Solving Vt swap for cells using linear programming

by Steve Kommrusch

Department of Computer Science, Colorado State University

December 16, 2016

Abstract

As silicon designs have moved into sub-micron technologies like 28nm, 14nm and beyond, the problem of minimizing power is increasingly important. Lower power designs save power supply component cost, reduce heat generated, and increase battery life in mobile products. At one point in the design process the logic gates which will implement the design intent have been specified and there is a need to optimize the speed versus power of these gates by adjusting threshold voltages (Vt). This paper discusses linear programming methods that can be used to minimize power while still meeting the timing constraints of the design.

1. Introduction

This paper is written for the Math 510 Linear Programming class at Colorado State University, and discusses a topic of interest to silicon design companies. Data on a small design (3102 gates) was gathered from silicon analysis tools and used to provide a cost function and constraints for a linear programming problem, as discussed in section 2. The power and timing data was linearly scaled to protect proprietary details of the design. Results show measurable value in using this approach and further enhancements to handle larger designs appear warranted.

2. Problem formulation

2.1 Overview of digital logic

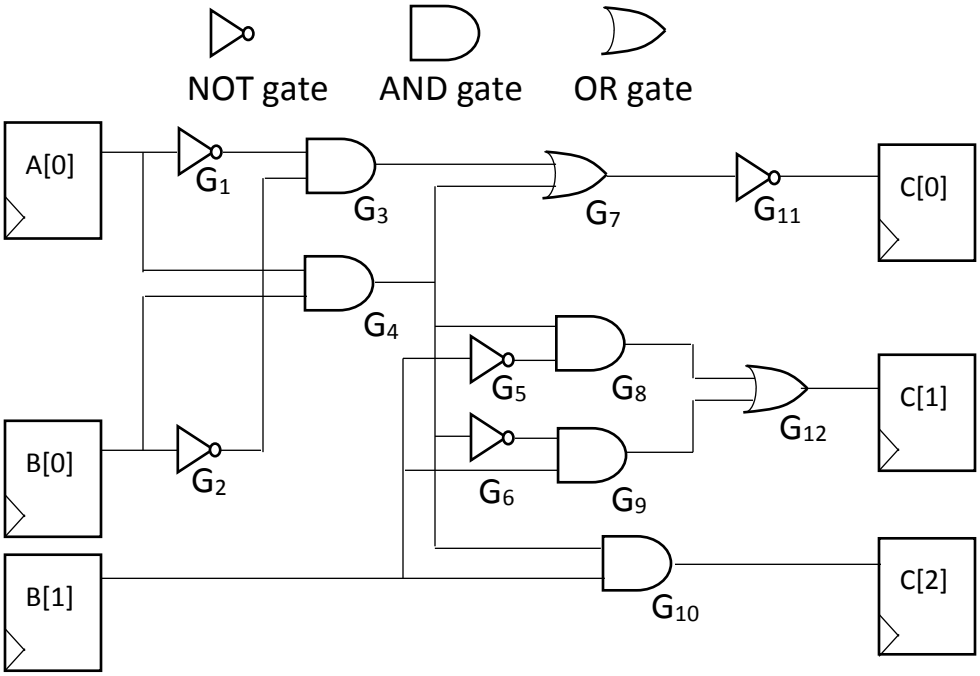
Digital logic is used to implement functions between flip-flops that are running at the full target clock period (well below 1 nanosecond in modern designs). As an example, consider the equation written in Verilog code as: C[2:0] = A[0] + B[1:0]. This equation takes 3 binary inputs (1 bit for A[0] and 2 for B[1:0]) and produces a 3-bit binary sum as the output. The logic for each bit in the C value can be represented as:

$$C[0] = (A[0]B[0] \mid \overline{A[0]}\,\overline{B[0]}); \quad \text{// NOT ((A[0] AND B[0]) OR (NOT A[0] AND NOT B[0]))}$$

$$C[1] = A[0]B[0]\overline{B[1]} \mid (A[0]B[0])B[1];$$

$$C[2] = A[0]B[0]B[1];$$

Those 3 binary functions of the 3 input variables can be drawn using these 12 AND, OR, and NOT gates connecting together flip-flops running at a common clock period.



In this example, consider the case where each of the 3 gate types has 3 different Vt options, detailed in the table below. Note that the power and delay values don't have to be integers.

Gate type	Low Vt power	Low Vt delay	Mid Vt power	Mid Vt delay	High Vt power	High Vt delay
NOT	50	70	10	100	2	130
AND	100.7	100	20	140	4	180
OR	110	110	22	155	4.5	200

Note than some silicon foundries support 4 Vt flavors and there are techniques to create discrete partial flavors so it is possible to have 8 or more power vs delay settings for each gate. For the proposed problem formulation, each gate will have 3 free variables such that G_1 has the variables G_{1L}, G_{1M}, and G_{1H}, corresponding to Low Vt (low delay but high power), Mid Vt (middle delay, middle power), and High Vt (high delay but low power).

In this example, consider that the clock period we want to target is 450, so none of the paths from the A or B flip-flops to the C flip-flops can add up to more than 450 (in actual designs, the 450 number can vary slightly because the flip-flops may have routing delays or loading, but for this example, I will use 450 for all the paths in this example).

2.2 Linear program formulation

Now we can formulate the linear program for the circuit shown:

$$\begin{aligned}\text{minimize } & 50G_{1L}+10G_{1M}+2G_{1H} + 50G_{2L}+10G_{2M}+2G_{2H} + 100.7G_{3L}+20G_{3M}+4G_{3H} \\ & + 100.7G_{4L}+20G_{4M}+4G_{4H} + 50G_{5L}+10G_{5M}+2G_{5H} + 50G_{6L}+10G_{6M}+2G_{6H} \\ & + 110G_{7L}+22G_{7M}+4.5G_{7H} + 100.7G_{8L}+20G_{8M}+4G_{8H} + 100.7G_{9L}+20G_{9M}+4G_{9H} \\ & + 100.7G_{10L}+20G_{10M}+4G_{10H} + 50G_{11L}+10G_{11M}+2G_{11H} + 110G_{12L}+22G_{12M}+4.5G_{12H}\end{aligned}$$

subject to:

$$\begin{aligned}& 70G_{1L}+100G_{1M}+130G_{1H} + 100G_{3L}+140G_{3M}+180G_{3H} \\ & \quad + 110G_{7L}+155G_{7M}+200G_{7H} + 70G_{11L}+100G_{11M}+130G_{11H} < 450 \\ & 100G_{4L}+140G_{4M}+180G_{4H} \\ & \quad + 110G_{7L}+155G_{7M}+200G_{7H} + 70G_{11L}+100G_{11M}+130G_{11H} < 450\end{aligned}$$

… (8 more path constraints not shown here) …

$$100G_{10L}+140G_{10M}+180G_{10H} < 450$$

For all i: $G_{iL}+G_{iM}+G_{iH} = 1$

For all i: G_{iL}, G_{iM}, $G_{iH} \geq 0$, and all are integers (i.e., all are 0 or 1).

I wrote a Perl script that transformed the timing results from a common industry tool output into the matrixes needed for Matlab to use. The command to solve the Vt swap problem was:

[x, fval] = linprog(c,A,b,Aeq,beq,zeros(freevariables,1),ones(freevariables,1));

This command solves the classic linear programming problem using the free variables in x:

minimize:	$c^T x$
subject to the constraints:	$A*x <= b$
	$Aeq*x = beq$
	$0 <= x_i <= 1$ for all i.

The variables in the command are described here in relation to this problem:

c: This is the cost vector which includes the power for each of the 3 Vt flavors for each gate (for example, it is a vector of size 9306 for a problem with 3102 gates).

A: This is the constraint matrix that identifies delay contribution for each gate type in each delay path. For example, for 9460 paths and 3102 gates, A is a 9460x9306 array.

b: This is the maximum delay allowed on each path through the gates. Each entry in **b** is about the same, but they vary to account for source flop output delay and destination flop setup time requirements.

Aeq: This matrix has 3 "1"s in each row otherwise populated with "0"s and insures the 3 Vt flavors for a gate all sum up to 1. For 3102 gates, this is a 3102x9306 array.

beq: This is just "ones(numgates,1)" to constrain the 3 Vt flavors of each gate sum to 1.

zeros/ones: These commands in the linprog call insure all the Vt flavors selected are less than or equal to 1 and greater than or equal to 0.

x: The result x is the vector of gate flavors that minimize $c^T x$ while meeting the delays.

fval: This is the final power of the gates chose and is equal to $c^T x$.

2.3 Hardware and tool description

The environment that I evaluated was on an HP ProBook 6565b running Windows 7 Service Pack 1 with 8GB of installed memory. I used Matlab version R2015b (8.6.0.267246).

3. Results

Early results showed that intlinprog, which will keep all the gate selections integers as ultimately required, takes significantly longer than linprog for this problem. Below are timing and output results for various problem sizes of my initial coding for this problem. Time is measured using Matlab's cputime to track start and end of command.

Paths	Gates	File size (MB)	File load time	linprog					intlinprog				
				time	pwr	LVT	MVT	HVT	time	pwr	LVT	MVT	HVT
50	242	0.45	1.7s	0.72s	3875	175.5	66.2	0.3	58.3s	3967.8	180	62	0
100	318	0.84	3.9s	0.27s	5281	234.3	82.0	1.7	29min	5444.6	233	84	1
200	388	1.44	6.9s	2.14s	6366	280.3	106.2	1.5	>20m				
500	611	4.2	19.7s	5.2s	948.6	420.3	187.3	3.4	>10m				
1000	853	9.8	46.4s	10.7s	12186	557.0	288.9	7.1					
2000	1253	24.9	124s	21.3s	16698	796.4	445.4	11.2					
5000	2373	105	599s	84.5s									
9460	3102	233	>90m										

3.1 Improved load time

Noting that the load time was impacting the total time to solve the problem, I learned from internet surfing that using the load command ("A=load('A9460.txt');") is much faster than the *.m program I was trying to use. The file sizes are similar, but loading the A and Aeq arrays with the "load" command only takes a few minutes, even for the 9460 path problem.

Paths	Gates	Total File size (MB)	Total File load time	linprog				
				time	pwr	LVT	MVT	HVT
9460	3102	238	~5m	304s	32066	1740.1	1293.5	68.4

3.2 Full results

After getting the full 9460-path problem to load and abandoning intlinprog, I created a realizable Vt solution by rounding all gate types to the faster (higher power) gate. So, gates that linprog recommended as 0.6 MVT and 0.4 HVT were set to 1 MVT. After that rounding, I ran a loop that tested power saving opportunities in order to see which cells could be slowed down individually. The final result of 34,363 power units is very encouraging, representing a significant 36% power savings from the original gates which used the fastest, but highest power gates for all the paths.

Description	Power	LVT	MVT	HVT
Original gates before Vt swap	53430	3102	0	0
linprog	32066	1740.1	1293.5	68.4
Round all x results to faster integer cell type	35553	1975	1102	25
Test lower power gate and swap if Ax<=b still met	34363	1863	1202	37

4. Robustness

This problem is sensitive to the timing path requirements as would be expected from the gates synthesis process. In order to double-check the validity of the approach, the path constraints represented by the b variable were scaled as shown below. Scaling b by 0.99 creates an infeasible solution (indicating even all LVT cells in the paths can't meet this constraint). This result is to be expected from the way the gates are generated which are optimized to meet the timing target using the fewest total number of gates. Scaling b by 0.999 is still a feasible problem, but more LVT cells are used. Scaling by a sufficiently large value allows the timing paths to be met even with all HVT (slow, low-power cells).

Scaling on b path delay limit	Power	LVT	MVT	HVT
0.99	Problem infeasible			
0.999	32361	1760.7	1274.5	66.7
1.0	32066	1740.1	1293.5	68.4
1.1	16395	446.9	2255.4	399.7
1.5	6772	4.0	482.1	2615.9
1.9	5972	0	0	3102

5. Evaluation of Approach

Compared with other techniques in use in the industry [3] the results for this problem are very encouraging. Although intlinprog became unusable after 100 paths, I compared the procedure developed in the previous section to the intlinprog results for 50 and 100 paths, as shown below. These results show some benefit remains from an optimal linear programming solution.

Paths	Gates	linprog				intlinprog				Proposed procedure			
		pwr	LVT	MVT	HVT	pwr	LVT	MVT	HVT	pwr	LVT	MVT	HVT
50	242	3875	175.5	66.2	0.3	3967.8	180	62	0	4033.1	183	59	0
100	318	5281	234.3	82.0	1.7	5444.6	233	84	1	5609.1	246	70	2

6. Extensions of the Methodology

Some further areas of work in this area would be:

- Test recommended cell swap list in silicon timing tool to insure proper recommendations are being generated. May need more complete path list depending on result.
- Explore sparse matrix integer linear programming solutions for optimal integer results. The A matrix in this problem is 99.4% zeros.
- Explore alternate clean-up algorithms after rounding to faster cell type step.
- Explore larger problem sizes on larger machines.

With a brief internet search, I did not find articles that evaluate this particular approach. Tao Luo, et. al. [3] describe algorithms for gates optimizations, some of which include linear programming, but their Vt swap algorithm uses something like my final pass where lower power cells are swapped in and tested for delay effect. Given my results detail the full integer linear program results for 50 and 100 paths, that algorithm is demonstrably worse than running the linear program to get the initial Vt settings.

It's possible this idea is worthy of publishing, but to provide value to the community some data from a silicon design company may be relevant and the company would need to approve the publication.

7. Conclusions

This paper describes a procedure to formulate the common Vt swap problem in silicon design as a linear program which can be solved efficiently using Matlab. Initial indications are that the optimal solution to the problem formulation maps well as a solution to the original Vt swap problem and further investigations may be fruitful.

8. References

[1] Transistor threshold voltages affect design performance
https://en.wikipedia.org/wiki/Threshold_voltage

[2] This paper is an accessible summary of the Vt swap problem that adds technical detail:
https://www.einfochips.com/component/k2/item/download/143_e233354968e0b94977429143ef0d2b73.html

[3] Total Power Optimization Combining Placement, Sizing, and Multi-Vt Through Slack Distribution Management. By Tao Luo, et. al.
https://www.researchgate.net/publication/4327441_Total_power_optimization_combining_placement_sizing_and_multi-Vt_through_slack_distribution_management