Core for Large Datasets: Rough Sets on FPGA

Maciej Kopczynski, Tomasz Grzes, and Jaroslaw Stepaniuk

Faculty of Computer Science
Bialystok University of Technology
Wiejska 45A, 15-351 Bialystok, Poland
{m.kopczynski,t.grzes,j.stepaniuk}@pb.edu.pl
http://www.wi.pb.edu.pl

Abstract. In this paper we propose the FPGA and softcore CPU based device for large datasets core calculation using rough set methods. Presented architecture has been tested on two real datasets by downloading and running presented solution inside FPGA. Tested datasets had 1 000 to 10 000 000 objects. The same operations were performed in software implementation. Obtained results show the big acceleration in computation time using hardware supporting core generation in comparison to pure software implementation.

Key words: Rough sets, FPGA, hardware, core

1 Introduction

The rough sets theory developed in the eighties of the twentieth century by Prof. Z. Pawlak is an useful tool for data analysis. Therefore a lot of rough sets algorithms were implemented in scientific and commercial tools for data processing. But data processing efficiency problem is arising with the increase of the amount of data. Software limitations led to searching new possibilities.

Field Programmable Gate Arrays (FPGAs) are the digital integrated circuits which functions can be programmed by engineer at any time. It gives the possibility of evaluating any boolean functions. That's why they can be used for supporting rough sets calculations.

At the moment there are some hardware implementations of specific rough set methods. The idea of sample processor generating decision rules from decision tables was described in [8]. In [5] the authors presented architecture of rough set processor based on cellular networks described in [7]. In [1] a concept of hardware device capable of minimizing the large logic functions created on the basis of discernibility matrix was developed. More detailed summary of the existing ideas and hardware implementations of rough set methods can be found in [2] and in [13]. Previous authors' research results focused on this subject can be found in [3, 4, 12].

The paper is organized as follows. In Section 2 some information about the notion of core and datasets used during research are provided. The Section 3 focuses on description of hardware solution, while Section 4 is devoted to the experimental results.

2 Introductory Information

2.1 The Notion of Core in the Rough Set Theory

In decision table some of the condition attributes may be superfluous (redundant in other words). This means that their removal cannot worsen the classification. The set of all indispensable condition attributes is called the core. None of its elements can be removed without affecting the classification power of all condition attributes. In order to compute the core we can use discernibility matrix. The core is the set of all single element entries of the discernibility matrix.

A much more detailed description of the concept of the core can be found, for example, in the article [9] or in the book [11].

2.2 Algorithm CORE-DDM for Generating Core Using Discernibility Matrix

Below one can find pseudocode for simple algorithm CORE-DDM (**CORE** **D**irect **D**iscernibility **M**atrix) for calculating core using discernibility matrix. More detailed description of this approach can be found in [9, 11].

```
INPUT: discernibility matrix $DM$
OUTPUT: core $C \subseteq A$
1: $C \leftarrow \emptyset$
2: for $x \in U$ do
3:    for $y \in U$ do
4:       if $|DM(x, y)| = 1$ and $DM(x, y) \not\subseteq C$ then
5:          $C \leftarrow C \cup DM(x, y)$
6:       end if
7:    end for
8: end for
```

The main concept of algorithm CORE-DDM is based on a property of singleton ie. cell from discernibility matrix consisted of the only one attribute. This property tells that any singleton cannot be removed without affecting the classification power.

Input for the algorithm is the discernibility matrix DM. Output is core C as a subset of condition attributes set denoted as A. Core is initialized as empty set in line 1. Two loops in lines 2 and 3 iterates over all objects (denoted as U) in discernibility matrix. Condition instruction in line 4 checks if matrix cell contains only one attribute. If so, then this attribute is added to the core C.

2.3 Algorithm CORE-IDM for Generating Core with No Discernibility Matrix

The main disadvantage of using discernibility matrix for big datasets is its size. Memory complexity of creating this type of matrix is $|U|^2|A|$, where A is the condition attributes set. This makes a simple solution showed in previous subsection unusable for big data. For example the table of 1,000,000 objects and 8 condition attributes encoded as single bits requires 1TB of memory.

We have proposed our algorithm CORE-IDM (**CORE** **I**ndirect **D**iscernibility **M**atrix) which uses discernibility table indirectly to perform comparison between each row of given decision table. This approach, basing on CORE-DDM algorithm in principles, has been designed by authors of this paper. Below is the pseudocode for this algorithm:

```
INPUT: decision table $DT = (U, A \cup \{d\})$
OUTPUT: core $C \subseteq A$
 1: $C \leftarrow \emptyset$
 2: for $x \in U$ do
 3:   for $y \in U$ do
 4:     if $d(x) \neq d(y)$ then
 5:       $count \leftarrow 0$
 6:       for $a \in A$ do
 7:         if $a(x) \neq a(y)$ then
 8:           $count \leftarrow count + 1$
 9:           $candidate \leftarrow a$
10:         end if
11:       end for
12:       if $count = 1$ and $candidate \notin C$ then
13:         $C \leftarrow C \cup \{candidate\}$
14:       end if
15:     end if
16:   end for
17: end for
```

Input to the algorithm CORE-IDM is decision table DT, and output is core C. A denotes condition attributes set. In the first step core C is initialized as empty set. Two loops in lines 2 and 3 take subsequent objects from decision table for comparison. Line 4 performs the comparison between decision attribute value of two objects x and y. If these two objects belong to different decision classes, the rest of the algorithm is processed. $count$ variable, responsible for storing the number of differences on condition attributes values between objects x and y is set to 0 in line 5. Loop in line 6 iterates over set of condition attributes A. Values of a condition attribute is compared between objects x and y in line 7. In case of difference, the $count$ variable is incremented and a attribute is stored in $candidate$ variable. When the attribute loop finishes, attribute in $candidate$ variable is added to the core if $count$ variable is equal to 1 and this attribute is not in core (lines 12 to 14).

2.4 Algorithm CORE-HIDM for Hardware Supported Core Calculation with No Discernibility Matrix

Algorithm described in previous section cannot be run in hardware because of FPGA resources limitations. It is impossible to store large dataset inside hardware module. This is the reason why we present the modification of previous algorithm - CORE-HIDM (**CORE** **H**ardware **I**ndirect **D**iscernibility **M**atrix). Main idea is to divide the entire dataset into parts stored in two independent memory units for hardware module. These parts are subsequently processed by the unit. Pseudocode for the algorithm is given below:

```
INPUT: decision table $DT = (U, A \cup \{d\})$
OUTPUT: core $C \subseteq A$
1: $C \leftarrow \emptyset$
2: for $cnt_1 \leftarrow 0$ to $m - 1$ do
3:     $RAM1 \leftarrow \{x \in U : x_{cnt_1 \cdot n}$ to $x_{(cnt_1+1) \cdot n-1}\}$
4:     for $cnt_2 \leftarrow cnt_1$ to $m - 1$ do
5:         $RAM2 \leftarrow \{x \in U : x_{cnt_2 \cdot n}$ to $x_{(cnt_2+1) \cdot n-1}\}$
6:         for $x \in RAM1$ do
7:             for $y \in RAM2$ do
8:                 if $d(x) \neq d(y)$ then
9:                     $count \leftarrow 0$
10:                    for $a \in A$ do
11:                        if $a(x) \neq a(y)$ then
12:                            $count \leftarrow count + 1$
13:                            $candidate \leftarrow a$
14:                        end if
15:                    end for
16:                    if $count = 1$ and $candidate \notin C$ then
17:                        $C \leftarrow C \cup \{candidate\}$
18:                    end if
19:                end if
20:            end for
21:        end for
22:    end for
23: end for
```

Input to the algorithm CORE-HIDM is decision table DT, and output is core C. A denotes condition attributes set. In the first step core C is initialized as empty set. Two loops in lines 2 and 4 are responsible for choosing parts of input decision table. Decision table is divided into m parts, where each of them have the size of n objects. In lines 3 and 5 chosen parts are loaded into RAM memories of hardware unit. The rest of the algorithm (lines 6 ro 21) is similar to the previously described. The only difference is that the objects for comparison are loaded from RAM memories.

2.5 Data to Conduct Experimental Research

In this paper, we present the results of the conducted experiments using two datasets: Poker Hand Dataset (created by Robert Cattral and Franz Oppacher) and data about children with insulin-dependent diabetes mellitus (type 1).

First dataset was obtained from UCI Machine Learning Repository [6]. Each of 1 000 000 records is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one decision attribute that describes the ”Poker Hand”. Decision attribute describes 10 possible combinations of cards in descending probability in the dataset: nothing in hand, one pair, two pairs, three of a kind, straight, flush, full house, four of a kind, straight flush, royal flush

Insulin-dependent diabetes mellitus is a chronic disease of the body's metabolism characterized by an inability to produce enough insulin to process carbohydrates, fat, and protein efficiently. Treatment requires injections of insulin. Twelve condition attributes, which include the results of physical and laboratory examinations and one decision attribute (microalbuminuria) describe the database used in our experiments. The data collection so far consists of 107 cases. The database is shown at the end of the paper [10]. A detailed analysis of the above data (only with the use of software systems) is in chapter 6 of the book [11].

The Poker Hand database was used for creating smaller datasets consisting of 1 000 to 500 000 of objects by selecting given number of first rows of original dataset. Diabetes database was used for generating bigger datasets consisting of 1 000 to 10 000 000 of objects. New datasets were created by multiplying the rows of original dataset. Created datasets had to be transformed to binary version. Numerical values were discretized and each attributes' value was encoded using four bits for both datasets.Every single object was described on 44 bits for Poker Hand and 52 bits for Diabetes. To fit to memory boundaries in both cases, objects descriptions had to be extended to 64 bits words filling unused attributes with binary 0's.

3 Hardware Implementation

Solution created by the authors uses combination of softcore processor and hardware unit designed to calculate the core. Diagram of the device is shown on Fig. 1. Core calculation process for large data sets is based on algorithm CORE-HIDM described in Section 2.4. The same input size of the module was used for both datasets.

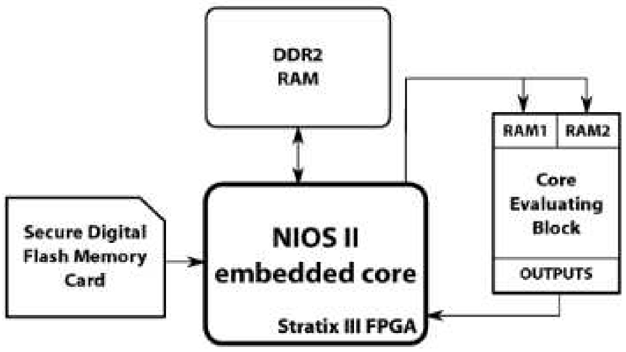

Fig. 1. Diagram of core calculation module for large datasets

Purpose of processor is to:

1. Control the process of dividing large input decision table.
2. Control the core hardware calculation block.

3. Reload the data between internal and external RAM memories.
4. Process the results returned by core hardware calculation block.

Selected processor is NIOS II. This is the proprietary softcore unit provided by Altera for its FPGAs. It is fully functional, 32-bit, RISC processor with support of modern solutions enhancing the calculation power (e.g. multi-stage pipeline, dynamic branch prediction, separate instruction and data cache, MMU, MPU, . . .).

DDR2 memory stores the large input decision table. SD card is the temporary solution for transferring data from PC to FPGA based solution. Data from SD card is copied to FPGAs DDR2 memory in the beginning of calculation process.

Core calculation unit is responsible for hardware support related to calculating subcores for given parts of decision table. Authors have used modified version of sequential core hardware calculation unit described in paper [4]. This unit has been extended in order to allow processing large datasets. To give reader better overview of prepared solution, the previous sequential core module is shortly described below.

The architecture of the sequential hardware core calculation unit shown on Fig. 2. Input of this block is decision table. Circuit consists of five functionally separated blocks:

1. **Comparators** – block of identical comparators which calculate the single row of discernibility matrix.
2. **OR-gates cascade** – block of OR-gates connected in a cascade. Every gate calculates logical OR operation on two elements: one from previous gate in a cascade and second from comparator.
3. **Singleton Detector** – block for checking if single row in discernibility matrix is a singleton (contains only one logical '1'). Outputs from this block are connected to OR-gates cascade.
4. **Multiplexer MUX** – in every turn selects the following object from decision table and puts it into the comparators in order to calculate single row of discernibility matrix.
5. **Control Logic** – responsible for storing calculation data and controls overall operations of the module.

Discernibility matrix entries are calculated by comparators very quickly, mostly because of simplicity of each comparator architecture. Then all entries goes to OR-gates cascade. The time to calculate the result depends on the size of discernibility matrix, increasing with its size. Last gate in cascade stores the result of calculations in the **CORE** register.

Sequential type of the core calculation module limits the utilization of LEs (Logical Elements) in FPGA. The disadvantage of this unit is its processing speed. Number of cycles needed to complete the calculation is equal to the number of objects in the input decision table.

Core calculation unit described previously cannot handle bigger data than size of the input register related to decision table. In order to process large datasets two blocks of fast static RAM memories were added to the solution. The control unit of the module has also been modified. Block diagram of modified unit is presented on Fig. 3.

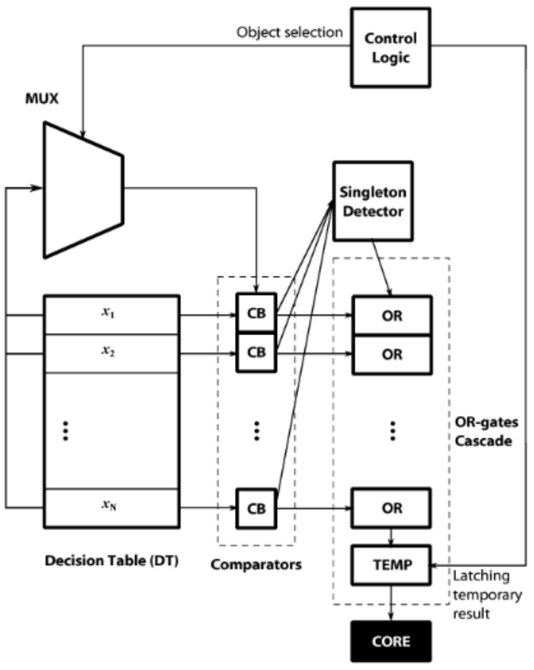

Fig. 2. Block diagram of the hardware implementation of sequential core calculation module

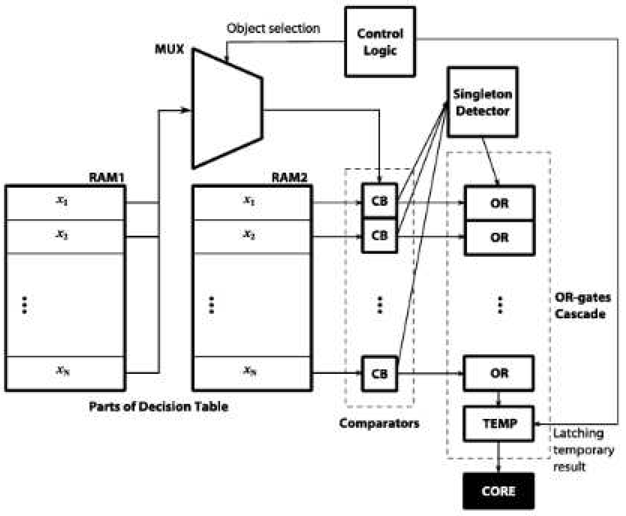

Fig. 3. Block diagram of the hardware implementation of sequential core calculation module with modifications for large datasets

RAM were created as instances of dedicated FPGA blocks (MLAB, M9k and M144k). MLAB blocks are synchronous, dual-port memories with configurable organisation 32 x 20 or 64 x 10. M9k and M144k blocks are synchronous, true dual-port memory blocks with registered inputs and optionally registered outputs with many possible configurable organisations. Both data memories used in module, denoted as RAM1 and RAM2, store parts of input decision table to comparison. In the beginning, memories contain the same part of decision table. When objects from RAM2 were compared with all objects from RAM1, then RAM2 is reloaded with next part of decision table, until decision table has any not compared elements in it. Then RAM1 and RAM2 are loaded with second part of dataset and whole process continues.

4 Experimental Results

For the research purpose the core calculation was implemented in C language. Algorithms CORE-DDM described in Section 2.2 and CORE-IDM described in Section 2.3 were used.

The results of the software implementation were obtained using a PC equipped with an 8 GB RAM and 4-core Intel Core i7 3632QM with maximum 3.2 GHz in Turbo mode clock speed running Windows 7 Professional operational system. The source code of application was compiled using the GNU GCC 4.8.1 compiler. Given times for smaller datasets are averaged for 1 000 runs of algorithm with the same input data.

Quartus II 13.1 was used for design, compilation, synthesis and verifying simulation of the hardware implementation in VHDL language. Synthesized hardware blocks were downloaded and run on TeraSIC DE-3 equipped with Stratix III EP3SL150F1152C2N FPGA chip. FPGA clock running at 50 MHz for the sequential parts of the project was derived from development board oscillator. Implemented algorithm CORE-HIDM is presented in Section 2.4.

NIOS II softcore processor, as well as most parts of embedded system were instantiated using Qsys 13.1 tool. Software for NIOS II was implemented in C language using NIOS II Software Build Tools for Eclipse IDE.

Timing results were obtained using LeCroy waveSurfer 104MXs-B (1 GHz bandwidth, 10 GS/s) oscilloscope. For longer times, hardware time measurement units instantiated inside FPGA were used.

It should be noticed, that PCs clock is $\frac{clk_{PC}}{clk_{FPGA}} = 64$ times faster than development boards clock source.

All calculations were performed using datasets described in Section 2.5 with sizes between 1 000 and 10 000 000 objects.

Table 1 presents the results of the time elapsed for hardware and software solution using indirect row-by-row discernibility matrix calculation (algorithms CORE-IDM and CORE-HIDM described in Sections 2.3 and 2.4). Table 2 presents the results of the time elapsed for hardware and software solution using direct discerniblity matrix calculation (software algorithm CORE-DDM and hardware algorithm CORE-HIDM described in Sections 2.2 and 2.4). Last two columns in both tables describe the speed-up factor without (C) and with (C_{clk}) taking clock speed difference between PC and FPGA into consideration. Abbreviations in objects number are: $k = 10^3$, $M = 10^6$.

Core generation related to software implementation using direct discernibility matrix calculation could not be performed for datasets having more than 10 000 objects because of extensive memory usage which is $\frac{n^2k}{2}$, where k denotes number of conditional attributes and n is the number of objects in decision table. One must remember that each cell of matrix is described by 32 bit value using linked list consiting of another 32 bit values, what results in extensive memory usage.

Table 1. Comparison of execution time between hardware (algorithm CORE-HIDM) and software (algorithm CORE-IDM) implementation without using discernibility matrix explicit calculation for both datasets

Objects	**Hardware -** t_H	**Software -** t_S	$C = \frac{t_S}{t_H}$	$C_{clk} = 64\frac{t_S}{t_H}$
—	$[s]$	$[s]$	—	—
		Poker Hand dataset		
1k	0.003	0.033	10.875	695.992
2.5k	0.013	0.143	11.119	711.632
5k	0.055	0.603	10.951	700.876
10k	0.207	2.410	11.623	743.851
25k	1.225	14.721	12.015	768.940
50k	4.710	58.726	12.469	798.043
100k	21.737	237.942	10.946	700.572
250k	130.947	1 515.449	11.573	740.674
500k	506.225	6 092.916	12.036	770.302
1M	1 850.523	24 313.094	13.138	840.864
		Diabetes dataset		
1k	0.003	0.018	5.911	378.325
2.5k	0.013	0.078	6.044	386.827
5k	0.055	0.328	5.953	380.980
10k	0.207	1.31	6.318	404.340
25k	1.225	8.002	6.531	417.978
50k	4.710	34.216	7.265	464.970
100k	21.737	135.309	6.225	398.390
250k	130.947	861.781	6.581	421.194
500k	506.225	3 464.821	6.844	438.043
1M	1 850.523	13 825.976	7.471	478.169
2.5M	11 729.536	81 234.655	6.926	443.242
5M	46 392.722	331 931.342	7.155	457.908
10M	185 182.545	1 293 245.545	6.984	446.952

FPGA resources utilization is fixed and is independent of the input dataset size. Datasets are divided into parts which are processed by the module. Module uses 21 562 Logical Elements (LE) of 113 600 total available.

Table 2. Comparison of execution time between hardware (algorithm CORE-HIDM) and software (algorithm CORE-DDM) implementation using explicit discernibility matrix calculation for both datasets

Objects	**Hardware** - t_H	**Software** - t_S	$C = \frac{t_S}{t_H}$	$C_{clk} = 64\frac{t_S}{t_H}$
—	$[s]$	$[s]$	—	—
Poker Hand dataset				
1k	0.003	0.296	97.209	6 221.346
2.5k	0.013	1.843	142.813	9 140.023
5k	0.055	7.496	136.044	8 706.788
>10k	0.207	N/A	N/A	N/A
Diabetes dataset				
1k	0.003	0.154	50.575	3 236.782
2.5k	0.013	1.096	84.928	5 435.413
5k	0.055	5.864	106.425	6 811.180
10k	0.207	29.021	139.961	8 957.531
>25k	1.225	N/A	N/A	N/A

Fig. 4 presents a graphs showing the relationship between the number of objects and execution time for hardware and software solution using indirect discernibility matrix calculation for both datasets. Both axes have the logarithmic scale.

Presented results show big increase in the speed of data processing. Hardware module execution time compared to the software implementation using row-by-row discernibility matrix calculation is 5 to 12 times faster, while comparing time to need to obtain full discernibility matrix is around 50 to 142 times faster in case of biggest processed dataset. If we take clock speed difference between PC and FPGA under consideration, these results are much better - average speed-up factor is 378 to 840 for indirect discernibility matrix method and up to 9 140 for method using discernibility matrix calculation first. The hardware core calculation unit was not optimized. Resuls are expected to be few times better after optimization.

We have used the same module size (configuration) for both datasets, that is why time results for the hardware are the same. It is not important what type of data is processed for hardware unit.

Let comparison of attribute value between two objects or retrieving the element from discernibility matrix be an elementary operation. k denotes number of conditional attributes and n is the number of objects in decision table. Computational complexity of software implementation for the core calculation is $\Theta(kn^2+n^2)$ according to algorithm CORE-DDM in Section 2.2 (discerniblity matrix calculation and core calculation). For algorithm CORE-IDM described in Section 2.3 it is $\Theta(kn^2)$. Using hardware implementation (CORE-HIDM), complexity of core calculation is $\Theta(n^2)$. The k is missing, because our solution performs comparison between all attributes in $\Theta(1)$ - all attributes values between two objects are compared in single clock cycle. Additionally, core module performs comparisons between many objects at time. In most cases $k << n$, so we

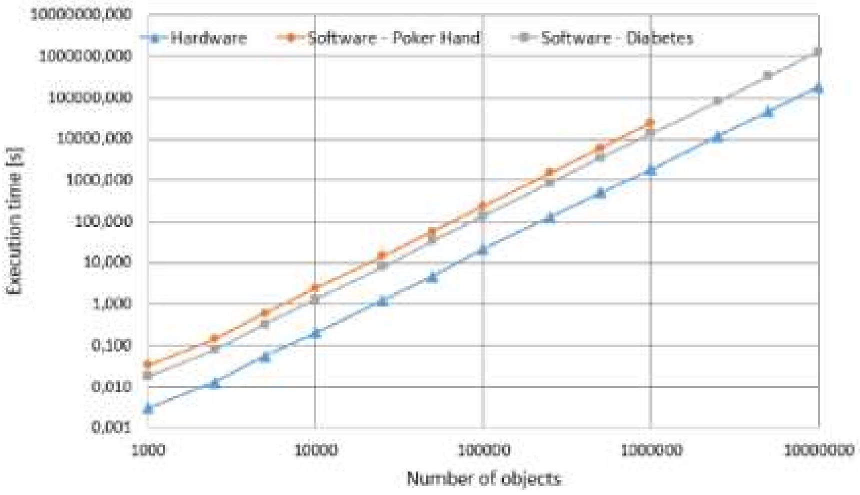

Fig. 4. Relation between number of objects and calculation time for hardware (algorithm CORE-HIDM) and software implementation (algorithm CORE-IDM) using indirect discernibility matrix calculation method for both datasets

can say, that computational complexity for software and hardware implementations are the same.

5 Conclusions and Future Research

Performing core calculations using hardware implementations gives us a big acceleration in comparison to software solution. If we compare the results to core calculation using discernibility matrix we can notice two advantages: shorter time to finish calculations and possibility of processing much bigger datasets.

Core hardware calculation unit was not optimized for performance in this paper. Processing time can be substantially reduced by increasing FPGA clock frequency and by modifying control unit to introduce triggering on both edges of clock signal. This will speed-up processing time nearly twice.

Hardware solution presented in this paper is easibly scalable. Duplicating calculation unit will improve processing speed. One must remember that this approach needs preparation of specialized control unit responsilbe for controlling performing concurrent operations (data flow of both input and output).

Another calcultion speed impact factor is the size of hardware processing module in terms of capacity to process given number of objects. Bigger size of the unit will allow to shorten calculation time.

Further research will focus on checking different sizes of core module and obtaining results of performing the calculations in parallel by multiplying units.

6 Acknowledgements

The research is supported by the Polish National Science Centre under the grant 2012/07/B/ST6/01504 (Jaroslaw Stepaniuk) and by the scientific grant S/WI/3/2013 (Tomasz Grzes and Maciej Kopczynski).

References

1. A. Kanasugi, A. Yokoyama, A basic design for rough set processor, In The 15th Annual Conference of Japanese Society for Artificial Intelligence, 2001.
2. M. Kopczyński, J. Stepaniuk: Hardware Implementations of Rough Set Methods in Programmable Logic Devices, Rough Sets and Intelligent Systems - Professor Zdzisław Pawlak in Memoriam, Intelligent Systems Reference Library 43, Heidelberg, Springer, 2013, pp. 309–321
3. M. Kopczyński, T. Grześ, J. Stepaniuk: FPGA in Rough-Granular Computing : Reduct Generation, WI 2014 : The 2014 IEEE/WCI/ACM International Joint Conferences on Web Intelligence Vol.2, Warsaw, IEEE Computer Society, 2014, pp. 364Ű-370.
4. M. Kopczyński, T. Grześ, J. Stepaniuk: Generating core in rough set theory : Design and implementation on FPGA, Lecture Notes in Computer Science Vol.8537, Berlin, Springer-Verlag, 2014, pp. 209Ű-216.
5. T. Lewis, M. Perkowski, L. Jozwiak, Learning in Hardware: Architecture and Implementation of an FPGA-Based Rough Set Machine, euromicro, vol. 1, 25th Euromicro Conference (EUROMICRO '99)-Volume 1, 1999, pp. 1326.
6. M. Lichman, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2013.
7. M. Muraszkiewicz, H. Rybiński, Towards a Parallel Rough Sets Computer In: Rough Sets, Fuzzy Sets and Knowledge Discovery, Springer-Verlag, 1994, pp. 434–443.
8. Z. Pawlak, Elementary rough set granules: Toward a rough set processor, Rough-Neurocomputing: Techniques for Computing with Words, Cognitive Technologies. Springer-Verlag, Berlin, Germany, 2004, pp. 5–14.
9. Z. Pawlak, A. Skowron, Rudiments of rough sets. Information Sciences, 177(1) 2007, pp. 3–27.
10. J. Stepaniuk, Knowledge discovery by application of rough set models, Rough Set Methods and Applications. New Developments in Knowledge Discovery in Information Systems, Physica–Verlag, Heidelberg, 2000, 137–233.
11. J. Stepaniuk, Rough–Granular Computing in Knowledge Discovery and Data Mining, Springer, 2008.
12. J. Stepaniuk, M. Kopczyński, T. Grześ: The First Step Toward Processor for Rough Set Methods, Fundamenta Informaticae Vol. 127, 2013, pp. 429-Ű443.
13. K. S. Tiwari, A. G. Kothari, Design and Implementation of Rough Set Algorithms on FPGA: A Survey, in: (IJARAI) International Journal of Advanced Research in Artificial Intelligence, vol. 3, no. 9, 2014, pp. 14–23.