Using GF2 Matrices to Simplify Boolean Logic

Peter M. Maurer, Dept. of Computer Science, Baylor University, Waco Texas.
Peter_Maurer@Baylor.edu

1. Introduction.

GF2 is another term for the integers modulo 2. It is a number field consisting of only of one and zero, but nevertheless possessing many of the same algebraic properties of number fields like the real and complex numbers. The AND function replaces multiplication and the XOR function replaces addition, but in other respects the algebra is the same as that of the real numbers. Polynomials over GF2 have been used extensively in coding theory for many decades. In our recent work on gate and function level simulation we have found it advantageous to use matrices over GF2 to transform multi-bit inputs of a function prior to simulation. The linear algebra of GF2 matrices is virtually identical to that for real and complex matrices, which means that the effect of using GF2 matrices is predictable and well-understood.

In our simulation work, we combine GF2 matrices with Boolean functions to make various non-symmetric functions appear to be symmetric to our simulation engine. This approach is useful because our simulation engine can compute the matrix virtually for free, and symmetric functions can be simulated much more efficiently than non-symmetric functions. We have long conjectured that there are many other potential applications for GF2 matrices in Design Automation.

GF2 matrices seem to be an ideal subject for a WACI, perhaps several WACIs ideas. In most contexts, the computation of the matrix is not free but it is relatively small. If the cost can be amortized in some way, then the benefits could be substantial. Indeed, the potential for new research is enormous. Virtually everything about using GF2 in Design Automation algorithms is an open problem. Nevertheless, there is an enormous body of mathematical research that is just waiting to be exploited.

In this paper we turn our attention to the problem of simplification of Boolean functions. This is only one potential application out of thousands, but we hope to show that in this one small area there are many interesting and potentially useful problems to be solved.

2. Quine-McCluskey with Hamming Distance >1

In our simulation work we dealt only with non-singular matrices, but for simplification both singular and non-singular matrices can be useful. We will begin with singular matrices, because these matrices allow us to express the Quine-McCluskey minimization algorithm in an elegant form and will allow us to proceed from there to a more powerful minimization algorithm. Let M be a singular $n\times n$ matrix. Because M is singular, there will be pairs of vectors $a\neq b$ where $M(a)=M(b)$. A Boolean function f is said to be *compatible* with M if $f(a)=f(b)$ whenever $M(a)=M(b)$. Surprisingly enough, if a function is compatible with one singular matrix, it will be compatible with many others as well. The important properties are rank, and the size and distribution of sets of rows that sum to zero. For simplicity, we confine our attention to $n\times n$ matrices of rank $n-1$ and containing a single set of rows that sum to zero. Matrices of rank $n-1$ are two-to-one mappings. Because any matrix with the same properties will do, we can choose a set of canonical matrices. The canonical 4×4 matrices are listed below.

$$M_1=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&0\end{pmatrix}\quad M_2=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&0\end{pmatrix}\quad M_3=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&1&1&0\end{pmatrix}\quad M_4=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\1&1&1&0\end{pmatrix}$$

All four matrices of rank 3, however, M_1 maps $abcd$ and $abcd'$ to the same value, while M_2 maps $abcd$ and $abc'd'$ to the same value, M_3 maps $abcd$ and $ab'c'd'$ to the same value, and M_4 maps $abcd$ and $a'b'c'd'$ to the same value.

The Quine-McCluskey algorithm is based on combining minterms that are compatible with M_1 and matrices similar to it. It is possible to expand the algorithm to include matrices similar to M_2 through M_4. M_1 compares vectors that are Hamming-Distance 1 apart, while M_2 through M_4 compare vectors that are Hamming-distance 2, 3, and 4 apart respectively. Note that applying M_2 (for example) is not the same as performing two steps of the

original Quine-McCluskey algorithm. In applying M_2 we can combine the two vectors 0011 and 0000 without combining 0010 and 0001. Unfortunately, there is a cost for using matrices M_2 through M_4. Although functions compatible with these matrices are indeed functions of three variables, the variables are not a, b, and c. If a function f is compatible with M_4 (for example), it is a function of the three variables $a+d$, $b+d$, and $c+d$. If these values are not already available, then logic must be added to compute them. It is may be necessary to minimize at least two functions with the same variables to realize any savings.

Virtually everything regarding this form of minimization is still an open problem. Some of the more interesting questions we could ask are: Under what conditions this technique will be advantageous? What is the best way to organize minterms? And is there any advantage to using non-canonical matrices? There are many questions that need to be answered.

3. Functions in Transformed Spaces

We can also use non-singular matrices to enhance function minimization. As in ordinary linear algebra, a non-singular linear transformation creates a new vector space which is of "a different shape" than the original space. We can use this reshaping to reduce the Hamming distance between minterms thus increasing the opportunities for conventional functional minimization. For example consider the 4-input symmetric Boolean functions. There are 32 such functions, some of which are the most complex functions to implement, in terms of the number of gates required. We have obtained minimal sum-of-products Boolean expressions for each of these functions. This required a total of 164 terms, 8 with one factor, 28 with two factors, 56 with 3 factors and 72 with 4 factors. If we transform our vector space using the following matrix, which requires three additional 2-input XOR gates, significant savings can be realized. The 32 new expressions contain a total of 90 terms, 17 with one factor, 33 with two factors, 32 with 3 factors and 8 with four factors. Applying the linear transformation not only cuts the number of terms nearly in half, but also significantly reduces the size of the terms.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The set of non-singular $n \times n$ matrices is too large to deal with on an ad-hoc basis. It much easier to deal with a subset of non-singular matrices called *single bit matrices.* A single-bit matrix is identical to the identity matrix, but has a single one off the main diagonal. A single-bit matrix represents the conditional inversion of one signal by another and can be implemented with a single 2-input XOR gate. The effect of two or more single-bit matrices can be combined by multiplying them together as shown above. In fact, it is easy to show that any non-singular matrix can be decomposed into a product of single-bit matrices, so concentrating exclusively on single-bit matrices is not restrictive.

Single-bit matrices can be used to rearrange a portion of a Karnaugh map. Each row of the matrix controls one specific portion of the map and each position within the row determines how the controlled area is rearranged. For example, in a 4-input map, row 1 of the matrix controls the last two rows of the map, and a 1 in column two causes these rows to be swapped. By carefully cataloging and using the effects of each single-bit matrix, a product matrix can be created that moves the ones of a function into their optimal positions.

In the Quine-McCluskey algorithm, single-bit matrices can be used to move minterms up and down the table. Matrices are applied when there are no remaining adjacent minterms (or other implicants). By selecting the proper row, one can maximize the number of minterms that will move, and by choosing the proper position within the row, one can maximize the number of new pairings will be are created. Unlike Karnaugh maps, which require an all-or-nothing approach, one can selectively apply different matrices to different minterms. Again, because the cost of the matrix is not zero, it will usually be necessary to use the same matrix more than once to realize any savings.

4. Conclusion

I hope that these few remarks have shown the potential of using GF2 matrices in Boolean function minimization. Obviously there are many more potential uses both in minimization and in other areas. Another potential area is using GF2 matrices in circuit synthesis where the cost of the matrix could be folded into the design process, making it essentially free. I hope that these remarks are enough to pique the interest of researchers to further explore this fascinating and potentially fruitful area.