AN EFFICIENT TRANSISTOR OPTIMIZER FOR CUSTOM CIRCUITS†

Xiao Yan Yu[1,2], Vojin G. Oklobdzija[1,2], William W. Walker[2]

[1]ACSEL Laboratory
Electrical and Computer Engineering Department, University of California
Davis, CA 95616
http://www.ece.ucdavis.edu
(yanzi,vojin)@ece.ucdavis.edu

[2]Advanced LSI Research
Fujitsu Laboratory of America
Sunnyvale, CA
walker@fla.fujitsu.com

Abstract

We present an equation-based transistor size optimizer that minimizes delay of custom circuits. Our method uses static timing analysis to find the critical paths and numerical methods to optimize transistor sizes continuously without using simulation. Consequently, it is faster than simulation-based optimizers, and more general than standard cell optimizers. We demonstrate its efficacy and accuracy on a dynamic adder, where we achieve a 54% speed-up and final critical path delay that matches Spice within 1%.

1. INTRODUCTION

Transistor sizing of CMOS circuits is an essential design step to improve performance. In pipelined microprocessors, paths between registers that exceed the clock cycle time require transistor sizing to reduce delay. Sizing transistors is one of the most difficult and time-consuming designer tasks, consequently, much research has been done on algorithmic sizing procedures to reduce the designer burden. Approaches to optimization that have been presented in the literatures can be classified into four major categories:

First is Spice-based Optimization**,** Star-HSPICE®, for example, can be programmed to optimize delay, power, or any products, such as Delay×Power^2. However, due to the long run times associated with Spice simulation, and the nonlinear behavior of transistors, the algorithm is inefficient for circuits containing more than a handful of transistors or fails to converge at all to a global minimum.

Second is event-driven simulator optimization such as Jiffytune [3], which replace the Spice simulator with a faster event-driven simulator. Event-driven simulation is approximately two orders of magnitude faster than Spice, and can process about 1,000 transistors efficiently. However, the problem of manually finding and sensitizing critical paths for a large circuit becomes intractable for the user.

† This work has been supported by SRC Research Grant No. 931.001, Fujitsu Laboratories of America and California MICRO 01-063.

Furthermore, circuit blocks in modern microprocessors frequently contain thousands of gates, which exceed the capacity of even this class of optimizer.

Third is static timing based standard cell optimization. Since standard cells come in discrete sizes, these optimizers rely on substitution of different size gates to find delay and power minima rather than gradient-based optimization. One of the problems with these optimizers is the lack of application to full custom circuits with arbitrary transistor sizes and design styles.

Finally, there is the method of Logical Effort [1]. This is a gate-based critical path optimization strategy used to minimize delay. It can be applied to standard cells or custom circuits. Because it uses a very simple delay formulation ($d = d_0 + RC_L$) and normalizes delay to an inverter, a simple general principle can be derived – that of equalizing stage effort - allowing back-of-the-envelope optimization of a critical path. The problem with logical effort is that it does not lend itself well to optimization of multiple interconnected paths, and its delay model error can exceed 20%.

Our optimizer uses the best features of the previous methods. First, we use accurate scalable delay equations rather than a circuit simulator because we desire to handle large circuits, and we can – and will – show that our delay equations produce less than 10% error compared to Spice simulations. Second, we use static timing analysis to find critical paths, but unlike previous static timing based optimization, our method uses continuously variable transistor sizes, and also applies to dynamic circuits. Consequently, it is applicable to custom design, although it retains the efficiency of static timing based standard cell optimizers.

2. OPTIMIZATION ALGORITHM

2.1 Overview of the Optimizer

Our optimizer uses an iterative algorithm to perform delay optimization. Fig.1. is a flow chart describing the

algorithm. The subsequent sections describe the details. We solve the following problem: given a Spice

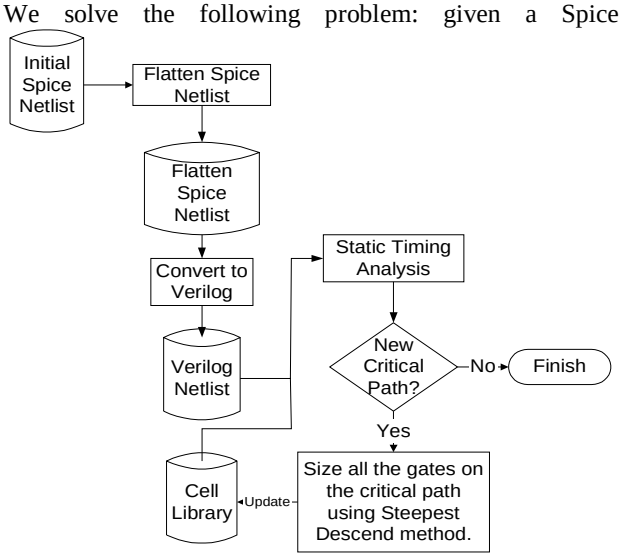

Fig. 1. Flow-Chart describing the algorithm.

netlist with an initial set of (poorly sized) primitive cells with delays modeled by scalable equations and no knowledge of the critical paths, what is the optimum size of each cell that minimizes the critical path delay(s) of the circuit?

2.1.1 Initialization

The initial Spice netlist is flattened down to the primitives in the library. A primitive can be any inverting static or dynamic gate. Flattening is needed so that multiple instances of the same primitive gate can be sized independently. The netlist is then converted to Verilog format to be read in by the static timing analyzer together with the library file. Furthermore, the library cells are copied, one for each instance name, with instance names appended to the cell names. This facilitates independent sizing of every gate in the design.

2.1.2 Static Timing Analysis

This block uses a commercial static timing analyzer (Synopsys Primetime) to analyze the entire circuit and report the critical path based on the delay model described in Section 2.2. If no new critical path is found in the current loop, the algorithm has converged.

2.1.3 Size Gates in Critical Path

This procedure sizes the gates in the critical path to minimize the delay. The optimization involves computation of the delay gradient with respect to the sizes of each gate in the critical path, then using the method of steepest descent to find a minimum. The library is updated with the new gate sizes. The details of the algorithm are described in Section 2.3.

2.2 Gate Delay Formulation

We model the gate delay, $d_{rise,}$ from input to output for output rising as follows:

$$d_{rise} = d_{0\,rise} + r_{rise} \times C_L + k_{fall} \times t_{fall} \quad (1)$$

where C_L is the load capacitance and t_{fall} is the 10-90% input fall time, and model coefficients are d_{0rise}, the intrinsic gate delay for output rising, r_{rise}, the rise resistance, and k_{fall}, the dependence of delay on input fall time. Similarly,

$$d_{fall} = d_{0\,fall} + r_{fall} \times C_L + k_{rise} \times t_{rise} \quad (2)$$

where t_{rise} is the input rise time and model coefficients are t_{0fall}, the intrinsic gate delay for output falling, r_{fall}, the fall resistance, and $k_{rise,}$ the dependence of delay on input rise time.

The 10-90% output rise time (t_{rise}) and output fall time (t_{fall}) are modeled as follow:

$$t_{rise} = t_{0rise} + r_{rt} \times C_L \quad (3)$$

$$t_{fall} = t_{0\,fall} + r_{ft} \times C_L \quad (4)$$

where t_{0rise}, t_{0fall}, r_{rt} and r_{ft} are model coefficients. All model coefficients are determined using a least-square fit to Spice simulations in which the output load and input transition time of the gate being characterized are varied. In the case of a dynamic gate, only d_{fall} and t_{fall} equations are used – assuming the dynamic node is pulled low during evaluation.

Our algorithm scales gates in a design during optimization. In our scaling formulation, we assume intrinsic delay, intrinsic rise-times and rise time dependence of delay are constant, resistances scale inversely with scale factor, S, and input capacitance scales linearly with S. i.e, delay equations for a scaled gate are related to the original equations as follows:

$$d_{rise} = d_{0rise} + \frac{r_{rise}}{S} \times C_L + k_{fall} \times t_{fall} \quad (5)$$

$$t_{rise} = t_{0rise} + \frac{r_{rt}}{S} \times C_L \quad (6)$$

The validity of these assumptions will be justified by the close agreement between the post-sizing model results and Spice simulations shown in Section 3.

2.3 Transistor Sizing

This section describes the sizing procedure for the transistors in the critical path reported by the static timing analyzer. The delay formulations were explained above. To show this procedure will produce a closed-form solution, consider the example path below.

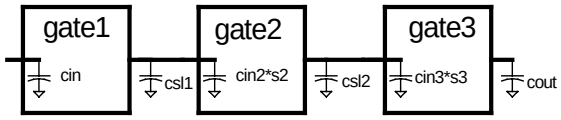

Fig. 2. A sample path demonstrating transistor sizing

The path consists of Gate 1, Gate 2 and Gate 3. Gate 1 has a fixed size (which we arbitrarily set to 1) and drives Gate 2 which is sized by a factor S_2 and a side capacitance of C_{sl1}. Gate 2 drives Gate 3 which is sized by a factor S_3 and a side capacitance of C_{sl2}. Our optimizer determines S_2 and S_3 numerically by using the method of Steepest Descent, which is explained below. The equations for the delays through each gate are listed below:

$$d_1 = d_{0_1} + r_1 \times (C_{sl1} + C_2 S_2) + k_1 \times t_{tran_1} \quad (6a)$$

$$d_2 = d_{0_2} + r_2 \times (C_{sl2} + C_3 S_3) + k_2 \times t_{tran_2} \quad (6b)$$

$$d_3 = d_{0_3} + r_3 \times C_{out} + k_3 \times t_{tran_3} \quad (6c)$$

where t_{01}, t_{02} and t_{03} are the intrinsic delay values for gate 1, gate 2 and gate 3 respectively, r_1 through r_3 are the on-resistance value of each gate, k_1 through k_3 are the input transition time constants for each gate and t_{tran1} through t_{tran3} are the input transition times to each gate. Note that the critical path can be either rising-falling-rising at outputs of gates1-3, or falling-rising-falling, so we have omitted the rise/fall subscripts on the model coefficients. S_2 and S_3 are the unknown sizes that we select using the Steepest Descent method to minimize the delay. This method starts with the initial size, $\mathbf{S}_0$. At each step of the iteration, the size vector, **S**, is updated by equation (7) where ∇d is the gradient vector of the delay with respect to the sizes, and δ is the step size. Iteration stops when the relative delay improvement between iterations is below the convergence threshold.

$$\vec{S}_{i+1} = \vec{S}_i - \delta \bullet \nabla d \quad (7)$$

2.4 Target Technology and optimization setup

The evaluation of this optimizer is based on a commercial CMOS technology [4], with 0.11µm minimum feature size and 1.2V power supply. Pre-layout wiring estimates were based on 10 µm data-path bit pitch. All circuits were optimized using 12 times the minimum sized inverter as driver and 40fF capacitor as the load at 25°C temperatures.

3. SIMULATION RESULTS

Our optimizer was tested on several adder critical paths and one complete adder proposed in [5]. These tests were performed on a SunFire 280R computer.

3.1 General Simulation Procedure

The procedure is as follows.

1. The primitive library coefficients of all cells used by the circuits are derived by fitting to Spice simulations using least squares.
2. Schematics are created and the wiring capacitances are drawn as lumped capacitance estimated using the technology profile in Sec. 2.4.
3. Initial Spice netlists are extracted from schematic capture with input and output pins at the top level specified.
4. The optimizer is run to minimize the delay by using the algorithm shown in Fig. 1.

3.2 Optimization Results on Different Adder Circuits

Table 1 shows the actual optimization results. Park Adder, with implementations shown in Fig. 4, is the complete 64-bit dynamic adder from [5] where: dynamic_HC_p2, static_HC_p2, dynamic_KS_p2 and static_KS_p2, are dynamic Prefix-2 Han Carlson adder critical path, static Prefix-2 Han Carlson adder critical path, dynamic Prefix-2 Kogge Stone Adder critical path and static Prefix-2 Kogge Stone Adder critical path respectively with shown in Fig. 3. Since Park Adder is a complete adder with no initial effort on sizing, the optimizer has to iterate over different paths and optimize the sizes of the cells on each path for minimum delay. It converged at the 174th loop. The critical path circuits for both Dynamic and Static version of Prefix-2 Han-Carlson and Kogge Stone Adders converged at the second loop since the optimizer does not have to iterate over a single path. The final optimized delays reported by the optimizer using our delay equation are shown in the third column together with the Spice times in column 4, and the error in column 5. The improvement, which is the percentage difference between the initial and final critical path delay after optimization, is shown in column 6, and the CPU times for each run are shown in column 7. We must point out again that no attempt to pre-optimize the circuits before running the software was made; all circuits were designed with unit gates. However this is the preferred design style because it reduces the primitive library size and designer effort.

4. CONCLUSION

This paper presented a delay optimizer that is capable of sizing primitive gates in any custom circuit, including dynamic circuits, without the need to spend any effort on initial sizing. It iteratively makes use of a static timing analyzer to report critical paths and then sizes transistors in the paths using a delay equation model. Several test circuits have been used to prove its efficacy. The final output critical path delay, modeled by scaling the initial

primitive delay equations, is within 5% of Spice simulations.

Acknowledgement

The authors would like to thank Hoang Q. Dao for providing the test circuits.

REFERENCES

[1] I.E. Sutherland and R. F. Sproull, "Logical Effort: Designing for Speed on the Back of an Envelope", in C.H. Sequin, Ed., *Advanced Research in VLSI. Cambridge, MA: MIT Press, 1991.*

[2] I. Sutherland, B. Sproull, D. Harris, "Logical Effort: Designing Fast CMOS Circuits," *Morgan Kaufmann Publisher, 1999.*

[3] Conn et. al, "Optimization of Custom MOS Circuits by Transistor Sizing", *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on , 1996, pp. 174 -180.*

[4] Y. Takao et al, "A 0.11 μm CMOS Technology with Copper and Very low-k Interconnects for High-Performance Sytem-On-a Chip Cores", *Electron Devices Meeting, 2000. IEDM Technical Digest. International, 2000.*

[5] J. Park, H. C. Ngo, J. A. Silberman, S. H. Dhong, "470ps 640-Bit Parallel Binary Adder," *Symposium on VLSI Circuits Digest of Technical Papers, pp. 192-193, 2000.*

[6] H. Ling, "High-Speed Binary Adder", *IBM J. Res. Dev., vol.25, p.156-66, 1981.*

[7] Mathew, S.K., Krishnamurthy, R.K., Anders, M.A., Rios, R., Mistry, K.R., Soumyanath, K., "Sub-500-ps 64-b ALUs in 0.18-/spl mu/m SOI/bulk CMOS: design and scaling trends", *Solid-State Circuits, IEEE Journal of Volume 11 , Nov. 2001.*

[8] Press, W. H, et al, "Numerical Recipes, the Art of Scientific Computing", *Cambridge Univ. Press, 1986.*

[9] Hoang Dao, Vojin G. Oklobdzija, "Performance Comparison of VLSI Adders Using Logical Effort", *12th International Workshop on Power And Timing Modeling, Optimization and Simulation, Sevilla, SPAIN, September 11-13, 2002.*

[10] V. G. Oklobdzija, "High-Performance System Design: Circuits and Logic", *IEEE Press, 1999.*

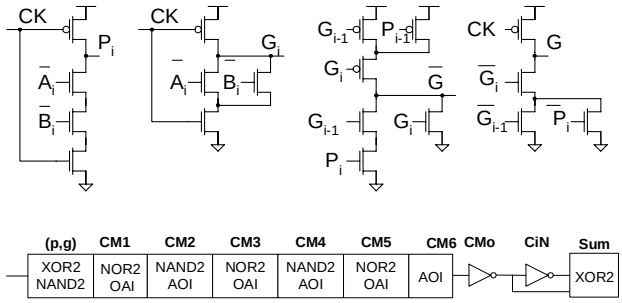

Fig. 3: Critical path block and gate implementation for Prefix-2 Han Carlson and Kogge Stone Adder

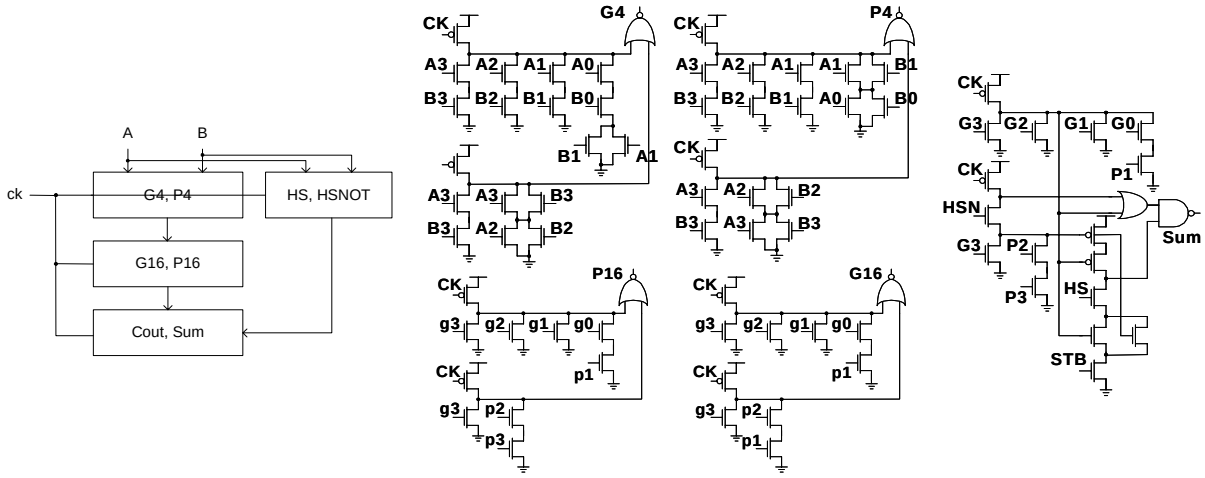

Fig. 4: Critical path block and gate implementation Park Adder

Table 1 Simulation Results

		Final Critical Path Delay (ps)			Improvement over initial sizing	
Name	# of Cells	Model Time	Spice Time	Error (%)	Improvement (%)	CPU Time (s)
Park_Adder	1712	394.68	393.18	0.88	53.92	3811
Dynamic_HC_p2	16	480.59	474.02	1.39	25.39	22
Static_HC_p2	16	534.45	512.70	4.24	24.12	18
Dynamic_KS_p2	14	400.07	396.66	0.86	27.32	23
Static_KS_p2	14	511.32	500.25	2.21	10.99	17