Intrinsic Evolution of Sorting Networks: A Novel Complete Hardware Implementation for FPGAs

Jan Kořenek and Lukáš Sekanina

Faculty of Information Technology, Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
{korenek, sekanina}@fit.vutbr.cz

Abstract. A specialized architecture was developed and evaluated to evolve relatively large sorting networks in an ordinary FPGA. Genetic unit and fitness function are also implemented on the same FPGA. We evolved sorting networks up to N=28. The evolution of the largest sorting networks requires 10 hours in FPGA running at 100 MHz. The experiments were performed using COMBO6 card.

1 Introduction

Sorting networks (SN) have recently been recognized as potentially suitable objects for the evolutionary design and optimization [2,4]. They are also interesting from a hardware viewpoint because of their regular and combinational nature suitable for pipeline processing. For instance, Koza et al. have used genetic programming to evolve small sorting networks directly in a field programmable gate array (FPGA) [6].

Similarly, effective hardware implementations of median circuits are crucial for high-performance signal processing. By the median circuit we mean a circuit calculating the median value from its inputs. That can be accomplished either by reading the middle value of the output sorted vector calculated by a corresponding sorting network or by designing of a specialized median circuit [10]. All the mentioned approaches share a common feature – the time of a candidate SN evaluation grows exponentially with growing number of inputs.

The objective of this paper is to evolve as large as possible sorting networks in a reasonable time. In order to perform these investigations, a novel virtual reconfigurable circuit architecture optimized for evolution of sorting networks has been proposed and implemented on the top of a conventional FPGA. The architecture is configured using the chromosomes generated by evolutionary algorithm which is implemented on the same FPGA. The chromosome encodes the functions performed by virtual programmable elements; however, the interconnection of these elements remains fixed. Since the FPGA implementation of the programmable element is inexpensive, it can operate as a wire and thus in fact the evolutionary algorithm also modifies the interconnection. As the fitness calculation is also carried out in the same FPGA, we can benefit from pipeline processing allowing reasonable time of a candidate circuit evaluation. The main

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 46–55, 2005.

feature of the proposed implementation is that everything is implemented in a cutting-edge reconfigurable hardware platform available today. For the experiments presented we utilized the COMBO6 card developed in the Liberouter project [7]. A personal computer is used only for a communication with the COMBO6 card, i.e. for reading the results. We evaluated various variants of the evolvable sorting network, including the size of the virtual reconfigurable circuit and the parameters of the evolutionary algorithm. The main objective is to find as large correct sorting network as possible in minimal time; neither area nor delay are optimised.

The paper is organized as follows. Section 2 briefly introduces sorting and median networks and evolutionary approaches to their design. In Section 3 the proposed complete hardware implementation is described. Results of synthesis for COMBO6 are reported in Section 4. Section 5 summarizes the obtained results. Section 6 deals with discussion of the obtained results and directions of future work. Conclusions are given in Section 7.

2 A Brief Survey of Relevant Research

2.1 Sorting and Median Networks

A *compare–swap* of two elements (a, b) compares and exchanges a and b so that we obtain $a \leq b$ after the operation. A sorting network is defined as a sequence of compare–swap operations that depends only on the number of elements to be sorted, not on the values of the elements [5]. The advantage of the sorting network is that the sequence of comparisons is fixed. Thus it is suitable for parallel processing and hardware implementation, especially if the number of sorted elements is small. Figure 1 shows an example of a sorting network.

The number of compare–swap components and the delay are two crucial parameters of any sorting network. Table 1 shows the number of compare–swap components and delay of the best currently known sorting networks (for $N \leq 16$). These values are derived from the Knuth's book [5] and from paper [1].

Having a sorting network for N inputs, the *median* is simply the output value at the middle position (odd Ns only). For example, efficient calculation of the

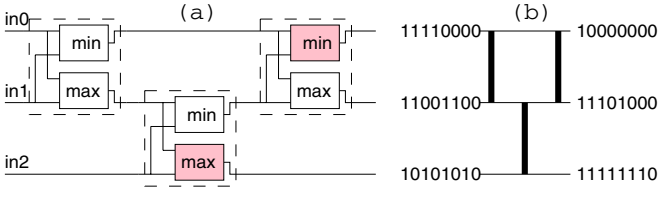

Fig. 1. (a) A 3-sorting network consists of 3 components, i.e. of 6 subcomponents (elements of maximum or minimum). A 3-median network consists of 4 subcomponents. (b) Alternative symbol.

Table 1. Parameters of the best-known sorting networks

N	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Delay	0	1	3	3	5	5	6	6	7	8	8	9	10	10	10	10
Comparators	0	1	3	5	9	12	16	19	25	29	35	39	45	51	56	60

median value is important in image processing where median filters are widely used with N = 3x3 or 5x5 [9]. Note that the popular implementation of the 9-median circuit in an FPGA proposed by Smith is also area-optimal (in terms of the number of components) [14].

The *zero–one* principle helps with evaluating sorting networks (and median circuits as well). It states that if a sorting network with N inputs sorts all 2^N input sequences of 0's and 1's into nondecreasing order, it will sort any arbitrary sequence of N numbers into nondecreasing order [5]. This principle will be utilized in the fitness function.

2.2 Evolutionary Approaches

Some of sorting and median networks were (re)discovered using evolutionary techniques [2,4,6,10]. Evolutionary techniques were also utilized to discover fault-tolerant sorting networks [12]. Since fitness function is typically based on the use of the *zero–one* principle, the evolution of larger sorting networks is not scalable (because the size of the test set doubles by increasing the number of inputs by 1). It is usually impossible to obtain the perfect solution (that sorts all 2^N input vectors) if only a subset of input vectors is utilized during the evolutionary design [3].

2.3 Intrinsic Evolution in FPGAs

In order to speed up candidate networks evaluation, Koza et al. have evaluated candidate sorting networks in Xilinx XC6216 FPGA. Genetic programming utilized for designing sorting networks was running in PC. For example, using population size 60k, minimal 8-SN was evolved on generation 58, and using a population size 100k, minimal 9-SN was evolved on generation 105. The evolution of minimal 7-SN required 69 minutes on the FPGA (31 generations, population size 1000). The evaluation of a candidate sorting network in XC6216 FPGA was 46 times faster than in Pentium 90MHz [6].

Some other FPGA-based implementations of complete evolvable systems have been proposed for various problems in the recent years. There are some examples: In Tufte and Haddow's approach only register values representing coefficients of a digital filters were evolved [15]. Sloarch and Sharman [13] have proposed intrinsic evolution of small combinational circuits in FPGA. An automatic feature identification algorithm that utilizes functional level operators was developed for multi-spectral images in [8]. The concept of virtual reconfigurable circuit (i.e. the second level of reconfiguration implemented in a conventional

FPGA) was utilized in papers [11,16]. We use a hardware implementation of evolutionary algorithm because it overcomes the bottleneck introduced by slow communication between the FPGA and a personal computer (in which the evolution is usually performed). The proposed architecture for evolution of large sorting networks is based on Koza's seminal work [6] and Sekanina and Friedl's complete hardware implementation of an evolvable combination circuit [11]. Unlike in Koza's approach evolutionary algorithm will be implemented in hardware.

3 The Proposed Architecture

The proposed architecture for sorting network evolution consists of four basic components—Control Unit, Fitness Unit, Genetic Unit and Virtual Reconfigurable Circuit Unit (VRC Unit). All the units are implemented on a single FPGA. The block structure of the architecture is shown on the Figure 2.

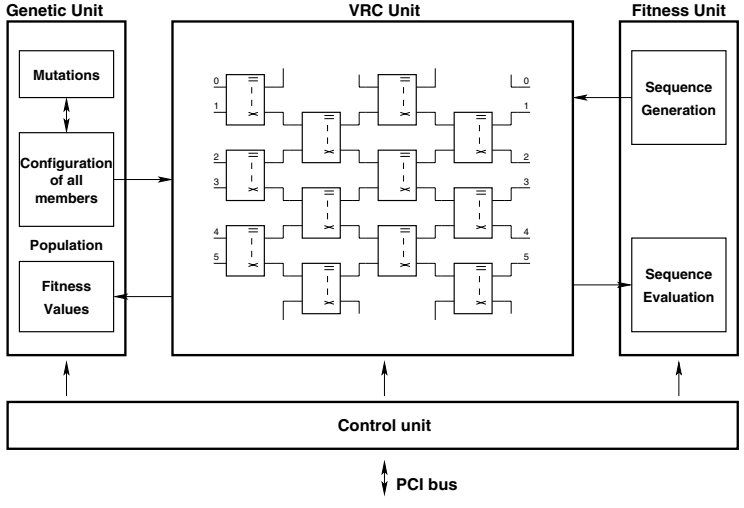

Fig. 2. Block structure of the proposed architecture

All operations are controlled by the control unit which is connected to PCI bus and executes the commands entered by user. For example, evolution can be started or stopped, the number of iterations can be specified, etc. The Genetic Unit executes genetic algorithm and contains all population members. VRC unit is a reconfigurable circuit in which the evolution is performed. That is implemented as a second level of reconfiguration on the FPGA. VRC is configured using chromosomes generated by Genetic Unit.

3.1 Genetic Unit

Genetic algorithm is based only on the mutation operator (bit inversion); crossover is not taken into account in this paper. We are going to investigate its

usefulness in next research. Population size is configurable. The new population is always generated from the best member of the previous one. Genetic algorithm operates in following steps: (1) Initialization Unit generates the first population at random (Linear Feedback Shift Register seeded from software is utilized). (2) Mutation Unit changes a given number of genes (bits) of a population member (this number is configurable) and the modified member is loaded into the VRC—it represents an image operators. (3) Genetic Unit is waiting for the evaluation performed by Fitness Unit and if the fitness value obtained is better than the parent's fitness then the chromosome replaces its parent. (4) This is repeated until the appropriate number of generations is produced.

3.2 VRC Unit

The unit consists of VRC elements that can perform different operations according to the selected configuration. Figure 3 shows its inteface.

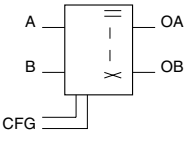

Fig. 3. VRC element architecture

Each element consists of two input and two output ports (everything is 1 bit). The functionality is determined by two configuration bits, which are used to select one of four different operations. All operations can be performed in one clock cycle. The result is stored into the register (local in each element) which offers to use VRC elements in a pipelined structure.

Sorting networks are usually composed of Compare&Swap components with various interconnection. For this reason, the proposed VRC element is designed to perform Compare&Swap operation. Alternatively, it can operate as a wire or cross-wire. The relation between configuration bits and functionality is shown in the following list:

- 00 – direct connection from inputs to outputs
- 01 – Compare&Swap operation – maximum on the upper output
- 10 – Compare&Swap operation – minimum on the upper output
- 11 – cross connection inputs to outputs

The VRC unit is composed of VRC elements in a fixed structure which is shown in Figure 3. Although the interconnection is invariable it can be changed if CFG = 00 or 11 is selected. The architecture is different for even Ns (left side in Fig. 4) and odd Ns (right side in Fig. 4).

A VRC element is connected to the four nearest neighbors. Even columns have inputs and outputs shifted by one item. This one-item-shift is necessary to

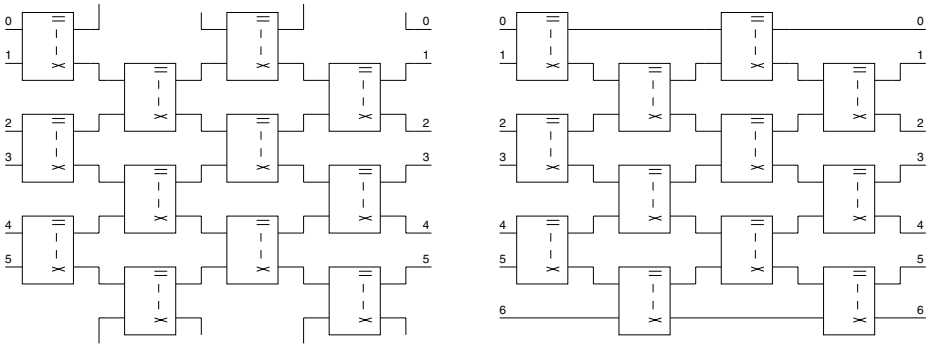

Fig. 4. Odd and even VRC array architecture

establish interconnection between arbitrary two Compare&Swap components or to compare different items. The architecture for odd Ns differs only in the first and last row where the remaining item is always connected to the next column via the synchronization register.

The proposed VRC array supports fully pipelined processing because every element contains synchronization register for the output values. Therefore, VRC array can produce one result in a clock cycle. Unlike VRCs in [11,16] this VRC element architecture is also optimized for hardware resources. Only two LUTs have to be utilized to create one VRC element. This optimization enables to fit a large VRC array within the single chip and thus to find sorting networks with many inputs.

3.3 Fitness Unit

The Fitness unit is used to generate unsorted sequences and evaluate results calculated by VRC unit. N-bit counter generates the unsorted input vectors, i.e. all possible combinations over N bits. In the evaluation process all vectors unsorted by VRC have to be identified.

The fitness value is defined as the number of unsorted vectors coming from the VRC unit. The unsorted vector is detected if for any item a_i in the vector $a_i < a_{i+1}$ does not hold. This is performed in parallel by a set of comparators in only one clock cycle. The number of unsorted vectors is stored in a counter which is incremented when an unsorted vector is identified. The content of the counter is presented as a fitness value which is valid after all input vectors are evaluated. In this paper, we are interested only in functionality of sorting networks; the number of components is not optimized.

4 Results of Synthesis

The proposed architecture is designed for evolution of sorting networks having different number of inputs. From this point of view, the size of the VRC array

Table 2. XC2V-3000 FPGA utilization for various VRC and N

N length	VRC Elements	Slices	Chip Utilization
10	5x30	1731	12 %
12	6x36	2262	15 %
14	7x42	2735	19 %
16	8x48	3207	22 %
18	9x54	3795	26 %
20	10x60	4431	30 %
22	11x66	5675	39 %
24	12x72	6412	44 %
26	13x78	7314	51 %
28	14x84	8173	57 %
30	15x90	9232	64 %
32	16x96	10223	71 %
36	18x108	12468	86 %

has to be scalable and support as many rows and columns as possible. On the other hand, the design with VRC array has to fit within the single chip. For this reason, the proposed VRC element was optimized for the hardware resources utilization.

The implementation of this architecture and evaluation of results have been performed on available COMBO6 hardware platform. COMBO6 is a PCI card equipped with a Field Programmable Gate Array XC2V-3000, TCAM memory, static and dynamic Random Access Memories and some other components.

The synthesis results in Table 2 show hardware resources utilization of the XC2V-3000 FPGA for different size of the VRC array and N. It can be seen that the FPGA utilization for the largest VRC array 18×108 is only 86 % without performance lost.

5 Experimental Results

Various VRC architectures were synthesized up to $N = 20$. For each VRC size, 80 independent experiments are performed and analyzed. We used four-member population and produced 50000 generations. Only mutation operator is used; 4 bits are inverted in chromosome in average.

Table 3 shows that it is possible to find correct sorting networks (for relatively large Ns) in a reasonable time. The first column shows the vector length (i.e. N). The number of correct sorting networks discovered out of 80 runs is reported in the second column. In 4th column there is the average number of generations needed to find the perfect solution (and its standard deviation in 5th column). The last column contains the time needed to generate and evaluate one individual (i.e. 2^N test cases). The evaluation of a candidate network requires 1.3 ms for $N = 16$ and 40s for $N = 32$ (at 50MHz).

Table 3. Sorting networks evolved in FPGA

N length	VRC size (elements)	#Perfect solutions	Average num. of generations	Standard deviation	Evaluation time of one candidate
4	2x8	80	94	2.55556	512 ns
6	3x16	80	458	31.44444	1.28 us
8	4x16	80	2217	24.66667	5.12 us
10	5x32	80	6378	65.66667	20.48 us
12	6x32	76	8673	666.88889	81.92 us
14	7x32	75	11322	718.55556	327.67 us
16	8x32	66	19467	477.44444	1.31 ms
18	9x64	20	25306	3732.77778	5.24 ms
20	10x64	17	31344	150.00000	20.97 ms

Table 4. Large sorting networks evolved in FPGA

N length	VRC size (elements)	Number of generation	Evaluation time of one candidate	Total time of evolution
22	11x64	4044	83.89 ms	5.6 min
24	12x64	4804	335.54 ms	26.9 min
26	13x64	10027	1.342 s	3.7 h
28	14x64	13483	5.368 s	20.1 h

In order to evolve larger sorting networks we applied an adaptive mutation. With respect to N we mutated $4 - 12$ bits per chromosome. If no improvement in fitness value is observed in last 1000 generations, the number of mutated bits is increased by 2. If an improvement is observed, the mutation ratio is changed back to the previous value. Table 4 presents some of the evolved sorting networks up to $N = 28$. The evolution of a 28-input sorting network requires more than 20 hours (at 50 MHz). The design can easily work at 100 MHz as well.

6 Discussion

In fitness functions, all possible input combinations are evaluated, i.e. 2^N test vectors are evaluated for N-input sorting network. In [10] median networks (whose evaluation is of the same complexity as for sorting networks) were evolved up to $N = 25$ in software. However, component-optimal solutions were not obtained for larger N. It was reported that the fitness calculation (performed in software) is very time consuming for $N \geq 23$ and evolution requires days to find a solution. Here we demonstrated that a special architecture implemented in hardware could make the evolutionary design significantly faster. Alternatively, we could reduce the training set; however, we have never obtained a perfect solution with the reduced training set.

We have evolved relatively large combinational circuits (28 inputs, 28 outputs) from scratch in a relatively short time (about 20 hours) and in (relatively low-cost) commercial off-the-shelf hardware. On the other hand, we have used a lot of domain knowledge for solving this problem (the usage of compare&swap components, invariable interconnection of components etc. is typical only for this problem). We demonstrated what complex circuits can be evolved on commercially available FPGAs. The evaluation of a single candidate sorting network for $N = 28$ was compared against highly optimised SW implementation running in Xeon 3 GHz. Our FPGA evaluation running at 100 MHz is 40× faster then the software approach.

A strongly generic approach was utilized during VHDL design. All the implemented units are parameterized using various constants (such as the size of chromosome, the number of mutations etc.). Therefore, it is easy to modify the design and to obtain a totally different evolvable system in a very short time. The FPGA communicates with PC via special software allowing designer to prepare scripts describing experiments that have to be performed. Typically, designer specifies the VRC, EA and fitness function, perform synthesis, upload the evolvable system into FPGA and execute all experiments described in scripts.

7 Conclusions

A specialized architecture was developed and evaluated to evolve relatively large sorting networks in an ordinary FPGA. We evolved sorting networks up to $N = 28$. The evolution of the largest sorting networks requires 10 hours in FPGA running at 100 MHz. In next research, the number of components utilized in the evolved networks will be optimized. As target future application of this approach we consider adaptive routing in computer networks.

Acknowledgments

The research was performed with the financial support of FRVS 3042/2005/G1 project *Evolutionary design of sorting and median networks in FPGAs*. Lukas Sekanina was supported from the research project of the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based applications design methods*.

References

1. Devillard, N.: Fast Median Search: An ANSI C Implementation. 1998 http://ndevilla.free.fr/median/median/index.html
2. Hillis, W. D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42 (1990) 228–234
3. Imamura, K., Foster, J. A., Krings, A. W.: The Test Vector Problem and Limitations to Evolving Digital Circuits. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, IEEE CS Press, 2000, p. 75–79

4. Juillé, H.: Evolution of Non-Deterministic Incremental Algorithms as a New Approach for Search in State Spaces. In Proc. of 6th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1995, p. 351–358
5. Knuth, D. E.: The Art of Computer Programming: Sorting and Searching (2nd ed.), Addison Wesley, 1998
6. Koza, J. R., Bennett III., F. H., Andre, D., Keane, M. A.: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann, 1999
7. Liberouter project. www.liberouter.org
8. Porter, R.: Evolution on FPGAs for Feature Extraction. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2001, p. 229
9. Sekanina, L.: Evolvable components: From Theory to Hardware Implementations, Springer-Verlag, Natural Computing Series, 2003
10. Sekanina, L.: Evolutionary Design Space Exploration for Median Circuits. In: Applications of Evolutionary Computing, Coimbra, Portugalsko, LNCS 3005, Springer Verlag, 2004, p. 240-249
11. Sekanina, L., Friedl, S.: On Routine Implementation of Virtual Evolvable Devices Using COMBO6. In: Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, USA, IEEE Computer Society Press, 2004, p. 63-70
12. Shepherd, R., Foster, J.: Inherent Fault Tolerance in Evolved Sorting Networks. In Proc. of GECCO 2003, LNCS 2723, Springer Verlag, 2003, p. 456–457
13. Sloarch, C., Sharman, K.: The Design and Implementation of Custom Architectures for Evolvable Hardware Using Off-the-Shelf Progarmmable Devices. In: Proc. of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00, LNCS 1801, Springer-Verlag, Berlin, 2000, p. 197–207
14. Smith, J. I.: Implementing Median Filters in XC4000E FPGAs. Xcell 23, Xilinx, 1996 http://www.xilinx.com/xcell/xl23/xl23_16.pdf
15. Tufte, G., Haddow, P.: Evolving an Adaptive Digital Filter. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware. IEEE Computer Society, 2000, p. 143–150
16. Zhang, Y., Smith, S., Tyrrell, A.: Intrinsic Evolvable Hardware in Digital Filter Design. In: Applications of Evolutionary Computing, Berlin, DE, Springer, LNCS 3005, 2004, p. 389-398