Erciyes Üniversitesi Fen Bilimleri Enstitüsü Dergisi **22**(1-2) 26-36 (2006)
http://fbe.erciyes.edu.tr/
ISSN 1012-2354

MINIMAL SORTING NETWORK REALIZATION ON FPGA FOR GENETIC PROGRAMMING

Şerife SUNGUN[a], Yavuz ŞENOL*[b], Mustafa GÜNDÜZALP[b]

[a] Computer Engineering Dept., Dokuz Eylul University, 35100 Bornova, İzmir, *TURKEY*
[b] Electrical and Electronics Eng.Dept., Dokuz Eylul University, 35160 Buca, İzmir, *TURKEY*

ABSTRACT

This paper describes how the minimal sorting network (MSN) for selection operation of genetic programming can be implemented on Flex10K FPGA by using MaxPlus II tool of Altera.Four different implementations of sorting network have been realised to accelerate the time-consuming process of sorting individuals in the tournament in ascending order of their fitness values.Especially in the fourth one, with the help of the parallel processing capability of the reconfigurable computing on FPGA, total number of comparison-exchange operations has been successfully reduced. Therefore, the fitness measurement process of the genetic programming has been accelerated.

Keywords: Minimal sorting network (MNS), Reconfigurable computing, Genetic programming, FPGA.

GENETİK PROGRAMLAMA İÇİN FPGA ÜZERİNDE EN KÜÇÜK SIRALAMA AĞININ GERÇEKLENMESİ

ÖZET

Bu makale, Altera MaxPlusII aracını Flex 10K FPGA üzerinde kullanarak genetik programlamanın seçim işlemi için en küçük sıralama ağının nasıl gerçeklenebileceğini anlatmaktadır. Zaman alıcı bir işlem olan turnuvadaki bireylerin uygunluk değerlerinin azalan bir düzende sıralanması işleminin hızlandırılmasını sağlamak amacıyla sıralama ağının dört farklı uygulaması gerçeklenmiştir.Özellikle dördüncü sırada gerçeklenen ağ ile FPGA üzerindeki yeniden yapılandıralabilir bilişimin paralel işleme yeteneği yardımıyla karşılaştırma-yerdeğiştirme işlemlerinin toplam sayısı başarılı bir şekilde azaltılmıştır. Böylece, genetik programlamada uygunluk ölçüm işlemi hızlandırılmıştır.

Anahtar kelimeler: En küçük sıralama ağı, Yeniden yapılandırılabilir bilişim, Genetik programlama, FPGA.

***E-posta:** yavuz.senol@eee.deu.edu.tr

1. INTRODUCTION

Genetic Programming (GP) is the answer to the question, how a computer can learn to solve problems without being explicitly programmed 1. . This is the definition of the automatic programming 2. and it is performed by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations. The genetic algorithm starts with a population having certain number of individuals N. The next step is to choose M individuals to use in a tournament (M<<N). A fitness value is assigned to each individual in the tournament using the fitness measure. Then, the individuals are ranked in ascending order with respect to fitness measure. Finally, the new M/2 individuals are generated form fittest M/2 individuals by using genetic operators, reproduction, crossover, mutation, to replace with the M/2 worst individuals. The process of ranking the individuals and separating the M/2 fittest individual is performed by selection operation. The execution of the algorithm goes on until to find a program or individual that solves the problem.

Computationally intensive part of GP is the fitness measurement of each individual in each generation of the evolving population since each individual is to be run for many different combinations of inputs. Implementation of the GP on traditional Von Neuman type architecture requires M times calculations of fitness measurements of each individual for a tournament 3. .

The intense computational requirement of GP applications can be met with dynamic Field Programmable Gate Arrays (FPGA) 4. that are massively parallel computational devices. Now, there is a need for a sorting algorithm to realise the ranking of fitness values in selection operation that is the consequence of competition among individuals in a population. The hardware implementations of general purpose sorting algorithms 5. bring many difficulties, inefficiencies, and also increase the complexity of the system. The minimal sorting network 6. that is an algorithm for sorting small number of items is the best way of realisation of ranking process. In addition, the use of FPGA allows the parallel implementation of the algorithm. Therefore, the complete run time of the population evolution has been dramatically reduced.

Section 2 describes reconfigurable computing and the technology of FPGA. Section 3 gives definition of minimal sorting network. Section 4 gives realisation of minimal sorting algorithm in three subsections. Section 5 describes the results. Finally, section 6 gives the conclusion.

2. FPGA

Field Programmable Gate Array technology has emerged as the principle technology behind reconfigurable/configurable computing (RCC) due to it's electronically re-programmable characteristics 7. , wide cross-section of computing structures containing memory and logic, speed and density.

An FPGA blurs the distinction between hardware and software. The "hardware" fabricated by a foundry is general purpose; consequently they are mass produced and affordable. The logic of an FPGA is customized by loading a "configuration," which is similar to a software program. The resulting FPGA combines the best features of both hardware and software. It is faster and smaller than truly general-purpose hardware. It has also smaller Non-recurring Engineering (NRE) costs and transition costs, since it can be easily recustomized — without modifying the hardware — by designing and loading a different configuration. A reconfigurable computer could be upgraded, or even reconfigured for a completely different function, from a remote location.

FPGAs have an architecture similar to Mask-Programmable Logic Devices in that they consist of an array of logic cells and wire segments 7. . The functionality of the logic cells and the inter-connection between the blocks are user programmable. The programming process of FPGAs is more closely related to traditional Programmable Logic Devices (PLD) than Mask-Programmable technologies. On the other hand, the routing and logic implementation of a FPGA is more varied than that available from a PLD. The central advantage of FPGA technology lies in their real-time re-programmability and an ability to provide an explicit hardware representation of the application algorithm, thus supporting fine grained parallelism and application specific instructions.

FPGAs typically consist of a regular matrix of programmable interconnect with at least two types of interconnection resources linking logic cells to logic cells or logic cells to I/O blocks. Manufacturers differ in their respective approaches to programmability, logic cell structure and routing methodology. Depending on the logic cell, the device density and functionality varies. Moreover, the device logic cell will also influence the flexibility of the routing and predictability of any timing analysis.

3. MINIMAL SORTING NETWORK

A sorting network is an algorithm for sorting items. Sorting process is performed with a sequence of comparison-exchange operations that are executed in a fixed order.

Figure 1 shows a sorting network for four items that are a, b, c and d. The numbers within the circles represent states for each comparison-exchange operation.

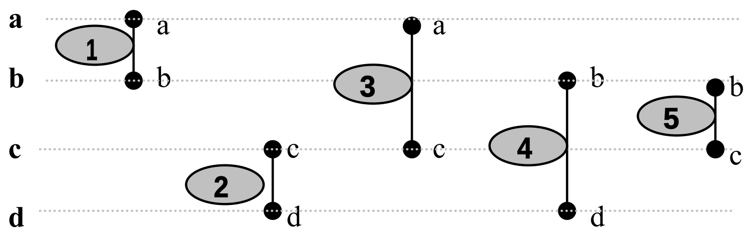


Figure 1. Minimal sorting network for four items.

The items, a, b, c and d to be sorted are given at the left end of the horizontal lines. The first vertical line connecting two upper horizontal lines indicates that items a and b are to be compared and exchanged, if necessary, so that the larger one is always at the bottom. This step and the next consecutive three steps cause the largest and smallest items to be routed down and up, respectively. The fifth step ensures that the remaining two items end up in the correct order and therefore correctly sorted output is obtained at the end of the fifth step. A five-step network is known to be minimal for four items 5. .

Sorting networks are independent of their data in the sense that they always perform the same fixed sequence of comparison-exchange operations. Sorting networks are also more efficient for sorting small numbers of items than the well-known non-oblivious sorting algorithms such as Quicksort 8. .

4. MINIMAL SORTING ALGORITHM

To efficiently sort a small number of items, we neither attempt to directly map a general purpose-sorting algorithm into hardware or use a brute force hardware implementation. Instead, the concept of minimal sorting networks is used 6. , where the principle objective of a Minimum Sorting Algorithm (MSA) is to sort items by using the minimal number of pairwise comparison-exchange operations. In the case of 4 items the minimum number of such operations is five. Table 1 summarises the processing steps of the MSA.

Table 1. Example of 4 items sort using MSN.

Items	**Initial Values**	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**Sorted Items**
A	7	5		4			4
B	5	7			6	5	5
C	6		4	5		6	6
D	4		6		7		7

As an example, consider the case in which the four items take the following values: a(7), b(5), c(6), d(4). Table 1 illustrates the progress of the search for this example. However, from a hardware perspective this is expensive, because exchange operations of long integer values require more hardware components. If the index values that are pointing to the corresponding items are used then the implementation will be independent of item value sizes. This process is summarised in Table 2 for 4 items.

As described in section 1 there are four fitness values for individuals in each tournament. This well suit to the MSA to select the best two individuals. Two bits index is enough for four fitness measurements.

One way of sorting items using index values can be implemented as given in Figure 2. Here, counters represent fitness values that are the items to be sorted. Each counter has an associated 2-bit address {00, 01, 10, 11}. The contents of counters are loaded into A and B inputs of the comparator in the order given in Figure 1. The main operation in this implementation is the comparison of fitness values A, and B and exchange of corresponding index values, that are represented as a, b, c, and d. If A is greater than B, the output of the comparator is set to true (T), and then the index values of corresponding counters are swapped.

Table 2. Example of 4 items sort using MSN and indexed location.

Items	**Index Values**	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**Sorted Items**
7	0	**1**		**3**		3	4
5	1	**0**			**2**	**1**	5
6	2		**3**	**1**		**2**	6
4	3		**2**		**0**	0	7

F represents false situation and in this case, there is no swap. TR is a dummy register that is used during the swap operation of index values. The content of the first value is saved into TR before it is loaded with the new value. The final piece of hardware, state machine, represents the control circuitry responsible for coordinating the entire process (pairwise loading of counter values to the 'm' bit comparator and manipulating the order of the counters' indexes in the registers accordingly). Thus, the contents of registers from a to d following the execution of the sort algorithm indicates the required ascending ordered results.

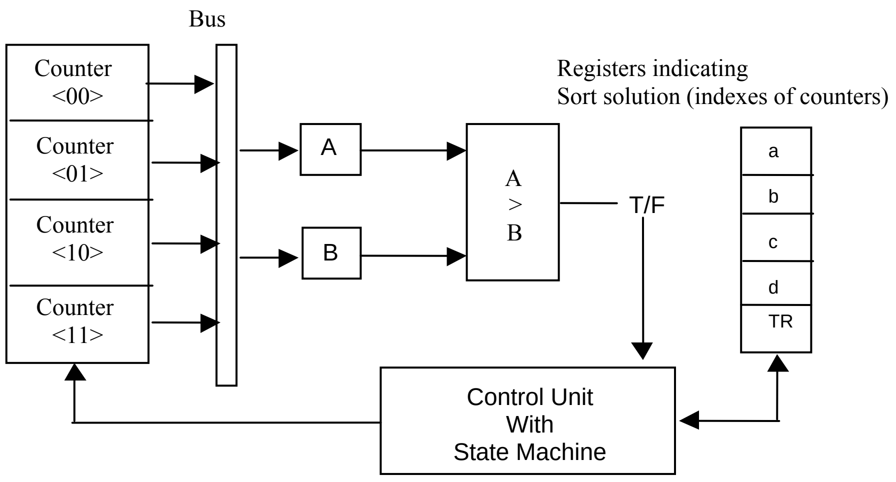

Bus
Counter <00>
Counter <01>
Counter <10>
Counter <11>
A
B
A > B
T/F
Registers indicating Sort solution (indexes of counters)
a
b
c
d
TR
Control Unit With State Machine

Figure 2. Minimum sorting network hardware.

Although it is not illustrated in Figure 2 the comparator and four m-bit counters are linked by a suitable bus structure. This requires a pair of 4 to 1 multiplexer for each bit of counters, one for A and one for B input of the comparator. It is also possible to use m-bit tri-state bus in place of multiplexers.

4.1 Implementation of MSN on FPGA, Sorter-1

The specific action of the controller unit given in Figure 2 is expressed as a Finite State machine using the algorithm in Figure 3. This algorithm provides a particularly efficient mapping to the FPGA reconfigurable computing platform, especially when the one-hot methodology is used 9. . For one-hot implementation of the sorter in Figure 3 eleven D flip-flops labelled with states names are required. Table 3 contains the information needed to produce the inputs to the one-hot flip-flops. Then the MSA described in the previous section is implemented on FPGA platform to sort fitness values of each tournament with four individuals.

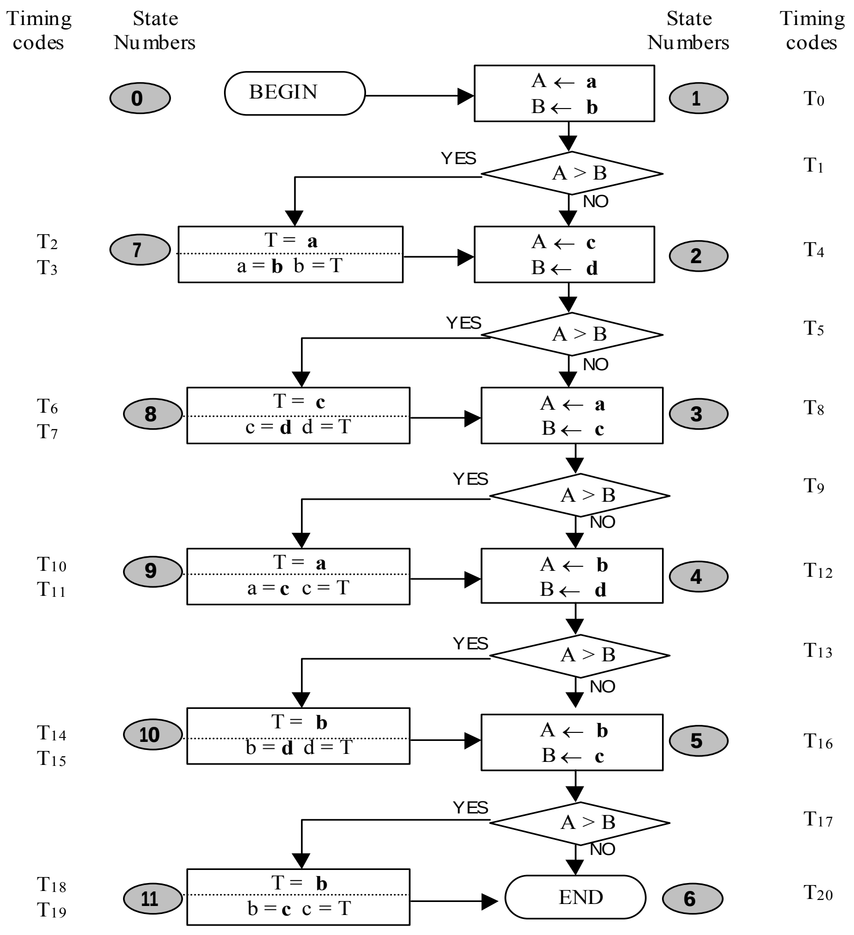


Figure 3. Sorting Algorithm for four values (Sorter–1).

Instead of the registers containing the fitness values of four individuals, the index addresses of them are used in the sorting process. By doing so, the sorting device becomes domain size independent.

The hardware realization of selection operation of GP by using MSN here named as Sorter-1 was obtained on Flex10K device using MaxPlusII platform. This realization is summarised in Figure 3. The realized unit then was tested for several input values. The following timing results were obtained when the clock period is equal to T.

Time requirement	: 20 T (the worst case, If all comparison results are TRUE)
	: 15 T (the best case, If all results are already sorted)

Table 3. State transition data for one-hot implementation of the Sorter-1.

NEXT STATE	PRESENT STATE	CONDITION	FLIP-FLOP FUNCTION
1	0		$D_1 = 1$
2	1	$\overline{COMP}$	$D_2 = Q_1.\overline{COMP} + Q_7$
	7		
3	2	$\overline{COMP}$	$D_2 = Q_2.\overline{COMP} + Q_8$
	8		
4	3	$\overline{COMP}$	$D_4 = Q_3.\overline{COMP} + Q_9$
	9		
5	4	$\overline{COMP}$	$D_5 = Q_4.\overline{COMP} + Q_{10}$
	10		
6	5		$D_6 = Q_5.\overline{COMP} + Q_{11}$
	11		
7	1	$COMP$	$D_7 = Q_1.COMP$
8	2	$COMP$	$D_8 = Q_2.COMP$
9	3		$D_9 = Q_3.COMP$
10	4	$COMP$	$D_{10} = Q_4.COMP$
11	5	$COMP$	$D_{11} = Q_5.COMP$

4.2 Sorter-2 Design

In MSN algorithm of the Sorter-1 the use of temporary register TR for swapping the index addresses causes the design to spend 5 extra T clock periods. This waste of time can be eliminated by exploiting the advantage of consecutive synchronous operation of D flip-flops. In new design Q output of the first D flip flop connected to the input of the second D flip flop. The Q output of the second D flip flop is also connected to the D input of the first one. Both are derived with the same clock. With this realization, each of the state numbers 7, 8, 9, 10 and 11 can be performed within a single clock period T instead of two clock periods. The logical explanation and illustration of the new design is given in Table 4 and, Figure 4 respectively.

Table 4. Transition table for swap function.

PRESENT STATE	COMP. INPUT		NEXT STATE	NEXT REGISTER VALUES			
	A	**B**		**a**	**b**	**c**	**d**
1	A	b	7	B	A		
2	C	d	8			B	A
3	A	c	9	B		A	
4	B	d	10		B		A
5	B	c	11		B	A	

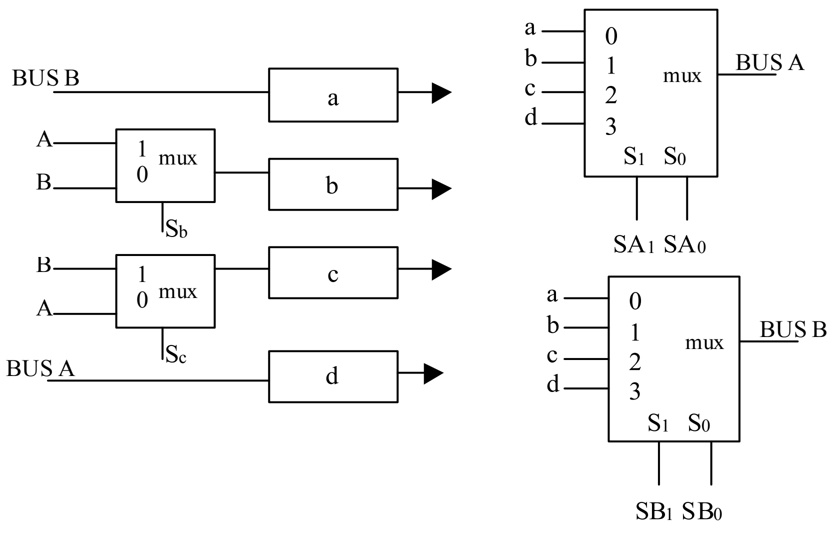


Figure 4. Block diagram of swap device (for sorter-2).

4.3 Sorter-3 and Sorter-4 Design

In sorting algorithm 2, c and d are compared after a and b comparison. Due to the use of a single comparator. By using an additional comparator and a multiplexer within the logic circuit, two individual discrete comparison steps can be performed at the same time instance in parallel. This new sorting algorithm was realized with a new sorter named as sorter 3. With this modification it is possible to have the shortest response time to sort the fitness cases. The processing states for sorter 3 can be seen in Figure 5.

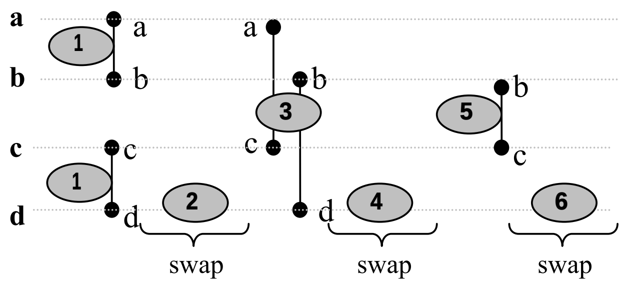


Figure 5. Processing states for Sorter-3.

As seen in Figure 5 a-b and c-d comparisons (state-1) are performed at the same time in parallel. The comparison results are shown as Comp-X and Comp-Y, respectively. T2, T4 and T6 time instances are used for swap operations according to the comparison results. The swap operation takes place only if the first item is greater than the second. Whether the swap takes place or not, time instances reserved for swap operation are always consumed. Therefore, the total time spent for completing sorting process is always equal to 6 clock cycles. It is also possible to realize a more efficient sorter in terms of time consumption. However, this can be obtained with a cost of more complicated system. This idea was also realised with a new sorter and that is called as Sorter-4. In addition, this idea is possible for only 25% of whole processing time of comparisons. The more the numbers are in order the shorter execution time for sorter-4.

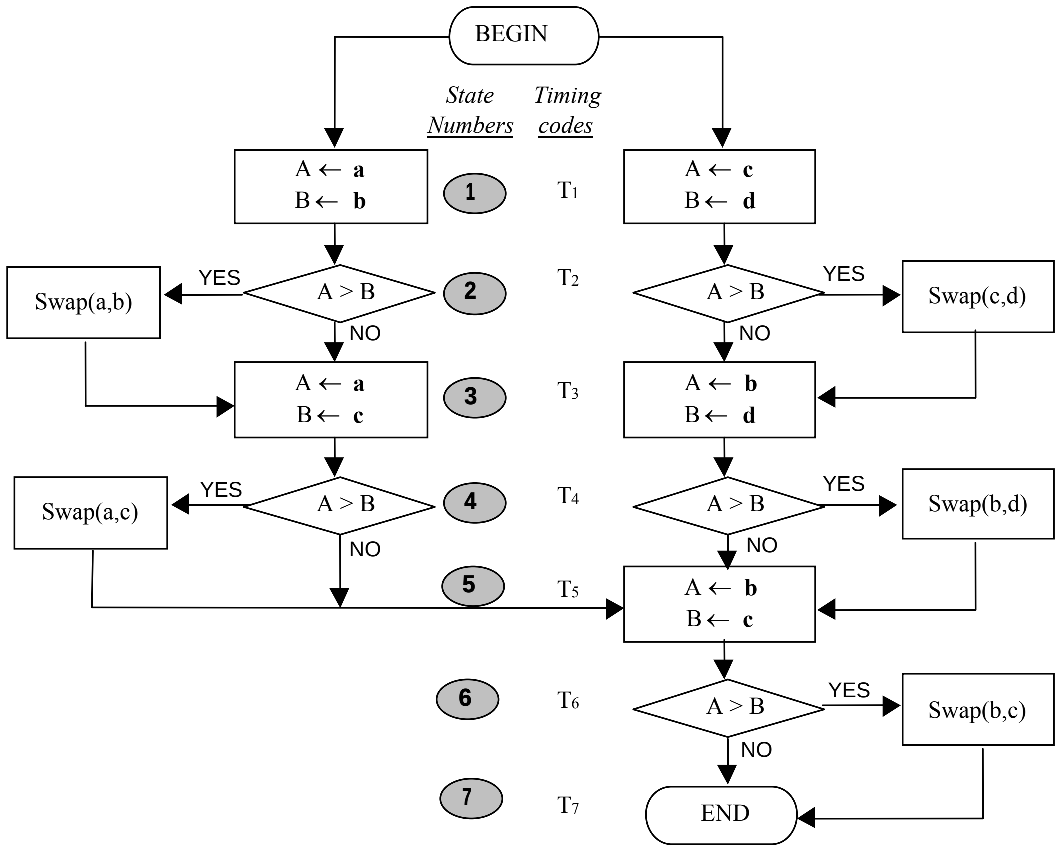
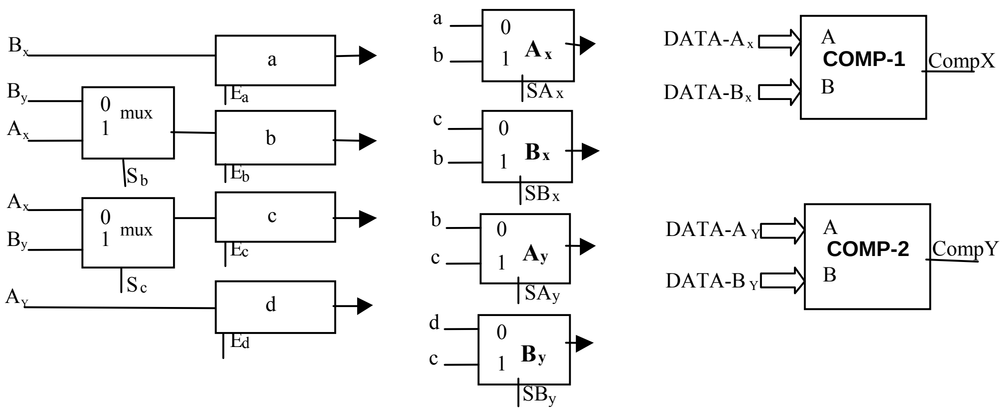


Figure 6. Sorter 3.

State transitions shown in Figure 6 are given in Table 5 with corresponding swap operations. Table 6 gives the explanations of symbols used for Sorter 3.

Table 5. Transition table for swap function of Sorter-3.

PRESENT STATE	COMP-X INPUT		COMP-Y INPUT		NEXT STATE	NEXT REGISTER VALUES			
	A_x	**B_x**	**A_y**	**B_y**		**a**	**b**	**c**	**d**
1	a	b	c	d	2	B_x	A_x	B_y	A_y
3	a	c	b	d	4	B_x	B_y	A_x	A_y
5			b	c	6		B_y	A_y	



$SA_x = T_5$	$Sb = T_1$	$E_a = (T_{2+} T_4)$ CompX
$SB_x = T_1$	$Sb = T_1$	$E_b = T_2$. CompX + $(T_{4+} T_6)$ CompY
$SA_y = T_1$		$E_c = T_2$. CompY + $(T_{4+} T_6)$ CompX
$SB_y = T_5$		$E_d = (T_{2+} T_4)$ CompY

Figure 7. Block diagram of swap device (for sorter-3).

Table 6. Explanation of symbols used for Sorter-3 circuit.

Symbol	Explanation	Bit number
CompX, CompY	Outputs of Comparator with A and B inputs ("1" if A>B)	1
DATA-A_x,DATA-B_x DATA-A_Y,DATA-B_Y	A_x, B_x, A_y, B_y are address of tournament results	4
a, b, c, d	Address index registers	2
E_a, E_b, E_c, ve E_d	Enable for writing to address registers	1
A_x, B_x	Address of data which is sent to inputs (A and B) of the first comparator (Comp-1)	2
A_y, B_y	Address of data which is sent to inputs (A and B) of the second comparator (Comp-2)	2
SA_x,SB_x,SA_y,SB_y	Selection input of multiplexer determining address of the data to comparator	1

5. RESULTS

In this work an optimal 5-step 4-sorter was designed that has less steps than the sorting network described in 1962 O'Conner and Nelson patent on sorting networks. MSN has been realised in four different ways and named as the sorter-1, sorter-2, sorter-3, and sorter-4 respectively. Figure 8 shows the comparison results in terms of swap numbers and execution times. It is clearly seen that in all cases the sorter-4 takes less execution time than all other sorters. Moreover, the sorter-3 is always executed at constant time duration since all swap operations execution times are spent without considering the requirement of swap operation. The minimum execution time for sorter-4 is 1.1μsec. The success of sorter-4 totally depends on how the numbers are located. This is illustrated as optimistic and pessimistic in Figure 8. For two swap operations the pessimistic corresponds to the scenario where two swaps takes place at different steps. Whereas the optimistic assumes that two swap operations are located at the same step.

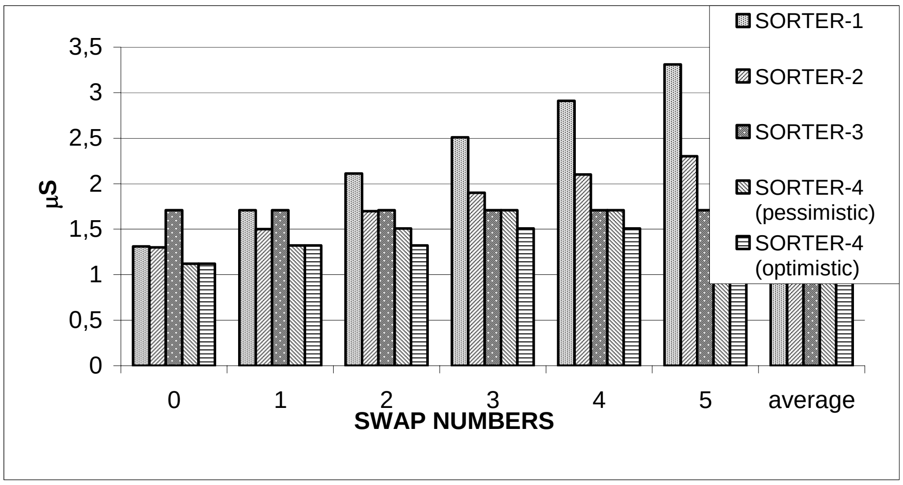


Figure 8. Comparison of the sorters.

6. CONCLUSION

This paper gives the realisation of minimal sorting network for selection operation of genetic programming. The implementation of MSN was realised on Flex10K FPGA by using MaxPlus II tool of Altera. Firstly, 5-step 4-sorter was implemented as described by O'Conner and Nelson, and the sorter was named as sorter1. With the advantage of using FPGA's sorter-1 was improved by eliminating the use of the temporary register as described in section 5.2. Finally, parallel processing capability of FPGA's allowed further improvement of the sorter-1. The final sorter that has the minimum execution time for obtaining fully ordered numbers was named as sorter-4. All designed sorters are independent of item sizes.

ACKNOWLEDGMENTS

We acknowledge TUBITAK for supplying the computational facilities, on which this research was conducted by TUBITAK research grant 199E023.

REFERENCES

1. Banzhaf, W, Nordin, P., Keller, R.E., Genetic Programming-An Introduction on the Automatic Evolution of Computer Programs and Its Applications, pp. 106-123, Morgan Kaufmann Publishers, San Francisco, 1998.
2. Koza, J.R, Andre, D., Evolution of Both the Artchitecture and the Sequence of Work-Performing Steps of a Computer Program Using Genetic Programming with Architecture-Altering Operations, AAAI Fall Symposium on Genetic Programming, pp 50-60, November 10-12, 1995.
3. Heywood, M.I., Zincir-Heywood, A.N., Register Based Genetic Programming on FPGA Computing Platforms, EuroGP2000, 3rd European Conference on Genetic Programming, Lecture Notes in Computer Science. (vol. 1802), pp 44-59, 2000.
4. Zhang, X , Ng, K.W. A Review of High-Level Synthesis for Dynamically Reconfigurable FPGAs, Elsevier Science-Microprocessors and Microsystems, 24, 199-211, 2000.
5. Koza, J.R., Evolving Computer Programs using Rapidly Reconfigurable Field-Programable Gate Arrays and Genetic Programming, FPGA'98 Sixth International Symposium on Field Programmable Gate Arrays, pp. 209-219, February 22-24, 1998.
6. Knuth, D.E., The Art of Computer Programming: Sorting and Searching, volume 3. Addison-Wesley Publishing

Company, Reading: MA, 1973.
7. Salcic, Z., Smailagic, A. Digital Systems Design and Prototyping Using Field Programmable Logic, 2nd ed., Kluwer Academic Publishers, 2000.
8. Koza, J.R., Genetic Programming II – Automatic Discovery of Reusable Programs, The MIT Press, London, 1998.
9. Alfke,P., New,B., Implementing State Machines in LCA devices, In The Programmable Logic Data Book XAPP027.001, San Jose, CA: Xilinx, Inc., pp 8-172 - 8-172, 1996.