ECEC-672 PROJECT

Cassius Ali Garcia[1], Varun Iyengar[2]

The purpose of this project was to create our own SSTA that not only works, but applies some form of statistical analysis to optimize some form of the calculation. We had chosen to use simulated annealing to optimize our cost, and we achieved an average decrease in cost of 8.5 percent while increasing the computational time by 2100 percent.

***Index Terms*—Timing; Uncertainty; Piecewise linear techniques; Digital integrated circuits; Integrated circuit technology; Density functional theory; Delay; Probability distribution;Accuracy; Performance analysis**

I. INTRODUCTION

Real life circuits are not static objects, there is uncertainty and variability in them hence the most accurate way to model and analyze real-life circuits is by using a tool that can handle the variability. The goal behind this project was to design a Statistical Static Timing Analysis tool. In the rapidly evolving field of circuit design, the accurate prediction of circuit performance under varying conditions is crucial. Standard STA tools deal with the worst-case timing of any gate and wire with a deterministic approach where an SSTA tool performs timing analysis using normal distributions of delays on gates and wires. This aims to give the user a more accurate analysis of the circuit instead of the worst-case scenario. STA tools are overly conservative and deal with compounding worst-case situations that would rarely occur, SSTA tools avoid this worst-case timing and offer designers the ability to construct more efficient circuits and let them get as close to timing constraints as possible. In this paper, we present a detailed overview of the SSTA tool, including its underlying data structures, mathematical foundations, and algorithmic implementations. We also provide empirical results demonstrating the tool's effectiveness in accurately predicting timing behavior and optimizing circuit performance under uncertainty.

II. DATA STRUCTURES

Our program utilized 3 simple data structures. The first is the "Wire" class. This class held the beginning and end of the gate, as well as the delay values for that wire (i.e. a0, a1, a2, a3).

The second is the "OP" class. This held all of the information pulled from the cell library ".time" file. This file held information relating to the available cells, including model name, operational name, cost, and delay. The OP class held this same information.

The Last was the "Gate" class, which held various information about a specific gate. To be specific, it held the gate name, the different OP variations available for that gate type (up to three) as OP objects, all input and output wires as Wire objects.

Manuscript received December 1, 2012; revised August 26, 2015. Corresponding author: M. Shell (email: http://www.michaelshell.org/contact.html).

III. MATH AND ALGORITHM

In the base un-optimized SSTA algorithm, we start at the end of the circuit and work our way towards the inputs, appending each critical gate onto a list to then return to our main method. To do this we recursively call the traverseCircuit function with the current gate being analyzed. This function adds the cost of the gate to a tracking variable and looks at the gate input with the highest delay called maxDelayGate. We then find the gate with the highest delay from maxDelayGate to have the proper inputs for our math functions. We compute the delays by adding together the delays of the current gate and output wire from maxDelayGate and the maxDelayGate and the output wire from the highest delay gate attached to that. The addition of the gate and wire gets stored in a wire object which the two of them get passed into the math function maxObject which takes two wires and effectively combines them. This output is also treated as a wire object which makes it easier to track the delays of the critical path. We then update the critical path delay variable with the delays of this wire object now created and again recursively call traverseCircuit. The program terminates once it finds a gate that has no inputs, hence it would be an input signal and be the start of the circuit. We implement a few helper functions to make designing this algorithm easier. The math functions were all broken up into their component parts so when maxObject was called it would call a slew of helper functions that would compute the smaller pieces of the function separately before returning them to the greater max function. In our traverseCircuit function, we use two helper functions to add together the delays of two wires and a gate and a wire. As all of these computations are performed multiple times it was easier to hold them in a function.

The optimization algorithm we used was Simulated Annealing based on what was presented in the lecture slides. This algorithm starts with an initial solution, the base unoptimized critical path, created off of our base SSTA. Simulated annealing uses a "temp" variable to control the randomness of the search. Initially we set the temp to be high and as the algorithm runs the temp slowly decreases which converges us onto an optimal solution. With our implementation, we generate another critical path but swap out a random gate for a random variation and if the cost is better the new solution is immediately accepted, if it is worse then it is only

selected depending on the temp. Higher temps lead to a larger chance of ”worse” solutions being picked. This probability decreases as the temperature decreases, allowing the algorithm to escape local optima early on but gradually converge to the best solution as the temperature decreases. The algorithm terminates once the temp falls below a specified threshold. By iteratively exploring and refining solutions while gradually reducing the temperature, simulated annealing can effectively optimize the critical path of the circuit, balancing between exploration and exploitation to find a near-optimal solution. Simulated annealing is a very flexible solution and it explores a wide range of solutions early in the search process and gradually focuses on promising regions as the temperature decreases. The behavior of simulated annealing can be controlled through parameters such as initial temperature, cooling rate, and termination threshold, allowing us to fine-tune the algorithm for this specific problem. However, compared to more deterministic optimization algorithms it reaches the optimal solution considerably slower, especially with problems with complex search spaces such as this. Choosing the appropriate parameters is vital as well as a poor choice can drastically decrease the efficiency of the program. If the temp is too low then the algorithm will be stuck in a non-optimal solution. If the temp is too high then the runtime of the program will increase drastically as the time to convergence towards the optimal solution will be extended. For our data, we chose the values of 1000, 0.95, and 0.001 for our temp, threshold, and cooling values respectively. These values were chosen after some experimentation and led to a relatively shorter runtime as compared to other temp, threshold, and cooling values.

IV. Data Analysis

Using our algorithm, we incited an average decrease in cost of 8.6 percent. There is however a large trade-off because while you can expect some cost decrease in cost, the time it takes to execute this new algorithm takes nearly 2100 percent more time to complete.

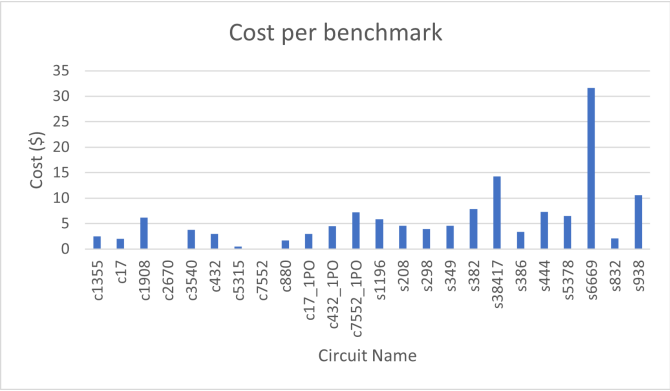

Fig. 1. Cost by dollar of each benchmark circuit

V. Conclusion

SSTA is still a developing field of study and many potential future uses for it have already been identified. SSTA tools are extremely useful when dealing with nanometer-sized technology and with the rate devices are shrinking most analyses will be done with SSTA tools in the future. Similarly, as GPUs and CPUs are being designed to be faster and more efficient, the accuracy SSTA offers would be vital in hitting these extremely tight timing margins. The field of semiconductor design will be heavily dependant on SSTA tools.

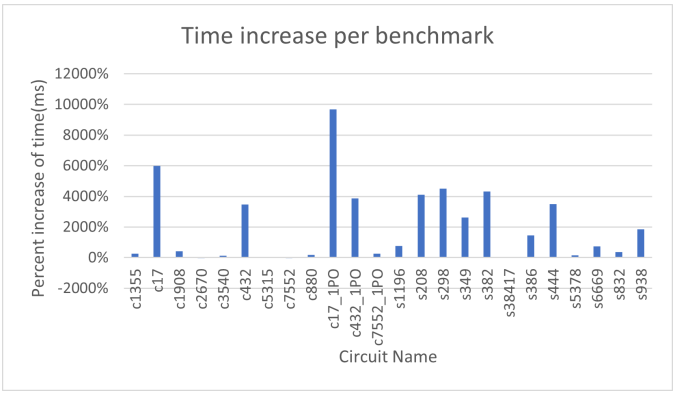

Fig. 2. Percentage increase of runtime between optimized and non-optimized SSTA

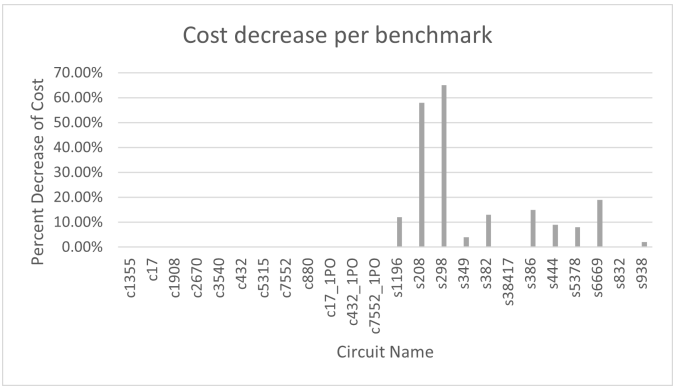

Fig. 3. Percentage decrease of cost between optimized and non-optimized SSTA

As a side note one of the benchmark files seemed to be broken as it didn’t follow the same format as the other files. The benchmark being C3540 1PO

With the optimization we picked for our SSTA tool there is a definite slowdown of the algorithm however we focused solely on optimizing the cost and the method of simulated annealing is a rather simple algorithm. It may not be the most efficient but implementing it based on our base SSTA code proved not to be too challenging. We picked Simulated annealing due to the simplicity of the algorithm, the ability for it to avoid local minima, and the flexibility of the program giving us the ability to apply this to any of the circuits at hand. However, the trade-off for the simplicity and flexibility is that the algorithm is very computationally intensive requiring a large amount of recursions and multiple runs of the original traverseCircuit method leading to an exponential increase in runtime. There is also the fact that we may not have chosen the most optimal cooling schema with other values that we had not tested potentially being more efficient at finding the global

optima than the chosen values. Overall given that there was a clear cost optimization for about half the circuits, simulated annealing might not have been the best choice but it provided a functional optimization.

REFERENCES

[1] A. Devgan and C. Kashyap, "Block-based static timing analysis with uncertainty," ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486), San Jose, CA, USA, 2003, pp. 607-614, doi: 10.1109/ICCAD.2003.159744.