Formal Design Procedures for Pass Transistor Switching Circuits

DAMU RADHAKRISHNAN, MEMBER, IEEE, STERLING R. WHITAKER, MEMBER, IEEE, AND GARY K. MAKI, MEMBER, IEEE

Abstract—**One goal in modern LSI and VLSI design is to increase the number of digital functions per unit area. Circuit density and power dissipation are major determining factors in limiting circuit complexity. Pass logic, which is composed solely of pass transistors, has attractive features of having high density, high performance, and low power dissipation.**

Most pass transistor logic currently being realized by practicing logic designers involves intuition, experience, and perseverance. The only formal procedures that potentially could apply are those which have been developed for relay logic. The formal procedures advanced in this paper are quite different and produce circuits that require fewer transistors.

Two formal design techniques are presented to realize pass logic networks in NMOS and CMOS technologies. Cells drawn in CMOS and NMOS are compared.

I. Introduction

WITH THE ADVANCEMENTS in integrated circuit technology, especially in LSI and VLSI, increasingly more complex functions are achievable on a single IC chip at a decreasing cost per function. The maximum number of circuit functions per unit area is determined either by the power dissipation density or the area occupied by the transistors, communication paths, and passive devices, if any. There is strong emphasis on increasing circuit density. Switching speed is another important consideration. In MOS circuits, the gate load is due to the capacitance associated with the gate of the succeeding stage, the capacitance of the signal path connecting the two elements, and the nodal junctions. This capacitive load times the equivalent output resistance of the driving stage forms the delay time constant. The delay of a given stage can be decreased by increasing the area of the first element, thus decreasing its output resistance but this in turn increases the capacitance seen by the previous stage. The goal of current LSI design is to simultaneously reduce the total delay and to minimize the required area through the use of the smallest possible circuits with minimum communication paths [1].

A solution to meeting the above objectives is to use pass transistor logic [2], [3]. The average dc power is only due to the switching power consumed by the driving circuits in charging and discharging the pass transistor control gates and driving the pass network inputs. Moreover, pass transistor realizations using minimum size transistors, result in area savings and higher operating speed when compared with the corresponding gate logic realizations. The formal pass transistor design techniques have been successfully used with several NMOS and CMOS LSI circuits to reduce area, increase speed, and reduce power consumption. These circuits include, for example, Boolean functions, magnitude comparators, full adder cells, next state logic for state machines, control logic for counters, and digital multiplier circuits in both NMOS and CMOS designs.

Manuscript received August 22, 1984; revised December 26, 1984.

D. Radhakrishnan is with the Department of Electrical Engineering, Old Dominion University, Norfolk, VA 23508.

S. Whitaker, is with Gould AMI, 2300 Buckshin Rd., Pocatello, ID 83501.

G. Maki is with the Department of Electrical Engineering, University of Idaho, Moscow, ID 83843.

II. Pass Networks

The basic element of pass networks is the MOS transistor. The gate of this transistor is driven by a control signal. This signal may be high or low depending on the manufacturing process—NMOS or PMOS. The CMOS process uses both states of the control signal (high for NMOS and low for PMOS transistor). A simple NMOS pass transistor gate and its truth table are shown in Fig. 1. When the control input is held high, the input is passed to the output, else the output is allowed to float. Since the MOS transistor is a bidirectional element, the input and output leads can be arbitrarily selected.

III. Notations and Definitions

A pass network is an interconnection of a set of pass transistors to achieve a particular switching function. These networks resemble earlier contact networks [4], [5]. Unlike contact networks which pass a "1" to the output, pass networks pass any value from the set $[0, 1, Xi, Xi', Z]$ where Xi is any input variable and Z is the high impedance state. The signals which drive the gates of the MOS transistors are called control signals and the associated variables are called control variables. Similarly, the signals which feed the inputs of the MOS transistors are called pass signals since these will be passed to the output, and the associated variables are called pass variables.

0018-9200/85/0400-0531$01.00

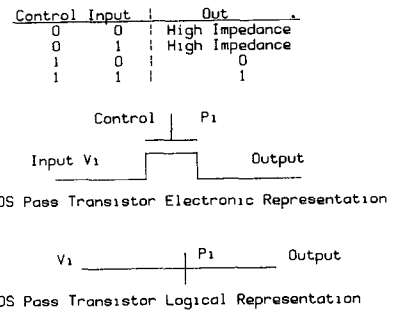


Fig. 1. Pass transistor.

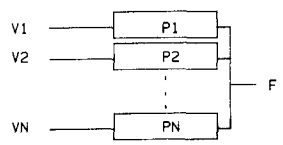


Fig. 2 Pass network representation.

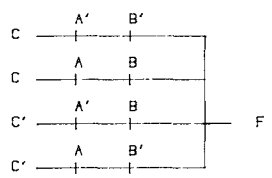


Fig. 3. Pass network example.

IV. NMOS Pass Networks

The following notation will be used in the initial presentation where the networks are realized with only NMOS transistors. A generalized pass network can be depicted as shown in Fig. 2. Each Pi is composed of a series of transistors that, when enabled, will present Vi to the output F. Pi can thus be represented as a product term, where each literal of Pi represents a pass transistor. If $Xj(Xj')$ is a literal in Pi, then there is a transistor with $Xj(Xj')$ controlling the gate. When all the literals of Pi are $=1$, pass variable Vi is presented to the output. The output pass function F can be expressed as $F=P1(V1)+P2(V2)+\cdots+Pn(Vn)$. When $Pi=1$, the transistors that compose Pi are enabled and the pass variable Vi is presented to the output F. Fig. 3 is an example of the sum generator for a full adder, where the pass function is $F=A'B'(C)+AB(C)+A'B(C')+AB'(C')$.

V. Design of a Pass Network Using a Karnaugh Map

Before the procedure for the minimization of a pass function using a K-map is discussed in detail, it is in order to analyze the basic principles behind the conventional map minimization method. Considering the sum of products form, it is observed that when a minterm mi is true, 1 is passed to the output. This can be expressed as $mi(1)$ in the notation introduced above. Hence if $P1, P2,\cdots, Pn$ represent the prime implicants of a minimized switching function, then an equivalent representation for it in the new notation is: $F1=P1(1)+P2(1)+\cdots+Pi(1)+\cdots+Pn(1)$.

When all prime implicants $P1, P2,\cdots, PN$ become false, the conventional switching function produces a 0 at the output since it has only 2 states. However, a pass network produces a high impedance state at the output when all Pi terms are false. To avoid a high impedance output under this condition, a 0 must be passed whenever a 0 is required at the output. This suggests that like 1's, all the 0 entries on the map must also be grouped together, and a 0 passed to the output with each of these groups expressed as a product term. If Yi represents a minimized product term for a group of 0's, then this can be denoted as $Yi(0)$. Hence if $Y1, Y2,\cdots, Ym$ represent the product terms for different 0 groups on a K-map, then the pass function to produce 0 is: $F0=Y1(0)+Y2(0)+\cdots+Ym(0)$. Thus, the pass function for the whole network becomes: $F=P1(1)+P2(1)+\cdots+Pn(1)+Y1(0)+Y2(0)+\cdots+Ym(0)$.

This can be directly implemented as a pass network. Attempting to implement the above equation with pass logic will produce a functionally correct output, however, the realization would suffer from requiring more transistors than necessary. A pass network need not be limited to passing only 0's or 1's. In general, a pass network can produce an output in the set $[0,1, Xi, Xi', Z]$ where Xi is any input variable, and Z is the high impedance state. The nature of this set suggests that logic design with pass transistors is more complex. This notion is confirmed by the limited complexity of pass networks designed without formal techniques. Following is the development of a formal procedure to realize a circuit with pass transistors.

Definition 1: If Vi denotes the pass variable and Pi denotes the product term, then $Pi(Vi)$ is called the pass implicant, where Vi is in the set $[0,1, Xi, Xi']$ with Xi being any input variable.

The pass implicant is similar to the traditional implicant but, instead of representing only the 1's of the function, it represents either the 1's, 0's, or a combination of both 1's and 0's. The number of pass transistors in the circuit realization is equal to the number of literals that appear in Pi. A pass function is composed of a sum of pass implicants and can be expressed as F is the summation $i=1$ to n of $Pi(Vi)$.

Procedure A specifies the method to find a pass implicant which consists of the following:

1) 2**i cells in the K-map, each cell in the implicant has i adjacent cells in the implicant. The product term defining the implicant is identical to the traditional implicants,

2) The pass variable Vi in the implicant is equal to one of the elements $[0,1, Xi, Xi']$. Vi must be equal to one of these elements throughout the implicant.

Example 1 illustrates these concepts. A three variable Karnaugh map is shown in Fig. 4(a). Cells 0 and 1 denote a pass implicant with the pass variable 0; the pass implicant is $A'B'(0)$. Cells 2 and 3 constitute a second pass implicant $A'B(C)$. Similarly, cells 4 and 5 form pass implicant $AB'(C)$

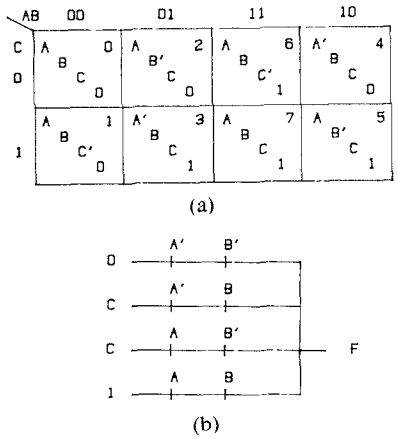


Fig 4. Pass prime implicant example.

and cells 6 and 7 form $AB(1)$. The complete pass network could be expressed as $F = A'B'(0) + A'B(C) + AB'(C) + AB(1)$, which forms the carry generator for a full adder. The implementation is shown in Fig. 4(b).

Definition 2: A pass prime implicant is a pass implicant which subsumes no other pass implicant with fewer number of literals which implies the function [5]. The pass prime implicants are formed with the following additions to the usual K-map rules:

1) Each and every cell in the map be covered at least once. This condition prevents the output from entering the high impedance state. In some cases, it may be desired to allow the output to float for certain input states [*Ii*] which are not covered by any pass implicant.

2) Pass implicants and pass variables are identified in accordance with Procedure *A*.

In the previous example (Fig. 4), all the implicants used to form the pass function are prime implicants.

A conflict condition occurs whenever two pass implicants $Pi(Vi)$ and $Pj(Vj)$ are both enabled to present Vi and Vj to the output and $Vi = Vj'$. The pass network design presented here guarantees that no conflict can occur in completely specified functions. Consider the pass function: $F = P1(V1) + \cdots + Pi(Vi) + Pj(Vj) + \cdots + Pn(Vn)$. If $Pi = Pj = 1$, then Vi and Vj are passed to the output. For a completely specified function, a given input state that permits both Pi and Pj to be 1 would be presenting a single entry on the K-map to the output hence, Vi and Vj would be in the same logic state (0 or 1). Therefore, for a completely specified function, it is impossible for a conflict to occur. For an incompletely specified function, it is possible for an input condition to exist that allows both Pi and Pj to be 1 with different values for Vi and Vj because $Pi(Vi)$ and $Pj(Vj)$ may specify a don't care state differently. This conflict can produce a short circuit through the pass network. The assumption of using minimum-sized transistors eliminates the possibility of a "dead short" and guarantees that the short will be of the order of several kilo ohms. However, a short condition is undesirable, and a judicious engineer will guarantee that two pass implicants do not specify a don't care state such that a conflict occurs.

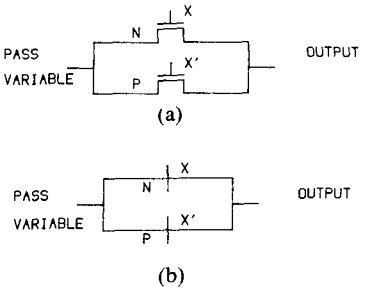


Fig. 5. (a) CMOS pass gate electronic description. (b) CMOS pass gate logic representation.

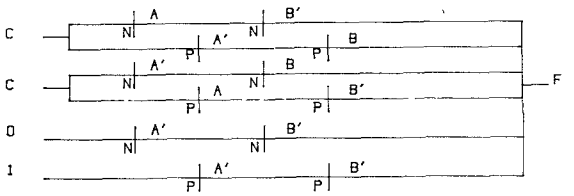


Fig 6. CMOS implementation for pass network of Fig. 4.

VI. CMOS Pass Networks

CMOS pass gates are normally composed of an NMOS and PMOS transistor, as shown in Fig. 5(a) and 5(b). When the gate and drain of an NMOS transistor are driven high, the source only rises to the gate voltage minus a body effect modified threshold. An NMOS transistor therefore passes a good 0, but a poor 1. A PMOS transistor similarly passes a good 1 and a poor 0. A CMOS gate will pass both a good 0 and a good 1, since both devices are present. A CMOS pass network is designed using a K-map using the following procedures:

1) The K-map is read in the same manner as defined above to obtain each $Pi(Vi)$.

2) The NMOS devices form each Pi in the same manner as defined earlier with the control variables being the literal of Pi.

3) The PMOS devices, forming each Pi, are connected in series to form a parallel path to the NMOS devices. The control variables are the complement of the literals of Pi.

4) The pass variable of Vi is the same for both NMOS and PMOS transistors of Pi.

5) Whenever the pass variable Vi is a 0, only NMOS transistors are needed.

6) Whenever the pass variable Vi is a 1, only PMOS transistors are needed.

The pass function of Fig. 4 is $F = A'B'(0) + A'B(C) + AB'(C) + AB(1)$. The CMOS pass network for this function is shown in Fig. 6.

VII. Tabular Approach for the Design of a Pass Network

An algorithmic procedure similar to the Quine–McCluskey tabular approach is developed here for the design of pass networks. In the conventional approach, 1's

Fig. 7. Pass implicant record.

(0's) alone are used to arrive at the prime implicants (implicates) of the function. But for the generation of a minimal pass network from a conventional switching function, both the 1's and 0's of the function are used to arrive at the pass prime implicants. This is necessary because a pass implicant represents both 1's and 0's of the function.

The data structure used for representing a pass network is a linear list of pass implicant records. A record has three fields—a base field, a difference field, and a pass field. The basic structure of a record is defined as follows: (see Fig. 7)

1) Base Field: This is an integer field that represents the cells in the truth table. The base field functions identically to the base field of the traditional tabular method and can be represented in any desired base in the presentation; here base 10 is used. When the pass implicant covers more than one cell, the lowest value will be listed. For example, when two adjacent cells *a'bc'* and *a'bc* represented by their decimal equivalents 2 and 3 combine together to form a pass implicant, 2 becomes the base field entry.

2) Difference Field: This is also an integer field, consisting of one or more integer entries separated by commas. The difference field has an identical counterpart in the traditional tabular method. The entries in this field represent the difference between the cell representations of the terms which are combined together to form the pass implicant. The difference field for the implicant formed by the combination of two terms whose decimal equivalents are 2 and 3 is (1), whereas it is (1,4) for an implicant formed of 2(1) and 6(1). The ordering of the entries in a Difference field bears no significance thus (1,4) and (4,1) are equivalent.

3) Pass Field: This is an alphanumeric field which represents the pass variable that is to be passed by the pass implicant. The pass field can contain 0, 1, *Xi*, or *Xi'*.

LEMMA: Two pass implicant records cannot combine unless:

a) the base fields differ in only one binary bit, and
b) the difference fields agree.

Proof: The proof of this lemma is identical to that in the literature for the traditional tabular method.

Theorem 1: Consider the cells *Ci* and *Cj* of the function $f(Xn,\cdots, X1, X0)$ with outputs *F* assigned to be *Fi* and *Fj*, respectively. Let *Ci* and *Cj* be adjacent with variable $Xk=0$ in *Ci* and $Xk=1$ in *Cj*. The pass implicant records are:

a) If $Fi=0$ and $Fj=0$ then *Ci* (2**k) 0.
b) If $Fi=1$ and $Fj=1$ then *Ci* (2**k) 1.
c) If $Fi=0$ and $Fj=1$ then *Ci* (2**k) *Xk*.
d) If $Fi=1$ and $Fj=0$ then *Ci* (2**k) *Xk'*.

Proof: For each of the above, the base field is clearly *Ci*, the difference field is 2**k. Both of these fields are arrived at using the same techniques as the traditional tabular method. The pass field entry depends on the states of *Fi* and *Fj*.

a) If *Fi* and $Fj=0$, then the value to be passed is 0.
b) If *Fi* and $Fj=1$, then the value to be passed is 1.
c) If $Fi=0$ and $Fj=1$, then the variable to be passed is *Xk* since all other variables are constant and the output is *Xk*.
d) If $Fi=1$ and $Fj=0$, then the variable to be passed is *Xk'* since the output is *Xk'*.

Corollary: Two pass implicant records can be combined if the conditions of the lemma are met and the pass fields contain:

a) a constant (with the new pass variable defined by Theorem 1); or
b) an identical pass variable.

The algorithm is divided into two parts: Part *a* determines all the pass prime implicants for the function, and part *b* selects a minimal cover.

Part a

1) List all the pass implicant records by tabulating the decimal numbers 0 to $(2^n)-1$ for *n* variable function and their corresponding output function as the pass variable in an ascending order of their index (number of 1's in their binary representation). The terms listed in this manner are in full agreement with the data structure mentioned above except that the difference fields are left blank. The don't care terms are listed twice, once with an output value of 1 and then with an output value of 0.

2) Separate the records of equal index by drawing lines.

3) Let $i=0$.

4) Compare each record of index *i* with each record of index $i+1$. Two records combine only if the conditions of the corollary are satisfied. For each pair of records that can combine, form a new record where the Difference field has an additional entry included in it equal to the difference between the base field entries, and place it in the section of index *i* of a new list unless it is already present. In either event, place a check mark next to the two records that combined. After all pair of records with indexes *i* and $i+1$ from both lists have been compared in this manner, a line is drawn under the last record in the new list. (A checked record does not disqualify it from future comparisons).

5) Increment *i* by 1 and repeat step 4 until all the records have been compared. The new list has one more entry in its difference field than the corresponding ones in the earlier list.

6) Steps 3, 4, and 5 are repeated on the new list to form another list.

7) The process terminates when no new list can be formed.

8) All the records without check marks form the pass prime implicants.

Part b

The next step in the minimization procedure is to select a minimal set of pass prime implicants using a prime implicant table. This table is formed in a manner exactly similar to the one used in conventional switching circuits,

TABLE I

TRUTH TABLE (BINARY) X2 X1 X0	(DEC) X	OUTPUT f	BASE (DIFFERENCE) PASS (LIST 1)	(LIST 2)
0 0 0	0	0	0(1)0 0(2)X1 0(4)0	0(1,2)X1
0 0 1	1	0	1(2)X1 1(4)X2	
0 1 0	2	1	2(1)1 2(4)X2	
1 0 0	4	0	4(1)X0 4(2)0	4(1,2)X0
0 1 1	3	1	3(4)1	
1 0 1	5	1	5(2)1	
1 1 0	6	0	6(1)X0	
1 1 1	7	1		

TABLE II

	m0	m1	m2	m3	m4	m5	m6	m7
0(1,2)X1	x	x	x	x				
4(1,2)X0					x	x	x	x
0(4)0	x				x			
1(4)X2		x				x		
2(4)X2			x				x	
3(4)1				x				x

except that, instead of considering only 1's or 0's of the function, all the 2^n cells (1's and 0's) of the n-variable function are used. Every column must be covered by at least one pass prime implicant. Procedures such as row and column dominance apply in the covering problem of this table.

Once the prime implicants forming a minimal cover for the function are selected, the pass function is formed by ORing all these prime implicants.

Table II illustrates the simplification of a pass network using the above approach. The function is $f(X2, X1, X0) = \epsilon m(2,3,5,7)$. All the eight-pass implicant records of this three-variable function are listed in ascending order of their index as shown in Table I. The prime implicant table is shown in Table II.

The pass prime implicants to cover all the eight terms of the three-variable function are $0(1,2)X1$ and $4(1,2)X0$. These are represented as $X2'(X1)$ and $X2(X0)$, respectively. Hence the pass function for the network is $f(X2, X1, X0) = X2'(X1) + X2(X0)$, which can be implemented with two NMOS transistors.

VIII. Conclusions

This paper represents two different approaches for the design of a pass network. The first one uses a modified Karnaugh map minimization procedure which can be an effective tool for the design of networks up to five or six variables. For networks involving more than six variables, an algorithmic procedure is developed by modifying the conventional Quine–McCluskey approach.

The savings in silicon area depends on the transistor count as well as the interconnect structure. Maximum topological regularity for an array of pass transistors can be achieved if the intersection of the set of control variables with the set of pass variables is a null set. This allows the pass variables and the control variables to flow at right angles to each other. This requirement may increase the transistor count in the design, hence there is a tradeoff between topological regularity and transistor count. Higher transistor count also provides a higher load to the driving circuits due to the increased gate capacitance causing another tradeoff consideration.

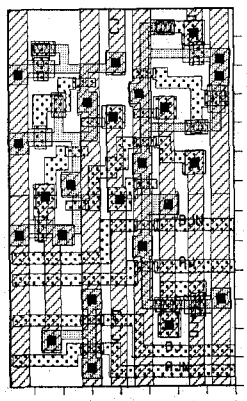


Fig. 8. CMOS cell comparison.

NMOS and CMOS cells for digital magnitude comparison were designed and used to compare the performance of pass and gate logic. The NMOS gate and pass logic cells required $79 \times 130 = 10{,}270$ and $52 \times 43 = 2{,}236$ sq. μm, respectively, as shown in Fig. 8. CMOS gate and pass logic cells were designed with size requirements of $182 \times 237 = 43{,}134$ and $133 \times 217 = 28{,}861$ sq. μm respectively, as shown in Fig. 9. These examples demonstrate considerable savings in silicon area that can be achieved by using pass logic instead of gate logic. The savings amount to a 33-percent reduction in silicon area for CMOS and 78-percent reduction for NMOS. A computer simulation was done using ASPEC to compare their performance. The NMOS versions had a typical delay of 8 ns for the gate logic and 1 ns for the pass logic.

For arrays of small size, the pass network exhibits faster operation when compared to gate networks. However, the delay characteristics were quadratic in the number of cells of pass logic, whereas it was linear for gate logic. Thus longer chains of pass logic cells must be partitioned into smaller groups with buffer stages interposed to overcome this quadratic delay characteristic.

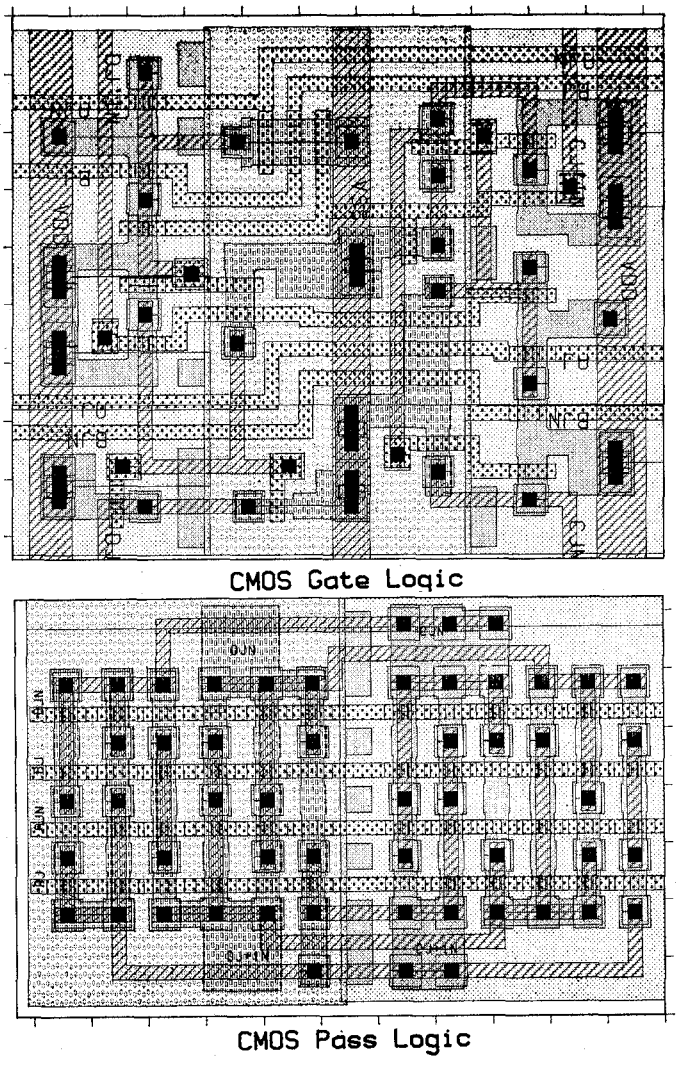
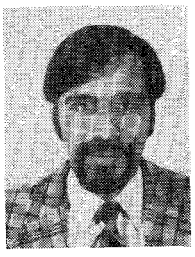


Fig. 9. NMOS cell comparison.

References

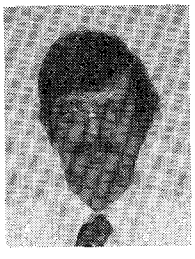
[1] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.
[2] S. R. Whitaker, *Design of Combinational Logic with Pass Transistors*. Moscow, ID: Univ. Idaho, 1982.
[3] D. Radhakrishnan and G. Maki, *Theory and Design of Pass Transistor Switching Circuits*. Moscow, ID: Univ. Idaho, 1983.
[4] E. J. McCluskey, *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill, 1965.
[5] D. D. Givone, *Introduction to Switching Circuit Theory*. New York: McGraw-Hill, 1970.



Damu Radhakrishnan (S'74–M'76) received the B.Sc. degree in electronics and communication engineering from the University of Kerala, India in 1968, the M.Tech. and Ph.D. degrees from the Indian Institute of Technology, Kanpur, India, and the University of Idaho, Moscow, ID, both in electrical engineering in 1975 and 1983, respectively.

He is an Assistant Professor in the Department of Electrical Engineering at Old Dominion University, Norfolk, VA. His research interests include design for testability, VLSI systems, fault tolerant computing, and computer architecture.

Dr. Radhakrishnan is a member of Sigma Xi.



Sterling R. Whitaker (S'76–M'77) received the B.S.E.E. degree at Brigham Young University, Provo, Utah, in 1977 and he received an M.S.E.E. degree through the University of Idaho, Moscow, in 1982.

He started as a Design Engineer working on custom MOS IC's at Gould AMI Semiconductors and he is currently managing a section of Design engineers while pursuing the Ph.D. at the University of Idaho.

Gary K. Maki (S'67–M'69) received the B.S.E.E. degree at Michigan Technological University, Houghton, MI and the M.S.E.E. and Ph.D. degrees in electrical engineering from the University of Missouri, Columbia.

He currently is a professor at the University of Idaho, Moscow. His current research interests include computer architecture and VLSI design, especially custom processors for space applications, which involve NMOS, CMOS, and GaAs.