Pass-transistor networks optimize n-MOS logic

Formal methods for transfer-gate logic design achieve minimum area, delay, and power for complex circuits

by Sterling Whitaker, *American Microsystems Inc., Santa Clara, Calif.*

□ Conventional methods of designing logic chips employ blocks of discrete and small-scale integrated circuits—an approach that is now crumbling under the impact of very large-scale integration. Designers who set out to create high-performance cost-effective VLSI chips must minimize the power, delay, and area of MOS ICs. But traditional logic design, with its black-box representation of Boolean functions, does not shrink them.

However, these three parameters can be minimized by experimenting with the many combinatorial logic circuits, transfer gates, and MOS pass transistors that affect an IC's power consumption, speed, and size. Systematically designed pass-transistor networks can reduce complex functions to highly regular structures that operate more quickly than conventional n-channel MOS logic, fill only one third as much space, and consume only one eighth as much power.

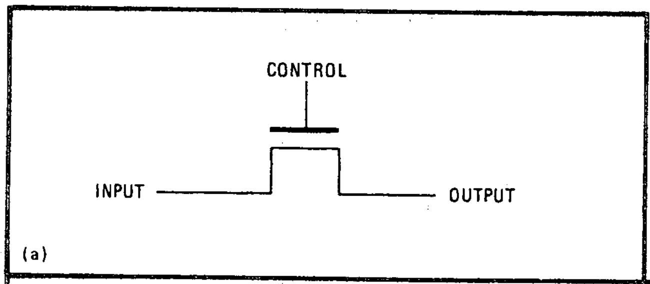
Designing pass-transistor networks has been more a craft than a science. Early masters—Carver Mead and Lynn Conway among them—popularized pass transistors in latches, flip-flops, multiplexers, and some combinatorial logic structures [*Electronics*, Oct. 20, 1981, p. 102]. The more formal design techniques presented here extend the benefits of pass transistors to networks too complex to be realized intuitively, and they also permit informally designed pass networks to be verified in a systematic way.

The basics

Figure 1 shows the pass gate and its logical function. When control inputs are high the transistor conducts, passing the logic level at its input to its output. When the control input is low the transistor is off, leaving the output in a high-impedance (or undefined) state. Designers form pass-gate networks (see Fig. 1c) by joining together the outputs of pass transistors to feed the inputs of succeeding transistors.

A high signal level at a pass transistor's input is reduced by a threshold voltage at the output. This reduced level is passed, without further degradation, through additional pass transistors. As a result of this reduced signal, conventional buffers, not pass-gate outputs, drive control inputs.

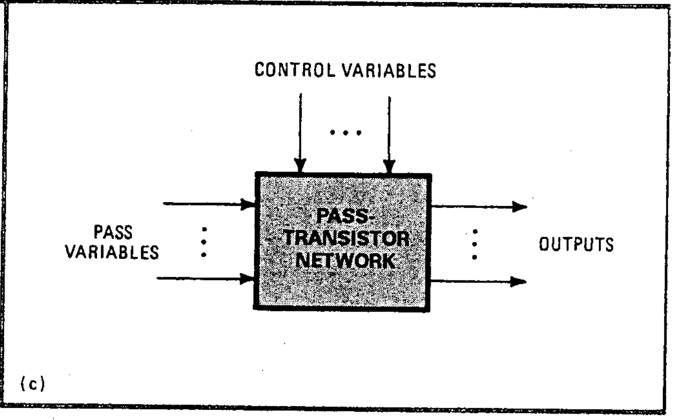
Consider an example of formal pass-network design: an exclusive-NOR gate (Fig. 2). Added to an ordinary truth table is a column that indicates, for each input state, which input variables (or complements) can be passed to the output to get a desired function. When



PASS-GATE TRUTH TABLE

INPUT	CONTROL	OUTPUT
0	0	UNDEFINED
1	0	UNDEFINED
0	1	0
1	1	1

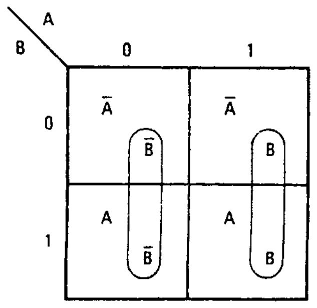
(b)



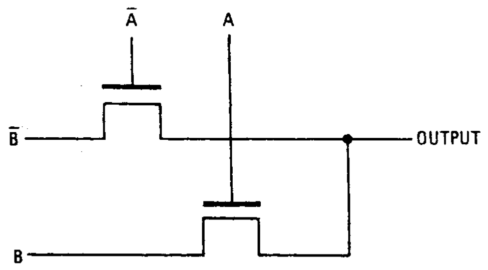
1. Pass gate. When an MOS transistor is operated as a pass gate, as shown in (a), the device passes the signal at its drain to its source. As the truth table given in (b) demonstrates, the output is in a high-impedance, or undefined, state when the transistor is shut off. Networks of pass gates make pass and control variables flow at right angles (c).

EXCLUSIVE NOR TRUTH TABLE			
A	B	OUTPUT	PASS FUNCTION
0	0	1	$\overline{A}+\overline{B}$
0	1	0	$A+\overline{B}$
1	0	0	$\overline{A}+B$
1	1	1	$A+B$

(a)



(b)



(c)

2. Straightforward logic. With the aid of a truth table (a) that includes pass functions, it is possible to modify conventional logic design for pass transistors. The pass functions are entered in a Karnaugh map (b) with pass variables looped together. The resulting exclusive-NOR gate needs just two devices (c).

both A and B equal 0 the output should be 1. Either $\overline{A}$ or $\overline{B}$ can be passed to the output. $\overline{A} + \overline{B}$ is called the pass function.

For each state of the input variables, pass functions are then entered into corresponding cells of a conventional Karnaugh map (Fig. 2b). To steer the pass variables to the output, they are grouped to produce the control functions that drive the pass-gate control inputs.The left loop in Fig. 2b groups $\overline{B}$ under the control of $\overline{A}$; the other one groups B under the control of A.

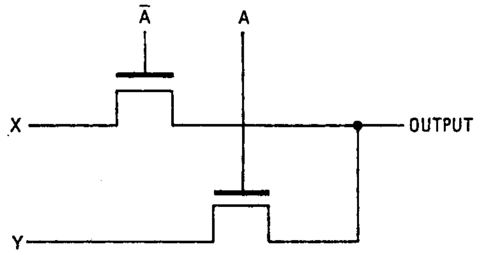
Figure 2c shows the resulting pass-transistor network, which has one pass-transistor delay and comprises two devices. The network's steady-state power dissipation—consisting only of leakage currents between the substrate and the transistors' source and drain regions—is negligible. Figure 3 shows the other basic logic functions—AND, NAND, OR, NOR, and exclusive-OR—that can be derived in the manner of exclusive-NOR.

Contrast this simplicity with the traditional n-MOS version, which comprises five transistors, is plagued by pull-down or pull-up delays, and has one node that dissipates steady-state power. Complementary-MOS gates do not dissipate steady-state power, but they do comprise eight transistors and have a pull-up or -down delay.

Synthesizing pass networks

Logic functions that are more complex can be approached as the simple gates are: with a truth table that includes the pass functions and with a modified Karnaugh map to derive control functions. The control-function groupings in the Karnaugh map for a pass network can be reduced with techniques that resemble minterm reduction in classical logic design. But the reduction rules differ in three ways from those of conventional Karnaugh maps.

For one thing, a pass transistor's output is not defined when its control gate is not asserted, so a variable must be passed in every map state for which the output has been defined. (In a conventional map, only those states with true outputs must be looped.) Then, too, more than one variable may be passed in one state because the pass

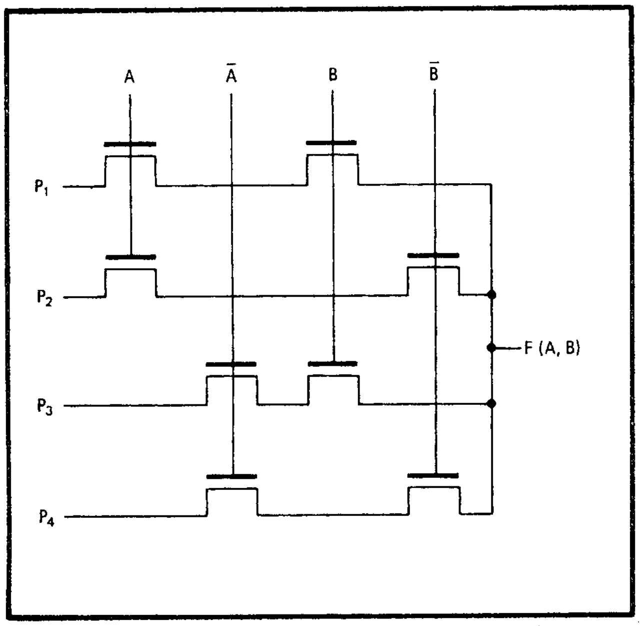


(a)

OUTPUT	X	Y
AB	0	B
$\overline{AB}$	1	$\overline{B}$
A+B	B	1
$\overline{A+B}$	$\overline{B}$	0
$A \oplus B$	B	$\overline{B}$

(b)

3. Fundamentals. Like the exclusive-NOR gate of Fig. 2, the five other basic logic functions can be implemented with the assistance of just two transistors. The X and Y inputs in (a) take the values shown in the table (b) to produce the logic functions.



4. Function generator. Each state of the input word AB connects just one of the four pass lines, P_1 through P_4, to the output. Any function of A and B can therefore be implemented with the proper choice of logic values for the pass lines.

functions in the map ensure that the passed variables are all at the same logic level. Finally, once a "don't care" state has been included in a loop, it acquires a pass function determined by the variable being looped.

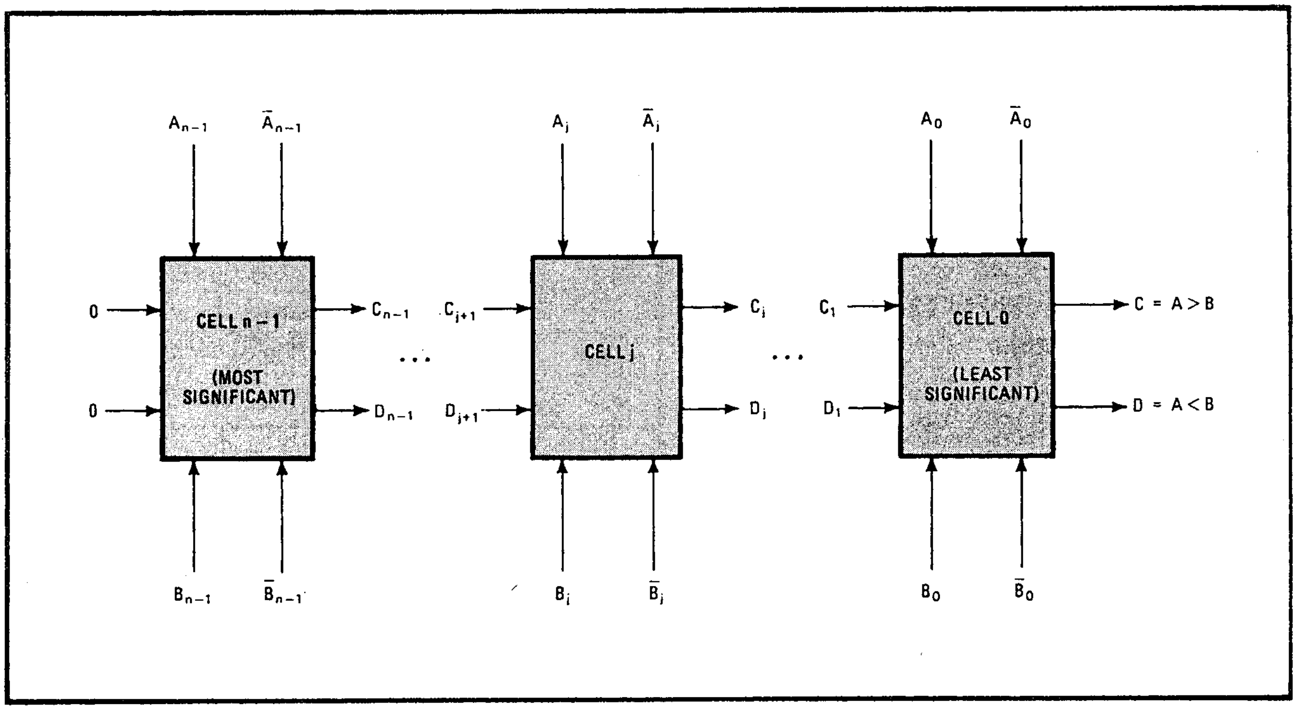
The input variables must be divided between two sets—pass variables and control variables—for modified Karnaugh maps to be used in pass-network design. Maps do not always make it clear whether or not such a division is worth making, especially with more than a few variables. One thing can help: understanding the connection between pass networks and the canonical sum-of-products form of Boolean equations.

All possible functions of variables can be synthesized in a single pass-transistor network. Mead and Conway discussed such a function generator for an arithmetic and logic unit. Consider Fig. 4, for example. In each of four possible states of control variables A and B, one of pass lines P_1 through P_4 is connected to the output. To implement the function "A exclusive-OR B," for example, input 0110 is applied to the pass lines.

Input determines function

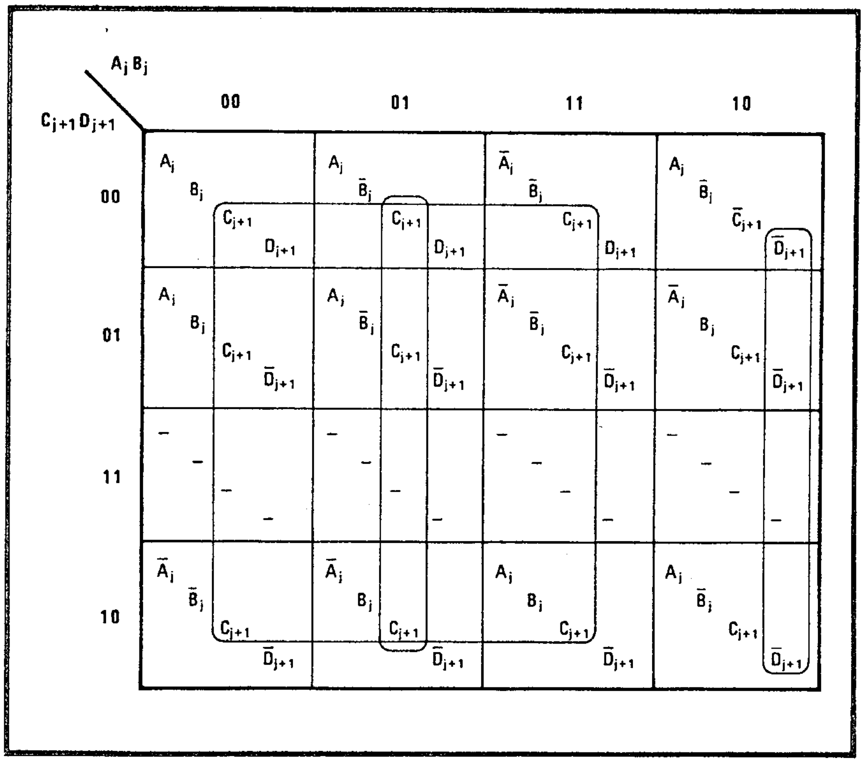
This circuit is useful because different functions can be implemented just by placing the proper input on the pass lines. In fact, the scheme can be extended to implement all the functions of N variables, which require 2^N pass lines and $N2^N$ transistors. Many random-logic designs do not need this flexibility. But they do need to implement the function with the least possible area, power, delay—and effort.

In the network shown in Fig. 4, properly assigning a third variable—C—or its complement, or the constants 1 or 0 to the pass lines achieves any particular function of three variables. (This claim can readily be verified with the canonical sum-of-products form for Boolean logic functions.) Of the eight minterms that can be formed from three variables, four at most enter any particular function. A fifth minterm can always be reduced in combination with one of the others. Each pass line of the network implements a three-term product. (The first is ABP_1.) Since every possible state of inputs



5. Comparator in operation. Bit by bit, an iterative array compares two digital words A and B. The comparison of the most significant bits (left) provides an intermediate result that passes to the right and is used in comparing the next bits. Each cell performs the same operations, and the final result is available from the least significant cell.

6. Map. For cell j of the comparator shown in Fig. 5, output C_j is mapped as a function of the corresponding bits of A and B and of outputs C_{j+1} and D_{j+1} from the previous cell. The loops (color) show that $\bar{A}_j+B_j$ passes C_{j+1}, while $A_j\bar{B}_j$ passes $\bar{D}_{j+1}$.



A and B leads to the selection of some pass line, the output is always defined.

Moreover, a particular function of N variables can always be implemented with 2(N−1) control lines, 2^{N-1} pass lines, and $N2^{N-1}$ transistors. This, however, is a worst case. Fewer than the maximum number of minterms may be present, and some can be reduced in combination.

When written in what might be called the "pass canonical form," Boolean equations can generally be translated straight into pass-transistor networks. Pass networks, as their structures show, directly implement a Boolean equation of the form:

$$F = P_1F_1(C_1, \ldots, C_m) + \ldots + P_nF_n(C_1, \ldots, C_m)$$

where the F_i are the control functions formed with series and parallel combinations of pass transistors, the C_i are the control variables that drive the gates of those transistors, and the P_i are the pass variables.

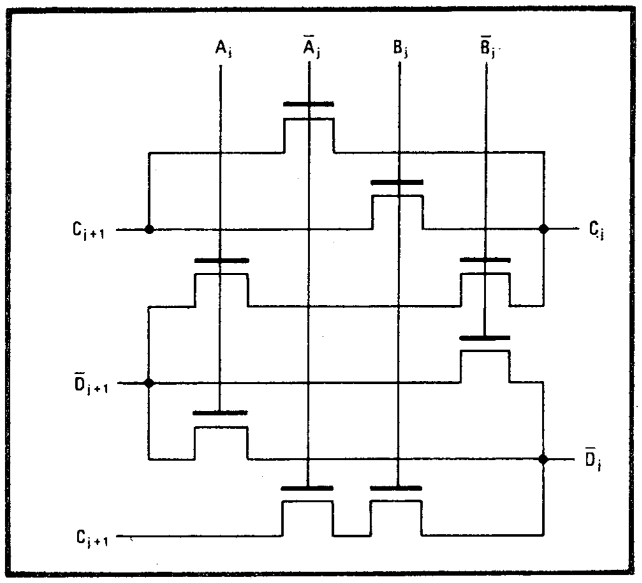
The F_i are a complete and disjoint set: their logical sum is 1, and the logical product of any two is 0. These conditions guarantee that every state of the control variables selects one and only one pass variable. They are easy to meet, for on a Karnaugh map of the C_i the F_i are nonintersecting loops that cover the whole map. Of course, when a high-impedance output state is desired, the corresponding control function can deliberately be left out of the network.

As the above equation shows, no pass variable enters any control function, so an equation does not of necessity allow more than one pass variable. Several pass variables, however, usually produce a denser and faster network. A simple algorithm can be used to calculate the maximum number of pass variables for a Boolean equation written in reduced minterm form.

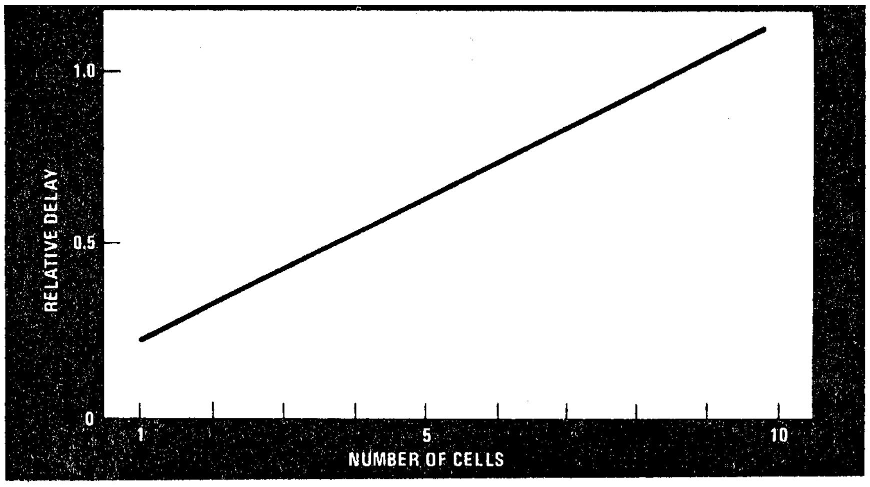
Simple logic

For simple logic functions, pass networks do significantly improve on the traditional implementation, but performance requirements limit their size and usefulness. A signal passing to a network's output travels through several device channels, each with on-resistance and capacitances to gate and substrate. Delay through the network therefore increases with the square of the number of pass transistors. Conventional logic gates merely add delays in turn. The resistance-capacitor time constant associated with the channel of a single pass transistor is typically about 0.2 nanosecond. The delay of a conventional logic stage is about 2 ns.

Conventional NAND or NOR gates can implement any function in three levels of logic. Pass networks, however, may require conventional buffer stages at the output to restore logic levels and drive further stages. For a fair comparison, one logic-gate delay should therefore be added to the pass-network delay.



7. Iteration. One cell of the comparator takes just eight pass transistors. The layout occupies an area of 3,024 square micrometers, only a third of the area of a conventional n-channel MOS implementation, and the cell dissipates almost no standby power.



8. Delay comparison. The propagation delay of a pass-transistor implementation of the magnitude comparator increases with the number of stages. For more than eight stages, it turns out that conventional buffers are required with pass gates.

A signal can pass through five pass transistors and a conventional buffer stage in a time about equal to the delay involved in passing through three conventional logic stages. Those five transistors correspond to five control variables, which can pass at least one and possibly several pass variables. Functions of six or fewer variables are thus generated more quickly in pass networks than in conventional logic; more complex functions are not always suited to them.

Like conventional NAND logic, programmable logic arrays incur only three logic-stage delays, however many variables may team up in the function that is implemented. Unlike pass networks, which basically are wired-OR functions, both PLAS and conventional NAND logic independently form each minterm of a function, permitting a single minterm to be used in more than one output function. Several are often derived from one set of input variables, so cutting the number of transistors in PLAS and conventional NAND logic often offsets the additional area both need for ground connections and load devices and the larger number of transistors per function.

Pass-transistor networks are not the best solution for all problems in combinatorial logic; their unique features suit particular applications—fairly simple random-logic functions, for instance. They also implement a disabled output's high-impedance state naturally and thus make great sense for bus drivers and multiplexers.

Iterative arrays

Pass-transistor networks are usually the best choice for another important class of logic functions: iterative arrays. Iterative logic circuits, often used in magnitude comparators and adders, form a series of intermediate results along the way to the final output. Serial processing is needed in all logic implementations, so pass-transistor designs can always have lower delays than conventional ones.

Consider the iterative magnitude comparator in Fig. 5. A 1-bit cell in it compares the most significant bit of A and the most significant bit of B. Intermediate results C and D are then passed to the next-most significant cell, where they are used with the next bits of A and B to complete the comparison. The process continues until the final result is available from the least significant cell.

For cell j, the output C_j is 1 if A_j is greater than B_j and D_{j+1} is 0, or if C_{j+1} is 1. Output D_j is 1 if D_{j+1} is 1, or if A_j is less than B_j and C_{j+1} is 0. These conditions make the final output, C, high if A is greater than B. D is high if B is greater than A. If both C and D are low, A and B are equal.

Truth tables are constructed for C_j and D_j (with the methods described earlier) and the pass functions are entered in Karnaugh maps. The map for C_j (Fig. 6) shows how the pass variable $\overline{C}_{j+1}$ is looped under the control function $\overline{A}_j+B_{jj}$, while $\overline{D}_{j+1}$ is looped under the function $A_j\overline{B}_j$. The map for D_j is completed in a similar manner.

The resulting pass-network implementation of one cell of the magnitude comparator (Fig. 7) occupies 3,024 square micrometers—only 34% of the 8,840 mm^2 conventional logic designs need. Simulations of the two versions using parameters extracted from the layout are employed to compare circuit delays.

As mentioned previously, pass-transistor networks form RC delay lines, and signals on them obey a diffusion equation. The delay takes the form $An^2 + B$, where n is the number of cells in the comparator and A and B are constants. Delays through the conventional logic network take the form Cn. Constant C is larger than A, in part because pass networks cut capacitance by cutting the number of transistors needed to implement a function. This reduction increases the circuit's regularity, cuts the amount of wiring and gate capacitance, and completely eliminates depletion-load devices and their gate capacitance.

Figure 8 compares the delay of a pass network with that of a conventional circuit by plotting them as functions of the number of bits in the comparator. For 8 or fewer bits, pass networks are faster than conventional ones. For more bits, they can still be faster if conventional buffer stages are inserted when the delay from an additional pass-gate stage would exceed the delay through a buffer followed by the pass stage.

With 8 or fewer bits, the pass-transistor network's power consumption is negligible. For more than 8 bits, the conventional buffers added to the circuit contain two power-dissipating nodes. The conventional logic design has two power-consuming nodes in each cell, so its total power consumption is eight times that of the pass network. □