AN EVOLUTIONARY APPROACH TO CHIP DESIGN: AN EMPIRICAL EVALUATION

Gregor Papa

"Jožef Stefan" Institute, Ljubljana, Slovenia

Key words: chip design, area and time optimization, low-power circuits, evolutionary algorithms

Abstract: This paper presents a new method with an evolutionary approach to some parts of integrated-circuit (IC) design. This study, however, is focused on application-specific integrated circuits (ASICs), which need an even more sophisticated design (in terms of size, speed and low-power) because of their specific uses.

Optimally scheduled operations are not necessarily optimally allocated to units. To enable optimal allocation we need to consider some allocation criteria while the scheduling is being done. Therefore, algorithms with concurrent scheduling and allocation produce the best results. It is obvious that we have to deal with a trade-off between the quality of the solution and its design time. The main part of the paper is a presentation of an improved method of the evolutionary search for the optimal design of ICs. The evolutionary approach considers scheduling and allocation constraints and ensures a globally optimal solution in a reasonable time.

The evaluation of our method shows that the evolutionary method is able to find a solution that is more appropriate in terms of all the considered and important objectives than is the case when working with classical deterministic methods.

Evolucijski pristop pri načrtovanju čipov: empirično ovrednotenje

Ključne besede: načrtovanje čipov, energijsko varčna vezja, optimizacija velikosti in časa, evolucijski algoritmi

Izvleček: V delu je predstavljena metoda sočasnega razvrščanja operacij in dodeljevanja enot, ki temelji na evolucijskem načinu in je uporabna v postopku načrtovanja digitalnih integriranih vezij. Delo obravnava predvsem načrtovanje namenskih vezij, ki potrebujejo zaradi svoje specifičnosti toliko bolj dodelano strukturo, tako s stališča velikosti, kakor tudi s stališča hitrosti delovanja in majhne porabe energije.

Optimalno razvrščenih operacij v splošnem ni mogoče tudi optimalno dodeliti posameznim enotam. Da bi lahko operacije dodelili optimalno, je treba pravila za dodeljevanje upoštevati že med razvrščanjem. Problema smo se lotili z evolucijsko tehniko, ki je pogosto uporabljena v metodah za iskanje rešitev na najrazličnejših področjih. Razvit je bil evolucijski algoritem, ki upošteva razvrščevalne in dodeljevalne zahteve, omogoča kratek načrtovalni čas ter globalno optimalne rešitve. Algoritem je bil tudi računalniško realiziran ter uporabljen pri množici preizkusnih vezij. Vezja so bila izbrana glede na pogostost pojavljanja v sorodni literaturi, in sicer vezja različnih velikosti z različnim številom tipov operacij.

Obravnava rezultatov je pokazala, da je opisani evolucijski algoritem, v primerjavi s klasičnimi determinističnimi metodami, sposoben poiskati rešitev, ki je v splošnem ugodnejša s stališča vseh obravnavanih in pomembnih parametrov.

1 Introduction

Whenever a new integrated circuit (IC) is designed the problem of selecting the best register-transfer level (RTL) specification has to be faced. And as circuits get bigger, so too does the problem: more combinations have to be examined before the optimal combination is found. Therefore, automatic circuit optimization is required, as this speeds up the whole design process and eliminates some of the errors.

High-level synthesis /2/, /4/ is an automatic design process that transforms the initial behavioral description into the final specification of the RTL. The process consists of the following tasks: compilation, transformation, scheduling, allocation and binding. Of these, the operation scheduling and the resource allocation are the most important subtasks of the high-level synthesis because they are at the core of the design and crucially influence both the design and the final layout. Due to the interdependence of these two tasks, the solution of one task depends on an estimation of the solution of the other task, which is not solved yet. The scheduling of the operation into different control steps therefore affects the allocation of operations to different units. The interaction of these two tasks presents formidable obstacles to the goal of optimization /1/. There are, however, some approaches to concurrent solving, but their solutions, to some extent, are less than optimal.

The evolutionary technique is used in various search methods for a range of different optimization areas /8/. Its undetermined approach, i.e., a probabilistic approach, reduces its popularity, but at the same time gives it an advantage when it comes to multicriteria problems and problems with more local optima. The genetic algorithm (GA), as a frequent implementation of evolutionary techniques, is an optimization method based on the mechanism of evolution and natural genetics.

2 Definitions

An IC described with a hardware description language, e.g. VHDL, can be presented as a control/data-flow graph (CDFG) /4/, and in our case it is enough to consider only the data-flow part, i.e., the data-flow graph (DFG). Each node *i* represents the operation to be executed, and the type of node determines the type of operation. The edges *e* represent dependencies between operations. An edge e_{ij}, between nodes *i* and *j*, represents the data produced by the node *i* and used by the node *j*. All edges are unidirectional.

In the final design there is a group of resources that define the implementation. Here we have different functional units (FUs) FU_i ($i=1..N$), where *N* is the number of different types of functional units. There are also storage (registers *Reg*) and connection (buses *Bus*) units, while *T* represents the execution time of the DFG. Functional units (adders, multipliers, ...) perform transformations on data values, connection units transport values from one unit to another, while storage units preserve those values over time.

The parameters are calculated as follows:

- the number of FU_i is the highest number of the *i*-th functional unit needed in a separate control step;
- the number of registers *Reg* is the highest number of variables needed in a separate control step. We consider variables that are needed by the functional unit as input data, variables that are returned as output data and variables that are not used at the moment but will be used in some of the later control steps or must be available until the end of the execution of all operations. Of course, more functional units can use the same data from the same register in a control step:
- the number of buses *Bus* is the highest number of data transmissions (into or from the functional units) in a separate moment;
- the execution time *T* is the time needed to execute all the operations of the schedule;
- the weights *w* are the weights of the IC parameters to be considered in the IC quality evaluation cost function. They depend on the ratio of different units' (functional, storage) sizes.

3 Concurrency

Optimally scheduled operations are not necessarily optimally allocated to units. To be also optimally allocated some allocation constraints should be considered during the scheduling. Therefore, algorithms that perform concurrent scheduling and allocation are better in terms of the optimality of the solutions. These algorithms are, however, very time consuming. Therefore, we have to deal with a trade-off between the quality of the final solution and the length of the design-time for it. There are some approaches to concurrent solving, but their solutions, to some extent, are less than optimal.

When tasks are performed separately (Figure 1a) the solution is not necessarily optimal; it is better to use an approach with iterative repetition of the scheduling and allocation (Figure 1b). Here, though, the problem of the next operation/unit to be changed appears, since the order of changes can influence the final solution. It is a similar situation with the approach that involves partitioning of the operations into small groups, within which there is an iterative repetition of the scheduling and the allocation (Figure 1c). Since there are fewer operations in the group there is no problem with the order of changes, but there is a problem with the appropriate partitioning of the operations. Obviously, the best approach is the one with purely concurrent scheduling and allocation (Figure 1d), where the iterative-refinement order does not influence the quality of the solution /7/. Concurrency is achieved through the use of algorithms that do not depend on the order of transformations. Therefore, there is no influence of the changed start time on the allocated unit, nor is there any influence of the allocated unit on the start time. When all the transformations are made, then the appropriateness of the changes is checked.

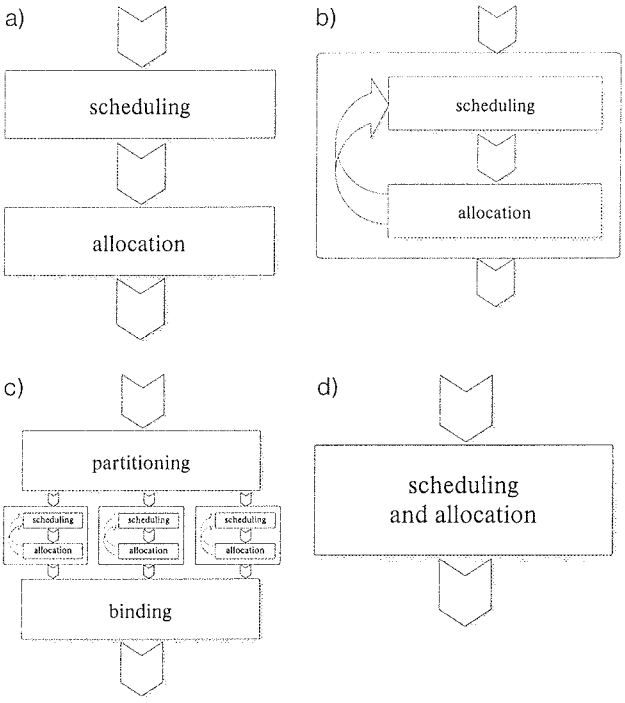

Figure 1: Scheduling and allocation concurrency

4 The ECSA algorithm

The facts presented in the introduction paragraphs and the promising results of different evaluations /8/, /9/ led us to the ECSA (Evolutionary Concurrent Scheduling and Allocation) design approach /7/. This approach considers scheduling and allocation constraints, allows a short design time and can find globally optimal solutions. The input description of the circuit is transformed into two basic (initial) schedules, obtained with the ASAP and ALAP algo-

rithms. The FUs used in the first case are those that are the fastest for each operation, and in the second case are those that are the slowest for each operation. These two schedules present some kind of boundary solutions, since all the other solutions are executed in between the time limits defined by these two schedules. In other words, no other solution can be faster or slower, irrespective of the combinations of used units.

Each solution has to be properly encoded (into the chromosome), i.e., each operation's start time and functional unit have to exist in the chromosome. The initial population is built upon the two initial solutions, which are multiplied to form the population with the so-called boundary solutions. The optimal solution has to be somewhere in between the boundaries, therefore genetic operators (crossover, mutation, variation) transform those encoded solutions. With the transformations their start times and allocated functional units are changed. Within that, also the multicycling solutions are supported by the approach. The final solution obtained using the genetic operators is also influenced by the simulated annealing algorithm, which improves the solution if it stopped somewhere near the globally optimal point.

4.1 Encoding

The performance of the algorithm depends on the proper encoding. In the ECSA algorithm integer encoding is used, i.e., in the chromosome string are the numbers that represent the starting time of each operation and the allocated unit for each operation, where the position in the string depends on the order of the operations in the input IC description. This means that the chromosome consists of pairs of time/unit information for each operation. And the genetic operators can influence both parts of that information, either together or separately.

Integer encoding was chosen, since it does not need any transformation (into binary values) at the beginning and at the end (back into decimal values). Also, the used implementation of genetic operators can check the changed values instantly, without any transformation. The correctness of the transformation, i.e., the crossover, the mutation or the variation, can therefore be checked within the function itself.

4.2 Cost function

One of the most important parts of the algorithm is its cost function. Our algorithm is multi-objective /3/, which means it takes control over more criteria or objectives. The cost function, represented by Eqn. 1, considers the number of functional units, the number of registers, the number of buses and the execution time of all the operations in the DFG.

$$
\begin{aligned}
Cost &= \sqrt[2]{\sum_{i=1}^{n}(cost_{FU_i})^2 + cost_{Reg}^2 + cost_{Bus}^2 + cost_T^2} \\
cost_{FU_i} &= w_{FU_i} \cdot FU_i \\
cost_{Reg} &= w_{Reg} \cdot Reg \\
cost_{Bus} &= w_{Bus} \cdot Bus \\
cost_T &= w_T \cdot T
\end{aligned}
\qquad (1)
$$

To obtain the cost of a certain DFG, the algorithm has to evaluate the required number of resources. In contrast to the other multi-objective functions that give more than one final solution, this one already includes the decision-making part, i.e., it chooses one solution from all the solutions on the Pareto front. The chosen solution has the shortest distance to the origin, where the origin represents the ideal, costless, solution and the axis represents the considered objectives.

The main weakness /3/ of this approach is the difficulty in determining the appropriate weights when there is not enough information about the problem. Since we are aware of the problem specifics and we know the cost weights of the resources being used, this weakness is not so significant.

4.3 Evolutionary operators

In each iteration (or generation) of the algorithm there are four genetic operators that transform the chromosome. They consider data dependencies and the given library of available functional units. Each time after the genetic operators transform the chromosome, the chromosome is checked to see if it meets all the constraints by considering data dependencies and unit types. Besides the basic implementation of the operators, we also applied the independent operators, which do not need any parameter value to be set in advance: they depend only on the progress of the search and on the size of the problem to be solved.

4.3.1 Basic operators

Selection. Based on the cost-function values the worst solutions are aborted in the selection step. And to ensure that the size of the population remains the same, these solutions are replaced with the best solutions. This ensures that the best solutions of a given generation are involved in the creation of the next generation (elitism).

Crossover. In a crossover task, two approaches are used, with each task expressing the dominancy of the characteristics. After two crossover points are determined, in the first case the unit information is changed between the two chromosomes and the start times are adapted, and in the second case the start times are changed and a suitable unit is allocated. So the dominancy is expressed either in functional units or the start times of operations.

Mutation. We also have two similar approaches for transforming the chromosome. In both cases the starting time

is changed. Either it is moved to later control steps, with the use of faster functional units, or it is moved to earlier control steps, if data dependencies allow this, with slower units.

Variation. After two operations are selected, and when they are of the same type, e.g. additions, their functional units are switched. If needed, their start times are also updated.

4.3.2 Independent operators

The advantage of the independent GA approach /12/ is that there is no need to preset some working parameters, e.g. the number of generations, the population size, and the probabilities of crossover, mutation and variation. These parameters are set automatically during the optimization phase, depending on the progress and the speed of the optimization.

Setup. If the chromosome that presents a solution is large, then the population size also has to be large enough to ensure that a lot of different chromosomes will be involved in a search. The population size therefore depends on the size of the chromosome or the complexity of the problem.

Crossover. Considering four candidates - two parents and their two offspring - only the first and the third, rated according to their fitness, pass to the next generation. This forces at least one of the offspring to be passed to the next generation in addition to the best candidate. Otherwise the offspring have only a small influence on new generations, since the crossing of two good parents probably produces offspring that are not so good; however, they might be good after a few more transformations.

Mutation. Chromosomes with low fitness are mostly exposed to mutation. Each bit in the chromosome is mutated if that position of the chromosome is of the same value in the majority of chromosomes in the population. This is the way to change the bad characteristics in "poorly fitted" chromosomes and to redirect the search to another direction. In the case of "well-fitted" chromosomes, bits are mutated if the value of the bit differs from majority of bits in other good chromosomes at the same position. This ensures faster convergence in the final stages of the optimization.

Variation. The interchange of the values of two bits, as described for the basic operators, is performed if the frequency of the value in that position in the population of one bit is high and the frequency of another bit is low.

4.3.3 Simulated annealing

When the GA finishes its work and the most appropriate solution is found, that solution is additionally influenced by the simulated annealing algorithm. It checks whether the search for the optimal solution stopped in the vicinity of the global optimum, and it changes the solution if needed.

5 Evaluation

The appropriateness of the proposed approach was tested by a computer implementation of the ECSA algorithm, which was used with test-bench ICs. The ICs used for the evaluation were chosen based on their appearance in the literature and similar studies. They differ in terms of size and the number of operation types.

5.1 Test-bench circuits

5.1.1 Differential equation

The relatively small circuit of differential equation /14/ has only 11 operations, but 4 different operation types (6 multiplications, 2 additions, 2 subtractions, 1 comparison). This circuit is useful when testing libraries with different implementations of the same operation types.

5.1.2 Elliptic filter

This filter /6/ consists of 34 operations, but only two operation types: 26 additions and 8 multiplications. The circuit is suitable for comparison, due to its size and operation dependencies, since they form two independent, similar critical paths, both influencing the circuit delay.

5.1.3 Bandpass filter

One of the implementations of the bandpass filter /5/ is the circuit used for our evaluation. It consists of 29 operations: 11 multiplications, 10 additions and 8 subtractions. Due to data dependencies, almost all the operations influence the circuit delay.

5.1.4 Least-mean-square filter

This filter for signal adaptation (noise reduction) is based on the least-mean-square method /2/. It consists of 47 operations: 24 multiplications and 23 additions. This test-bench circuit is useful due to its size and unique data dependencies.

5.2 Functional units

For an easier and more realistic comparison of different algorithms when testing the size and delay of implemented circuits we made a library of different functional units, which differ in terms of their sizes and delays. Table 1 shows the sizes and delays of various implementations of the arithmetic logic operations. Here, different types of logic were used to make the units with different delays and sizes. The values are based on an analysis of data on circuits and their complexities /11/. The number of gate transitions defines the delay, while the overall number of gates needed to implement the unit defines its size. These values are just for orientation, since the real numbers depend on the chosen technology /11/. The delays presented in Table 1 are relative, e.g., normalized to the fastest functional unit among all the operations. Most of the units are multifunctional, i.e., they can perform different types of operation.

Table 1. Technical characteristics of the functional units

functional unit {operations}	delay [No. of steps]	No. of gates
FE1 {+, -}	6, 6	370
FE2 {+, -}	1, 1	665
FE3 {<}	6	353
FE4 {+, -, <}	1, 1, 1	696
FE5 {x}	4	3040
FE6 {x}	2	7296
FE7 {÷}	21	3040
FE8 {÷}	9	7296
FE9 {x,÷}	4, 21	3344
FE10 {x,÷}	2, 9	8025
FE11 {+, -, <, x, ÷}	1, 1, 1, 4, 21	3692
FE12 {+, -, <, x, ÷}	1, 1, 1, 2, 9	8373

5.3 Parameters

By considering 18750 different schedules of each circuit with the ECSA algorithm and 3125 different combinations of the parameters, we statistically compared (using the procedure described in /10/) the results according to their cost function (Eq. 1). To ensure that most solutions were time-constrained, i.e., executed in shortest possible time, the weight w_T was set to an extremely high value.

As presented in *Table 2*, the high-quality solutions are mostly obtained with the following values of the parameters: probability of crossover, 0.7; probability of mutation, 0.04; and probability of variation, 0.03. In addition, considering the sizes of the circuits, the number of generations and the population size should be set to 3-times and 3.5-times the size of the circuit, respectively.

Table 2. Optimal values of the parameters for different testbench circuits

	Differential equation	Fifth-order elliptic filter	Bandpass filter	Least-mean-square filter	Average optimal values
number of generations	40	100	90	130	3 x DFG size
population size	55	120	110	160	3.5 x DFG size
probability of crossover	0.8	0.6	0.7	0.6	0.7
probability of mutation	0.04	0.05	0.04	0.05	0.04
probability of variation	0.04	0.02	0.02	0.05	0.03

The values of the parameters in this combination are referred to as the optimal values. These optimal values are determined on the basis of the percentage of solutions with certain parameters from among the good solutions. A parameter value that is to be considered as optimal should have at least a 25% share of the high-quality solutions, as well as having a less than 10% share of the low-quality solutions.

The ECSA algorithm was used with the values of the parameters as presented in *Table 2*. Other parameters needed to run the FDS and ECSA algorithms and the cost function depended on the sizes of the FUs.

5.4 Results

The ECSA algorithm was evaluated by a comparison with nearly optimal /13/ force-directed scheduling (FDS) /12/. FDS tries to optimally schedule the DFG considering a uniform distribution the operations of the same type over the available control steps.

Table 3 presents the results of the following evaluations: FDS with fast units, FDS with slow units, and ECSA with basic and independent genetic operators. There are two types of DFGs for each circuit. The first, or plain, is an ordinary data-flow graph with nodes that represent operations, as described in similar studies; and the second, or improved, considers the input variables (start registers) via some additional nodes to ensure a more accurate estimation of the registers and the buses needed to implement the circuit.

5.4.1 Differential equation

Because of the small circuit size there is no improvement in the solutions obtained with the ECSA algorithm (either basic or independent) when considering an ordinary DFG - all the solutions are of a larger size. But when we consider the start registers (input variables) there are some ECSA solutions with a slightly larger size and a smaller number of buses.

5.4.2 Fifth-order elliptic filter

The evolutionary method with a basic approach found a smaller circuit with a smaller number of buses and a slightly longer execution time for the ordinary DFG, while the independent approach could not find any improved solution. When dealing with the improved DFG, both approaches (basic and independent) found considerably smaller circuits with a slight increase in the execution time, while the independent approach also found the solution with a substantial decrease in the required number of registers and buses.

5.4.3 Bandpass filter

Both ECSA methods found, when dealing with the ordinary DFG, the solutions with a smaller number of registers and buses; the basic approach also found the smaller circuit, but with a slightly longer execution time. When dealing with the improved DFG, both approaches found the solutions with the same circuit size and execution time as the comparable FDS solution, but the required number of registers and buses was considerably smaller for the ECSA solutions.

5.4.4 Least-mean-square filter

At the expense of a small increase in the delay, the basic ECSA was able to decrease the size and lower the number of registers and buses of the ordinary DFG; but the independent ECSA was not able to improve any parameter. When dealing with the improved DFG, the basic ECSA was able to keep the initial delay, to decrease the circuit size and to lower the number of required registers and buses. The independent ECSA was only able to decrease the number of buses while increasing the circuit size.

Table 3. The evaluation results of the ECSA algorithm with different test-bench ICs

algorithm	functional units	size	registers	buses	delay	runtime [s]
differential equation						
FDS-fast	1xFE2 + 1xFE4 + 3xFE6	23249	17	6	6	0.01
FDS-slow	2xFE1 + 1xFE3 + 2xFE5	7173	18	6	20	0.01
ECSA-basic	2xFE2 + 1xFE4 + 3xFE6	23914	18	8	6	0.11
ECSA-independent	2xFE2 + 1xFE4 + 3xFE6	23914	17	6	6	0.09
differential equation with start registers						
FDS-fast	1xFE2 + 1xFE4 + 3xFE6	23249	10	9	6	0.01
FDS-slow	2xFE1 + 1xFE3 + 2xFE5	7173	11	9	20	0.01
ECSA-basic	2xFE2 + 1xFE4 + 4xFE6	31210	10	7	6	0.15
ECSA-independent	2xFE2 + 1xFE4 + 3xFE6	23914	10	7	6	0.35
fifth-order elliptic filter						
FDS-fast	3xFE2 + 3xFE6	23883	26	8	17	0.02
FDS-slow	5xFE1 + 3xFE5	10970	30	6	78	0.03
ECSA-basic	2xFE2 + 1xFE5 + 2xFE6	18962	29	4	21	3.80
ECSA-independent	4xFE2 + 4xFE6	31844	30	8	17	1.60
fifth-order elliptic filter with start registers						
FDS-fast	3xFE2 + 3xFE6	23883	21	16	17	0.02
FDS-slow	5xFE1 + 3xFE5	10970	25	16	78	0.04
ECSA-basic	2xFE2 + 1xFE5 + 2xFE6	18962	24	16	19	4.80
ECSA-independent	2xFE2 + 2xFE6	15922	18	9	21	3.40
bandpass filter						
FDS-fast	3xFE2 + 4xFE6	31179	34	10	10	0.01
FDS-slow	4xFE1 + 3xFE5	10600	35	8	44	0.04
ECSA-basic	3xFE2 + 3xFE6	23883	33	8	11	1.70
ECSA-independent	3xFE2 + 1xFE5 + 4xFE6	34219	33	8	10	1.30
bandpass filter with start registers						
FDS-fast	3xFE2 + 4xFE6	31179	25	23	10	0.02
FDS-slow	4xFE1 + 3xFE5	10600	26	23	44	0.04
ECSA-basic	3xFE2 + 4xFE6	31179	23	19	10	2.40
ECSA-independent	3xFE2 + 4xFE6	31179	23	19	10	2.90
least-mean-square filter						
FDS-fast	3xFE2 + 6xFE6	45771	68	12	13	0.40
FDS-slow	3xFE1 + 4xFE5	13270	72	8	70	2.79
ECSA-basic	3xFE2 + 6xFE5 + 3xFE6	42123	67	10	14	6.30
ECSA-independent	9xFE2 + 6xFE5 + 9xFE6	89889	69	30	15	8.05
least-mean-square filter with start registers						
FDS-fast	3xFE2 + 6xFE6	45771	33	29	13	0.48
FDS-slow	3xFE1 + 4xFE5	13270	37	27	70	3.52
ECSA-basic	4xFE2 + 2xFE5 + 5xFE6	45220	32	25	13	9.20
ECSA-independent	5xFE2 + 3xFE5 + 7xFE6	63517	33	25	13	12.30

6 Conclusions

Optimally scheduled operations are not necessarily optimally allocated to functional units. To enable optimal allocation we need to consider some allocation criteria while the scheduling is being done. This paper describes such an evolutionary approach that considers scheduling and allocation constraints and ensures a globally optimal solution in a reasonable time. To evaluate our method we built an algorithm and implemented it with a computer. It was used with a group of test-bench ICs. These circuits were chosen because the same type were used in similar studies. They differ in terms of their size and the number of operation types.

It turned out that the evolutionary method (either basic or independent) is able to find a solution that is more appropriate in terms of all the considered and important parameters than is the case when working with classical deterministic methods. There are slightly longer runtimes when the ECSA algorithm is used. But considering the speed (a few seconds) and the computational dependence, where the runtimes for larger circuits increase enormously (exponentially) when the FDS algorithm is used, we can conclude that small and large circuits can be designed and optimized with the use of the proposed evolution-based algorithm, which exhibits a linear increase in the design time with an increase in circuit size.

References

/1/ J. R. Armstrong, F. G. Gray, VHDL Design: Representation and Synthesis, Prentice Hall, Upper Saddle River, 2000.

/2/ J. Benesty, P. Duhamel, A Fast Exact Least Square Adaptive Algorithm, IEEE Transactions on Signal Processing 40, 1992, pp. 2904-2920.

/3/ C. A. Coello Coello, A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques, Knowledge and Information Systems, Vol. 1, No. 3, pp. 269-308, August 1999.

/4/ D. Gajski, N. Dutt, A. Wu, S. Lin, High-Level Synthesis: Introduction to Chip and System Design, Norwell, Massachusetts, Kluwer Academic Publishers, 1992.

/5/ G. W. Grewal, T. C. Wilson, An Enhanced Genetic Algorithm for Solving the High-Level Synthesis Problems of Scheduling, Allocation, and Binding, Intl. Journal of Computational Intelligence and Applications. 1, 2001, pp. 91-110.

/6/ T. Kung, H. J. Whitehouse, T. Kailath, VLSI and Modern Signal Processing, Prentice Hall, 1985.

/7/ G. Papa, Concurrent operation scheduling and unit allocation with an evolutionary technique in the process of integrated-circuit design, Ph.D. Thesis, Faculty of Electrical Engineering, University of Ljubljana, Ljubljana, 2002.

/8/ G. Papa, B. Koroušić-Seljak, B. Benedičič, T. Kmecl, Universal motor efficiency improvement using evolutionary optimization, accepted for publication in IEEE Transactions on Industrial Electronics.

/9/ G. Papa, J. Šilc, Automatic Large-Scale Integrated Circuit Synthesis Using Allocation-Based Scheduling Algorithm, Microprocessors and Microsystems 26, 2002, pp. 139-147.

/10/ G. Papa, J. Šilc, Evolutionary Synthesis Algorithm - Genetic Operators Tuning, In: A. Grmela, N. Mastorakis (ed.) Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, WSEAS Press, 2002, pp. 256-261.

/11/ B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, New York, 2000.

/12/ P. G. Paulin, J. P. Knight, Force-directed Scheduling in Automatic Data Path Synthesis, Proc. 24th ACM/IEEE Design Automation Conference, Miami, USA, June 1987, pp. 195-202.

/13/ P. G. Paulin, J. P. Knight, Scheduling and Binding Algorithms for High-Level Synthesis, Proc. 26th ACM/IEEE Design Automation Conference, Las Vegas, NE, pp. 1-6, June 1989.

/14/ P. G. Paulin, J. P. Knight, E. F. Girczyc, HAL: A Multiparadigm Approach to Automatic Data Path Synthesis, Proc. 23rd ACM/IEEE Design Automation Conference, Las Vegas, USA, June 1986, pp. 263-270.

Gregor Papa
Institut "Jožef Stefan"
Computer Systems Department,
Jamova c. 39, 1000 Ljubljana, Slovenia
tel. +386 1 4773 514
fax. +386 1 4773 882
Email: gregor.papa@ijs.si

Prispelo (Arrived): 13.05.2003 *Sprejeto (Accepted): 26.08.2003*