

pass network by partitioning the Karnaugh map. An alternate technique involves factoring control variables out of the pass function.

Several synthesis approaches for pass-transistor logic were proposed in recent years [6], [10], [7], [8], [9]. For the special class of pass-transistor networks, BTS pass networks, no work has been published with the systematic methods to obtain a BTS network with a minimal transistor realization.

3. Minimization by Partitioning K-Maps

An efficient technique for partitioning the Karnaugh map and synthesizing BTS networks with reduced transistor count is described next.

3.1 General Description

Theoretically, the minimal BTS pass network can be found through exhaustively searching all the possible permutations and combinations of the control variables. This exhaustive search process, however, uses large amounts of computer time for a medium to large number of variables as shown in the following section.

An efficient method is needed to get the minimal or near minimal BTS pass network. For logic with a small number of variables, a Karnaugh map can be used to find the minimal solution by partitioning the Karnaugh map such that the resulting sub-maps correspond to the mapping of a single pass variable whenever possible. This sheds some light on how to choose the control variable for each branch of the BTS network. It is reasonable that there are some rules which can be followed which result in minimal or near minimal BTS pass network.

3.2 Exhaustive Search Technique

The BTS network minimization problem is computationally intensive. A general exhaustive search enumerates all possible realizations of the BTS network using different non-fixed hierarchies of the control variables and selects the one with the smallest number of transistors. The major problem with this technique is that the search complexity increases dramatically with the number of variables of the BTS network. The total search number s_n of n -variable BTS network can be found by recursion. Since the topmost control variable of a n -variable BTS network has n choices, and either of the two sub-branch has s_{n-1} search number, the following holds,

$$s_n = ns_{n-1}^2 \text{ and} \\ s_n = n(n-1)^2(n-2)^4(n-3)^8(n-4)^{16} \dots$$

It is not difficult to calculate that for only 5 variables, the search number increases rapidly to 1,658,880. It is almost computationally intractable to search for the minimal solution of a logic problem involving a large number of variables. To reduce the search complexity, an alternative exhaustive search technique is described in the next section.

3.3 Exhaustive Search with Fixed Hierarchy

Definition 3: A fixed hierarchy of the control variables in a

BTS network is one where all the control variables in a certain level of the tree structure are the same.

To minimize the BTS network with a reasonable amount of search complexity, there is an exhaustive search technique which uses a fixed hierarchy of the control variables in the BTS network, as shown in Figure 1. It enumerates all possible fixed hierarchies of the BTS network by changing the order of the control variables. The relatively minimal BTS network can be found in the resulting list by comparison. If there are n variables, then $P_n^n = n!$ combinations are needed to search for an exhaustive search technique with fixed hierarchy.

There are two major problems with this method. First, the exhaustive search complexity is quite large and can require a large amount of computing time. Second, the fixed hierarchy in all the levels of branches limits the possibility to find the absolutely minimal BTS network.

3.4 Recursive Control Partitioning

Finding the optimal control variable in each level of the branches can lead to a non-fixed hierarchy of a BTS network. Thus, the near minimum BTS pass network can be found recursively by searching for the optimal control variable at each level. By doing this, the search complexity is also reduced.

3.4.1 Partitioning Karnaugh Maps

The minimization oriented partitioning procedure may be explained as follows:

- 1) Choose a control variable, X_i , and form a BTS pass network with two branches and one node. One branch is controlled by X_i , and the other branch is controlled by $\overline{X_i}$. Now the original map is partitioned into two segments defined by $X_i = 0$ and $\overline{X_i} = 1$. The method to choose control variables effectively so that the transistor count is reduced will be discussed later.
- 2) Apply step 1 recursively to each segment of the map, supplying the output as a pass term to the higher level branch controlled by X_i , or $\overline{X_i}$. Do this repeatedly until the pass variable of each branch is in the set $\{0, 1, X_j, \overline{X_j}\}$.

3.4.2 Choosing the Control Variables

The control variables are chosen in a way that the largest pass implicants will be intact if possible. The algorithm works in the following way. If $f = \overline{X_i}(f1) + X_i(f2)$, pick X_i such that $f1$ and $f2$ are the simplest, which contain the largest pass implicants as possible. Namely, the largest portion of the Karnaugh map corresponding to the largest pass implicant is covered with the least number of control variables. In this way, the minimization of the transistor count in a BTS network may be achieved. This method of choosing is used in each step of the partitioning procedure. Example 1 will illustrate this.

Look for the largest pass implicant in the Karnaugh map and $\overline{B}(C)$ is found, which has 8 cells. If B is selected as the first control variable, the largest pass implicant $\overline{B}(C)$ will be intact.

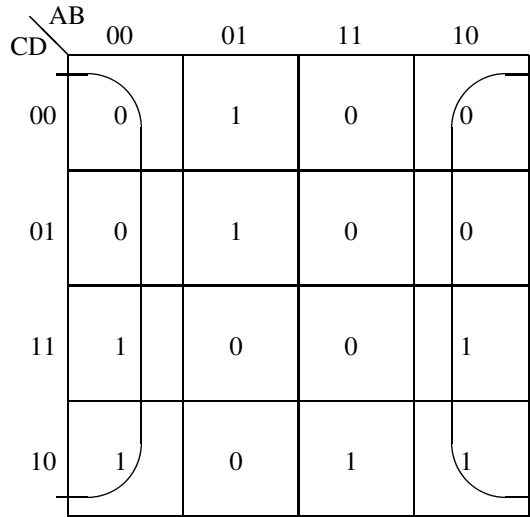


Fig. 2. Karnaugh map for example 1

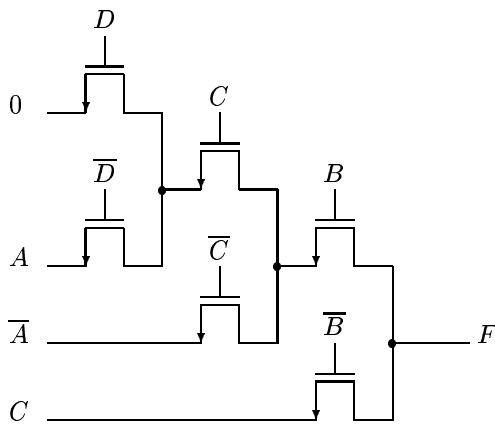


Fig. 3. BTS pass network for example 1

One branch of the BTS network is $\overline{B}(C)$. Then consider the other branch, $B\overline{C}(\overline{A})$ is the largest pass implicant, so choose C as the second control variable. Similarly, choose D as the third control variable. Up to now the partitioning is finished. The minimal number of transistors are used in this network. The resulting BTS network is shown in Figure 3. The pass function is $F = \overline{B}(C) + B(\overline{C}(\overline{A}) + C(\overline{D}(A) + D(0)))$

As the number of variables in the Karnaugh map increases, the situation will be more complicated. The control variables should be chosen more efficiently. At times, there is not just one largest pass implicant, but several largest pass implicants with the same size. When this happens, the control variable should be chosen in the way that keeps as many of the largest pass implicant intact as possible.

The algorithm can be summarized as follows:

- 1) Find the largest pass implicant(s).
- 2) If there is only one largest pass implicant, then choose

the control variable which keeps the largest pass implicant intact.

- 3) If there are more than one largest pass implicant, then find the number of intact largest pass implicants corresponding to the chosen control variables.
- 4) Choose the control variable which leads to the largest number of intact largest pass implicants.
- 5) If there are more than one largest number of intact largest pass implicants, then choose the control variable which leads to the largest number of cells covered by the intact largest pass implicants.

3.4.3 Finding the Largest Pass Implicants

If the number of variables is more than six, it is difficult to find the largest pass implicant with a Karnaugh map. A computer is needed to aid in this work. The procedure can be explained as follows,

- 1) Decide the number of control variables for the largest pass implicants that are being searched. Increase this number from 1 to $n-1$, where n is the total number of variables for the pass logic function.
- 2) For each control variable number, choose that number of control variables by mathematical permutation and see if that portion of the Karnaugh map corresponding to those control variables is a pattern of a pass implicant. If yes, then the size of the largest pass implicants is determined. Continue searching until all the pass implicants of the same size are found. They are the largest pass implicants wanted. If not, continue searching until a pattern of a pass implicant is found.

3.5 Discussion

The whole procedure shown in the previous section can efficiently generate the minimum or close to minimum BTS network in the following considerations.

- 1) The hierarchy of the control variables is not fixed. This gives the most flexibility of the technique. It enables that this technique can find the minimum realization without fixed hierarchy of the control variables.
- 2) The procedure starts with a function $f(X_1, \dots, X_n)$. By choosing a control variable X_i , f can be expressed as $f = X_i(f_1) + \overline{X}_i(f_2)$. If the control variable is the optimal one, f_1 and f_2 will be the simplest. The simplest pass functions f_1 and f_2 lead to the minimum transistor number. Repeating the procedure to find simplest sub-functions is equivalent to find the minimum BTS network solution.
- 3) Compared with the exhaustive searching technique, this new technique requires less computing time, hence the high efficiency.

4. Performance and Results

The proposed technique was implemented in C code and tested on a HP 9000 workstation. To compare the results

Number of variables	Proposed technique	Exhaustive search	Exhaustive search with fixed hierarchy
4 (Example 7)	6	6	6
5 (Example 3)	14	14	18

TABLE I
NUMBER OF TRANSISTORS

Number of variables	Proposed technique	Exhaustive search	Exhaustive search with fixed hierarchy
	10(1)	10(1)	10(1)
	12(2)	12(2)	14(2)
	14(12)	14(12)	14(3) 16(4) 18(4) 22(1)
5	16(16)	16(16)	16(6) 18(7) 20(3)
	18(29)	18(29)	18(21) 20(7) 22(1)
	20(28)	20(28)	20(17) 22(9) 24(2)
	22(12)	22(12)	22(8) 24(4)

TABLE II
NUMBER OF TRANSISTORS F(G): F = NO OF TRANSISTORS, B = NO. OF
BTS NETWORKS WITH F TRANSISTORS

generated by the new technique with the minimal results, an exhaustive search technique with non-fixed hierarchy of control variables was also written in C code and tested.

Table I shows the number of transistors required by each technique for several pass functions.

Table II shows the number of transistors required by each technique for 100 different 5-variable pass logic circuits. The notation "14(12)" under the "new technique" column indicates that there are 12 pass logics realized with 14 transistors by the new technique. By the exhaustive search technique with non-fixed hierarchy, the 12 pass logics are also realized with 14 transistors. By the exhaustive search technique with fixed hierarchy, only 3 of them are realized with 14 transistors, 4 of them are realized with 16 transistors, 4 of them are realized with 18 transistors, and 1 of them is realized with 22 transistors.

As Table I and Table II show, the exhaustive search technique with non-fixed hierarchy always generates BTS networks with minimal number of transistors. The new technique also produces BTS networks with a minimal number of transistors. The exhaustive search technique with fixed hierarchy sometimes generates results with more transistors. For 5-variable logic, the new technique requires 1 transistor less than the exhaustive search technique with fixed hierarchy on average, as can be concluded from Table II.

The proposed technique was also compared with the genetic programming (GP) approach presented by A. Hernandez

Aguirre, B. P. Buckles, and C. Coello Coello on three examples in [13]. Table III shows the comparison results.

Logic Function	Proposed technique	Genetic programming
$F = X_1 X_2 + X_3 X_4 + X_5 X_6$	20	12
$F = X_1 X_4 + X_2 X_5 + X_3 X_6$	20	20
$F = X_1 + X_2 + X_3$	6	10

TABLE III
COMPARISON WITH GENETIC PROGRAMMING

As shown in Table III, the GP approach gives a better solution to the first logic function. This is due to the restriction of the proposed technique that fan out of transistors is 1. A fan out of 1 restriction greatly enhances the layout hence densities of final circuits. For this reason, the fan out restriction was employed in this work. The GP approach did not have this restriction. The number of required transistors is 20 for both methods for the second logic function. For the third logic function, the new technique requires 4 transistors less than the GP approach. The point is that the GP approach allows only 0 and 1 to be pass variables while any element in the set $\{0, 1, X_i, \bar{X}_i\}$ can be passed in the new technique. Furthermore, the calculation complexity of the GP approach is large.

Table IV shows the corresponding CPU time required to execute the algorithm for each technique.

Number of variables	Proposed technique	Exhaustive search	Exhaustive search with fixed hierarchy
4 (Example 7)	0.01	0.16	0.89
5 (Example 3)	0.02	147.0	0.94

TABLE IV
CPU TIME

The CPU time required by the new technique is the smallest, which is far less than the other two techniques, as shown in Table IV.

Table shows the CPU time required by the new technique for large number of variables. It is applicable of the new technique for variable numbers up to 12.

Number of variables	CPU Time (sec)
4	0.01
5	0.02
6	0.05
7	0.30
8	2.23
9	10.40
10	79.30
11	604.83
12	3205.24

TABLE V
CPU TIME

Thus, high efficiency is obtained while the computational complexity is reduced by the proposed new technique.

5. Conclusion

A new design technique for the minimization of BTS pass networks was presented. The proposed technique exploited the potential for circuit design optimization using BTS pass networks since they have inherent advantage in reducing the area. The resulting implementations are promising when compared to the exhaustive search technique with or without fixed hierarchy of control variables in terms of transistor count, computational complexity and CPU time.

The new technique was also compared with the Genetic Programming (GP) synthesis approach [13]. The new technique is more efficient compared with the GP approach in several aspects. The initial point will affect the performance of the GP approach. At present, finding the initial point is not a trivial issue. A large convergence time often occurs and is determined by the nature of genetic programming. A given design using the GP approach may not converge.

REFERENCES

- [1] D. Radhakrishnan, S.R. Whitaker, and G.K. Maki, "Formal Design Procedures for Pass Transistor Switching Circuits", *IEEE Journal of Solid-State Circuits*, Vol. SC-20, pp. 531-536, April 1985.
- [2] T.S. Cheung and K. Asada, "Regenerative pass-transistor logic: A circuit technique for high speed digital design", *IEICE Transactions Electron.*, E79-C(9):1274-1283, 1996.
- [3] F.S. Lai and W. Hwang, "Design and implementation of differential cascode voltage switch with pass-gate (dcvspg) logic for high-performance digital systems", *IEEE Journal of Solid-State Circuits*, 32(4):563-573, April 1997.
- [4] A. Parameswar, H. Hara, and T. Sakurai, "A high speed, low power, swing restored pass-transistor logic based multiply and accumulate circuit for multimedia applications", *Proceedings of the Custom Integrated Circuits Conference*, pages 278-281, May 1994.
- [5] K. Yano, T. Yamanaka, T. Nishida, and M. Satio, "A 3.8-ns CMOS 16x16 multiplier using complementary pass-transistor logic", *IEEE Journal of Solid-State Circuits*, 25(2):388-395, April 1990.
- [6] R. Chaudhry, T.H. Liu, A. Aziz, and J.L. Burns, "Area-oriented synthesis for pass-transistor logic", *International Conference on Computer Design*, pages 160-167, 1998.
- [7] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass-transistor logic design", *IEEE Journal of Solid-State Circuits*, 31(6):792-803, June 1996.
- [8] M. Suzuki, N. Ohkubo, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome, "A 1.5ns CMOS 16x16 multiplier using complementary pass-transistor logic", *IEEE Journal of Solid-State Circuits*, 28(11):599-602, Nov. 1993.
- [9] V.G. Oklobdzija and B. Duschene, "Pass-transistor dual value logic for low-power CMOS", *Proceedings of the International Symposium on VLSI Technology*, Taiwan, May-June 1995.
- [10] T.-H. Liu, A. Aziz, and J.L. Burns, "Performance driven synthesis for pass-transistor logic", *International Workshop on Logic Synthesis*, pages 255-259, 1998.
- [11] A. Hernandez Aguirre, C. Coello Coello, and B.P. Buckles, "A Genetic Programming Approach to Logic Function Synthesis by Means of Multiplexers", *The First NASA/DoD Workshop on Evolvable Hardware*, pages 46-53, 1999.
- [12] G.E. Peterson and G.K. Maki, "Binary Tree Structured Logic Circuits: Design and Fault Detection", *Proceedings of International Conference on Computer Design*, pages 671-676, October 1984.
- [13] A. Hernandez Aguirre, C. Coello Coello, and B.P. Buckles, "Circuit Design Using Genetic Programming: An Illustrative Study", *10th NASA Symposium on VLSI Design*, pages 4.1.1-4.1.10, March 2002.