International Journal of Innovative
Computing, Information and Control
Volume **11**, Number **2**, April **2015**

 ISSN 1349-4198
pp. 641–656

DESIGN AND IMPLEMENTATION OF ROUGH SET CO-PROCESSOR ON FPGA

Kanchan S. Tiwari[1,2] and Ashwin G. Kothari[1]

[1]Electronics and Communication Engineering Department
Visvesvaraya National Institute of Technology
South Ambazari Road, Nagpur, Maharashtra. Pin 440010, India
kanchan.s.tiwari@gmail.com; ashwinkothari@ece.vnit.ac.in

[2]E&TC Department
Modern Education Society's College of Engineering
Pune, Maharashtra 411001, India

Received April 2014; revised August 2014

Abstract. *Rough set theory is a mathematical approach to process and interpret incomplete information system. Several researchers have dealt with the problem of finding reduct from the set of attributes, cores, and rules from databases using different software deployed on multiprocessor system. Recently researchers have started using Field Programmable Gate Array (FPGA) implementation as an alternate option. Software approach is versatile but slow as compared to hardware implementation. The goal of this work is to design an exemplary rough set co-processor based on rough set theory and map it on FPGA. This paper gives an insight of a rough set co-processor's modules. The theory of dealing with large databases is studied. With the usage of dual port RAM and pipelining in design, a considerable time is saved thus making it suitable for real time applications. The application for rough set co-processor is explained with the case study of a typical fault dictionary of a Very Large Scale Integrated (VLSI) chip. It can be used as a Built-in-Self-Test controller for testing VLSI chip. Simulation results show that proposed hardware is significantly faster than algorithms running on general-purpose processor. The rough set co-processor can also be used as hardware classifier unit in personal computer.*

Keywords: Rough set co-processor, Reduct, FPGA, Hardware accelerator, Discernibility matrix, Classification, Rules, HDL, Testability

1. **Introduction.** With the rapid growth in usage of Internet and cloud computing, there is an exponential growth in the quantity of data collected. Data mining is the process of extracting hidden patterns and discovering important rules from large amount of data. Data mining techniques are becoming an important tool for converting data into meaningful knowledge. Moreover, if data is incomplete or vague, the quality of rules generated from the database during the knowledge discovery phase can be inferior or incorrect. This has led to development of various data mining procedures for extracting correct information from such large fields of data.

Rough set theory (RST) offers a viable solution for dealing with such kind of databases. RST [1] is a new mathematical approach to intelligent data analysis and data mining. It has become a very popular tool for managing uncertainty and vagueness arising in databases, as the database used for data mining may contain imperfections, such as noise, error due to inaccurate measuring instruments. It is used for discovering data dependencies by reducing the number of attributes contained in a data set using the data alone. It does not require any further additional information like degree of membership

as required in fuzzy or probability in probability theory [2]. It is successfully applied in various fields including intelligent automation, fault diagnosis, pattern recognition, artificial intelligence, machine learning, etc.

Since more than quarter century, researchers have developed several efficient algorithms for reduct and rule generation and benchmarked their results using different software like ROSE, RSES, WEKA, C, ROSETTA, and MATLAB [3-5]. The software provides flexibility but becomes slow while handling larger databases. Another approach that has gained lots of interest amongst researchers in last decade is FPGA implementation of rough set algorithms. The need for hardware implementation of RST arises from the vast computational complexity of problems that causes delay in the optimization process of software implementations. In order to meet real-time requirements, power, and flexibility goals, a combination of general CPU and reconfigurable fabrics like FPGAs, are a promising solution leading to heterogeneous computing. In such systems, also called as hardware accelerators, multicore CPU provides high computation rates while the reconfigurable logic offers high performance per watt and adaptability to the application constraints. The speed advantage of hardware and its ability to parallelize, offers great advantages to rough set algorithms in overcoming the problems of speed-ups posed by purely software routines. However, these implementations focus on solving one specific problem due to the hardware resources constraints. These accelerators are dedicated fixed-function peripherals designed to perform a single computational intensive task repeatedly. They offload the main processor, allowing it to do general-purpose tasks that have little regularity in the structure. Using dedicated hardware offers a cost-effective way to increase the overall computational power of a processor because the system designer gains the flexibility of a general-purpose processor coupled with the computational advantage of dedicated hardware. This is the main motivation behind using FPGA for implementation of rough set algorithms in proposed work.

The literature survey shows hardware implementation of genetic, neural, and signal processing algorithms [6-8] for classification purpose. However, there is no such generic hardware classifier based on RST. In this paper, a design of rough set co-processor is presented. All blocks of rough set co-processor are discussed in brief along with their algorithms. This co-processor utilizes pipelining between discernibility and reduct block.

An application of testing Very Large Scale Integrated Circuit (VLSI) chip using the proposed rough set co-processor is also elaborated. It extracts important test vectors from database as a part of reduct generation activity and classifies chip as good or bad. This eventually helps in deciding whether chip is fault-free or faulty. The results obtained from FPGA mapping of rough set algorithms shows a significant acceleration of the computation time in comparison to software implementation.

The current state of art shows few hardware implementations of rough set algorithms. Pawlak described the idea of sample processor [9]. The processor generated decision rules from decision tables. This idea, however, was not realized on programmable logic devices. Lewis, Perkowski, and Jozwiak in 1999 presented architecture of rough sets processor based on cellular networks described in [10]. They implemented basic rough set operation of basic category, upper approximation, and lower approximation, indispensable and external comparison. In this paper, there is no discussion on time complexity, space complexity and type of data. Kanasugi initially presented the design and architecture of the RSP [11], [12] and then implemented his idea of computing reduct and generating rules using Skowron's discernibility matrix concept in 2001. They have dealt with binary attributes, leaving discretization process, a task for future development. There is no discussion on time complexity and space complexity. However, their algorithm is based on computing discernibility matrix and discernibility function, whose time complexity will

no longer be less than $O(|U|^2|A|^2)$. Sun et al. in his paper [13] have implemented RST algorithms on FPGA in 2011. Authors have provided a new and effective method for hardware fault diagnosis and verified the effectiveness of method through simulation. He made use of genetic algorithm along with RST and presented a case study of nonlinear aircraft model. Grzes et al. in their paper [14] presented reduct and core generation algorithm based on discernibility matrix. They have presented hardware solution architecture for binary decision table. They have discussed architecture of discernibility and reduct block. They used VHDL simulator and the development board equipped with an Altera FPGA during the research. The reduct generation algorithm is simple and based on attribute count frequency. The algorithm gives super reduct, however it does not discuss the case of resolving the conflict between two attributes having same count value. Authors in [15] presented a hardware approach for finding reduct using binary discernibility matrix. This paper presents design of complete model and is refinement over previous version as pipelining and dual port RAM is used. The algorithm used in this paper, is also based on attribute count frequency but the tie is resolved if the attributes count frequency is equal.

In this paper in Section 2, the basics of RST used in this paper are presented. Readers can refer [16,17] for better understanding of other terminologies. In Section 3, features of co-processor are discussed. In Section 4, rough set algorithms used for implementation on FPGA are studied. Section 5 elaborates an application used for testing the proposed co-processor followed by results and Section 6 concludes the paper.

2. **Rough Set Preliminaries.**

RST is an effective tool for mining deterministic rules from a database. The rough set philosophy is founded on the assumption that with every object of the universe of discourse we associate some information, i.e., knowledge is associated. The main motto of RST is "Let the Data Speak for themselves" [18].

Any set of all indiscernible (similar) objects is called an elementary set, and forms a basic granule (atom) of knowledge about the universe. Any union of elementary sets is referred to as a crisp (precise) set. A set, which is not crisp, is called rough (imprecise, vague). Consequently, each rough set has boundary region cases, i.e., objects which cannot with certainty be classified either as members of the set or of its complement. RST is an effective tool for mining deterministic rules from a database. It offers mathematical tools to discover patterns hidden in the data. It can be used for feature selection, feature extraction, data reduction, decision rule generation, and pattern extraction (templates, association rules), etc. It identifies partial or total dependencies in data, eliminates redundant data, and gives approach to null values, missing data, dynamic data and others. The terminologies of RST used in this paper are explained below.

2.1. **Information system.**

The basic vehicle for data representation in the rough set framework is an Information System (IS). An IS is a table listing attributes of objects. Every column represents an attribute (also called a variable or feature or can be observation) that can be measured for each object. Formally, IS is defined as IS = (U, A) where U is non-empty finite set of objects called as universe, i.e., $U = \{x_1, x_2, x_3, \ldots, x_n\}$; and A is a non-empty finite set of attributes such that a:U$\rightarrow$V$_a$ for every $a \in$ A. The V$_a$ is called as value set of a. In lots of applications, outcome of classification is known. This a posteriori knowledge which is expressed by one distinguished attribute called decision attribute. Such IS are called decision systems. A decision system is any IS of the form IS = (U, C $\cup$ $\{d\}$), where $d \notin$ A is the decision attribute. The elements of C are called conditional attributes. Table 1 shows an example of a typical IS.

TABLE 1. An information system

Objects	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	d
x_1	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*
x_2	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*2*
x_3	*1*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*3*
x_4	*1*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*4*
x_5	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*1*	*2*
x_6	*1*	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*3*
x_7	*1*	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*4*
x_8	*0*	*0*	*0*	*1*	*0*	*1*	*0*	*0*	*1*

In an IS shown in Table 1, d column is decision attribute column and c_1, c_2, c_3, ..., c_8 are condition attributes of 8 objects. A decision system expresses all the knowledge about the model.

2.2. **Reducts and core.** The decision table may contain unnecessarily large amount of redundant and superfluous attributes. The same or indiscernible objects may be represented several times. Removal of such attributes cannot worsen the classification. The RST provides us the tool to deal with this problem. Core and reduct are the two fundamental concepts of rough set. A reduct is the essential part of an IS, which can discern all objects discernible by the original IS. A reduct is the minimal attribute set preserving classification power of original dataset. Finding a reduct is similar to feature selection problem. A reduct contains a subset of condition attributes that are sufficient to classify the decision table. A reduct may not be unique. It has been shown that finding minimal reduct or all reducts are both NP-hard problems. The core is contained in all the reduct sets, and it is the necessity of the whole data.

2.3. **Discernibility matrix.** An IS can also be presented in terms of a discernibility matrix [19]. A discernibility matrix is a square matrix in which rows and columns are objects, and cells are attribute sets that discern objects. Two objects are considered discernible if and only if they have different values for at least one attribute. The discernibility matrix, denoted by M, for a decision table DT, of an IS is given as –

$$c_{ij} = \begin{cases} \phi, & f_D(x_i) = f_D(x_j) \\ a \in A;\ a(x_i) \neq a(x_j), & f_D(x_i) \neq f_D(x_j) \end{cases} \tag{1}$$

Using discernibility matrix, Skowron and Rauszer, proved several properties and constructed efficient algorithms related to reduct, core, dependencies from IS and decision tables. Reducts obtained from above given definition of discernibility matrix is called as decision relative reduct. A reduct is any minimal subset of condition features that discerns all pairs with different decision values. It is complete if the deletion of any attribute of a reduct makes at least one pair of objects with different decision attribute values indiscernible. The intersection of all reducts is called the core of the decision table. In Table 2 partial discernibility matrix for IS shown in Table 1 is tabulated.

2.4. **Inconsistent decision table.** A decision table is inconsistent if for a given pair of object, all condition attributes are same but differ in decision attribute, i.e., it belongs to two or more different classes [20]. Such cases are more prevalent in medical databases. Table 3 presents a medical database for 10 patients having symptoms of conjunctivitis disease. The symptoms of conjunctivitis disease forms condition attributes that includes redness, swelling, watering, etc. The last column of decision attribute indicates whether

TABLE 2. Partial binary discernibility matrix for the IS in Table 1

Objects	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
x_{12}	*1*	*1*	*1*	*1*	*0*	*0*	*1*	*0*
x_{13}	*0*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
x_{14}	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*1*
x_{15}	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*
x_{16}	*0*	*1*	*1*	*0*	*1*	*0*	*1*	*1*
x_{17}	*0*	*0*	*1*	*0*	*1*	*1*	*1*	*1*

TABLE 3. Inconsistent decision table

U	*Redness*	*Swelling*	*Watering*	*Photophobia*	*Itching*	*Headache*	*Pain*	*Discharge*	*Loss of vision*	*Conjunctivitis*
1	*1*	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*0*	*0*
2	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*0*
3	*1*	*0*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*1*
4	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
5	*0*	*0*	*1*	*1*	*0*	*1*	*1*	*1*	*0*	*1*
6	*1*	*0*	*0*	*0*	*1*	*0*	*1*	*1*	*0*	*0*
7	*0*	*1*	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*1*
8	*0*	*1*	*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*
9	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
10	*0*	*1*	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*0*

the patient is suffering from conjunctivitis disease or not (1 or 0). In Table 3, objects 7 and 10 make database inconsistent.

3. Proposed Rough Set Co-Processor.

3.1. **Features of proposed rough set co-processor.** The proposed hardware accelerator will run as co-processor in host computer sharing main memory. It will read attributes stored in memory of FPGA. The decision table consists of binary attributes. Although the rough set co-processor is generic, its basic specifications were frozen after studying characteristics of several databases from UCI Machine learning repository [21].

- Attributes: 65
- Data objects (N): 256
- Memory size: 256 MB
- Internal data bus: 32 bits
- Address bus: 8 bit
- Clock frequency: 100 MHz

3.2. **Rough set co-processor modules.** The controller's state diagram of the proposed Rough Set Co-Processor is shown below in Figure 1. Its controller is a state machine that is directed through various states viz: discern, discern_reduct, reduct and rule. The first state of controller is discern. In this state formation of discernibility matrix takes place. The next state is discern_reduct. It is an intermediate state in which the process of discernibility matrix building and reduct computation operation is pipelined. A change of state from discern to discern_reduct takes place after 80% of discernibility matrix is built. The third state is reduct. In this state reduct computation operation is continued. In reduct computation process, important features are extracted from discernibility matrix. A reduct_flag used as a control signal decides the change of state from reduct to rule. The controller stays in reduct state as long as the value of reduct_flag is zero and exits this state when reduct_flag is one.

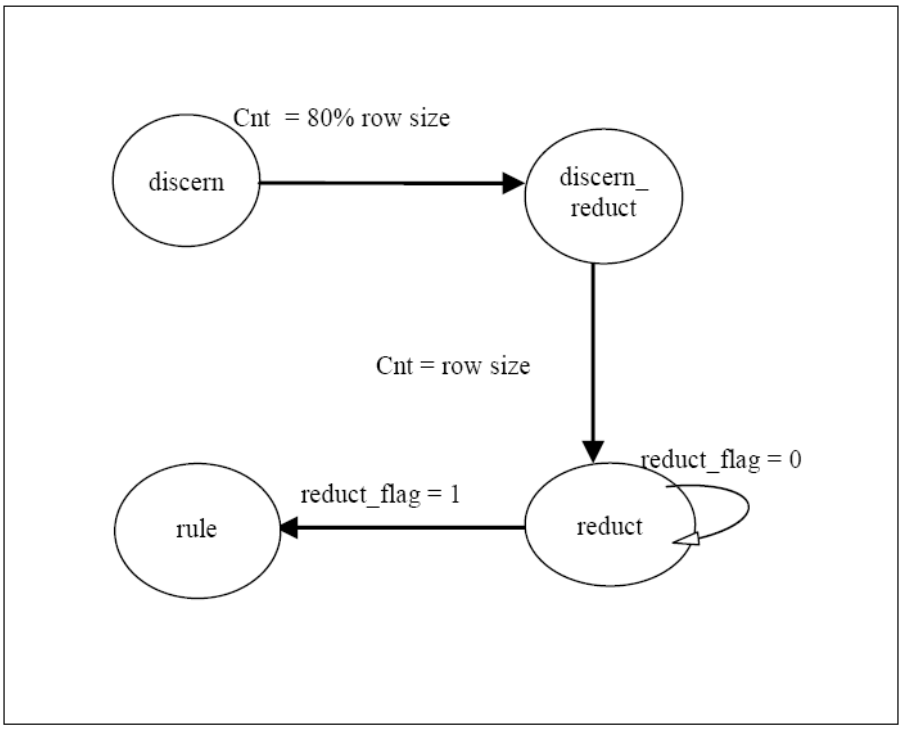

FIGURE 1. State diagram of rough set hardware accelerator

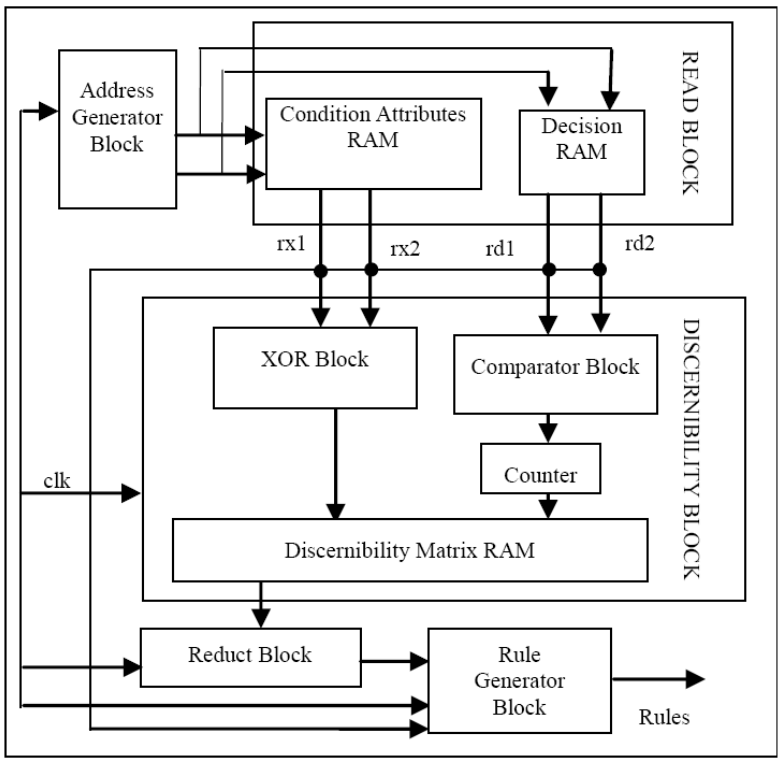

FIGURE 2. Rough set hardware accelerator block diagram

The block diagram of Rough Set Hardware Accelerator is shown in Figure 2. There are five main modules in it.

a. Address Generator Block: The address generator block operates on a clock signal along with some control logic and generates two addresses. It is used for generating addresses of dual port RAMs.

b. Read Block: This block consists of two dual ports RAM. These dual port RAMs are used for storing condition and decision attributes. Read block reads content of selected memory locations specified by addresses generated by address generator block. These contents are named in block diagram as rx1, rx2, rd1, and rd2. The rx1, rx2, rd1, and rd2 are output data lines of dual port condition and decision RAM respectively.

c. Discernibility Block: Discernibility block generates discernibility matrix elements from decision table stored in dual port RAMs. It uses comparator block for comparing rd1 and rd2. If unequal, comparator block sends a signal to XOR block which performs XORing operation on rx1 and rx2. This XORed data is stored in consecutive location of discernibility matrix. A counter embedded in discernibility block itself generates the address of discernibility matrix.

d. Reduct Block: This block is heart of rough set co-processor. It processes the elements of discernibility matrix and uses the algorithm discussed in next section for computing single reduct.

e. Rule Generator Block: This block classifies the given set of objects in different classes. It uses reduct generated by reduct block for generating rules in if-then form.

4. Rough Set Algorithms Used for Implementation.

The algorithm used for implementation of blocks of rough set co-processor is discussed in this section.

4.1. Address generator block.

Algorithm 1: Address Generator Block

Input: *clk, reset*

Address generator block is invoked on changes in *clk* or *reset*. The *reset* value is set to 1 at the start and also when state changes to rule. The *row_size* is number of objects in database.

Output: *addr1 and addr2*

Algorithm:

```
if (reset == 1) then
  addr1 = 0;
  addr2 = addr1 + 1;
elsif (addr1 < row_size) then
  if (addr2 >= row_size) then
    addr1 = addr1 + 1;
    addr2 = addr1 + 1;
  elsif (addr2 < row_size) then
    addr2 = addr2 + 1;
  end if;
end if;
```

4.2. Read block.

Algorithm 2: *Read Block*

Input: *addr1, addr2*

Process of read block is invoked on changes in *clk* or *addr1* or *addr2*.

Output: *rd1, rd2, rx1, rx2*

Algorithm:

```
rd1 = read decision attribute value at row addr1;
rd2 = read decision attribute value at row addr2;
```

rx1 = read condition attribute value at row addr1;
rx2 = read condition attribute value at row addr2;

4.3. **Discernibility block.**
Algorithm 3: Simplified Discernibility Matrix building
Input: *rd1, rd2, rx1, rx2*

The process of discernibility block is invoked on changes in *clk* or *rd1* or *rd2*. The code is executed when state is either *disern* or *disern_reduct*. The *disMatAddr* is address of RAM used for storing discernibility elements.

Output: Discernibility Matrix. The signal *disMat* values are stored in discernibility matrix ram at address *disMatAddr.*

Algorithm:

if (rd1/= rd2) then
disMat = rx1 XOR rx2;
disMatAddr = disMatAddr + 1;
end if;

4.4. **Reduct block.** Attribute reduction algorithms are based on positive region, discernibility matrix, heuristic, information entropy, genetic, fuzzy-rough, rough-neural, granular computing, information view, etc. It can also be achieved by evaluating various attribute significant parameters like dependency, consistency, information gain, etc. in case of heuristic based methods [22]. In this work, discernibility matrix concept is used since literature survey shows popularity of discernibility matrix method amongst researchers as the method is very simple and intuitive. The only drawback of discernibility matrix method is requirement of huge memory and they tend to become slow while dealing with large databases. With the recent growth in FPGA, all these issues are largely resolved. The proposed reduct algorithm is based on attribute count frequency [23] of all objects in discernibility matrix. In this algorithm, sum of elements of each column is computed. A column whose sum is unique and maximum is marked as a part of reduct vector. However, if the sum of more than one column is having same maximum value, then the conflict is resolved by computing row length and it is multiplied with each column attribute value. All values are summed up to obtain a final significant factor. The ratio of column sum with final significant factor is taken and whichever columns give maximum value, it is marked as a part of reduct vector. This block is most critical block in the entire system in terms of resources consumed and speed. An example showing the computation of reduct from a sample discernibility matrix is illustrated below in Table 4.

In Step 1, significance of all attributes is computed by finding sum of each column which gives, $F_1(c_1) = F_1(c_2) = F_1(c_4) = 3$, where F_1 represents first significant factor.

TABLE 4. Reduct computation from partial discernibility matrix

	c_1	c_2	c_3	c_4	*Row_sum*
x_{12}	*0*	*1*	*0*	*0*	*1*
x_{13}	*1*	*1*	*0*	*1*	*3*
x_{14}	*0*	*0*	*0*	*1*	*1*
x_{23}	*1*	*0*	*1*	*1*	*3*
x_{34}	*1*	*1*	*0*	*0*	*2*
F_1	*3*	*3*	*1*	*3*	
F_2	*0.375*	*0.5*	–	*0.428*	

Since there is tie between three attributes, hence second significant factor F_2 is computed for finding reduct as given below:

The initial value of reduct vector = 0000

$F(c_1) = 1*0+3*1+1*0+3*1+2*1 = 8$, $F(c_2) = 1+3+2 = 6$, $F(c_4) = 3+1+3 = 7$;

$F_2(c_1) = F_1(c_1)/F(c_1) = 3/8 = 0.375$.

Similarly, $F_2(c_2) = 3/6 = 0.5$, and $F_2(c_4) = 3/7 = 0.428$.

In this case, the maximum value is of c_2; hence, it is chosen as a member of reduct set. Hence, the reduct vector becomes 0100. However, while performing division, since the numerator is same (for conflicting attributes), hence division operation is skipped. Instead of choosing maximum values amongst F_2 of conflicting attributes, minimum value of F is chosen. This further saves hardware required and computation time. In next iteration, reduct vector is ANDed with discernibility matrix and if the result of ANDing is zero, that row in discernibility matrix is retained. However, if the result is nonzero, that row is eliminated. This process ensures removal of attribute marked as a member of reduct set before start of next iteration. This results in creation of a modified discernibility matrix. The process of computing F_1 and F_2 is repeated until discernibility matrix becomes empty. The algorithm is described below:

Algorithm: Reduct Block

Input: *discern_mat, disMat_addr*

Process of reduct block is invoked on changes in *clk* or *disMat_addr*. The code is executed when the state is either *disern_reduct* or *reduct*.

Output: *reduct*.

Algorithm:

Step 1: Initialize reduct with 0. Initialize valid_row array with 1 (1 represents valid row and 0 represents invalid row).

Step 2: Calculating column significance value

Step 2.1: Mark invalid rows

```
if (discern_mat == ZERO or (discern_mat AND reduct)/= ZERO)) then
valid_row(disMat_addr) = 0
```

Step 2.2: For valid rows calculate significance value

```
for i in 0 to COL_SIZE −1    loop
  if (discern_mat(i) == 1) then
    row_sum_arr(disMat_addr) = row_sum_arr(disMat_addr) + 1;
    col_sum_arr(i) = col_sum_arr(i) + 1;
  end if;
end loop;
for i in 0 to COL_SIZE −1    loop
  if (discern_mat(i) == 1) then
    sin_col_dnr_arr(i) = sin_col_dnr_arr(i) + row_sum_arr(disMat_addr);
  end if;
end loop;
```

Step 3: Setting bit corresponding to highest significant attribute to 1 in reduct

```
max_col_sum = max(col_sum_arr);
min_val = ROW_SIZE;//Initialize with any random high number
for i in 0 to COL_SIZE −1    loop
  if max_col_sum == col_sum_arr(i) then
    //if more than one column has max value then decision is made on the basis of
min value of sin_col_dnr_arr
    if (min_val > sin_col_dnr_arr(i)) then
      min_val = sin_col_dnr_arr(i);
```

reduct_col_index = i;
end if;
end if;
end loop;
reduct(reduct_col_index) = 1;

The time complexity of building binary discernibility matrix under worst condition is $O(|A||U|^2)$. In Step 1 of reduct generation algorithm time complexity is not more than $O(|A||U|^2)$, these steps are again repeated R times, hence its time complexity modifies to $O(|R||A||U|^2)$. Reduct computation is a critical task, as this block consumes most of the processing time. In order to speed up the process of reduct computation, not only dual port RAM is used for storing discernibility matrix elements but also pipelining is used between discernibility and reduct block.

4.5. **Rule generator block.** The reduct generated by the reduct block are used for creating valid meaningful rules using the algorithm given below:

Algorithm: Rule Generator Block Algorithm

Input: *rd1, rd2, rx1, rx2, addr1, addr2*, and *reduct*

The process of rule generator block is invoked on changes in *clk*. The code is executed when state is *rule*.

Output: *rules*

Algorithm:

duplicate_row; // duplicate_row is array with size = row_size. Its elements are initialized with 0.
if (duplicate_row(addr1) == 0) then
rx1M = rx1 AND reduct;
if (rx1M /= ZERO) then
rule_out = rx1M;
end if ;
r_x2M = rx2 and reduct;
if ((rd1 == rd2) AND (rx1M == rx2M)) then
duplicate_row(addr1) = 1;
end if;
end if;

Condition attributes and reduct Vector are ANDed together. This gives modified decision table consisting of important rules. In the modified decision table, objects belonging to same decision class are merged together if they have similar condition attributes; to form a rule otherwise the rows are kept unchanged. This step is repeated till no merging is possible. Each entry in the new reduced table corresponds to a rule, which can be expressed in conventional if-then form.

For example, if the reduct vector is taken as 000101001, ANDing it with condition attributes modifies decision table, in which only values of 3 condition attribute columns are preserved. They are c_4, c_6, c_9 and rest of the values become 0. The modified decision table is shown in Table 5.

Rule extraction consists of merging identical pair objects belonging to same decision class that further reduces the decision table as shown in Table 6.

Now one can get the control rules from Table 6 and deduce inferences from it as follows:

- If $c_4 = 1$ and $c_6 = 1$ and $c_9 = 1$ then $d = 1$.
- If $c_4 = 1$ and $c_6 = 1/0$ and $c_9 = 1/0$ then $d = 2$.
- If $c_6 = 1$ and $c_9 = 0$ and $c_4 = 0/1$ then $d = 3$.
- If $c_4 = 0$ and $c_6 = 1/0$ and $c_9 = 0/1$ then $d = 4$.

5. Application and Results.

5.1. Application.

RST has been widely used in variety of fields. In this paper, the application related to testing of VLSI chips is discussed. There has been a continuous pressure on VLSI chip manufacturing industry to increase the manufacturing yield. Integrated circuit (IC) manufacturers are constantly trying to decrease the number of faulty parts they produce. The reliability of System on Chips must be ensured to a certain extent since a single fault is likely to make the whole chip useless. VLSI testing is the backbone of manufacturing reliable IC. Therefore, fault diagnosis and fault repairing techniques are gaining importance these days. A manufacturer may be able to improve the circuit design or the manufacturing process by analyzing the parts that fail production tests and determining the cause of failure for each part. During the design as well as manufacturing, several faults may creep in. Therefore, there is a need to test the VLSI chips at the design level as well as after manufacturing. Detection of fault and the type of fault present in a circuit is known as fault diagnosis. With increase in the complexity of the integrated circuits, the effort to detect and diagnose the fault is becoming time consuming and demanding more and more resources. Built in Self-test (BIST) is nowadays gaining more popularity as it reduces the cost of using Automatic Test Equipment (ATE). Such dedicated rough set co-processor can effectively work as BIST controller. RST philosophy can be applied in detecting a set test vectors, which can model more than 90% faults. This theory is applied in diagnosis of faulty VLSI chips by making use of fault dictionary [24]. A fault dictionary is defined as a database of faults that can be used by a simulator to help determine the fault coverage. A fault dictionary may also be used by diagnostic system to help analyse faults when trying to diagnose problems. A typical fault dictionary is shown in Table 7. Each fault (single stuck at fault, stuck open, stuck short, bridging fault, etc.) for a chip (or system) is listed down the left side of the table and the test vectors that activate and propagate the faults are listed along the top. It is a 2 class system, which gives decision as whether chip is faulty or fault free. If not, an unnecessary amount of faults will share the same test syndrome (described below). The test vectors typically consist of a complete test set for the chip if one is known. The matrix of ones and zeros in the fault dictionary relate the faults to the tests that detect each fault. For matrix element $[i, j]$, a "1" means that the particular test set, Test$j$, detected the fault, Fault$i$. A "0" means Test$j$ did not detect Fault$i$. Fault dictionary consists of huge number of test sets. The count will depend on number of inputs and the relationship is 2^I, where 'I' is the number of inputs. In that case, a 16 input chip will have 65,536 test sets. Few from these test sets can give more than 90% of fault coverage while rests of them are redundant. These redundant test vectors can be identified and removed with the help of reduct generation algorithm.

TABLE 5. Modified decision table

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	d
x_1	$*$	$*$	$*$	*1*	$*$	*1*	$*$	$*$	*1*	*1*
x_2	$*$	$*$	$*$	*1*	$*$	*1*	$*$	$*$	*0*	*2*
x_3	$*$	$*$	$*$	*1*	$*$	*1*	$*$	$*$	*0*	*3*
x_4	$*$	$*$	$*$	*0*	$*$	*1*	$*$	$*$	*0*	*4*
x_5	$*$	$*$	$*$	*1*	$*$	*0*	$*$	$*$	*1*	*2*
x_6	$*$	$*$	$*$	*0*	$*$	*1*	$*$	$*$	*0*	*3*
x_7	$*$	$*$	$*$	*0*	$*$	*0*	$*$	$*$	*1*	*4*

TABLE 6. Merged decision table

c_4	c_6	c_9	d
1	*1*	*1*	*1*
1	*1*	*0*	*2*
1	*0*	*1*	*2*
1	*1*	*0*	*3*
0	*1*	*0*	*3*
0	*1*	*0*	*4*
0	*0*	*1*	*4*

TABLE 7. Fault dictionary

	Test1	*Test2*	*Test3*	*Test4*	*Test5*	*Test6*	*Test7*
Fault1	*0*	*1*	*0*	*0*	*0*	*1*	*0*
Fault2	*1*	*0*	*0*	*0*	*0*	*0*	*0*
Fault3	*1*	*0*	*0*	*0*	*0*	*0*	*0*
Fault3	*0*	*1*	*0*	*1*	*0*	*0*	*0*
Fault4	*0*	*0*	*0*	*1*	*0*	*0*	*0*
Fault5	*0*	*0*	*1*	*0*	*0*	*0*	*0*
Fault6	*0*	*0*	*0*	*0*	*0*	*1*	*1*

5.2. **Experimental results.** The modules of rough set co-processor are tested and verified on Xilinx 14.5 platform using ISIM simulator. A random database is used for testing. The device used for implementation is Virtex 5. The simulation waveforms are shown in Figure 5. The clock frequency used for simulation is 100 MHz. For benchmarking timing results of hardware, all above-mentioned algorithms are implemented in C. C language is used for implementation as its compiler produces machine code that can run nearly as fast as machine language. The environment used is Windows 7, RAM 4 GB, Processor i5 2.54 GHz. Table 10 presents the results of the time elapsed for software and hardware solutions while calculating rules for varying size of database. Figure 4 shows a significant decrease in the execution time as algorithm mapping is changed from software to hardware. Logarithmic scale is used for time plotted on Y-axis. The time required for computing discernibility matrix, reduct, and rules is less as compared to [14]. In [14], authors have shown the time required for computing only reduct and cores.

In Figure 5 complete simulation result for database of size 8 by 8 is shown in two parts. A small database is used as it makes waveforms more readable. In first part, current_state signal shows three distinct states of controller. In discern state discernibility matrix is built; its value is shown by signal dismat and address by dismataddr. Discernibility matrix completes when current_state changes to reduct. The value of reduct is shown through signal reduct. Its value is 'U' until reduct is completely formed. In Figure 5 reduct state starts at 290 ns and completes at 630 ns. At this instant, current_state is changed to rule state. The rule signal is the set of final rules generated by rough set co-processor. The rule signal when clubbed with decision specifies the rule vector generated for a specific decision class. In above figure, e.g., when rule is 12 and decision is 3, it implies that for a rule vector of "00001100"; decision class is 3.

TABLE 8. A typical IS

Patient	*Headache*	*Muscle-pain*	*Temperature*	*Flu*
p_1	*No*	*Yes*	*High*	*Yes*
p_2	*Yes*	*No*	*High*	*Yes*
p_3	*Yes*	*Yes*	*Very high*	*Yes*
p_4	*No*	*Yes*	*Normal*	*No*
p_5	*Yes*	*No*	*High*	*No*
p_6	*No*	*Yes*	*Very high*	*Yes*
p_7	*Yes*	*No*	*Extremely high*	*Yes*

TABLE 9. A binarised IS

Patient	*Headache*	*Muscle-pain*	*Temperature*		*Flu*
p_1	*0*	*1*	*0*	*1*	*1*
p_2	*1*	*0*	*0*	*1*	*1*
p_3	*1*	*1*	*1*	*0*	*1*
p_4	*0*	*1*	*0*	*0*	*0*
p_5	*1*	*0*	*0*	*1*	*0*
p_6	*0*	*1*	*1*	*0*	*1*
p_7	*1*	*0*	*1*	*1*	*1*

TABLE 10. Comparison of execution time for calculating rules

Sr.No.	*Object size*	*Software (μs)*	*Hardware (μs)*
1	4×4	*13076.67*	*0.31*
2	8×8	*15296.67*	*0.95*
3	12×12	*23100*	*2.2*
4	16×16	*30183.33*	*3.6*
5	32×32	*96743.33*	*14.86*
6	64×64	*291400*	*63.85*
7	128×128	*1091667*	*257.2*

5.3. **Handling symbolic attributes and larger databases.**

5.3.1. *Symbolic attributes.* The proposed co-processor is designed for handling 64 condition attributes and 1 decision attribute. It can deal with 256 objects. It can handle inconsistent attributes as well. A database consisting of nominal values is discretized to binary values. For example in an IS shown below in Table 8, yes or no value of attribute can be coded as 1 or 0. However, for coding normal, high and, very high 2 bits is required. Hence temperature attribute is coded by 2 bits; normal -00, high -01, very high -10 and extremely high -11. The corresponding discretized values are shown in Table 9.

5.3.2. *Larger databases.* For a larger database, modular approach based on divide and conquer technique is used. The larger database is divided into smaller IS consisting of 256 objects each and the reduct is calculated from each independently. From these set of reducts, a final reduct is calculated using union operator (logical OR). The approach used for generating reduct (as shown in Figure 3) starts with checking objects size of IS; where objects size (N) corresponds to total number of objects present in given IS. If N of given IS is greater than allowed size of 256, then divide and conquer strategy is

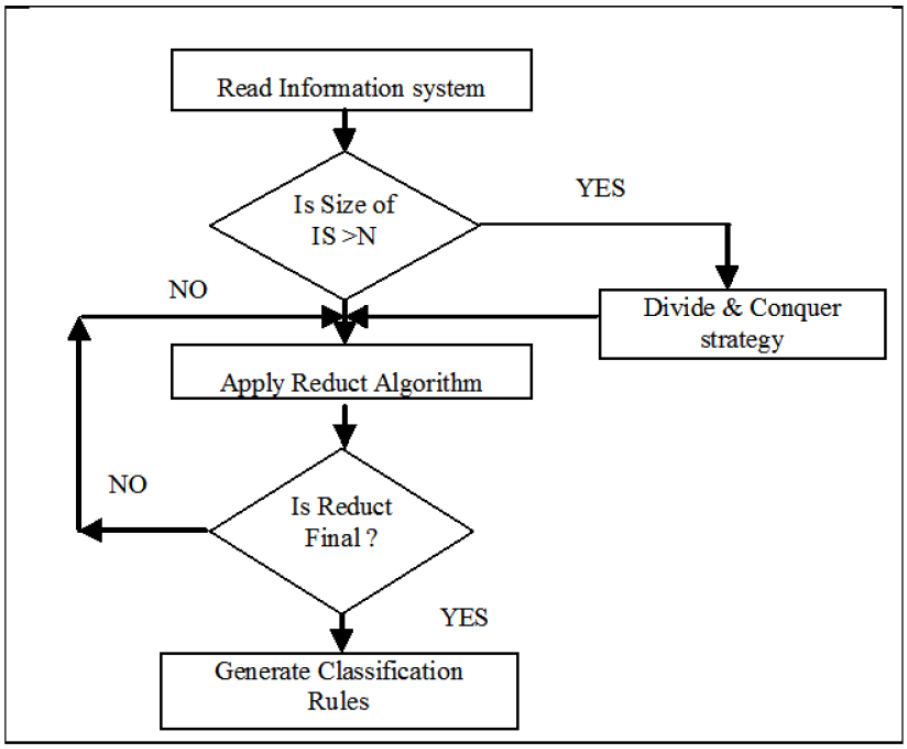

FIGURE 3. Flow chart for classification using rough set philosophy

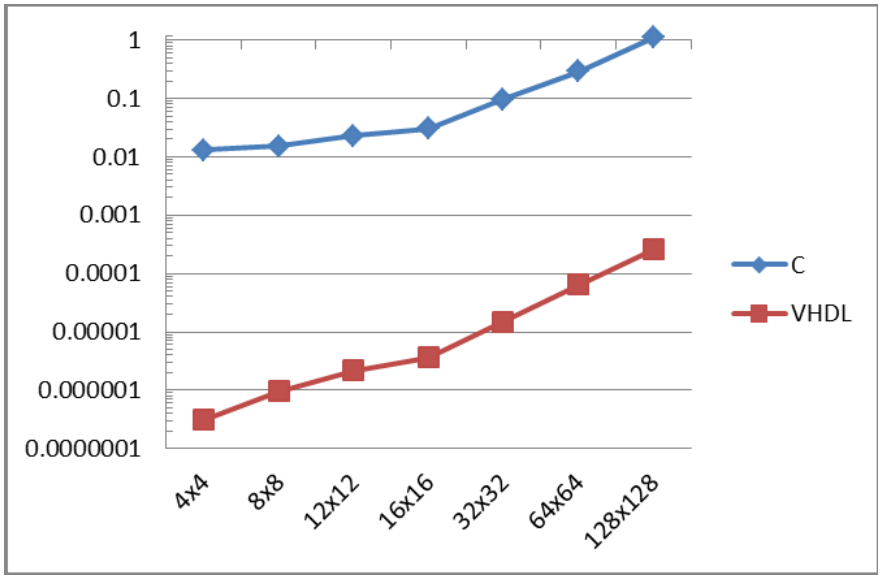

FIGURE 4. Comparison of execution time for software and hardware

used. It is divided in small IS where size of each IS will be 256. These IS are treated in parallel and independently for getting partial reducts and final reduct is generated by applying union operator on all generated reducts. The maximum conditional attributes it can treat at a time is 64, however if given IS attributes exceeds mentioned capacity, then IS is segregated column wise of size 256 by 64 and treated independently. In this case, the final reduct vector is obtained by appending all partially generated reduct vectors. These reduct subsequently gives classification rules.

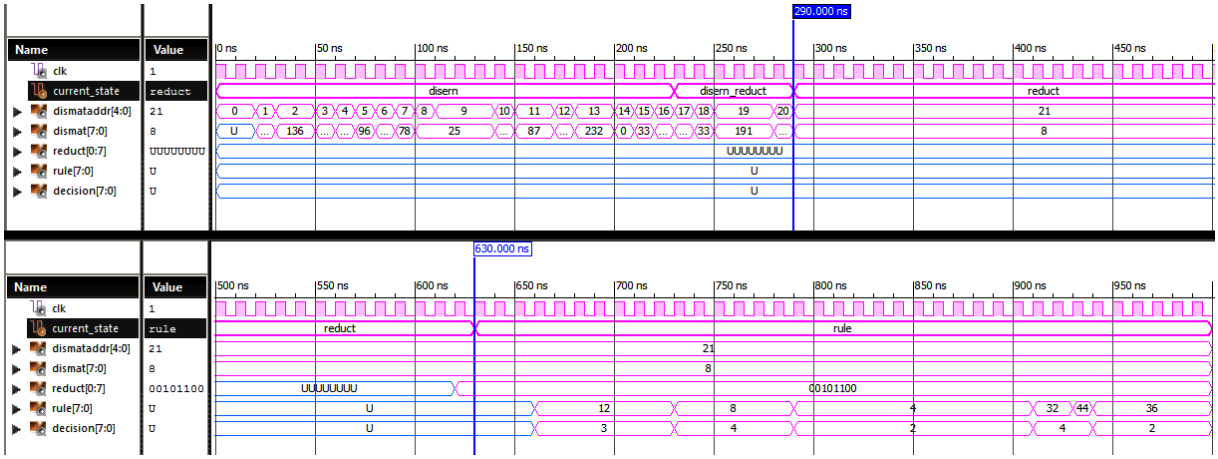

FIGURE 5. Simulation result for database of size 8 by 8

6. **Conclusion and Future Work.** Hardware implementation of rough set algorithms and their usage in real life will gain more importance in due course of time. In this paper, design and architecture of rough set co-processor on FPGA is presented. It is a hardware accelerator, which will perform a certain set of operations like discernibility matrix computation, reduct as well as rule generation. A comparative study of hardware and software simulation time shows a vast time difference. An analytical reasoning show that time complexity of proposed algorithm ($O|R||A||U|^2$) is less than existing ones. Divide conquer strategy along with parallel processing is used for dealing with larger databases. A dedicated hardware for approximate reasoning will offload main processor from computational overhead, thereby increasing the speed of the operation. Simulation results are shown using Xilinx software. Our future work includes handling continuous attributes and discretization of continuous attributes.

Acknowledgment. We hereby acknowledge the funding support by Center of Excellence (CoE), Commbedded System-hybridization of Communication and Embedded System under TEQIP 1.2.1, VNIT, Nagpur. This work is also partially supported by Board of College and University Development, Pune. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] Z. Pawlak and A. Skowron, Rudiments of rough sets, *Information Sciences*, vol.177, pp.3-27, 2007.
[2] Z. Pawlak, Rough sets, *International Journal of Computer and Information Sciences*, vol.11, pp.341-356, 1982.
[3] B. Predki et al., *ROSE – Software Implementation of the Rough Set Theory, Rough Sets and Current Trends in Computing*, Springer-Verlag, Berlin Heidelberg, 1998.
[4] A. Skowron et al., Logic Group, Institute of Mathematics, Warsaw University, Poland, *http://logic.mimuw.edu.pl/∼rses/*, 2010.
[5] M. Kierczak et al., ROSETTA Development Team, *http://www.lcb.uu.se/tools/rosetta/*, 2010.
[6] S. D. Scott, A. Samal and S. Seth, HGA: A hardware-based genetic algorithm, *ACM the 3rd International Symposium on Field-Programmable Gate Arrays*, New York, USA, pp.53-59, 1995.
[7] P. Moerland and E. Fiesler, Neural network adaptations to hardware implementations, *Handbook of Neural Computation*, vol.1, no.2, 1997.
[8] A. E. Nelson, *Implementation of Image Processing Algorithms on FPGA Hardware*, Master Thesis, Vanderbilt University, 2000.
[9] Z. Pawlak, Elementary rough set granules: Toward a rough set processor, *Rough-Neural Computing Cognitive Technologies*, pp.5-13, 2004.

[10] T. Lewis, M. Perkowski and L. Jozwiak, Learning in hardware: Architecture and implementation of an FPGA – Based rough set machine, *Proc. of the 25th IEEE EUROMICRO Conference*, pp.326-334, 1999.

[11] A. Kanasugi, A design of architecture for rough set processor, *Proc. of the Joint JSAI 2001 Workshop on New Frontiers in Artificial Intelligence*, vol.2253, pp.406-410, 2001.

[12] A. Kanasugi and M. Matsumoto, Design and implementation of rough rules generation from logical rules on FPGA board, *Rough Sets and Intelligent Systems Paradigms, LNCS*, vol.4585, pp.594-602, 2007.

[13] G. Sun, X. Qi and Y. Zhang, A FPGA-based implementation of rough set theory, *Control and Decision Conference*, pp.2561-2564, 2011.

[14] T. Grzes, M. Kopczynski and J. Stepaniuk, FPGA in rough set based core and reduct computation, *RSKT*, pp.263-270, 2013.

[15] K. S. Tiwari, A. G. Kothari and A. G. Keskar, Reduct generation from binary discernibility matrix: An hardware approach, *International Journal of Future Computer and Communication*, pp.270-272, 2012.

[16] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht, Boston, London, 1991.

[17] Z. Pawlak, Rough sets and intelligent data analysis, *Information Sciences*, vol.147, nos.1-4, pp.1-12, 2002.

[18] Rough set theory: A true landmark in data analysis, in *Studies in Computational Intelligence*, A. Abraham et al. (eds.), Berlin Heidelberg, Springer-Verlag, 2009.

[19] A. Skowron and C. Rauszer, The discernibility matrices and functions in information systems, *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory*, vol.11, pp.331-362, 1992.

[20] D. Q. Miao et al., Relative reducts in consistent and inconsistent decision tables of the Pawlak rough set model, *Information Sciences*, vol.179, no.24, pp.4140-4150, 2009.

[21] K. Bache and M. Lichman, *UCI Machine Learning Repository*, http://archive.ics.uci.edu/ml, Irvine, University of California, School of Information and Computer Science, 2013.

[22] K. Thangavel and A. Pethalakshmi, Dimensionality reduction based on rough set theory: A review, *Applied Soft Computing*, vol.9, no.1, pp.1-12, 2009.

[23] P. Yang, J. Li and Y. Huang, An attribute reduction algorithm by rough set based on binary discernibility matrix, *IEEE the 5th Int. Conf. Fuzzy Systems and Knowledge Discovery*, pp.276-280, 2008.

[24] A. Ray, *Fault Diagnosis Using Fault Dictionaries and Probability, VLSI Testing*, Auburn University, 2005.