AN EVOLUTIONARY-BASED ALGORITHM TO THE MODULE SELECTION PROCESS IN HIGH-LEVEL SYNTHESIS

Azeddien M. Sllame, Lukas Sekanina

Faculty of Information Technology,
Bozetechova 2, 612 66 Brno, Czech Republic,
E-mail:sllame@fit.vutbr.cz

Abstract
This paper proposes a new module selection algorithm for high-level synthesis. The algorithm uses an evolutionary approach to find the modules configuration set that satisfies design timing constraints while minimizing the total design cost (area). The algorithm has been incorporated in a well-characterized design space exploration strategy that aims to help designers to systematically find efficient implementation(s) of their designs that meet the design constraints [1]. Incorporating module selection axis to the design space enable designers to evaluate large number of design alternatives by varying module selection and the latency or the resources required to implement the given design. We also present some experimental results for standard benchmarks to show the effectiveness of the algorithm.

Key words : High-level synthesis, module selection, evolutionary algorithm.

1. Introduction

The role of high-level synthesis (HLS) during the hardware-software codesign process is needed in order to allow designers to quickly explore different implementations to the system under design, see the cost and performance figures of the synthesized modules in terms of area, time, power, testability and money. HLS is the design task of mapping an abstract behavioral description of a digital system onto a register-transfer-level (RTL) design to implement that behavior. By using HLS tools in the current state-of-the-art CAD flow systems the designers are allowed to explore the design space more efficiently.
Given that the main design decisions such as the number of hardware resources, clock cycle time, and implementation styles (pipeline, multi-cycle operation, etc.) are made during the scheduling phase, the scheduling step is considered as the most important step in the whole HLS process. In addition, these decisions have strong influences on the following data-path allocation and binding subtasks. For that reason, scheduling directly controls the throughput rate of the produced RTL design and determines the cost-speed tradeoffs of the given design. Thus, the process of exploring the design space must include scheduling phase as one of its main characterization lines, or the exploration process can be viewed (diminished) as solving the scheduling problem. In such a way we can view the solution of the scheduling problem as the process of exploring a 2-dimensional (2D) design space, with one axis representing time (schedule length), and the other representing the area of the design (ideally total design area, but often simplified to functional unit area)**.** Adding other characterization lines such as *module selection* and *pipelining* as main features to the design space exploration process enable designers to get high performance design implementations.
Pipelining can be classified to *structural pipelining* and to *functional pipelining*. Structural pipelining assured by using pipelined functional units during the scheduling phase to implement some operations in the produced schedule, while functional pipelining means subdividing the algorithm description into sequences of operation stages that will be performed concurrently. Including pipelining axis into the design space exploration methodology allows designers produce high performance designs.
Module selection is the problem of choosing a particular functional unit from a library of components for each operation in the given initial schedule. The component library contains different alternative implementations for each resource type, which are characterized by different area and delay estimates. Incorporating module selection axis to the design space enables designers to evaluate large number of design alternatives by varying module selection and the latency or the resources required to implement the given design.
This paper proposes a new module selection algorithm for high-level synthesis. The algorithm uses an evolutionary approach to find the *modules configuration set* that satisfies design timing constraints while minimizing the total design cost (area). However, we mean by the term *modules configuration set* the complete set of modules that are selected from the components library (*CL*) to implement the design schedule such that it satisfies the total design specified latency. This set may include none or many instances of the same module that exists in the *CL*.
The algorithm has been incorporated in a well-characterized design space exploration strategy that aims to help designers to systematically find efficient implementation(s) of their designs that meet the design constraints [1]. We also present some experimental results for standard benchmarks to show the effectiveness of the algorithm While our well-characterized design space exploration methodology incorporates scheduling, module selection and structural pipelining only for the time being, we are currently working to include functional pipelining feature into the proposed methodology [1].

The paper is organized as follows: Section 2 reviews related research. Section 3 describes the proposed algorithm. Section 4 illustrates the Experimental results. Finally, concluding remarks are given in Section 5.

2. Previous Work

During the design space exploration process, designers seek to find one or more "optimized" implementation(s) for a given behavioral specification. In this paper, we consider the design area estimated through the cost of functional units only, and performance of the final design measured as the total latency of the produced schedule.

MOSP algorithm [8], provided a solution to the more restricted form of module selection problem wherein all instances of the same operation type are implemented using the same module type. Even though MOSP initially starts with a multiple implementation library, the final design implementation is inefficient since it contains only single implementation for each operation type.

MSSR algorithm [7], attempts to solve the scheduling, resource sharing and module selection at the same time. The algorithm is based on iterative improvement and structured in three procedures. The procedure *OPSL*, which is the third procedure in the algorithm structure, is based on the longest path algorithm. The procedure *OPSL* is used to select appropriate resources from the associated library to be mapped to each hyperedge. While the MSSR solves the scheduling, resource sharing and module selection iteratively, our module selection algorithm selects proper modules configuration set to the initial schedule that produced by list-based scheduler in which resource sharing and the use of pipelined functional units is already ensured.

Bakshi in [9] [10] proposed a component selection algorithm combined with pipelining and scheduling process to characterize the design space exploration. The mentioned component selection algorithm uses a heuristic approach to create component substitution lists to guide the greedy approach being used to solve the module selection problem, whereas our methodology uses an evolutionary based approach.

Timmer in [3] proposed module selection algorithm and scheduling using unrestricted libraries. The algorithm starts by initial module selection, and then a list scheduler tries to meet the time constraints with this selection. If the scheduler does not succeed, the module selection has to be reviewed until a correct selection has been made. The algorithm formulates the module selection problem using MILP formulation and iterates the module selection process before finding the satisfactory schedule, which is a very time consuming process. The algorithm also constrains the number of states or the number of modules. Unlike Timmer algorithm, our module selection algorithm finds the modules configuration set that satisfies design time to the given optimized schedule, we don not need to repeat the given schedule, we may need to repeat module selection process only if the components library is updated.

The algorithm presented in [11] combines module selection with pipelining, but the algorithm does not support the use of pipelined components and the algorithm uses *integer linear programming formulation* (ILP) that requires exponential execution time, as well as it selects only one candidate module to implement all operations of the same type.

Chantana in [4] reported a module selection and scheduling algorithm. While in that work they presented a module selection algorithm based on fuzzy logic theory, but their algorithm selects only one candidate module to implement all operations from the same selected module type, which in turn will produce inefficient design.

Ahmed in [6], integrated scheduling, allocation and module selection in the design space exploration process using a problem-space genetic approach (PSGA). Genetic algorithms are global probabilistic search techniques that start from an initial population of generated potential solutions to a problem, and gradually evolve towards better solutions through a repetitive application of genetic operators like (crossover and mutation). In PSGA approach, each chromosome in the initial population consists of two parts: priorities of nodes, and type of functional units and number of functional units of each version. The priority of each node in the first chromosome is determined by the maximum number of successors on the critical path of that node [6]. Whereas in our case we are applying the evolutionary algorithm only to find the efficient modules configuration set to the given schedule, which was produced by a list-based scheduler that incorporates the mobility and underlying DFG structure in the scheduler priority function in order to produce a well-organized schedules [2]. In addition, the approach incorporates simple priority function in the same chromosome that will lead to inefficient schedules; hence, it will affect the module selection process as well.

3. The proposed algorithm overview

In this section, we intend to describe the evolutionary based module selection algorithm. The algorithm starts by reading the following inputs: (1) the initial schedule that satisfies design area constraints produced from the list-based scheduler described in [2]; (2) the required design completion time (design total latency) for the final design; and (3) *CL,* which is used by the module selection algorithm to find the best *modules configuration set* that satisfies design timing with a minimum design cost. A sample of *CL* that is adopted from [10] is shown in Table 1.

Evolutionary algorithms were successfully applied to optimization [13] and design [14] in recent years. It is also known from those references that the evolutionary algorithm does not guarantee that the optimal solution is reached in every run. Furthermore, No Free Lunch theorems claim that the average performance of an evolutionary algorithm and random search across all possible problems is identical [15]. On the other hand, if the problem domain is ingeniously captured in the algorithm then the algorithm can outperform traditional solutions [15] [16]. The main goal of our *evolutionary based module selection algorithm* is to optimize the design area (estimated only by the functional units areas) while satisfying the design specified total latency. In another words, for every design point (initial schedule),

which is supplied as an input, there are many modules configuration sets each for every individual design total latency value and the goal of the evolutionary algorithm is to find the set that leads to the minimum design area. An example of *fifth order elliptic filter* (EWF) [12] initial schedule with two multipliers and two adders as a resource set is illustrated in Table 2. It is clear from the table that operations are still not mapped to their final modules that will execute them. Moreover, we need to mention that the term *resource set* which appear in the paper context and the note *resource set as (+2, *3)*, which appear in figures 3 through 6 and Tables 5 and 6, means: the maximum allowable number of adders and multipliers for each c-step for the given schedule is 2 adders and 3 multipliers. This in turn, will help the designers in the case of modifying the design final schedule by applying resource sharing.

Problem representation

The initial schedule supplied as an input to the algorithm is represented as a sequence of integers. For instance, Table 2 describes an initial schedule for the EWF with a resource set of two adders and two multipliers. The chromosome representation of the same schedule is illustrated in Table 3 where: (0 denotes a *, 1 is an +, 2 is a -, 3 is a <>) and SEP is a separator of the schedule c-steps (e.g. SEP = -1).

The type of a component is given by its position in the string implicitly. A variant of the component is given by a number (allele of the gene) in the chromosome at a given position. The fastest component is always represented as 0. Thus, the number of variants of a given component defines a set of legal values of a given gene. The representation of the problem is natural, but more variants of a component than it is required may appear in the final schedule.

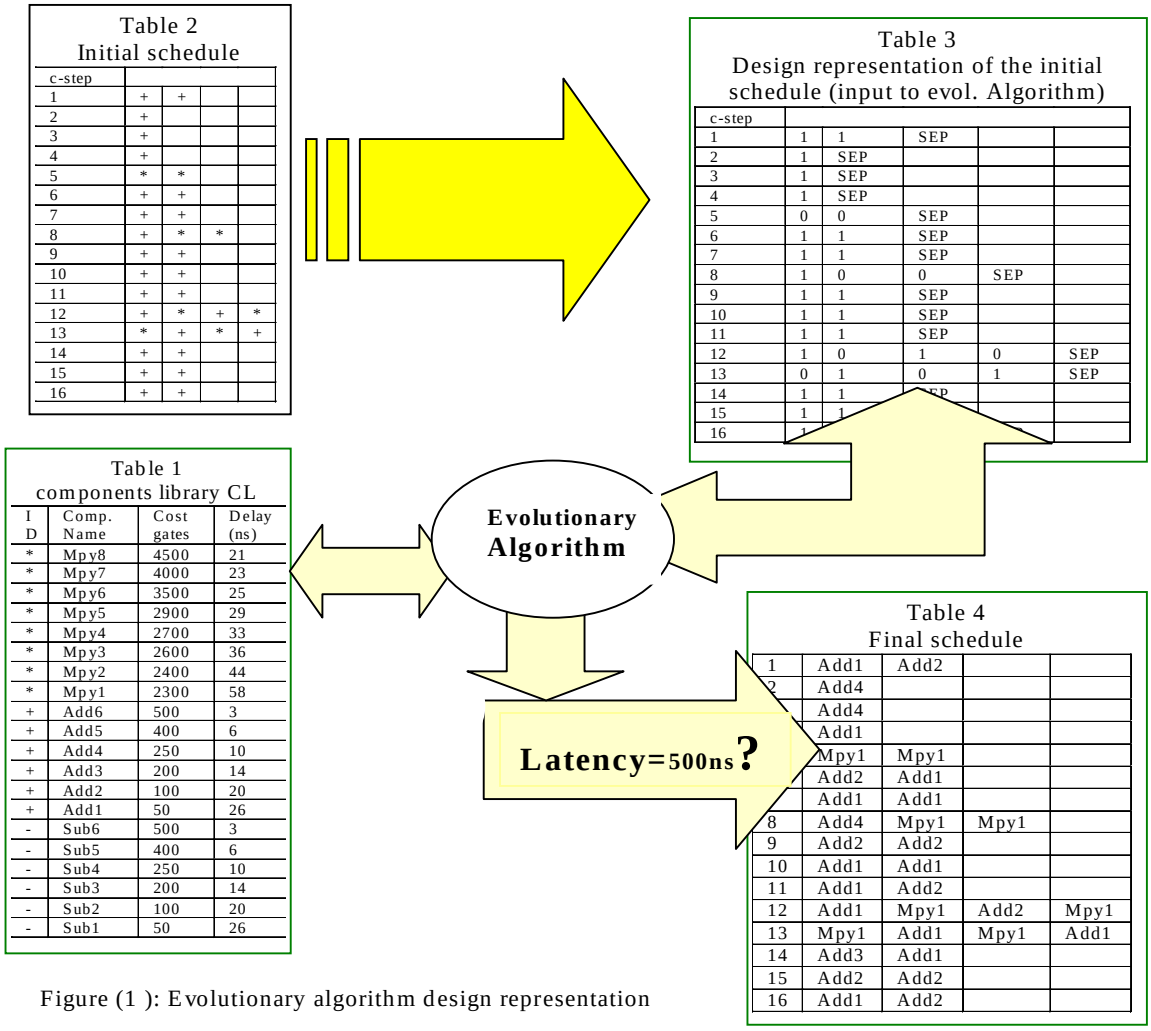

Figure (1): Evolutionary algorithm design representation

Parameters of the evolutionary algorithm

One-point crossover is applied with probability p_c = 60%. Two randomly selected genes are mutated per chromosome if crossover is not used. Both two operators produce correct circuits according to the schedule. Tournament selection with base 2 and elitism are employed. Initial population is generated either from the fastest variants or the slowest variants or from a combination (50:50) of the fastest and the slowest one. We have found from the experiments that (50:50) combination yields better convergence than if we start from the fastest combination or from the slowest one.

Fitness function

The fitness function assigns higher values to chromosomes that exhibit latency (L) equal to the required latency (RL) and that minimize the number of gates (G) needed for implementation.

FitnessValue= 1 *if* *L > RL;*
*=BEST_FITNESS - G - |L - RL| * 5* *otherwise;*

where *BEST_FITNESS* is a sufficiently high value.

Design latency calculation

Latency L is calculated as a sum of latencies L_i of the slowest components in each schedule c-step used in the chromosome, Which means the component that has the maximum delay (the slowest) value represents the delay of the corresponding c-step in the schedule.

$$L = \sum_{i=1}^{n} L_i$$

where n is the number of schedule c-steps.

For instance the latency of the design example given in Table 4, calculated as follows: (L(Add1)+ L(Add4)+ L(Add4)+ L(Add1)+ L(Mpy1)+ L(Add1)+ L(Add1)+ L(Mpy1)+ L(Add2)+ L(Add1)+ L(Add1)+ L(Mpy1)+ L(Mpy1)+ L(Add1)+ L(Add2)+ L(Add1) = 26+ 10+ 10+ 26+ 58+ 26+ 26+ 58+ 20+ 26+ 26+ 58+ 58+ 26+ 20+ 26 = 500 ns.

Design area calculation

The area (G) occupied by the solution, which used in fitness calculation, is estimated according to the number of resources. Let $V^i = (v_1^i, ..., v_k^i)$ denote a vector of implementation costs of k resources in the i-th c-step.

$$G = \sum_{j=1}^{k} Max_{i=1}^{n}(v_j^i)$$

The design total area (A) is calculated according to the number of resources in every c-step.

$$A = \sum_{i=1}^{n} \sum_{j=1}^{k} v_j^i$$

Following the Table 4, the design total is calculated as follows: (A(Add1)+ A(Add4)+ A(Add4)+ A(Add1)+ A(Mpy1)+ A(Add1)+ A(Add1)+ A(Mpy1)+ A(Add2)+ A(Add1)+ A(Add1)+ A(Mpy1)+ A(Mpy1)+ A(Add1)+ A(Add2)+ A(Add1) = 50+ 250+ 250+ 50+ 2300+ 50+ 50+ 2300+ 100+ 50+ 50+ 2300+ 2300+ 50+ 100+ 50=10300 gates.

From the presentation above we see that the algorithm always tries to find the best modules configuration set that meets the specified design latency if one exists, otherwise either the components library is updated or the resource-constrained scheduling is repeated again (i.e. the designer creates new initial schedule) with another set of resources. However, the designer always has the ability to modify the produced results or to repeat the entire process with new synthesis constraints. The designer can also decide to save more design area by introducing resource sharing respecting the resource set used in the initial schedule. This in turn will restrict the pipelining of the design if the pipelining concept supposed to be employed on the given design.

Evolutionary algorithm:
t = 0
Create initial population
Evaluate population
while (t < tmax) do
    select parents for mating using
    2-tournament selection
    apply crossover with p_c = 60%
    if crossover is not used then mutate
    2 genes of parents
    create a new population from child
    chromosomes (ensure elitism)
    evaluate population
    t = t + 1
end

Figure 2: Overview of the evolutionary algorithm

4. Experimental results

The presented module selection algorithm has been implemented in C language. Figures 3 through 6 show the experimental results of the proposed algorithm. These experiments demonstrate the impact of module selection on the design cost of *discrete cosine transform* (DCT) [6] and *fifth order elliptic wave filter* (EWF) [12] benchmarks. In addition, the graphs in figures 3 through 6 illustrate the design space exploration time-area curve with different design constraints for DCT and EWF examples. For the illustrated experiments, it is assumed that the delay of a multiplier and an adder is 1c-step for simplicity only; our scheduling algorithms support muticycled and pipelined functional units.

In all employed experiments, we compare designs obtained using complete set of components in CL with a reduced set of components in CL, by keeping constant all design parameters other than the component library. The reduced set of components include only the fastest and the slowest components from each operation type that found in the complete CL. The reason behind our choice of using the fastest and the slowest from each type is to use the same starting and ending points for the design space in both experiments in order to clearly illustrate the difference between both complete CL and reduced CL design costs.

From all carried experiments we can remark that the lowest-area designs are those obtained with module selection using a complete set of components. Using reduced CL the designers are unable to find many design points, as well as many design points with different latency values are mapped to the same area value, which in turn produces a disturbed design space curve. However, the design space curve is smoother in the case of using complete CL.

Also, we need to mention that we are unable to compare our results with those reported in [10] because the results given in [10] include functional pipelining, which is not supported yet with our design space exploration methodology.

Tuning the algorithm

Tuning the evolutionary algorithm was done using the changing of the population size, maximum number of generations, and % of crossover in order to find desirable values that lead to the maximum number of occurrences of the best solution within ten runs. The worst CPU time (PIII machine) for the ten runs of DCT example was about (06:36 minutes), while it is found as (02:42 minutes) for the ten runs of EWF example. Tables 5 and 6 report such results.

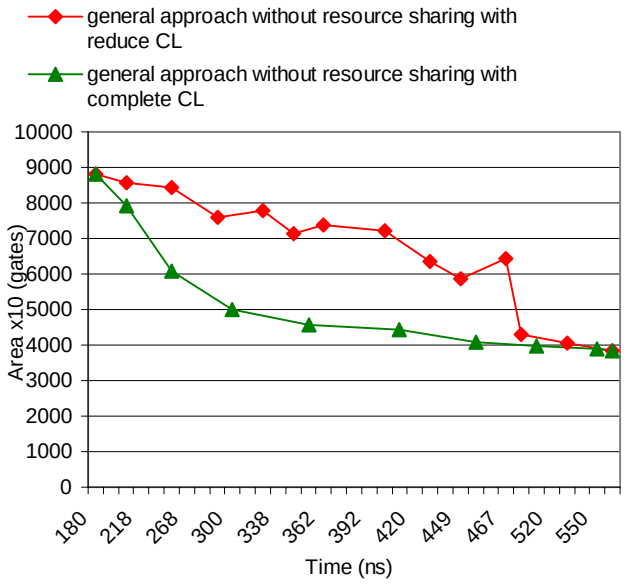

Figure (3): Design space exploration of DCT design with resource set as(+3, *2)

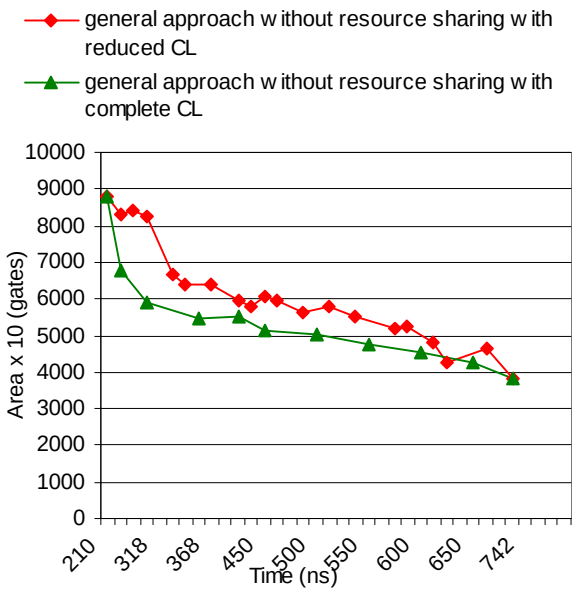

Figure (4): Design space exploration of DCT design with resource set as (+2, *3)

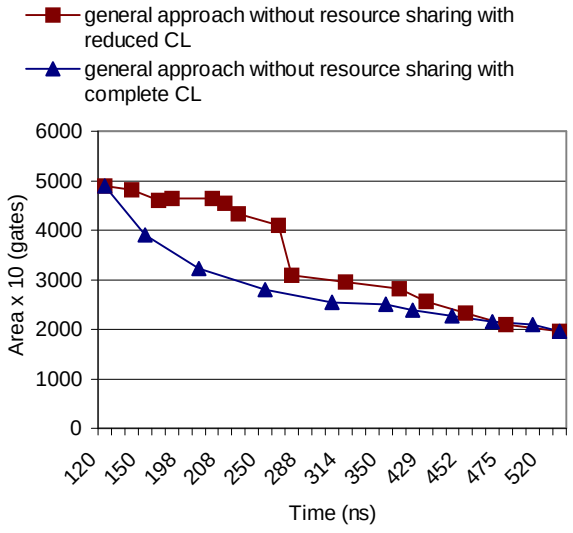

Figure (5): Design space exploration of EWF design with resource set as (+2, *2)

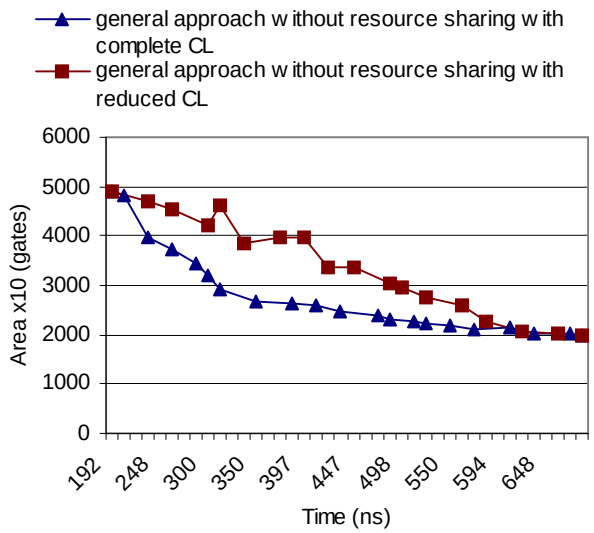

Figure (6): Design space exploration of EWF design with resource set as (+2, *1)

5. Conclusions

We have presented an evolutionary based algorithm to solve the module selection problem in high-level synthesis systems. However, the algorithm has been integrated in a well-characterized design space exploration methodology [1] that helps designers to explore and hence, produce efficient designs in a reasonable time. We have carried out many experiments with different benchmarks, these experiments allow us to claim that our new algorithm is efficient enough and able to help designers to find the efficient modules configuration set that satisfies design total latency with the minimum design cost (area).

The results presented in this paper are encouraging, so we propose to continue this research by further tuning the algorithm toward more efficient results by including resource-constraints to the core of the presented algorithm, and we will incorporate functional pipelining as a primary feature in our design space exploration methodology.

It should be noted that unlike other HLS problems, for design space exploration, there is no unified components library (*CL*) that exist as a benchmark (to the authors' knowledge), which could be followed by all research groups to compare results and improve the available design space exploration methodologies.

Table 5
Algorithm results and tuning for the DCT design with resources (+2, *3)

no. of runs=10, population size =130, max. no. of generations = 10000				
Required Latency	Min. area achieved	The number of runs the best fitness occurred	The best fitness during the runs	The average generation in which the best fitness of the experiment has occurred
210	88000	10	8550	0
250	67550	7	9090	1030.29
300	59050	3	9180	5001.67
350	54400	2	9230	10011.00
400	55150	5	9260	4291.40
450	51500	1	9270	5140.00
500	50200	1	9290	5616.00
550	47400	6	9290	9712.67
600	45400	8	9290	1782.75
650	42600	3	9300	3115.67
704	38400	10	9300	0

Table 6
Algorithm results and tuning for the EWF design with resources (+2, *2)

no. of runs=10, population size =60, max. no. of generations = 70000				
Required Latency	Min. area achieved	The number of runs the best fitness occurred	The best fitness during the runs	The average generation in which the best fitness of the experiment has occurred
120	49000	10	3000	0
150	39050	2	3200	17546.50
200	32150	6	3380	4516.83
250	28050	1	3440	1433.00
300	25500	5	3470	7569.60
350	25000	2	3490	1684.50
400	23850	6	3490	1160.50
450	22750	9	3500	4270.22
500	20900	6	3515	1657.00
544	19700	10	3530	0

Acknowledgements
The research was performed with the grant agency of the Czech Republic under no. 102/01/1531 Formal approach to digital circuit diagnostic - testable design verification, and the research intent no. CEZ. J22/98: 262200012 – Research in information and control systems.

References

1. A. M. Sllame, V. Drabek: *A Design Space Exploration scheme for High-Level Synthesis Systems*, In proceedings of 36th International Conference Modelling and Simulation of Systems MOSIS '02, Ostrava, Czech Republic, April 2002, pp. 305-312.
2. A. M. Sllame, V. Drabek: *An Efficient List-Based Scheduling Algorithm for High-Level Synthesis*, EuroMicro DSD'2002, Dortmund, Germany, September 2002, pp.8, (to appear).
3. A. H. Timmer, M. J. M. Heijligers, L. Stok, J. A. G. Jess: *Module Selection and Scheduling Using Unrestricted Libraries*, Proceedings of the EDAC/EuroASIC Conference, Paris, France, February 1993, pp. 547-551.
4. C. Chantrapornchai, E. H.-M. Sha, X. S. Hu: *Efficient Design Exploration Based on Module Utility Selection*, IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 19, No. 1, January 2000, pp. 19-29.
5. D. D. Gajski, N. Dutt, A. We, S. Lin: *High-Level Synthesis Introduction to Chip and System Design*, Kluwer Academic Publishers, 1994.
6. I. Ahmed, M. K. Dhodhi, C. Y. R. Chen: *Integrated Scheduling, Allocation and Module Selection for Design-Space Exploration in High-Level Synthesis*, IEE Proc. Comput. Digit. Tech., Vol. 142, No. 1, Jan. 1995, pp. 65-71.
7. M. Ishikawa, G. De Micheli: *A Module Selection Algorithm for High-Level Synthesis*, In Proc. Of the IEEE Intr. Symp. On Circuits and Systems, 1990, pp. 1777-1780.
8. R. Jain: MOSP: *Module Selection for Pipelined Designs with Multi-Cycle Operations*, In Proc. Of the IEEE/ACM Int. Conf. On CAD, September 1990, pp. 212-215.
9. S. Bakshi, D. D. Gajski, H.-P. Juan: *Component Selection in Resource Shared and Pipelined DSP applications*, IEEE EURO-DAC'96 with EURO-VHDL'96.
10. S. Bakshi, D. D. Gajski: *Component Selection for High-Performance Pipelines*, IEEE Transactions on Very Large Scale Integration Systems, Vol. 4, No. 2, 1996, pp. 181-194.
11. S. Notes, F. Catthoor, G. Goossens, H. J. De Man: *Combined Hardware Selection and Pipelining in High-Performance Data-Path Design*, IEEE Transaction on Computer-Aided Design, Vol. 11, No. 4, April 1992, pp. 413-423.
12. S. Y. Kung, H. J. Whitehouse, T. Kailath: "*VLSI and Modern Signal Processing*", Prentice-Hall, Inc., 1985.
13. T. Bäck: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.
14. P. Bentley: *Evolutionary Design by Computers*. Morgan Kaufmann Publisher, 1999.
15. D. H. Wolpert, W. G. Macready: *No Free Lunch Theorems for Optimization*. IEEE Transaction on Evolutionary Computation, Vol. 1, No. 1, April 1997.
16. Z. Michalewicz, D. B. Fogel : *How to solve it: Modern Heuristic*. Springer Verlag, 2000.