**TABLE II**

<table>
<thead>
<tr>
<th>i</th>
<th>r_{1,i} = J_i</th>
<th>r_{2,i} = J_i</th>
<th>r_{3,i} = J_i</th>
<th>r_{4,i} = J_i</th>
<th>r_{5,i} = J_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>r_{1,0} = J_0</td>
<td>r_{2,0} = J_0</td>
<td>r_{3,0} = J_0</td>
<td>r_{4,0} = J_0</td>
<td>r_{5,0} = J_0</td>
</tr>
<tr>
<td>1</td>
<td>r_{1,1} = J_1</td>
<td>r_{2,1} = J_1</td>
<td>r_{3,1} = J_1</td>
<td>r_{4,1} = J_1</td>
<td>r_{5,1} = J_1</td>
</tr>
<tr>
<td>2</td>
<td>r_{1,2} = J_2</td>
<td>r_{2,2} = J_2</td>
<td>r_{3,2} = J_2</td>
<td>r_{4,2} = J_2</td>
<td>r_{5,2} = J_2</td>
</tr>
<tr>
<td>3</td>
<td>r_{1,3} = J_3</td>
<td>r_{2,3} = J_3</td>
<td>r_{3,3} = J_3</td>
<td>r_{4,3} = J_3</td>
<td>r_{5,3} = J_3</td>
</tr>
<tr>
<td>4</td>
<td>r_{1,4} = J_4</td>
<td>r_{2,4} = J_4</td>
<td>r_{3,4} = J_4</td>
<td>r_{4,4} = J_4</td>
<td>r_{5,4} = J_4</td>
</tr>
</tbody>
</table>

Fig. 3. An optimal testing tree for Example 3.

**IV. Conclusions**

In this paper, we provided an efficient algorithm to construct testing procedures for optimally identifying a single defective unit in a series system. A series system such as a local loop of telephone networks is modeled as a sequence of units. The costs incurred by the testing process are quite general in that both traveling costs and testing costs are taken into consideration. Although the model assumes that only one defective unit can exist in the system, the testing tree still leads to the isolation of a defective unit if there exists two or more. This is because each time we proceed to a subtree, it is ensured that there is a defective unit corresponding to a vertex within that subtree.

**References**


**Clock Skew Optimization**

**JOHN P. FISHBURN**

Abstract—This paper investigates the problem of improving the performance of a synchronous digital system by adjusting the path delays of the clock signal from the central clock source to individual flip-flops. Through the use of a model to detect clocking hazards, two linear programs are investigated: 1) Minimize the clock period, while avoiding clock hazards. 2) For a given period, maximize the minimum safety margin against clock hazard. These programs are solved for a simple example, and circuit simulation is used to contrast the operation of a resulting circuit with the conventionally clocked version. The method is extended to account for clock skew caused by relative variations in the drive capabilities of N-channel versus P-channel transistors in CMOS.

Index Terms—Clocking, clock skew, finite-state machines, linear programming, optimization, synchronous circuits.

I. INTRODUCTION

Synchronous circuit designers ordinarily try to eliminate clock skew, which may be defined as variations in the delays from the central clock source to the flip-flops (FF's) of the system. This effort can involve equalization of wire lengths, careful screening of off-the-shelf parts, symmetric design of the distribution network, and design guidelines to eliminate skew due to process variations [1], [2]. Clock skew can limit the clocking rate of a synchronous system or cause malfunction at any clock rate. Some static timing analyzers [3] detect

(\(k + 1, r_{k+1,j}^T\)). An inductive proof that BUILDTREE(i, j, D) correctly constructs \(T_{i,j}^T\) should thus be evident. 

**REFERENCES**


**Clock Skew Optimization**

**JOHN P. FISHBURN**

Abstract—This paper investigates the problem of improving the performance of a synchronous digital system by adjusting the path delays of the clock signal from the central clock source to individual flip-flops. Through the use of a model to detect clocking hazards, two linear programs are investigated: 1) Minimize the clock period, while avoiding clock hazards. 2) For a given period, maximize the minimum safety margin against clock hazard. These programs are solved for a simple example, and circuit simulation is used to contrast the operation of a resulting circuit with the conventionally clocked version. The method is extended to account for clock skew caused by relative variations in the drive capabilities of N-channel versus P-channel transistors in CMOS.

Index Terms—Clocking, clock skew, finite-state machines, linear programming, optimization, synchronous circuits.

I. INTRODUCTION

Synchronous circuit designers ordinarily try to eliminate clock skew, which may be defined as variations in the delays from the central clock source to the flip-flops (FF's) of the system. This effort can involve equalization of wire lengths, careful screening of off-the-shelf parts, symmetric design of the distribution network, and design guidelines to eliminate skew due to process variations [1], [2]. Clock skew can limit the clocking rate of a synchronous system or cause malfunction at any clock rate. Some static timing analyzers [3] detect

(\(k + 1, r_{k+1,j}^T\)). An inductive proof that BUILDTREE(i, j, D) correctly constructs \(T_{i,j}^T\) should thus be evident. 

**REFERENCES**

incorporate operation in the presence of skew, and allow user tuning of clock and data paths until correct operation is verified.

In this paper, we examine the following question: How can a synchronous system be improved by adjusting the delays between the central clock and individual FF's? We have two goals in mind: 1) To speed up the clock rate at which the circuit will function correctly. 2) In order for a FF to operate correctly when the clock edge arrives at time \( x_i \), it is assumed that there are constants SETUP and HOLD such that correct input data must be present and stable during the time interval \( (x_i - \text{SETUP}, x_i + \text{HOLD}) \).

3) Timing conditions attach to each primary input and output. It is assumed for simplicity that these inputs and outputs are connected to FF's outside the synchronous system, each of which is controlled by the central clock source through its own delay line. Thus, all timing paths begin and end at a FF, making unnecessary a separate terminology for the I/O constraints. FF\(_1\), \ldots, FF\(_K\) denote the internal FF's, and FF\(_{K+1}\), \ldots, FF\(_L\) denote the external FF's. We do not have the ability to vary the clock delays to FF's external to the synchronous system, so while \( x_1, \ldots, x_L \) are variables, \( x_{K+1}, \ldots, x_K \) are constants determined by the circuit and its environment.

4) For \( 1 \leq i, j \leq L \), we compute lower and upper bounds \( \text{MIN}(i, j) \) and \( \text{MAX}(i, j) \) for the time that is required for a signal edge to propagate from FF\(_i\) to FF\(_j\). Since it is possible that multiple paths exist from FF\(_i\) to FF\(_j\), \( \text{MIN}(i, j) \) and \( \text{MAX}(i, j) \) must be computed as the minimum and maximum of these path delays. If no such path exists, we define \( \text{MIN}(i, j) = \infty \) and \( \text{MAX}(i, j) = -\infty \) for notational convenience. Although the data delay internal to the FF itself could be included in SETUP and HOLD, we choose to include it in MIN and MAX. Besides simplifying the notation, this is desirable because the FF internal delay can be variable due to data-dependent delay, or due to the construction and output loading of the FF's.

To avoid double-clocking between FF\(_i\) and FF\(_j\), the data edge generated at FF\(_i\) by a clock edge must arrive at FF\(_j\) no sooner than a period of time \( \text{HOLD} \) after the latest possible arrival of the same clock edge. The earliest that the clock edge can arrive at FF\(_j\) is \( \alpha x_i \), the fastest propagation from FF\(_i\) to FF\(_j\) is \( \text{MIN}(i, j) \). The latest arrival time of the clock at FF\(_j\) is \( \beta x_j \). Thus, we have

\[ \alpha x_i + \text{MIN}(i, j) \geq \beta x_j + \text{HOLD}. \]

To avoid zero-clocking, the data edge generated at FF\(_i\) by a clock edge must arrive at FF\(_j\) no later than \( \text{SETUP} \) amount of time before the earliest arrival of the next clock edge. The latest that the clock edge can arrive at FF\(_j\) is \( \beta x_j \), the slowest propagation from FF\(_i\) to FF\(_j\) is \( \text{MAX}(i, j) \), the clock period is \( P \), and the earliest arrival time of the next clock edge at FF\(_j\) is \( \alpha x_j + P \). Hence,

\[ \beta x_j + \text{SETUP} + \text{MAX}(i, j) \leq \alpha x_j + P. \]

### III. Two Linear Programs

#### A. Minimize \( P \) Subject to Clocking Constraints

If we desire to make the period \( P \) as short as possible while satisfying the system of inequalities (1) and (2), and if the values SETUP, HOLD, \( \alpha \), \( \beta \), \( \text{MIN}(i, j) \) and the \( x_i \) for \( i = K+1, \ldots, L \) are assumed to be constant, while \( P \) and the \( x_i \) for \( i = 1, \ldots, K \) are variable, then what we have is a linear program. In a standard form [6], this system is as follows:

\[ \text{LP}_\text{SPEED}: \text{minimize } P \text{ subject to} \]

\[ \alpha x_i - \beta x_j \geq \text{HOLD} - \text{MIN}(i, j), \quad \text{for } i = 1, \ldots, L; \]

\[ \alpha x_j - \beta x_i + P \geq \text{SETUP} + \text{MAX}(i, j), \quad \text{for } i = 1, \ldots, L; \]

\[ x_i \geq \text{MIN}_\text{DEL}, \quad \text{for } i = 1, \ldots, K. \]
B. Maximize Minimum Margin for Error in Clocking Constraints

Clocking hazards can be particularly vexatious because they are potentially intermittent. A system on the verge of double-clocking, for instance, might pass system diagnostics but malfunction at unpredictable times due to fluctuations in ambient temperature or power supply voltage. One way to armor a system against this problem would be to increase the values of the constants SETUP and HOLD in LP_SPEED, at the cost of an increase in the clock period $P$. If, however, $P$ is a fixed quantity it would be desirable to maximize the minimum over all the constraints of the slack, or amount by which the inequality is satisfied. This converts the problem into a maximin problem [6], which can be stated as a linear program in the following way. Introduce a new variable $M$, which is added to each of the main constraint inequalities so that when it is maximized by the program, it will be the minimum slack over all the inequalities.

\[ P \text{ is a fixed quantity it would be desirable to maximize the minimum over all the constraint inequalities so that when it is maximized by the program, it will be the minimum slack over all the inequalities.} \]

\[ \text{LP SAFETY: maximize } M \text{ subject to } \]

\[ \alpha x_i - \beta x_j - M \geq \text{HOLD} - \text{MIN}(i, j), \quad \text{for } i, j = 1, \ldots, L; \]

\[ \alpha x_j - \beta x_i - M \geq \text{SETUP} + \text{MAX}(i, j) - P, \quad \text{for } i, j = 1, \ldots, L; \]

\[ x_i \geq \text{MIN}_\text{DEL}, \quad \text{for } i = 1, \ldots, K. \]

An additional benefit of LP SAFETY is that it maximizes the tolerance of a system to variations in the speed of its parts, thereby lowering manufacturing costs. For a given reliability that is desired in a machine built from off-the-shelf parts, less stringent screening is required. For VLSI systems, LP SAFETY would serve to improve yield in the face of process variations by centering the design away from clocking hazards.

IV. A SIMPLE EXAMPLE

At the current time, there exists no CAD tool that can automatically perform clock skew optimization. However, with the help of the circuit timing simulator ADVICE [8] and the PORT Linear Programming package [10], a simple circuit has been optimized by hand. This is a 4-bit ripple-carry adder with accumulation and input register in 1.25-μm CMOS (Fig. 2). For simplicity, the carry-in is held at zero, and the carry-out is ignored. The four primary inputs $I_{0-3}$ feed the input FF's IFF$_{0-3}$, whose outputs $L_{0-3}$ are the $A$ inputs of the adder. The four adder outputs $S_{0-3}$ are fed to the sum FF's SFF$_{0-3}$, whose outputs $L_{S0-3}$ are primary outputs. $L_{S0-3}$ are also fed back into the $B$ inputs of the adder.

\[ x_i \geq \text{MIN}_\text{DEL}, \quad \text{for } i = 1, \ldots, K. \]

Fig. 2. A ripple-carry adder with input and accumulation register. Various delays are interposed between the central clock source and the clock inputs to the individual FF's.

A. Circuit Characterization

Numerous ADVICE runs yielded estimates for MAX and MIN. It was found that under various conditions, the delay from IFF$_i$ or SFF$_i$ to SFF$_o$ was always bounded by $\text{MIN}(i, j) = 2.1 + 1.5^*(j - i)$ ns and $\text{MAX}(i, j) = 3.2 + 2.7^*(j - i)$ ns. It was assumed that FF's external to the circuit had no clock delay. Delay from an external FF to an input FF was assumed to be exactly $\text{MAX} = \text{MIN} = 6$ ns, while from a sum FF to an external FF was exactly $\text{MAX} = \text{MIN} = 3$ ns. SETUP and HOLD were both set equal to $1$ ns. The delay lines were constructed by the method described in Section VII-C, with fine-tuning by iterative simulations with the delay-lines driving their actual loads in the final circuit. The unskewed adder was the accumulator described above, with all eight FF's connected directly to the central clock source. The skewed adder was the same accumulator with delay lines interposed between the central clock and the individual FF's.

B. Linear Program Solutions

With the measured circuit parameters as given above, LP_SPEED and LP SAFETY were solved by the PORT linear programming software. The program’s solution, in nanoseconds, for LP_SPEED was as follows. The delays to IFF$_{0-3}$ were 0.00, 0.05, 1.55, and 3.05, and to SFF$_{0-3}$ were 0.00, 0.05, 1.55, and 4.15. $P$ was 8.15. For $P = 13.0$, the solution for LP SAFETY was: The delays to IFF$_{0-3}$ were 0.82, 1.50, 2.18, and 2.86, and to SFF$_{0-3}$ were 0.00, 0.68, 1.36, and 2.04. $M$, the safety margin, was 1.92. For both LP SAFETY and LP_SPEED, the limiting constraints at the optimum points came from paths beginning or ending outside the circuit.

C. Performance of the Resulting Circuit

Since LP SAFETY also speeds up the circuit, its solution was selected to construct the delay lines in the skewed adder. An ADVICE simulation at nominal conditions exhibited correct behavior by both adders. Temperature was 25°C, and the clock period was 10 ns. The clock was initially stopped then cycled for three ticks separated by the given clock period, and then stopped again. The sum register was initialized to 0001, the input register to 1111. The values 1111, 0001, and 0001 were made available at the primary inputs for loading into the input register on the three ticks. This caused a signal to travel
the length of the carry chain on both the first and second clock ticks, with values 0000, 1111, and 0000 appearing after the three ticks of the clock on the LS_0 outputs.

A series of simulations was then performed with progressively shorter clock periods, but holding constant the other conditions, to determine the minimum feasible clock period for each circuit. The unskewed adder worked correctly down to a clock period of 9.5 ns before zero-clocking. The skewed adder was able to work correctly at a clock period of 7.5 ns before zero-clocking, or 2.0 ns less than the critical path delay.

How is it possible that the circuit can be run at a clock period less than the critical path delay? The answer, of course, is that a logic path can act as a delay line, containing more than one signal wavefront at a single instant. The simulation of the skewed adder showed that this was in fact happening in the carry chain at the 7.5 ns clock period. This phenomenon has been exploited in a pipelining technique known as maximum-rate pipelining [1], [5], [9], [13].

In maximum-rate pipelining, the clock period is determined by the maximum path delay through the logic, but by the difference between the maximum and minimum delays. When the clock runs at this maximum rate, the pipeline contains more bits of information than FF's. For this reason, the clock in a maximum-rate pipeline cannot be single-stepped or even slowed down significantly. In the present scheme, by contrast, single-stepping is always possible. Any sequence $P_1, P_2, \ldots$ of intervals between clock edges will drive the circuit correctly as long as each $P_i$ is large enough to satisfy (2).

A second series of simulations stressed the adders in the direction of double-clocking by adding variable amounts of additional clock delay to SFF, in the case of the skewed adder, in addition to the 2.04 ns already present. Other conditions were kept unchanged from the nominal case. With clock period held constant at 10 ns, the unskewed and skewed adders were able to tolerate 3.0 and 4.1 ns clock delay to SFF, respectively, before double-clocking took place between IFF and SFF. The extra resilience of the skewed adder was due to the fact that, in each bit position, the speedup was limited not by the maximum-rate pipelining limit, but by boundary constraints. The best results can be obtained by encompassing as much as possible in the synchronous system that is to be optimized. In general computer systems, the boundary might be pushed out to include all of the most tightly coupled, delay-critical parts of the system. At the boundary, additional FF's might be used to decouple the optimized system from the external world.

Adding FF's to a system before optimizing might result in increased throughput as a result of a kind of "poor man's pipelining." For example, one might double the number of FF's in a system by replacing every FF with two connected in series. The optimization procedure would tend to assign more delay to the first one of the pair. If conditions are right (i.e., if MIN is close to MAX in the right places), the end result would be to almost double the clock rate, and hence the throughput. The effect is similar to conventional pipelining because although the transit time of a single data through the system is not decreased, the throughput is increased. Conventional pipelining requires the designer to partition combinational logic into stages of comparably delay. In poor man's pipelining, on the other hand, the clock skew serves to compensate for inequalities among stage delays. This relaxed constraint on stage delays might reduce the number of FF's required by allowing partitions with fewer inter-stage signals.

VI. CLOCK SKEW VERSUS RETIMING

As Fig. 3 illustrates, adding clock delay to an FF has an effect similar to movement of the FF backwards across combinational-logic module boundaries [11]. This movement, called retiming, can be controlled by a mixed-integer linear program [12] to minimize clock period. In this sense, clock skew and retiming are continuous and discrete optimizations with the same effect. Although Fig. 3 illustrates a situation in which the designer can choose between the two transformations, the methods can be extended to do so.

1) Since retiming moves FF's across discrete (and perhaps large) amounts of logic delay, the resulting system can still benefit from (smaller amounts of) clock skew.

2) Retiming to minimize clock period may cause an unacceptable increase in the number of FF's. Retiming to minimize the number of FF's [12] may be preferred if speed can be bought back more cheaply with clock skew.

3) Retiming does not address the problem of double-clocking or process-dependent timing variation (although presumably it could be extended to do so).

4) Efficient linear programming software packages are widely available, and mixed-integer linear programming packages are not.

5) FF's whose inputs are the primary inputs of a system can be skewed but not retimed.

6) FF's integrated with combinational logic in off-the-shelf parts can be skewed (in tandem, if controlled by a single pin) but not retimed.

VII. PROCESS-DEPENDENT CLOCK SKEW IN VLSI

It is assumed in this section that the synchronous system to be optimized is completely contained on a single chip. As has been men-
tioned, the model for detecting clock hazards has a built-in allowance for uncertainty in both logic and clock delays. This allowance and the design centering given by LP_SAFETY can make a system more robust in the face of delay variations. Unfortunately, this allowance is more conservative than necessary for some of the sources of delay variance. For example, the speed of logic gates can vary considerably due to unavoidable variations in the manufacturing process. But linear programs might be modified to take into account process variation, and restrict the uncertainties modeled by gate delays to the more nearly random causes, such as temperature variation or noise. We will now briefly sketch how the linear programs might be modified to take into account process variability.

A. Uniform Variation of Gate Delays

If all the gate delays on a chip vary uniformly, then there is no problem. It is easily seen that if a chip is free of clock hazards, if the delays of all the gates on a chip increase or decrease by some factor, and if the clock rate changes by the same factor, then the resulting clock solution will also be free of clock hazards.

B. Independent Variance of PFET and NFET Drive Capability in CMOS

A more complicated situation occurs in CMOS, where the current drive of NFET's varies somewhat independently of the current drive of PFET's. Shoji [2] solves the problem of the resulting clock skew in CMOS VLSI by means of a more detailed accounting and control of the sources of clock delay. We will call this the "NP-matched-clock solution." Instead of assigning a single delay $x_i$ to the passage of an input-rising edge through a given chain of inverters in the clock distribution network, the delay is separated into two parts: the pulldown delay $n_i$ and the pullup delay $p_i$. $n_i$ is the sum of the delays of the odd inverters in the chain, each of whose NFET's is pulling down its output, and $p_i$ is the sum of the delays of the even inverters, each of whose PFET's is pulling up its output. The delay of a gate is defined to be the difference in time between when the input and output cross 50% of voltage swing. Rather than size the transistors in the chains to equalize $n_i + p_i$, as is done conventionally, a more stringent matching is performed. The transistors in all the chains are sized to make all the $n_i$ equal, and to simultaneously make all the $p_i$ equal. The result is that all the chain delays track each other in the face of independent variance in the NFET and PFET current drive. If the NFET and PFET current drives change by factors of ND and PD, respectively, all the chain delays remain equal:

$$\frac{n_i}{ND} + \frac{p_i}{PD}. \quad (4)$$

Many ADVICE runs for a 1.75-μm CMOS process showed [2] that (4) was reasonably accurate as long as the pullup and pulldown delays were kept balanced by satisfying the inequalities

$$0.35 \leq \frac{p_i}{n_i + p_i} \leq 0.5 \quad (5)$$

which are equivalent to the linear inequalities $n_i - p_i \geq 0$ and $13p_i - 7n_i \geq 0$.

By replacing the clock delay variables $x_i$ with the pulldown and pullup variables $n_i$ and $p_i$, we can modify LP_SAFETY to take into account the NFET/PFET process variation. The resulting program, called CMOS_LP_SAFETY, will allow a finer control over the generation of clock delay, since $n_i$ and $p_i$ are controlled separately, rather than their sum. A similar transformation can be used to convert LP_SPEED into a program CMOS_LP_SPEED that minimizes clock period while avoiding clock hazards across all CMOS processes. It is necessary to sample the NFET/PFET process parameter space at a finite number of points: ND takes on $A$ values $ND_1, ND_2, \ldots, ND_A$, and PD independently takes on $B$ values $PD_1, PD_2, \ldots, PD_B$. Each sample $(a, b)$ can be considered a separate process, and is characterized by its NFET and PFET relative drive parameters, ND and PD. In [2], for example, the process parameter space is broken up into nine processes, with ND taking on the values 0.556, 1.000, and 1.730, and PD independently taking on the values 0.620, 1.000, and 1.630.

The constraint inequalities for CMOS_LP_SAFETY can now be written down by repeating the constraint inequalities of LP_SAFETY for each process. Since SETUP, HOLD, MAX, and MIN will have different values in different processes, they are now functions of $a$ and $b$: SETUP$(a, b)$, HOLD$(a, b)$, MAX$(i, j, a, b)$, and MIN$(i, j, a, b)$. MIN_DEL will be replaced by minimums for the delay line pulldown and pullup delays MIN_DELD and MIN_DELU. For a given $P$, we wish to maximize $M$ by adjusting the pulldown and pullup variables $n_i$ and $p_i$, for $i = 1, \ldots, K$:

$$\text{CMOS LP_SAFETY: maximize } M \text{ subject to}$$

$$\alpha \left( \frac{n_i}{ND} + \frac{p_i}{PD} \right) - \beta \left( \frac{n_i}{ND} + \frac{p_i}{PD} \right)$$

$$-M \geq \text{HOLD}(a, b) - \text{MIN}(i, j, a, b) \quad (6)$$
Both $\alpha$ and $\beta$ in the above program can be much closer to 1 than their counterparts in the original LP_SAFETY program, since they no longer have to account for process variation in the clock delay lines. If in fact $\alpha = \beta = 1$, and as long as $\min(i, j, a, b) \geq \max(i, j, a, b) - p$ for all processes $(a, b)$ and FFs $i$ and $j$, and if $P$ is big enough, the feasible region of CMOS_LP_SAFETY is nonempty. Simply set all the pulldown delays $n_i$ equal to a single nonnegative constant, and all pullup delays $p_i$ to some other nonnegative constant, such that inequality (5) is satisfied. This equalization of pulldown and pullup delays across all clock delay lines is in fact the NP-matched-clock solution. Unlike the NP-matched clock solution, the solution to CMOS_LP_SAFETY does not necessarily have the property that all clock delays track each other across all process variations. However, both the solution to CMOS_LP_SAFETY and the NP-matched-clock solution are in the feasible region of CMOS_LP_SAFETY, and so both represent solutions that avoid clocking hazards in the face of all NFET/PFET process variations. In general, however, the solution to CMOS_LP_SAFETY enjoys a greater margin-of-safety $M$, and hence is relatively more immune to other kinds of delay variation. Likewise, the solution to CMOS_LP_SPEED allows a higher clock rate than the NP-matched-clock solution.

C. Construction of Clock Delay Lines with Given Pullup and Pulldown Delays

This section demonstrates the construction of delay lines in 1.25-$\mu$m CMOS with various values of $n$ and $p$, the pulldown and pullup delays. It is not claimed that this method is in any sense optimal. A more refined procedure involving transistor sizing has been investigated [7]. The purpose here is simply to demonstrate that the range of achievable values is continuous above acceptably small lower bounds for $n$ and $p$, and within bounds on $n/p$. The delay line consists of a chain of an even number of inverters, with a capacitor attached to the output of each of the first two inverters. The inverters are identical in size. This size is made large enough to reduce to an acceptable level the delay variation due to data-dependent capacitance variation in the

Fig. 4. Delay lines with two (●), four (+), and six (×) inverters plotted with respect to (a) pulldown versus pulldown delays, and (b) pulldown delay versus output rise time.

$$
\alpha \left( \frac{n_j + p_j}{ND_y} \right) - \beta \left( \frac{n_i + p_i}{PD_b} \right) - M \geq \max(\min(i, j, a, b) - p, 0) \\
\text{for all processes } (a, b) \text{ and } i = 1, \ldots, L; \\
n_i \geq \min_{i, j, a, b} \text{ and } p_i \geq \min_{i, j, a, b}; \\
n_i - p_i \geq 0 \text{ and } 13p_i - 7n_i \geq 0, \text{ for } i = 1, \ldots, K.
$$
FF. The inverter PFET/NEFET size ratio was set to equalize pullup and pulldown delays. The capacitor is constructed from the gate of a FET, with source and drain tied to ground. If only the total delay is to be controlled, then only the capacitor on the first inverter is necessary. Inverter chains with two, four, and six inverters were simulated by ADVICE, with each capacitor constructed with FET gate areas taking on the values 0, 100, . . . 900 μm². Each simulation measured not only n and p, but also the rise time (time from crossing 10% to 90% of voltage swing) of the delay line output. Rise time is of interest because too large a value can increase FF internal delay. Fig. 4(a) plots these 300 delay lines with respect to the n and p values that were achieved. Fig. 4(b) plots the delay lines with respect to p and the rise time of the delay line output, for the 30 delay lines with maximum capacitance attached to the first inverter. With two inverters, the delay line exhibits quite a large variation in rise time, as the second capacitance is varied. Depending on the technology and application, this might not be acceptable, and it may be necessary to use at least four inverters. Four inverters with no extra capacitance yield n = -0.58 and p = 0.62 ns.

VIII. CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

Inequalities (1) and (2) govern the correct operation of synchronous systems. The conventional approach of eliminating clock skew is a feasible point of the linear programs generated by these inequalities. In general, however, this point is neither the fastest nor the safest. These can be discovered by solving the linear programs. The optimized system can be constructed with little extra cost, and will provide faster and more reliable operation than a conventionally-clocked system.

It is not known how fast the number of constraints grows with circuit size. This number cannot be greater than twice the square of the number of FF’s, but for practical circuits may be much smaller. If the linear program becomes too large, it may become necessary to investigate solution procedures that take advantage of the special form of the constraints.

The current approach should be extended to higher performance clocking schemes, such as one-phase level-sensitive latches [1], [5]. This high performance brings with it an increased susceptibility to double-clocking, but the current approach explicitly guards against this danger.

Ideally, all variables should be considered jointly when optimizing a design. Although this is usually impossible, progress is made when two or more formerly separate optimization steps are joined into one. Clock skew and retiming are both of a linear character. It is likely that efficient procedures could be given for optimizing systems by jointly considering both sets of variables.

A third linear optimization, insertion of delay lines in combinational logic, has also been studied [13]. Although this work was in the context of maximum-rate pipelining, logic delay lines can provide additional safety margin against clock skew, as well as enhance the ability of clock skew to reduce the clock period of a single-steppeable machine, by reducing \[ \max(i, j) - \min(i, j) \]

Transistor sizing in CMOS has been shown [14] to be a posynomial [15] programming problem. Posynomial programs, though generally nonlinear, can always be transformed into convex programs, and thus enjoy the property that a local minimum is guaranteed to be a global minimum. Unfortunately, when both transistor sizes and clock delays are varied, the result is a signomial program [15]. A signomial program is not necessarily equivalent to a convex program. A proof of equivalence for this problem would open up the possibility of efficiently optimizing all classes of variables jointly: clock delays, FF positions, logic delay lines, and logic-gate transistor sizes.

ACKNOWLEDGMENT

Thanks to E. Rosenberg for enlightening me about max-min programs, to N. Schryer and L. Kaufman for providing the PORT linear programming software, and to J. Chardross, R. Finkel, L. Fishburn, J. Javitt, T. Szymanski, V. Visvanathan, S. Vogel and the four reviewers for helpful comments.

REFERENCES


Performance Analysis of a Message-Oriented Knowledge-Base

WANG-CHAN WONG, TATSUYA SUDA, AND LUBOMIR BIC

Abstract — We present a message-driven model for function-free Horn logic, where the knowledge base is represented as a network of logical processing elements communicating with one another exclusively through messages. The lack of centralized control and centralized memory makes this model suitable for implementation on a highly-parallel asynchronous computer architecture.

Manuscript received August 15, 1987; revised March 12, 1988 and October 5, 1989. W.-C. Wong and L. Bic were supported in part by the National Science Foundation under Grants DCR-8503589 and CCR-8709817. T. Suda was supported in part by the National Science Foundation under Grants DCT-8602052 and NCR 8907909, and is also supported by the University of California MICRO program.

The authors are with the Department of Information and Computer Science, University of California, Irvine, CA 92717.

IEEE Log Number 904552.