A Fast And Accurate Technique To Optimize Characterization Tables For Logic Synthesis

John F. Croix
Advanced Micro Devices, Inc.
5900 E. Ben White Blvd, MS 615
Austin, TX 78741

D. F. Wong
The University of Texas at Austin
Department of Computer Sciences
Austin, TX 78712

Abstract

Cell characterization data is used by synthesis and timing verification tools to compile and validate a cell netlist which meets timing constraints imposed by the designer. Characterization tables contain data for multiple, simple equations representing a cell’s behavior and are an alternative to the single, monolithic characteristic equation. Data in the table is fit to a function whose form is fixed by the application, and the cell’s response is interpolated from the function. Tables can potentially increase accuracy, but large tables can cause a program to use dramatically more memory and run much slower. The optimization of characterization tables, in which accuracy is maintained but table size is significantly reduced, is important if large programs, such as synthesis, are to complete accurately and in a reasonable runtime. In this paper we address some of the issues involved in characterizing cells and optimizing characterization tables quickly and accurately. Experimental results from the use of these techniques within AMD for a Synopsys cell library is also presented.

1 Introduction

Accurate cell characterization is critical to meeting performance goals in deep-submicron designs. This paper addresses the issue of accurately modeling the propagation delay and output transition time of standard cells in such a cell library. Typically a single equation is used to describe the behavior of a cell, both in terms of cell pin-to-pin delay as well as the transition time of the signal at the output pin. Often these equations are quite complex in order to model the inherent nonlinearities exhibited in measured data. Characterization tables, on the other hand, tend to use many relatively simple, fixed-form equations to describe a cell’s behavior, embedding the nonlinearities within the table itself, in the same manner piecewise linear approximations can represent complex functions. A $6 \times 6$ table, for example, provides 25 different equations to describe a cell’s behavior.

The primary thrust of the paper deals with a new technique to maintain the accuracy of optimized characterization tables. The typical approach is to measure and record all responses of a cell to varying input transition times and output capacitive loads. We advocate the use of dynamic programming and oversampling to filter out the more linear regions of the response surface. By judiciously picking the characterization points to store from the response surface, we can maintain the accuracy of the large characterization table while reducing the size of the table.

2 Background

In its most basic form, cell characterization is the process of applying a voltage stimulus to an input pin, placing a capacitive load ($C_i$) on an output pin, and measuring the propagation delay ($t_d$) through the cell and rise/fall times on the output pin ($t_{LH}$ for a low-to-high transition or $t_{HL}$ for a high-to-low transition). Transition times, both input and output, are measured across the more linear portion of the voltage response. Typically transition times are measured between the 10% and 90% points or the 20% and 80% points. Propagation delays and output transition times are thus a function of the capacitive load ($C_i$) and input voltage transition time ($t_{LH}$ or $t_{HL}$) [5].

Because cells in the design do not receive truly linear voltage inputs, the characterization model usually contains some type of driving cell (usually an inverter or buffer). A linear input is used to drive this cell, and a capacitor on the output pin ($C_{rise/fall}$) is used to alter the input transition time. This model is shown in figure 1. The process of characterizing the cell is now one of sweeping across a range of capacitive loads on the buffer and a range of capacitive loads on the output pin of the cell being characterized. The input transition times, propagation delays, and output transition times are then measured. Usually the results are fit to a characteristic equation, using some generalized curve-fitting method such as least-squares, or to a characterization table.

![Figure 1: Use of voltage-controlled-voltage-source (VCVS) to duplicate the voltage across $C_{rise/fall}$ on the input pin](image)

In order to accurately represent the measured data, characteristic equations need to have some nonlinear element to represent the inherent nonlinearities in the circuit response. Thus the equations can include product and log terms on the function variables and can range from just a few coefficients to many (20 or more). Another...
approach is to model the driver as a parameterized voltage supply and linear resistor [3].

3 Characterization Tables

An alternative to the monolithic characteristic equation is to use a linear characteristic equation and a table of values to represent the characterization data [4]. This method is akin to the piecewise linear approximation of a curve. Measurements are taken at specific input transition times ($t_{LH}$ and $t_{HL}$) and capacitive loads ($C_i$) and the values stored into a table. When determining the response of the circuit for a specific transition time and load, the four points in the table that bound the desired point are determined. Using these four values, the coefficients of the characteristic equation that describes the cell’s behavior in that localized region are determined. Once the unique equation has been determined, the transition time ($t_{LH}$ and $t_{HL}$) and load ($C_i$) are used to interpolate the cell’s response.

As an example, consider cells $R_{ij}$ in the response table, $R$, represents two potential indices ($i$ for the transition time and $j$ for the capacitive load) in the reduced table. Stated another way, any cell $R_{ij}$ may be the lower left corner of a rectangle which defines the reduced table area coverage. Therefore, the first step is to determine the error associated with choosing cell $R_{ij}$ and any cell above and to the right as the bounds on a reduced table entry. This is done by calculating the coefficients describing the corners of this rectangle, interpolating for any points in the rectangle, and summing the difference between the interpolated values and the measured values from SPICE.

3.1 Table Size vs Accuracy

Program memory and runtime constraints place practical limits on the size of the tables. Obviously, smaller tables are less of a drain on system resources than larger tables. The data stored in the table should be selected such that the interpolation across them yields highly accurate response times. There is no guaranteed method, however, to determine the capacitive loads and response times should be used to characterize a cell before the data is generated. One method to solve this problem is to make more characterization measurements than will be used in the table.

Because linear interpolation is performed to determine the response time, any row (column) in the table whose values can be linearly interpolated by the row (column) above and below (left and right) need not be present. Thus, the table size can be reduced by removing that row (column). An alternative argument says that the space used to store such a row (column) can be used to store a different row (column), resulting in increased accuracy. Thus the goal in picking the transition times and capacitive loads to serve as indices is to pick those which reduce the global interpolation error or which reduce the table size within an error bounds.

3.2 Error Calculation

In trying to determine which indices in a large table, say $50 \times 50$, to use in the final, reduced table, the error must be calculated for those indices used in the reduced table and compared against any other potential indices. Thus, any cell $R_{ij}$ in the response table, $R$, represents two potential indices ($i$ for the transition time and $j$ for the capacitive load) in the reduced table. Stated another way, any cell $R_{ij}$ may be the lower left corner of a rectangle which defines the reduced table area coverage. Therefore, the first step is to determine the error associated with choosing cell $R_{ij}$ and any cell above and to the right as the bounds on a reduced table entry. This is done by calculating the coefficients describing the corners of this rectangle, interpolating for any points in the rectangle, and summing the difference between the interpolated values and the measured values from SPICE.

As an example, consider cells $R_{ij}$ in figure 3(a). The coefficients of the characteristic equation are calculated for values in cells $R_{i0}, R_{i0}, R_{i2},$ and $R_{i2},$ the four corners of the rectangle defined by $R_{i0}$ and $R_{i2}$. Next, the interpolated values are calculated for indices which correspond to cell entries $R_{i0}, R_{i0}, R_{i2}, R_{i2}, R_{i1}, R_{i2}, R_{i2}, R_{i2},$ and $R_{i2}$. The differences between each of these values and the SPICE measured values in the table is summed as the error associated with this interpolation. This calculation is done for every cell in the table to every cell above and to the right of it.

3.3 Dynamic Programming

In order to reduce the $50 \times 50$ table into a $10 \times 10$ representation, a dynamic programming solution is used [1]. In the previous step the error was calculated from any cell to another above and to the right in one step. It is now possible to estimate the error from any cell

$$
\begin{bmatrix}
  t_r^- & C_i^- & 1 \\
  t_r^+ & C_i^+ & 1 \\
  t_r^- & C_i^+ & 1 \\
  t_r^+ & C_i^- & 1 \\
\end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = 
\begin{bmatrix} r_-^- \\ r_+^+ \\ r_-^+ \\ r_+^- \end{bmatrix}
$$

Once the characteristic equation has been determined, the interpolated response at the desired input transition time and capacitive load can be calculated. Figure 2 graphically illustrates the solution.
to \( R_{40,40} \) in two steps as is shown in figure 3(b). The approximate error is the error between that cell and any intermediate cell above and to the right plus the error between the intermediate cell and \( R_{40,40} \) (areas 1 and 2 in figure 3(b)). Additionally, the selection of the intermediate cell implies that its two indices would also divide the entire table, so the error associated with regions 3 and 4 must also be included.

This method of searching is performed for every cell in the table for any two steps. For any cell, the algorithm finds the intermediate cell that it can step to such that the accumulated error is minimum for all cells above and to the right of it. This value and a pointer to the intermediate cell is now stored on that cell as the path from that cell to \( R_{40,40} \) in two steps.

This process is repeated for 3 steps, as shown in figure 3(c). To go from any cell to \( R_{40,40} \) in 3 steps, the error can be approximated by the error from that cell to an intermediate cell and the error from that cell to \( R_{40,40} \) in 2 steps. Since all 2-step data has been calculated and stored, this value need not be recalculated. The selection of an intermediate cell implies the acceptance of two intermediate table indices, the error for which must also be included. This corresponds to areas 3, 4, 5, and 6 in figure 3(c).

This entire process is repeated until the data for 9 steps is calculated for every cell. It is now a simple matter to trace the path from cell \( R_{0,0} \) to \( R_{40,40} \) in 9 steps to arrive at the ten indices to use for the table. The error associated with 9 steps from cell \( R_{0,0} \) is the error associated with using these indices with respect to the large 50 \( \times \) 50 table.

3.4 A Faster Algorithm

An alternative approach is to select the indices for the input transition time in one pass and the capacitive load indices in a second. This method causes some loss in accuracy but requires significantly fewer computing resources both in terms of compute time and memory.

In this algorithm, dynamic programming is still used to determine the indices. However, instead of calculating the characteristic equation between every cell pair in the table, the equation is only generated for regions that span entire rows or columns of the table. Thus, every cell in the bottom row of the table is matched with every cell on the top row and to the right of it. The characteristic equation is generated and errors are measured. Dynamic programming is used to determine the “step points” between these equations in the same manner as previously described. The same approach is used to independently determine column indices.

4 Experimental Results

Implementation of this characterization methodology was done on Linux using C++ (G++ 2.7.2). Solution of the characteristic equation was performed using LU decomposition and forward and backward substitution [2]. Results were run on a 100Mhz Pentium system with 32MB physical memory. HSPICE was used to measure circuit response.

4.1 Traditional Characterization

In this model, the indices for input transition time (\( t_{HI} \) and \( t_{LH} \)) and capacitive load (\( C_l \)) were predetermined and encoded into the SPICE deck as the measurement points. The characterization points were picked by the cell designer and refined over several iterations.

The errors are shown in figure 4. This shows that the largest error occurs at the smaller values of transition time and capacitive load. However, this is also the most likely region of use for a given cell in a critical timing path.

![Figure 3: Traversal of the characterization. (a) Resource table. (b) Two steps from intermediate node. (c) Three steps from intermediate node.](image-url)

![Figure 4: A 3-D surface of the error percentages between the fixed index table and the 50 \( \times \) 50 table.](image-url)
4.2 Reduced Table Generation

Error values for the large $50 \times 50$ table are stored in an upper-triangular matrix due to the way the algorithm was designed. Even so, this matrix consumed 3.125 million double-word values, or approximately 25Mb of memory. Production of a $10 \times 10$ table took approximately 30 CPU minutes, 27 of which were used to calculate the errors in the initial stage. The number of steps also had an effect approximately 25Mb of memory. Production of a $25 \times 25$ table was generated from a $20 \times 20$ matrix. The first library was generated in under one week, and error margins calculated. Without any further tuning, error margins for over 200 tested critical timing paths were within $\pm 2.5\%$. CPU time overhead using this method was approximately three CPU hours.

4.3 Fast Reduced Table Generation

The assumption of independence resulted in somewhat increased error rates, but required only 10 CPU seconds to complete and under 100Kb of memory. In a constrained CPU and/or memory environment, the increase in error rates of up to 3X over the independent variable solution may be a reasonable trade-off. The error rates ($+0.5\%$ to $-0.8\%$) are still substantially below those exhibited in the traditional model which was used as the baseline. 

5 Experimental Results Within AMD

The accuracy of cell characterization data within a cell library must be determined if designers are to have any faith in synthesis results. The manner used within AMD is to compile designs using the optimized cell library. Next, many timing critical paths are extracted and SPICE netlists generated which model these paths. SPICE is run against these netlists and specific measurements recorded. Detailed Synopsys timing reports are then compared against all SPICE measured values.

Prior to the use of oversampling and pruning described in this paper, SPICE results were used to “tune” characteristic tables. Significant differences between SPICE and Synopsys timing reports would result in the selection of different input transition times and capacitive loads for cell characterization. After multiple iterations over several months, error margins could fall from $\pm 10\%$ to as low as $\pm 3\%$ for most cells.

The AMD characterization flow was completely rewritten to take advantage of oversampling and pruning of characterization data. Using the “fast mode” pruning method described previously, a $6 \times 6$ table was generated from a $20 \times 20$ matrix. The first library was generated in under one week, and error margins calculated. Without any further tuning, error margins for over 200 tested critical timing paths were within $\pm 2.5\%$. CPU time overhead using this method was approximately three CPU hours.

6 Conclusions

Applying dynamic programming to the determination of characterization table indices can result in significant accuracy improvements over a priori selection. The penalty is the larger number of SPICE runs made against a cell (2500 compared to 100 for the example in this paper) and the CPU time to apply either of the two techniques described here. The additional SPICE time is insignificant for most cells in a cell library as they tend to consist of no more than three logic levels of simple gate combinations.

We have presented two algorithms which can be applied to the characterization problem to greatly improve the accuracy of the characterization model. In environments where accuracy is the primary concern and CPU time is plentiful (for example a networked environment of hundreds of workstations with a job scheduler), interdependent variable analysis can be performed in a reasonable amount of time and memory. In a more tightly constrained environment, simplifying the model by assuming variable independence still yields very accurate results but at a much reduced CPU time and memory requirement. Either approach can yield error rates greater than an order of magnitude better than traditional “hit-and-miss” a priori methodologies to choosing the characterization points.

References