Survey of HDL Compiler Optimization Techniques

M.Joseph\textsuperscript{a} \textsuperscript{*}, Narasimha B.Bhat\textsuperscript{b} and K.Chandra Sekaran\textsuperscript{a}

\textsuperscript{a}Department of Computer Engineering, National Institute of Technology, Srinivas Nagar (Post), Mangalore, Karnataka, India - 575025.

\textsuperscript{b}Manipal Dot Net P. Ltd, 37. Ananth Nagar, Manipal, Karnataka, India - 576104,

This paper presents a detailed survey of Hardware Description Language (HDL) compiler optimization techniques available in the research literature. Applicability of High Level Language compiler optimization techniques to Hardware Description Language compilers is exhaustively presented.

1. INTRODUCTION

The optimal design of hardware with low power, faster performance and less chip area is the prime requirement for the digital systems. Due to low power requirements in many portable applications such as mobile phones, as well as packaging cost consideration, low power design is imperative [16]. Low power implies better cooling (avoiding high system cost of fans), portability (extended battery life), reliability (reduced electron migration), lower cost (power is expensive) and friendly environment (power has negative impact). Technical feasibility of high performance computation is due to heat extraction and is determined by total number of control steps (clock cycles) a system takes, which is the clock period of the slowest logic stage in the system. Minimization of chip area reduces the amount of silicon used and also increases yield since the causes of failure like crystal defects, defects in the masks, defects due to contact with dust particles, etc are less likely to affect a chip when area is smaller. The use of an Hardware Description Language (HDL) based design flow therefore becomes a de facto standard for any digital system designer, and an understanding of optimization techniques at the HDL level of abstraction helps write better design descriptions, eventually leading to higher performance, lower area and lower power designs.

1.1. HDL compiler

The HDL Compiler takes the code in a HDL and converts into RTL structures. The front end consists of scanner, parser and intermediate code generator and the usual functions are performed by these. But the intermediate representation is Control/Data Flow Graph (CDFG), which is a variant of the syntax tree representation along with the control information. The back end consists of the optimizer and hardware generation circuits. The optimizer applies compiler optimization techniques on CDFG to improve it. Then hardware generation functions are applied on the optimized CDFG to generate RTL structure, which are are scheduling, and allocation. Scheduling assigns operations to the control steps and allocation assigns operations to functional units like ALU, multiplexer and storage elements. The HDL compilation process is popularly called as High Level Synthesis (HLS) [6], [13], [21]. Popularly used HDLs are VHDL, Verilog, and SystemC, which are used to describe the hardware system.

1.2. HDL compiler optimization

In the classical sense, code optimization refers to the process of modifying the working code into more optimal code based on a particular goal. The goal may be reducing execution time or reducing memory space. But in case of HDL compilers, code optimization has a different perspec-
criterion to be optimized. The influence of performance optimization techniques on system energy have been studied by Kandemir et al [14]. The inferences are 1. Memory consumes more energy than the un-optimized codes. 2. Though Performance optimizations reduce the power consumption, they increase the data path energy consumption. 3. Increase in cache size leads to increase in data cache energy, but it reduces the energy by minimizing the number of accesses to main memory.

2. Testability
The chips fabricated have to be tested before being used in a product. It is important that a chip should be easily testable as testing equipment is expensive. This necessitates the minimization of the time spent to test a single chip. If the chip area is more, testability is improved [18]. So optimization of area may have negative effect on the testability.

5. OPEN ISSUES

1. Semantic gap
The hardware description languages adhere to a simple, sequential programming style, which mimics the HLL programming model. They are not capable of expressing the synchronous, concurrent processing nature of the hardware circuits. This gap between the HDLs and hardware circuitry is called semantic gap. This problem leads to sub optimal design of digital systems [5].

2. Order of optimization
The order in which the optimization techniques to be applied is not clear because application of one optimization will open avenues for other optimizations. E.g: constant propagation gives scope for constant folding [8]. Some optimizations are interactive, so the user has to decide the order in which order optimizations to be applied and should be aware of the details [18]. The order for applying the optimization techniques for HLL compilers is discussed by Steven S. Muchnick [19] and the applicability of that order can be studied.

3. Design time
A chip satisfying the specifications should be designed and made available as soon as possible. A good CAD tool should help to reduce the runtime (design time) considerably [18]. This is actually a different issue; however it has implication in the optimization issues.

6. CONCLUSION

This paper presents HDL compiler optimization techniques comprehensively and also some open issues. It does not address the implementation issues of these techniques. Influence of optimization techniques on a system power perspective is only done and addressed in the paper and influence on other parameters can be further extended. VLSI design as a whole can be thought as an optimization problem but this paper presents only compiler transformations not the other issues like gap between the CAD tools and target technologies, which is a major cause for suboptimal hardware generation.

REFERENCES

5. Greg Snider, Barry Shackleford and Richard J. Carter, Attacking the Semantic Gap between Application programming Languages
and Configurable Hardware (FPGA 2001),
6. Daniel D. Gajski, Loganath Ramachandran ,
Introduction to High Level Synthesis, IEEE Design and Test of Computers, 1994
7. Haifeng Zhou, Zhenghui Lin and Wei Cao,
Research on VHDL RTL Synthesis system,
Proceedings of the First International Workshop on Electronic Design, Test and Applications (DELTAS02), 2002
9. Jinhhwan Jeon and Kiyoung and Choi,
Loop pipelining in Hardware Software partitioning,
22. www.xilinx.com
23. www.geda.seul.org
24. www.icros.com