EE382m: Term Projects

Adnan Aziz
Electrical and Computer Engineering
The University of Texas at Austin

March 26, 2007

Introduction

The purpose of these projects is threefold:

1. It is worth 30% of your grade (but this should be the least important item)
2. working on this project should be training on how to go about approaching an original research problem
3. the project should yield results—
   - experimental findings on the efficacy of proposed algorithms, and/or
   - new techniques for design synthesis

Projects can be done individually, or in pairs.

Timeline

I do not want a student going off on a tangent, only to learn at the end of the semester that this happened. On the other hand, I don't want to stifle creativity by monitoring things too closely. Most of all, I don't want rush jobs, where everything is crammed into a few days at the end of the semester. You need time to think about these projects.

Project selection: You should make a decision as to the projects you are interested in working on by **Friday, April 6.**

It's fine for more than one person to select the same project. I will also allow two person groups to work on a project, with the caveat that I will expect significantly higher deliverables.

Progress report: To ensure progress is made continuously, I would like you to write a two-page progress report on the status of your project, and turn it in on Monday, April 23.

Final report: A 10 to 20 page document (in the format of a conference paper) describing the project should be turned in by 5:00pm Friday, May 4. There will be no penalty for submissions that turned in by Friday, May 11.

Evaluation

Your grade on the project will be based on a number of factors, particularly the originality and quality of the work. Other considerations include clarity (both the written report and the presentation), and attention to detail.

Suggested Projects

Below, I give sketches of topics that I would like to see work done on.[1]

1. **Synthesizing a small microprocessor**

   The goal of this project is to design and experiment with a SIS standard cell library that uses reasonable values for loading capacitances and drive strengths.

   Rather than trying to figure out the exact delays for a 90 nanometer TSMC technology, I would suggest that you express all the delays in terms of some nominal unit values (i.e., the drive strength and the capacitances of a minimum sized device). I would expect that a library consisting of INV, 2NAND, 3NAND, 4NAND, 2NOR, 3NOR, 4NOR, AOI32, AOI42, MUX21, MUX41 gates each in relative drive strengths of $1 : e : e^2 : e^3$ together with a simple latch should be fine.

   Experiment with your library using SIS to optimize a reasonably large design (say a 10000 gate processor) and report your results.

2. **Building minimum height BDDs**

   The example $f = c \cdot a + c' \cdot b$ shows that a function can be evaluated by examining less than all the variables it depends on. (For this example, check the value of c, then check a or b appropriately.)

   The goal of this project is to devise an algorithm for determining a variable ordering under which the BDD height, as measured by the number of nodes on the longest path from root to terminals, is minimum.

   A natural generalization, which you should explore, involves assigning a "cost" to each variable, in which case the goal is to minimize the cost of the maximum path. If you are more ambitious, you can try solving this problem when the inputs are generated probabilistically—your goal then would be to select an ordering which minimizes the expected cost of computing the result.

   I expect you should be able to devise an algorithm using Dynamic Programming. Chapter 15, Section 5 of the Algorithms text by Cormen, Leiserson, Rivest and Stein (CLRS)

[1] You are welcome to suggest your own project; however it must meet with my approval.

discusses this problem for binary search trees, and you should get some ideas from their analysis, as well as the references they cite. See [3] for an example of the use of DP for BDD minimization, and also the papers [2, 1] for computing orderings. (Your algorithm may have extremely high complexity, but exponential is still acceptable for functions of a few variables.)

3. **SAT**

   In recent years, there has been a great deal of interest in heuristic algorithms for checking whether a logic formula in conjunctive-normal form (CNF) is satisfiable. (This problem is exactly equivalent to checking if a cover represents tautology.) The state-of-the-art in SAT solving is the miniSAT tool, which is remarkably small (1200 lines, including parser). The goal of this project would be to think of ways in which to enhance miniSAT, and/or look for new applications for SAT (many constrained optimization problems such as routing, placement, etc. can be cast as SAT problems).

   Chapter 34 from CLRS talks about how problems can be mapped to SAT.

4. **Tarski**

   Static verification consists of checking a design's correctness without applying specific test cases; instead all possible legal inputs are considered. This approach has in the past applied to gate-level designs, since Boolean analysis techniques such as SAT and BDDs can be directly applied.

   Operating at a higher level makes automatic verification significantly easier for a number of reasons. One of my graduate students, Hari Mony, `harimony@us.ibm.com`, has been working on a tool for expressing designs at a high-level of abstraction.

   The goal of this project is to add synthesis capabilities to Tarski, that is, the ability to write a Tarski model out as a Boolean netlist, either Verilog or `blif` which can then be synthesized using DC or SIS. You would be responsible for designing a small unit (e.g., a pipelined DLX) in Tarski, and seeing the quality of results of the synthesized netlist compared to direct design and synthesis in Verilog.

   You can read about the commercial Bluespec tool that is based on research done at MIT to get some ideas of the tradeoffs involved. System-C also is similar in spirint.

5. **Case studies in synthesis**

   The goal of this project is to design a nontrivial unit (e.g., a small processor, an Ethernet controller, an MPEG decoder, an FFT, etc.), and see how much synthesis can improve on the design. I am particularly interested in comparing SIS with DC, specifically in terms of performance optimizations, e.g., retiming.

6. **c-slow transformation**

   Replacing each flop in a sequential design D by c flops in series results in a design D_c that is referred to as the c-slow version of D. The design D_c is not functionally equivalent to D but it can be used to mimic c independent copies of D, by interleaving the different input sequences. In some circumstances (datapath, simulation, etc.) c copies of D are advantageous.

In addition to area, another advantage D_c has over D is that its *iteration bound* (IB) is $1/c$ that of D. The iteration bound of a design is the largest average delay on a cycle of gates and flops in the design.

In theory, a design can always be clocked to achieve the iteration bound, through combinations of retiming, logic replication, time-borrowing and useful skew. In practise, it may be hard to achieve the iteration bound with edge triggered flops, and a fixed clock.

The goal of this project is to see what the practical impact of c-slow is in terms of clock speed. You are to take benchmarks and c-slow them, and run synthesis (through SIS) to determine the critical path after optimization.

Resources:

- sequential benchmarks: `/home/projects/logic_synthesis/benchmarks/seq-blif`
- generic libraries and optimization scripts: `/home/projects/logic_synthesis/sis/sis/s` (look carefully at `script.timing`)

7. **Building circuits from relations**

   Often it is most natural to describe the behavior desired from a hardware block in terms of relations. Suppose the inputs are x_1, x_2, x_3, and the outputs are y_1, y_2. Then a declarative specification could be

$$\begin{aligned} x_1 \cdot x_2 \cdot x_3 &\rightarrow (y_1 + y_2) \\ (x_1)' \cdot (x_2)' + x_3 &\rightarrow (y_1)' \\ (x_1)' + x_2 &\rightarrow (y_2)' \end{aligned}$$

   There are a number of techniques for implementing logic circuits from such relations, and the goal of this project is to experiment with them, and see if there are ways in which they can be improved. See [4] for a good overview.

8. **Synthesis with noisy gates**

   The goal of this project is to experiment with techniques for implementing reliable computing with unreliable gates. The idea is that through redundancy, a design with unreliable can be made more reliable. The redundancy can come in several ways, e.g., through simple replication or by more sophisticated coding. See the following article for an overview:

   - `http://ieeexplore.ieee.org/iel5/7729/33729/01605221.pdf?arnumber=160`

   I am particularly interested in seeing if there are ways in which a logic circuit with unreliable gates can be efficiently simulated on an FPGA.

Miscellaneous

A project of this nature will naturally build upon existing work. You are encouraged to build upon existing code/results, and in your report you may copy/adapt from others papers. However, you must explicitly make it clear that you have done so; failure to report this will be considered **plagiarism** and will be dealt with severely.

References

[1] Steven J. Friedman and Kenneth J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, 39(5):710–713, May 1990.

[2] Yi-Yu Liu, Kuo-Hua Wang, TingTing Hwang, and C. L. Liu. Binary decision diagram with minimum expected path length.

[3] A. Prakash, R. Kotla, T. Mandal, and A. Aziz. A high-performance architecture and bdd-based synthesis methodology for packet classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 83(10), June 2003.

[4] J. Yuan, K. Albin, A. Aziz, and C. Pixley. Constraint Synthesis for Environment Modeling in Functional Verification. *Design Automation Conference*, 2003.