Delay Estimation of VLSI Circuits from a High-Level View†

Mahadevamurty Nemani and Farid N. Najm

ECE Dept. and Coordinated Science Lab.
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Abstract

Estimation of the delay of a Boolean function from its functional description is an important step towards design exploration at the register transfer level (RTL). This paper addresses the problem of estimating the delay of certain *optimal* multi-level implementations of combinational circuits, given only their functional description. The proposed delay model uses a new complexity measure called the *delay measure* to estimate the delay. It has an advantage that it can be used to predict both, the minimum delay (associated with an optimum delay implementation) and the maximum delay (associated with an optimum area implementation) of a Boolean function without actually resorting to logic synthesis. The model is empirical and results demonstrating its feasibility and utility are presented.

1. Introduction

Rapid increase in the design complexity and reduction in design time have resulted in a need for CAD tools that can help make important design decisions *early* in the design process. To do so, these tools must operate with a design description at a high level of abstraction. Performance being an important design criterion, there is an increasing need for high-level delay estimation.

In this paper we will focus on the estimation of the delay associated with a circuit from its high-level description, i.e., given only a *functional* view of the circuit, such as when a circuit is described only with Boolean equations. We will estimate the minimum, maximum and other feasible delay points, from a high level description of the circuit. The advantages of estimating the delay of a circuit from a high-level view are in the areas of high-level power estimation [1], high-level timing analysis and logic synthesis.

Our delay estimation technique is based on the novel concept of *delay measure* of a Boolean function, to be defined later in the paper. Based on this complexity measure we will provide a delay model which makes predictions of the minimum and maximum delays attainable by a circuit with reasonable accuracy. This will be demonstrated with experimental results on a large set of benchmarks. An overall high-level delay estimation flow is given in Fig. 1. In our work, the model characterization is done using SIS, which uses a simple static timing analysis to estimate the delay. This, however, is not a limitation of the proposed model, which can be used in conjunction with more accurate timing models.

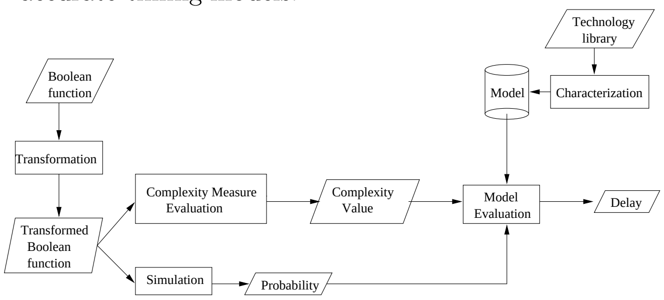

Figure 1. High-level delay estimation flow.

The proposed delay model, however, is inherently limited to circuits which do not include large exclusive-or arrays. One way around the problem of exclusive-or arrays is to require that the Boolean function specification explicitly list exclusive-or gates. In that case, these can be identified up-front and excluded from the analysis, so that the proposed method is applied only to the remaining circuitry. In any case, in this paper we will not consider circuits composed of large exclusive-or arrays.

Before leaving this section, we should mention some previous work on high-level delay estimation. In [2], a delay model is proposed for randomly generated Boolean functions (RGBF) (functions in which each point in the Boolean space is placed in the on- or off-set in a random fashion) based on empirical data collected on these functions (n = 8, 9, 10. Here n is the number of primary inputs). Their model relates the delay of a RGBF function to its entropy and primary-input count and exhibits an *exponential* in n behavior, leading to unrealistically large predictions in delay on realistic circuits. In [3], the authors propose a hierarchical timing estimation method. Their method relies on the concept of conditional delay matrix, which along with the delay information also contains conditions under which the delay information is true. However, in order to build a hierarchical model, each module is assumed to be accurately characterizable, which is unreasonable for random logic modules at higher levels of abstractions (as they are specified as Boolean equations). In [4], an approach for early layout-driven delay prediction is proposed, which includes the effect of wiring on the delay of a design. For an RT level design, this approach assumes that

† This work was supported in part by Intel, by Rockwell, and by the NSF under CAREER Award MIP-9623237.

the pin-to-pin delays of the black-box combinational circuits are already known. In [5], the authors perform technology-independent synthesis operations on the RTL design to map it to a technology-independent library. Curve fitting techniques, using representative previous designs, are then used to relate the delays and areas of the technology-independent generating functions, and hence the overall function, to that of their actual implementations. Several technology-independent optimization steps may have to be performed to obtain the area-delay trade-off curve. Our approach does not resort to any optimization and is a one-shot scheme to predict the area-delay curve.

2. The Delay Complexity Measure

The problem of estimating the delay of a Boolean function pertains to estimation of the delay associated with the slowest path ($\mathcal{D}$), also called the critical path, in an implementation of the function, given only its high level description (Boolean equations) and a target technology library. It must be noted here that by "implementation" we mean a multi-level implementation of the Boolean function, optimal in a certain sense.

Let us, for now, assume that the function at hand is a single-output function. Also, let us assume for now that we are interested in predicting the minimum delay associated with a Boolean function, given its high-level description. In the results section (section 4), we will show that the proposed complexity measure along with the delay model (section 3) can be used to predict the delay at any point on the area-delay trade-off curve.

We will define a *delay measure* of a Boolean function, in terms of delay complexity measures associated with the on and off-sets of the function. For a single-output Boolean function f, let C^{on} be a minimal cover [6] of the on-set of f. Also let $c_i \in C^{on}$ denote a cube of C^{on}, i.e., a prime implicant, for $i = 1, 2, \ldots, |C^{on}|$, where $|C^{on}|$ is the size of C^{on} (number of cubes). Finally, let $|c_i|$ denote the size of cube c_i, defined as the number of literals in c_i. With this, the delay complexity measure of the on-set of f, denoted $\mathcal{M}_1(f)$, is defined as:

$$\mathcal{M}_1(f) = \log_2\left(\max_{c_i \in C^{on}} |c_i|\right) + \log_2\left(|C^{on}|\right) \qquad (1)$$

The intuitive reason for the above definition is the following. The size of a prime implicant of a function directly influences the delay associated with the function, as the larger the number of literals in the implicant, the longer the time it would take for a function to be evaluated. Also, the number of prime implicants that need to be evaluated influences the delay, as the larger the number, the longer it would take to combine the results of the individual prime implicants. Likewise, we can define the delay complexity measure of the off-set of f, denoted $\mathcal{M}_0(f)$. Using the delay complexity measures of the on and off-sets of f, we define the *delay measure* of f as follows:

$$\mathcal{M}(f) = \min\left\{\mathcal{M}_1(f), \mathcal{M}_0(f)\right\} \qquad (2)$$

This measure requires the computation of a minimal cover of a (single output) Boolean function, which was obtained using the heuristic minimizer ESPRESSO [6].

2.1. Extension to Multi-Output Functions

We now propose a method by which the above delay complexity measure (defined for single-output functions) can be extended to measure the delay of multi-output functions. We transform an n-input, m-output Boolean function, f, by the addition of an m to 1 multiplexor, into an equivalent $(n + \lceil \log_2 m \rceil)$ input, single-output function $\hat{f}$, where the $\lceil \log_2(m) \rceil$ inputs correspond to the control inputs of the multiplexor required to control the m outputs of the Boolean function f.

Since $\hat{f}$ is a single-output function, its delay complexity measure can be computed. However, in order to compute the delay of f from the predicted delay of $\hat{f}$, we must be able to compute the influence of the multiplexor on the delay of $\hat{f}$. This problem of recovery of delay of f from that of $\hat{f}$ will be discussed in the next section.

3. The Delay Model

In this section we present the delay model to compute the delay, $\mathcal{D}(f)$, of Boolean functions. The delay model uses the delay measure of the Boolean function along with its output entropy to estimate the delay of the function at hand. We begin by presenting the delay model corresponding to the prediction of delay associated with the minimum delay implementation.

For a given n, consider the set of all Boolean functions on n inputs *and* whose output entropy is $\mathcal{H}(f) = 1$, based on all inputs being independent and with 0.5 probability. For a number of RGBFs from this set, we computed their delay measure, $\mathcal{M}(f)$, using our algorithm and obtained an estimate of the delay $\mathcal{D}(f)$ from a minimum delay implementation of the function using SIS. It was observed that the set of RGBFs for each n is clustered around specific points. Moreover, the variation of delay $\mathcal{D}(f)$ of a minimum delay implementation with delay measure, $\mathcal{M}(f)$, was linear. We have found that not only do RGBFs fall on this linear curve, but also typical VLSI functions fall on it or close to it, but spread all over the curve. Thus these linear curves of $\mathcal{D}(f)$ (of minimum delay implementation) versus $\mathcal{M}(f)$ are very important and are in fact the essence of our minimum delay prediction model.

We generate a family of such curves capturing the variation of $\mathcal{D}$ with $\mathcal{M}$ for various values of output entropy. Hence given the output entropy of a Boolean function and $\mathcal{M}(f)$, we use the curve corresponding to that value of output entropy and predict $\mathcal{D}(f)$. Note that these curves need to be generated only once, which is an up-front once-only cost, and they can be used to predict the minimum delay of various functions. An important consideration is what the largest n should be for which these curves need to be generated. Obviously, the curves are going to be more difficult to generate for larger n because of the cost of running synthesis to obtain the $\mathcal{D}(f)$ values. Luckily, there are two reasons why this is not a problem so that considering $n \leq 12$ is sufficient. Firstly, we have found that for typical VLSI functions, the value of $\mathcal{M}(f)$ turns out to be much smaller than n in most cases. Indeed, all the test cases that we will present (for which n ranges from 4 to 70) had $\mathcal{M}(f) \leq 13$. Secondly, the curves corresponding to minimum delay seem to exhibit a linear variation of minimum delay with delay measure and given by

$$\mathcal{D}(f) = k_1\mathcal{M}(f) + k_2 \tag{3}$$

where k_1 and k_2 are the slope and intercept of the minimum least squares fit.

So far, we have discussed the delay model for predicting the minimum delay. A similar reasoning can be applied to derive the model for predicting the maximum delay or any other delay. For example, we found that the maximum delay versus delay measure curves exhibit a slightly nonlinear variation, for which the best fit was an exponential:

$$\mathcal{D}(f) = k_1\beta^{\mathcal{M}(f)} \tag{4}$$

where k_1 and β are constants ($\beta \approx 1.3$), which can be obtained by performing a least squares fit.

For a multi-output function, we apply the delay model on $\hat{f}$. We recover the delay of f from that of $\hat{f}$ by using the following empirical model:

$$\mathcal{D}(f) = \mathcal{D}(\hat{f}) - \alpha\mathcal{D}_{mux} \tag{5}$$

where $\mathcal{D}_{mux}$ is the delay of an m to 1 multiplexor, and $0 \leq \alpha \leq 1$ is a coefficient that represents the contribution of the multiplexor to the delay of $\hat{f}$.

Let A_i denote the number of (multiplexor) control inputs in the ith prime implicant, and define A_{on} to be the average number of control inputs in a prime implicant belonging to the on-set of $\hat{f}$, so that:

$$A_{on} = \frac{\sum_{i=1}^{K_{on}} A_i}{K_{on}}$$

where K_{on} is the number of prime implicants in the on-set of $\hat{f}$. Similarly, one can define A_{off}. It can be shown [1] that the delay measure of an m to 1 multiplexor is given by:

$$\mathcal{M}_{mux} = \log_2(\lceil \log_2 m \rceil + 1) + \log_2 m$$

and that the delay measure of a reduced multiplexor (in an optimized version of $\hat{f}$) is given by [1]:

$$\mathcal{M}^{red}_{mux} = \log_2\Big\{\min\{A_{on}, A_{off}\}+1\Big\}+\min\{A_{on}, A_{off}\}$$

Then α, at the minimum-delay point, can be computed as [1]:

$$\alpha = \frac{k_1\mathcal{M}^{red}_{mux} + k_2}{k_1\mathcal{M}_{mux} + k_2} \tag{6}$$

The same approach can be used to compute the delay contribution of the reduced multiplexor at the maximum or any delay point.

4. Implementation and Results

Empirical results on delay estimation will be reported for three delay points on the area-delay trade-off curve, namely, the minimum, maximum and 50%-delay (delay point midway between minimum and maximum delays) points. The proposed delay models corresponding to these three points were tested on 31 ISCAS-89 [8] and MCNC [9] benchmark circuits, which are listed in Table 1. For each circuit, the table also includes the total execution time (in CPU seconds) required to predict the delay for that circuit using our approach, on a SUN Sparc-5 machine with 32MB RAM. Also, it includes the time taken by SIS to synthesize the circuits at the minimum area and minimum delay points. It must be noted that while a new run of SIS is required to synthesize a circuit to a new point on the area-delay curve, only one run of our tool would suffice to make predictions regarding the delay and area [1] at any point on the area-delay curve.

TABLE 1.
Benchmarks and run-times (sec).

Circuit Name	Circuit Function	#I	#O	CPU time	SIS time min-area	SIS time min-delay
s298	Controller	17	20	4.4	18.1	44.9
s386	Controller	13	13	4.2	22.1	58.3
s400	Controller	24	27	8.5	20.7	59.3
s444	Controller	24	27	8.5	20.6	60.3
s510	Controller	25	13	6.9	37.1	118
s526	Controller	24	27	10.4	26.3	72.6
s526n	Controller	24	27	10	26.0	71.3
s641	Controller	59	43	41.4	29.1	83.5
s713	Controller	58	42	42.3	29.1	88.4
s820	Controller	37	24	16.3	57.3	168.8
s832	Controller	37	24	16.5	57.3	160.4
s953	Controller	39	52	38.8	64.5	223.5
s1196	Logic	28	32	163	284	670.9
s1238	Logic	28	32	141	323	782.1
s1494	Controller	27	25	26.8	155.4	601.2
s1488	Controller	27	25	29.3	136.8	432.1
b9	Logic	41	21	5.7	10.6	29.8
c8	Logic	28	18	4.9	9.9	32.1
frg2	Logic	143	139	268	201.6	501.0
vda	Logic	17	39	39.3	278	497.2
i7	Logic	199	67	23.1	54.8	221.1
i6	Logic	138	67	17.5	40.2	151.5
example2	Logic	85	66	28	23.6	97.8
cht	Logic	47	36	6.5	10.8	35.4
x1	Logic	51	35	12.8	27.0	89.4
ttt2	Logic	24	21	6.25	20.4	65.9
x3	Logic	135	99	53	76	278.7
x4	Logic	94	71	28.6	50.0	108.1
apex6	Logic	135	99	45.3	62.0	236.9
apex7	Logic	49	37	20.3	17.8	61.0
k2	Logic	45	45	170.1	591.0	1352.6

In practice we have found that there is a trade-off between run-time and the value of m (number of outputs to which the transformation is applied). A heuristic is to partition the outputs of a function into groups of 16 outputs each (of course, one group may be left with less than 16 outputs), and to apply the delay prediction to the resulting sub-functions, one at a time, before recombining the results to get the delay of the whole function. If we denote by g_i the sub-function corresponding to the ith group of $m = 16$ outputs, we propose the following model for recombining the results:

$$\mathcal{D}(f) = \max_i\{\mathcal{D}(g_i)\} \tag{7}$$

In order to generate the minimum delays, the circuits were optimized in SIS using *script.rugged* followed by *script.delay* [6, 7] for delay optimization, and mapped using the library *lib2.genlib*. A comparison of the actual delay with the predicted delay obtained using the model (7), for the benchmark circuits, is given in Fig. 2. It can be seen that the overall delay can be predicted with reasonable accuracy. In order to generate the maximum delay, the circuits were optimized in SIS using *script.rugged* for area optimization, and mapped using the library *lib2.genlib*. For the 50%-delay point, this was followed by speeding up the circuit using the command *speed_up* [7] in the SIS environment. A comparison of the actual maximum and 50%-delays with the predicted delays obtained using the model (7), for the benchmarks, is given in Fig. 3 and Fig. 4, respectively. The overall delay prediction is reasonably accurate for all except two circuits (*i6* and *i7*). This is due to the fact that these circuits have a small amount of control logic which controls all the outputs. Thus this method may not be suitable for circuits which have both control *and* data in them. Also note that the above is not a problem for the minimum delay point because the best way to reduce the delay is to ensure that the same control circuit is not overloaded by feeding multiple outputs.

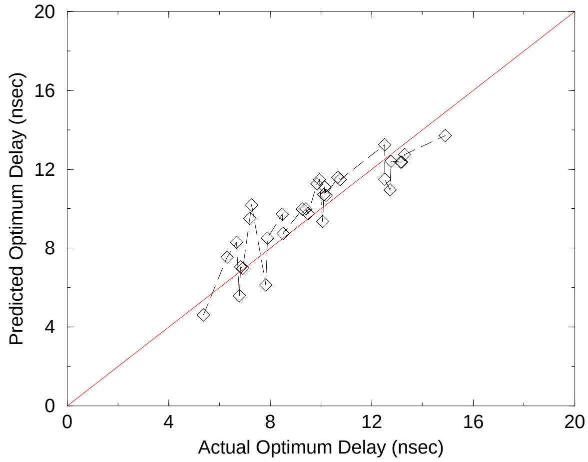

Figure 2. Actual versus predicted optimum delays.

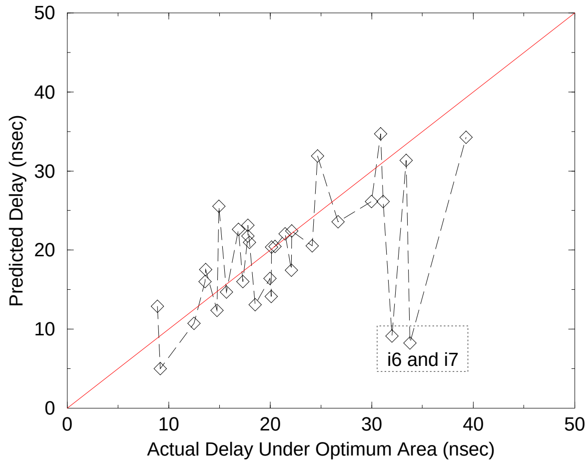

Figure 3. Actual versus predicted maximum delays.

5. Conclusions

In this paper, we have presented a new delay model to predict the delay of combinational logic, given only its functional specification and no structural information. This was achieved through a new delay complexity measure called *delay measure*. Empirical evidence showing the utility of the proposed approach in estimating any feasible delay from the given high-level description has been presented. While this approach is being further developed, certain potential limitations of this method were pointed out for circuits with control and data, with the same control logic controlling all outputs.

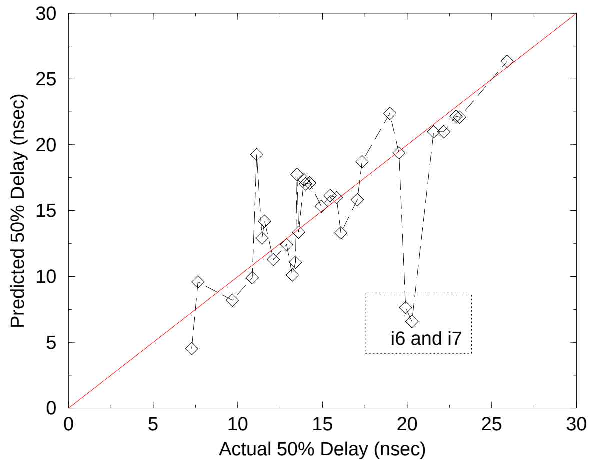

Figure 4. Actual versus predicted 50% delays.

References

[1] M. Nemani, "High-Level Power Estimation," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1998.

[2] K-T Cheng and V. Agrawal, "An entropy measure for the complexity of multi-output Boolean functions," *27th ACM/IEEE Design Automation Conference*, pp. 302–305, 1990.

[3] H. Yalcin, J. Hayes and K. Sakallah, "An approximate Timing Analysis Method for Datapath Circuits," *IEEE International Conference on Computer Aided Design*, pp. 114-118, 1996.

[4] C. Ramachandran and F. Kurdahi, "Combined Topological and Functionality-Based Delay Estimation Using a Layout-Driven Approach for High-Level Applications," *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, vol. 13, no. 12, pp. 1450-1460, Dec. 1994.

[5] A. Srinivasan, G. D. Huber and D. P. LaPotin, "Accurate Area and Delay Estimation from RTL Descriptions," *IEEE Transactions on VLSI systems*, vol. 6, no. 1, pp. 168-172, March 1998.

[6] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, New York, NY: McGraw-Hill Inc., 1994.

[7] *SIS-1.2, Reference Manual*, University of California, Berkeley, 1992.

[8] F. Brglez, D. Bryan and K. Koźmiński, "Combinational profiles of sequential benchmark circuits," *IEEE International Symposium On Circuits and Systems*, pp. 1929-1934, 1989.

[9] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," *Rep. Microelectronics Center of North Carolina*, 1991.