A Delay Model for Logic Synthesis of Continuously-Sized Networks

Joel Grodstein, Eric Lehman, Heather Harkness, Bill Grundmann, and Yosinatori Watanabe

Digital Equipment Corporation, 77 Reed Rd, Hudson, MA

Abstract: We present a new delay model for use in logic synthesis. A traditional model treats the area of a library cell as constant and makes the cell's delay a linear function of load. Our model is based on a different, but equally fundamental linearity in the equation relating area, delay, and load: namely, we may keep a cell's *delay* constant by making its *area* a linear function of load. This allows us to technology map using a library with continuous device sizing, satisfies certain electrical noise and power constraints, and in certain cases is computationally simpler than a traditional model. We give results to support these claims. A companion paper [14] uses the computational simplicity to explore a wide search space of algebraic factorings in a mapped network.

1. Introduction. Most technology mapping algorithms for logic synthesis have been targeted at technologies with a limited number of cell sizes. A straightforward modeling technique will then model each library element with a unique cell, whose area is fixed and whose delay varies with output loading. A class of technology-mapping algorithms called *tree-mapping* [1,2,3] is well suited to such a model. Given a tree-structured network and a fixed cell library, tree-mapping algorithms run in time linear in the number in the number of circuit nodes. They are also linear in the number of library cells, which is of course not a problem for these reasonably-small libraries.

At the other end of the performance spectrum, full-custom design can achieve high device densities and clock speeds [13]. However, it requires, among other things, the ability to create gates of any desired size. This conceptually implies an unbounded number of library cells, and clearly precludes the direct use of a tree mapper, whose execution time is linear in the library size. One alternative is to approximate the continuous library with a discrete, near-continuous (and very large) cell library. However, this produces suboptimal results (since the library is still not continuous), and is also slow.

We propose a new model for continuously-sized CMOS gates. In this model, a cell's *delay* will be held constant. As the cell's load changes, the cell's size automatically grows exactly enough to hold delay constant; making its *area* a function -- in fact a linear function -- of load. This model will enable us to use a modified tree-mapping technology to efficiently produce continuously-sized netlists satisfying certain electrical noise and power constraints.

Our own application is for continuously-sized, full-custom designs. However, the delay model is also applicable to other methodologies, such as high-end standard cell, where there are many sizes of each cell. Essentially, it applies to any technology where cell sizing to obtain a desired delay is viable.

Constant-delay modeling has been used frequently in technology-independent algorithms. For example, Wang [15,pg.167] proposed decomposing a network into bounded-fanin NAND gates, assigning a unit delay to each level of logic, and determining and restructuring critical regions with the resulting arrival times.

Singh[12, pp.13-19] has measured the accuracy of various technology-independent delay models. He concluded that the unit-delay model on bounded-fanin gates was the most accurate. His *speedup*[8] made technology-independent decomposition decisions by first breaking down the network into two-input NAND gates, and then modeling each NAND gate as a unit delay.

Saldanha et. al.[10] used path sensitization to make the longest real path in a network false. [10]'s initial implementation uses a unit-delay model on technology-independent two-input NAND gates. [10] argues that the unit-delay model is a reasonable heuristic, citing Singh's results from [16].

Each of the above approaches uses the constant-delay model as a simple, fast approximation to real delay. This simplicity allows them to quickly consider many different structurings of a technology-*independent* network. Our approach is different. We use the constant-delay model in a technology-*mapped* realm as the synthesis tool's most exact library representation. We will present, in Section 2, circuit simulation data to show that ours is a valid timing model. Section 3 demonstrates the model's use for technology mapping, and Section 4 gives results.

A companion paper [14] takes a different approach. Just as previous works have used a simple delay model to explore many different structurings of a technology-independent network, [14] uses our delay model's compu-

tational simplicity to explore a wide space of network restructurings in the *mapped* realm.

2. Our Delay Model. We have spent several years using a technology mapper based on [2] to synthesize the control logic of high-performance microprocessors. Our design methodology allows continuous sizing and the mapper did not, so we gave the mapper a large library of discrete gate sizes. As we included more cell sizes in our library, the mapper obtained progressively better results, but at the cost of more CPU time.

The mapper in [2] models the minimal arrival time at any node as a piecewise-linear (PWL) function of load. These PWL functions typically resemble Figure 1. Each node has many segments, each representing the use of a progressively larger cell as the node loading increases. Almost every size of gate is represented in the PWL arrival function at every node. Larger NAND2s typically have a higher base delay but a lower load-dependent delay; thus there will be some load at which any given size is optimal.

Figure 1 - PWL solution at a node

Unfortunately, this data representation becomes more redundant as we add more gate sizes to the library. The graph essentially implies that we should use a NAND2 and size it based on the actual loading. Our new model essentially lets us represent the solution at any node with exactly this information.

Any delay model must represent the relationship between cell area, load, and delay. A common model has the size of a gate being constant, and the gate's delay then varying linearly with output load. This model is intuitive and is accurate to a first order [9, pg.254].

There is another fundamental linear relationship between the variables of area, load, and delay. We can hold the delay of a NAND2 gate is held constant; as the output load changes, we change the gate size as necessary. SPICE simulation shows the relationship to be almost exactly linear. Intuitively, driving a constant amount more output load in the same time requires driving a constant amount more current, which requires a constant amount more device size. We give the following theorem [6].

> **Theorem I**. Given a gate obeying the Elmore[5] delay model: i.e., all transistors can be modeled as resistors with $R \propto (1/\text{device\_width})$, the capacitance on any node varies linearly with device width, and the delay to the gate output q is the elmore $\tau = \sum_i R_{iq} C_i$, where R_{iq} is the resistance of the common path between any internal node i and q. Then, if the external loading C_{ext} is treated as an independent variable, and the device sizes are scaled as needed to keep τ constant, it will result in gate area varying linearly in C_{ext}.

We next plot multiple delay lines for the same gate in Figure 2 (all plots are for a recent CMOS technology; delay numbers are scaled for proprietary reasons). If we then take a vertical line representing some given constant output load, we notice that as the gate speed gets faster, it requires more area increase to keep making it faster by a constant delay increment Δt. At some point, it becomes quite impossible to make the gate any faster.

Figure 2 - W vs. load, constant D

Any gate drives two loads -- the external load that we have plotted against, and also its *self-loading* capacitance. The self-loading consists of its internal cell wiring, parasitic and internal channel capacitances. For good cell layouts, we can assume these are dominated by effects proportional to device size. We now define a gate's self-loading ratio in unitless terms as the ratio of C_{self_load} / (C_{ext} + C_{self_load}). It represents the proportion of the electrical energy which is charging internal loads, as opposed to doing "real" work. We now state another theorem[6]:

> **Theorem 2**: Given a gate obeying the Elmore model as before. Then, at any given gate delay, its unitless self-loading ratio is an invariant. That is, self-loading ratio depends only on delay, and is independent of external loading as long as the gate size is increased to keep delay constant.

Figure 3 - Self-load vs. delay

Figure 3 shows this graphically (the four plots lie on top of each other). Theorem 2 also explains the phenomenon observed in Figure 2. As a gate gets faster, its self loading increases. It thus takes more extra area to make it faster by the same constant increment Δt.

2.1. Electrical and power constraints. We can also relate self-loading ratio to power. Self-loading capacitance implies work which is "useless" in that it is not charging external loads. Thus, keeping the self-loading ratio low implies a high ratio of useful work to self-loading work. Figure 3 tells us that when designing for minimum self-loading ratio, it is sufficient to consider a gate's delay only -- its area may be whatever it needs to be for the proper delay, without affecting the ratio.

We can thus use Figure 3 to pick a worst allowable self-loading and directly translate it into the fastest "reasonable" delay for that gate type. At the other end, making the gate delay slower and slower gives us diminishing area savings, as noted in Figure 3. This, and the fact that slower transition times result in reduced noise immunity to electrical coupling effects [7], give us a maximum "useful" gate delay.

Consider next the following definition of a delay-power efficiency product. Pick a given external load C_{ext}. Allow the size of the driving gate to vary and define d as the gate's delay at any given size. Define d_{min} as the gate delay at the 100% self-loading point and d_{rel} in unitless terms as d/d_{min}. Next define E_{rel} in unitless terms as $E_{rel}=E / C_{ext}V_{dd}^2$. Finally, define the delay-power efficiency as $(d_{rel} * E_{rel})^{-1}$. Figure 4 plots E_{rel} as a function of gate delay for a NAND2 gate swept over a wide range of gate sizes. Four plots at four different loads are superimposed. We see that the delay-power efficiency, as well, is a function of gate delay only, virtually independent of load.

Figure 4 -Efficiency vs. delay

2.2. Range of the delay model. When we combine the narrow peak of the delay-power efficiency curve with the aforementioned lower and upper bounds on gate delay, we find that the ratio between the fastest and slowest useful delays is on the order of 3:1 for every CMOS technology we have seen. We can now explain Singh's conclusion [12] that the unit-delay model is so reasonable. With a narrow delay range, simply counting the depth of a circuit in gates will approximate the real delay; especially since variations in gate delays tend to cancel over a long path.

3. Incorporation into a Tree-Mapping Technology Mapper. Our global strategy will be straightforward. A conventional delay model would use constant-area cells, and attempt to approximate the continuous area range by having many discrete cells. Instead, we use several constant-delay cells, and approximate the continuous delay range. Our mapper produces a wirelist containing cells with known delays, which are assigned their correct sizes by a simple sizing technique such as [9, pp 252].

Both the constant-area and constant-delay models relate area, load, and delay. Given a library with enough cells, either model can take any two of the three variables and predict the third, and thus are functionally equivalent in the limit case of densely-populated libraries. However, given the 3:1 range in cell delay and a nearly 100:1 range in cell area, our library can be far smaller than conventional ones -- for a methodology allowing continuous sizing.

We incorporate our delay model within a tree-mapping technology mapper [1,2]. Tree mappers can be used to optimize many different cost functions: e.g., area, delay, and area under a delay constraint.

Since our model essentially reverses the roles of area and delay, minimal-area tree mapping can now be done very similarly to existing min-delay algorithms (e.g., [2] ch. 2). Min-delay mapping is analogously done with an existing min-area algorithm (e.g., [1]).

Min-area mapping under a delay constraint is perhaps the most useful and difficult problem. There are several methods in the literature of dealing with it, e.g., [2, p.22] and [4]. We focus on the two-pass algorithm in [4]. For the first pass, [4] chose a constant value K, and assumed that the expected load at all tree-internal nodes was equal to K. This reduced the delay of each cell to a constant, and allowed [4] to store simple (arrival time, area) pairs for the solutions at each node instead of piecewise-linear functions. The inaccuracies due to [4]'s simplified model were assumed to be minimal, and were heuristically adjusted on a later pass. We keep the first pass from [4] exactly. Our cells have their native constant delays. Their area is heuristically assumed constant and equal to the slope of their actual area vs. load line. We then eliminate the second pass of [4] altogether and, as mentioned, replace it by a sizing technique similar to [9, pg.252].

4. Results and Conclusions. We have built a technology mapper using these ideas on top of SIS [11]. It is based on a tree mapper which minimizes area under a delay constraint (MADC), as described in Section 3.3, and followed by a simple device sizer based on [9, pg.252]. We have then built both a constant-area library and a constant-delay library. The constant-area library has approximately 16 sizes for each gate type. The constant-delay library has two. We have taken several networks from a low-power, high-performance microprocessor currently in design, and processed them with our technology mapper using the constant-delay library.

For comparison purposes, we have then converted the results to the best possible equivalent using the constant-area library. At nodes which are speed-critical, we choose the next-larger cell size to reduce delay. At nodes which are bounded by the methodology's slowest possible delay, we likewise round up to the next larger cell to avoid violating electrical constraints. At other nodes, we round to the nearest legal discrete cell size.

Table 1 shows the results. Column 1 gives the name of each example (the very bottom row gives geometric means of all examples). Column 2 gives the example's size. Columns 3 and 4 compare the minimum cycle time which can be produced by each library. Throughout the table, columns labelled "CD" give results for the constant-delay, continuously-sized library and columns labelled "CA" are for the discretely-sized, constant-area library. Note that, as expected, upsizing gates to the next discrete size past what our library designer considered to be the point of diminishing returns produced very little effect; a geometric mean of only 1.3% delay improvement. This will be dwarfed by routing effects when the network layout is done. In fact, one network even had a slightly slower delay with the larger cells. This is attributed to an increase in the back-biased source/drain diffusion diode capacitances associated with the larger devices.

Columns 5 and 6 give the area results for each library. As expected, the constant-delay library used significantly less area (14.8%) than did the discrete library. This is partially because the constant-delay library was able to use exactly the smallest cell size on noncritical nodes, where the constant-area library had to use the next larger size. It is also due to the constant-delay library avoiding area overkill on critical nodes. Columns 7 and 8 give the total power expended for each circuit. They disregard switching probabilities and use a simple model where power $\propto CV^2$. Note that the power measurements are roughly in line with the area measurements.

Finally, columns 9 and 10 compare delay-model complexity. Column 9 gives the total number of solution points used by our MADC mapper. Column 10 contrasts this to a min-delay piecewise-linear mapper such as in [2]. As mentioned in Section 2, the piecewise-linear mapper uses at least one solution point for each cell strength in the library at every node while calculating the

minimum-delay solution. A true MADC solution such as proposed in [2, pg. 22] would be substantially more expensive still. As expected, the computational gain from the 3:1 range in delays vs. the 100:1 range in areas is substantial. We observe that it is so substantial that it enables a true MADC mapper to use 15% fewer solution points than a simpler min-delay mapper. This computational simplicity will be used to good effect in [14].

In conclusion, we have developed a new delay model. Our model keeps the *delay* of any cell constant by varying the cell's size in proportion to changes in its output load. We have shown the model to be both accurate and computationally efficient, and motivated it with circuit-integrity and power considerations. We have used it to give insight into previous technology-independent delay modeling, and demonstrated its use in technology mapping. A companion paper [14] uses its computational simplicity to explore a wide range of structurings of a mapped network.

References

1. Rudell, R., "Logic Synthesis for VLSI Design," Memo UCB/ERL M89/49, U.C. Berkeley, 1989
2. Touati, H., "Performance-Oriented Technology Mapping," Memo UCB/ERL M90/109, U.C. Berkeley, 1990.
3. Keutzer, K., "DAGON: Technology Binding and Local Optimization by DAG Matching", Proc. DAC 1987, pp. 341.
4. Chaudhary,K. and M. Pedram,"A Near-Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints," Proc. DAC 1992, pp. 492.
5. Elmore, W.C., "The Transient Response of Damped Linear Networks ... ," J. Appl. Phys., V19, #1,Jan. 1948
6. Proofs available on request from the authors.
7. Grundmann, B, and YT Yen,"XREF/Coupling: Capacitive Coupling Error Checker", Proc ICCAD, 1990, p.244.
8. Singh, KJ, et. al.,"Timing Optimization of Combinational Logic," ICCAD-88, pp 282
9. Glasser, L. and D. Dobberpuhl, "The Design and Analysis of VLSI Circuits", Addison Wesley, 1985
10. Saldanha,A., H.Harkness, P.McGeer, et. al., ,"Performance Optimization Using Exact Sensitization", Proc. DAC 1994
11. Sentovich,E.et.al., Sequential Circuit Design using Synthesis and Optimization," Proc ICCD 1992, pp 328.
12. Singh,K.J.,"Performance Optimization of Digital Circuits," PhD Thesis, U.C.Berkeley, 1992.
13. Bowhill, et. al., "A 300Mhz 64b Quad-Issue CMOS RISC Microprocessor," Proc ISSCC 1995, pp.182.
14. Lehman,E.,Y.Watanabe,J.Grodstein,and H.Harkness,"Logic Decomposition During Technology Mapping," ICCAD 1995.
15. Wang, A., "Algorithms for Multi-level Logic Optimization," Memo UCB/ERL M89/50, U.C.Berkeley, 1989.

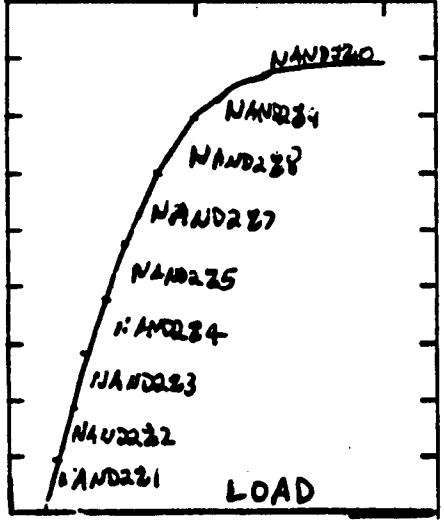

Figure 1 - PWL solution at a node

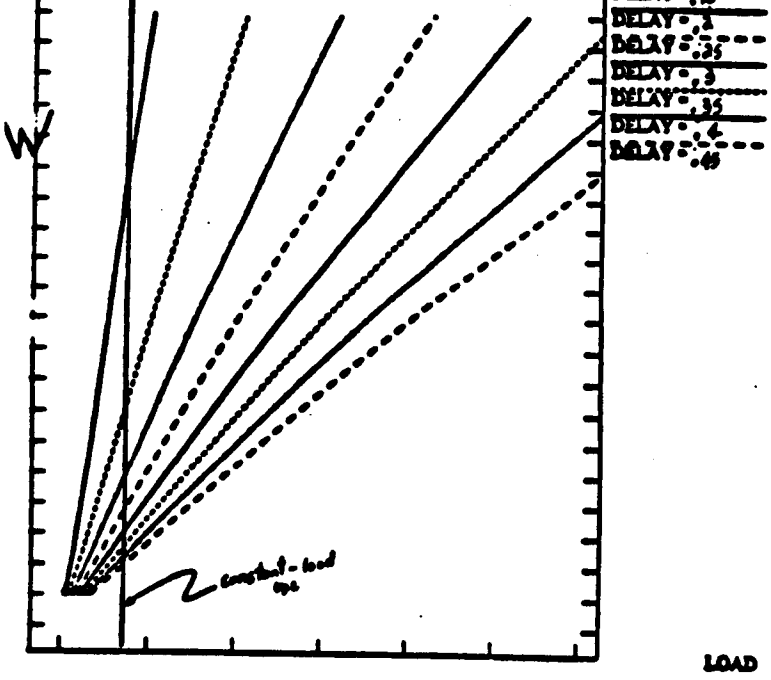

Figure 2 - W vs. load, constant D

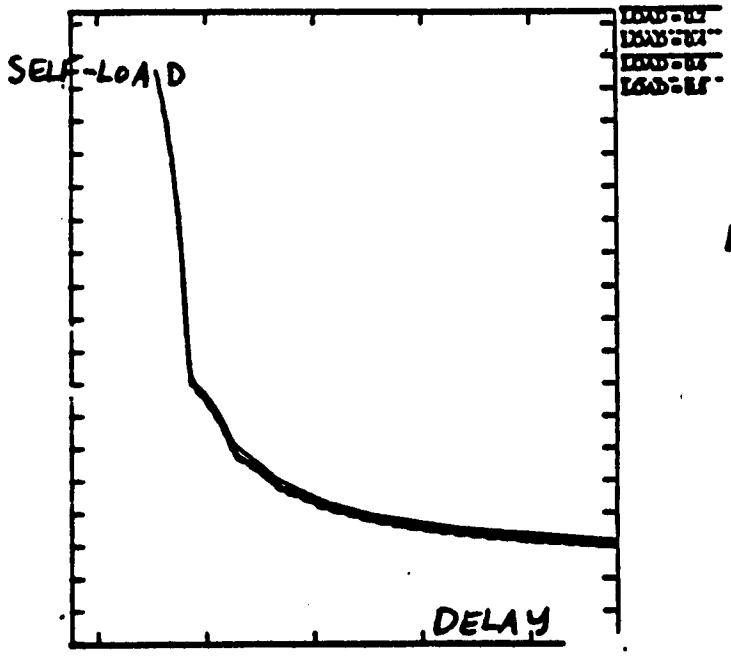

Figure 3 - Self-load vs. delay

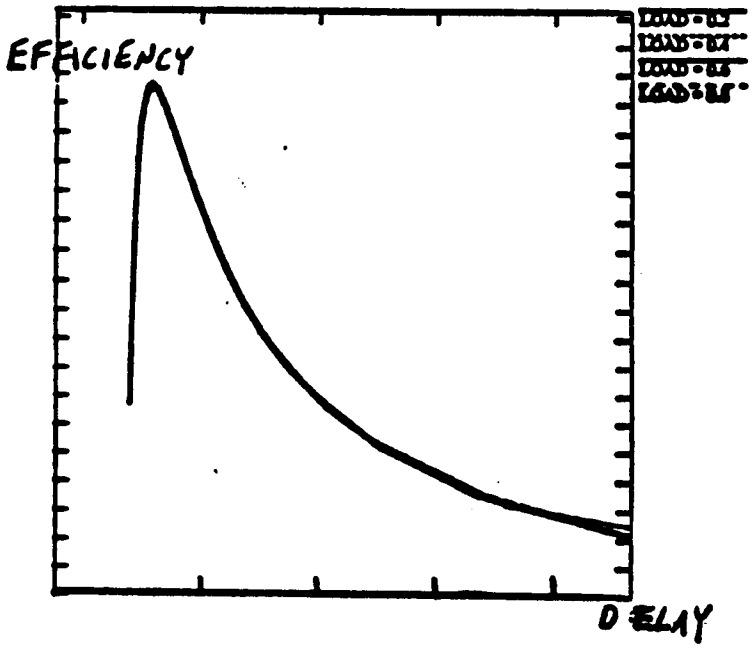

Figure 4 - Efficiency vs. delay

Table 1 - Results

circuit	gates	cycle time		area		power		# solution pts	
		CD	CA	CD	CA	CD	CA	CD	CA
an32a	26	1.17	1.14	270	318	11.4	13.3	320	768
cond	30	1.79	1.80	1124	1404	47.9	59.8	429	468
iclk	45	2.34	2.34	2069	2357	88.8	100.9	600	900
iexc	27	.46	.46	611	664	26.0	28.1	224	384
isctl	67	5.5	5.5	1765	2044	79.6	91.3	2727	1212
maccctl	43	.48	.45	896	998	39.6	44.1	780	720
mcastctl	50	1.66	1.65	2383	2667	104.7	116.8	737	804
g. mean:		1.387	1.37	1054	1210	45.7	52.2	603	706