Chapter 5

Pass Transistor Combinational Logic Circuits

A pass transistor is an nMOS or pMOS transistor used to block or conduct electrical signals in MOS logic circuits. The propagation of the signal through the transistor is controlled by a signal applied to the Gate of the transistor, called control signal and the associated variable is called control variable. Similarly, the signals that feed the inputs of the MOS transistors are called pass signals since these are passed to the output, and the associated variables are called pass variables.

Many combinational logic functions can be efficiently realized using an interconnection of these transistors. The basic elements used in these designs are the three types of pass transistor gates - an nMOS pass transistor, a pMOS pass transistor, and the transmission gate.

nMOS Pass Transistor

In an nMOS transistor logic 1 at the Gate passes the input from Source to Drain and logic 0 opens the Drain from Source. The truth table for an nMOS pass transistor is shown in Figure 1-1. The MOS transistor basically is a bidirectional element, and hence the input-output leads can be arbitrarily selected.

![Truth Table of nMOS Pass Transistor](image)

If X and Y are connected to the Gate and Source of an nMOS transistor respectively, then this is represented as X (Y) and read as “X passing Y”. The logic symbol for the nMOS transistor is shown in Figure 1-2. When logic 0 is passed through an nMOS transistor, the voltage output from the transistor will be very close to 0. This happens because the Gate voltage in this case is more than one threshold voltage \(V_{th}\) above its Source voltage. But on the other hand, when a 1 is passed through the nMOS transistor, the voltage at the Gate has the same value as the input. The output voltage will then be lower by one threshold voltage \(V_{th}\). This implies that an nMOS transistor behaves like a perfect switch when it is used to pass a 0, where as it behaves like an imperfect switch when it is used to pass a 1. Hence a 0 output from an nMOS transistor is labeled as a strong 0 and a 1 output as a weak 1.

![A nMOS Pass Transistor Gate Z = X (Y)](image)
pMOS Pass Transistor

A pMOS transistor and its logic representation are shown in Figure 1-3. It conducts and passes the input to the output when logic 1 is applied to its Gate. To avoid any confusion, the same pass logic notation is used to represent the switching function of both nMOS and pMOS transistors, but the control signals applied to the Gates of the pMOS transistors are inverted during implementation. Hence for a pMOS transistor when \( \bar{X} \) and \( Y \) are connected to the Gate and Source respectively, this is represented as \( X (Y) \). This is shown in Figure 1-3. Now by using arguments similar to the ones used in nMOS transistors, we can conclude that a pMOS transistor passes a strong 1 and a weak 0. Hence to provide the same signal strength at the output the size of the pMOS transistor is usually made larger than that of an nMOS transistor.

![Figure 1-3. A pMOS Pass Transistor Gate Z = X (Y)](image)

CMOS Transmission Gate

A CMOS transmission gate consists of an nMOS and a pMOS transistor connected in parallel as shown in Figure 1-4. This combination is used to pass a strong logic value (both 0 and 1) to the output independent of the state of the input signal value. The Gates of the two transistors are connected to complementary logic levels. Thus it is obvious that the use of CMOS transmission gates in logic circuit designs increases the overhead due to doubling of the transistor count, increased chip area and the power consumption.

![Figure 1-4. A CMOS Transmission Gate Z = X (Y)](image)

Pass Networks

A pass network is an interconnection of a number of pass transistors to achieve a particular switching function. Pass networks resemble earlier contact networks. The basic element of pass networks is the MOS transistor. Unlike contact networks that pass a 1 to the output, pass networks pass any value from the set \( \{0, X, \bar{X}, Z\} \) where \( X_i \) is an input variable and \( Z \) is the high impedance state. Pass networks are classified into three types depending on the type of transistors used in their implementation. They are nMOS pass networks, pMOS pass networks and CMOS pass networks. An nMOS pass network is implemented fully by nMOS transistors, a
pMOS pass network is implemented fully by pMOS transistors, and a CMOS pass network is implemented by a combination of both nMOS and pMOS transistors.

A series connection of a number of nMOS transistors passes the input to the output when all control signals are held high. This is represented by the expression \( X_1 X_2 \ldots X_n (V) \), where \( X_1, X_2, \ldots, X_n \) are the control variables applied to the Gates of the nMOS transistors and \( V \) is the pass variable. The logic function realized by a pass network is referred to as a pass function. A generalized pass network can hence be represented by a pass function \( f \) as: 
\[
 f = \sum_{i=1}^{n} P_i(V_i) 
\]
with \( P_i \) representing product of the variables controlling the transistors. The variable \( V_i \) is in the set \( \{0,1,X_i,X_i'\} \), \( X_i \) being an input variable. A pass network corresponding to the pass function \( f \) is shown in Figure 1-5.

If \( X_j/X_j' \) is a literal in \( P_i \) then there is a transistor in \( P_i \) with \( X_j/X_j' \) controlling its Gate. When all the literals of \( P_i \) are equal to 1, the transistors that compose \( P_i \) are enabled and the pass variable \( V_i \) is presented to the output \( f \).

![Figure 1-5. Pass Network Representation](image)

Figure 1-6 is an example pass network where the pass function is given by:
\[
 f = A\overline{B}'(D') + A\overline{B}'(C) + B\overline{C}'(0) + BD(0) + BCD'(A') 
\]

![Figure 1-6. An nMOS Pass Network for](image)
Pass Logic Multiplexer

A multiplexer (MUX) is a very powerful basic building block in logic networks. A MUX implementation of logic functions often results in compact circuits. A $2^n$:1 MUX has $n$ selector lines and $2^n$ input lines. For each and every state of the selector variables one of the input lines will be connected to the output. Hence for a 2:1 MUX with selector variable $X$, the two input states are $X=0$ and $X=1$. This is shown in Figure 1-7. For $X=0$, $Y$ is connected to the output, and for $X=1$, $Z$ is connected to the output. The logic expression for the MUX function is therefore given by: $f = X(Y) + X(Z)$. In pass logic this can be written as: $f = \bar{X}(Y) + X(Z)$. This function is referred to as MUX function in the remainder of this book. The pass logic MUX can be implemented using two transmission gates as shown in Figure 1-8.

Pass Network Theorems

The following Identities and Theorems can be used in the simplification of pass functions.

Pass Logic Identities

1. $1(Y) = (Y)$
2. $X(X) = X(1)$
3. $X(X') = X(0)$

Pass Logic Theorems

1. $X(Y) + (Z) = (Z)$
Pass Network Design

The design of pass networks has been an art that in the past was left to the cleverness of the logic designer. With the increased interest in this area a number of formal techniques are available these days for the analysis and synthesis of these networks. The simplest and easiest one is of course based on a truth table. Even though the design using a truth table is very straightforward the minimization of the number of transistors is usually a tedious job. Hence this is not a preferred approach even for very small circuits. A fast and efficient approach for the design of circuits with a few variables is to use a Karnaugh map. Again as the number of variables increases Karnaugh map minimization becomes difficult. Quine-McCluskey type algorithmic approaches are available for circuits with more variables.

All the above-mentioned techniques produce the so-called two-level circuits. Many times these two-level pass functions can be factored to reduce the number of literals. This factorization permits the sharing of transistors among different pass implicants, thereby minimizing the number of transistors in their implementation.

When more than one output function is involved, factorization techniques are not straightforward. The synthesis of such circuits is more involved.

1. Truth Table Approach

In this approach, first a truth table is formed. Then this is extended by adding an extra column to it to include the P-terms for each row of the truth table. If \( X_1, X_2, \ldots, X_n \) are the input variables and \( Z \) is the output variable, then the pass variables in each row of the truth table are \( X_i \) if \( Z = X_i \) and \( \overline{X_i} \) if \( Z = \overline{X_i} \). All these pass variables corresponding to each row of the truth table are OR’ed together to form the P-term. This is illustrated in Table 1 for an AND gate.

Table 1. Truth Table of an AND gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Z</th>
<th>P-Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A + B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A + B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A + B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A + B</td>
</tr>
</tbody>
</table>

1. \( X(1) + X'(0) = (X) \)
2. \( X'(Y) + X(Y) = (Y) \)
3. \( X(Y + Z) + X'(Y' + Z) = (Z) \)
4. \( X(Y + Z) + X'(-) = (Y) = (Z) \)
5. \( XY(Z) + XW(Y) = X(Y(Z) + W(Y)) \)
6. \( X(Y) + X'(Y') = Y(X) + Y'(X') \)
7. \( X(0) + X'(0) = Y(0) + Y'(X) \)
8. \( X(1) + X'(Y) = Y(I) + Y'(X) \)
The pass implicants for each row of the truth table consists of a product term and a pass variable. The product term is formed in a manner similar to the traditional Boolean method, and the pass variable is selected by choosing either the actual output logic value or any one literal from the P-term. For Table 1, the product term for the first row is \( \overline{A} \overline{B} \), and the pass variable can be chosen as 0, A or B. Hence the pass logic expression for the first row could be one of \( \overline{A} \overline{B}(0) + \overline{A}B(A) \) or \( A \overline{B}(B) \). Similarly, the pass logic expressions for other rows can be found. The complete pass function can now be written by ORing the expressions for each row as:

\[
f = \overline{A} \overline{B}(0) + \overline{A}B(A) + A \overline{B}(B) + A B(A)
\]

From the above discussion it may be noted that the pass functions obtained using the above technique is not unique. For each row in Table 1, there are three choices thereby giving a total of 12 different pass functions for f. The pass function can also be derived easily from its switching function if it is expressed in the sum of products form. To do this, first the function is converted to the standard sum of products form. When the function is expressed in this form, it implies that when any of the minterms forming the function is true, 1 is passed to the output. This can be expressed simply as \( m_i(l) \) in pass logic notation. Hence if \( m_1, m_2, \ldots, m_n \) represent the different minterms of a switching function, then an equivalent representation for it in the pass function notation is:

\[
\sum m_i(l) = m_1(l) + m_2(l) + \ldots + m_n(l)
\]

Now the question arises as to what happens when all the minterms \( m_1, m_2, \ldots, m_n \) become false? The answer is simple. A conventional switching function passes a 0 to the output since it has only two states, whereas the pass network defined above produces a high impedance state at the output. To avoid a high impedance state at the 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 minterms corresponding to the 0’s of the function must also be added to the pass function and a 0 passed to the output with each of these minterms. If \( M_1, M_2, \ldots, M_k \) represents the minterms corresponding to the 0’s of the function, then the pass function to produce 0 output is:

\[
f = m_1(l) + m_2(l) + \ldots + m_n(l) + M_1(0) + M_2(0) + \ldots + M_k(0)
\]

This function can be minimized using the identities and theorems defined earlier and can then be realized as a pass network. This is illustrated by the following example.

Example 1: Consider the XOR (Exclusive-OR) function of two variables, \( Z = A \overline{B} + A B \). The minterms associated with the 1’s of the function are \( A \overline{B} \) and \( A B \). Similarly, the minterms associated with the 0’s of the function are \( \overline{A} \overline{B} \) and \( \overline{A} B \). Hence the pass function becomes:

\[
Z = \overline{A} \overline{B}(1) + \overline{A}B(0) + \overline{A}B(0) + \overline{A}B(B) + A \overline{B}(B) + A B(0)
\]

The above function can now be simplified further by factoring the common variables, thus

\[
Z = \overline{A}(B(1) + B(0)) + A(B(1) + B(0))
\]

Using pass network Theorems this can be rewritten as:

\[
Z = \overline{A}(B) + A(B)
\]
This function can be implemented using two pass transistors.

2. Karnaugh Map Approach

Before the procedure for the minimization of a pass function using Karnaugh 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 \( m \) is true, 1 is passed to the output. This can be expressed as \( m \) (1) in the notation above. Hence if \( P_1, P_2, \ldots, P_n \) represent the prime implicants of a minimized function, then an equivalent representation for it in the new notation is: \( f = P_1(1) + P_2(1) + \ldots + P_n(1) \)

Now to avoid a high impedance state at the output when all \( P_i \) terms are false, a 0 must be passed whenever a 0 is required at the output. This suggests that like 1's, all 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 \( Y_i \) represents a minimized product term for a group of 0's, then this can be denoted as \( Y_i(0) \). Hence if \( Y_1, Y_2, \ldots, Y_m \) represent the product terms for different 0 groups on a Karnaugh map, then the pass function to produce 0 output is:

\[
Y_1(0) + Y_2(0) + \ldots + Y_m(0)
\]

Thus the pass function for the whole network becomes:

\[
f = P_1(1) + P_2(1) + \ldots + P_n(1) + Y_1(0) + Y_2(0) + \ldots + Y_m(0)
\]

This can be directly implemented as a pass network. Attempting to implement the above function with pass logic produces a functionally correct output, however the realization would suffer from requiring many more transistors than necessary. A pass network need not be limited to passing only constants 0's or 1's. In general a pass network can produce an output in the set \( \{0,1,\overline{X_i},\overline{X_i},Z\} \) where \( X_i \) is a 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 derive a pass function using a Karnaugh map.

The Karnaugh map minimization procedure consists of three steps.

1. Mapping the function onto the Karnaugh map, which is done in the normal manner.
2. Combining the entries together to form pass prime implicants.
3. Reading the simplified function from the map.

Definition: If \( V_i \) denotes the pass variable and \( P_i \) denotes the product term, then \( P_i(V_i) \) is called the pass implicant: \( V_i \) is in the set \( \{0,1,\overline{X_i},\overline{X_i}\} \).

The following procedure specifies the method to find a pass implicant:

Procedure A:

1. A pass implicant \( P_i(V_i) \) consists of \( 2^j \) adjacent cells in the Karnaugh map and the product term \( P_i \) is formed in an identical manner to traditional implicants.
2. The pass variable $V_i$ in the implicant must be equal to one of the elements from the set $\{0,1,X_i,\bar{X}_i\}$ where $X_i$ is one of the variables in the k-map.

The pass implicant is similar to the traditional implicant but instead of representing only the 1’s of the function, it represents the 1’s, 0’s or a combination of both 1’s and 0’s. The number of transistors in the circuit realization is equal to the number of literals that appear in $P_i$.

**Example 1:** Consider the 4 variable Karnaugh map shown in Figure 1-9. The different pass implicants and their corresponding pass variables are listed below.

1. Unlike the traditional k-map, any two adjacent cells can always be grouped together to form a pass implicant. A few of the pass implicants formed by combining two cells are:
   
   a. cells 0 and 1 - pass variable 0, and pass implicant $\bar{ABC}(0)$
   
   b. cells 0 and 4 - pass variable 0, and pass implicant $\bar{ACD}(0)$
   
   c. cells 0 and 2 - pass variable $C$, and pass implicant $\bar{ABD}(C)$
   
   d. cells 6 and 7 - pass variable $D$, and pass implicant $\bar{ABC}(D)$

2. Pass implicants with four adjacent cells are:
   
   a. cells 0,1,2,3 - pass variable $C$, and pass implicant $\bar{ABC}(C)$
   
   b. cells 0,1,4,5 - pass variable 0, and pass implicant $\bar{AC}(0)$
   
   c. cells 0,2,4,6 - pass variable $C$, and pass implicant $\bar{AD}(C)$
   
   d. cells 1,5,9,13 - pass variable 0, and pass implicant $\bar{CD}(0)$
   
   e. cells 4,5,12,13 - pass variable 0, and pass implicant $\bar{B}(C)$
   
   f. cells 5,7,13,15 - pass variable 0, and pass implicant $BD(0)$
   
   g. cells 8,10,12,14 - pass variable $\bar{B}$, and pass implicant $\bar{AD}(\bar{B})$

![Figure 1-9. Pass Implicant Example](image-url)
h. cells 9,11,13,15 - pass variable 0, and pass implicant AD (0)
i. cells 12,13,14,15 - pass variable 0, and pass implicant AB (0)
j. cells 8,9,10,11 - pass variable $D - D$, and pass implicant $A\bar{B}(D)$

3. There are no pass implicants with 8 adjacent cells.

**Definition**: A pass prime implicant is a pass implicant, which subsumes no other pass implicant with fewer numbers of literals, which implies the function.

The pass prime implicants are formed with the following additions to the usual Karnaugh map rules:

1. Each and every entry in the map must be covered at least once, and
2. Pass implicants and pass variables are identified in accordance with Procedure A.

Condition 1 prevents the output from entering the high impedance state for all input conditions. In some cases it may be desired to allow the output to float for certain input states $I_i$. In this case the cells of $I_i$ are not covered by any pass implicant.

In the previous example (Figure 1-9), all the pass implicants formed using 4 adjacent cells is pass prime implicants. A minimal pass function can now be obtained by selecting a minimal set of pass prime implicants to cover the whole map. This gives the function as $f = \overline{A}\overline{B}(C) + \overline{A}\overline{D}(C) + BD(0) + AB(0) + A\bar{B}(D)$

One of the problems faced by the above Karnaugh map procedure is that it does not guarantee an optimal solution for a pass network in terms of the transistor count used in its implementation. This is due to the simple fact that the number of transistors will be smaller for a tree network configuration than for any other structural configuration. The network formed from the k-map minimization procedure mentioned above need not necessarily be a tree network. Hence the function obtained using the k-map can be factored for common variables thus reducing the total transistor count in the implementation. This is illustrated in the following example.

**Example 2**: The Karnaugh map for a 4 variable function is shown in Figure 1-10. The nMOS pass function is given by: $f = AD(1) + CD(A) + D(B) + \overline{A}\overline{C}(B)$. Factoring $f$ yields: $f = D(A(1) + C(A)) + D(B) + A\overline{C}(B)$

Its implementation as an nMOS pass network is also shown in Figure 1-10.

![Figure 1-10. A 4 Variable Karnaugh map and its Minimal nMOS Pass Network](image-url)
A second problem faced in the k-map design is when the function is incompletely specified. In such cases it may happen that complementary logic values (both 1 and 0) may be passed simultaneously to the output by two different paths. This introduces a conflict at the output of the network. A conflict occurs whenever two pass implicants $P_i(V_i)$ and $P_j(V_j)$ are both enabled to present $V_i$ and $V_j$ to the output and $V_i = \bar{V}_j$. If $V_i$ and $V_j$ are directly connected to ground and $V_{DD}$ respectively, then a low resistance path will be formed from $V_{DD}$ to ground through the pass transistors in the two paths, thus causing large current to flow through them, thereby resulting in gradual performance deterioration and an ultimate device failure. The assumption of using minimum sized transistors eliminates the possibility of a "dead short" and guarantees that the short will be of several Kohms. However, a short condition is undesirable and a designer will guarantee that two pass implicants do not specify a don't care state differently such that a conflict does occur.

For a completely specified function, a given input state that permits both $P_i$ and $P_j$ to be 1 would be presenting a single entry on the Karnaugh map to the output; hence $V_i$ and $V_j$ would be in the same logic state (either 0 or 1). Therefore the pass network design presented above guarantees that no conflicts occur in completely specified functions.

**CMOS Pass Networks**

CMOS pass networks use nMOS transistors to pass 0's, and pMOS transistors to pass 1's, and a combination of both (CMOS transmission gate) to pass a variable to the output. In general, they use less number of transistors compared to CMOS gate networks and can be used to advantage in the design complex CMOS networks. In the following $f(P)$ and $f(N)$ represent respectively a partial pMOS pass function and a partial nMOS pass function. An nMOS pass network can be easily converted a CMOS pass network by the following two steps:

1. Replace all nMOS transistors with pMOS transistors if the pass variable is a 1, and with transmission gates if the pass variable is not a constant (a variable).
2. Complement all the variables applied to the Gates of pMOS transistors.

The modified pass network implementation for the pass network in Figure 1-10 to form a CMOS pass network using the above steps is given in Figure 1-11. A CMOS pass network designed in this manner may not always be optimal in its transistor count.

![Figure 1-11. A CMOS Pass Network](image-url)
A minimal transistor design of a CMOS pass network must minimize the nMOS transistors and pMOS transistors separately. A primary objective in the design of the nMOS in a CMOS pass network is to pass all 0's of the switching function, but it may also be used to pass some of the 1's if the transistor count can be reduced. The same applies to pMOS transistors also, with their logical values interchanged. The design of a minimal transistor CMOS pass network is illustrated by the following example.

**Example 3:** Consider the Karnaugh map in Figure 1-10. The nMOS pass network to cover the 0's of the function is given by: \( f(N) = D'(B') + CD(A) + A'B(0) \). The first two pass prime implicants \( D'(B') \) and \( CD(A) \) cover all the 0 cells except cell 5. This 0 cell can be covered in three different ways. They are \( A'B(0) \), \( A'C(B') \), and \( BD(A) \). If \( BD(A) \) is chosen, the function could be simplified as: \( f(N) = D'(B') + D(C(A) + B(A)) \). This needs 4 nMOS transistors to implement. Similarly, the pMOS network to cover the 1's of the function is given by: \( f(P) = D'(B') + AD(I) + B'C'(I) \). A second choice is possible by replacing the last term \( B'C'(I) \) with \( A'C'(B') \). Both of them use 5 pMOS transistors to implement. The complete pass network is shown in Figure 1-12.

\[
\begin{align*}
A & \quad C & \quad D \\
\_A & \quad B & \quad D \\
\_B & \quad D & \\
\_B & \quad A & \quad C & \quad D \\
\_I & \quad A & \quad D \\
\end{align*}
\]

\[
f_N = D(B) + D(C(A) + B(A)) \\
f_P = D(B) + AD(l) + A'C(B) \\
f = f_N + f_P
\]

**SPECIAL PROBLEMS**

1. A Full adder is a combinational logic circuit that receives inputs from the two numbers to be added and a third carry input and produces two outputs, a sum and a carry output.
   (a) Design the Full adder using logic gates
   (b) Design using nMOS pass logic
   (c) Modify (b) to make it a pMOS pass network
   (d) Modify (b) to make it a CMOS pass network
   (e) Find an optimal CMOS pass network for the Full adder
2. Find an optimal CMOS pass network for the function: \( f(A, B, C, D) = ABC + CD + B'C'D + B'C \)

3. A 4X1 MUX has four inputs \( I_0, I_1, I_2, \) and \( I_3 \) and two control signals \( A \) & \( B \). When \( AB = 00 \), \( I_0 \) is passed to the output and so on. Implement the MUX in CMOS pass logic.

4. A four variable function is given by: \( f(A, B, C, D) = \Sigma m(2, 4, 6, 7, 10, 12) \)
   
   (a) Using pass logic techniques specify the minimum size of a MUX to implement it.
   
   (b) Design the MUX in CMOS pass logic.

5. A Binary multiplier has two inputs \( A \) and \( B \) and an output \( Z \). \( A \) and \( B \) are 2 bit numbers \( A_1A_2 \) and \( B_1B_2 \) and \( Z \) is a 4 bit number \( Z_1Z_2Z_3Z_4 \).
   
   (a) Design an nMOS pass logic multiplier
   
   (b) Design a CMOS pass logic multiplier

REFERENCES


