A Survey of High-Level Synthesis Techniques for Area, Delay and Power Optimization

S.M. Logesh
Department of Electronics and Communication Engineering
Amrita Vishwa Vidyapeetham University
Coimbatore, India

D. S. Harish Ram
Department of Electronics and Communication Engineering
Amrita Vishwa Vidyapeetham University
Coimbatore, India

M.C. Bhuvaneswari
Department of Electrical and Electronics Engineering
PSG College of Technology
Coimbatore, India

ABSTRACT
With increasing complexity of digital signal processing VLSI circuits in recent decades, design methodologies and tools have moved to higher abstraction levels. High level Synthesis has been gaining lot of interest in recent years since the major design objectives such as area, delay and power of the circuit are mutually conflicting thereby necessitating trade-offs between different objectives. The electronic system-level (ESL) paradigm facilitates exploration, synthesis, and verification that can handle the complexity of today's system-on-chip (SoC) designs. Processor customization and High Level Synthesis have become necessary paths to efficient ESL design. This paper presents the survey of high level synthesis approaches and methodologies for simultaneous area, delay and power optimization.

Keywords
High level synthesis; Design space exploration; System level design; Genetic Algorithm; Optimization; Allocation; Scheduling; Binding; Dataflow graph; Behavioral description

1. INTRODUCTION

High-level synthesis (HLS) is the process of translating abstract behavioral description into a hardware implementation at register transfer level (RTL). The design specification is usually written as a behavioral description, in a language such as C and compiled into data flow graphs (DFGs). DFG's are then mapped to the functional units that are selected from the resource library to meet design goals (power, area, and performance). Automation of the translation from a behavioral level of abstraction to RTL results in considerable savings in design cycle time. VLSI designs are multi-objective by nature, since they have to trade-off several conflicting design objectives such as chip area, circuit delays, and power dissipation. The shorter design time using behavioral synthesis allows one to examine many alternative circuit realizations during the design process, thus enabling a wider design space exploration for more optimal designs.

2. PHASES OF HIGH-LEVEL SYNTHESIS

2.1 Allocation
Allocation is a process of allocating resources or Functional Units (FUs) to execute a particular operation. Allocation defines the type and the number of hardware resources like FU's, registers and multiplexer which are necessary to satisfy the design constraints. Resources are selected from the RTL component library. The library must also include component characteristics (such as area, delay, and power) and its metrics to be used by other synthesis tasks. For instance, one possible allocation for the DFG in Fig 1 is three adders and one multiplier.

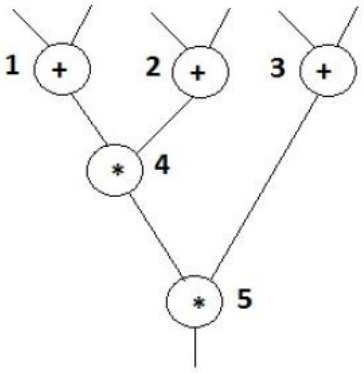


Fig. 1 Data Flow Graph (DFG)

2.2 Scheduling

All operations required in the specification model must be scheduled at time steps or clock cycles. For each operation, variables must be read from their sources (either Registers or FUs) and brought to the input of a functional unit that can execute the operation and the result must be stored in registers. Depending on the functional component to which the operation

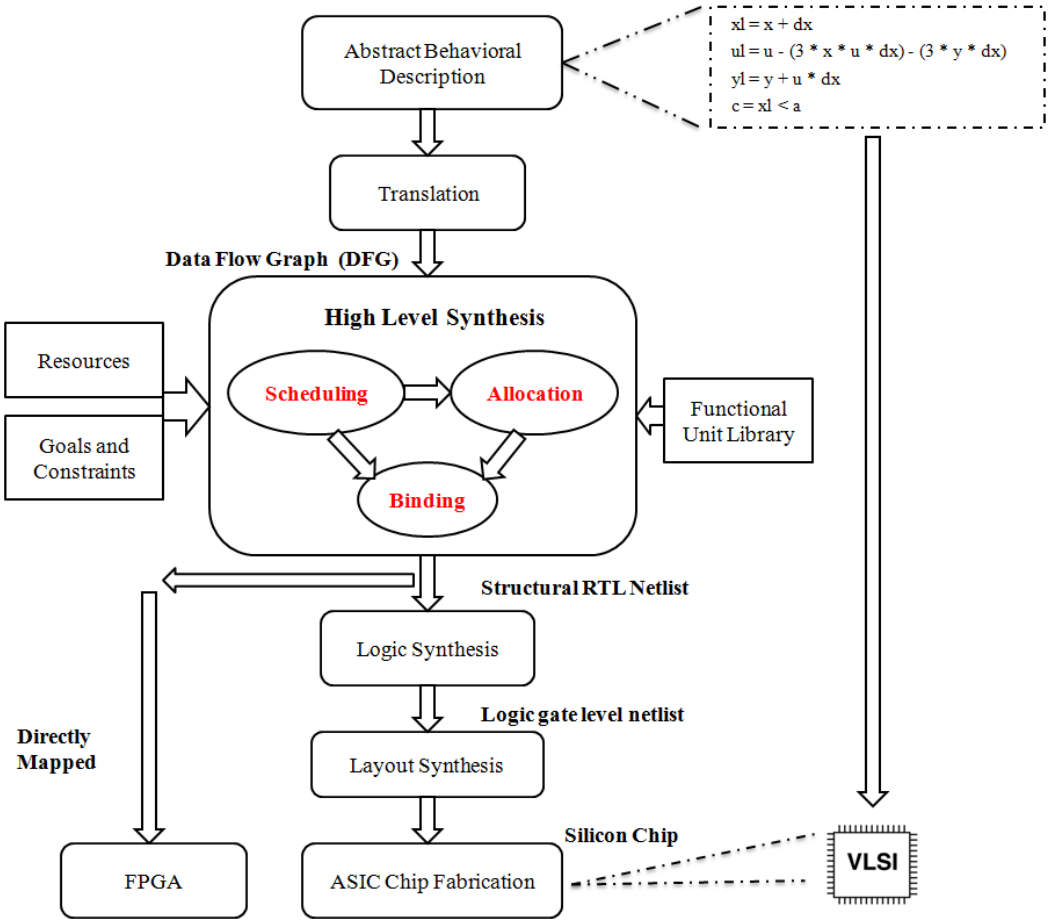


Fig. 2 Silicon Compilation through High Level Synthesis Subtasks

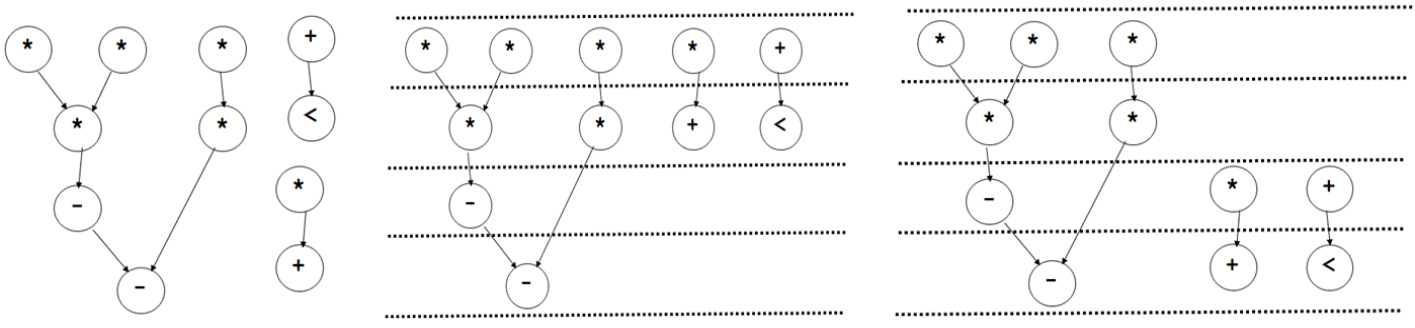
is mapped, the operation can be scheduled within one clock cycle or scheduled over several cycles. Operations can be scheduled to execute in parallel provided there are no data dependencies between them and there are sufficient resources available at the same time. In the DFG in Fig. 1, the node 1 is scheduled during the first time step. Some nodes have mobility in that they can have several potential execution instances as in the case of node 3 which can execute in time step 1 or 2 without affecting the schedule.

2.3 Binding

Each variable that carries values across cycles must be bound to a storage unit. In addition, several variables with nonoverlapping or mutually exclusive lifetimes can be bound to the same storage units. Storage and functional unit binding also influence connectivity binding such as a bus or a multiplexer. In the above allocation, node 1 may be bound to Adder 1, node 2 to Adder 2 and node 3 to Adder 3. The above sub-tasks can be performed in any order or simultaneously. The result of each sub-task influences the others. Ideally, high-level synthesis estimates the connectivity delay and area as early as possible so that later HLS steps can better optimize the design. An alternative approach is to specify the complete architecture during allocation so that initial floor planning results can be used during binding and scheduling.

2.4 RTL Generation

Once allocation, scheduling and binding decisions are made, the goal is to generate RTL architecture. All design decisions are applied and an RTL model of the synthesized design is generated. Overall flow of silicon compilation through HLS flow is shown in Fig 2.



(a) (b) (c)

Fig. 3 (a) Data Flow Graph (b) ASAP Schedule (c) ALAP Schedule

3. CLASSIFICATION OF HLS TECHNIQUES

Over past decades researchers have come up with various kinds of solutions to HLS problem. Well known HLS methodologies and approaches are categorized as shown in Fig 4.

3.1 Constructive Approaches

3.1.1 ASAP and ALAP Algorithm

The "As Soon as Possible" (ASAP) [1] scheduling starts with the nodes that have no predecessors in the DFG and assigns time steps in increasing order for the dependent nodes. The successor node can execute only after its predecessors has executed. ASAP algorithm gives the fastest schedule possible as shown in Fig 3(b). The "As Late as Possible" (ALAP) algorithm [1] works exactly as ASAP expect that it starts at the bottom of the DFG and proceeds upwards as shown in Fig 3(c). ALAP gives the slowest possible schedule and takes the maximum number of control steps but does not reduce the number of functional units.

3.1.2 Force Directed Scheduling

The "Force Directed Scheduling" (FDS) [2] is a heuristic method which tries to reduce the total number of FUs used. The FDS algorithm achieves this by uniformly distributing the operations of the same type over the available control steps. The FDS algorithm does not always produce an optimal solution; this can be alternated [3] by pruning one control step from its mobility range and postponing the decision. Hence it is classified under constructive approaches.

3.1.3 List-Based Scheduling

List based scheduling [4] is a generalization of the ASAP algorithm with the inclusion of resource constraints. These maintain a priority list of nodes called ready nodes whose predecessors have already been scheduled. Scheduling a node in a control step makes its successor nodes as ready nodes which could be added in priority list.

3.1.4 Static List Scheduling

This algorithm [5] first creates a single static large list before scheduling preventing dynamic growth. It uses ASAP and ALAP to obtain least (LCS) and the greatest possible control step assignments (GCS) for each operation. All operations are sorted in ascending order with GCS and descending order with LCS to form priority list. Operations are scheduled sequentially with highest priority, when the limit for the number of resources is reached rest of the operations is deferred to later control steps. Scheduling mainly depends upon the type of operation. For Ex: A low priority addition operation can be even scheduled in spite of high priority multiply operation if adder unit is available and if its predecessors are already scheduled.

3.1.5 Iterative Transformational Approaches

An iterative transformational approach usually starts with default schedules and applies semantic preserving transformations to improve the initial schedules. The Yorktown Silicon Compiler system described in [6] starts with a maximally parallel schedule which is then serialized to satisfy the resource constraints. The CAMAD design system described in [7] starts with a maximally serial schedule and attempts to parallelize it to reduce the schedule length. These systems attempt to escape local minima in the design space by incorporating suitable hill-climbing heuristics. However, the quality of solutions found by transformational techniques depends, to a large extent, on the variety and effectiveness of the transformations used, as well as the heuristics used to select between applicable transformations.

3.2 Probabilistic Techniques

3.2.1 Simulated Annealing

Schedules are represented as a two dimensional table of control steps [8] versus available FUs, while scheduling is viewed as placement problem. Beginning with an initial schedule it iteratively modifies the table by displacing an entry and

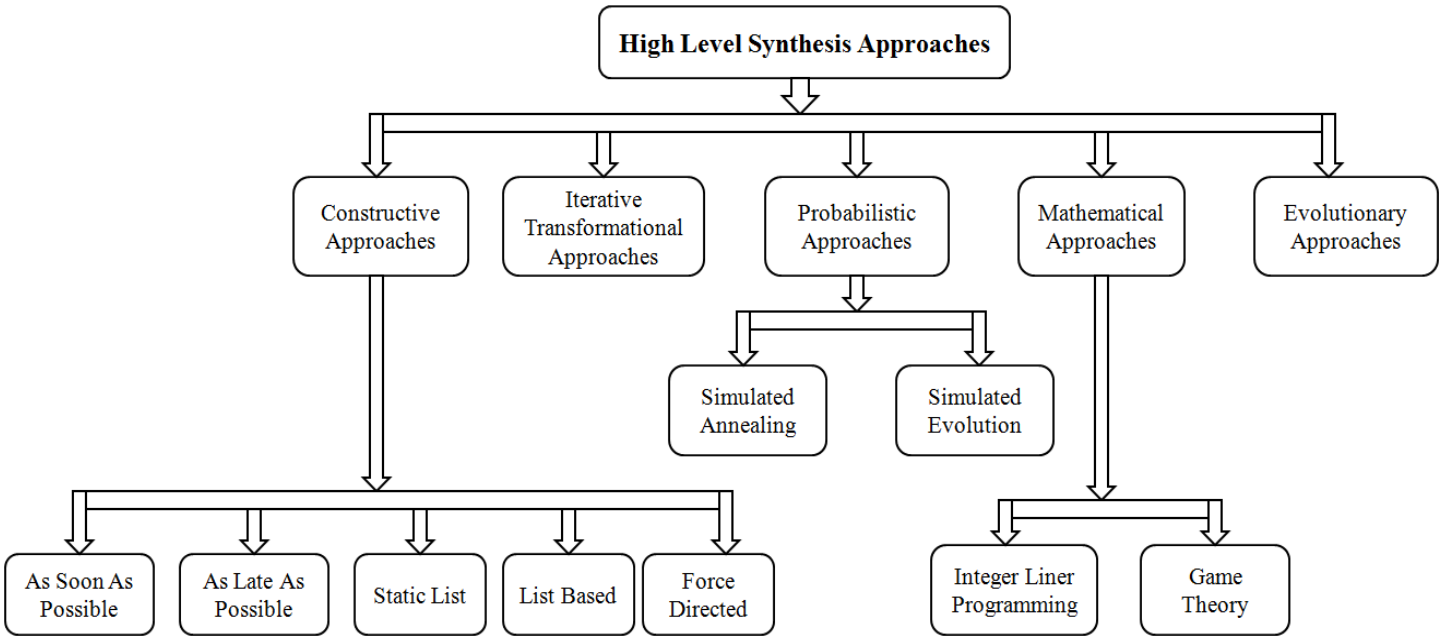


Fig. 4 Classification of High-Level Synthesis Methodologies

determining the cost of the displacement. Modification is accepted with a probability that resulting schedule is not better, so that it is possible to search solution space by climbing out of local minima in search of a global optimal solution.

3.2.2 Simulated Evolution

Simulated evolution approach was applied by Ly and Mowchenko [9] for simultaneous scheduling and resource allocation in high-level synthesis. Though these probabilistic methods produce good quality solutions, their run times are often large and increase rapidly with the size of the problem.

3.3 Mathematical Approaches

3.3.1 Integer Linear Programming

"Integer Liner Programming" (ILP) [10] tries to find an optimal schedule using a branch-and-bound search algorithm. It also uses backtracking mechanisms to change the decisions taken earlier if necessary. Complexity increases rapidly with the number of control steps which increases the execution time rapidly. Thus in practice the ILP approach can be applied to very small problems.

3.3.2 Game Theory

A game-theoretic approach for power optimization of a scheduled DFG is described in [11]. The functional units are modeled as bidders for the operations in the DFG with Power Consumption as the cost. The algorithm does not scale well for larger number of functional units since the complexity increases exponentially as the number of players (bidders).

3.4 Evolutionary Approaches

Genetic Algorithms (GAs) are potentially good candidates when large design space is to be explored since their behavior is similar to that of a designer. They produce alternative designs of different area, power and delay objectives. The architecture which meets the desired requirements can be obtained from the design space. Thus GAs provides a fully automated solution for HLS problems providing better performance and faster exploration. Several HLS system using GAs have been proposed, some of the techniques are as follows;

3.4.1 Integrated Genetic Algorithm Approach to HLS

An Integrated approach to the scheduling, allocation and binding problems based on hierarchical application of two genetic algorithms was used in [19]. The first genetic algorithm (GA1) performs allocation from library and tries not to exceed the area constraints specified by user. The second genetic algorithm (GA2) performs scheduling and binding using the allocation provided by GA1. If a schedule which satisfies the timing constraint is found the process terminates, otherwise GA2 returns the length of the best schedule to GA1 and whole concurrent process runs till a schedule meeting user defined area and timing constraints are met. Chromosome representations are through Priority-based encoding. Here a single functional unit capable of performing all operations in the DFG in a single control step is assumed. Partially-Mapped Crossover (PMX) proposed by Goldberg has been used as crossover operator and "swap" mutation operator is used as mutation operator.

3.4.2 Problem- Space Genetic Algorithm Approach

Problem-space GA (PSGA) [12] was proposed for design space exploration of datapaths. A problem-specific chromosome representation was used, in each operation is assigned a priority known as work remaining. This assignment is based on the length of the longest path from the node to the output. Each chromosome is mapped to a valid schedule using a problem-specific heuristic, called the work remaining heuristic. This heuristic is used to decode each chromosome into a schedule and a module allocation. Simple crossover and mutation operators are used to generate valid chromosomes. The main disadvantage of this technique is that the number of unique chromosomes that map to the same solution is large due to their chromosome representation. This makes the design space exploration vast even for medium sized problems.

3.4.3 GA approach to Allocation and Binding

A GA approach to allocation and binding for the high-level synthesis of data paths (GABIND) is proposed in [13]. An unconventional crossover mechanism which relies on the force of the directed datapath binding completion algorithm was used. The main feature of the proposed system was the use of bus-based interconnection scheme and the use of multiport memories. The system does not handle scheduling of operations as a scheduled data-flow graph is assumed as its input

3.4.4 GA based Simultaneous Scheduling and Storage Optimization

A GA-based high-level synthesis system using binary encoding scheme for chromosomes was proposed in [14]. This mainly opposes problem-specific representations of chromosome. Binary encoding schemes were used to store information on the control step assigned to each operation in the data-flow graph, and the functional module assigned to the operation. The main and inherent disadvantage of using a binary encoding scheme in chromosome representation evidently exponentially increases the size of the chromosome with the size of the problem. This increases the size of the design search space which leads to large run times for real problems.

3.4.5 GA in SoC level and Microcode level

Recently GAs have been applied at SoC level [15] and at microcode level of instruction set processors [16]. In these GA approaches populations of solutions are iteratively improved through the application of genetic operators to the individuals. These systems mainly differ in their chromosome representations and the genetic operators used to search the solution space.

3.4.6 GA for Design Space Exploration

Recently, GAs has been applied to design space exploration using a priority-based approach [17]. Multi-chromosome encoding scheme is used to represent the chromosome. In this scheme a chromosome has a node scheduling priority field and a module allocation field. The structure of the chromosome is such that simultaneous scheduling of a DFG and FU allocation can be carried out. The DFG nodes are scheduled using a list scheduling heuristic. The nodes are taken up for scheduling in the order in which they appear in the chromosome known as node priority field. The module constraint is described in resource allocation field where the number of FU (adders and multipliers etc) is specified. Resource binding and the interconnection elements are not taken into account in an accurate way. Moreover, the main disadvantage of this method is that they work with just one objective or with linear combination of weighted objectives. The GA uses a weighted cost function incorporating both area and delay. But the weighted sum approach suffers from the drawback that in a sufficiently nonlinear problem, it is likely that the optimal solutions resulting from a uniformly spaced set of weight vectors may not result in a uniformly spaced set of Pareto-optimal solutions.

3.4.7 Multi-Objective GA for Design Space Exploration

Ferrandi et al [18] have proposed an approach based on Multi-objective GA using the algorithm NSGA II. The authors have used two different encoding schemes. The priority based scheme arranges nodes in the DFG in the order in which they have to be scheduled by a list scheduler. Whereas the binding based scheme incorporates binding information pertaining to each DFG node. Use of Non Dominated Sorting Genetic algorithm II ensures the solutions to be in Pareto optimal front and uniformly spaced solutions in search space. Area and performance are estimated using a model derived from actual evaluated solutions by applying regressive techniques. Power was not included in the fitness evaluation.

4. FUTURE DIRECTIONS

As seen there is growing need for raising the level of abstraction in hardware design to simplify the design process. Many HLS approaches have been proposed to solve this problem. Most of the approaches have been surveyed and presented in this paper. Combining the strengths of genetic algorithms and high-level synthesis approaches have proved to produce better results and able to explore the design space efficiently for better architectures. As VLSI objectives like area, delay and power are mutually conflicting in nature, while optimizing one objective the tradeoff with other objective should be taken into account. Thus efficient Multi-objective algorithms need to be developed to address these issues.

5. REFERENCES

[1] De Micheli, Synthesis and Optimization of Digital Circuits. New York: McGraw-Hill, 1994.

[2] Pierre G. Paulin and John P. Knight, “Force Directed Scheduling for the behavioral synthesis of ASIC’s, ”IEEE Trans. Computer Aided Design, Vol.8, pp 661-679, June 1989.

[3] W.F.J. Verhaegh, E.H.L Aarts, J.H.M. Korst and P.E.R Lippens, “Improved Force Directed Scheduling,” Proc. Of European Design Automation Conf., pp.430-435, 1991.

[4] S. Davidson et. al., “Some experiments in local microcode compaction for horizontal machines,” IEEE Trans. On Computer, pp. 460-477, July 1981.

[5] R. Jain, A. Mujumdar, A. Sharma and H. Wang, “Emprical evaluation of some high-level synthesis scheduling heuristics,” Proc. of 28th DAC, pp. 210-215, 1991.

[6] R. K. Brayton, R. Camposano, G. De Micheli, R. Otten, and J. van Eijndhoven, "The Yorktown silicon compiler system," in Silicon Compilation, D. D. Gajski, Ed. Reading, MA: Addison-Wesley, 1988, pp. 204–310.

[7] Z. Peng, "Synthesis of VLSI systems with the CAMAD design aid," in Proc. 23rd ACM/IEEE Design Automation Conf., 1986, pp. 278-284.

[8] S. Devadas and A.R. Newton, "Algorithm for allocation in datapath synthesis," IEEE Trans. on CAD of Interg. Cir. And Systems, Vol. 8, pp. 768-781, July 1989.

[9] T. A. Ly and J. T. Mowchenko, "Applying simulated evolution to high level synthesis," IEEE Trans. Comput.-Aided Des., vol. 12, no. 2, pp. 389-409, Feb. 1993.

[10] J.Lee, Y.Hsu, and Y.Lin, "A new Integer Linear Programming Formulation for the Scheduling Problem in Data-Path Synthesis," Proc of the Int. conf. on Computer-Aided Design, pp. 20-23, 1989.

[11] Ashok. K. Murugavel and Nagarajan Ranganathan, "A Game Theoretic Approach For Power Optimization During Behavioral Synthesis," IEEE Transactions on VLSI, Vol 11, No. 6, Dec 2003.

[12] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker, "Datapath synthesis using a problem-space genetic algorithm," in IEEE Trans, Comput.-Aided Des., vol. 14, 1995, pp.934-944.

[13] C. Mandal, P. P. Chakrabarti, and S. Ghose, "GABIND: A GA approach to allocation and binding for the high-level synthesis of data paths," IEEE Trans. Very Large-Scale Integrated Circuits, vol. 8, no. 5, pp. 747-750, Oct. 2000.

[14] E. Torbey and J. Knight, "Performing scheduling and storage optimization simultaneously using genetic algorithms," in Proc. IEEE Midwest Symp. Circuits Systems, 1998, pp. 284–287.

[15] G. Ascia, V. Catania, and M. Palesi, "A GA-based design space exploration framework for parameterized system-on-a-chip platform, "IEEE Trans. Evol. Comput., vol. 8, no. 4, pp. 329–346, Aug. 2004

[16] D. Jackson, "Evolution of processor microcode," IEEE Trans. Evol Comput., vol. 9, no. 1, pp. 44–54, Feb. 2005.

[17] V. Krishnan and S. Katkoori, "A genetic algorithm for the design space exploration of datapaths during high-level synthesis," IEEE Trans. Evolutionary Computation, 10(3): 213–229, 2006.

[18] Fabrizio Ferrandi, Pier Luca Lanzi, Daniele Loiacono, Christian Pilato, Donatella Sciuto, "A Multi-Objective Genetic Algorithm for Design Space Exploration in High-Level Synthesis," IEEE Computer Society Annual Symposium on VLSI, pp 417-422, 2008.

[19] G. Grewal, M.O'Cleirigh and M.Wineberg, "An Evolutionary Approach to Behavioral-Level Synthesis", Proc. of Evolutionary Computation, Vol 1, 264-272, 2003.