Survey on Optimization Techniques in High Level Synthesis

B. Saravanakumaran[1] and M. Joseph[2]

[1] E.G.S. Pillay Engineering College
Nagore, Nagapattinam - 611002, India
saravanakumaranbalu@yahoo.com
[2] Mother Terasa College of Engineering and Technology
Illuppur, Pudukkottai - 622102, India
mjoseph_mich@yahoo.com

Abstract. This paper provides a detailed survey of optimization techniques available in high level synthesis. This survey contemplates on two parts. The first part deals with the applicability of optimization techniques available in high level language compiler into high level synthesis. The second part address the topics such as Area optimization, Resource optimization, Power optimization and Optimization issues pertaining to the notions value-grouping, value to register assignment, Transfer to wire assignment and wire to FU port assignment.

Keywords: High Level Synthesis, Very Large Scale Integrated Circuits, Functional Units, HDL Compiler.

1 Introduction

High level synthesis is generally used in Integrated Circuit design. The main goal of the HLS is to optimize silicon area, power and time. The use of a Hardware Description Language (HDL) is to understand the optimization techniques at the HDL level of abstraction for the accomplishment of better design, low power and reduced area. The use of a Hardware Description Language is to examine the optimization techniques at the HDL level of abstraction to achieve possible changes in design realization of low power and area reduction. In Very Large Scale Integrated Circuit (VLSI) design, it is required to achieve higher level abstraction. The higher level abstraction of the circuit is a basic requirement due to

- Higher Complexity
- Shrinking device size and
- Shorter time to market

Due to low power requirements in many portable applications such as mobile phones as well as packaging cost consideration, low power design is imperative. To design the chip designer automates on higher levels of abstraction where functionality is easier to understand and tradeoff is more influenced. There are

N. Meghanathan et al. (Eds.): CCSIT 2012, Part II, LNICST 85, pp. 11–21, 2012.

several advantages to automate part of all of the design process and moving automation to higher levels. First, automation assures a much shorter design cycle. Second, it allows for more exploration of different design styles since different designs can be generated and evaluated quickly. Finally, if synthesis algorithms are well understood, design automation tools may-out perform average human designers in meeting most design constraints and requirements.

1.1 High-Level Synthesis

Synthesis is a translation process from a behavioral description into a structural description, similar to the compilation of a high level language program in C or Pascal into an assembly program. Each component in the structural description is in turn defined by its own (low level) behavioral description. Synthesis, sometimes called design refinement, adds an additional level of detail that provides information needed for the next level of synthesis or for manufacturing of the design. High Level Synthesis is also defined as the process of mapping a behavioral description at the algorithmic level to a structural description in terms of functional units, memory elements and interconnections (e.g. multiplexers and buses).The functional units normally implement one or more elementary operations like addition, multiplication, etc. Another important point in high level synthesis is the formal description of the algorithm to be mapped. The study of the HLS leads to the manifestation of shorter design cycle, minimization of errors, exposure to identify the design space and the capability of the design process in self documenting.

1.2 Hardware Description Language

The job of the hardware in modeling, design and documentation of digital systems is wholly governed by the hardware description language. The hierarchical representation of functional and wiring details of the systems receive a complete and compact format only from these languages.

1.3 HDL Compiler

The tool which performs high level synthesis is called Hardware Description Language complier. The main objective is to optimize the code, so that optimized hardware is obtained after synthesis. The optimization factors are different since the aim of synthesis is to generate optimal hardware circuit. They are performance(speed), area and power. As in HLL compiler, optimization techniques are applied onto the intermediate representation but improve the speed, minimize the silicon area and power.

The compiler driven optimization techniques are trying to optimize the logic of the digital system and can be better. The parameters on which the systems optimally can be assessed are testability and design time. The few optimization techniques are Dead code elimination, constant propagation, common sub

expression elimination, inline expansion of procedures and loop unrolling. The objective of HDL compiler is to obtain the optimized hardware, which runs faster, occupies the silicon area and consumes less power. Other parameters on which the systems can be accessed are testability and design time.

2 Related Surveys

Youn-Long Lin[8] has examined the latest developments in high level synthesis technology design. This paper provides the detailed survey for basic techniques for various optimization techniques in High level synthesis. Techniques have been proposed in the past few years for various sub tasks of high level synthesis are surveyed. The prime goal of the HLS for testability, low power and reliability were detailed. M.Joseph, Narasimha B.Bhat and K. Chandrasekaran[2] survey paper presents Hardware description language compiler optimization techniques and applicability of high level language compiler optimization techniques to Hardware Description Language Compilers was exhaustively presented. Esther Andres and Maria C.Molina[19] have made an attempt to optimize the area in high level synthesis using combined integer and floating point circuits. Dake Liu and Christer Svensson[20] have investigated the consumption estimation techniques in CMOS VLSI chips. Jason Coog et.al.,[21] have studied the various resource optimization techniques available in high level synthesis.

3 HDL Compiler Optimization Techniques

Program analysis techniques like Data Flow Analysis (DFA), Control Flow Analysis (CFA) and Dependence Analysis (DA) are essential for applying optimization techniques. The major optimization techniques available in HDL compiler are

3.1 Elementary Optimization Techniques

Constant folding: It is an optimization technique, which replaces the runtime computation by compile time computation and is generally done for constants. Eg.:pi=22/7;the value of pi can be replaced by the evaluated value 3.14. This avoids the division operation and the related hardware. Constant folding reduces the number of computations, which in turn reduces the synthesized hardware and thus area and power are reduced.

3.2 Redundancy Elimination

This category of optimization techniques avoids re-computation of the computations which have been done already. In this category of optimization necessitate the Data Flow Analysis.

Common Sub-expression Elimination (CSE): CSE allows no re-computation of an expression which is evaluated already. The common sub expression should not be redefined along the path of its use. This technique can be applied locally or globally. In general the goal of CSE can be defined as follows.

- Identify multiple patterns in the coefficient set.
- Remove these patterns and calculate them only once.

The problem to solve is how to identify the proper patterns for elimination so the optimization impact can be maximal. Our algorithm is based on a combination of an exhaustive search technique with a steepest descent (or greedy) approach in order to select the proper patterns for elimination [17].

Eg.:
a=b+c;
d=e+b+c;

The above expression is an un-optimized expression.

t=b+c;
a=t;
d=e+t;

The above expression is optimized expression.

3.3 Loop Optimization

Strength Reduction: It is an optimization technique, which replaces the costly operator by a cheap operator. Consider the statement; c=2*a this can be replaced by c=a+a. Here the multiplication operator is replaced by addition operator. These optimizations are done inside the loop, so it leads to greater improvement. Strength reduction minimizes the area and power.

3.4 Code Scheduling

This technique discusses about the scheduling of instructions, which will lead to improvement in the speed. Scheduling here refers to ordering or reordering the instructions with respect to clock cycles. The optimization in the scheduling mostly improve the performance i.e. either reduction in the number of cycles needed or reduction in the clock period of particular logic device.

Instruction Scheduling: This optimization technique schedules number of independent instructions in a same cycle. In example, the instructions are scheduled in three cycles in the un-optimized case. The first and second statements are data independent and can be scheduled in a single cycle. The third statement is a data dependant on both first and second. So it should be scheduled after both [15].

a=b+c;
d=e+f;
g=a+d;

The above expression is an un-optimized.

a=b+c; d=e+f;
g=a+b;

The above expression is an optimized.

Loop fusion: This technique allows computations in adjacent loops to be combined into a single loop. If possible two loops can also be integrated together. This reduces the total number of cycles required. This will increase the clock period of that logic.

Resource sharing: The resources like adder, multipliers and other functional units can be shared. This avoids instantiating hardware resources for every operator. The resources can be shared for the operations, which are not scheduled in the same cycle. The resources must be compatible. For example adder is compatible with subtracter and relational operator. When the scheduled operations for a functional unit are completed then those functional units are free and can be shared by other operations. This optimization technique reduces the area and power. The above optimization techniques are already available in High level language compiler. Our present work is to try and apply the programmer level optimization techniques into HDL complier.

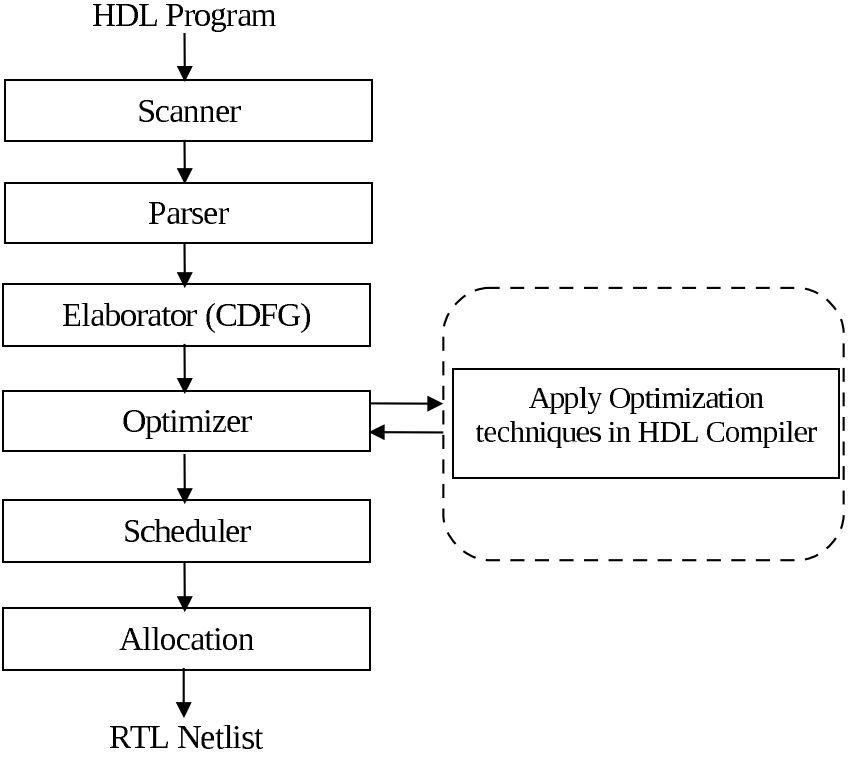

Fig. 1. HDL Compiler Optimization and Proposed Architecture design

3.5 Register Allocation

This is popularly used optimization techniques in HLL compiler for register assignment to the variables. Graph coloring is one such technique used for register allocation. In HDL compiler this problem can be understood in a different perspective. Here storage elements are flip flops and variables can be assigned to them. The assignment of variables to flip flops can be understood as a register allocation problem and accordingly techniques like graph coloring can be applied. The classification is made whether the optimization reduces or increases one of the three metrics: clock period, power consumption and area in the FPGA [14].

4 Recent Developments

Young Long Lin[8] has studied the High level Synthesis. The chief objectives of his investigation are Testability, Power efficiency and Reliability of High Level Synthesis. He has applied Scheduling, Allocation and Binding to achieve the transforming behaviour into structure. By virtue of ASAP (As Soon As Possible) operations are stored into a list according to their scheduling. By virtue of ALAP(As Late As Possible)operations define the latest step into which an operations can be scheduled optimization is realized by minimizing the number of self loops. Along this line Flottes[6] has extensively studied the register allocation. To manifest the testability Potkonjak[7] has proposed a two stage objective function for estimating the area and to reduce the fan-ins and fan-outs of the interconnection wires, capacitance. Rabaey[9] has invoked scheduling, Assignment and allocation techniques. Raghunathan and Jha[11]has applied data path allocation method to the circumstances where low power is prevalent. Musoli and Cortadelia[13] has applied scheduling and resource binding algorithm for reducing the activity of functional units by minimizing the input operations. Martin and Knight[23]attempted an investigation by lowering supply voltage and disabling the clock of idle computers. These findings are highly applicable in Digital Signal Processing design environment, embedded system design environment, Hardware design verification and to solve the software-hardware code design problem.

Hao Li Srinivas Katkori Zhipeng Lin[22] have established that HLS design flow improve the circuit performance once the placement phase is done. They have also established that once the placement is computed estimation of the circuit performance is done by the Xilinx. They proposed a method so that feed back is generated and given to the Automatic design installation. This has provided confidence that the mixture of the HLS tool and physical design that has excellent potential to enhance the performance of modern VLSI design.

Hao Li Srinivas Katkoori, Zhipeng Liu[22], have exposed HLS tool is able to iteratively enhance the system performance with the guidance information obtained from the physical design phase. According to their study estimation shares that the best synthesized design could satisfy the original system design objective. Testing is applied with regard to the predefined number of iterations. This revelation has reduced the running the efficiency.

Jason Coog, Bin Liu and Junjuan XG[21] have examined the coordinated Resource optimization in behavioural synthesis. This study is aimed to reduce resource usage in terms of power, performance and cost. In this study the separating synthesis process in transformed into a sequence of optimization steps.

Esther Andres, Maria C.Molina, Guilerno Botella[19] have proposed HLS algorithms associated with floating point operations. In this paper the authors propose the substitution of the floating point operation by their corresponding fixed point operations. Due to huge area requirement of the floating point operations the authors propose the optimization of the systems through the reuse of the internal modules to perform other operations.

5 Important Optimizations

A major part of the design process for an optimizer is to decide which optimization techniques are most important and which are less important. Typically, a few optimization techniques will provide a very high return in terms of improved execution speed while the remainder provides only marginal improvements. Given that compile-speed and compile space always have limitations, this means the best returns are yielded by focusing on a few, important optimizations.

In general, this investigation requires complicated analysis, some combination of inter procedural analysis, higher level Verilog analysis, or user assertion. While it is desirable to solve the general problem in the long run, in the short run, this information is often available in local cases by a simple analysis of the code. In any case, once the information is available, it is clear that the key global optimizations are constant propagation and dead code elimination. Given a local strategy for code generation that attempts no memory reuse, it should also be evident optimizations focused on recognizing and exploiting memory reuse is important. In particular, users tend to work with the same variables repeatedly in a section of code. Code generators rarely recognize this reuse; they load the locations which they need at the beginning of a unit and store everything changed at the end of a unit.

There are three ways of improving this reuse: 1.Eliminating Common Sub Expressions, 2.Changing variable allocations, 3.Doing good register allocation.

5.1 Area Optimization

Improvement in circuit performance is an important factor in area optimization. For this purpose conventional high level synthesis algorithms employ multi cycle operators to the cycle length. For the execution of one operation operators require several cycles, but at the same time the entire functional unit is not employed in any cycle. Also, the execution of operations in multi cycle operators is infeasible if the results should be available in a smaller number of cycles than the functional unit delay. This obliges to add new functional resources to the data path even if multi-cycle operators are idle when the execution of the operation begins.

Operations are executed in such a manner that functional units are allowed for their internal reuse, which eventually reduces the area of functional units. Since it facilitates the use of multi cycle operators for the calculation of narrower operations faster than the functional units, the possibilities of common hardware sharing get the expansion. At the end of the high level synthesis this technique is employed as an optimization phase. It can also optimize the circuits synthesized by any high level synthesis tool, consequently the area of the circuits synthesized by regular HLS algorithm using multi cycle operators gets reduced by the proposed optimization technique.

5.2 Resource Optimization

One of the most pertinent optimization objective is the abstract reducing resource usage in behavioral sysnthsis. Since it has its influence on power, performance and cost. Functional units, registers and multiplexers are the distinct components of the data path in a typical design. If a behavioral synthesis tool considers all kinds of tools simultaneously then it would optimize the overall resource usage.

Nevertheless the earlier work on behavioral synthesis has its own limitations in terms of the inability to consider all kinds of resources globally and the splitting of the synthesis process into a sequence of optimization steps which is void of a consistent optimization objective. Neilson[12] has examined a behavioral synthesis flow by allowing all types of components in the data path are modeled and optimized consistently. The key idea is to feed to the scheduler the intentions for sharing functional units and to register in favor of the global optimization goal such as total area, so that the scheduler could generate a schedule that makes the sharing intentions feasible.

Performed experiments reveal the fact that minimizing functional unit requirements is scheduling and employing the east number of functional units and registers in binding when compared with our solution promotes a 24 percent reduction on average. Neilson[12] has presented a behavioural synthesis flow, in which all resources in the data path are consistently optimized throughout the whole synthesis flow. This is achieved by guiding the scheduler using sharing intentions in favor of efficient resource usage. Experimental results show significant reduction compared to previous approaches in total area for a set of benchmarks. Our idea can also be used for optimizations of other objectives (e.g., power), by consistent consideration throughout different steps in the behavioral synthesis flow.

5.3 Power Optimization

To address the challenge to reduce power, the semiconductor industry has adopted a multifaceted approach, attacking the problem on four fronts [1]:

- Reducing the chip and package capacitance
- Scaling the supply voltage
- Employing better design techniques
- Using power management strategies

6 Issues in Compiler Driven Optimization Techniques

System Power: When a chip dissipates too much power, it will either become too hot and cease working or will need extra cooling. Besides, there is a special category of applications, viz. portable equipment powered by batteries, for which low power consumption is of primary importance. Here again there are trade-offs: designing for low power may e.g. lead to an increase in the chip area.

Design Time: The design of an integrated circuit is almost never a goal on its own: it is an economical activity. So a chip satisfying the specifications should be available as soon as possible. The design costs are an important factor, especially when only a small number of chips need to be manufactured. Of course, good CAD tools help to shorten the design time considerably as does the use of semicustom design.

Testability: As a significant percentage of the chips fabricated is expected to be defective, all of them have to be tested before used in a product. It is important that a chip is easily testable as testing equipment is expensive. This asks for the minimization of the time spent to test a single chip. Often, increasing the testability of a chip an increase in its area.

7 Optimization Issues

In High Level Synthesis the scheduling and assignment tasks are interrelated. For an optimal design they should be solved simultaneously. Most systems first solve the scheduling problem and then try to find a good assignment given a certain schedule. The assignment problem itself consists of several sub problems.

Operation-to-FU assignment: This is the problem of mapping a computation to functional unit of an appropriate type. Value grouping: This is the problem of partitioning all storage values in such a way that subset does not contain values that are read or written simultaneously. Then each subset can be realized as a register bank. In the case of multiport memories, the conditions for grouping should be adapted accordingly.

Value-to-register assignment: This is the problem of assigning memory location to storage values in the same group. Values with non overlapping life times can share the same location. The life time of a storage value is the time interval starting at the instant that it is created, and ending the moment that it no longer is required.

Transfer-to-wire assignment: A transfer is the actual transport of data from one hardware unit to another. In a bus-based architecture, one has the choice of which bus to write. The choice affects the number of three-state drivers connected to the unit from which the transfer originates and the type of multiplexers connected to the receiving the transfer.

Wire to FU-port assignment: In the case of commutative operations, one can choose one of the two equivalent input ports to feed the data to the functional unit. More in general, the problem exists when a functional unit has ports that are functionally equivalent.

8 Summary

This paper is on the basis of the detailed survey on the optimization techniques available in high level synthesis. This paper exposes the possible application of optimization techniques aimed to transform high level language compiler into high level synthesis. The later part of the paper discusses in detail the notions such as area optimization, resource optimization, power optimization and optimization issues which eventually lead to the concept of value grouping, value to register assignment, transfer to wire assignment and port assignment.

Acknowledgments. We thank to the anonymous reviewers for their numerous insightful and constructive comments.

References

1. Lin, C.-C., Yoon, D.-H.: New Efficient High Level Synthesis Methodology for Low Power Design. In: International Conference on New Trends in Information and Service Science (2009)
2. Joseph, M., Bhat, N.B., Chandra Sekaran, K.: Right inference of Hardware in High-Level Synthesis. In: International Conference on Information Processing, ICIP 2007, Bangalore, India (2007)
3. Zhang, J., Zhang, Z., Zhou, S., et al.: Bit-level optimization for high level synthesis and FGPA-based acceleration. In: Proceedings of FPGA 2010, Monterey, USA (2010)
4. Molina, M.C., Ruiz Sautra, R., Mendias, J.M., Hermida, R.: Area Optimization of multi-cycle operators in high level synthesis. In: Dte Conference Proceedings (2007)
5. McFarland, M.C., Parker, A.C., Campasona, R.: Tutorial on High-Level Synthesis. In: 25th ACM/IEEE Design Automation Conference (1998)
6. Flottes, M.L., Hammad, D., Rouzeyre, B.: High Level Synthesis for easy Testability. In: Proceedings of the European Design and Test Conference, Paris, France, pp. 198–206 (1995)
7. Potkonjak, M., Rabaey, J.: Optimizing Resource Utilization using Transformation. IEEE Trans. Computer Aided Design Integrated Circuits Systems 13(3), 227–292 (1994)
8. Lin, Y.L.: Recent Developments in High-Level Synthesis. ACM Transactions on Design Automation of Electronic Systems 2(1), 2–21 (1997)
9. Rabaey, J., Guerra, L., Mehra, R.: Design guidance in the power dimension. In: International Conference on Acoustics, Speech and Signal Processing, pp. 2837–2840 (1995)
10. Roy, S., Banerjee, P.: An Algorithm for Converting Floating-Point Computations to Fixed-Point in MATLAB based FPGA design. In: Design Automation Conference - DAC 2004, San Diego, California, pp. 484–487 (2004)
11. Raghunathan, A., Jha, N.K.: Behavioral Synthesis for low power. In: Proceedings of the International Conference on Computer Design (ICCD), pp. 318–322 (1994)
12. Neilson, S.G.: Behavioral synthesis of asynchronous circuits. PhD dessertation Technical University of Denmark, Department of Informatics and Mathematics modelling (2005)

13. Musoli, E., Cortadella, J.: Scheduling and Resource binding for low power. In: Proceedings of the Eighth Symposium on System Synthesis, pp. 104–109 (1995)
14. Ozer, E., Nisbet, A., Gregg, D.: Classification of Compiler Optimizations for High Performance. Small Area and Low Power in FPGAs (2003)
15. Joseph, M., Bhat, N.B., Chandra Sekaran, K.: Technology driven High-Level Synthesis. In: International Conference on Advanced Computing and Communication-ADCOM 2007. IEEE, Indian Institute of Technology, Guwahati, India (2007)
16. Taylor, S., Edwards, D., Plana, L.: Automatic compilation of data driven circuits. In: 14th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 3–14. IEEE (2008)
17. Pasko, P., Schaumont, P., Derudder, V., Vernalde, S., Durackova, D.: A New algorithm for Elimination of Common Sub Expressions. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 18(1) (1999)
18. Muchnick, S.S.: Advanced Compiler Design Implementations. Harcourt Asia PTE Ltd. (1997)
19. Andres, E., Molina, M.C.: Area Optimization of Combined Integer and Floating Point Circuits in High Level Synthesis. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 8(1) (2008)
20. Liu, D., Svensson, C.: Power Consumption Estimation in CMOS VLSI Chips. IEEE Journal of Solid State Circuits 29(6) (1994)
21. Coog, J., Liu, B., Xu, J., et al.: Coordinated Resource Optimization in Behavioural Synthesis 20(2) (2009)
22. Katkoori, H.L., Liu, S.Z.: Feedback driven High Level Synthesis for performance optimization. In: ASICON 2005, 6th International Conference ASIC Proceedings, pp. 961–964 (2006)
23. Martin, R.S., Knight, J.P.: Power-profiler: Optimizing ASICs power consumption at the behavioral level. In: Proceedings of the Design Automation Conference (DAC), San Francisco, CA, p. 4247 (1995)
24. Free Floating-Point Madness, `http://www.hmc.edu/chips`
25. Electronic Design Interchange Format, `http://www.edif.org`
26. FPGA, CPLD, and EPP Solutions, `http://www.xilinx.com`
27. Icarus Verilog Simulation and Synthesis Tool, `http://www.icraus.com`