Methodology for Circuit Optimization

by

KENDRA L. MARKLE

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Bachelor of Science in Electrical Science and Engineering and Master of Science in Electrical Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1995

Author .....
Electrical Engineering and Computer Science
January 27, 1995

Certified by .....
Srinivas Devadas
Associate Professor, Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler

Methodology for Circuit Optimization

by

KENDRA L. MARKLE

Submitted to the Department of Electrical Engineering and Computer Science on January 16, 1995

in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical Science and Engineering and Master of Science in Electrical Engineering and Computer Science.

ABSTRACT

This thesis presents a methodology for optimization of static and domino logic. The optimization techniques that are discussed were developed to improve propagation delay, dissipated power and area for circuit examples while taking noise margin and other constraints into consideration. Where needed, a weighted combination of the these four concerns is used to provide realistic goals and solutions. A guideline for the trade-offs between these goals is also presented. Static and domino circuits are analyzed to reveal the interdependencies between these design goals.

Optimization is considered at three levels of resolution. Circuits which have individual transistors, P/N ratios, and entire standard cell gates parameterized are analyzed. The techniques for improving circuit performance that are discussed include recursive optimization, fast-path gates, skewed gates, and minor topology changes such as switching the logically equivalent pins of a gate, and combining and expanding small logic stages. The advantage of each method is characterized in terms of final circuit performance achieved.

Thesis Supervisor:

Srinivas Devadas
Title: Associate Professor, Electrical Engineering and Computer Science

ACKNOWLEDGMENTS

Many individuals offered assistance and support for this thesis, and I would like to offer my thanks.

I would like to thank my MIT thesis advisor Professor Devadas, and my Intel supervisor Tom Fletcher.

Thank you RW for your help.

Thanks JF for proofreading, MR for researching, and JM, SN, JL, and PB for your support.

This thesis was done at Intel Corp. under the direction of Tom Fletcher.

Contents

List of Figures

List of Tables

Introduction

1.0 Introduction

The computer industry has grown rapidly in recent years. Software applications have been increasing the demand for higher performance computer systems and microprocessor design is forced to keep up. The complexity of the circuitry that makes up microprocessors has dramatically increased, and the sheer number of transistors that comprise a single chip has increased by more than a factor of 100 in the last 12 years, from 29,000 in the 8086 to more than 3.1 million in the current Pentium Processor [13].

To keep up with the rising demand for higher performance microprocessors, chip designers take two approaches. They make architectural design changes to improve the way in which the chip performs its functions, and they improve the efficiency of the circuitry through optimization.

High performance circuit design involves a lot of conflicting objectives. The techniques that can make a processor faster can also increase the power it consumes, the space it takes up, and the cost of fabrication and production. Reliability is also of the utmost importance for computer applications and this can limit the realization of some design goals.

The circuit design process consists of several stages of representation of the processor in more and more detailed ways (see Figure 1). Optimization plays a role in several of these stages. Once the initial circuit schematics are decided on, circuit optimization can be used to improve circuit performance, as specified by the design goals. The final circuits are laid out, fabricated, and tested to verify circuit functionality and objectives.

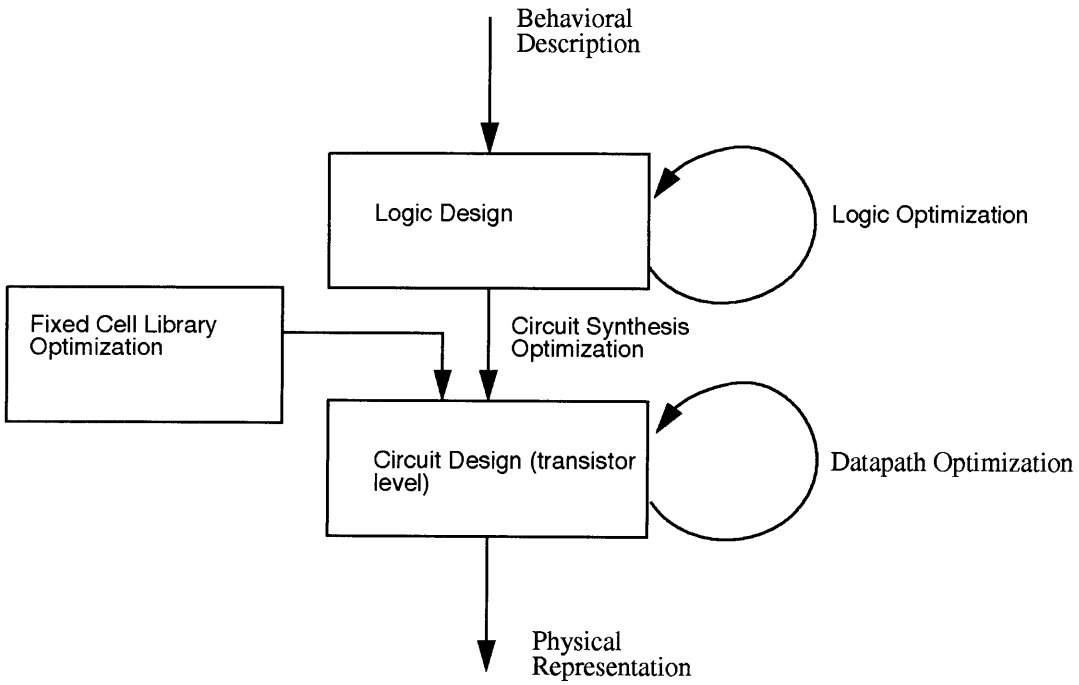

FIGURE 1. Circuit design process flow chart

Circuit designs in this day and age are complex enough that they must be constructed from defined library cell blocks, referred to as fixed cells, such as inverters, nand and nor gates, flip flops, etc. Since each design will be built almost entirely of these predefined cells, an optimized cell library is imperative. The library cells should have minimal propagation delay, conserve power, and most importantly, function well when put together in new circuit configurations.

The logic which implements a behavioral description can always be designed in many different ways. Care must be taken to produce a circuit which can meet the circuit design goals for timing, power, area, and reliability. Optimization of the topology produced in this stage can make the process of fine tuning the circuit to meet a complex set of design rules much easier.

Once the topology is determined, and the fixed cells are pieced together, the circuit's sizing must be fine tuned to make sure it meets all the design goals. This is where most optimization takes place. Any modifications from resizing entire gates to incremental transistor size changes can be made to improve performance.

Sometimes, even with the best of techniques, a circuit's performance will not be sufficient after this stage. In these cases, the process becomes iterative and more changes to the topology can be made. Unfortunately, this forces reoptimization of the gate sizes and can add quite a bit of time to the optimization process, which is why it is important to come up with a good topology early on.

1.1 Circuit Design Goals

Circuits today are bigger and more complex than before, and as a result, there are many more design goals and constraints that must be met by each circuit at each stage of the design. Increased performance requirements demand that every circuit be faster than before and this creates more timing considerations. Circuit blocks that interface together place more stringent timing demands on each other, in order to force the total propagation delay of each signal to be minimized. Circuits that are pipelined are frequency limited by the longest path from input to output, even if all the other paths are considerably faster, making optimization play an even more important role.

As die sizes continue to grow, area becomes an important consideration also. No longer can devices be maximum sized just to increase speed without consequences. Sizing and layout must be carefully designed to keep the circuit area down. If a chip is large, it costs more in resources to fabricate and fewer chips will fit on a wafer. Also, the yield, the ratio of the number of fully functional chips that are fabricated to the number of flawed or non-functional chips, is lower. More attention must be payed to trade-offs between area and delay and area needs to be conserved whenever slack exists in the delay specifications.

Power, also, is becoming a factor in large scale designs as system cooling becomes an issue. Wasted power must be eliminated at all costs. What used to be considered necessary power, the amount of power needed to actually make the circuit function, is now being reevaluated. Powering down sections of the circuit that are not being used and reducing

sizing at the expense of delay are now considered common practice, and necessary to reduce the power consumed to acceptable levels.

The more complex a circuit is, the more complicated and numerous the associated constraints become. These constraints must be kept in mind throughout the design process, as the finished circuit design must meet them all. More specifically, circuits must meet maximum and minimum sizing limits and sizing ratios to be considered reliable. The circuit must reach its design goals over an applicable range of temperatures and frequencies, and must be able to withstand realistic power fluctuations and fabrication variations. All of these factors must be taken into account during the design process in order to produce a robust design.

Designers are still struggling to make circuits meet all of the requirements, even as the list of considerations for the design of a circuit continues to increase. Many times optimization is only used if one of the critical goals is not met, and only to fix that one requirement. Often, a circuit will meet its required goals but will still be far from optimal. These untapped performance benefits would surface if an organized optimization methodology existed that could bring them to the designer's attention and help fix them. Circuit designers need to be aware of the performance increases that exist for different types of circuits and how to go about obtaining these increases. This thesis documents an optimization methodology to better enable this to happen.

This thesis will discuss optimization of static CMOS and domino circuit types, but will not cover BiCMOS circuits, passgates or array optimization. Techniques for logic, fixed cell library and datapath optimization will be discussed, in the context of power, area, delay and reliable noise margin design goals. Circuit synthesis and physical design considerations will not be included. Algorithms will not be considered in terms of minimizing compute time. Exotic and risky design techniques will also not be covered.

1.2 Successful Optimizations

Using the optimization algorithms discussed in this thesis, many circuit design goals have been realized, and solutions found. Hours of hand simulation and excessive iteration were saved.

The circuits where optimization is most effective tend to have several aspects in common. When few trade-offs exist, there are few parameters in the circuit, or the design goals are simple, an optimal design seems to be found much more easily. Also, circuits which are completely specified and bounded have a smaller solution space and tend to have a more obvious optimal point.

For example, when trying to determine the optimal skew to be used in a chain of CMOS gates, the number of parameters can be reduced to one. The size of the NMOS transistors in each of the gates can be fixed at one value, and the PMOS size allowed to vary, effectively parameterizing the skew of the gates. There is one simple design goal:

minimize the propagation delay through the chain of gates for a rising and a falling output transition. The problem is bounded, since only realistic P/N ratios should be considered. There is only one major trade-off. A faster circuit will use more power, but reducing the power means reducing the speed.

Another good example of when optimization played an important role was in the design of the clock distribution tree for a recent microprocessor. A recursive strategy was used to design first the parts of the tree that drove the required circuit blocks, thereby specifying exact loading for the last stage, and then completing the design stage by stage back to the generation of the clock signal. A clock distribution design such as this one has the stringent requirement that clock skew between any two outputs be minimal. To design the tree by hand, simulation would be forced after each stage of the circuit was added to make sure the skew had not increased above a certain small value, taking up a lot of time, and slowing the entire design process down. By using an optimization program, these simulations can be performed, clock skew can be measured and compared and redesign can be implemented automatically with little interaction with the designer once the restrictions, goals, and algorithm have been specified.

1.3 Background Information

A generalized flow chart for optimization includes generating an initial design, deciding on design goals and circuit parameters, prediction of a set of potentially better circuits,

simulation, comparison of results, possible iteration, and production of the best results (see Figure 2).

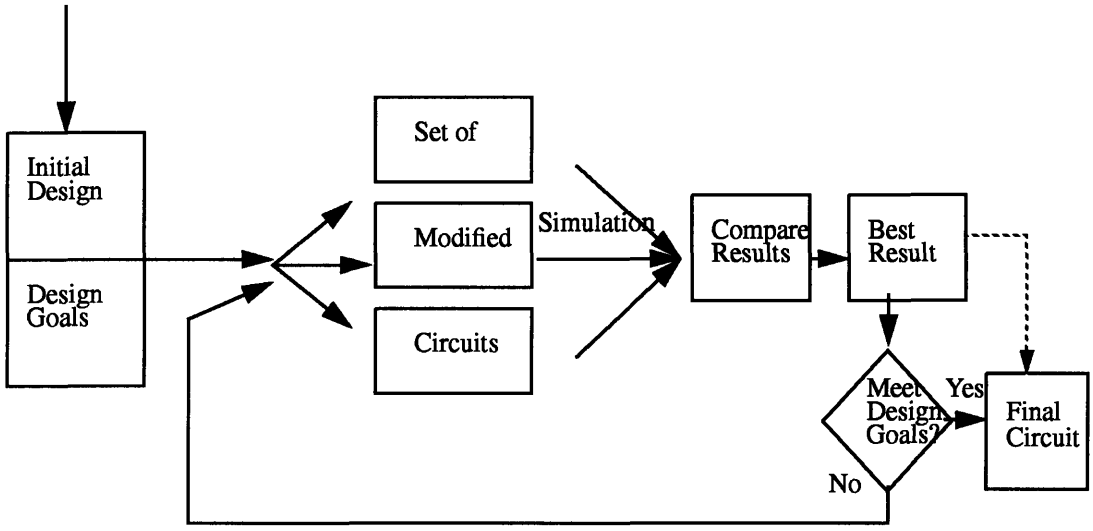

FIGURE 2. Optimization flow chart

This process can be and often is performed by hand, though that can be slow and tedious. Instead, intelligent CAD tools can make optimization much faster and more efficient by automating the process. These tools have the ability to vary the circuit parameters in accordance with a specified algorithm. Working with a circuit simulator such as SPICE, they can simulate modified circuits and compare the resulting performance changes, as well as determine when a produced circuit is optimal.

1.4 Documented Optimization Algorithms

Over the years, many algorithms have been applied to circuit design, each with its own set of advantages and limitations. They differ in how they sample the design space and in the way in which they determine what part of the search space to simulate next. They each use either the minimax [1] or the weighted sum [4] method of incorporating many design objectives into one mathematical equation to help determine the optimal design while taking all the goals into account. Some search globally over the entire solution space, whereas some use methods which search only locally to the initial design, but usually with much more precision. Some allow the parameters to assume discrete values, and some require continuously varying parameters. Some are 'greedy' algorithms; they can only move to a better design from the current 'best' design, and some have no memory of the previous set of designs, and only consider the current set. Some must be actively constrained to cut down on the solution space, and some cannot take constraints into account.

Circuit design optimization is a non-linear multi-variable multi-objective constrained optimization, and requires CAD tools that can take these characteristics into consideration. Algorithm choice can make a significant impact on the specific solution returned, the solution precision, the time taken to converge to a solution, and whether the solution is really 'optimal' according to the design goals.

Some of the more well-know algorithms used in circuit design optimization are briefly described below [1][4][7].

1.4.1 Random Grid Search

Defines and bounds entire solution space in N dimensions, where N is the number of parameters being optimized, by bounding each of the parameters. Uses an error function as comparison criterion, and randomly simulates points in the space, returning the 'optimal' design.

1.4.2 Steepest Descent

A 'greedy' algorithm that follows a gradient vector, computed by taking the derivative of the least squares error function with respect to each of the defined parameters. It will stop at a local minimum, since it cannot make uphill moves, so the 'optimal' result depends on the initial conditions. It is limited by active constraints. Variables can be made discrete, or can vary continuously.

1.4.3 Simulated Annealing

Performs random sampling in the solution space, and keeps track of the best design. Can only find near-optimal solutions.

1.4.4 Newton-Raphson

Forms a Jacobian matrix of partial derivatives of the sum of the weighted squared differences with respect to each parameter N. The inverted matrix is used to determine a vector which will make the error as close to zero as possible. It has no memory of data from previous steps.

1.4.5 Gauss-Newton

Samples the steepest descent gradient, then projects a concave up parabola onto the solution space based on the gradient, and moves to the minimum of the parabola. Converges quickly when near the optimal solution.

1.4.6 Levenberg-Marquardt

A modification of the Newton-Raphson and Gauss-Newton strategies that converges even when initial values are very far from the optimal solution.

1.4.7 Hybrid Strategies

Many programs use more than just one algorithm to exploit the benefits of each one while working around the associated limitations. For example, it can be useful to use an algorithm which searches globally and does not get trapped by local minima until the general area in which the optimal design point lies is identified. Then, when the first algorithm becomes less effective, a new algorithm which is highly accurate and converges rapidly when close to the optimal point can be applied.

1.5 Terminology

When comparing circuit waveforms, it is useful to refer to the *propagation delay* between two points in a circuit. This refers to the difference between the time at which the first signal waveform crosses half of the power supply value (or *Vcc*) to the time at which

the second waveform crosses half of Vcc. When the second signal is making a high to low transition, this propagation delay is commonly referred to as *tphl*. When the second signal is making a low to high transition, it is referred to as *tplh*. The slope of the waveforms can be described by *rise time*, the time it takes a wave to move from 10% of Vcc to 90% of Vcc and *fall time*, the time it takes a wave to move from 90% of Vcc to 10% of Vcc.

Chains of gates can be described in terms of their *stage ratio*. This refers to the factor by which a gate is larger than the previous gate. The *fanout* of a gate refers to the ratio of a gate's load capacitance to its input capacitance. The *skew* of a gate is equal to its effective PMOS device size divided by its effective NMOS device size.

A *path* through a circuit refers to the connections from an input through gates and interconnect to an output. A *critical path* is a path that has an unacceptably long or relatively long propagation delay from input to output.

Optimization Circuit Types

2.0 Optimization Circuit Types

Different kinds of circuits require slightly different methods of optimization, since the circuit design goals will differ. The timing requirements and limitations placed on the circuit will also change with circuit type. Optimizing simple generic gates to create a fixed cell library, for example, has entirely different goals than optimizing domino circuits to reduce precharge and evaluate phase times or static CMOS circuits to reduce critical paths. Each of these three mentioned circuit types is commonly used and usually customized optimization can provide large performance boosts.

2.1 Optimizing a Library of Fixed Cells

Each cell that is included in a defined library earns its place there by nature of the fact that it will be incorporated in larger designs many times over. It is important to carefully

check these cells to make sure they are perfect, since any flaw in a gate will occur each time it is used. However, this also means that an optimized cell library can produce circuits that are significantly better than circuits from unoptimized libraries.

Library cells should individually take into account such design goals as minimal gate propagation delay and reduced power consumption. The library as a whole must be designed so that the cells work well when cascaded together.

It has been assumed for some time that a standard inverter should be ratioed such that the high to low propagation delay was roughly equal to the low to high propagation delay. This kind of inverter does not favor one transition over the other, and can be used generically throughout circuits without fear that it will skew the waveforms that pass through it. This is extremely important for clock distribution circuits, for example, where skew must be minimized. In more tolerant circuits, however, a different scheme is much more effective. If fixed cell gates are designed so that the low to high propagation delay is slightly slower than the high to low propagation delay, it can be shown that the overall delay through a chain of gates is less than if each of the gates had equal high to low and low to high propagation delays.

SKEW (effective P/N ratio)	tplh/tphl	Average Propagation Delay per gate
0.263	0.870	0.214
0.368	0.861	0.181
0.474	0.849	0.162
0.579	0.859	0.150
0.684	0.868	0.142
0.789	0.868	0.138
0.894	0.882	0.135
1.00	0.906	0.132
1.105	**0.913**	**0.132**
1.211	**0.917**	**0.133**
1.316	0.921	0.133
1.421	0.926	0.133
1.526	0.928	0.134
1.632	0.930	0.134
1.737	0.936	0.135
1.842	0.944	0.136
1.947	0.956	0.137
2.053	0.967	0.138

TABLE 1. Propagation delays produced by using different skews

Table 1 shows the average propagation delay per gate through a chain of inverters as the skew of the inverters is varied. The smallest delay is produced when the skew is about 1.05. At this point, the high to low and low to high propagation delays differ by roughly

10%. The P/N ratio that must be used to produce equal high to low and low to high propagation delays is significantly larger than the optimal skew.

The skew that shows the fastest propagation times is dependant on the fanout used in the circuit. A larger fanout results in a higher optimal skew. A fanout of 2.7 was used to produce the data in Table 1. A similar experiment with a fanout of 3.0 yielded an optimal skew of 1.33.

When minimizing delay through a cascade of cells from a fixed cell library, each gate must have the appropriate drive strength to charge the input capacitance of the gate that comes after it. When a small gate has to drive a large capacitance, the result is a much longer propagation delay than is necessary. It can be shown that a chain of buffers with the right stage ratio and number of stages will charge the load capacitance the fastest [11].

The number of buffers to use in the chain and the appropriate stage ratio can be determined as follows. If K is defined to be the ratio multiplication factor from gate to gate, and N is the number of buffer stages used, then the scaling ratio is given by $K = \left(\frac{C_{load}}{C_{input}}\right)^{1/N}$,where $N = ln\left(\frac{C_{load}}{C_{input}}\right)$ rounded to the nearest integer value.

The optimal stage ratio is dependant on the capacitive load ratio and on the process. For the general case, assuming these are constant, the optimal point is found to be equal to e (~2.7) [12].

It seems logical, from a minimal delay standpoint, to design a series of fixed cells with this multiplication factor of 2.7, such that each chain of gates will approach this optimal drive strength for the load which is to be driven.

It is also known, however, that this optimal multiplication factor is a shallow minimum and the multiplication factor can be reduced slightly to save area. In the design of fixed cells, saving a small percentage of area per cell can translate to a tremendous area difference in large designs.

2.2 Optimizing Static Circuits

Optimizing static CMOS circuits is not an easy task. There are many considerations and limitations. The problem can sometimes become so constrained that few realistic solutions actually exist. Often it is hard to determine what the optimal circuit configuration is because circuit design involves so many non-linear trade-offs between goals. Many dependencies exist in circuits that prevent simple modelling or generalizations from helping with the design process. Propagation delay, dissipated power and raw area are three important design goals, all of which are interdependent.

2.2.1 Static Delay

For each gate in a circuit that drives a load, there exists a range of capacitances that can be driven reliably and in a reasonable amount of time. If the load is too large, the delay is also very large and the hot electron currents can cause transistor degradation [13]. If the load is too small, both power and area are wasted. This places boundaries on the fanout of

each and every gate that gets used in the circuit. Somewhere within this range lies the optimal driver size for each load.

Fanout also has a direct influence on the output wave's slope. A small fanout allows faster output transitions (see Figure 3). Since this output is actually the input for another gate, when a fanout is too large, the slow rise and fall time of the output wave not only increases the propagation delay of the first gate, but also that of the second gate.

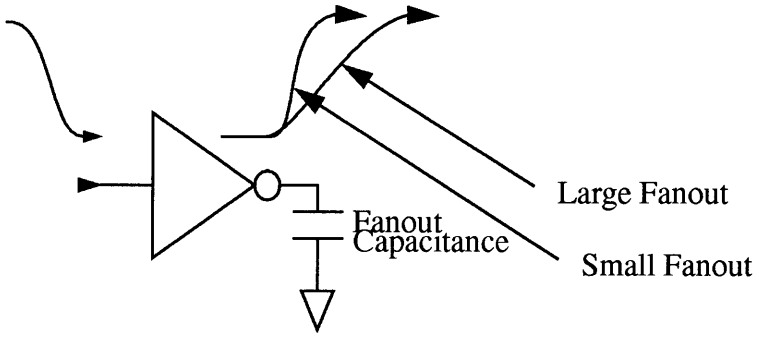

FIGURE 3. Output slope decrease produced by increasing fanout

In general, delay can be reduced by increasing transistor and gate driver sizes, thereby decreasing fanout.

2.2.2 Static Power Considerations

Power is dissipated in static CMOS circuits in both static and dynamic ways. Dynamic dissipation usually accounts for most of the total power consumed.

Static power dissipation consists of any current that is drawn constantly from the power supply and is not caused by a transition on the output. This includes reverse bias

leakage and subthreshold conduction [3] and decreases linearly with supply voltage. It also includes current drained between transitions, such as through a pseudo-NMOS device, when a DC path between power and ground exists, even if it is only for a very short time.

Static power in a circuit can be reduced slightly by using smaller transistor sizes. For the most part, it must be accepted that a certain amount of static power will be used, with few options to reduce it. Dissipated static power can be estimated using the following equation [12]:

$$StaticPower = \sum_{AllDevices} SupplyVoltage \times i_s \left(e^{\frac{qV}{kT}} - 1 \right)$$

where i_s is the reverse saturation current, $\frac{kT}{q}$ is the thermal constant, and v is the diode voltage.

Dynamic power dissipation refers to the current that is used to charge the capacitive loads and current that is dissipated when the output transitions. In CMOS circuits, this is roughly proportional to the area of the circuit scaled by an average activity factor A_F. The activity factor represents the ratio of the number of cycles during which the circuit transitions to the number of cycles during which the circuit does not change state. Lower frequencies and lower supply voltages, a lower activity factor, and fewer transistors and less interconnect capacitance to drive can dramatically reduce the dynamic power used.

When a CMOS gate transitions, there is a brief time during which both the NMOS and the PMOS devices are at least partially on, causing a direct path from power to ground. The amount of current that is dissipated here is called overlap current, and varies according to how long the devices are simultaneously conducting current and how 'strongly on' they are. The overlap current is increased when rise and fall times are long, since long transition times mean that gate devices are turned on longer. The current dissipated during switching can be reduced a great deal by decreasing the capacitance that must be driven. This current accounts for a smaller percentage of the total dynamic power used as load and routing capacitance becomes larger.

Dissipated dynamic power can be estimated with the following equation:

$$DynamicPower = \sum_{AllGates} C_L V^2 f \cdot A_F + \sum_{AllGates} OverlapCurrent \times SupplyVoltage$$

where C_L is the capacitive load on a gate's output, V is the supply voltage, f is the frequency, and A_F is the activity factor.

2.2.3 Static Area

One good way to save area when designing static CMOS circuits is to reduce the size of gates that are in non-critical paths. This can cause the propagation delay through these paths to get larger, so this technique should only be used where delay is not critical. For example, take the simple circuit shown in Figure 4. A critical path is caused by the late

arrival of the incoming signal on Input 2. The path runs from Input 2 through gate B and gate C to the output.

The timing of the signals relative to each other are shown in Figure 5, where it is revealed that one of the inputs to the nand gate becomes valid much earlier than the other input does. Here is a place where the trade-off between delay and area can come into play.

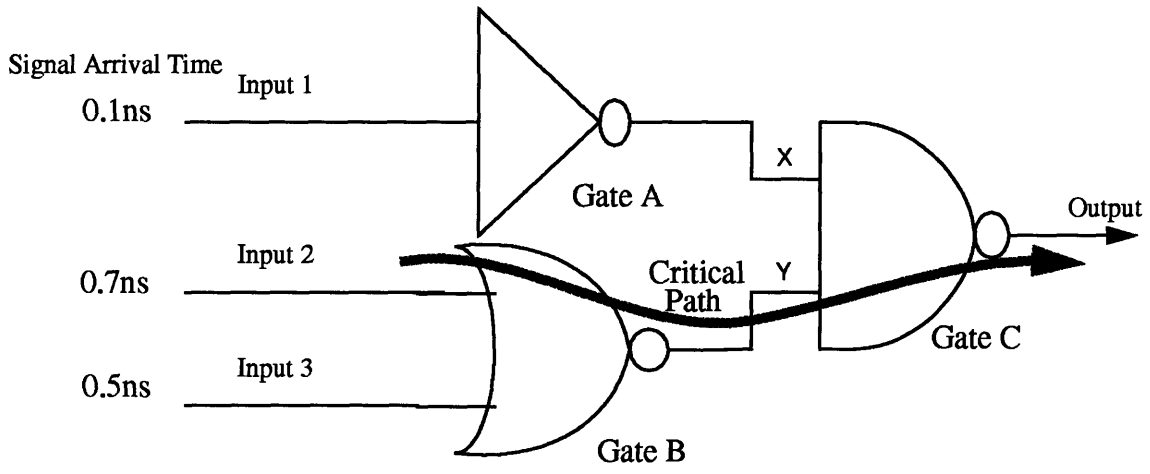

FIGURE 4. Circuit example showing a critical path

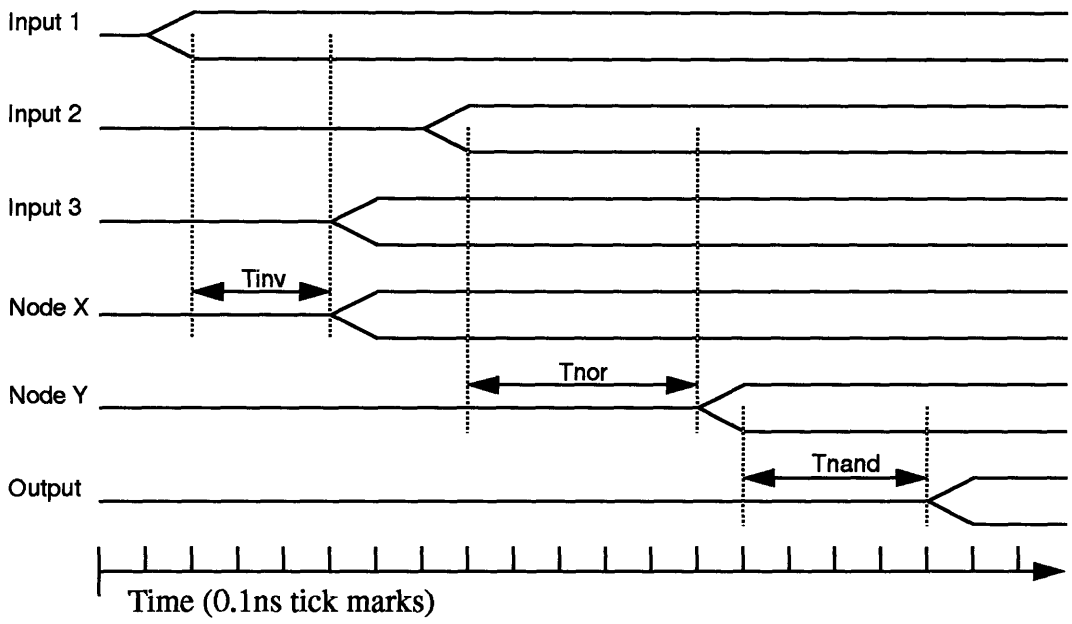

FIGURE 5. Timing diagram showing valid signal times at each node

Since gate A delivers it's output to gate C early compared to gate B, the size of gate A could be reduced. This decreases gate A's drive strength and therefore increases the propagation delay through gate A (see Figure 6). However, since there is significant leeway in this case, the timing on the output of the circuit will not be affected.

Figure 6 was generated by varying the area used to implement a fixed-skew inverter driving a fixed load. The trade-off between delay and area is fairly straight forward; in general, a larger sized gate will be faster than a similar smaller one.

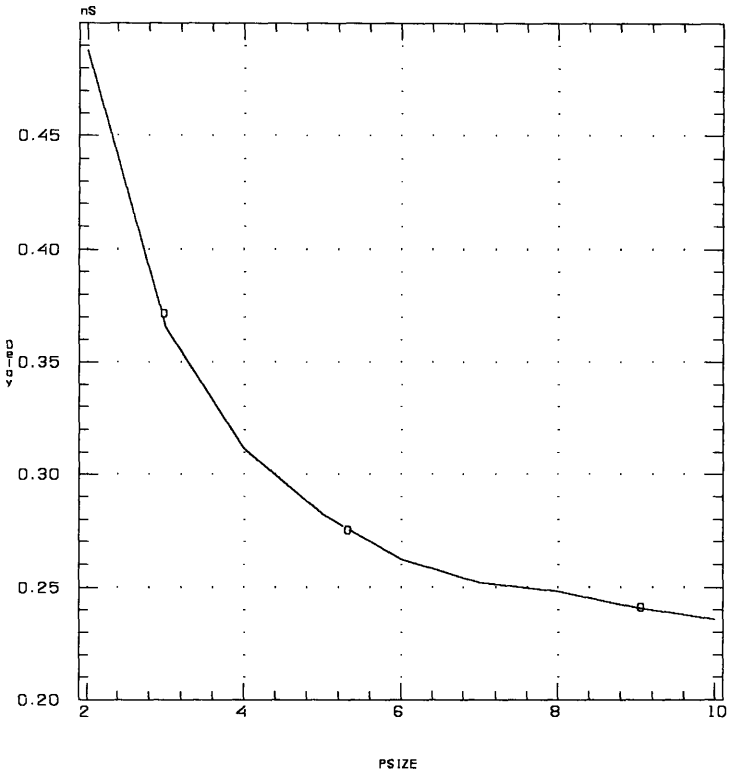

FIGURE 6. Propagation delay as a function of area used

2.3 Optimizing Domino Circuits

As with static circuits, the important considerations for domino circuits are timing, power, and area.

There are two kinds of standard domino gates: clock-anded and non clock-anded (see Figure 7). A clock-anded gate has an NMOS device in series with the NMOS block to be evaluated, which the non clock-anded gates lack. This allows greater flexibility for the

inputs of the gate while still preventing a direct path from power to ground from existing at any time. By allowing the inputs to be on during the precharge phase, the gate is much less restricted and can be used in more places than a non clock-anded gate can be, since not all potential inputs will necessarily follow that timing rule. Non clock-anded gates, on the other hand, are smaller, faster for the same functionality, and use less power because they reduce the clock loading.

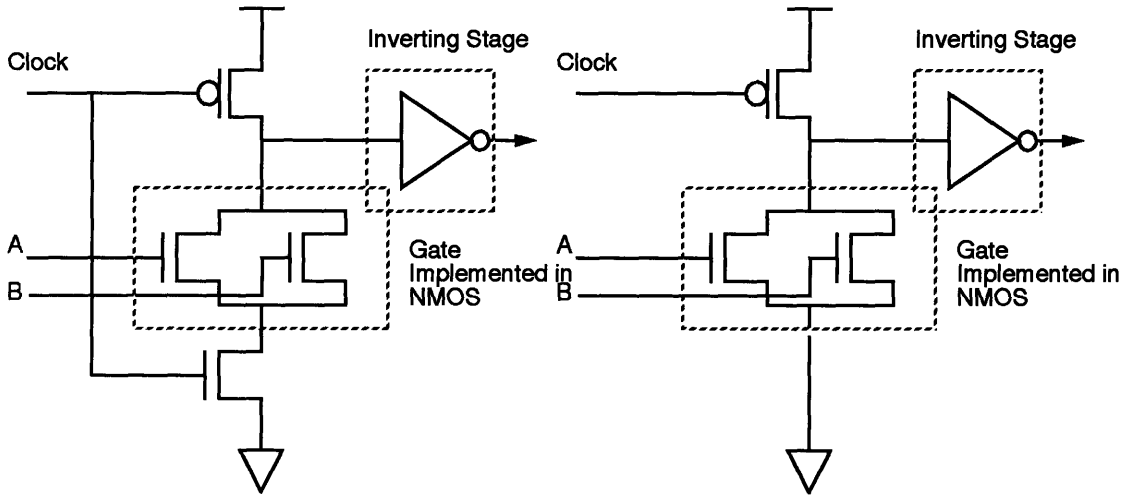

FIGURE 7. Clock-anded and non clock-anded domino gates

2.3.1 Domino Timing

Because the non clock-anded type domino circuit eliminates an NMOS device in series with the evaluate net, it is faster per gate to evaluate. Substituting non clock-anded

circuits wherever it is safe to do so can decrease the overall circuit delay, sometimes up to 20% [13].

To optimally increase the frequency while depending on a simple clocking control for the domino circuit, such as clock and clock-bar signals, precharge and evaluate phase times should be optimized to be as close to equal as possible.

Usually, the inverting stage on the output of a domino gate of either type is skewed high so that it will transition quickly when the precharged node discharges. This speeds up the evaluate time just a bit. Since the time it takes this inverting gate to make a low transition is of no importance as long as it is less than the precharge phase time, the gate is skewed as high as it is allowed to be without degrading the noise margin to an unacceptable level.

2.3.2 Domino Power Usage

Domino is a high power circuit type. Though static CMOS dissipates dynamic power only when a logical transition occurs, domino gates use power precharging and evaluating with every clock cycle that the input data forces the output (through an inverting stage) of the gate high. The devices that are connected to the controlling clock signals also add to the power dissipated by the clock network. It is very important to make use of all the power saving techniques that are available to try to cut down on dissipated power.

One somewhat simplistic way of cutting down on dynamic power is to design the logic that each gate implements so that the most common output state is low. This means the precharged node would not discharge more often than it would, saving power each clock

cycle. This technique will not always be possible to implement due to the nature of the logic.

Another way to reduce the power in clock-anded gates is to make sure the inputs are positioned such that the top NMOS devices in any stack in the evaluate net are turned off during precharge. Not only does this reduce the loading capacitance on the precharge node, eliminating some of the precharge power that is used, but it allows the precharge PMOS device to be smaller (since it does not have to precharge quite as fast) and this reduces the clock loading and power. This introduces a potential charge sharing problem where the precharged node shares it's charge with other intermediate nodes in the NMOS block when the top NMOS device first turns on at the beginning of the evaluate stage. To avoid this, the intermediate nodes should also be precharged.

Overlap current, in domino logic made up of the current that flows from power through the clocked PMOS device and the evaluate net of NMOS devices to ground with each transition, must be minimized at all costs. This means all P devices must be fully 'off' before any N devices can start to switch 'on'. For a domino clock-anded gate, a small pulse of current gets wasted with every clock tick unless the precharge and evaluate clock signals are sufficiently delayed from one another, such as with a two-phase non-overlapping clock scheme. For non clock-anded gates, current can also be wasted if the inputs start to switch during or too closely after the precharge phase, since this could cause a DC path to ground to appear. To avoid these effects, domino gates should be timed appropriately and special attention should be paid to the valid times of the inputs.

When cascading domino circuits together, care must be taken to avoid DC paths to ground. For example, the precharge time for a non clock-anded gate cascaded after a clock-anded gate must be delayed by the worst case precharge time. Similarly, successive non clock-anded stages need to start evaluating before a previous non clock-anded stage is done evaluating.

2.3.3 Domino Area

There are several ways to try to conserve area in domino circuits. First, try to reduce the loading capacitance on the precharge node by downsizing any keepers that are driven by the node. Also, the inverter on the output can be downsized if the length of the evaluate stage is not a bottleneck. This allows the precharge pullup transistors to be made smaller, with a minimum of increase in precharge time.

Similarly, the series NMOS transistor in clock-anded gates can be made smaller with only a small increase in evaluate time. Using non clock-anded type domino gates also saves the area of this one transistor.

Optimization Resolution

3.0 Resolution of Optimization Searches

Optimization of a circuit can be performed at several different levels. For example, we might specify a set of fixed cell gates to be used within a circuit, and allow the various sizes of the gates to be interchanged during the optimization process. Or we might specify a range of P/N ratios to be used, and allow the ratio within each gate in the circuit to be varied. We could go even further and specify only a range of allowed P and N device sizes to be used, and let each device size in the circuit be varied. By parameterizing smaller and smaller portions of the circuit, more detailed design goals can be achieved and the solution can be made more precise. However, as the number of degrees of freedom for a solution increases, so does the complexity of finding that solution.

3.1 Fixed Library Cells

This kind of optimization involves parameterizing the fixed cell gates that make up a larger circuit and allowing fixed cells of other sizes (but the same functionality) to replace them. As shown by the dashed boxes in Figure 8, an inverter can be replaced with an inverter of another size. It cannot be replaced with a nand or nor gate, and the internal sizing of the gates cannot be changed.

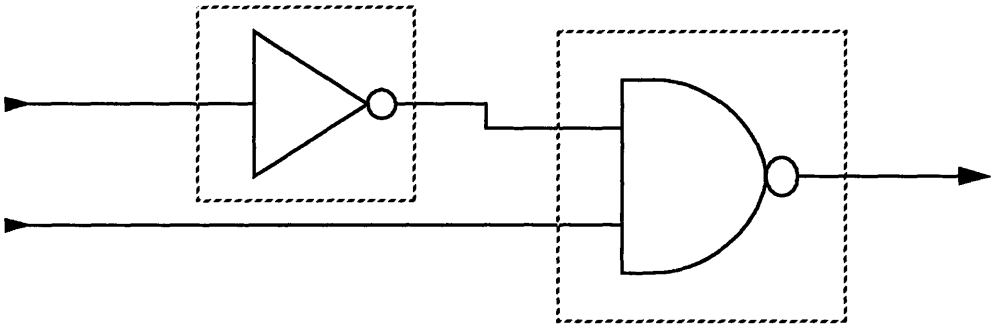

FIGURE 8. Fixed cell level of resolution

This is an important dividing line because it is impractical to customize each gate that is incorporated into a large design. Not only does that dramatically increase the amount of layout work that must be done, but it prevents careful characterization and study of each of the parts of the circuit, since that could not be done for a huge number of gates in a reasonable amount of time. Each cell in a library is thoroughly simulated and characterized before being released for general usage, which not only takes quite a bit of

effort and time, but would be nearly impossible if every gate was custom done. This extensive characterization can be used to produce in depth static timing models that can significantly speed up circuit simulation time without sacrificing accuracy. These models can also be used in optimization programs to quickly estimate performance goals for fixed cell combinations.

The fixed cell level of resolution produces the best circuit performance that is available when designers are forced to use only cells from a fixed cell library for reasons of practicality. It can be used as a first pass method, to generate a good initial circuit before beginning fine-tuning optimization.

3.2 PMOS/NMOS Ratios

As the trend towards a lower supply voltage level continues, one must be extra careful that an adequate noise margin exists in every circuit. This requires that the effective P/N ratio for each gate be between the upper and lower bounds that are decided on. The P/N ratio level of resolution involves parameterizing the effective P/N ratio of each gate in a circuit (see Figure 9). As discussed in Section 2.1, an optimal P/N ratio between these bounds does exist for each chain of gates. The optimal point is dependent on the fanout of each gate, the stage ratio, which input is used, and also on the type of gates in the chain.

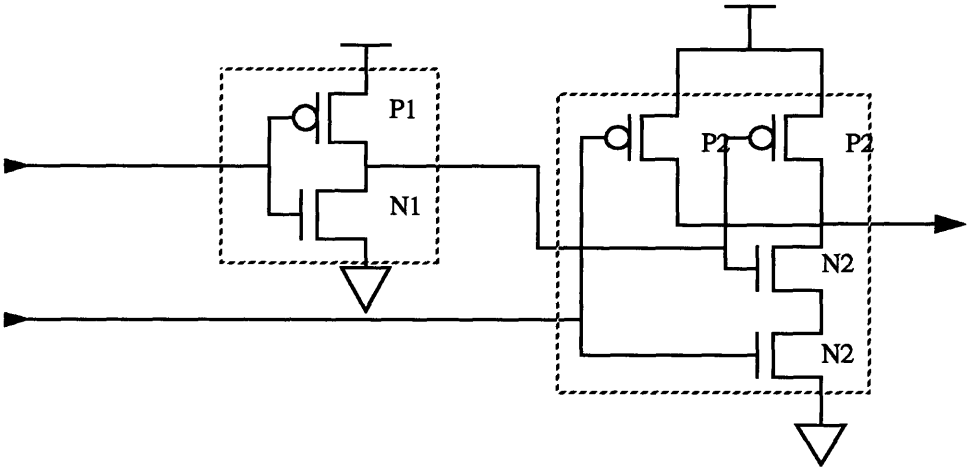

FIGURE 9. P/N ratio level of resolution

Using experiments like the ones in Section 2.1 but with NAND and NOR gates instead of inverters, the optimal P/N ratio can be calculated for both high and low skewed NAND and NOR gates (see Figure 10) [13]. The gates in this figure with only one input are inverters.

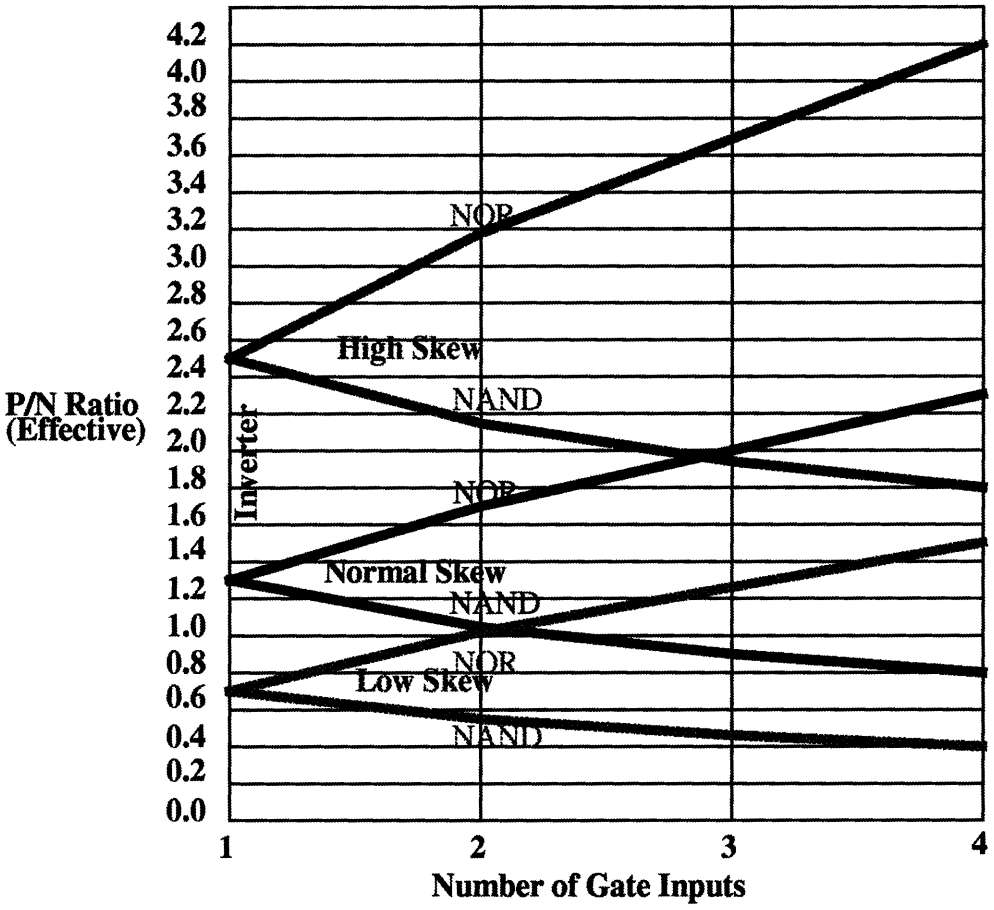

FIGURE 10. Optimal P/N ratio for inverters, NAND gates and NOR gates

For paths in which only either the rising or the falling edge is critical, the gates in the chain can be alternately skewed high and then low to produce the fastest propagation delay for that edge. In the example in Figure 11, the low to high transition on the output will be significantly faster than the high to low transition. Skewed gates sacrifice the speed of one edge for the other and must be used with care, since speeding up one critical path in this way can easily cause the other edge to become critical when it was not before.

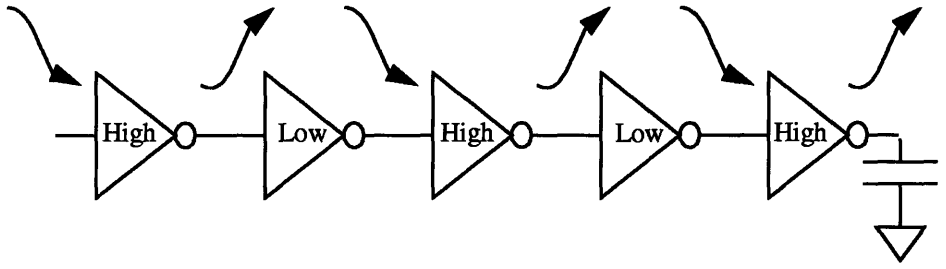

FIGURE 11. Minimal delay for one edge through alternating high and low skewed gates

3.3 Varying Individual Transistors

The finest level of resolution involves the ability to parameterize the length and width of each transistor in a circuit (see Figure 12). This will more than double the total number of variables and dramatically increase the search space of the problem. Parameterization at this level allows the final design to include gates which are skewed or have fast paths even if these gates are not included in the cell library being used. It automatically takes into account any difference in signal arrival time or slope since any change in the gate is directly driven by the need for a change of that nature.

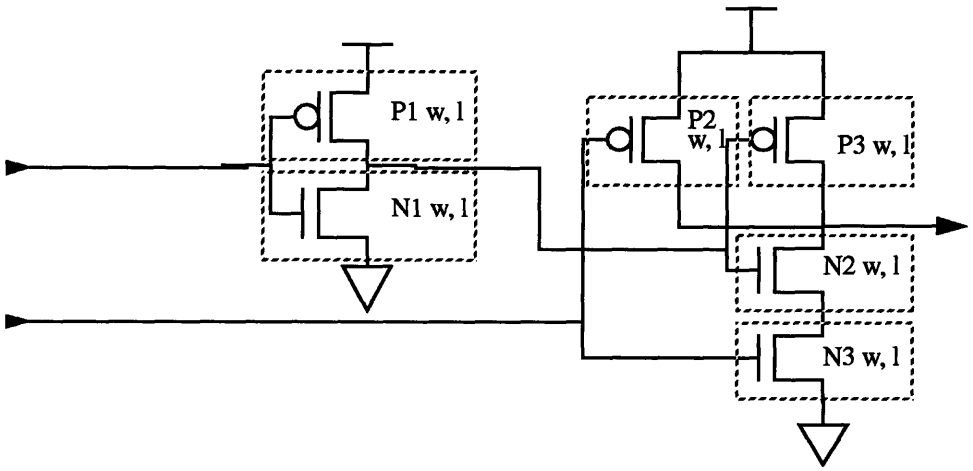

FIGURE 12. Transistor level of resolution

This method works best for extracting the last bit of performance out of a circuit that has already been optimized with other methods, since it perturbs the circuit only a little at a time.

Optimization Techniques

4.0 Optimization Techniques

Several useful techniques can be kept in mind when optimizing a circuit. Optimization that changes the circuit topology should be considered first, and optimization that relies on exact input timing information should be used last.

4.1 Change Circuit Type

Before a lot of optimization is performed with a fixed topology, it must be decided which kind of circuit is the most appropriate for the situation. There is no point in wasting valuable design time optimizing a block of static gates when a domino configuration would be more efficient or vice versa.

Static CMOS and domino gates both have their advantages and disadvantages. In general domino gates are faster but they consume more power and require clocked control

signals to be distributed to their location. In fact, both rising and falling output transitions from any nand or nor domino gate are faster than the equivalent static CMOS gate [9]. Unfortunately from a delay standpoint, domino gates are significantly harder to include in volume in a circuit since they cannot be cascaded together in chains as easily as static CMOS gates can and they use significantly more power. Domino gates also place more requirements on the timing of their inputs than static gates. For all of these reasons, certain logical functions can be much better implemented in one of the two circuit types.

For example, the propagation delay for a low to high transition on the output of a static NOR gate with a lot of inputs is much greater than that for a domino gate because the output of a static NOR must charge through a stacked series of PMOS transistors. Comparators lend themselves to being made in a domino configuration. When made of static CMOS (see Figure 13a), a comparator with more than 4-5 inputs will take a long time to charge node X high, signifying a match. Since many comparators are not only in a critical path, but also need to compare 32 bits, this clearly will not suffice. A domino comparator (see Figure 13b) uses precharging techniques on this node instead, charging it through only one PMOS transistor regardless of how many bits are to be compared.

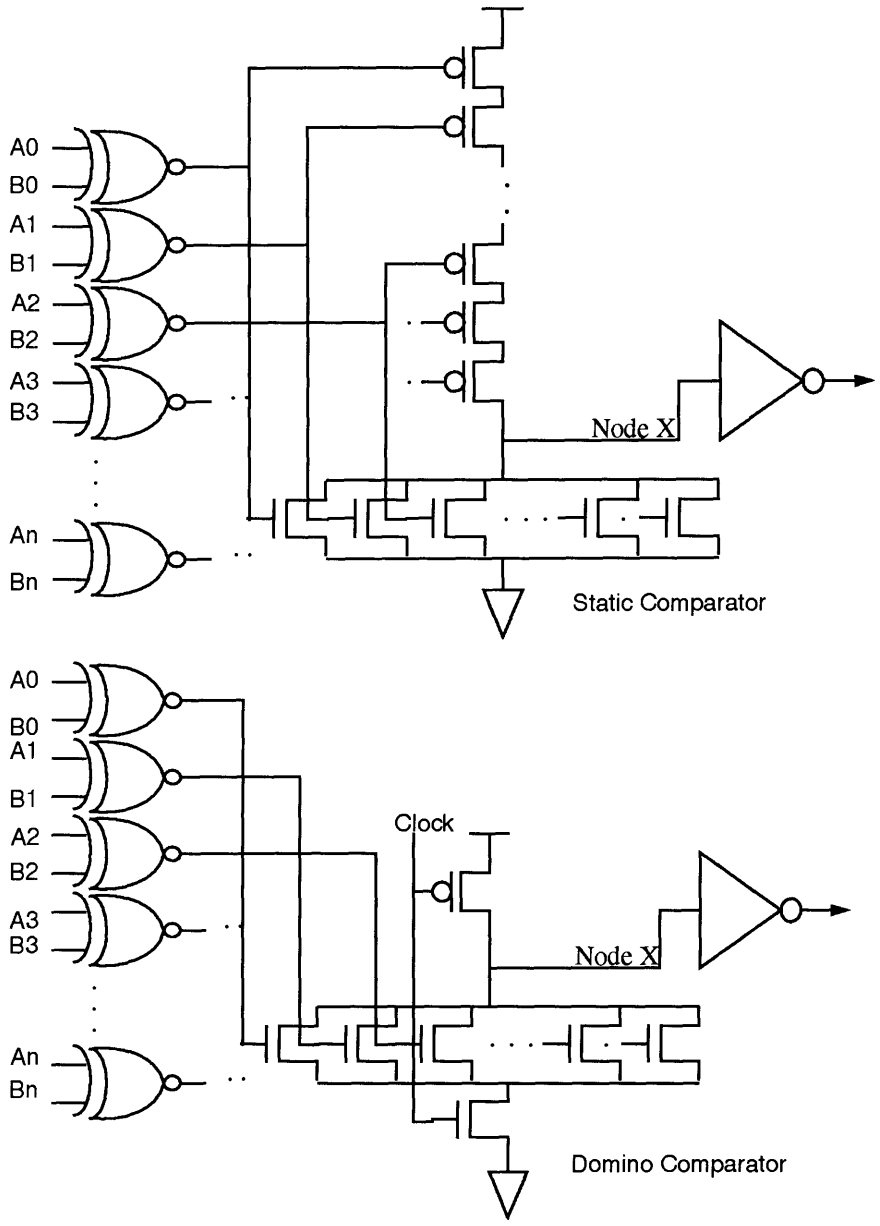

FIGURE 13. Static and domino comparators

4.2 Redesigning Physical Implementation of Logic

Sometimes critical paths can be fixed by redesigning some of the logic implementation that the path passes through. In general, optimizing the topology of a circuit before putting in a lot of effort optimizing the sizes, loading and delay can save time later, since making a small topology change will affect these other parameters and force reoptimization of them.

4.2.1 Implementation of Large Gates

A large nand gate with many inputs can be redesigned to be several cascaded stages of smaller gates with an overall decrease in delay. Figure 14a shows a four-input NAND gate with an inverted output. This configuration will have a relatively slow low to high transition because of the four stacked NMOS devices in the NAND gate. To speed this up, the circuit can be redesigned as two two-input NAND gates and a NOR gate (see Figure 14b). With some careful gate sizing, each of the paths through the new configuration can be made to be faster for the average transition. The new circuit uses more area for the same functionality.

Alternatively, the circuit can be redesigned to have one fast path (see Figure 14c). This can be useful when only one input is critical and the other signals arrive relatively early.

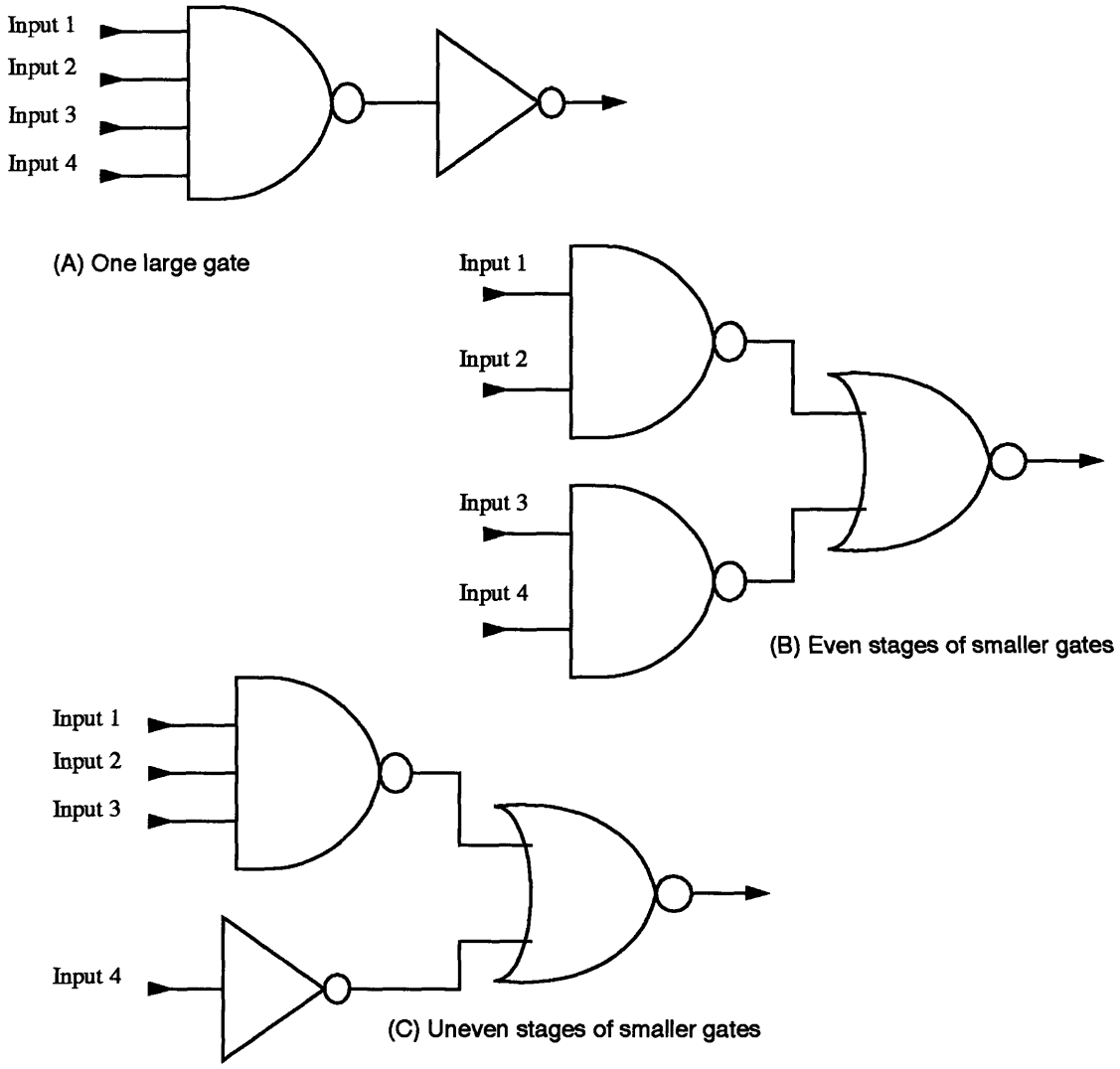

FIGURE 14. Physical implementation changes to the same circuit functionality.

Table 2 shows the improvement in propagation delay through each of the paths in the circuit. Configuration B speeds up all the paths by roughly the same amount (the difference comes from the natural difference in delay from logically equivalent pins in static

CMOS gates). Configuration C, however, speeds up one path considerably and the others only marginally. This scheme can be exploited when redesigning the physical implementation of a circuit by carefully examining the timing of the signal paths and determining which type of changes will provide the greatest advantage.

Path	% Improvement using configuration B	% Improvement using configuration C
Input 1 --> Output	28%	18%
Input 2 --> Output	30%	20%
Input 3 --> Output	37%	21%
Input 4 --> Output	37%	60%

TABLE 2. Delay improvement from redesigning a large NAND gate

4.2.2 Using Complex Gates

Complex gates can be created for logical functions that are more complicated than simple NAND and NOR structures but that recur often and can be made faster by using one gate instead of several. For example, the logical function $f = \overline{(A \cdot B) + C}$ can be implemented in three separate gates, or can be combined into one complex gate (see Figure 15). Table 3 shows the percentage decrease in delay and area for this logical function. In general, complex gates are faster and use less area as well.

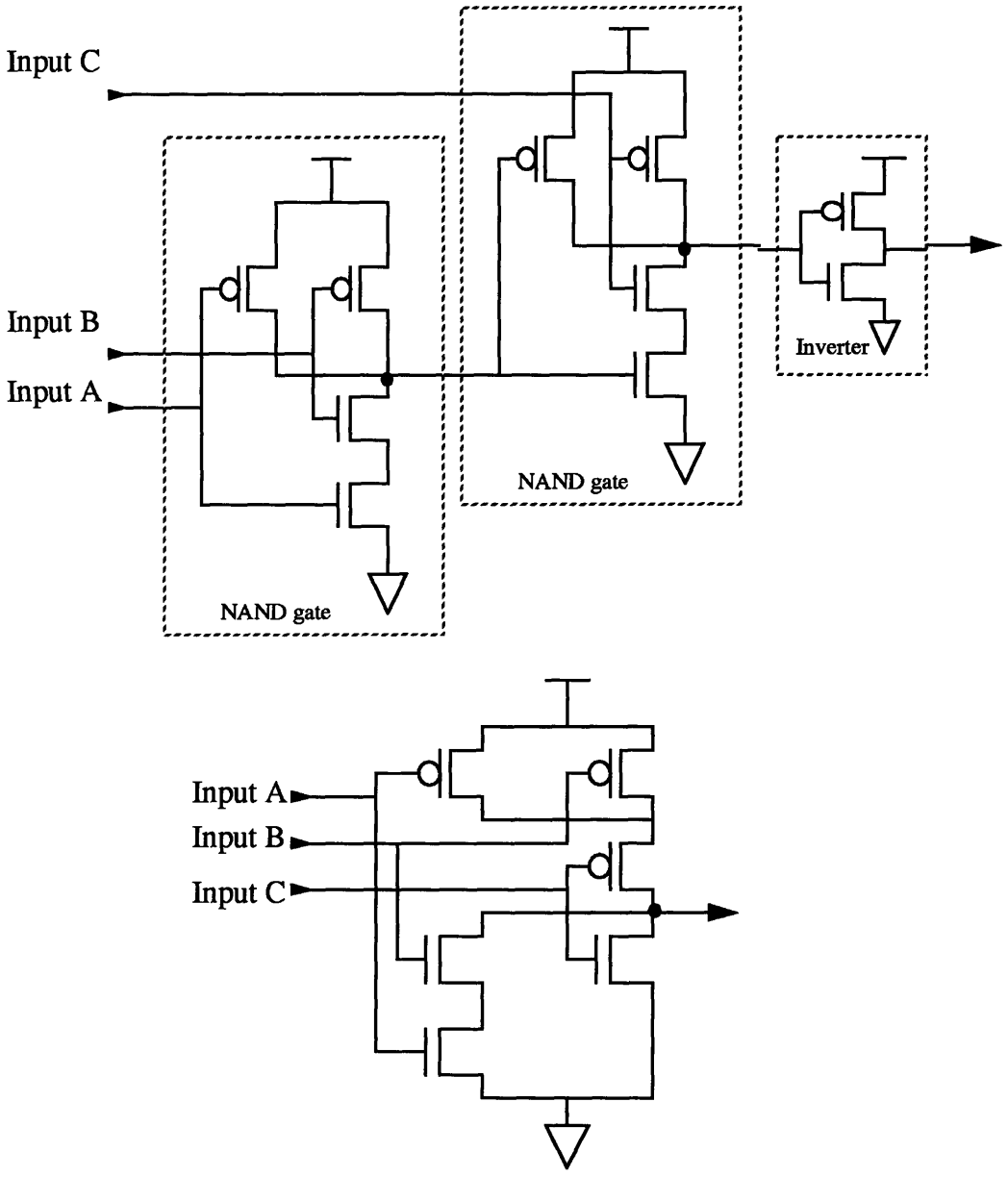

FIGURE 15. Logic function implemented as three separate gates and as one complex gate

Logical Function	Input	% Delay Improvement	% Area Decrease
	A	27%	
$f = \overline{(A \cdot B) + C}$	B	30%	70%
	C	27%	

TABLE 3. Delay and area improvement gained by using a complex gate

On occasion, the inputs to a complex gate will arrive such that replacing a complex gate with separate simple gates can help speed up a critical path. This happens at the expense of the other paths through the circuit. Figure 16 shows a case where this is true. When one input signal arrives very late relative to the others, it is most efficient to have it travel through the smallest and the least number of gates as possible. By rearranging the logic, the path from Input 3 to the output can be made to be only one simple gate long, but it means the paths from the other inputs are now three gates long. By doing this, the critical path can be made several times faster than the non-critical paths. In this case, the new propagation delay from Input 3 is ~3.4 times the speed of either of the other delays. The non-critical path delays were almost doubled with the changes.

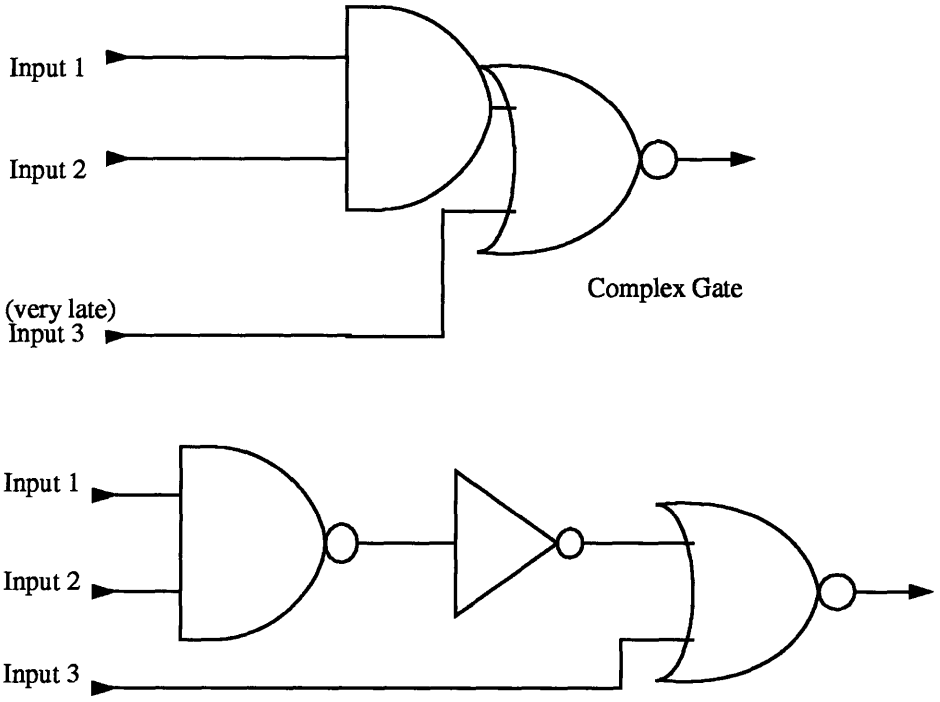

FIGURE 16. A complex gate can be broken up into smaller gates to speed up a critical path

4.2.3 Adding Buffering Stages

One way to speed up a path with a large amount of capacitance to drive is to add buffering stages that are sized correctly. The sizing process is described in Section 2.1.

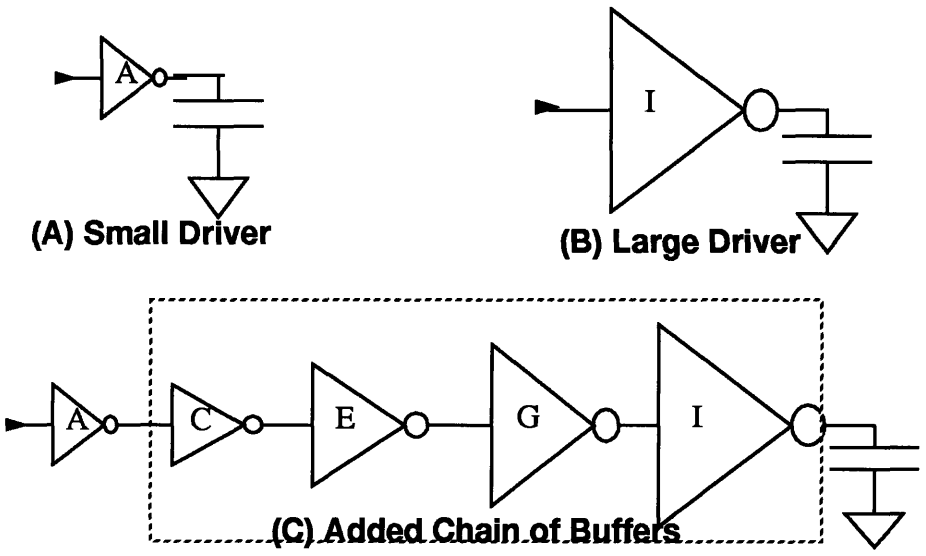

FIGURE 17. Addition of a buffering chain to help drive a large load

Driving a large capacitance is not efficient with a small driver since it does not have the channel width necessary to provide enough current to charge the load in a reasonable amount of time (see Figure 17a). On the other hand, using a large driver does not solve the problem, since it has a large input capacitance which must be driven by the previous stage, resulting in the same problem (see Figure 17b). The best way is to insert a buffering chain that starts out with a small driver, works up to a large output driver and is optimally sized (see Figure 17c).

4.3 Pin Switching

The logically equivalent pins of simple CMOS gates do not have the same signal propagation times. For example, a 3-input nor gate can show a difference of more than 10% between propagation delays from input A to the output and from input C to the output (see Table 4). For a critical path that runs through a chain of gates, this relatively small difference can add up to a significant speed increase over all the gates.

Input	tplh	tphl	Average Propagation Delay	% Faster than slowest input
C	.97	.40	.68	%13
B	.93	.38	.66	%10
A	.86	.34	.60	--

TABLE 4. Propagation delays for different inputs of a 3-input NOR gate.

This effect is due to the different amount of capacitance to be charged or discharged when each of the inputs causes a transition on the output. When inputs B and C are both low, and input A switches from high to low, the effective capacitance 'seen' by the output includes loading capacitance, drain capacitance from the NMOS devices, and drain capacitance from the PMOS devices controlled by inputs B and C (see Figure 18). However when inputs A and B are low and input C switches from high to low, the effective capacitance is less because the drain capacitance from the PMOS devices controlled by inputs A and B is already charged. This accounts for a faster transition on the output.

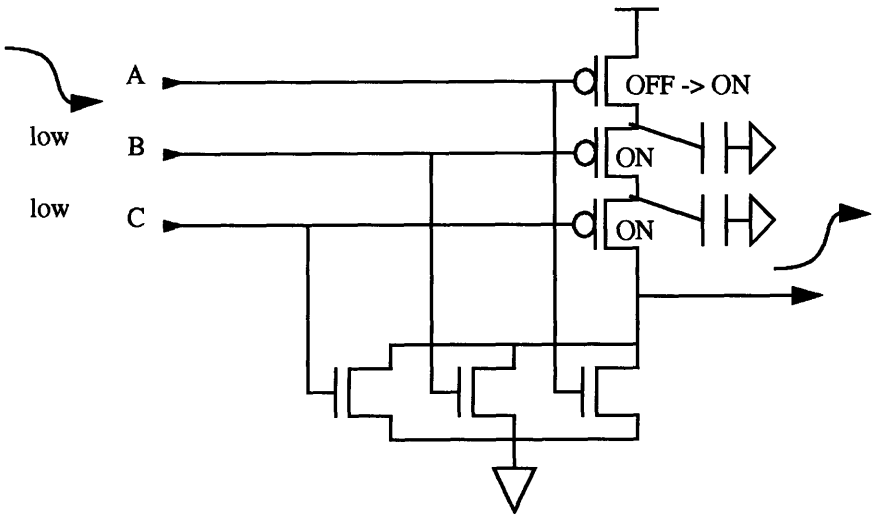

FIGURE 18. Effective capacitance 'seen' by the output for input A

Knowing that a certain pin of a gate has a faster propagation time than the other pins allows the circuit designer to prioritize the incoming signals according to how early they arrive and to place the most critical signal on the fastest pin. The performance increase varies depending on the size of the circuit, the number of pin switches that can be made and the relative times of the incoming signals. This technique applies to domino logic, as well as to static.

As an example, consider the circuit in Figure 19a. To gain a small percentage of speed increase, input pins on some of the gates can be interchanged, as can be seen with a simple analysis of the relative times of each of the inputs to the gates (see Table 5).

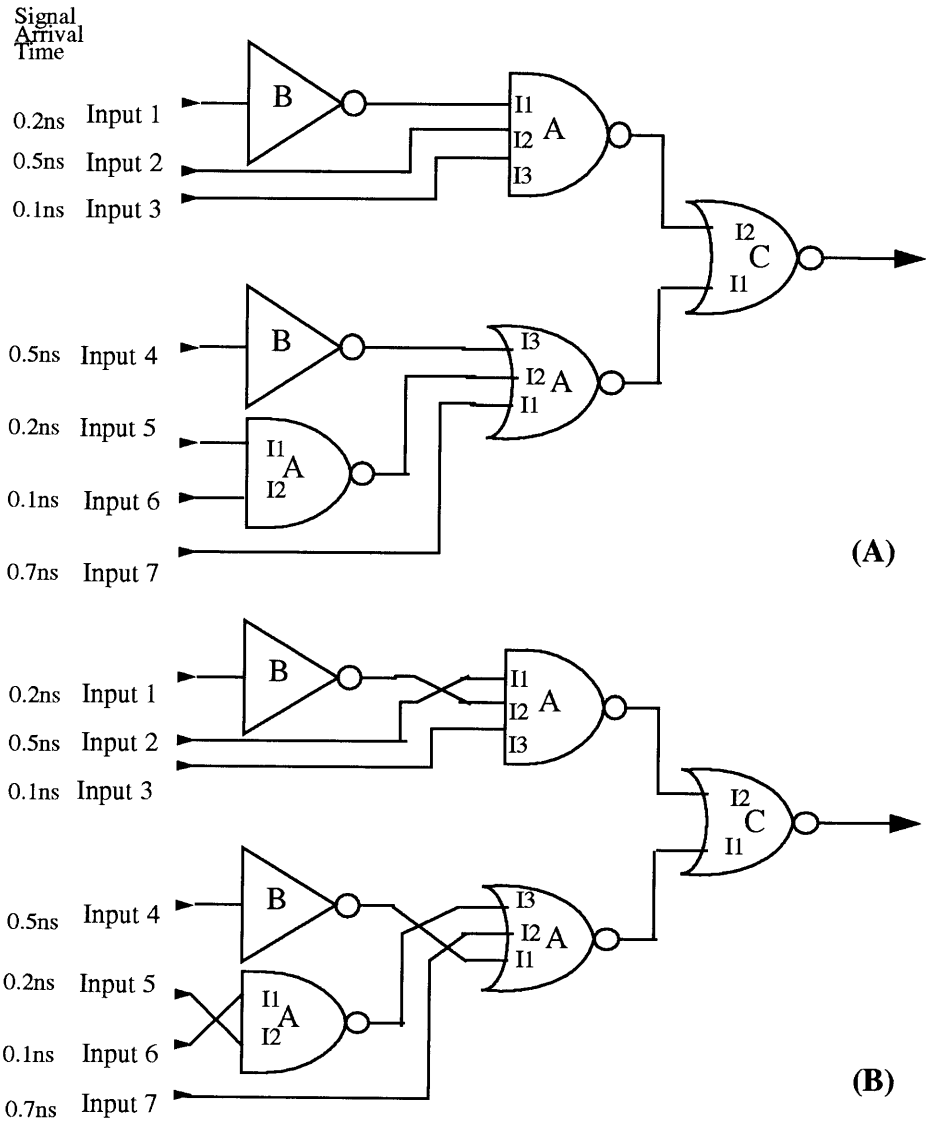

FIGURE 19. Example circuit (A) Before pins are switched (B)After pins are switched

Gate	Gate Input	Latest Signal Relative Arrival Time (nsec)	Propagation delay for this input pin (nsec)	Input signal should be moved to:
2 input nand ('A' size)	i1	0.1	0.46	i2
	i2	0.2	0.50	i1
3 input nand ('A' size)	i1	0.44	0.66	i2
	i2	0.50	0.73	i1
	i3	0.10	0.77	--
3 input nor ('A' size)	i1	0.76	0.64	i2
	i2	0.70	0.75	i3
	i3	0.70	0.79	i1
2 input nor ('C' size)	i1	1.17	0.31	--
	i2	1.55	0.34	--

TABLE 5. Timing data to determine if pins should be switched

The signal coming into i2 of the 2-input NAND gate arrives the latest and therefore should be moved to the pin that has the fastest average propagation delay for rising and falling edges, in this case i1. Similarly, the signal at i1 arrives the earliest, and should be moved to i2 which has the longest propagation delay. Using this kind of analysis, the rela-

tive priority of the input signals to each gate can be determined and the most critical signals can be connected to the fastest pins.

In the case of the 3-input NOR gate, incoming signals on i2 and i3 arrive at the same time. However, after the pins of the 2-input NAND gate are switched, i2 will become valid before i1. This explains why i2 should be moved to the slowest pin, i3, and i3 should be moved to the second slowest pin, i2.

After making the changes suggested in Table 5, (see Figure 19b), the more critical paths through the circuit are faster (see Table 6). The delay through the longest path in the circuit has been reduced by more than 5% (compare the old longest delay, from Input 4 to the output, to the new longest delay, from Input 6 and 7 to the output). This will not always be the case, as it is possible to have two incoming signals arrive at about the same time and only one can be placed on the fastest pin. The designer must decide which of the paths is more critical and which signal should connect to the faster pin. Notice that the decrease in delay through the improved paths is directly reflected in an increase in delay through the remaining paths.

Signal path	Old Delay	New Delay	% Improvement
Input 1 --> Out	1.44	1.51	-4.8%
Input 2 --> Out	1.57	1.50	4.5%
Input 3 --> Out	1.21	1.21	0.0%
Input 4 --> Out	1.86	1.71	8.0%
Input 5 --> Out	1.76	1.70	3.4%
Input 6 --> Out	1.62	1.76	-8.6%
Input 7 --> Out	1.65	1.76	-6.6%

TABLE 6. Signal path improvements produced by switching pins

4.4 Fast Path Gates

When the path through a gate is limited by one slow input, it is possible to replace that gate with one that is skewed to reduce the delay of that path at the expense of the other paths. In gates that contain two or more logically equivalent inputs, such as simple nand and nor gates, there naturally exists a difference between the inputs in propagation delay. This difference can be enhanced by careful sizing of the gate, so that when the latest arriving signal flows through the fastest path and the earlier signals flow through the paths that take longer, the gate delay for the latest signal is minimized.

For example, consider a 2-input nand gate (Figure 20a). The propagation delay from the input that controls the n-transistor which is connected to the output (Input 1) is slightly faster than the propagation delay from the input that controls the n-transistor which is con-

nected to ground (Input 2). However, if the transistors are sized such as in Figure 20b, the propagation delay from Input 1 for a low to high output transition can be made to be much faster than from Input 2.

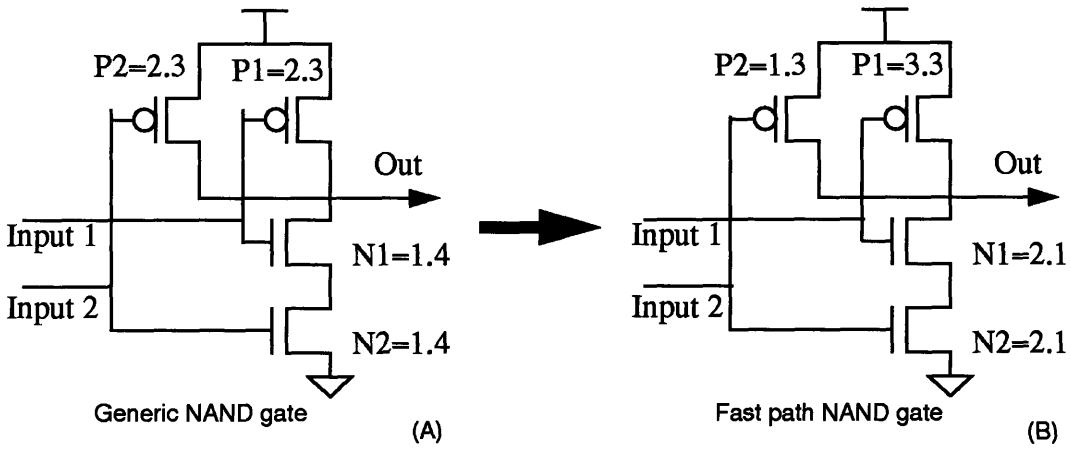

FIGURE 20. Sizing changes which produce a fast path 2-Input NAND gate

To increase tplh from Input 1, the size of transistor P1 can be increased. To increase tphl from Input 1, N1 can be increased. Since a high to low transition on the output will force current through the series combination of N1 and N2, N2 also needs to be increased to speed up tphl. P2, however, can be decreased since it is controlled by Input 2 and it is less important that it be speedy. These sizing modifications change the input capacitance of the gate for both inputs. Since a larger input capacitance will slow the rise and fall times of the incoming signals and therefore the propagation delay of the fast path gate itself, the sizing changes should be balanced to prevent significant capacitance increases.

The results of the sizing changes shown in Figure 20 are tabulated in Table 7. The path through Input 1 is faster on both transitions. Input 1 shows an average speed improvement of 12.4%, whereas Input 2 shows an average loss of speed of 21.6%. Notice that Input 2's high to low propagation delay actually improved because both NMOS device sizes were increased. The delay values listed take the change in input capacitance into account, since they include the propagation delay of fixed signal drivers.

TPHL	Input 1	Input 2
Old tphl	0.47	0.49
New tphl	0.39	0.39
Percentage Improvement	17%	20%

TPLH	Input 1	Input 2
Old tplh	0.45	0.49
New tplh	0.41	0.80
Percentage Improvement	9%	-63%

TABLE 7. Propagation delays resulting from fast path gate sizing changes

It is clear that the critical paths should be routed through the faster pins before considering fast path gates for improvement (as discussed in the pin switching section). After each gate's input signals have been switched so that the latest arriving signals go through the fastest paths in each gate, there can still be critical paths through that gate. By again

comparing the relative arrival times of each of the inputs, gates which could be replaced with fast-path gates can be identified. For any circuit, there could exist a wide range of differences in relative arrival time for gate inputs. For some gates, only one input could be the limiting signal, and the other signals arrive early enough that any deterioration of the propagation delay from those inputs would not adversely affect the circuit performance. For other gates, the signals may arrive fairly closely in time or there may be more than one critical input signal in which case a balance in the gate needs to be found between speeding some inputs up at the expense of the others.

When changing the transistor sizes to increase the speed of one path through a gate, the speed of paths through other inputs of that gate are decreased. If the other inputs arrive sufficiently early, though this decrease will affect propagation delays through these other paths, it will not cause them to be slower than the original critical path. On the other hand, if one or more of the other inputs do not arrive sufficiently early, fixing the Original critical path can create a new one, possibly worse than the first. When designing fast path gates, it is important to take input signal timing into account when sizing each gate such that new critical paths are not created.

4.5 Highly Skewed Gates

When only one edge of a transition is critical, a fast path gate does not exploit the full potential to speed up a single path. Skewed gates, simple gates in which the P/N ratio is either very much larger or very much smaller than usual, can do this.

Skewed gates by nature increase the speed on only one transition on the output: high to low, or low to high. The threshold on their input is shifted considerably. A gate skewed high has a lower threshold, and a gate skewed low has a much higher threshold (see Figure 21). Since noise margins are derived directly from the transfer functions of the gates they are associated with, noise margins for high skewed gates also change considerably. This means a high skewed gate can only tolerate a fraction of the noise on a low input before it will transition high by mistake. Similarly, a low skewed gate can tolerate less noise when its input is high. For this reason, skewed gates are not as safe to use, and should only be incorporated when the noise level is sufficiently low.

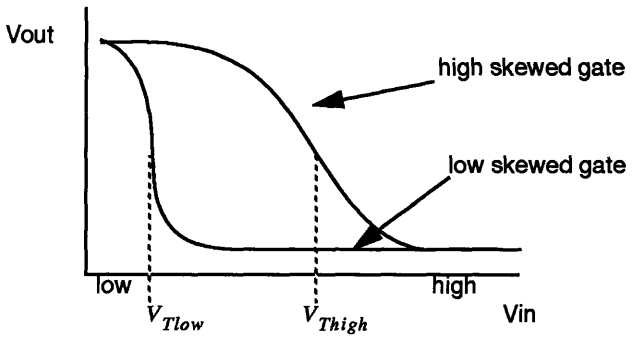

FIGURE 21. Threshold shift for high and low skewed gates

Optimization Strategy

5.0 Optimization Strategy

Different optimization strategies can be used. Several are explained in detail below.

5.1 Extracting Critical Paths

When optimizing very large circuits, it can be hard to decide what aspect to start with. Parameterizing every gate or every transistor dramatically increases the search space and can cause the optimal solutions to be very hard to find. Big circuits also have more design goals than small ones, and changing the size of one gate at a time can show only a small effect on these goals if any at all. This makes the search for the best results even harder and more tedious. In this situation, extracting the critical paths for separate optimization can be both useful and practical.

Extracting the critical path can be very useful when one path is significantly worse than the others and most of the optimization work would be done on this path anyway. In general, improved performance for the one path comes at the expense of other paths and therefore some leeway should be allowed for this. Sometimes a single path can be forcing the minimum period that can be used with a circuit to be larger and any improvement to that one path will directly allow the frequency to be increased. In this case, small increases in other paths would probably be acceptable.

To isolate one path from the rest of the circuit, the connections to other parts of the circuit must be modelled and the path must be sensitized to any stimulation from the inputs. Connections can be modelled as simple capacitors or as actual transistors, depending on the accuracy desired in the extracted path. If the critical path runs through a NOR gate, the non-critical inputs should be set 'low' so that the critical input can influence the output. Similarly, non-critical NAND gate inputs should be set 'high'.

Once a path has been extracted, the optimization process becomes easier. The circuit to work with is considerably smaller, with fewer variables, and fewer input and output requirements. Many of the interconnections that may have existed have been generalized to lumped capacitances. The design goal priorities are much clearer and less complicated.

The extracted path can be optimized with any of the techniques described in earlier chapters. After optimization, some small modifications might need to be made to the surrounding circuit to facilitate connecting the circuit pieces back together again.

Extracting the critical path can cause several problems. When the optimized path is put back into the original circuit, it can cause other paths to be slower due to loading differences. The critical path itself can be slower when put back in if it was not modelled carefully when it was extracted.

5.2 Recursive Strategies

Recursive strategies are fine tuning strategies that can be used on large circuits as well as small ones. They can use gate snapping or individual transistor variable methods and can be adapted for either static or domino logic. Recursive strategies have several variations but each include dividing the circuit up into slices by some qualifications, optimizing each slice by itself, putting the pieces back together to get one large circuit again, verifying the result and iterating until the best result is found.

5.2.1 Recursive Strategy #1

One variation for static logic optimization divides the circuit into groups of gates by its outputs and the logic cones that propagate back from them to the inputs (see Figure 22). Each of these groups is optimized as separately as possible, given that there might be some overlap between the groups, using what ever techniques and resolution apply best. Afterwards, the resulting complete circuit is simulated to verify performance. If the results are not satisfactory, the process is repeated.

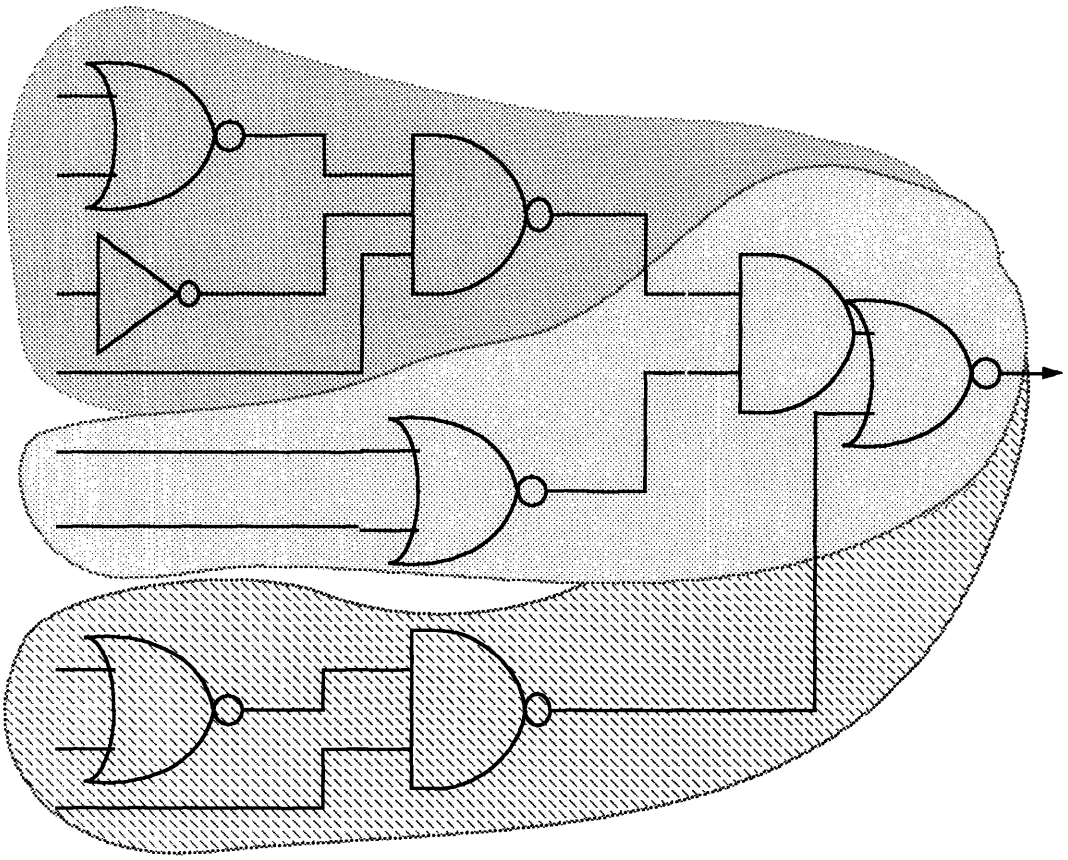

FIGURE 22. Recursive strategy circuit divisions by output

This strategy has several limitations. If a lot of gates are included in more than one logic cone and conflicting sizing goals for these gates are created, the average sizes that will be used in the resulting circuit will keep each of the paths from seeing a full potential speedup.

5.2.2 Recursive Strategy #2

A variation that works well for both static and domino circuit types includes dividing the circuit into slices by logic stages (see Figure 23). The slices are then optimized, either with gate snapping or individual transistor variable techniques, starting with the slice that includes the output, since its capacitive load is already known and fixed at one value. Then as each slice is optimized, the load for the slice before becomes known. After each of the slices are finished, the whole circuit is simulated and the process is iterated if necessary.

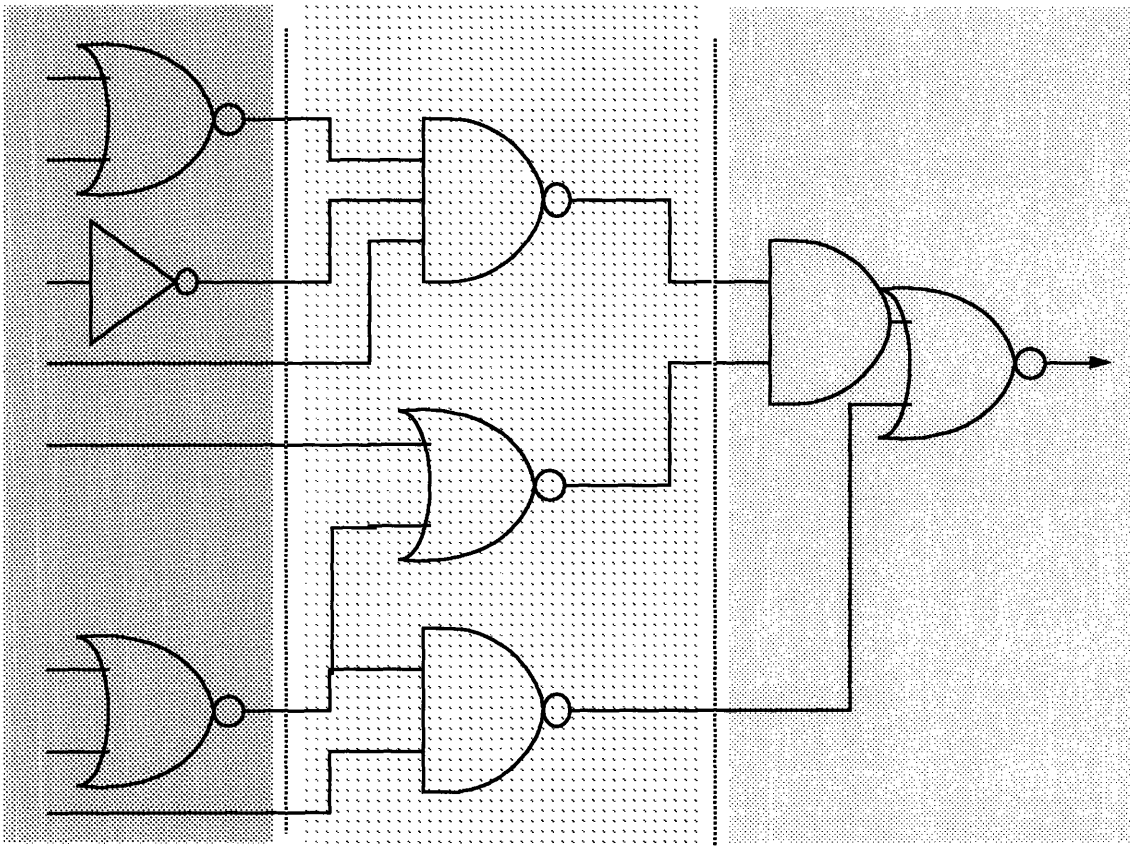

FIGURE 23. Recursive strategy circuit divisions by logic stage slices

5.3 Simple Sizing Strategy

A fairly simple sizing strategy is needed as a first pass technique to convert schematics that were created from architectural specifications with few sizing considerations into roughly sized schematics that are ready for fine tuning. The strategy discussed below prioritizes the optimization goals by first considering only capacitive loading violations, then

considering timing improvements that can be made without introducing any more loading problems, then considering area in places where loading is not affected and timing will not deteriorate. It was designed for and works best on static logic, as it relies primarily on some fairly basic indicators.

A sample circuit that contains mostly 'A' size gates is the initial design for this strategy. This kind of circuit could have been generated by a logic synthesis tool from an architectural specification. The only things that are known about the circuit are the capacitive loading and timing requirements on the output and the valid signal times on the inputs. The object at this point in time is to modify the sizes of the gates in a fast and efficient manner to produce a working circuit that meets the basic timing and load requirements. Snapping to existing gate library sizes is a good idea for a first pass.

The valid load requirements are looked at first, and the timing issues and area concerns are saved for later. First the circuit is statically simulated to determine the timing margins. Starting with the output, one gate at a time is sized up to drive the load that it is known to have in a reasonable amount of time. Sometimes some of the initial sizings are large and proceeding gates cannot meet the load requirements because they are not allowed to become large enough to drive that much capacitance. Then the initial large gates must be downsized a step and the resizing process for the preceding gates happens again. This can iterate until the loading requirements are met, or until the gates being driven fall out of their allowed range for the load they have to drive. In this case, this simple sizing method

fails and more complex methods must be used, such as reevaluating the logic, or making some topology changes.

After the loading requirements are met, resizing with the intent of reducing delay to meet timing requirements begins. Gates are sized in a random order either one library step up or down, always keeping within the specified range to ensure that the work that was just completed sizing each gate with respect to its load is not destroyed. In-between each sizing, static simulation can be repeated to check timing margins from each input to each output. To allow for circuit delay trade-offs to be realized, the total margin for the circuit block being run is added up and used as a comparison to determine whether the latest size change has made the circuit better or worse, from a timing margin point of view. If the total margin decreases, the sizing change is kept and the next gate is sized. If the total margin increases, the last change made is changed back, and the next gate is sized.

When the delay is as minimal as it can be, the circuit area is considered. Some signals in the new circuit will be arriving earlier than they need to, and some of the gates that drive these early signals can be sized down without moving out of the acceptable range for the load they are driving, and without adversely affecting the timing margins. Each gate is considered again, and sized down if they fit these requirements. After each sizing change, the timing is verified to make sure the margins are intact.

This strategy is good for a first pass at a circuit that has not been optimized before. The best result it can achieve is limited to the same circuit topology with optimal fixed gate

sizes, and there are a number of obstacles to reaching this optimal point. However, it is easy to set up the circuit to be optimized with this strategy. No waveforms need to be specified, and objectives are automatically incorporated. The result does not depend on complex measurements or calculations. The decision to keep or discard a size change is a simple comparison and is easy to make.

This strategy also has disadvantages. It's simplicity allows it to be easy to use but prevents it from returning a truly optimal result. Using the sum of the timing margins as a comparison to determine whether a change should be kept or not can sometimes lead in the wrong direction, especially if a lot of cases exist in the circuit where increasing one margin slightly would allow another margin to decrease significantly on the next resizing pass. Because only one gate is changed at time, possible decreases in delay that are only seen when two or more gates are modified together are not found. There is the added problem of this strategy getting into an infinite loop. Since certain size changes might make some margins better but others worse, the circuit may start to oscillate between improving one margin at the expense of another by making one type of size change, but then improving the other at the expense of the first by reversing the size change.

The timing margin sum includes margins that are calculated from each input to each output, in accordance with static timing conventions. However, some inputs will never influence certain outputs because of the way the logic is designed and therefore should not be considered in the timing margin sum. This relatively simple method of comparison can allow time to be wasted optimizing for these paths when really they are not important.

Conclusion

6.0 Conclusion

To avoid wasting valuable design time and effort reworking during the optimization process, a carefully constructed design flow is important. Several optimization techniques have been discussed in this thesis, but the order in which they are used can play a big role in the results that are obtained.

It is my recommendation to optimize circuits using the discussed techniques in the order shown in Figure 24.

It is very important to construct circuits from a fixed cell library that is optimal. The library should include a set of basic gates that are optimally sized so that when they are cascaded together, the input capacitance of one is equal to the optimal fanout of the previ-

ous one. The skew of each gate should be optimized so that the total propagation delay is minimized. The library should include a number of complex gates, fast path gates, and high and low skewed gates so that each of the common problems that cause critical paths can be dealt with effectively.

After an initial circuit has been designed, any topological changes should be made as soon as possible. Gates that could be implemented with a different circuit type should be changed and the small stages of logic should be redesigned if necessary.

Next, an analysis of the relative arrival time of the input signals can help determine if any input pins should be switched to allow the most critical signals to travel through the fastest paths. After any pins have been switched as needed, sometimes several small logic stages can be reoptimized.

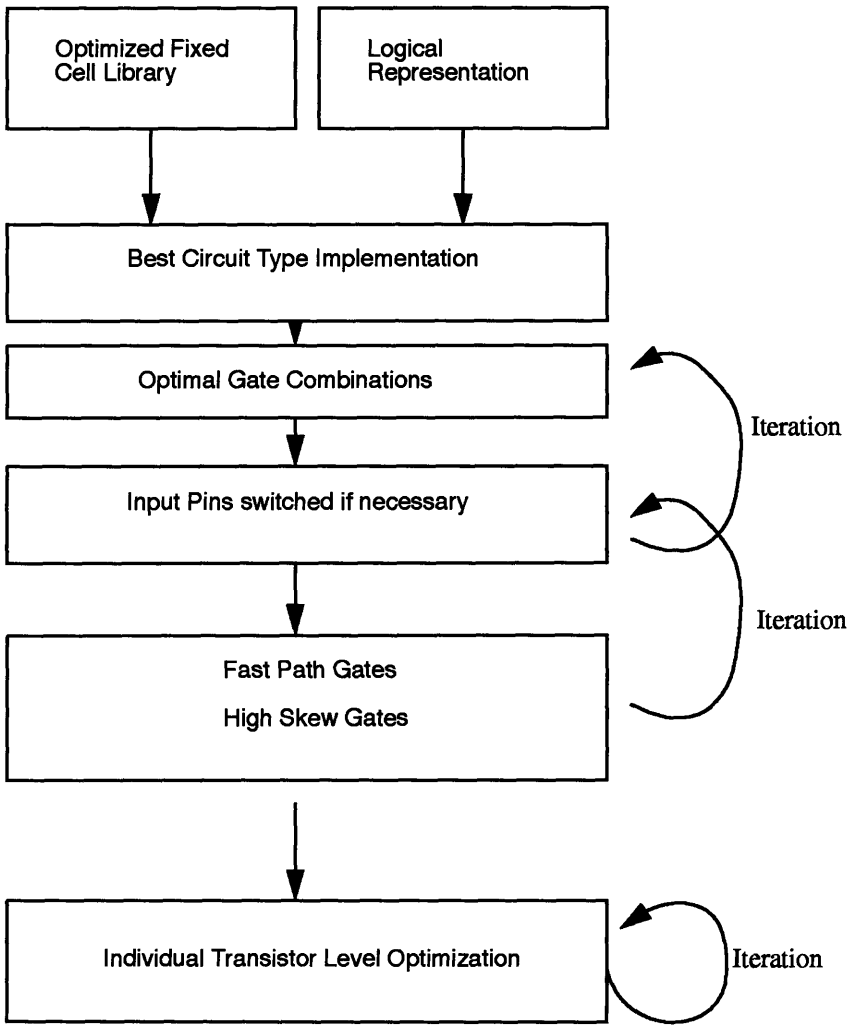

FIGURE 24. Optimization flow chart

Another analysis of the input signals can help determine if fast path gates or very high or low skewed gates could help fix any critical paths. Incorporation of these gates can sometimes change the relative arrival time of signals for other parts of circuit, and it may be useful to determine whether switching input pins again could provide any more of an increase in speed.

Lastly, individual transistor optimization can be performed to fine tune the design and to extract the last bit of performance from the circuit.

6.1 Further work to be done

Optimization is a complex problem and much work on the subject remains to be done. Circuit synthesis, for example, instead of producing a circuit with mostly minimum sized devices, could be combined with timing requirement information and expanded to produce circuits that already had an optimal topology, freeing up design time. Tools that could consider larger circuits, possibly entire chip designs, are definitely needed. Large scale topology changes could then be made to reflect the real relative priority of signal paths as they are revealed and as the design comes together.

Automating the optimization process is also important. There are many things that still must be done by hand, including decisions that must be made for special cases, and other things that cannot yet be automated. The circuit designer certainly has enough to do without spending a lot of time checking circuits for optimization opportunities that could be

characterized and reliably automated. A good optimization tool could take specific design goals and a weighting of their relative importance and come up with the optimal circuit that fit all the known constraints and requirements, using a full range of all the design techniques. The tool would have to have quite a bit of intelligence about circuit waveforms and characteristics and be able to make decisions normally made by the circuit designer to be useful.

Upcoming trends also need to be incorporated into optimization tools. As the power supply voltage shrinks, noise margins will become more important for producing reliable circuits. Power and area issues will become greater priorities, and new strategies will have to be able to handle smart power down of circuit sections, reduction of overlap current, and analysis of delay and area trade-offs. They will also have to be flexible enough to be able to incorporate new circuit types and new sophisticated timing schemes as they are developed.

Bibliography

[1] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated-Circuit Design", *Proceedings of the IEEE*, 1981.

[2] S. Dubagunta, "Advanced Tools for Integrated Circuit Design", Private Communication, Semiconductor Research Corp., 1993.

[3] P. Gray and R. Meyer, *Analog Integrated Circuits*, John Wiley & Sons, New York, 1993.

[4] L. Huelsman, "Optimization--A Powerful Tool for Analysis and Design", *IEEE Transactions on Circuits and Systems*, 1993.

[5] C. Michael, H. Su, M. Ismail, A. Kankunnen, and M. Valtonen, "Statistical Design and Optimization of Analog MOS Integrated Circuits", Journal Preprint, Ohio State University, OH, 1993.

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

[7] K. Singhai, C. McAndrew, S. Nassif and V. Visvanathan, "The Center Design Optimization System", *AT&T Technical Journal*, 1989.

[8] A. Shen, "Performance Optimization of Large Sequential Circuits", *Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology*, 1992.

[9] M. Shoji, *CMOS Digital Circuit Technology*, Prentice Hall, New Jersey, 1988.

[10] S. Trimberger, *Automated Performance Optimization of Custom Intergrated Circuits*, UMI Research Press, Ann Arbor, MI, 1986.

[11] J. Uyemura, *Circuit Design For CMOS VLSI*, Kluwer Academic Publishers, Boston, MA, 1992.

[12] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd ed., Addison-Wesley, Reading, MA, 1992.

[13] Internal Documents, Intel Corp.