OPTIMAL UNIVERSAL LOGIC MODULES FOR THE SYNTHESIS OF ARBITRARY LOGIC FUNCTIONS

Behrooz Parhami

Dept. of Electrical & Computer Engineering
Univ. of California
Santa Barbara, CA 93106, USA

It is well-known that any n-variable logic function can be realized by a 2^{n-1}-input multiplexer (universal logic module). In this paper, we formulate a computationally simple condition to determine whether a given n-variable logic function is realizable by a multiplexer having 2^{n-2} or fewer inputs and present an algorithm for finding the set of control variables for an optimal multiplexer realization.

Index Terms: Boolean algebra. Combinational logic functions. Multiplexers. Switching circuits. Universal logic modules.

1. INTRODUCTION

Many standard logic design textbooks (e.g., [MANO84], [ROTH85]) discuss the implementation of logic functions by multiplexers, using no additional external logic; thus the name *universal logic module* [YAUS70] for a multiplexer. A 2^m-input *multiplexer* or *selector*, sometimes called a 2^m-to-1 multiplexer, is a circuit with m control inputs c_i $(0 \le i \le m-1)$, 2^m data inputs d_j $(0 \le j \le 2^m-1)$, and a single binary output e. The output e is equal to the *selected* input d_k, where k is the value of the binary number $c_{m-1} \ldots c_1 c_0$. To use a 2^{n-1}-input multiplexer for realizing a given n-variable logic function $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$, we arbitrarily select $n-1$ of the input variables, say $x_{n-1}, \ldots, x_2, x_1$ as control inputs, with 0, 1, x_0, or x_0' connected to the data input d_j as determined by the pair of values in lines $2j$ and $2j+1$ of the given function's truth table (i.e., $00 \Rightarrow 0$, $01 \Rightarrow x_0$, $10 \Rightarrow x_0'$, $11 \Rightarrow 1$). This realization may require a single external inverter if complemented inputs are not available. It is natural to ask whether for realizing a given n-variable logic function, a smaller multiplexer (i.e., one having 2^{n-2} or fewer inputs) will do.

The above question can of course be answered by an exhaustive search. To see whether a 2^m-input multiplexer can be used to realize the given n-variable logic function, one writes down the Shannon expansion of the function for every possible set of the m expansion variables. Any expansion in which all coefficient (residual) functions are single-variable can be the basis of the desired realization. Although techniques for facilitating such an exhaustive search have been developed [TABL76], the process is still very time consuming. Thus we are motivated to formulate a computationally simple condition to determine whether a given n-variable logic function is realizable by a multiplexer having 2^{n-2} or fewer inputs and an algorithm for finding the subset of $n-2$ or fewer variables to be used as control inputs of such a multiplexer [PARH89].

2. NOTATION AND DEFINITIONS

Suppose that we have realized a given n-variable logic function $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ using a 2^m-input multiplexer with at most $n-m$ inverters as the only external logic. Let S denote the set of n input variables, S_c the set of m control-input variables, and $S_d = S - S_c$ the set of $n-m$ data-input variables (henceforth the *single-dependence* set). The problem of finding the smallest multiplexer that can realize the function f can be restated as that of determining the maximal single-dependence set S_d. Note that for the straightforward 2^{n-1}-input multiplexer realization discussed above, we have $S_d = \{x_0\}$. We will assume that the logic expressions for *all* minimal sum-of-products representrations of f are at hand. Other specifications are convertible to minimal sum-of-products form(s) using efficient known algorithms. If a minimal sum-of-products representation for f consists of t product terms, we write

$$f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0) = \pi_0 + \pi_1 + \ldots + \pi_{t-1} ,$$

where the product term π_p is

$$\pi_p = \prod_{q=0}^{n-1} (a_{pq} x_q \oplus c_{pq}) .$$

The binary constant a_{pq} signifies the "appearance" of x_q in the pth product term π_p and the binary constant c_{pq} indicates the "complementation" of x_q in π_p. That is:

$a_{pq} = 0$	$c_{pq} = 0$	not allowed;
$a_{pq} = 0$	$c_{pq} = 1$	neither x_q nor x_q' appears in π_p;
$a_{pq} = 1,$	$c_{pq} = 0$	x_q appears in π_p;
$a_{pq} = 1$	$c_{pq} = 1$	x_q' appears in π_p.

A total of $2tn$ binary constants are needed to specify each distinct minimal sum-of-products representation of f, where $t \le 2^{n-1}$.

In Sections 3 and 5, we formulate a computationally simple condition for determining the existence of a multiplexer realization for the given logic function $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ using a multiplexer with 2^{n-2} or fewer inputs. Once the existence of such a multiplexer realization is established, Algorithm 1 given in Section 7 can be used to obtain a corresponding maximal S_d set.

89CH2785-4/90/0000-0212$01.00 © 1990 IEEE

3. A NECESSARY CONDITION

We start by formulating a necessary condition for the existence of a 2^{n-2}-input multiplexer realization of f.

Lemma 1: The logic function $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ has a 2^{n-2}-input multiplexer realization, with $S_d = \{x_i, x_j\}$ as the single-dependence set, only if f has a minimal sum-of-products representation in which x_i and x_j do not appear together in any product term; i.e., only if for some minimal sum-of-products representation of the function f, we have $\sum_p a_{pi}a_{pj} = 0$.

Proof: Suppose that f has a 2^{n-2}-input multiplexer realization with $S_d = \{x_i, x_j\}$ as the single-dependence set and that this realization corresponds to the following Shannon expansion of f with $S_c = S - S_d$ as the set of expansion variables:

$$f = \sum_{g=0}^{2^{n-2}-1} [\prod_{q \in S_c} (x_q \oplus \gamma_{gq})] [\phi_g(x_i, x_j)].$$

The $n-2$ binary constants γ_{gq}, $q \in S_c$, constitute the binary representation of g and each of the functions $\phi_g(x_i, x_j)$ is constant or single-variable; i.e., $\phi_g(x_i, x_j) = 0, 1, x_i, x_i', x_j$, or x_j'. We show that from the above expression, we can always obtain a minimal sum-of-products expression for f that does not have both x_i and x_j in any product term. We know from switching theory that any sum-of-products expression can be transformed to a minimal sum-of-products expression by repeated application of the following steps (see, e.g., [ZISS72], pp. 22-28):

1. Finding a pair of product terms of the form $x\alpha$ and $x'\beta$, leading to the "optional product" term $\alpha\beta$.
2. Deleting any product term of the form $\alpha\beta\gamma$ (such a term is covered by the two "parents" of $\alpha\beta$).
3. Replacing the parernt term $x\alpha$ (or $x'\beta$) by α (or β) if $\alpha\beta = \alpha$ (or $\alpha\beta = \beta$).
4. Replacing both $x\alpha$ and $x'\beta$ by $\alpha\beta$ if there are two other product terms that generate $x\alpha$ and $x'\beta$ as optional products when they are combined with $\alpha\beta$.

If we show that repeated application of these rules starting with the above Shannon expansion never results in an expression with a product term containing both x_i and x_j, the proof will be complete. We prove this result by induction. Clearly, the initial Shannon expansion does not contain both x_i and x_j in the same product term. Suppose that at some stage we do not have both x_i and x_j in any product term and consider the application of Rule 2, 3, or 4. Rules 2 and 3 only simplify the logic expression and thus do not introduce any extra variable. Rule 4 can potentially introduce x_i and x_j in the optional product term $\alpha\beta$ only if the parent products are of the form $xx_i\phi$ and $x'x_j\psi$ for some $x \in S_c$ (since if the parents are $x_i\alpha$ and $x_i'\beta$ or if they are $x_j\alpha$ and $x_j'\beta$, the optional product $\alpha\beta$ will not contain x_i or x_j). The optional product term is then $x_ix_j\phi\psi$ which would require a product term containing $x_i'x_j$ or x_ix_j' to reproduce its parent terms. But such product terms do not exist in our expression by assumption. Thus, $\alpha\beta$ is introduced as a new term in the expression only if it does not contain both x_i and x_j. ■

Corollary 1: The logic function $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ has a 2^{n-2}-input multiplexer realization, with $S_d = \{x_i, x_j\}$ as the single-dependence set, only if f has no essential prime implicant in which x_i and x_j appear together.

Proof: Immediate upon noting that an essential prime implicant appears in *all* minimal sum-of-products representations of f. ■

4. APPLICATION EXAMPLES

By Corollary 1, any function having a minterm as a prime implicant (e.g., the parity function) cannot be realized by a multiplexer with fewer than 2^{n-1} inputs, since such a prime implicant is always essential and contains all the n variables. Also, many symmetric functions do not have simpler multiplexer realizations. Corollary 1 enables us to rule out many functions as candidates for simpler multiplexer realizations directly from a list of essential prime implicants without knowing or having to obtain all minimal sum-of-products forms. Example 1 below shows that the necessary condition given by Lemma 1 (or Corollary 1) is not sufficient and Example 2 shows that Corollary 1 alone is not adequate in that some functions are ruled out by Lemma 1 and not by Corollary 1.

Example 1: Consider the 4-variable logic function

$$f(x_3, x_2, x_1, x_0) = x_0x_3 + x_0'x_1 + x_1'x_2.$$

There are three candidate S_d sets $\{x_0,x_2\}$, $\{x_1,x_3\}$, and $\{x_2,x_3\}$, whose members do not appear together in any product term of the above unique minimal sum-of-products form. However, none of the three sets leads to a 4-input multiplexer realization as seen from the three Shannon expansions

$$\begin{aligned} f &= x_1x_3\,[1] + x_1x_3'\,[x_0'] + x_1'x_3\,[x_0+x_2] + x_1'x_3'\,[x_2] \\ &= x_0x_2\,[x_1'+x_3] + x_0x_2'\,[x_3] + x_0'x_2\,[1] + x_0'x_2'\,[x_1] \\ &= x_0x_1\,[x_3] + x_0x_1'\,[x_2+x_3] + x_0'x_1\,[1] + x_0'x_1'\,[x_2]. \blacksquare \end{aligned}$$

Example 2: Consider the following 3-variable logic function given with its two minimal sum-of-products expressions:

$$f(x_2, x_1, x_0) = x_0x_1' + x_0'x_2' + x_1x_2 = x_0x_2 + x_0'x_1 + x_1'x_2'$$

The function does not have any essential prime implicants, so Corollary 1 is of no help. However, both minimal sum-of-product representations above contain product terms with every pair of variables. Thus by Lemma 1, f does not have a 2-input multiplexer realization. Again, this conclusion is verified by examining the three Shannon expansions of the function with respect to one variable. ■

5. MAKING IT SUFFICIENT

Even though the above conditions quickly rule out many functions as candidates for simpler multiplexer realizations, we need a necessary and sufficient condition to deal with other functions.

Lemma 2: The set $\{x_i, x_j\}$ satisfying the condition of Lemma 1 is a valid single-dependence set S_d for multiplexer realization of f if and only if for every pair of product terms $(x_i \oplus \gamma)\phi$ and $(x_j \oplus \delta)\psi$ in the corresponding minimal sum-of-products expression for f (i.e., so that one product term contains the literal x_i or x_i' and the other term contains the literal x_j or x_j'), the product $\phi\psi$ is either identically zero or an implicant of f.

Proof: We prove the "only if" part by contradiction. Suppose that we have a pair of product terms $(x_i \oplus \gamma)\phi$ and $(x_j \oplus \delta)\psi$ so that $\phi\psi$ is not identically zero and not an implicant of f. Then, there exists

an assignment of values to the $n-2$ variables in $S_c = S - \{x_i, x_j\}$ that makes $\phi\psi$ equal to 1 and f equal to 0. Substitute one such set of values in the expression for f to get

$$0 = (x_i \oplus \gamma) + (x_j \oplus \delta) + h(x_i, x_j) ,$$

where $h(x_i, x_j)$ is what remains from all other product terms. This is contradictory, since we can make the right hand side equal to 1 by choosing $x_i = \gamma'$ or $x_j = \delta'$. To prove the "if" part, suppose that for every pair of product terms $(x_i \oplus \gamma)\phi$ and $(x_j \oplus \delta)\psi$, the product $\phi\psi$ is either identically zero or an implicant of f. We show that the Shannon expansion of f cannot contain a component term of the form $[\Pi_q (x_q \oplus \gamma_{gq})]\ [(x_i \oplus \gamma) + (x_j \oplus \delta)]$; i.e. that no residual function in the Shannon expansion of f, with $S_c = S - \{x_i, x_j\}$ being the set of expansion variables, is of the form $(x_i \oplus \gamma) + (x_j \oplus \delta)$. For suppose we have such a component term in the Shannon expansion. This component must have resulted from expanding two product terms $(x_i \oplus \gamma)\phi$ and $(x_j \oplus \delta)\psi$ in the minimal sum-of-products form. But if there are such product terms, $\phi\psi$ and thus $\Pi_q (x_q \oplus \gamma_{gq})$ must be an implicant of f and the Shannon expansion of f must contain $\Pi_q (x_q \oplus \gamma_{gq})$ instead of $[\Pi_q (x_q \oplus \gamma_{gq})]\ [(x_i \oplus \gamma) + (x_j \oplus \delta)]$. ■

Example 3: It is easily seen that in the minimal sum-of-products representation $x_0x_3 + x_0'x_1 + x_1'x_2$ of the 4-variable function $f(x_3, x_2, x_1, x_0)$ of Example 1, the product terms x_0x_3 and $x_1'x_2$ violate the condition of Lemma 2 for both candidate S_d sets $\{x_1,x_3\}$ and $\{x_2,x_3\}$ because neither x_0x_2 nor x_0x_1' is an implicant of f. Thus, the function does not have a simpler multiplexer realization. ■

6. OPTIMAL REALIZATION

So far we have only dealt with single-dependence sets of size 2. The following result shows that larger single-dependence sets can be easily constructed from single-dependence sets of size 2 by a simple "transitivity" check. The largest single-dependence set for a given function obviously corresponds to its optimal multiplexer realization.

Lemma 3: If S_d is a single-dependence set for the n-variable function $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ and a variable $x_i \notin S_d$ satisfies the conditions of Lemma 1 and Lemma 2 with every variable $x_k \in S_d$ (i.e., $\{x_i, x_k\}$ is also a single-dependence set for all $x_k \in S_d$), then $S_d \cup \{x_i\}$ is a single-dependence set for f.

Proof: Consider the multiplexer realization of f based on the single-dependence set S_d, with x_i assigned as the least significant control variable. Then, in the smaller multiplexer realization of f based on the single-dependence set $S_d \cup \{x_i\}$, the jth data input line must assume the value $x_i'y_{2j} + x_iy_{2j+1}$, where y_{2j} and y_{2j+1} are single-variable input expressions corresponding to the original multiplexer realization. We must show that the above expression necessarily represents a single-variable function. The only way that this can occur is if both y_{2j} and y_{2j+1} are constants or identical literals. To show that this is indeed always the case, we have to prove that the four representative cases of

1. $y_{2j} = x_k$, $y_{2j+1} = 0$,
2. $y_{2j} = x_k$, $y_{2j+1} = 1$,
3. $y_{2j} = x_k$, $y_{2j+1} = x_k'$,
4. $y_{2j} = x_k$, $y_{2j+1} = x_l$,

with $x_k, x_l \in S_d$, are impossible. In these cases, when the values of all selection variables except for x_i are fixed, the function is transformed into $x_i'x_k$, $x_i'x_k + x_i = x_k + x_i$, $x_i'x_k + x_ix_k'$, and $x_i'x_k + x_ix_l$, respectively. In each case, $\{x_i, x_k\}$ cannot be a single dependence set; clearly a contradiction. ■

7. THE DESIGN PROCEDURE

Putting all of the previous results together, we are led to the following algorithm for obtaining all optimal multiplexer realizations of a given logic function.

Algorithm 1: (Constructing all maximal single-dependence sets for an n-variable logic function f).

Step 0 - Find all minimal sum-of-products representations of f. Then perform Steps 1 through 3 for each minimal sum-of-products form.

Step 1 - Construct a lower-triangular $n-1$ by $n-1$ table with all entries initially blank, labelling the $n-1$ rows as $x_1, x_2, \ldots, x_{n-1}$ and the $n-1$ columns as $x_0, x_1, \ldots, x_{n-2}$. Examine every product term in the selected minimal sum-of-products representation of f and for every pair of variables appearing together in a product term, mark the corresponding table entry with an "X". At the end of this process, entries that remain blank identify two-element S_d candidates.

Step 2 - For every two-element S_d candidate $\{x_i, x_j\}$ obtained in Step 1, check the condition of Lemma 2; i.e., check to see whether two product terms, one of the form ϕx_i or $\phi x_i'$ and the other of the form ψx_j or $\psi x_j'$, exist such that $\phi\psi$ is not identically zero and not an implicant of f. If so, delete the pair from the list of S_d candidates.

Step 3 - Merge the remaining sets into the largest possible S_d sets according to the procedure of Lemma 3; i.e., find the largest sets of variables such that every pair of variables in each set corresponds to a single-dependence set.

Step 4 - Select the largest S_d sets obtained from various minimal sum-of-products expressions. ■

We can now state the main result of this paper.

Theorem 1: Algorithm 1 yields all maximal single-dependence sets for a given logic function f.

Proof: Immediate from Lemmas 1, 2, and 3. ■

8. MORE EXAMPLES

We now present a few more examples to illustrate the application of Algorithm 1 and to identify several interesting special cases.

Example 4: Consider the problem of finding a minimal multiplexer realization for the 4-variable logic function defined by the following unique minimal sum-of-products expression:

$$f(x_3, x_2, x_1, x_0) = x_0x_2' + x_0x_3 + x_2'x_3 + x_1x_2$$

In Step 1 of the algorithm, we obtain two candidate S_d sets, $\{x_0,x_1\}$ and $\{x_1,x_3\}$. We know right away by the absence of $\{x_0,x_3\}$ in the above set, that the best we can hope for is a 4-input multiplexer realization. Neither set passes the test of Step 2 since we have the product terms x_0x_3 and x_1x_2 with neither x_2x_3 nor x_0x_2 an implicant of f. Thus, the given function f does not have a 4-input multiplexer realization. ■

Example 5: Consider the problem of constructing a minimal multiplexer realization for the 4-variable function

$$f(x_3, x_2, x_1, x_0) = x_0x_2 + x_0x_3 + x_1'x_2 + x_0'x_1x_2'$$

represented by its unique minimal sum-of-products expression. From Step 1 of the algorithm, we find two candidate S_d sets, $\{x_1, x_3\}$ and $\{x_2, x_3\}$. The set $\{x_2, x_3\}$ does not pass the test of Step 2. The remaining single-dependence set $\{x_1, x_3\}$ yields the 4-input multiplexer realization defined by the Shannon expansion:

$$x_0x_2\,[1] + x_0x_2'\,[x_3] + x_0'x_2\,[x_1'] + x_0'x_2'\,[x_1]. \blacksquare$$

Example 6: Consider the problem of constructing a minimal multiplexer realization for the 6-variable logic function

$$f = x_1'x_2 + x_0x_2 + x_1x_2'x_3' + x_1'x_4'x_5,$$

where the expression is the unique minimal sum-of-products form. In Step 1 of the algorithm, we find eight candidate S_d sets, $\{x_0,x_3\}$, $\{x_0,x_4\}$, $\{x_0,x_5\}$, $\{x_3,x_4\}$, $\{x_3,x_5\}$, $\{x_0,x_1\}$, $\{x_2,x_4\}$, and $\{x_2,x_5\}$, the first five of which pass the test of Step 2. Step 3 yields two maximal single-dependence sets, $\{x_0,x_3,x_4\}$ and $\{x_0,x_3,x_5\}$, with two corresponding 8-input multiplexer realizations having data inputs $x_5, x_3', 1, x_0, 0, x_3', 1, x_0$ and $0, x_3', 1, x_0, x_4', x_3', 1, x_0$. $\blacksquare$

Example 7: This last example is interesting because it involves a function with two minimal sum-of-products expressions, each of which yields a different minimal multiplexer realization. Consider the 4-variable logic function f defined by the following two minimal sum-of-products expressions

$$\begin{aligned} f &= x_0x_1x_3 + x_0x_1'x_2 + x_0'x_1x_2' + x_0'x_1'x_3' \\ &= x_2x_3x_0 + x_2x_3'x_1' + x_2'x_3x_1 + x_2'x_3'x_0' \end{aligned}$$

The first expression yields the candidate S_d set $\{x_2, x_3\}$ while the second expression yields the candidate S_d set $\{x_0, x_1\}$. Both sets pass the test of Step 2, giving the two different 4-input multiplexer realizations defined by the original expressions (that happen to be in the required Shannon expansion form, with the expansion variable set being $\{x_0, x_1\}$ and $\{x_2, x_3\}$, respectively). $\blacksquare$

9. CONCLUSION

We have presented an efficient algorithm for finding minimal multiplexer realizations for arbitrary logic functions. The algorithm requires all minimal sum-of-products representations of the logic function as input, but this presents no problem as other representations can be converted to minimal sum-of-products form using efficient known algorithms. It would be interesting to find efficient algorithms that work directly on other representations (e.g., on the function's truth table) without a need for finding the minimal sum-of-products representations first.

Analysis of the computational complexity of this algorithm constitutes another possible area for further investigation. Both worst-case and average-case analyses need to be considered. However, both analyses appear to be difficult. It is not clear what input conditions constitute the worst case for the algorithm. For example, the parity function of n variables has the largest possible number of product terms in its minimal sum-of-products representation. But for this function, the large number of terms only lengthens the execution of Step 1 of the algorithm. The results of Step 1 then lead directly to the conclusion that a smaller multiplexer realization is impossible, thus saving the execution time associated with Steps 2 and 3. Average-case analysis is similarly difficult but may be possible through the use of results on the characteristics of random Boolean functions [MILE64], [FLEI89].

It was noted in Example 7 that different minimal sum-of-products representations for a given function may yield different minimal multiplexer realizations (Example 6 showed that a single minimal expression can also lead to multiple minimal multiplexer realizations). It is conjectured that these realizations obtained from different minimal sum-of-products forms are always equally optimal, so that one can obtain a minimal multiplexer realization from *any* minimal sum-of-products form, thus saving a great deal of effort in generating and handling of multiple minimal sum-of-product forms. However, this conjecture has not been proven yet.

ACKNOWLEDGEMENT

The research reported in this paper was initiated while the author was a Visiting Professor with the School of Computer Science, Carleton University, Ottawa, Canada.

REFERENCES

[EDWA76] Edwards, C.R. and S.L. Hurst, "An Analysis of Universal Logic Modules," *International Journal of Electronics*, Vol. 41, No. 6, pp. 625-628, 1976.

[FLEI89] Fleisher, H., J. Giraldi, R. Phoenix, and M. Tavel, "Minimizability of Random Boolean Functions," *IEEE Transactions on Computers*, Vol. 38, No. 4, pp. 593-595, Apr. 1989.

[MANO84] Manò, M.M., *Digital Design*, Prentice-Hall, Englewood Cliffs, NJ, 1984, pp. 175-182.

[MILE64] Mileto, F. and G. Potzolu, "Average Values of Quantities Appearing in Boolean Function Minimization," *IEEE Transactions on Electronic Computers*, Vol. EC-13, pp. 87-92, Apr. 1964.

[PARH89] Parhami, B., "Minimal Multiplexer Realization of Logic Functions," *Canadian Journal of Electrical and Computer Engineering*, Vol. 14, No. 2, 1989.

[ROTH85] Roth, C.H., Jr., *Fundamentals of Logic Design*, West Publishing Company, St. Paul, Third Edition, 1985, pp.198-201.

[TABL76] Tabloski, T.F. and F.J. Mowle, "A Numerical Expansion Technique and Its Application to Minimal Multiplexer Logic Circuits," *IEEE Transactions on Computers*, Vol. C-25, No. 7, pp. 684-702, July 1976.

[YAUS70] Yao, S.S. and C.K. Tang, "Universal Logic Modules and Their Applications," *IEEE Transactions on Computers*, Vol. C-19, No. 2, pp. 141-149, Feb. 1970.

[ZISS72] Zissos, D., *Logic Design Algorithms*, Oxford Univ. Press, 1972.