A digital approach to the efficient synthesis of threshold gates

G. C. E. Winn

Department of Electronics, University of Southampton, Southampton SO9 5NH

The problems of logic circuit complexity encountered in employing a conventional Boolean implementation technique for the synthesis of threshold gates have resulted in proposals for an alternative technology for this purpose. In contrast to this, this paper presents a new technique for the efficient synthesis of threshold gates from Boolean logic gates. Results obtained using this technique for a range of different gate designs are shown to provide a highly efficient implementation without incurring the large propagation delays created by factorisation of the minimised sum-of-product or product-of-sum expressions. The technique thus provides an attractive solution for the implementation of threshold gates from conventional technologies.

(Received October 1973)

1. Introduction

A threshold gate with n inputs and threshold t (where $t < n$) provides a logic 1 output whenever the threshold t is exceeded and otherwise provides a logic ∅ output. In the case of all inputs having unity weighting, the threshold t is considered to be exceeded when t or more of the inputs are in the logic 1 state. In the case of inputs having other than unity weighting (usually integers > 1) the threshold t is considered to be exceeded when the sum of the weights of the inputs in the logic 1 state is equal to or greater than the threshold t.

The requirement for threshold functions* arises frequently in logic systems. In associative memories it is frequently necessary to distinguish between single and multiple (2 or more) responses to a match instruction requiring a gate with a threshold of two. In a fault tolerant decoder with, for example, eight inputs, it might be required to provide an output indicating when, say, six or more of the inputs are satisfied. This function could be provided either by a gate with a threshold of six detecting satisfied inputs or a gate with a threshold of three detecting unsatisfied inputs. An advantage of using threshold gates for such applications is that by employing a gate with more inputs than required, its effective threshold can easily be modified by connecting the spare inputs to a logic 1 or a logic ∅ as required.

Threshold gates are not, however, only useful in these more obvious threshold function areas. It is well known that threshold gates can be employed in the implementation of almost any logic function (including bistables) often offering a considerable improvement in implementation efficiency (assuming that the comparison is one of threshold gate count compared to Boolean gate count). Because of this there is a strong body of opinion which believes that ultimately threshold gates will provide a higher level, more powerful, general purpose building block in logic systems of the future, largely replacing NAND and NOR gates which are in fact themselves specific cases of threshold gates. Quite a lot of work has been published on the use of threshold gates in the design of logic circuits and is well summarised in the book by Hurst (1971).

Threshold functions, although conceptually very simple, have so far proved difficult to implement. This is due to the large number of terms appearing in their minimised sum-of-product (SOP) or product-of-sum (POS) expressions. A direct implementation in Boolean logic gates would therefore result in a prohibitively large number of gates. This could obviously be reduced to some extent by factorisation, but the propagation time is seriously degraded by this approach.

Because of the improvement in logic circuit implementation efficiency offered by threshold gates and their consequent potential as general purpose logic building blocks, it was felt that there was a strong need for fully integrated devices. However, due to the disadvantages arising from a straightforward Boolean implementation approach, alternative methods were sought. The technique eventually employed was one of analogue summation, whereby analogue currents generated by each input in the logic 1 state were summed and applied to a threshold level detector. Although this technique was used as early as 1967 to fabricate a fully integrated gate (Amodei, Winder, Hampel and Mayhew, 1967), the cost and fabrication problems of this process, so far, appear to have prevented fully integrated devices from becoming commercially available.

The technique presented here provides an attractive alternative by which threshold functions can be implemented from currently available Boolean logic gates. The future fabrication of fully integrated threshold gates from existing processes such as TTL and MOS, is thus made possible without invoking a new technology.

2. Principle of the proposed technique

The technique is best illustrated by an example. Consider the case of $n = 9$, $t = 3$. If the 9 input variables are represented by ABCDEFGHJ then the minimised sum-of-products expression consists of every possible combination of three of these,

i.e. ABC + ABD + ABE + ABF etc.

and contains 84 terms. A direct implementation of this in Boolean gates would therefore require more than 84 gates.

Consider the expression:

$$ABC + DEF + GHJ .$$

This contains all of the input variables, provides three of the required terms and would require four gates for its implementation. Consider now the expression:

$$(A + B + C)(D + E + F)(G + H + J) .$$

This also contains all of the input variables and requires an identical number of gates and gate inputs and outputs but instead of providing only three of the required terms it provides 27. Hence, by employing the minimum number of expressions of the above form necessary to provide all of the required sum of product terms an extremely efficient implementation can be achieved.

Initially the 9 (n) inputs are arbitrarily divided into 3 (t) equal groups (it is assumed that n/t is an integer), the members of each group being OR-ed together and the groups themselves being AND-ed together.

i.e. $(A + B + C)(D + E + F)(G + H + J)$.

*A threshold function is defined as a logic function which can be realised using only a single threshold gate.

As pointed out above, this expression provides 27 of the required 84 terms.

To obtain the next expression every possible expression of this form is exhaustively searched to determine which provides the largest number of new terms.* All expressions of the above form will provide 27 terms but in every case except the first, a number of these will have already been provided by the previous expressions. For this particular example the second expression is:

$$(A + D + G)(B + E + H)(C + F + J) .$$

This contributes 21 new terms, 6 having been provided previously. This procedure is repeated until no new terms can be found for any expression searched. In this particular example two more expressions are found before this occurs

$$\text{i.e. } (A + E + J)(B + F + G)(C + D + H)$$

and

$$(A + F + H)(B + D + J)(C + E + G) .$$

Each of these two expressions contributes 18 new terms, thus satisfying the required total of 84. An exhaustive search of all stored expressions for each of the required sum-of-product terms in turn can be used to check that all terms are provided. During this procedure any variables not required in the generation of any of these terms can, if required, be eliminated from the expression concerned (e.g. C and D from the final expression for $n = 6$, $t = 2$ in Appendix 1.1). In the example under consideration here, all terms are required in all expressions and the complete Boolean implementation expression is provided by the sum of all generated expressions,

$$\begin{aligned}\text{i.e. } T = {} & (A + B + C)(D + E + F)(G + H + J) \\ & + (A + D + G)(B + E + H)(C + F + J) \\ & + (A + E + J)(B + F + G)(C + D + H) \\ & + (A + F + H)(B + D + J)(C + E + G) .\end{aligned}$$

This, it can be seen, provides an extremely elegant and attractive solution requiring only 17 gates with a maximum fan-in of 4. It has the additional advantage of a small propagation delay (3 gate delays), a small loading on the circuit inputs (4 logic loads) and the symmetry of the overall expression makes the loading on every circuit input equal. The above expression therefore provides an attractive solution both for implementation as a logic circuit from discrete gates or as a fully integrated device from a conventional digital technology.

This approach has been applied with similar success to all gates with 12 and less inputs subject to the conditions $t \leqslant n/2$ and n/t is an integer. Their respective gate and gate input and output counts are summarised in **Table 1** and the complete Boolean implementation expressions of selected examples are given in Appendix 1. For low values of n and t (e.g. $n = 4$, $t = 2$) the number of sum-of-product terms is small and little or no improvement over the already simple solution provided by a conventional approach can be offered. However, as the values of n and t are increased and the number of sum-of-product terms increases it can be seen that the technique employed offers an increasingly efficient implementation. Since $t \leqslant n/2$ the number of SOP terms will always be greater than the corresponding number of POS terms. The terms themselves will, however, be proportionately smaller.* The SOP and POS expressions could, of course, be simplified by factorisation. This has not been attempted in every case since a fairly sophisticated computer program would be required for some of the larger circuits. Selected examples have, however, been examined in detail and have yielded the following results. In every case the number of gate delays is significantly greater than for the proposed technique. Typically for $n = 9$, $t = 3$ the factorised SOP and POS expressions yield 10 and 14 level circuits respectively as compared to 3. In many cases the gate and gate IP/OP's required remain significantly greater than for the proposed technique. Typically, for the above example these are 27 and 86, and 31 and 94 respectively as compared to 17 and 69. Factorisation techniques can often also be applied to the solutions obtained by the proposed technique. Small terms frequently occur in more than one expression and large terms can usually be split into factors which occur in more than one expression.

Table 1 Summary of gate requirements of threshold gate circuits

Threshold (t)	*No. of IPS* (n)	*No. of SOP Terms*	*No. of Gates*	*No. of Gate IP/OP's*
2	4	6	7	21
2	6	15	10	35
2	8	28	10	43
2	10	45	13	59
2	12	66	13	69
3	6	20	13	43
3	9	84	17	69
3	12	220	29	141
4	8	70	28	103
4	12	495	59	255
5	10	252	56	214
6	12	924	93	407

3. Extensions to the proposed technique

It is apparent that the frequent exhaustive searches required for execution of the algorithm described are only possible with the use of a computer. A DEC PDP 12 computer was used to obtain the results given here and the program was written in assembly language to provide the maximum possible speed of execution. Apart from the obvious disadvantage of requiring a computer to obtain solutions to any further problems, the execution time of the algorithm increases rapidly for larger circuits; this means that for a significantly larger circuit, the solution would be impossible to find due to the execution time of the algorithm even though a relatively simple solution may exist. Although the range of possible problems could be extended by improving the algorithm, employing a faster conventional computer and ultimately by reverting to parallel processing techniques the same problem would still eventually arise. To overcome this problem, an algorithm which arrives at each required expression without reverting to an exhaustive search of all those possible is required. This has not yet been achieved for the general case but in the special case of a threshold of two a technique has been devised and is discussed in Section 6.

4. Gate realisation when n/t is not an integer

Although solutions for the case where n/t is not an integer have not been attempted here, they are obviously possible to obtain using a similar technique, the only difference being that the expressions in this case would consist of unequal groups. Solutions can however be obtained from the existing solutions by the elimination of unwanted variables. These can either be replaced by logic $\emptyset$ in the final expressions or clamped to logic $\emptyset$ in the circuit. A further alternative exists in the case of $t > n/2$. It was hinted in the introduction in connection with error

*Frequently there will be more than one expression providing the same number of terms, the choice is then arbitrary and in this example the first one encountered is chosen.

*The product of the number of terms and the number of variables per term is identical for SOP and POS expressions for any n and t.

tolerant decoders that threshold gates have in fact two thresholds. One is exceeded when t or more of the inputs are in the logic 1 state causing the output to change from logic $\emptyset$ to logic 1. The other is exceeded when $(n - t + 1)$ or more inputs are in the logic $\emptyset$ state causing the output to change from logic 1 to logic $\emptyset$. Clearly if $t > n/2$ then $(n - t + 1) \leqslant n/2$. It can easily be shown by De Morgan's theorem that for any $t > n/2$ the solution can be obtained from that for a threshold of $(n - t + 1)$ by merely interchanging AND and OR signs. In general this will result in a more efficient implementation when these conditions apply.

5. Effect of non-unity input weightings

No mention has so far been made of the application of this technique to gates with non-unity input weightings which certainly represent an extremely important requirement. The technique can obviously be used to implement gates with any positive input weightings, provided they are integers, by initially including an appropriate number of additional inputs in the expressions. These can then either be allocated in the required numbers to the actual inputs in the final expressions or alternatively be connected to them in the final circuits. This approach, although adequate, does not appear to exploit the technique to its maximum advantage. Any increase in the weight of an input results in a simplification of the sum-of-products expression. Preliminary investigations have indicated that the technique, in all probability, can be successfully extended to take advantage of this.

Consider, for example, the case of $n = 12$, $t = 3$ where 9 of the inputs, ABCDEFGHJ, have unity weighting and 3, KLM, each have a weighting of 2. A solution in this case can be obtained intuitively from the unity weighted solution for $n = 9$, $t = 3$ given previously, i.e.

$$\begin{aligned} T = {} & (A + B + C)(D + E + F)(G + H + J) \\ & + (A + D + G)(B + E + H)(C + F + J) \\ & + (A + E + J)(B + F + G)(C + D + H) \\ & + (A + F + H)(B + D + J)(C + E + G) \\ & + [(A + B + C) + (D + E + F) + (G + H + J) + K] \\ & \quad [L + M] \\ & + [(A + B + C) + (D + E + F) + (G + H + J) + L] \\ & \quad [K + M] \ . \end{aligned}$$

Alternative implementations of similar efficiency are obviously also possible. It is anticipated that a detailed investigation into such possibilities will form an important area of future work.

6. Special cases

1. $t = 2$, $n = 2^r$ where r is an integer.

The solution in this case requires r expressions. Consider for example the case of 16 input variables denoted by ABCDEFGHJKLMNPQR. This will require 4 expressions. The input variables are divided into two, four, eight and sixteen equal subgroups in the manner shown below, the final subgroups having only one variable each. The four expressions are then formed by combining the alternate subgroups in each case, i.e.

$$\begin{aligned} T = {} & (A + B + C + D + E + F + G + H) \\ & \qquad (J + K + L + M + N + P + Q + R) \\ & + (A + B + C + D + J + K + L + M) \\ & \qquad (E + F + G + H + N + P + Q + R) \\ & + (A + B + E + F + J + K + N + P) \\ & \qquad (C + D + G + H + L + M + Q + R) \\ & + (A + C + E + G + J + L + N + Q) \\ & \qquad (B + D + F + H + K + M + P + R) \end{aligned}$$

ABCDEFGHJKLMNPQR

ABCDEFGH JKLMNPQR

ABCD EFGH JKLM NPQR

AB CD EF GH JK LM NP QR

A B C D E F G H J K L M N P Q R .

This is identical to the computer solution which would be obtained for this example. This approach can easily be applied to all cases which satisfy the above conditions.

2. $t = 2$, $n \neq 2^r$ where r is an integer.

The solution in this case will require r expressions where $2^r > n > 2^{r-1}$. The solution can be obtained by solving for $n = 2^r$ in the manner described above and then eliminating the unwanted variables from the expressions by assuming that they will always be in the logic $\emptyset$ state. Obviously the best solution is achieved by removing pairs of variables which appear in different groups in every expression (i.e. A and R, B and Q, C and P, etc. in the case of $n = 16$ given above) so that the symmetry of the expressions is preserved. An examination of the expressions for each of the required sum-of-product terms can then be used to eliminate any redundant variables. This usually only results in the elimination of a few variables from the final expression. The solutions obtained using this procedure are identical to the computer solutions obtained, except that the input variables are denoted by different letters.

7. Conclusions

Because of the problems encountered in applying a conventional Boolean implementation technique to the implementation of threshold gates and the undesirability of reverting to a new technology, an investigation into a more efficient means of implementing this type of gate from Boolean logic elements has been made. As a result, a technique whereby the required function is expressed as the sum of a number of product-of-sum terms has been proposed together with an algorithm for providing this expression in the general case. In addition, a simpler technique for the implementation of threshold gates with a threshold of two without the use of a computer has been described and the future possibility of extending the technique to handle non-unity input weightings is shown to appear promising. Results of all relevant cases up to and including 12 input variables are summarised and are shown, in general, to provide a highly efficient implementation without incurring the large propagation delays experienced in factorised SOP and POS expressions.

Hurst has predicted that, due to the problems of tolerancing, threshold gates with more than 7 input variables are unlikely to be built using 'threshold' technology. The proposed technique requires only 93 gates, most of which have only two inputs to implement a gate with 12 unity weighted inputs and a threshold of 6. This and larger gates could therefore easily be fabricated on a single IC chip from a conventional digital technology. The majority of the solutions obtained are of similar complexity to TTL MSI integrated circuits and would therefore prove even simpler to implement in MOS. The technique thus provides solutions equally attractive for implementation of threshold functions from discrete gates and for the fabrication of fully integrated devices from conventional technologies.

One of the attractions of threshold gates is that a single gate can be used to realise a large range of different Boolean logic functions (Hurst, 1970). This is also true of programmable read-only-memories; these, however, exhibit poorer propagation delays and are considerably costlier than conventional logic ICs as proposed here for the implementation of threshold gates.

Acknowledgement
The author would like to express his thanks to his project supervisors, Mr. H. A. Kemhadjian and Dr. D. W. Thomas, for their continual support and encouragement throughout this work and also wishes to acknowledge the financial support of the Science Research Council via an SRC Studentship.

Appendix 1 Examples of threshold gate designs

1.1. Threshold = 2
No. of Inputs = 6.
Input variables are denoted by ABCDEF.

$$T = (A + B + C)(D + E + F) + (A + B + D)(C + E + F) + (A + E)(B + F) .$$

1.2. Threshold = 3
No. of Inputs = 12.
Input variables are denoted by ABCDEFGHJKLM.

$$T = (A + B + C + D)(E + F + G + H)(J + K + L + M) + (A + B + E + F)(C + D + J + K)(G + H + L + M) + (A + B + L + M)(C + D + G + H)(E + F + J + K) + (A + C + E + G)(B + D + J + L)(F + H + K + M) + (A + C + K + M)(B + D + F + H)(E + G + J + L) + (A + B + E + J)(C + F + G + K)(D + H + L + M) + (A + E + H + L)(B + F + J + K)(C + D + G + M) .$$

References

AMODEI, J. J., WINDER, R. O., HAMPEL, D., and MAYHEW, T. R. (1967). An Integrated Threshold Gate, *Dig. Proc. 1967 International Solid-State Circuits Conference*, University of Pennsylvania.

HURST, S. L. (1971). *Threshold Logic*, Mills and Boon.

HURST, S. L. (1970). Universal Threshold Gates for 3-, 4- and 5-Variable Linearly Separable (Threshold) Functions, *Electronics Letters*, Vol. 6, pp. 324-326.

Book review

APL Congress 73, edited by P. Gjerløv, H. J. Helms and Johs Nielsen, 1973; 506 pages. (*North-Holland*, Dff 75.00.)

Since birth, APL has had its proponents and its opponents and, as in other areas, reactions generated in any discussion quickly become emotional to the extent of a love/hate relationship. Naturally, the book under review, which represents the proceedings of the APL Congress 1973 in Copenhagen, reports only the views and attitudes of lovers. But this reviewer was pleased to note that he is apparently not alone in developing a schizophrenic attitude to the language.

The conflict stems from the fact that, as demonstrated also by many papers in the book, APL is an extremely useful language when one wishes to get correct results in specific application areas. That is, it is a valuable tool for the researcher in almost any area and an aid whereby one may develop information processing tools that may subsequently be used effectively by the uninitiated. Pragmatically, APL has proved itself in many situations.

On the other hand, when APL is viewed from the point of view of the programming theorist or the software engineer, it may be held up as a model of what a programming language should *not* be. Richness of operators, a single machine-oriented control statement, outstanding interactive attitudes, are examples of language and system features that facilitate the creation of unintelligible, poorly structured and therefore, unmaintainable software. APL is just too good to be put into the hands of mere humans.

Perusal of *APL Conference 73* reveals that others share this uneasiness. In the first paper 'Program Writing, Rewriting and Style' for example, Phil Abrams in his references to APL pornography and to syntactic glue clearly outlines the traps one may so easily fall into in developing large APL programs. By the time it is completed the program will be unintelligible even to the author. And intelligibility of a program is perhaps its second most important attribute.

The most important property is, of course, 'correctness' and this topic is addressed in related papers by Feldbrugge and by Vervoort. But practical program verification makes demands on the programming language used, on programming style and on programming methodology. Papers in this volume discussing this issue suggest, correctly of course, that freely used APL *as such* is inadequate from this point of view and requires augmentation, particularly with user-oriented primitives for control structuring. Equally important—though the papers do not say this—is the fact that in practice, protocol limiting the use of language primitives and features would have to be adopted if confidence in the correctness of a proof is to be greater than confidence in the correctness of the program in the first place.

The proceedings contain some 67 papers, most of which are application oriented; many with exotic titles and even more exotic content. The breadth of coverage is remarkable and almost everyone should find at least one paper of interest, though papers will usually be understandable only to those who are already familiar with APL.

In general then this is an interesting book to browse through for those who know APL. It is not a book to buy. And we may look forward to even greater things from the (at the time of writing) forthcoming Second APL Conference in Pisa.

M. M. LEHMAN (London)

Field-effect Transistors in Integrated Circuits, by J. T. Wallmark and L. G. Carlstedt, 1974; 153 pages. (*Macmillan*, £4·50.)

This book is well written and well translated, well structured and well thought-out by authors who are truly authorities on the subject. It introduces the principles of field effect devices adequately and describes and demonstrates their use thoroughly in several relevant applications. It is also uncompromisingly dull; it has no real alternative to being so. The information with which it is packed is valuable and much of it is essential knowledge for computer hardware men if they are to be masters of the tools of their trade. But it is hard reading for anybody who is not an enthusiast about electronic devices for their own sake. It should provide a valuable library book for teaching and engineering organisations in the computer, field, since it contains the answers to many questions that computer engineers must want to know as the use of field effect devices becomes increasingly widespread. It is not a book for reading by a warm fireside with the TV on, after a hard day—but the wise hardware man should find time to read it.

B. S. WALKER (Reading)