices.

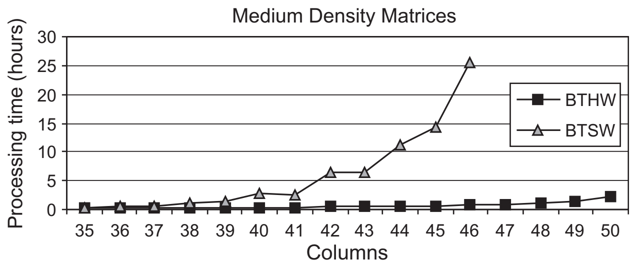


Fig. 10. Whole processing time in hours for medium density matrices.

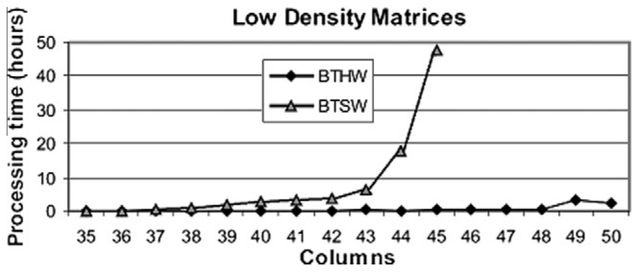


Fig. 11. *BM* Whole processing time in hours for low density matrices.

Table 6
Synthesis summary of FPGA Resources utilization a operating frequency for the architecture targeted for a Virtex-II XC2V3000 FPGA device ($N = 100$, $M = 325$).

Number of slices	11,137 (77%)
Number of 4-input LUTs	2,484 (8%)
Number of flip-flops	2,1262 (74%)
Maximum clock frequency	55.510 MHz

Table 5 shows the processing time for each stage of the data flow, for 45X100 low, medium, and high density matrices. This table shows that the proposed platform allows running BT 169 times faster that the software-only implementation, for a 45X100 low density matrix and 22 times faster for the medium density matrix of the same size. Moreover, taking into account only the last stage of the data flow, the improvement over the software is 540X for the low density matrix and 39X for the medium density one. The only type of matrices where the proposed platform does not perform better that the software-only BT implementation was the high density ones, this is because of computing irreducible testors is very fast for this kind of matrices. The software-only implementation of BT was executed on a PC with an Intel Centrino Duo processor running at 1.6 GHz, with 1024 MB of RAM.

The proposed platform 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 specific FPGA board. For example, for the Virtex-II XC2V3000 from Xilinx (Virtex-II, xxxx) embedded on an XtremeDSP board (XtremeDSP, xxxx), the biggest medium density matrix that can fit into the FPGA is about 100X325. Table 6 summarizes the resource utilization for this matrix.

Finally, Table 7 summarizes the resource utilization for a 50X100 matrix for different buffer sizes on the Dismiss Testor module, all for the same device. Although 8 and 32 are good choices, 16 results in a better trade-off between resource utilization (small when compared against the whole architecture), and the non irreducible testors eliminated (always over 79%, as Table 4 shows).

Table 7
FPGA Resource utilization for Virtex II XC2V3000 for $N = 50$ and $M = 100$ (Whole architecture vs Dismis testors module).

Resources	8 REG_DT		16 REG_DT		32 REG_DT	
	Whole Architecture	Dismiss Testors	Whole architecture	Dismiss Testors	Whole Architecture	Dismiss Testors
Slices	2365	338	2676	679	3331	1342
Flip-flops	1024	400	1451	800	2236	1600
LUTs	4256	258	4472	472	4871	892

7. Discussion

The proposed hardware–software platform provides higher processing performance than the software-only implementation of the BT algorithm for two of the three kinds of matrices used in the experimentation. This behavior is possible because the hardware component of the proposed platform is capable of testing if an *N*-tuple is a testor of a *BM* in a single clock cycle, independently of the number of columns and rows, whereas software-only implementation processing time will significantly increase for matrices with a large number of rows.

Moreover, the performance improvement is directly related to the percentage of candidates tested (c), which heavily depends on density and distribution of the values into the *BM* matrix. Proofs for this dependence are the resulting processing times of high density matrices. This kind of matrices has a high amount of 1's, thus there are low chances to find a row with only 0's when an *N*-tuple is evaluated, which results in a great reduction in the number of operations made by the BT algorithm. This occurs in the software-only as well as in the hardware–software implementations, but the proposed platform also needs to synthesize the specific architecture. Therefore, the best choice for computing all irreducible testors for high density matrices is the software-only implementation of BT algorithm. However, most of the real world pattern recognition problems produce medium density matrices, where using the proposed platform is the best choice for computing all irreducible testors.

Experiment results show that the proposed platform allows computing irreducible testors faster than the software-only implementation of the BT algorithm, with improvements in the range of 2 orders of magnitude. However, for very large real data this improvement could be significantly higher.

8. Conclusions

The high performance of the proposed platform is feasible due to the high level of parallelism implicit in the BT algorithm which can be efficiently implemented on an FPGA. The proposed architecture is capable of evaluating a testor candidate in a single clock cycle for any *BM* matrix, regardless of the number of columns and rows, the only limitation being the size of the FPGA device used. 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 (Gómez, 2001).

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 time, in proportion to the number of processing cores that can be accommodated on the FPGA device.

References

Al-Ani, A. (2009). A dependency-based search strategy for feature selection. *Expert Systems with Applications, 36*, 12392–12398.
Asaithambi, A., & Valev, A. (2004). Construction of all non-reductible descriptors. *Pattern Recognition, 37*, 1817–1823.
Chen, w. S., Tseng, S. S., & Hong, T. P. (2008). An efficient bit-based feature selection method. *Expert Systems with Applications, 34*, 2858–2869.
Cumplido, R., Carrasco, A., & Feregrino, C. (2006). On the design and implementation of a high performance configurable architecture for testor identification. *Lectures Notes on Computer Science, 4225*, 665–673.
Djukova, E. V. (2005). On the number of irreducible coverings of an integer matrix. *Computational Mathematics and Mathematical Physics, 45*, 903–908.
Dmitriev, A. N., Zhuravlev, Y. I., & Krendeliev, F. P. (1966). About mathematical principles of objects and phenomena classification. *Diskretni Analiz, 7*, 3–17.
Gómez, M. (2001). Hardware-in-the-loop simulation. *Embedded Systems Programming, 14*, 38–49.
Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.
Jain, A., & Zongker, D. (1997). Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 9*, 153–158.
Kudryavtsev, V. B. (2006). Test recognition theory. *Discrete Applied Mathematics, 16*, 319–350.
Kwan, N., & Choi, C. H. (2002). Input feature selection for classification problems. *IEEE Transactions on Neural Networks, 13*, 143–159.
Lazo-Cortés, M., Ruiz-shulcloper, J., & Alba-cabrera, E. (2001). An overview of the evolution of the concept of testor. *Pattern Recognition, 34*, 753–762.
Liu, H., & Setiono, R. (1998). Some issues on scalable feature selection. *Expert Systems with Applications, 15*, 333–339.
Martínez-Trinidad, J. F., & Guzmán-Arenas, A. (2001). The logical combinatorial approach to pattern recognition an overview through selected works. *Pattern Recognition, 34*, 741–751.
Rojas, A., Cumplido, R., Carrasco-Ochoa, J. A., Feregrino, C., & Martnez-Trinidad, J. f. (2007). FPGA based architecture for computing testors. *Lectures Notes on Computer Science, 4881*, 188–197.
Sánchez-Díaz, G., & Lazo-Cortés, M. (2002). Modifying BT algorithm for improving its runtimes. *Revista Ciencias Matemáticas, 20*, 129–136.
Valev, V., & Sankur, B. (2004). Generalized non-reducible descriptors. *Pattern Recognition, 37*, 1809–1815.
Virtex-II Pro Data Sheet Version 4.7. Xilinx Inc.
XtremeDSP Development Kit Pro User Guide Version 1.0. Xilinx Inc.