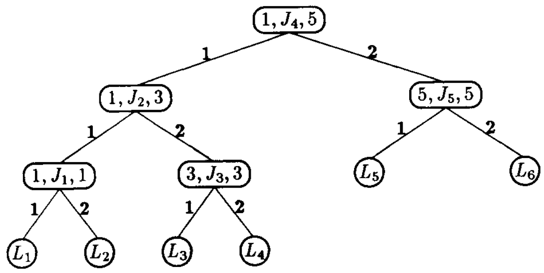

Fig. 3. An optimal testing tree for Example 3.

TABLE II
COMPUTATION DATA FOR EXAMPLE 3

l \ i	1	2	3	4	5
0	$\boxed{r^1_{11} = J_1}$ $r^2_{11} = J_1$	$r^1_{22} = J_2$ $r^2_{22} = J_2$	$r^1_{33} = J_3$ $\boxed{r^2_{33} = J_3}$	$r^1_{44} = J_4$ $r^2_{44} = J_4$	$r^1_{55} = J_5$ $\boxed{r^2_{55} = J_5}$
1	$r^1_{12} = J_2$ $r^2_{12} = J_1$	$r^1_{23} = J_3$ $r^2_{23} = J_2$	$r^1_{34} = J_4$ $r^2_{34} = J_3$	$r^1_{45} = J_5$ $r^2_{45} = J_4$	
2	$\boxed{r^1_{13} = J_2}$ $r^2_{13} = J_1$	$r^1_{24} = J_4$ $r^2_{24} = J_2$	$r^1_{35} = J_5$ $r^2_{35} = J_4$		
3	$r^1_{14} = J_3$ $r^2_{14} = J_2$	$r^1_{25} = J_4$ $r^2_{25} = J_2$			
4	$r^1_{15} = J_4$ $\boxed{r^2_{15} = J_4}$				

Fig. 3. The highlighted entries in the table correspond to the internal vertices created by the procedure BUILDTREE. The expected cost of this optimal testing tree is 7.6, as opposed to 8.0 for the testing tree in Fig. 2. □

Theorem 1: The algorithm constructs an optimal testing tree with time complexity $O(n^3)$ and space complexity $O(n^2)$.

Proof: For the time complexity, once we have computed the table of $r_{ij}^{1(2)}$, at Step 2 of the main routine, we construct a testing tree T_{1n}^2 with the recursive procedure BUILDTREE. This requires only $O(n)$ operations because the binary tree T_{1n}^2 has exactly n internal vertices, implying that there are only n calls of the procedure where each call takes constant time. The most time-consuming part is Step 1 of the main routine. At each iteration, it requires $O(j-i) = O(l)$ time to compute $C_{ij}^{1(2)}$ and constant time to compute all the other instructions. The outer loop is executed at most n times where the inner loop is executed at most n times for each iteration of the outer loop. Thus, the time complexity of the algorithm is $O(n^3)$.

For the space complexity, first observe that we need $O(n^2)$ space to store the input data. The size of the table for temporarily storing $C_{ij}^{1(2)}$ and $r_{ij}^{1(2)}$, $1 \le i \le j \le n$, is $O(n^2)$. Note that at Step 1 of the main routine, each iteration of the outer loop requires the computation results from earlier iterations. Finally, we need $O(n)$ space to store the output, an optimal testing tree. Thus, the total space complexity of the algorithm is $O(n^2)$.

For the correctness of the algorithm, note that from (2) and (3), $C_{1n}^1 = C_{1n}^2$ corresponds to the expected cost of an optimal testing tree. It is obvious that $r_{ij}^{1(2)}$ and $C_{ij}^{1(2)}$ are correctly computed at Step 1 of the main routine. To see that BUILDTREE$(1, n, 2)$ correctly constructs a testing tree (that is, T_{1n}^2) with expected cost C_{1n}^2, observe that the root v of T_{ij}^D constructed is labeled with $[i, J_k = r_{ij}^D, j]$, $D \in \{1, 2\}$. Then, the left (right) son of v will be a leaf vertex labeled with L_i (L_{j+1}) if $i = k$ $(j = k)$, or, otherwise, the root of a subtree $T_{i,k-1}^1$ $(T_{k+1,j}^2)$ labeled with $[i, r_{i,k-1}^1, K-1]$ $([k+1, r_{k+1,j}^2, j])$. An inductive proof that BUILDTREE(i, j, D) correctly constructs T_{ij}^D should thus be evident. □

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

[1] R. Rey, "Engineering and operations in the Bell Systems," AT&T Bell Labs., Murray Hill, NJ, 1983.
[2] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.
[3] R. Cartwright, "4-TEL automated subscriber line test system," in *Proc.* 1982 *Int. Symp. Subscriber Loops Syst.*, Toronto, Ont., Canada, Sept. 1982.
[4] M. Garey and F. Hwang, "Isolating a single defective using group testing," *J. Amer. Statist. Ass.*, vol. 69, no. 345, pp. 151-153, Mar. 1974.
[5] D. Siewiorek and R. Swarz, *The Theory and Practice of Reliable System Design*. Bedford, MA: Digital, 1982.

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

Manuscript received September 9, 1987; revised March 31, 1988 and June 6, 1988.
The author is with AT&T Bell Laboratories, Murray Hill, NJ 07974.
IEEE Log Number 9034544.

0018-9340/90/0700-0945$01.00 © 1990 IEEE

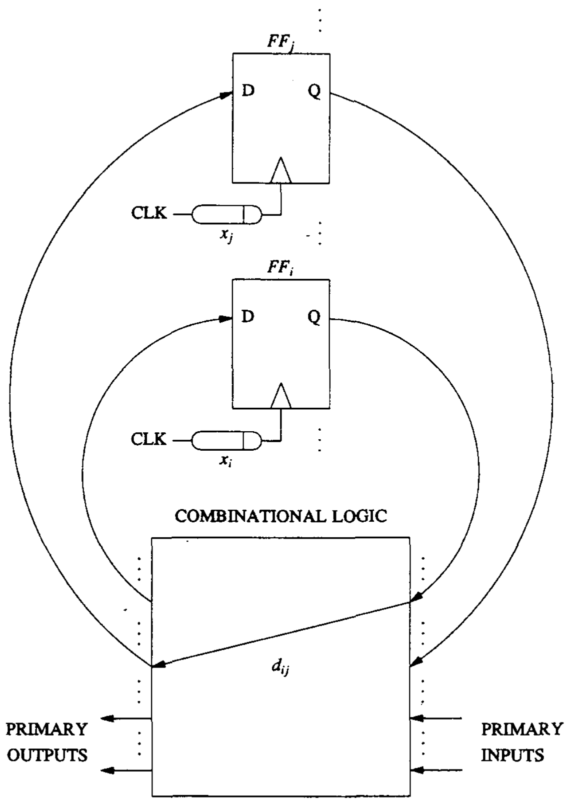

Fig. 1. Synchronous digital system with individual delays x_i interposed between the central clock source and the FF's of the synchronous system. For each FF pair (FF_i, FF_j), bounds MIN(i, j) and MAX(i, j) are computed for the time-of-flight d_{ij} from FF_i to FF_j.

incorrect 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) For a given clocking rate, to maximize the margin of safety against circuit malfunction due to clocking hazards. The analysis will use, for simplicity, positive-edge-triggered *D*-flip-flops.

II. Clock Hazards

A. Double-Clocking and Zero-Clocking

In 1965, Cotten [4] described a "data race" mechanism in which clock skew can cause a synchronous system to fail. In Fig. 1, if $x_j > x_i + d_{ij}$, then when the positive clock edge arrives at FF_i, the data "race ahead" through the fast path, destroying the data at the input to FF_j before the clock gets there. When the clock edge finally arrives at FF_j, the wrong data are clocked through. Since the data are clocked through two FF's with one clock edge, this has also been called *double-clocking*.

Analogously, *zero-clocking* can be used to designate the case when the data reach the FF too late relative to the *next* clock edge. This occurs in Fig. 1 when $x_i + d_{ij} > x_j + P$, where P is the clock period.

B. General Model for Detecting Clock Hazards

This section develops a set of inequalities that can tell us, in general, whether either of the above hazards is present. This development is similar to that found in [1] and [5], except that a minimum and maximum delay is calculated for every source/destination FF pair, as opposed to assuming overall delay bounds between banks of FF's. This is necessary for the linear programs that follow. Fig. 1 illustrates the basic components of the model:

1) FF_i receives the central clock delayed x_i by its own delay element. A later section will outline a method for delay line construction. Depending on the technology, there is some minimum delay MIN_DEL that can be generated: $x_i \geq$ MIN_DEL. We will assume some uncertainty in the clock delays. There will be two constants, $0 < \alpha \leq 1 \leq \beta$, with the property that if the nominal clock delay is x_i, then the actual clock delay x can vary from clock edge to clock edge, but must always fall in the interval $\alpha x_i \leq x \leq \beta x_i$.

2) In order for a FF to operate correctly when the clock edge arrives at time x, it is assumed that there are constants SETUP and HOLD such that correct input data must be present and stable during the time interval ($x -$ SETUP, $x +$ 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. $FF_1, \cdots, FF_K$ denote the internal FF's, and $FF_{K+1}, \cdots, FF_L$ 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 $x_1, \cdots, x_K$ are variables, $x_{K+1}, \cdots, x_L$ are constants determined by the circuit and its environment.

4) For $1 \leq i, j \leq L$, we compute lower and upper bounds MIN(i, j) and 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, MIN(i, j) and MAX(i, j) must be computed as the minimum and maximum of these path delays. If no such path exists, we define MIN(i, j) $= \infty$ and 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 HOLD after the latest possible arrival of the same clock edge. The earliest that the clock edge can arrive at FF_i is αx_i, the fastest propagation from FF_i to FF_j is MIN(i, j). The latest arrival time of the clock at FF_j is βx_j. Thus, we have

$$\alpha x_i + \mathrm{MIN}(i, j) \geq \beta x_j + \mathrm{HOLD}. \tag{1}$$

To avoid zero-clocking, the data edge generated at FF_i by a clock edge must arrive at FF_j no later than SETUP amount of time before the earliest arrival of the next clock edge. The latest that the clock edge can arrive at FF_i is βx_i, the slowest propagation from FF_i to FF_j is 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,

$$\beta x_i + \mathrm{SETUP} + \mathrm{MAX}(i, j) \leq \alpha x_j + P. \tag{2}$$

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, α, β, MAX(i, j), MIN(i, j) and the x_i for $i = K+1, \cdots, L$ are assumed to be constant, while P and the x_i for $i = 1, \cdots, K$ are variable, then what we have is a *linear program*. In a standard form [6], this system is as follows:

LP_SPEED: minimize P subject to

$$\alpha x_i - \beta x_j \geq \mathrm{HOLD} - \mathrm{MIN}(i, j), \qquad \text{for } i, j = 1, \cdots, L;$$

$$\alpha x_j - \beta x_i + P \geq \mathrm{SETUP} + \mathrm{MAX}(i, j), \qquad \text{for } i, j = 1, \cdots, L;$$

$$x_i \geq \mathrm{MIN\_DEL}, \qquad \text{for } i = 1, \cdots, K.$$

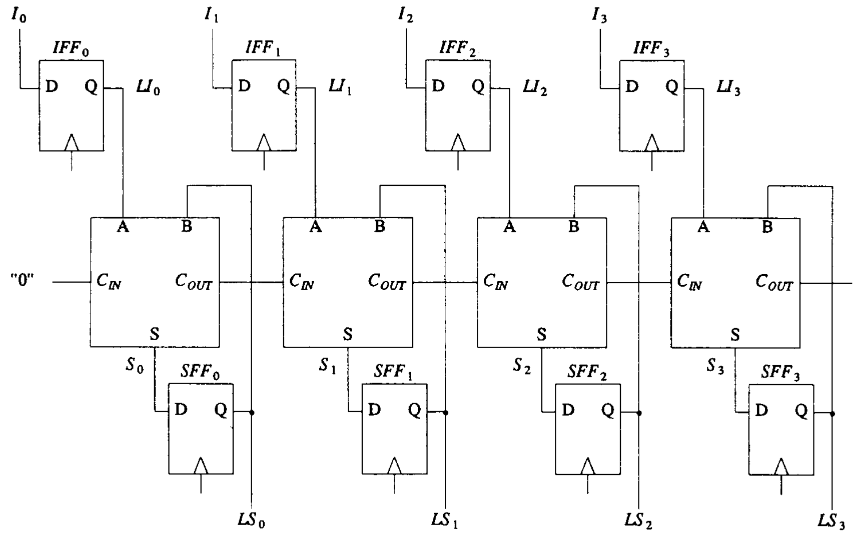

Fig. 2. A ripple-carry adder with input and accumulation register. Various delays are interposed between the central clock source and the clock inputs to the individual FF's.

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 is now a constant. The variables M and x_i, $i = 1, \cdots, K$ are to be determined.

LP_SAFETY: maximize M subject to

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

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

$$x_i \geq \mathrm{MIN\_DEL}, \quad \text{for } i = 1, \cdots, K.$$

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 LI_{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 LS_{0-3} are primary outputs. LS_{0-3} are also fed back into the B inputs of the adder.

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_j was always bounded by $\mathrm{MIN}(i, j) = 2.1 + 1.5^*(j-i)$ ns and $\mathrm{MAX}(i, j) = 3.2 + 2.7^*(j-i)$ 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 MAX = MIN = 6 ns, while from a sum FF to an external FF was exactly MAX = 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_{0-3} 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 not 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 $P_1, P_2, \cdots$ 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_3 (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_3, respectively, before double-clocking took place between IFF_3 and SFF_3. The extra resilience of the skewed adder was due to the fact that, in each bit position i, LP_SAFETY had assigned more clock delay to IFF_i than to SFF_i, thus centering the circuit away from double-clocking.

V. Some Practical Considerations

A CAD tool that optimizes clock delays would include a static timing analyzer that could compute MIN(i, j) and MAX(i, j) between an input i and an output j of combinational logic. In this regard, most static timing analyzers that exist today are deficient in two respects. First, lower bounds on delay are usually not computed, although formulas for lower bounds on RC network delays are available [16]. Second, delays are not computed per input–output pair. Rather, the user specifies particular data-ready times at inputs, and the analyzer computes resulting output times. A static timing analyzer that can compute maximum delays in this manner could be used as a subroutine in an algorithm to compute MAX. For any input i, MAX(i, j) could be computed for all j by setting the data-ready time of input i to zero, and all other inputs to $-\infty$. MAX(i, j) can then be assigned the resulting data-ready time at output j. A similar procedure could compute MIN using a static timing analyzer that could compute minimum delays.

If a system is to be built from off-the-shelf parts, only one clock delay variable can be attached to all the FF's controlled by a package pin. This may force a suboptimal solution, but the linear program can still correctly model the situation by coalescing into one the variables associated with these FF's.

Calculation of logic delays involves many uncertainties. The delays of both combinational logic and the clock distribution network are affected by process, temperature, noise, and voltage variation. Even if these physical parameters could be held constant, inaccuracies remain in the models used by the static timing analyzer. These uncertainties limit the performance that can be obtained from optimizing clock delays. It should be stressed, however, that these uncertainties have been explicitly taken into account in the linear programs presented here, through the use of upper and lower bounds on delay. The only delay uncertainty that does not equally affect the performance of conventionally clocked systems is that associated with the added clock delay lines. One scheme to generate these delays might be as follows. Between the central clock source and the FF's, the distribution network would be similar to that in conventional clock distribution networks. Every effort would be made to equalize the clock delay in this network. This network would have a certain distribution delay DD, and the linear program would have the constraints $x_i \geq$ MIN_DEL replaced by $x_i \geq$ DD + MIN_DEL1. The variable delay elements (Section VII-C) would be inserted between the leaves of this network and the actual FF's. This scheme allows the variable part of each delay to be in one spot (at the FF), avoiding the additional uncertainties that would be introduced if it were distributed.

In general, partitioning a system and optimizing the pieces separately can give suboptimal results. We saw this with the adder, where 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 could 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 datum through the system is not decreased, the throughput is increased. Conventional pipelining requires the designer to partition combinational logic into stages of comparable 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 interpartition 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 two methods can in general complement each other:

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, but 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-

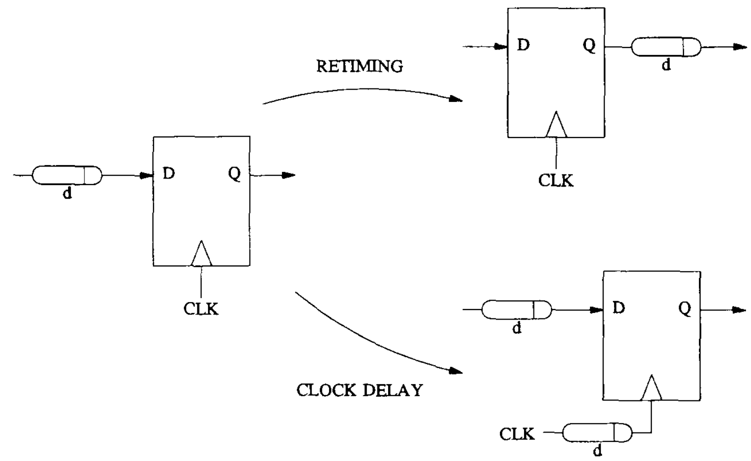

Fig. 3. Retiming and clock delay transformations applied to the same circuit.

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 a single silicon chip will tend to have very little process variation across its surface. For example in CMOS, the drive capability of a fixed-size N-channel or P-channel MOS transistor (NFET or PFET) varies insignificantly as a function of its position on the chip. The clock hazard model, on the other hand, allows the delay of a fixed-size logic gate with fixed output load to vary from place to place, and even from clock edge to clock edge. A more reasonable model would account explicitly for 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 variation.

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 chip 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 i 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 all 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:

$$\frac{n_i}{\mathrm{ND}} + \frac{p_i}{\mathrm{PD}}. \tag{4}$$

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

$$0.35 \le \frac{p_i}{n_i + p_i} \le 0.5 \tag{5}$$

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

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 $\mathrm{ND}_1, \mathrm{ND}_2, \cdots, \mathrm{ND}_A$, and PD independently takes on B values $\mathrm{PD}_1, \mathrm{PD}_2, \cdots, \mathrm{PD}_B$. Each sample (a, b) can be considered a separate process, and is characterized by its NFET and PFET relative drive parameters, ND_a and PD_b. 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 $\mathrm{MIN\_DEL}_D$ and $\mathrm{MIN\_DEL}_U$. For a given P, we wish to maximize M by adjusting the pulldown and pullup variables n_i and p_i, for $i = 1, \cdots, K$:

CMOS_LP_SAFETY: maximize M subject to

$$\alpha\left(\frac{n_i}{\mathrm{ND}_a} + \frac{p_i}{\mathrm{PD}_b}\right) - \beta\left(\frac{n_j}{\mathrm{ND}_a} + \frac{p_j}{\mathrm{PD}_b}\right) - M \ge \mathrm{HOLD}(a, b) - \mathrm{MIN}(i, j, a, b) \tag{6}$$

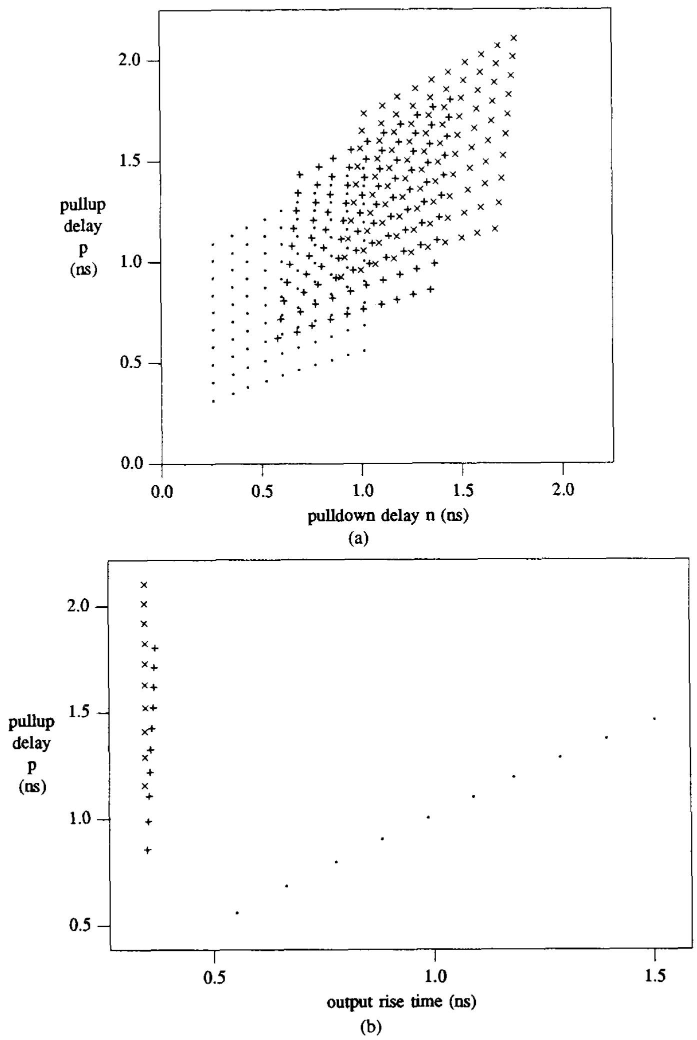

Fig. 4. Delay lines with two (●), four (+), and six (×) inverters plotted with respect to (a) pullup versus pulldown delays, and (b) pullup delay versus output rise time.

$$\alpha\left(\frac{n_j}{\mathrm{ND}_a}+\frac{p_j}{\mathrm{PD}_b}\right)-\beta\left(\frac{n_i}{\mathrm{ND}_a}+\frac{p_i}{\mathrm{PD}_b}\right)$$
$$-M \geq \mathrm{SETUP}(a, b)+\mathrm{MAX}(i, j, a, b)-P \qquad (7)$$
$$\text{for all processes } (a, b) \text{ and for } i, j = 1, \cdots, L;$$
$$n_i \geq \mathrm{MIN\_DEL}_D \text{ and } p_i \geq \mathrm{MIN\_DEL}_U, \qquad \text{for } i = 1, \cdots, K;$$
$$n_i - p_i \geq 0 \text{ and } 13p_i - 7n_i \geq 0, \qquad \text{for } i = 1, \cdots, K.$$

Both α and β 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 $\mathrm{MIN}(i, j, a, b) \geq \mathrm{HOLD}(a, b)$ for all processes (a, b) and FF's 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-μm 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

FF. The inverter PFET/NFET 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 $0, 100, \cdots, 900\ \mu m^2$. 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 higher 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 double-clocking, as well as enhance the ability of clock skew to reduce the clock period of a single-steppable machine, by reducing $\mathrm{MAX}(i, j) - \mathrm{MIN}(i, j)$.

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 four 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 maxi-min programs, to N. Schryer and L. Kaufman for providing the PORT linear programming software, and to J. Chandross, R. Finkel, L. Fishburn, J. Javitt, T. Szymanski, V. Visvanathan, S. Vogel and the four reviewers for helpful comments.

References

[1] P. M. Kogge, *The Architecture of Pipelined Computers.* New York: McGraw-Hill, 1981, pp. 21–39.

[2] M. Shoji, "Elimination of process-dependent clock skew in CMOS VLSI," *IEEE J. Solid-State Circuits,* vol. SC-21, no. 5, pp. 875–880, Oct. 1986.

[3] T. M. McWilliams, "Verification of timing constraints on large digital systems," *J. Digital Syst.,* vol. 5, no. 4, pp. 401–427, 1981.

[4] L. W. Cotten, "Circuit implementation of high-speed pipeline systems," in *AFIPS Proc. 1965 Fall Joint Comput. Conf.,* vol. 27, pp. 489–504.

[5] M. J. Flynn and S. Waser, *Introduction to Arithmetic for Digital Systems Designers.* CBS College Publishing, 1982, pp. 215–222.

[6] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization.* New York: Academic, 1981.

[7] P. V. Argade, "Sizing an inverter with a precise delay: Generation of complementary signals with minimal skew and pulse width distortion in CMOS," *IEEE Trans. Comput.-Aided Design,* vol. CAD-8, no. 1, pp. 33–40, Jan. 1989.

[8] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Memo ERL-M520, Univ. of California, Berkeley, May 9, 1975.

[9] L. W. Cotten, "Maximum-rate pipeline systems," in *AFIPS Proc. 1969 Spring Joint Comput. Conf.,* vol. 34, pp. 581–586.

[10] P. A. Fox and N. L. Schryer, "The PORT mathematical subroutine library," *ACM Trans. Math. Software,* vol. 4, no. 2, pp. 104–126, June 1978.

[11] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits.* Reading, MA: Addison-Wesley, 1985, pp. 345–347.

[12] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. Third Caltech Conf. Very Large Scale Integration,* R. Bryant, Ed., 1983, pp. 87–116.

[13] B. C. Ekroot, "Optimization of pipelined processors by insertion of combinational logic delay," Ph.D. dissertation, Dep. Elec. Eng., Stanford Univ., Sept. 1987.

[14] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD-85),* Santa Clara, CA, Nov. 1985, pp. 326–328.

[15] J. G. Ecker, "Geometric programming: Methods, computations and applications," *SIAM Rev.,* vol. 22, no. 3, pp. 338–362, July 1980.

[16] J. Rubinstein, P. Penfield, and M. Horowitz, "Signal delay in RC tree networks," *IEEE Trans. Comput.-Aided Design,* vol. CAD-2, no. 3, pp. 202–211, July 1983.

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 DCI-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 9034552.

0018-9340/90/0700-0951$01.00