Reducing the Number of Transistors in Digital Circuits Using Gate-Level Evolutionary Design

Zbysek Gajda
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
gajda@fit.vutbr.cz

Lukas Sekanina
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
sekanina@fit.vutbr.cz

ABSTRACT

This paper shows that the evolutionary design of digital circuits which is conducted at the gate level is able to produce human-competitive circuits at the transistor level. In addition to standard gates, we utilize unconventional gates (such as the NAND/NOR gate and NOR/NAND gate) that consist of a few transistors but exhibit non-trivial 3-input logic functions. Novel implementations of adders and majority circuits evolved using these gates contain fewer transistors than the smallest existing implementations of these circuits. Moreover, it was shown that the use of these gates significantly improves the success rate of the search process.

Categories and Subject Descriptors

B.6.1 [**Hardware**]: Design Styles—*Combinational logic*; B.6.3 [**Hardware**]: Logic Design—*Design Aids*; I.2.m [**Artificial Intelligence**]: Miscellaneous

General Terms

Algorithms

Keywords

digital circuits, evolvable hardware, evolutionary design

1. INTRODUCTION

In the context of the evolutionary circuit design, the *scalability problem* is often mentioned in literature [15, 18, 4]. The main implication of the scalability problem is that only relatively simple circuits have been evolved so far. Basically, there are two reasons for this problem: Firstly, complex solutions require long chromosomes to be represented but these long chromosomes imply large search spaces that are usually difficult to search. Secondly, in order to evaluate a complex candidate circuit, a relatively complex procedure has to be undertaken – which is time consuming and makes the evolutionary search very slow. Stoica et al. noted [12]: "From the evolvable hardware perspective, it is interesting to have programmable granularity, allowing the sampling of novel architectures together with the possibility of implementing standard ones. The optimal choice of elementary block type and granularity is task dependent." Therefore, its designer's challenge to define the representational bias in order to obtain a suitable search space within the entire space of possible solutions.

While the evolutionary design conducted at a certain level is able to provide optimized solutions for the particular level (e.g. the transistor level), it is very difficult to evolve at the particular level more complicated circuits that are typically designed at a higher level (e.g. the gate level). On the other hand, when the evolution is conducted at the higher level then a particular complex solution might be found; however, it is not usually optimal from the perspective of lower levels that have to be taken into account when the evolved circuit has to be fabricated.

This fact is also visible in the standard design process of digital circuits. Complex digital circuits are designed at the RT-level (register transfer level) or at the gate level. Various optimization techniques are utilized to obtain an optimized gate-level circuit. This solution is then implemented using well-known transistor-level implementations of standard logic gates which are available in the so-called *cell library*. However, the final transistor-level solution might not be optimal at all.

For example, the logic expression $Y = (AB + CD)'$ (a corresponding circuit is shown in Fig. 1a) seems to be optimized at the gate level. It requires two AND gates, a single OR gate and inverter, which costs 20 transistors (see Fig. 1c). However, Y can be implemented using 8 transistors only when a special AND-OR-Invert circuit is employed (see Fig. 1b). The reason is that the transistor level allows implementing this particular circuit much easier than a standard gate-level optimization suggests. As circuit designers know this fact, they do map RT-level designs to the elementary components of the cell library that are optimized for a particular fabrication process.

Unfortunately, in the field of evolutionary design of digital circuits, designers do usually optimize the number of standard gates (and/or delay), ignoring thus the real circuit cost on a chip [9, 15, 2, 20] (perhaps with an exception [1]). Although the evolutionary circuit design has proven that is able to generate the gate-level circuits that lie beyond the scope of human designs [9, 7, 5], these benefits have not

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

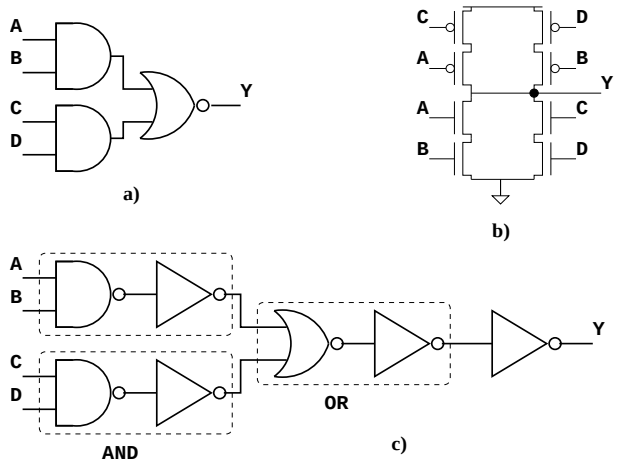

Figure 1: And-Or-Invert Circuit: a) at the gate level, b) CMOS implementation, c) a naive gate level implementation

been demonstrated so far for nontrivial digital circuits considered at the transistor level. Note that some implementations of simple gates were evolved at the transistor level [5, 8, 19]; however, as mentioned above, the evolutionary design of more complicated digital circuits directly at the transistor level is currently outside the capabilities of commonly available computers. In particular, Langeheine [8] argues that the evolutionary design of the XOR gate is difficult.

In this paper we utilize the evolutionary algorithm to evolve gate-level combinational circuits with the aim to minimize the number of transistors in target designs. The goal of this paper is to show that by using special elementary circuit components (unconventional gates) that are optimized for the transistor level we can reduce the number of transistors in some digital circuits. The adders and majority circuits are used as benchmark circuits.

2. GATE LEVEL VS. TRANSISTOR LEVEL DESIGNS

2.1 The 1-bit Full Adder

Consider a 1-bit full adder. This circuit has two operands, A and B, and an input carry, C_{in}. It generates the sum

$$S = A \oplus B \oplus C_{in} \tag{1}$$

and the output carry

$$C_{out} = AB + BC + AC \tag{2}$$

These equations were derived from the truth table of this circuit and optimized using standard operations of Boolean algebra [16]. As the circuit contains a 3-input XOR gate, three 2-input AND gates and a 3-input OR gate, it costs 38 transistors. In order to calculate the number of transistors, we have to look at Table 1 which gives transistor costs of standard gates.

However, a standard static CMOS VLSI implementation of the 1-bit full adder costs only 24 transistors [17] (see Fig. 2). The number of transistors can even be reduced to 22 when so-called transmission gates are utilized [21]. Zimmermann and Gupta compared different implementations

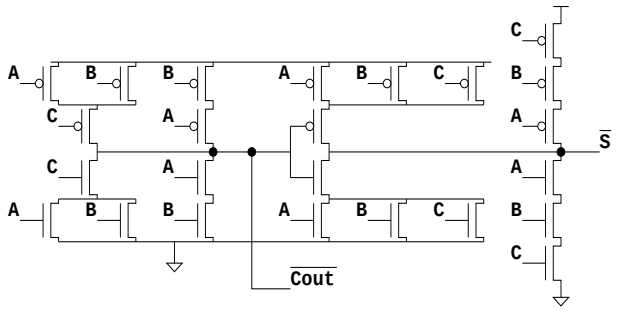

Figure 2: A standard 24-transistor implementation of a static 1-bit full adder

of full adders in terms of area, delay and power consumption [22]. The implementation cost of adders is strongly connected with the cost of the XOR gate. While a standard static CMOS 2-input XOR gate is implemented using 10 transistors, only 8 transistors are sufficient when transmission gates can be utilized. Cheng and Hsieh compared various implementations of 3-input XOR gate [3]. Their paper also shows that the 3-input gate can be implemented using 12 transistors.

Miller et al evolved a very unconventional implementation of the 1-bit full adder [9]. Although Miller et al have not optimized the number of transistors, Figure 6a shows that evolved circuit can be implemented using 22 transistors only as it contains two 2-input XOR gates and a single multiplexer.

Table 1: Logic gates and their implementation cost

Gate	Inputs	Symbol	Transistors
AND	2	AND2	6
	3	AND3	8
OR	2	OR2	6
	3	OR3	8
NOT	1	NOT	2
XOR	2	XOR2	8/10
	3	XOR3	12/16
NAND	2	NAND2	4
	3	NAND3	6
NOR	2	NOR2	4
	3	NOR3	6
MX	2+1	MX	6
NAND/NOR	2+1	NAND/NOR	10
NOR/NAND	2+1	NOR/NAND	8

2.2 Unconventional Gates: NAND/NOR and NOR/NAND

This section presents two unconventional three-input gates that are not usually considered by circuit designers. It will be shown in next sections that the evolutionary design approach is able to effectively utilize them as building blocks of more complex circuits.

The NAND/NOR gate as well as the NOR/NAND gate has three inputs A, B and Sel and operates according to Table 2. As Figure 3 shows, these gates perform multiplexing the NAND logic function and the NOR logic function according to Sel. Table 1 suggests that their implementa-

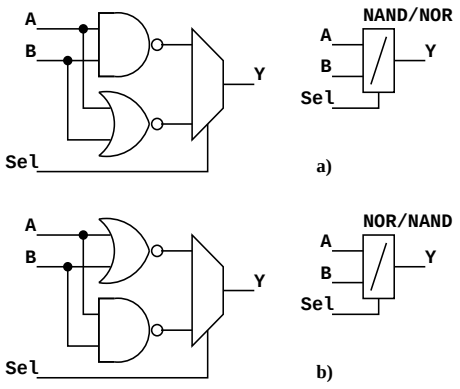

Figure 3: Gate-level implementation a) NAND/NOR gate, b) NOR/NAND gate

tion should cost 14 transistors (6 transistors for the multiplexer, 4 transistors for the NAND and 4 transistors for the NOR). However, as the NAND/NOR gate is identical with an inverted majority function, it can be implemented using 10 transistors only (see Fig. 4). An implementation of the NOR/NAND gate costs 8 transistors.

Table 2: Truth tables of NAND/NOR and NOR/NAND gates

Sel	A	B	NAND/NOR	NOR/NAND
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0

3. EVOLUTIONARY ALGORITHM

Various approaches have been proposed to evolve digital circuits [9, 2, 20, 14, 13]. In this work, we will use the evolutionary algorithm (EA) inspired by Cartesian Genetic Programming [9, 10]. A candidate digital circuit is represented using a one-dimensional array of programmable nodes. Each programmable node has three inputs, a single output and can be programmed to implement one of functions specified in function set Γ. The role of EA is to find the interconnection of nodes and functions performed by the nodes for a given specification expressed by means of a truth table. Circuits are encoded as arrays of integers of the size $4n + n_o$, where n is the number of nodes and n_o is the number of circuit outputs. Figure 5 shows a circuit consisting of three inputs, two outputs and five programmable nodes. Note that only nodes 3, 5 and 7 are utilized in this example (i.e. in the phenotype). As only combinational circuits will be evolved, no feedback links are allowed in candidate circuits. Hence a node input can be connected either to an output of a node with smaller index or to a primary circuit input. EA uses a single genetic operator – mutation – which modifies m_1 (typically, $m_1 = 3$) integers of the chromosome. Either a

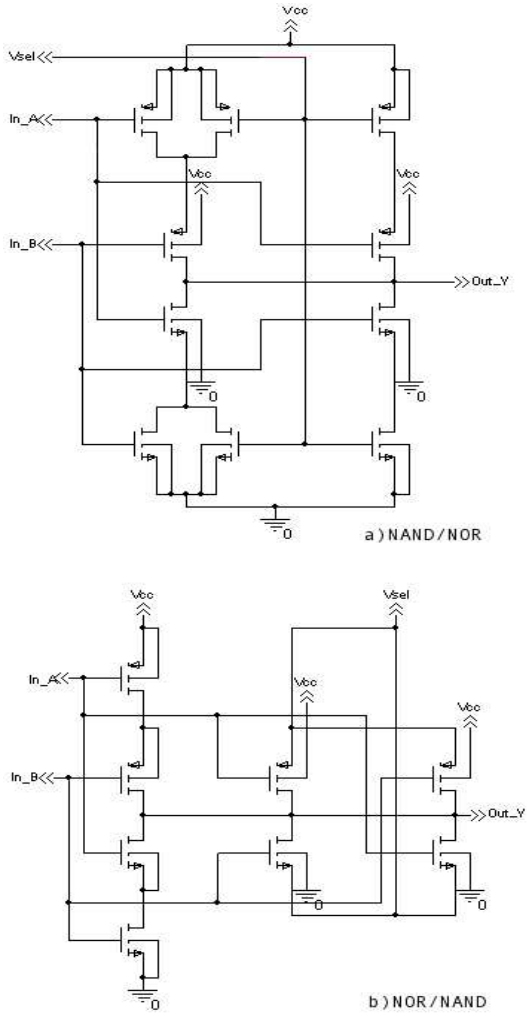

Figure 4: CMOS implementation a) NAND/NOR gate, b) NOR/NAND gate

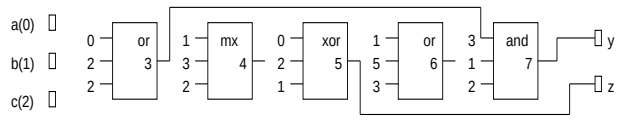

Figure 5: Example of encoding of a 3-input/2-output circuit. Chromosome: 0220 1323 0212 1530 3121 75. Logic functions are encoded as 0 (OR), 1 (AND), 2 (XOR), 3 (MX).

node or an output connection is modified. The EA operates with the population of λ individuals (typically, $\lambda = 15$). The initial population is randomly generated. Every new population consists of a parent (the fittest individual from the previous population) and its mutants. In case that two or more individuals have received the same fitness score in the previous generation, the individual which did not serve as the parent in the previous population will be selected as a new parent. This strategy was proven to be very useful [9, 15].

In case of the combinational circuit evolution, the fitness function is constructed to minimize the Hamming distance between the output vectors of a candidate circuit and the required output vectors. All possible input vectors are applied to obtain the set of output vectors. In addition to maximizing functionality, we minimize the number of transistors, the number of gates and delay (at the gate level). This is achieved by using a simple multicriteria fitness function whose value has to be minimized here:

$$fitness = 1 + a.g_3 + b.g_2 + c.g_1 + d.g_0, \quad (3)$$

where a denotes the number of wrong bits computed by the candidate circuit (which is the highest priority), b is the number of transistors, c is the number of gates and d denotes delay of the circuit (which is the lowest priority). Constants g_0, g_1, g_2 and g_3 are defined as follows:

$$g_0 = 1, \quad (4)$$
$$g_1 = (d_{max} + 1).g_0, \quad (5)$$
$$g_2 = (c_{max} + 1).g_1, \quad (6)$$
$$g_3 = (b_{max} + 1).g_2, \quad (7)$$

where b_{max}, c_{max} and d_{max} are maximum possible values for b, c and d. Typically, $b_{max} = 16n$, $c_{max} = n$ and $d_{max} = n$. The number of transistors is calculated for the nodes used in the phenotype according to Table 1.

4. EXPERIMENTAL RESULTS

This section reports the experimental results obtained for four target circuits: 1-bit full adder, 2-bit adder, 5-input majority circuit and 7-input majority circuit.

4.1 The 1-bit Full Adder

In order to evolve a 1-bit full adder, we utilized the following setup of EA: $\lambda = 15$, $n = 20$, $m_1 = 3$ and 1,000,000 generations were produced. Note that we do modify the mutation rate after achieving a perfect functionality, i.e. before the number of transistors is optimized. The second mutation parameter is denoted as m_2. We modified 7 integers on average of the chromosome, i.e. $m_2 = 7$. We tested different combinations of gates in Γ and different transistor costs for those gates. We run 50 independent experiments

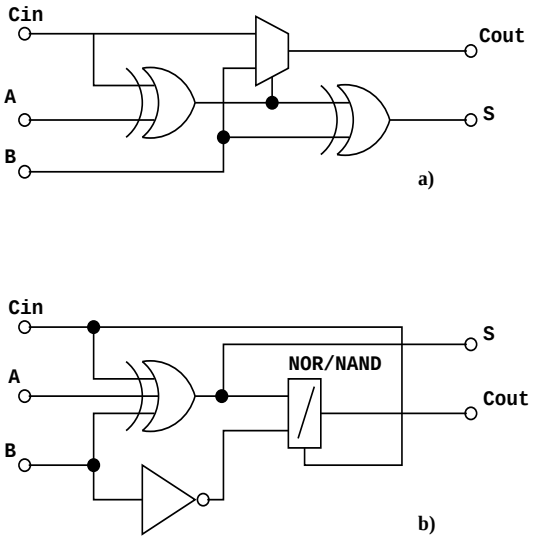

Figure 6: Evolved 1-bit full adders

for each setup. Table 3 lists the gates included into Γ, the average number of generations (AvrG) needed to find a fully functional solution (not necessarily optimized for transistor count) and examples of best-evolved implementations. We can observe that the 22-transistor implementation was found for all combinations of parameters. Figure 6 shows examples of the best implementations.

4.2 The 2-bit Adder

The 2-bit adder calculates the 2-bit sum and output carry for two 2-bit operands and an input carry, i.e. the circuit has 5 inputs and 3 outputs. In order to investigate which set of gates is suitable for this problem, we compared ten different sets Γ (denoted as #1–#10 in the following tables). Corresponding transistor counts per gate are taken according to Table 1. In this experiment and in the following experiments, XOR2 costs 10 transistors, XOR3 costs 12 transistors, NAND/NOR costs 10 transistors and NOR/NAND costs 8 transistors. We utilized the following setup of EA: $\lambda = 15$, $n = 60$, $m_1 = 3$, $m_2 = 7$ and 1,000,000 generations were produced. We run 100 independent experiments for each combination of gates in Γ.

The first part of Table 4 provides the average number of gates used in the best circuits of each run. Thus, we can observe those gates that are useful for this particular problem. An empty space means that the gate was not used in Γ. The second part of Table 4 gives the average values for some parameters calculated from the best circuits of the 100 independent runs: the number of transistors, the number of gates, delay (measured at the gate level), fitness value and the number of generations. "Succ. Run" denotes the number of runs (out of 100 runs) in which a correcly working circuit was evolved.

Figure 7 shows the best implementation of the 2-bit adder that we evolved. By connecting two conventional 1-bit full adders or two Miller's 1-bit full adders [9] into the carry-ripple adder we can obtain a solution with 44 transistors. Evolved solution which utilizes two NOR/NAND gates, two 3-input XOR gates and an inverter costs 42 transistors.

Table 3: Resulting implementations of the 1-bit full adder for different sets of utilized gates $\Gamma_1 \ldots \Gamma_6$. The number of transistors is given for each gate.

Gates	AvrG	XOR2	XOR3	MX	NOT	NAND/NOR	NOR/NAND	Best implementations
Γ_1	386.9	10	12	6	2	-	-	3.MX+2.NOT = 22 tr. (C1)
Γ_2	334.9	10	12	6	2	10	8	NOR/NAND+XOR3+NOT = 22 tr. (C2)
Γ_3	367.4	10	16	6	2	-	-	C1
Γ_4	259.3	10	16	6	2	10	8	C1 and C2
Γ_5	329.2	8	12	6	2	-	-	2.XOR2+MX = 22 tr.
Γ_6	394.9	8	12	6	2	10	8	C2

Table 4: The 2-bit adder circuit: Average values of some parameters calculated from 100 independent runs for 10 different sets of gates

Γ	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
AND2	3.27	0.18	0.03	11	7.93					
OR2	2.13	0.13	0.1		7.6					-
XOR2	3.95	1.55	0.44							
NOT	0.69	0.99	1.22	17	2.59			2.9	2.52	-
NAND2	0.86	0.22	0.11			17.71		15.69		
NOR2	0.88	0.2	0.07				17.26		15.95	
AND3		0	0	4	0.12					
OR3		0.01	0		0.21					-
XOR3		0.93	1.53							
NAND3		0	0			1.13		1.45		
NOR3		0.01	0.01				1.44		1.25	
MX		3.55	0.88							
NAND/NOR			0.55							
NOR/NAND			1.51							
Transistors	80.24	53.62	49.62	132	101	77.68	77.82	77.24	76.36	
Gates	11.78	7.77	6.45	32	18.45	18.84	18.74	20.03	19.73	
Delay	5.56	4.05	3.56	10	8.03	7.9	7.88	7.45	7.57	
Fitness	2,535,619	2,882,840	2,867,875	22,408,827	2,166,756	5,737,087	5,452,684	5,879,844	4,878,742	
Generation	38,686.8	32,793.2	21,386.8	997,862	156,150	863,785	855,925	848,595	755,830	
Succ. Rate	100	100	100	1	100	31	34	29	44	0

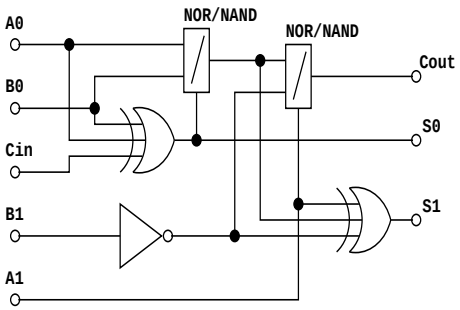

Figure 7: Evolved 2-bit adder

4.3 Majority Circuits

A majority circuit returns logic 1 only if more logic 1s than logic 0s are given at the circuit input. If only two-input AND and two-input OR gates can be utilized, then according to a sorting network-based implementation [6], the 5-input majority circuit consists of 10 such gates (i.e. 60 transistors) and the 7-input majority circuit consists of 20 such gates.

In order to evolve the 5-input majority circuit and 7-input majority circuit we utilized the same setup as in the previous section. Table 5 and Table 7 summarize average values of some circuit parameters calculated from 100 independent runs for 10 different sets of gates. Both tables illustrate that the use of unconventional gates significantly improves the quality of evolved solutions (see the #3 column).

Table 6 shows basic parameters of selected 5-input majority circuits that we evolved. It can be seen in Figure 8 that the best circuit contains only 32 transistors and utilizes three NOR/NAND gates, the NOR gate and the NAND gate. Without the use of the unconventional gates, the EA is able to find an implementation which costs 40 transistors. Note that a conventional implementation of this circuit con-

Table 5: The 5-input majority circuit: Average values of some parameters calculated from 100 independent runs for 10 different sets of gates

Γ	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
AND2	4.59	0.55	0.06	8	3.28					
OR2	4.23	0.75	0.12		2.99					8.67
XOR2	0.38	0.32	0.01							
NOT	0.11	0.42	0.1	12	0			2.97	2.76	11.33
NAND2	1.17	1.47	0.27			11.36		9.23		
NOR2	1.03	0.99	0.19				11.8		9.45	
AND3		0.3	0.09	2	1.3					
OR3		0.27	0.05		1.45					2.67
XOR3		0.33	0.01							
NAND3		0.15	0			2.25		2		
NOR3		0.08	0.03				2.01		2.09	
MX		3.25	0.36							
NAND/NOR			0.95							
NOR/NAND			2.48							
Transistors	65.74	51.08	36.14	88	59.62	58.94	59.26	54.86	55.86	96
Gates	11.51	8.88	4.72	22	9.02	13.61	13.81	14.2	14.31	22.67
Delay	5.49	4.51	3.18	7	4.61	5.12	5.23	5.59	5.49	7.33
Fitness	2,481,648	2,873,457	2,817,610	5,445,075	2,012,203	1,563,433	1,564,636	1,548,288	1,552,015	5,388,845
Generation	40,677.9	12,130.5	9,348.8	996,910	7,461.89	37,914.7	35,905.9	37,436.9	49,098.4	990,743
Succ. Rate	100	100	100	1	100	100	100	100	100	3

tains 60 transistors. An optimized conventional solution was proposed which contains 38 transistors [11].

Table 6: Some properties of the best evolved 5-input majority circuits

Circuit	1	2	3	4	5	6
# conventional gates	8	2	7	0	8	7
# NAND/NOR	–	0	–	1	–	–
# NOR/NAND	–	3	–	3	–	–
# transistors	40	32	42	34	46	48
# gates	8	5	7	4	8	7
delay	4	3	4	3	3	3

Table 8 shows basic parameters of selected 7-input majority circuits that we evolved. It can be seen that the best circuit contains only 60 transistors and utilizes two NOR/NAND gates, two NAND/NOR gates and three 3-input XOR gates. The best circuit evolved using standard gates contains 86 transistors.

Table 8: Some properties of the best evolved 7-input majority circuits.

Circuit	1	2	3
# conventional gates	15	2	13
# NAND/NOR	–	2	–
# NOR/NAND	–	2	–
# transistors	86	60	96
# gates	15	6	13
delay	9	3	6

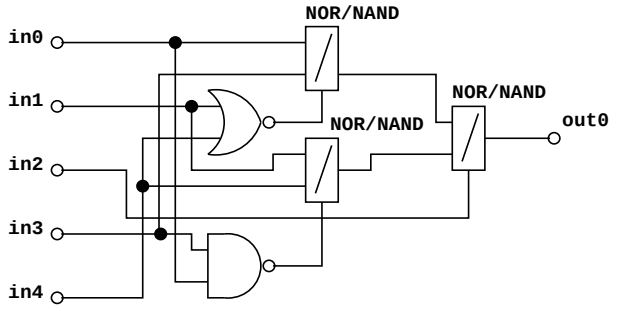

Figure 8: Evolved 5-input majority circuit

5. DISCUSSION

Our task was to design non-trivial digital circuits (more complicated than a single gate) and, simultaneously, to create such implementations of those circuits that are optimized for the target platform, i.e. rather for the transistor level than for the gate level. As we are not able to evolve these circuits directly at the transistor level[1], evolutionary design of this class of circuits was performed at the gate level. However, the implementation cost of candidate circuits was evaluated at the transistor level.

Except the standard gates we utilized two unconventional gates, well-suited as basic components for the circuits we wanted to evolve. In most cases, evolved solutions contain fewer transistors than well-optimized transistor-level conventional as well as existing evolved implementations. Experimental results show that the use of unconventional

[1]As far as we know, no similar results have been reported in available literature. Probably because available computing resources are not sufficient.

Table 7: The 7-input majority circuit: Average values of some parameters calculated from 100 independent runs for 10 different sets of gates

Γ	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
AND2	9.67	1.54	0.59	-	-					
OR2	8.67	1.87	1.21		-					-
XOR2	1.08	1.29	0.16							
NOT	0.14	0.79	0.65	-	-			5	6.12	-
NAND2	0.83	1.68	0.61			17		19.6		
NOR2	1.72	1.18	0.63				20.14		15.75	
AND3		0.48	0.28	-	-					-
OR3		0.63	0.32		-					-
XOR3		1.33	0.21							
NAND3		0.6	0.13			7.5		4.2		
NOR3		0.56	0.1				5.14		5.12	
MX		7.73	1.84							
NAND/OR			3.21							
NOR/NAND			3.36							
Transistors	131.44	124.6	97.49			113	111.43	113.6	106	
Gates	22.11	19.68	13.32			24.5	25.29	28.8	27	
Delay	8.44	7.55	5.68			6.5	7.86	9	8.62	
Fitness	5,399,959	3,392,049	3,156,677			9,046,069	8,728,553	7,006,954	7,309,033	
Generation	849,817	370,296	135,659			973,505	971,021	979,242	979,381	
Succ. Rate	36	94	98	0	0	4	7	5	8	0

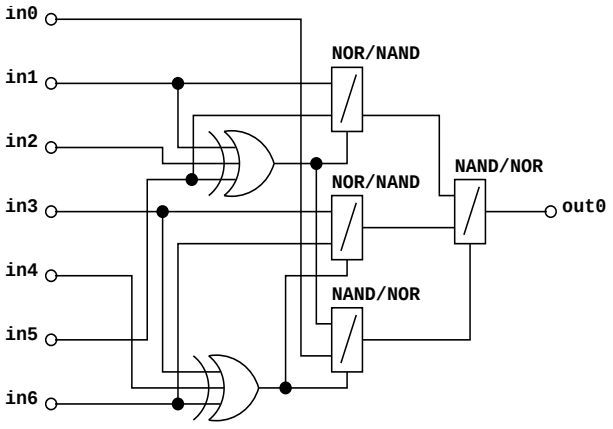

Figure 9: Evolved 7-input majority circuit

gates significantly helps the evolution to find good solutions (see the averages in the #3 column of Tables 4, 5 and 7). Table 9 summarizes the results. Therefore, identifying the suitable transistor-level components (i.e. the gates such as NAND/NOR, NOR/NAND, AND-OR-Invert etc.) and using them as building components at the gate level represents a promising method for designing non-trivial digital circuits optimized at the transistor level. As Table 3 shows, EA is really able to optimize the number of transistors in target circuits when different implementation costs are assigned to gates.

Another advantage of the proposed method is that when EA operates with gates and those gates are correctly implemented using transistors then the final solution would exhibit a desired electrical behavior. It often happens during evolutionary digital circuit design conducted at the transistor level that some transistors are not used as switches (e.g. they operate in the active region) and thus resulting circuits exhibit undesired properties such as a high power consumption.

Table 9: The number of transistors in the best implementations of test circuits. UG denotes "unconventional gates included"

Circuit/Method	Conventional	EA	EA(+UG)
1-bit adder	22	22	22
2-bit adder	44	44	42
5-input majority	38	40	32
7-input majority	-	86	60

On the other hand, we have not considered some important aspects in this study. Because we optimized the delay at the gate level, the delay of evolved solutions is not necessarily optimized for the transistor level. We did not deal with the power consumption of evolved circuits. We did not consider the placement and routing aspects of evolved circuits on a chip. These issues should be investigated as a part of future work.

In comparison with the Koza's highly computationally-demanding genetic programming method [7], our approach allowed us to design target circuits in a relatively short time. A single run of EA which utilizes 60 programmable nodes and produces 1,000,000 generations requires 30 sec. for the 1-bit full adder, 154 sec. for the 2-bit adder (or 5-input majority) and 198 sec. for the 7-input majority circuit on average when computed at a standard PC equipped with Athlon64 X2 4800+ processor.

6. CONCLUSIONS

In this paper we clearly demonstrated that the evolutionary design of digital circuits which is conducted at the gate level is able to produce human-competitive circuits at the transistor level. Novel implementations of adders and majority circuits that utilize unconventional NAND/NOR and NOR/NAND gates were proposed. Future work will be devoted to searching for those "unconventional" components which were overlooked in the past but which could serve as area-efficient building blocks for the evolutionary design conducted at the gate level.

Acknowledgments

This work was partially supported by the Grant Agency of the Czech Republic under contract No. 102/06/0599 *Methods of polymorphic digital circuit design* and the Research Plan No. MSM 0021630528 - *Security-Oriented Research in Information Technology*.

7. REFERENCES

[1] T. Aoki, N. Homma, and T. Higuchi. Evolutionary Synthesis of Arithmetic Circuit Structures. *Artificial Intelligence Review*, 20(3–4):199–232, 2003.

[2] D. Chen, T. Aoki, N. Homma, T. Terasaki, and T. Higuchi. Graph-Based Evolutionary Design of Arithmetic Circuits. *IEEE Trans. on Evolutionary Computing*, 6(1):86–100, 2002.

[3] K.-H. Cheng and V.-C. Hsieh. High Efficient 3-input XOR for Low-Voltage Low-Power High Speed Applications. In *Proc. of The First IEEE Asic Pacific Conference on ASICs*, pages 166–169, Seoul, Korea, 1999. IEEE Computer Society.

[4] T. Gordon and P. Bentley. Evolving hardware. In A. Zomaya, editor, *Handbook of Nature Inspired and Innovative Computing*, pages 387–432. Springer Verlag, 2006.

[5] A. Keane, M. J. Streeter, and W. Mydlowec. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Springer, New York, 2004.

[6] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.

[7] J. R. Koza, F. H. B. III., D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[8] J. Langeheine. *Intrinsic Hardware Evolution on the Transistor Level*. PhD thesis, Rupertus Carola University of Heidelberg, 2005.

[9] J. Miller, D. Job, and V. Vassilev. Principles in the Evolutionary Design of Digital Circuits – Part I. *Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[10] J. Miller and S. Smith. Redundancy and computational efficiency in Cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[11] J. M. Quintana, M. J. Avedillo, R. Jiménez, and E. Rodríguez-Villegas. Practical low-cost cpl implementations of threshold logic functions. In *Proc. of the 11th ACM Great Lakes Symposium on VLSI 2001*, pages 139–144, West Lafayette, Indiana, USA, 2001. ACM.

[12] A. Stoica, D. Keymeulen, A. Thakoor, T. Daud, G. Klimech, Y. Jin, R. Tawel, and V. Duong. Evolution of Analog Circuits on Field Programmable Transistor Arrays. In *Proc. of the 2000 NASA/DoD Conference on Evolvable Hardware*, pages 99–108, Palo Alta, CA, 2002. IEEE Computer Society.

[13] E. Stomeo, T. Kalganova, and C. Lambert. Generalized Disjunction Decomposition for Evolvable Hardware. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 36(5):1024–1043, 2006.

[14] A. Thompson. Silicon Evolution. In *Proc. of Genetic Programming GP'96*, pages 444–452. MIT Press, 1996.

[15] V. Vassilev and J. F. Miller. Scalability problems of digital circuit evolution. In *Proc. of the 2000 NASA/DoD Conference on Evolvable Hardware*, pages 55–64, Palo Alta, CA, 2000. IEEE Computer Society.

[16] J. Wakerly. *Digital Design: Principles and Practices*. Prentice-Hall, 2000.

[17] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective (3rd edition)*. Addison Wesley, 2004.

[18] X. Yao and T. Higuchi. Promises and Challenges of Evolvable Hardware. *IEEE Transactions on Systems, Man, and Cybernetics*, 29(1):87–97, 1999.

[19] R. Zebulum, M. Pacheco, and M. Vellasco. *Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. The CRC Press International Series on Computational Intelligence, 2002.

[20] S. Zhao and L. Jiao. Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm. *Genetic Programming and Evolvable Machines*, 7(3):195–210, 2006.

[21] N. Zhuang and W. Haomin. A new design of the CMOS full adder. *IEEE journal of solid-state circuits*, 7(5):840–844, 1992.

[22] R. Zimmermann and R. Gupta. Low-power logic styles: Cmos vs cpl. In *Proceedings of the 22nd European Solid-State Circuits Conference*, pages 112–115, Neuchatel, Switzerland, 1996.