Unit 18 Binary Decision Diagrams.

- A Binary Decision Diagram (BDD) is a directed graph which is traversed from an entry node to a terminal node.
- A BDD can be generated from:
  - A truth table.
  - A Boolean logic function.
  - A Karnaugh map.
  - An *If...then...else.* program.
- The optimum choice of ordering of variables in a BDD is an NP-hard problem.
- Methods exist for obtaining a good ordering of variables.
- Methods exist for simplifying BDDs.

In digital logic systems, a variable can have one of two values, either 0 or 1. As each new input variable is added, the number of possible configurations of the system input state increases by a factor of 2.

In the Boolean algebra approach to logic function computation of a function, Q, the full expression for Q is computed by computation of each of the terms of the Boolean expression which are then combined to give the final output value. There is an alternative representation for functions and an associated computation procedure called the Binary Decision Diagram (BDD) method or the Ordered Binary Decision Diagram (ODBB) method. In this representation, the values of the variables are tested in sequence and a route is traced through a tree structure from a root node to an output node. Some authors use the term vertex instead of node. Figure 18.1 shows binary decision diagrams for systems with (a) a single variable and (b) two variables.

In these diagrams, we will follow the convention whereby the node is labeled with the variable examined and there are exactly two exit paths from a node except for a terminal node. For clarity in the diagrams, we will

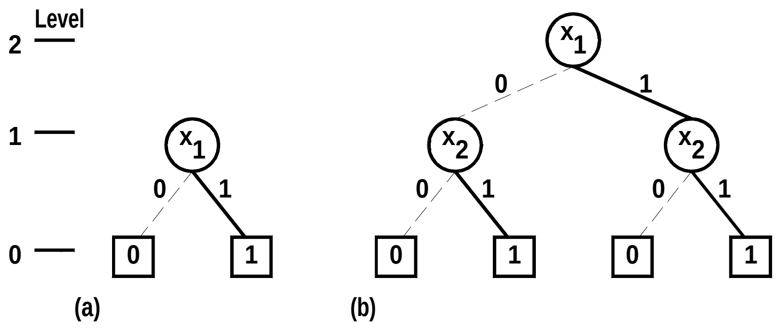

Figure 18.1: .

adopt the convention that the exit edge from a node for logic 0 is indicated with a dashed line and the exit edge for logic 1 is indicated with a full line. The 0 and 1 beside the exit edges shown in Figure 18.1 will be omitted in subsequent figures.

Nodes which represent variables are shown as circles. Terminal nodes are represented by squares. The variables are represented by x_1, x_2,... x_n. Since we anticipate extending the system to apply to applications having greater than 26 inputs we label the variables with subscripted x_n rather than the capital letters A, B, etc..

The level within a BDD is indicated from 0 to n, starting with 0 at the output level. Note that the levels in a BDD are in numerical sequence but that the variables do not necessarily appear in the BDD in numerical sequence. An n variable BDD will in general then have $n + 1$ levels. each root and internal node will have 2 outputs which we label 0 and 1 according to the value of the variable but we may not always have the 0 output from a node going to the left as shown. At the output (level 0) the value of the output may not be the same as the binary output of the previous node. This is shown in Figure 18.2 (a) and (b) which represent the functions $Q = x_1$ and $Q = \overline{x_1}$.

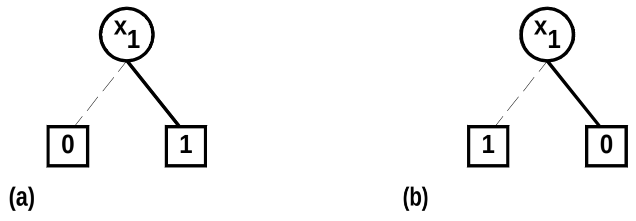

Figure 18.2: (a) $Q = x_1$ and (b) $Q = \overline{x_1}$. .

Another example is the exclusive OR function which is shown in the BDD in Figure 18.3. The symbol $\oplus$ is usually used to represent the Boolean exclusive OR operation.

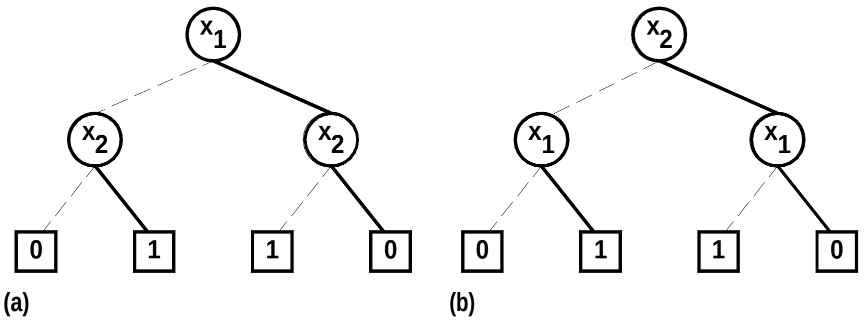

Figure 18.3: (a) $Q = x_1 \oplus x_2$ and (b) $Q = x_2 \oplus x_1$.

In Figure 18.3 (a) we first examine the variable x_1 at level 2 and then the variable x_2 at level 1 with the output at level 0. In Figure 18.3 (b) we first examine the variable x_2 and then the variable x_1 with the output again at level 0. Note also that the output values at level 0 are not always the same as the value of the output from the previous node.

In using these BDDs, each of the variables is examined in turn and the appropriate route is taken at the exit from the node corresponding to that variable. You should note that, while we have shown a particular variable as occurring only at one level in the BDD, this is not always the case. Examples which you may meet in the literature may have different variables located at different levels in different branches.

A specific path down through a BDD is a representation of a specific row of a truth table for which each of the variables have specific values of eother 0 or 1. The complete BDD is a representation of the complete truth table in which the variables in descending order are in the same order as the variables in the columns of the truth table. This is shown in the these two truth tables. They represent the same system but appear different because the variable orderin the columns is changed. In the resulting BDDs x_1 is at the top level in the first BDD and x_2 is at the top in the second

x_1	x_2	Q
0	0	0
0	1	1
1	0	0
1	1	0

x_2	x_1	Q
0	0	0
0	1	0
1	0	1
1	1	0

An alternative description of the BDD representation is to say that it is a diagrammatic representation of a sequence of *If...then...else* logical statements. For example, suppose we have a Boolean expression:

$$Q = x_1(\overline{x_2}x_3 + \overline{x_3}x_4)$$

this expression can be calculated in stages as follows(see Silva and David):

$$\begin{aligned} A &= \overline{x_2}x_3 \\ B &= \overline{x_3}x_4 \\ C &= A + B \\ Q &= x_1 C \end{aligned}$$

The *If...then...else* program statements for the same problem would be:

1	*If*	$x_1 = 1$	*then*	2	*else*	6.
2	*If*	$x_3 = 1$	*then*	3	*else*	4.
3	*If*	$x_2 = 1$	*then*	6	*else*	5.
4	*If*	$x_4 = 1$	*then*	5	*else*	6.
5		$Q = 1.$				
6		$Q = 0.$				

In computing the Boolean function, all of the elements of the function are computed before the answer Q becomes available. In the *If...then...else* structure, some routes to the output (e.g for $x_1 = 0$) are much shorter and can be computed more quickly. The saving is not significant in this problem but can be of significance in large problems, of say 64 or more variables, when the expression has to be computed in frequently called subroutines or is computed for use in fast, real time applications. The time saving can then be significant.

The BDD shown in Figure 18.4 is a representation of this *If...then...else* structure and illustrates some of the simplification that can be achieved relative to a full binary decision diagram which contains all possible routes through the diagram.

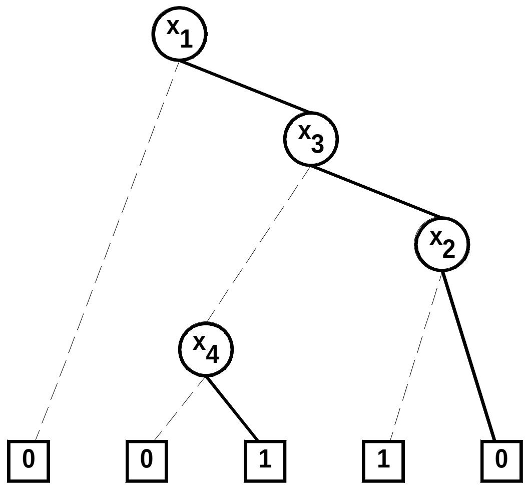

Figure 18.4: .

The truth table for this system is shown below:

x_1	x_2	x_3	x_4	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

and the truth table reduces to the Karnaugh map

	$\overline{x_1}.\overline{x_2}$	$\overline{x_1}.x_2$	$x_1.x_2$	$x_1.\overline{x_2}$
$\overline{x_3}.\overline{x_4}$	0	0	0	0
$\overline{x_3}.x_4$	0	0	1	1
$x_3.x_4$	0	0	0	1
$x_3.\overline{x_4}$	0	0	0	1

This BDD might represent the real world situation where: If the ignition switch in a car is ON (x_1), the driver's seat is occupied (x_3) and the seat belt is not fastened (x_2) then a warning buzzer will sound OR if the ignition switch is ON (x_1), the driver's seat is occupied (x_3) and the car door is open (x_4) then a warning buzzer will sound.

In this simple introductory example we have shown how the source information for the construction of a binary decision diagram can be any one of; a Boolean function, an *If...then...else* program, a truth table, a Karnaugh map or a car safety problem. We will now discuss the properties of, the construction of, the simplification of and finally the uses of binary decision diagrams in a more formal way.

While we have just shown how BDDs can be constructed from source information contained in truth tables, logic functions or Karnaugh maps, these representations need never be constructed and instead the BDD can be the primary and only representation of the problem. Indeed, because of the sequential nature of the BDD representation, the BDD can incorporate features which can not be represented in the truth table or logic functions.

It should, however, be pointed out that while the BDD, like the Karnaugh map, is suitable for visual interpretation by a human user, the formal mathematical or computer program representation of a BDD is neither straightforward nor trivial.

A considerable volume of theoretical work has been carried out which justifies the procedures which are described in this treatment of BDDs. This work is contained in the research papers of Silva, Bryant, Arborethy and others (see References) and is in the form of formal definitions, theorems, lemmas. This formal theoretical foundation has been incorporated in this discussion without detailed reference or attribution beyond the research papers referenced at the end of the unit. It is suggested that the reader obtain an overall appreciation of the operation and use of the BDD method before studying the grounding theorems of the method.

Properties of Binary Decision Diagrams. A full binary tree representation of a system having n inputs will have $2^n - 1$ test nodes and 2^n terminal nodes. In traversing the tree from the entry node to any terminal node, a total of n tests will be carried out and a total of n edges traversed.

For a large system, say 64 inputs, this gives an impossibly large system, containing $\approx 4 \times 10^{19}$ nodes, and the main aim of the techniques used in the BDD method is to minimize or reduce the size of this full binary tree.

In general, the problem of minimizing the binary decision diagram belongs to that class of problem characterized as NP-hard. The archetype of such problems is the traveling salesman problem in which a salesman is to visit each of n cities once only and return to his home at the end of the sales trip. The cities are separated by differing distances and the aim is to minimize the distance traveled. It is easily seen that on leaving home the salesman has n choices for the first city to be visited. He has $n-1$ possible second destination choices, $n-2$ third destination choices etc. which gives factorial n or $n!$ possible routes. For large values of n a very useful approximation to $n!$ is to use Stirling's formula $n! \approx n^n e^{-n}\sqrt{2\pi n}$. For a small sales trip of say 50 cities most hand held calculators have a factorial function which will give $50! = 3 \times 10^{64}$ possible routes. It is thus not possible to calculate all possible route distances and choose the shortest even for this moderate sales trip. As can be seen from the approximation for $n!$ given by Stirling's formula this problem has a number of routes which is not a simple polynomial function of n and is then termed a NP-hard (non polynomial) problem. The rapidly increasing number of possible routes and consequent increase in required computing time means that it is not possible to calculate an exact minimum route. It is, however, possible to calculate a near optimal route.

The BDD problem is also a NP-hard problem because the BDD is minimized for either the average path length or the number of tree nodes visited by choosing to visit the nodes in an optimum order. We will therefore have to consider methods for obtaining a good (optimal) BDD as distinct from the incalculable optimum BDD.

Variable ordering in BDDs. It has been found that one of the most significant simplifications that can be achieved in constructing a BDD is obtained by making an optimal selection of the order in which the variables appear in the BDD in going from level n to level 0. This is well illustrated in problem 18.3 in the problems at the end of the unit.

The computing time required to find an optimal ordering is not insignificant. Two methods have been compared by Friedman and Supowit which require computing times proportional to $n^3 3^n$ and $n!2^n$ where n is the number of variables. For $n = 12$ the times to compute an optimal ordering are proportional to 8×10^7 and 2×10^{12}, a ratio of 1:25000 so a good choice of method for selecting the ordering is important.

Choice of ordering does not have any effect on the correctness of the BDD. The main aim is to avoid orderings which cause exponential growth in the size of the BDD as the number of variables, n, increases.

Rather than search for optimal ordering, we present an approach which is based on the work of Silva and David and of Bryant which gives good ordering except in the most pathologically difficult problems.

The method of Silva and David uses the Boolean identity (T4 of Unit 9):

$$Q = A + B + C + D + \ldots = A + (B + C + D + \ldots)\overline{A}$$

(If $A = 1$ then $Q = 1$, if $A = 0$ then $\overline{A} = 1$ and $Q = 1$ if and only if one or more of B, C, $D, \ldots = 1$). The other identity which is used is from De Morgan whereby $\overline{x_1 x_2} = \overline{x_1} + \overline{x_2}$).

In the algorithm given by Silva and David for the construction of a good BDD from a Boolean expression there are essentially six steps. A slightly simplified version of the algorithm is presented here in the form of a worked example.

Step 1. Obtain the Boolean expression in a minimal sum of products:

$$Q = x_1x_2 + x_3x_4 + \overline{x_1}\,\overline{x_5}$$

Step 2. Order the prime implicant terms so that the terms have increasing numbers of literals with the terms having the greatest frequency of occurrence of a literal placed first. Then:

$$Q = x_1x_2 + \overline{x_1}\,\overline{x_5} + x_3x_4$$

Step 3. Use the Boolean identity $A + B + C = A + (B + C)\overline{A}$ to obtain:

$$\begin{aligned} Q &= x_1x_2 + \overline{x_1}\,\overline{x_5} + x_3x_4 \\ &= x_1x_2 + (\overline{x_1}\,\overline{x_5} + x_3x_4)\overline{x_1x_2} \\ &= x_1x_2 + (\overline{x_1}\,\overline{x_5} + x_3x_4)(\overline{x_1} + \overline{x_2}) \\ &= x_1x_2 + \overline{x_1}\,\overline{x_5} + \overline{x_1}x_3x_4 + \overline{x_1}\,\overline{x_2}\,\overline{x_5} + \overline{x_2}x_3x_4 \\ &= x_1x_2 + \overline{x_1}\,\overline{x_5}(1 + \overline{x_2}) + \overline{x_1}x_3x_4 + \overline{x_2}x_3x_4 \\ &= x_1x_2 + \overline{x_1}\,\overline{x_5} + \overline{x_1}x_3x_4 + \overline{x_2}x_3x_4 \\ &= x_1x_2 + H_0 \\ \text{where} \quad H_0 &= \overline{x_1}\,\overline{x_5} + \overline{x_1}x_3x_4 + \overline{x_2}x_3x_4 \end{aligned}$$

Step 4. Now repeat step 3 but operating on the function H_0. This will eventually give the function in the form:

$$\begin{aligned} Q &= x_1x_2 + \overline{x_1}\,\overline{x_2} + \overline{x_1}x_3x_4x_5 + x_1\overline{x_2}x_3x_4 \\ &= x_1(x_2 + \overline{x_2}x_3x_4) + \overline{x_1}(\overline{x_5} + x_3x_4x_5) \end{aligned}$$

The function Q is now in two parts containing x_1 or $\overline{x_1}$ so we can use x_1 as the variable for the entry node of the BDD. When $x_1 = 1$ the sub branch contains the function $F = x_2 + \overline{x_2}x_3x_4$ and when $x_1 = 0$ the sub branch contains the function $K = \overline{x_5} + x_3x_4x_5$

Step 5. Now repeat steps 2, 3 and 4, applied to each of the sub branch functions F and K. At this stage, two variants of the algorithm might be considered depending on whether or not we require that the variables appear in the same level in all of the sub branches of the BDD.

Step 6. Construct the BDD. For the example we have obtained the function in the form:

$$\begin{aligned} Q &= x_1(x_2 + \overline{x_2}x_3x_4) + \overline{x_1}(\overline{x_5} + x_3x_4x_5) \\ &= x_1(x_2 + \overline{x_2}(x_3(x_4))) + \overline{x_1}(\overline{x_5} + x_5(x_3(x_4))) \end{aligned}$$

which, in this example, then gives the BDD shown in Figure 18.5.

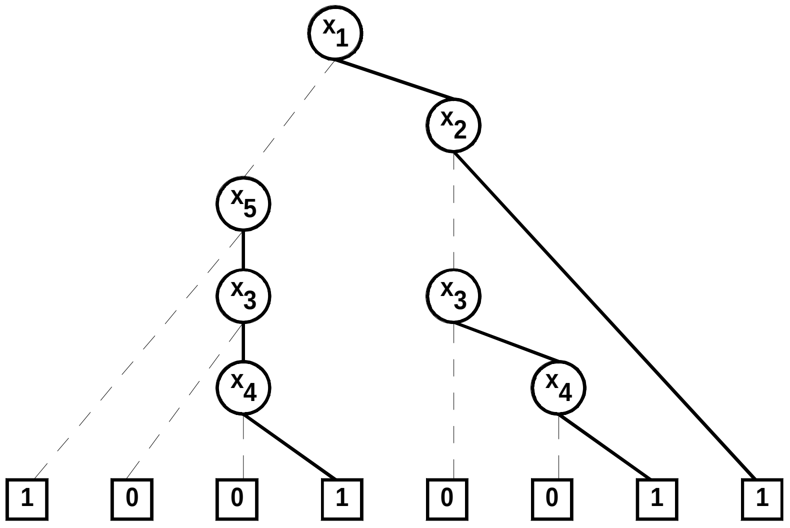

Figure 18.5: BDD for function $Q = x_1x_2 + x_3x_4 + \overline{x_1}, \overline{x_5}$.

Transfomation Rules for Graphical Simplification of BDDs. Bryant has formalized the set of transformation rules which can be applied to a BDD which do not alter the function represented but which manipulate the graph so as to obtain a maximally reduced graph.

The three rules are:

- **Remove duplicate terminals.** If a graph has duplicate final nodes, then these can be combined. Thus, if a representation has only the outcomes 0 and 1 there are then only two output nodes. More generally, if a function of n variables has m distinct outputs then it is possible to reduce the number of outputs nodes from 2^n to m. This is shown in Figure 18.6.

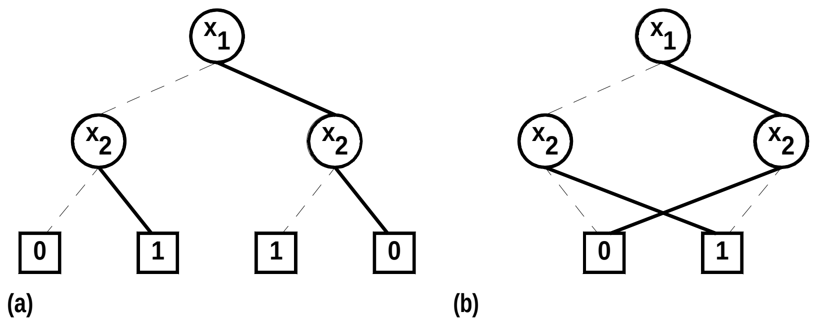

Figure 18.6: Remove Duplicate terminal nodes. XOR function

- **Remove Duplicate Nonterminal nodes.** If two nodes test the same variable, are reached by the same outcome of a previous node and have 0 and 1 outputs going to the same output nodes, then these nodes can be combined. This is shown in Figure 18.7.

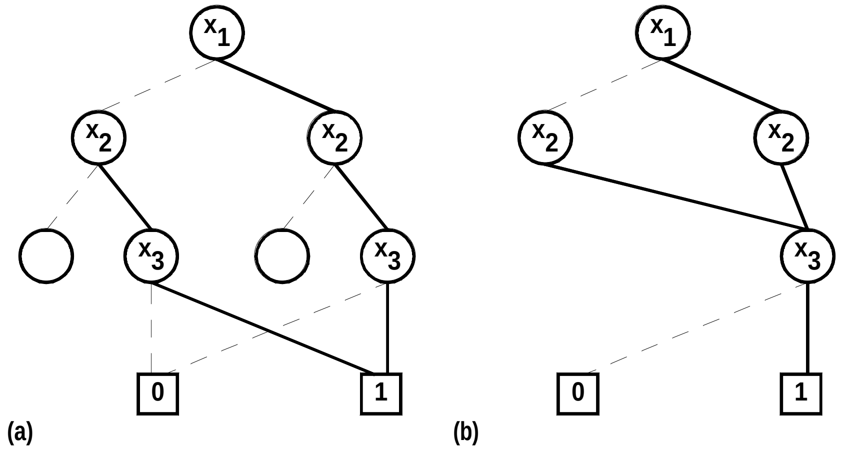

Figure 18.7: Duplicate nonterminal nodes. The x_3 nodes shown have been combined. For clarity other nodes have been omitted from the diagram.

- **Remove redundant tests.** If both the 0 and 1 outputs from a node go to the same next node then that node can be eliminated. This is shown in Figure 18.8.

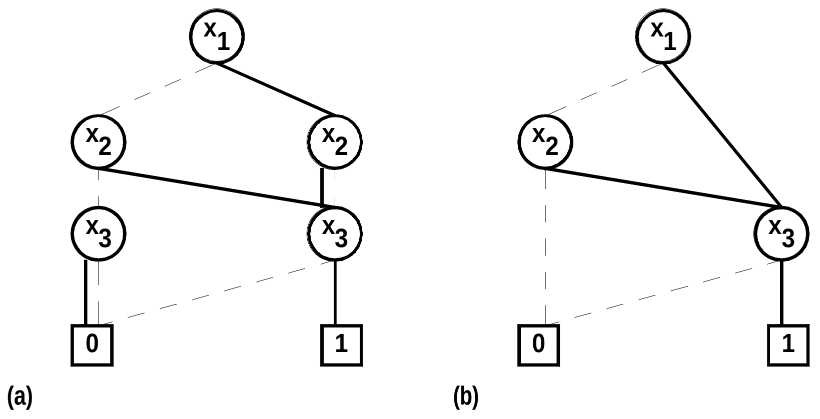

Figure 18.8: Removal of the redundant tests on x_2 and on x_3.

18.1 References

Bryant Randal E., *Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams*, ACM Computing Surveys, 24, (3), 293-318, 1992

Bryant Randal E., *Graph Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, C-35, (8), 677-691, 1986

Knuth Donald, *The Art of Computer Programming, Vol I.* Addison-Wesley 3rd Ed 1997

Aborhey, S., *Binary Decision Graph Reduction*, IEE Proc. 136, Pt. E, (4), 277-283, 1989

Lee C.Y. (1959), *Representation of switching circuits by binary decision programs*, Bell System Tech. J. 38(4) 985-999

Akers SR (1978), *Binary Decision Diagrams*, IEEE Trans Comp. C-27(6) 509-516

Friedman, S.J. Supowit, K.J. (1990), *Finding the optimal variable ordering for binary decision diagrams* , IEEE Trans Comp 39, 710-713

Silva M. and David R. (1985), *Binary-decision graphs for implementation of Boolean functions* , IEE Proceedings 132, Pt E, No 3, 175-185

Brace-Rudell-Bryant BDD package. Available by anon. FTP:
Host: n3.sp.cs.cmu.edu
Directory: /usr/cosmos/ftp
File: bdd.4.2.tar.Z

Dave Long's package. Distributed by Ed Clarke's group.
http://www.cs.cmu.edu/ modelcheck

18.2 Problems

18.1 Two three bit binary numbers, x and y, consist of the bits x_1, x_2, x_3 and y_1, y_2, y_3 where the x_1 and y_1 bits are the most significant bits. The two numbers are to be compared and a logic 1 output obtained whenever $x > y$. Construct the set of *If...then...else* tests which will test for the inequality and then implement these tests in a Binary Decision Diagram.

18.2 Construct the BDD for a 2 bit comparator which compares two two bit binary numbers, x and y and which has three outputs, Q_1, Q_2 and Q_3 corresponding to $x > y$, $x = y$ and $x < y$.

18.3 Draw the BDD diagram for the Boolean function:

$$Q = x_1(\overline{x_2}x_3 + \overline{x_3}x_4)$$

taking the variables in the order x_1, x_2, x_3, x_4. Compare the diagram with that shown in Figure 18.4 for the same problem but with the variables taken in the order x_1, x_3, x_2, x_4.