Unit 20 R-M Applications and Minimization

- A function such as $Q = A \oplus B \oplus C \oplus D$ will only take logic 1 when an odd number of A, B, C and D have logic 1. The function therefore acts as a parity generator.
- An n input Sum-of-Products function requires 2^n sets of test inputs. An n input Reed-Muller circuit can be tested using $4 + n + 2n_e$ sets of test inputs
- Reed-Muller expressions are minimized by selecting either the uncomplemented or complemented form of each of the inputs so as to obtain a minimized expression.
- The Polarity Vector is used to define which form of the input is to be used in the Reed-Muller expression.
- The techniques for minimizing Reed-Muller expressions are the subject of ongoing research.

Before examining the methods available for the minimization of Reed-Muller expression, there is a more fundamental question to be answered. It is, why use the Reed-Muller form at all? The simple and only really practical answer to this question is that the Reed-Muller form for an expression and the resulting hardware implementation yields a function and hardware implementation which is fully testable with a finite (proportional to n) number of discrete tests.

Testability is not a problem for small logic systems for which all possible inputs can be applied during testing but for large systems having n inputs the number of possible inputs is 2^n. For a moderate system having 60 inputs this gives 10^{18} distinct input tests which must be applied for exhaustive testing. This could take a few years to carry out by which time the system might have developed a new fault due to age. Also remember that we do not simply test the design but we also have to test each machine that is built for compliance to the design. Therefore, when we use conventional design strategies we are constrained to use essentially untested computers! These untested computers are then used to design roads, bridges, buildings, cars, aircraft, to control

traffic systems, national economies etc. It is a tribute to modern design and manufacturing techniques that there are so few inaccurate computers in use. (Note that a computer that stops working is safe; a computer that gives an incorrect answer which masquerades as a correct answer is the really dangerous machine. Unless you are in a modern, fly-by-wire aircraft when the computer stops in which case you will need a parachute!)

The testability problem has been considered by Reddy who has pointed out that the testability of a machine should be built into the design specifications so that the machine only includes easily testable networks. The Reed-Muller form which is implemented as AND/XOR gate arrays has the required testability properties.

The types of faults which may be expected to occur in normal circuits are "stuck at 0" and "stuck at 1" (represented by s-a-0 and s-a-1) which may occur on either the inputs or the outputs of the AND gates and also logic output faults at the output of the XOR gates.

Some restrictions are placed on the numbers of faults for which tests can be developed. A fault at only one of the inputs to the AND gates can be detected. A single faulty output from the AND gates can be detected. A single faulty XOR gate can be detected. Testing for the occurrence of two simultaneous faults requires access to the internal circuitry of the arrays which is not always possible. The reason for imposing these limitations to single system faults is that it is possible for two faults in a binary logic system to cancel each other out.

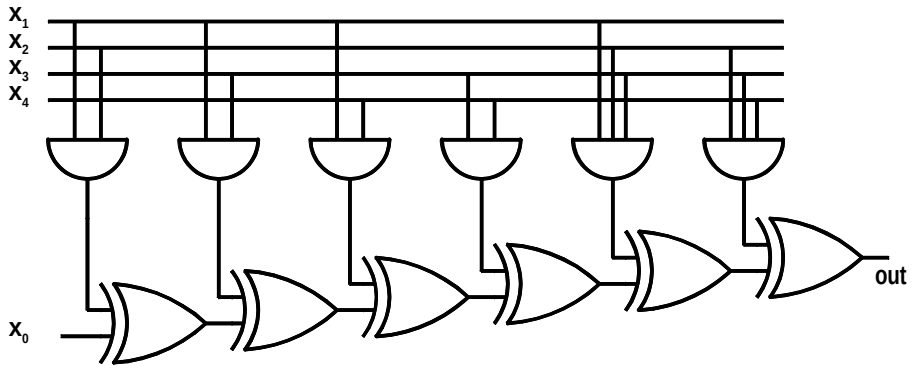

Figure 20.1: Circuit implementation of Reed-Muller AND/XOR function

If we take the gate array circuit shown in Figure 20.1 as a specific example we see that setting $x_1 = x_2 = x_3 = x_4 = 1$ causes the outputs of all of the AND gates to be 1's. Then if we switch the input x_0 from a 0 to a 1 the output

of the XOR cascade should give a value which depends on the accumulated parity of all of the inputs to the XOR cascade. In this example, the output should be a 1 when $x_0 = 1$ and a 0 when $x_0 = 0$. A test vector can then be written which details the values to be applied to the inputs and which also gives the expected output:

$$\begin{array}{cccccccc} & x_0 & x_1 & x_2 & x_3 & x_4 & \text{out} & \\ (& 0 & 1 & 1 & 1 & 1 & 0 &) \end{array}$$

This test vector tests all of the XOR gates but is not sufficient to fully test the cascade. There are three other test vectors which combine to form a test matrix for the cascade. It should be noted that a different cascade may have different parity and therefore the outputs may be complemented. If we remove the labels on the columns we then obtain the test matrix, $\mathbf{F_C}$, which has 4 rows of test vectors for locating faults in the XOR cascade:

$$\mathbf{F_C} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This test matrix tests each of the XOR gates in turn, working from left to right, by applying all four possible input combinations to the XOR gate and then using the output of the gate to form the input to the next XOR gate. The sequence is that the individual pairs of gate inputs and expected outputs which result from applying the test vector in the first row of the test matrix are:

$$(\ 0 \oplus 1 = 1\ ), \quad (\ 1 \oplus 1 = 0\ ), \quad (\ 0 \oplus 1 = 1\ ), \quad (\ 1 \oplus 1 = 0\ ), \quad \ldots$$

A s-a-0 or s-a-1 XOR gate input or output will prevent the propagation of at least one of the test vectors along the XOR cascade.

The application of the test matrix, $\mathbf{F_C}$, also serves to verify that none of the outputs of the AND gates are s-a-1 or s-a-0 and that the connections between the AND gate outputs and the XOR gate inputs are intact.

It is also necessary to verify that none of the inputs to the AND gates are s-a-1 or s-a-0. This is done by sequentially applying 1's to all AND gate inputs with the exception of a single AND gate and verifying that the correct output is obtained. For these tests, the x_0 input is not switched and can be assigned a "don't care" state. We will assign an arbitrary value of 0 to this x_0 input for the sake of definiteness in our predictions of expected output values. The test matrix, $\mathbf{F_A}$, which has n rows of test vectors for testing the

AND gate inputs of the circuit shown in Figure 20.1 is then:

$$\mathbf{F_A} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

The final column in this test matrix is, again, the output from the XOR cascade. In the case of the test input represented by the first row of the $\mathbf{F_A}$ test matrix, 001110, the inputs to the XOR gates are calculated by the following table in which the elements of the upper row are the AND gate outputs and the elements of the lower row are either the x_0 input or the outputs from the XOR gate. The final value in the lower row represents the expected output and is shown in the final column of the $\mathbf{F_A}$ test matrix.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & - \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

This $\mathbf{F_A}$ test matrix does not fully test all of the inputs to the AND gates since a fault on an input line which is connected to an even number of AND gates does not affect the parity of the inputs to the XOR gate cascade and therefore will not give a discrepancy between the expected and the actual output. For instance, in Figure 20.2, the input x_1 is applied to the first, second, third and fifth AND gates. Therefore a s-a-1 or a s-a-0 on this input will not affect the expected output. On the other hand, a fault in an input such as x_2 which connects to three AND gates will not cause a parity change and a change in the XOR gate cascade output whereas the expected result is that there will be a change in parity so such a fault in an input connected to an odd number of product terms will be detected.

We therefore require a test set which will detect faults in inputs which are connected to an even number of AND gates such as x_1, x_3 and x_4 in Figure 20.1.

What is needed is an input test set which will cause an odd number of AND gates or better still a single AND gate which feed into the XOR cascade to change state when the input is changed from a 0 to a 1. This will give an odd parity for the inputs to the XOR gates and force the final output from the XOR cascade to change state resulting in a changed output signal which can be compared to the expected signal.

In the example circuit in Figure 20.1, we take a minimum product term such as the x_1x_2 term which is formed by the first AND gate and assign the value $x_2 = 1$. The input x_1 is then assigned the two values $x_1 = 1$ and $x_1 = 0$ in turn with all of the other inputs $x_3 = x_4 = 0$. Only the first AND gate changes output as a result of x_1 changing state.

Therefore, if we assign an arbitrary value of $x_0 = 0$, we can test an input which connects to an even number of product terms or AND gates by using a test matrix such as:

$$\mathbf{F_{E1}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Similar test matrices are needed for each of the other inputs which also appear in an even number of product terms, so if there are n_e such input terms then there will be $2n_e$ tests associated with them.

The total number of tests which have to be carried out on a Reed-Muller AND/XOR type of gate array in order to test for correct functioning is therefore $4 + n + 2n_e$ with the one restriction that only single s-a-1 or s-a-0 type faults occur.

We therefore see that the number of tests required for testing AND/XOR gate arrays is a simple linear function of the number of primary inputs whereas the number on tests required to test an AND/OR Boolean type circuit is of the order of 2^n.

In the examples which we have discussed we have used the fixed polarity Reed-Muller form, that is the form in which the inputs appear only in the uncomplemented form as A, B, x_3, etc. We can also use some or all of the variables in the complemented form and then we have the mixed polarity Reed-Muller form. The only restriction is that an input variable can only appear in one of either the complemented form or the uncomlemented form and must not appear in both forms. We can keep track of which polarity is used for each input by use of a **Polarity Vector**, **k**. For the example discussed above, the polarity vector is $\mathbf{k} = (1, 1, 1, 1)$ because A, B, C, D were used in the uncomplemented form. If, instead, we had used A, $\overline{B}$, $\overline{C}$, D as primary inputs to the AND gates then the polarity vector would be $\mathbf{k} = (1, 0, 0, 1)$ in which case the Reed-Muller form of the function would be different while still yielding the same logical function. This method of specifying the polarity by use of a polarity vector is important because the minimization of the Reed-Muller form for a function is achieved by selecting the polarity which will give the simplest expression.

When the polarity vector contains zero elements, the inputs appear in the complemented form. This gives us a method of converting from the Reed-Muller form to the Generalized Reed-Muller form. All that has to be done is to make the substitutions for a variable using the identity $A = 1 \oplus \overline{A}$.

If we take the function which was used as a working example and which was illustrated in Figure 20.1:

$$f(x_{n-1}...x_0) = 1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_1x_4 \oplus x_3x_4 \oplus x_1x_2x_3 \oplus x_2x_3x_4$$

This expression has a polarity vector, $\mathbf{k} = (1, 1, 1, 1)$. Conversion to a polarity vector $\mathbf{k} = (1, 0, 0, 1)$ involves making the appropriate substitutions and gives:

$$\begin{aligned} f(x_{n-1}...x_0) &= x_0 \oplus x_1(1 \oplus \overline{x_2}) \oplus x_1(1 \oplus \overline{x_3}) \oplus x_1x_4 \oplus (1 \oplus \overline{x_3})x_4 \\ &\quad \oplus x_1(1 \oplus \overline{x_2})(1 \oplus \overline{x_3}) \oplus (1 \oplus \overline{x_2})(1 \oplus \overline{x_3})x_4 \\ &= x_0 \oplus x_1 \oplus x_1\overline{x_2} \oplus x_1 \oplus x_1\overline{x_3} \oplus x_1x_4 \oplus x_4 \oplus \overline{x_3}x_4 \\ &\quad \oplus x_1 \oplus x_1\overline{x_2} \oplus x_1\overline{x_3} \oplus x_1\overline{x_2x_3} \oplus x_4 \oplus \overline{x_2}x_4 \oplus \overline{x_3}x_4 \oplus \overline{x_2x_3}x_4 \\ &= x_0 \oplus (x_1 \oplus x_1 \oplus x_1) \oplus (x_4 \oplus x_4) \oplus (x_1\overline{x_2} \oplus x_1\overline{x_2}) \oplus (x_1\overline{x_3} \\ &\quad \oplus x_1\overline{x_3}) \oplus x_1x_4 \oplus \overline{x_2}x_4 \oplus (\overline{x_3}x_4 \oplus \overline{x_3}x_4) \oplus x_1\overline{x_2x_3} \oplus \overline{x_2x_3}x_4 \\ &= x_0 \oplus x_1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus x_1x_4 \oplus \overline{x_2}x_4 \oplus 0 \oplus x_1\overline{x_2x_3} \oplus \overline{x_2x_3}x_4 \\ &= x_0 \oplus x_1 \oplus 0 \oplus x_1x_4 \oplus \overline{x_2}x_4 \oplus x_1\overline{x_2x_3} \oplus \overline{x_2x_3}x_4 \end{aligned}$$

The equivalence of the two forms for this function can be verified by using a simple program to verify that the two forms give the same outputs for all possible inputs. It is possible to test for all inputs when there are only five variables but exhaustive testing would not be possible for systems with much larger numbers of inputs. A QuickBasic program for testing this system would be:

```
100 REM
INPUT x0, x1, x2, x3, x4
x0 = -x0: x1 = -x1: x2 = -x2: x3 = -x3: x4 = -x4
REM logic 1 is represented by -1 in QuickBasic
z = x0 XOR x1 XOR 0 XOR (x1 AND x4) XOR ((NOT x2) AND x4)
    XOR (x1 AND (NOT x2) AND (NOT x3)) XOR ((NOT x2)
    AND (NOT x3) AND x4)
p = x0 XOR (x1 AND x2) XOR (x1 AND x3) XOR (x1 AND x4)
    XOR (x3 AND x4) XOR (x1 AND x2 AND x3)
    XOR (x2 AND x3 AND x4)
PRINT z, p
GOTO 100
```

When different polarities, represented by different polarity vectors, are used it is found that some lead to expressions which are simpler than others in that fewer gates are required. The minimization problem then reduces to the problem of searching the set of alternative Generalized Reed-Muller

forms for the simplest form. Some techniques, based on the Karnaugh map approach, have been used to obtain this minimized form but these methods are only successful for small numbers of inputs.

Hand calculation of the Reed-Muller form for different polarity vectors is tedious and time consuming as can be seen by working through the example given above. Larger numbers of variables require rapidly increasing times. To obtain a minimum using the brute force approach, it is necessary to calculate the expression for all polarity vectors. The consequence is that the minimization problem becomes an exponential time problem.

What is needed is an analytic method or an algorithm which is suitable for implementation on a computer and which is capable of identifying the polarity vector which gives a minimized Generalized Reed-Muller form for the expression. The present state of the art in hardware development is that FPGAs (Field Programmable Gate Arrays) have been implemented in AND/XOR form and now have comparable speeds to the AND/OR versions. Unfortunately, the computerized minimization procedures for AND/XOR arrays do not yet match the capabilities of procedures such as SIS and ESPRESSO (Unit 17) and more theoretical work remains to be done.

The current situation is that algorithms have not yet been developed for identifying the polarity vector which specifies which of the possible 2^n Generalized Reed-Muller (GRM) forms gives the optimum minimized solution. When n is small it is feasible to calculate all of the GRM forms but for n greater than 16, corresponding to 16 primary inputs, the computational load starts to become excessive.

However, Tsai and Marek-Sadowska, Tran and also Fleisher, Tavel and Yeager have proposed algorithms and these will be now be presented as the present state of the art in Reed-Muller minimization and as an indication of lines of possible future development.

Representation of GRM functions. Tsai and Marek-Sadowska have developed a modification of the Binary Decision Diagram (BDD) representation which allows a Generalized Reed-Muller (GRM) function of eXclusive-OR cubes and the associated polarity vector to be represented in what is called a Functional Decision Diagram (FDD). The key to understanding the FDD representation is embodied in two rules:

1. If the variable is present in a cube (or term of the Reed-Muller representation) then the branch value (0 or 1) is the **same** as the value of the variable in the polarity vector.

2. If the variable is absent from a cube then the branch value is **opposite** to the value of the variable in the polarity vector.

Take, for example, the FDD used by Tsai and Marek-Sadowska (1994) and shown in Figure 20.2 (a).

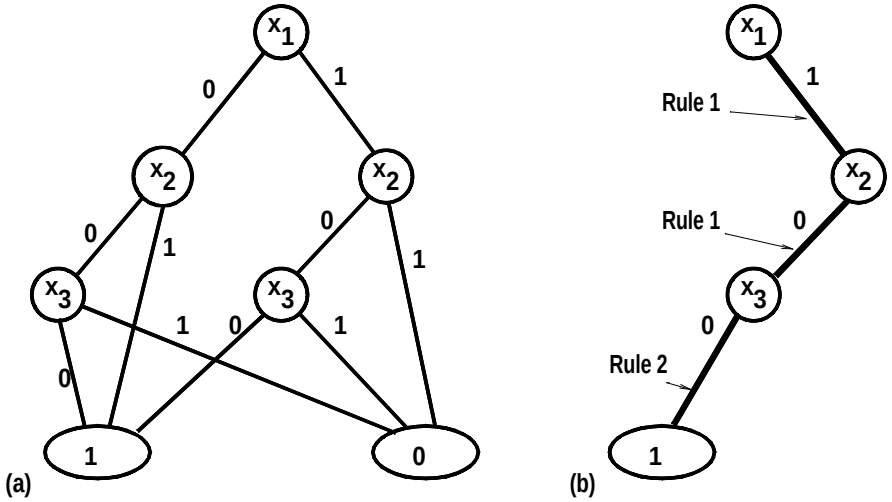

Figure 20.2: FDD for $f = \overline{x_2} \oplus 1 \oplus x_3 \oplus x_1\overline{x_2}$ and $V = (101)$

The polarity vector is $V = (101)$, that is the variables are x_1, $\overline{x_2}$ and x_3, and the Reed-Muller function is:

$$f = \overline{x_2} \oplus 1 \oplus x_3 \oplus x_1\overline{x_2}$$

Then the path through the FDD for the cube $x_1\overline{x_2}x_3$ is as shown in by the heavy lines in Figure 20.2 (b) with the indicated Rules applying at each branch.

Extraction of GRM expression. When the polarity vector is $V = (111)$ the first two rows of the Covering Matrix, $\mathbf{M_c}$, are given by a numerical index corresponding to the column number and below the index, the binary representation of that index number as shown in the segment of $\mathbf{M_c}$ shown below.

Index	0	1	2	3	4	5	6	7
Σm	5	4	7	6	1	0	3	2

When a different polarity vector is used the indices are the same but the vertices are obtained by performing a bit by bit addition without carry of the binary representation of the index and the 1's complememt of the polarity vector. For instance, when the index is 4 or 100 in binary and the polarity vector is $V = (010)$, the 1's complement is (101) and the the corresponding vertex is for index 4 calculated as follows:

1	0	1
1	0	0
0	0	1

In the covering matrix, $\mathbf{M_c}$, the other rows are obtained by inserting c's to denote that the cube covers the vertex or -'s to denote that the cube does not cover the vertex. In all covering matrices, for a given n, the pattern of c's and -'s is the same and is as shown in the matrix below. An extra row has been added to the header which shows the decimal equivalent of the binary boolean term. This allows the minterm belonging to a particular column to be identified easily when the matrix is used to generate a Generalized Reed-Muller form of the function.

	Index	0	1	2	3	4	5	6	7
	Vertices	101	100	111	110	001	000	011	010
	Σm terms	5	4	7	6	1	0	3	2
Cubes	1	c	c	c	c	c	c	c	c
	$\overline{x_3}$	-	c	-	c	-	c	-	c
	x_2	-	-	c	c	-	-	c	c
	$x_2\overline{x_3}$	-	-	-	c	-	-	-	c
	$\overline{x_1}$	-	-	-	-	c	c	c	c
	$\overline{x_1 x_3}$	-	-	-	-	-	c	-	c
	$\overline{x_1}x_2$	-	-	-	-	-	-	c	c
	$\overline{x_1}x_2\overline{x_3}$	-	-	-	-	-	-	-	c

In this example, if we want to find which vertices are covered by the cube, $\overline{x_1 x_3}$, locate the row for that cube and then move across the matrix to locate the c's so that we find that that particular cube covers the cubes (000) and (010).

The source vertex, which is in the first column, is covered only by the cube 1. The polarity vertex, which is in the right hand column, is covered by all of the cubes.

If a vertex is in the on-set of a GRM expression then it must be covered by an odd number of the XORed cubes of the expression.

If a vertex is in the off-set of a GRM expression then it must be covered by an even number of the XORed cubes of the expression.

The cube-vertex covering matrix can now be used to generate the GRM form of the function from a minterm list. Take as an example the function and polarity vector:

$$f(x_2, x_2, x_3) = \Sigma m(1, 3, 6) \quad \text{for} \quad V = (010)$$

start with the lowest index, 0, and examine the function to see if a minterm of the function is present in that column.
The minterm in the index 0 column is 5 which is not in the function so the cube 1 is absent from the GRM function.
The column for index 1 contains minterm 4 which is absent from the minterm function so that the cube $\overline{x_3}$ is absent.
The column for index 2 contains minterm 7 which is absent from the minterm function so that the cube x_2 is absent.
The column for index 3 contains the minterm 6 which is present in the minterm function so the cube $x_2\overline{x_3}$ is present in the GRM function.
The column for index 4 contains the minterm 1 which is present in the minterm function so the cube $\overline{x_1}$ is present in the GRM function.
The column for index 5 contains the minterm 0 which is absent from the minterm expression but $\overline{x_1}$ is already present in this column so that we must include the cube $\overline{x_1}\overline{x_3}$ in the GRM function so as to have an even number of cubes in the off-set of of the XOR expression.
The column for index 6 contains the minterm 3 which is present in the function but the cube $\overline{x_1}$ is already present in the column so the cube $\overline{x_1}x_2$ is absent so as to give an odd number of terms in the XOR expression.
The column for index 7 contains the minterm 2 which is not present in the function so we must have an even number of cubes in the XOR expression. The cubes $x_2\overline{x_3}$, $\overline{x_1}$ and $\overline{x_1}\overline{x_3}$ are already present so we must include the cube $\overline{x_1}x_2\overline{x_3}$ in order to get an even number of cubes in the expression

We therefore now have the equivalent functions:

$$f = \Sigma m(1,3,6) = x_2\overline{x_3} \oplus \overline{x_1} \oplus \overline{x_1}\overline{x_3} \oplus \overline{x_1}x_2\overline{x_3} \quad \text{for} \quad V = (010)$$

Identification of optimum polarity. Tsai and Marek-Sadowska propose a heuristic algorithm for finding the optimum polarity which is based on the conjecture that if there exists a point in n dimensional Boolean space that is equally distant (Hamming distance) to all k vertices in the on-set of f and the distance is a minimum, then that point is the best polarity for the GRM form of the function. For instance, the Hamming distance from each of the Boolean terms in $f(x_1 \ldots x_5) = \Sigma m(0,6,15)$ to the point (00110) is 2 and therefore the best polarity vector is $V = (00110)$.

It will not be possible to obtain a center for the vertices in Boolean space for all cases and it is then necessary to make a best choice. If in the on-set of a function, a particular variable, x_i, is examined and the number of vertices containing 1's and 0's are counted, then the value of the polarity for that variable is whichever is the greater. If there are equal number of 1's and 0's then Tsai and Marek-Sadowska found that no great difference results from assigning 1s to the polarity vector for that variable.

As an example, the function specified by the minterm $f = \Sigma m(1, 3, 6, 7)$ is used. The variables appear in the frequencies shown in the table and the polarity vector is calculated accordingly:

	minterm	x_1	x_2	x_3
	1	0	0	1
	3	0	1	1
	6	1	1	0
	7	1	1	1
Frequency of 1's		2	3	3
Frequency of 0's		2	1	1
Polarity	1/0	1	1	

With this procedure, a good polarity vector for $f = \Sigma m(1, 3, 6, 7)$ is found to be $V = (111)$

20.1 References

Fleisher H, Tavel M and Yeager J. (1987), *A computer algorithm for minimizing Reed-Muller canonical forms*, IEEE Trans on Computers. C-36, (2), 247-250.

Green, David (1986), *Modern Logic Design*, Addison-Wesley

Reddy, Sudhakar M, (1972), *Easily testable realizations for logic functions*, IEEE Transactions on Computers, C-21 (11), 1183-1188.

Tsai C C, Marek-Sadowska M (1994), *Minimization of fixed-polarity AND/XOR canonical networks*, IEE Proc.-Comput. Digit. Tech., 141,(6), 369-374.

Tsai C C, Marek-Sadowska M (1996), *Generalized Reed-Muller forms as a tool to detect symmetries*, IEEE Trans. on Computers, 45, (1), 33-40.

20.2 Problems

20.1 Trace the path through the FDD shown in Figure 20.3 for each of the terms of the function $f = \overline{x_2} \oplus 1 \oplus x_3 \oplus x_1\overline{x_2}$ when the polarity vector is $V = (101)$

20.2 Obtain the Reed-Muller function which is represented by the FDD shown in Figure 20.4 when the polarity vector is $V = (111)$.

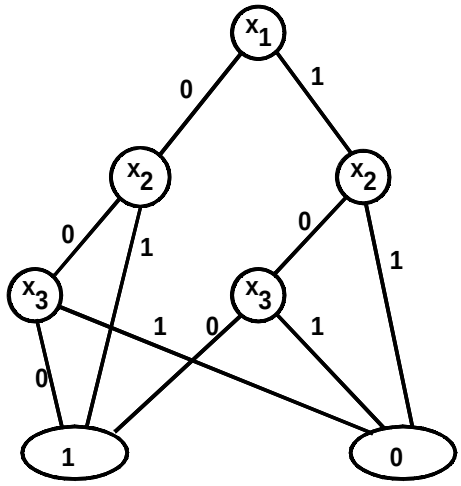

Figure 20.3: Obtain the RM function.

20.3 Obtain the FDD for the Reed-Muller function:

$$f(x_2, x_1, x_0) = x_0 \oplus x_2 \oplus x_1 x_0 \oplus x_2 x_1 \oplus x_2 x_1 x_0$$

when the polarity vector is $V = (111)$.

20.4 Calculate a suitable polarity vector for a GRM representation of the function:

$$f = \Sigma m(0, 2, 4, 6, 7)$$

20.5 Write a C or QuickBasic program which has as its input a polarity vector and the term of a Reed-Muller expression and which generates the Reed-Muller form of the expression corresponding to another polarity vector which is to be specified by the user.