VLSI Logic Design Automation
Quine-McCluskey Logic Minimization Algorithm

Shashank Rao, Kewal Raul and Ritu Agarwal
Stony Brook University

***Abstract*— The Quine McCluskey algorithm is a method used for minimization of Boolean functions.It is a tabular method based on the concept of prime implicants. The paper presents the implementation of Quine McCluskey logic minimization algorithm in Java, discussing the performance of code and its resulting output.**

I. INTRODUCTION

Digital gates are basic component of any digital circuit. Boolean logic is the basic concept that underlies all modern electronic digital systems.The complexity of digital logic gates to implement a Boolean function is directly related to the complexity of algebraic expression. An increase in the number of variables results in an increase of complexity.It is preferable to have the most simplified form of the algebraic expression.The process of simplifying the algebraic expression of a boolean function is called minimization. Minimization is important since it reduces the cost and complexity of the associated circuit. Minimization is hence important to find the most economic equivalent representation of a boolean function.Karnaugh map(K-map) and Quine McCluskey(QM) methods are well known methods to simplify Boolean expression.

Karnaugh proposed a technique for simplifying Boolean expressions using an elegant visual technique, which is actually a modified truth table intended to allow minimal sum-of products (SOP) and product-of-sums (POS) expressions to be obtained. The Karnaugh Map (K-Map) based technique becomes complex and breaks down beyond six variables. Quine and McCluskey proposed an algorithmic based technique for simplifying Boolean logic functions. The Quine McCluskey (QM) method is a computer-based technique for Boolean function simplification and has mainly two advantages over the K-Map method. Firstly, it is systematic for producing a minimal function that is less dependent on visual patterns. Secondly, it is a viable scheme for handling a large number of variables.

A. Example

Minimizing a function:
$f(A, B, C, D) = \sum m(4, 8, 10, 11, 12, 15) + d(9, 14)$

This expression says that the output function f will be 1 for the minterms 4,8,10,11,12 and 15 (denoted by the 'm' term).It also denotes don't care about the output for 9 and 14 combinations (denoted by the 'd' term).

TABLE I: STEP 1

Group	Minterm	a	b	c	d
1	4	0	1	0	0
	8	1	0	0	0
2	9*	1	0	0	1
	10	1	0	1	0
	12	1	1	0	0
3	11	1	0	1	1
	14*	1	1	1	0
4	15	1	1	1	1

TABLE II: STEP 2

Group	Minterm	a	b	c	d
1	4,12	-	1	0	0
	8,10	1	0	-	0
	8,12	1	-	0	0
	8,9	1	0	0	-
2	10,11	1	0	1	-
	10,14*	1	-	0	-
	12,14*	1	1	-	0
	9*,11	1	0	-	1
3	11,15	1	-	1	1
	14*,15	1	1	1	-

STEP 1: Arranging the given minterms in an ascending order and making groups based on the number of ones present in their binary representations.

STEP 2: Comparing the minterms present in successive groups. If there is a change in only one-bit position, then pair those two minterms & Place symbol - in the differed bit position keeping the remaining bits as it is.

STEP 3: Repeat with newly formed terms till we get all prime implicants.

STEP 4: Formulating the prime implicant table. It consists of set of rows and columns. Prime implicants are placed row wise and min terms are placed column wise.x is placed in the cells corresponding to the min terms that are covered by each prime implicant.

For selecting the prime implicants, we can use row and column dominance reductions. Row *i* of the prime implcant table dominates row *k* if every nonzero of row *k* is matched by a nonzero of row *i* in the same column; any set of columns that covers row *k* will also cover *i*. Hence dominating rows

TABLE III: STEP 3

Group	Minterm	a	b	c	d
1	8,10,12,14*	1	-	-	0
	8,12,9*,11	1	0	-	-
2	10,11,14*,15	1	-	1	-

TABLE IV: STEP 4

Term	Representation	Minterm					
		4	**8**	**10**	**11**	**12**	**15**
bc'd'	4,12	x				x	
ad	8,10,12,14		x	x		x	
ab'	8,9*,10,11		x	x	x		
ac	10,11,14*,15			x	x		x

may be deleted without affecting the size of the optimum solution. Similarly, column *j* of the prime implcant table dominates column *k* if every nonzero of column *k* is matched by a nonzero of column *j* in the same row; any set of columns that contains column *j* will also cover all rows *i* covered by column *k*.

Using this, we can reduce the boolean expression to the final form:
$f(A, B, C, D) = BC'D' + AB' + AC$

II. PROPOSED SOLUTION

The project describes the implementation of Quine Mc-Cluskey algorithm for minimization of boolean function using Java. The entire code is contained within one Java class "QM.java" wherein we perform all the necessary calculations required to simulate the algorithm.

We begin the program by getting the following input from the user:

1) Number of variables used in the Boolean function presented for simplification;
2) Number of minterms present in the Boolean function presented for simplification
3) The minterms for the Boolean function presented for simplification

Currently, we have only implemented the Boolean expression so that it can be allowed to input in the standard expression only. Depending on the number of variables, we have set the maximum limit on the number of minterms possible to be input for calculation. These minterms are accepted from the user.

Once the minterms have been presented to the algorithm, it proceeds with its execution. The minterms are stored in an array 'minTerms[]'. 2D matrices 'matrixA[][]', 'matrixB[][]' and 'matrixPrimeImpl[][]' are initialized with the number of rows as (((number of minterms)*(number of minterms +1))/2) and number of columns equal to the number of variables entered. The matrix A will store the binary form of each minterm present in 'minTerms[]' array. The matrix B will be used as an intermediate matrix for storing values. The matrixPrimeImpl matrix will store the prime implicants which are calculated by the algorithm. A 'checkerRow[]' array is also created which is used later for checking values in a row in the matrix.

The entire matrix A is filled with values -1 using a helper function 'fillMatrixWithVal()' which takes a 2D array and value to fill in the array as the inputs. After this, the binary forms of all minterms entered are calculated and stored in the matrix A. Following this, grouping is done and the minterms are reduced in a while loop continuously until no further further grouping is possible. We have used multiple variables to hold the intermediate values during this reduction process. Initially, matrixB[][] is completely filled with -1 and checkerRow[] is also filled with -1 using a second helper function 'fillRowWithVal()' which performs similar functionality as 'fillMatrixWithVal()' but for a 1D array. After this, the algorithm is looped throughout the length of matrixA and every binary bit is tested across the groups so as to find minterms whose binary bits differ by only one. Once such minterms are found, they are filled in matrixB using checkerRow array. This entire process will continue until no further grouping of minterms is possible, which will cause the count to become zero and, thus, the loop can be exited.

Once the program has completed its execution of the loop, matrixA now holds the prime implicants in its binary form. These prime implicant are displayed in the table form. The essential prime implicants are next identified and removed from the prime implicant table. Now that we are left with other prime implicants, we need to select the ones which are necessary for the Boolean expression. This is done through using row dominance and column dominance reduction techniques. The rows and columns which are dominated upon by other rows and columns will be remaining and the other dominating rows and columns can be eliminated. This will lead us to our final simplified Boolean expression. A helper function 'decode()' is used to map the binary bits to characters so that the final output can be displayed in a form which is understandable to the user.

III. IMPLEMENTATION ISSUES

Implementation of the algorithm proved tricky due to the large number of bit manipulations involved in the calculations. We had difficulties in implementing the main while loop where the minterms are grouped and reduced till the prime implicants are obtained. We finally settled on the infinite while loop with control to end the loop from within when no further grouping was possible. We also had difficulties in storing the grouped values, as a result of which we decided upon another storage structure(matrixB), despite the fact that it leads to greater wastage of storage space. Implementing row and column dominance were relatively simpler, due to the fact that we only had to compare the rows and columns which were common already stored in the data structure(matrixA).

We also faced difficulties in implementing "don't care" logic as a reult of which we have excluded that portion from our algorithm. Our program currently handles only the important minterms and is not able to handle "don't care" conditions. We were also not able to handle different types of inputs from the user, and we went ahead with only one type of input, which the user could pass to the program for execution. However, when a large number of inputs is to be given to the program, this could become a tedious and time-consuming task.

Our program was able to run successfully and give results as the number of variables increased without any issue.

Test Case No.	No. Of Variables	No. Of Minterms provided to the algorithm	Minterms provided to the algorithm	Minimized Boolean Expression terms Obtained
1	4	8	4, 8, 9, 10, 11, 12, 14, 15	BC'D' + AB' + AC
2	4	4	0, 3, 5, 13	A'B'C'D' + A'B'CD + BC'D
3	4	13	0, 1, 2, 3, 4, 5, 7, 8, 9, 11, 12, 14, 15	ABD' + ABC + A'B' + A'C' + B'C' + C'D' + A'D + B'D + CD
4	5	10	1, 2, 3, 4, 5, 6, 7, 8, 9, 28	ABCD'E' + A'C'D'E + A'BC'D' + A'B'D + A'B'C
5	5	15	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 22, 26	B'CDE' + BC'DE' + A'B'D + A'C'D + A'DE' + A'B'C + A'CD' + A'CE' + A'BC' + A'BD' + A'BE'
6	6	13	5, 6, 7, 8, 9, 10, 11, 12, 16, 19, 24, 27, 31	A'B'C'DF + A'B'C'DE + A'B'CE'F' + A'BD'E'F' + A'BD'EF + A'BCEF + A'B'CD'
7	6	15	5, 6, 7, 9, 13, 16, 17, 21, 23, 27, 28, 31, 35, 39, 43	A'BCDE'F' + A'B'DE'F + A'B'C'DE + B'C'DEF + A'B'CE'F + A'BC'D'E' + A'BC'E'F + A'BDEF + A'BCEF + AB'C'EF + AB'D'EF + A'C'DF
8	7	15	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	A'B'C'D' + A'B'C'E' + A'B'C'F' + A'B'C'G'
9	8	15	1, 2, 13, 16, 19, 25, 27, 29, 35, 38, 41, 46, 49, 57, 69	A'B'C'D'E'F'G'H + A'B'C'D'E'F'GH' + A'B'C'DE'F'G'H' + A'B'CD'E'F'GH + A'BC'D'E'FG'H + A'B'C'EFG'H + A'B'C'DF'GH + A'B'C'DEF'H + A'B'CD'FGH' + A'B'CEF'G'H + A'B'CDF'G'H
10	10	15	3, 17, 28, 33, 41, 57, 61, 72, 77, 79, 83, 88, 91, 95, 101	A'B'C'D'E'F'G'H'IJ + A'B'C'D'E'FG'H'I'J + A'B'C'D'E'FGHI'J' + A'B'C'DEF'G'HI'J + A'B'C'D'EF'H'I'J + A'B'C'D'EFGI'J + A'B'C'DE'GH'I'J' + A'B'C'DE'F'GHJ + A'B'C'DE'GHIJ + A'B'C'DE'FH'IJ

Fig. 1: Execution Result Table

However, as soon as the number of minterms exceeded 15, we were faced with ”Array Index Out Of Bounds” exception which meant that memory does cause an issue for execution of our program in case of large number of minterms. As long as the number of minterms stayed within 15, we were able to run our algorithm without any issue.

IV. EXPERIMENTAL RESULTS

We were able to successfully implement Quine McKluskey algorithm and the table in Fig. 1 shows the inputs we provided and the outputs we received.

As can be seen from the results, our program runs perfectly for any large number of variables. Though, the program fails to execute when the number of minterms exceeds 15. This means that our code still has scope for further improvement and optimization.

As the number of variables increases, it can be seen that the number of terms in the simplified Boolean expression also increases. This is because more variables are added to each term leading to difficulty in reduction of terms. For example, a minterm 5 with four variables might stand for A’B’CD but minterm 5 for five variables stands for A’B’C’DE; there is thus an increase in the number of variables leading to fewer grouping possible per term and thus leading to increase in number of terms in the simplified expression.

Handling of ”don’t care” conditions would also have reduced the number of terms obtained in the simplified expression. However, since we have not implemented these conditions, it is difficult to say how they could influence the final Boolean expression; theoretically, they should reduce the number of terms obtained at the end.

As per the result table, there is also a drastic change in the size of each term of the final simplified expression as the algorithm progresses from having 7 variables to having 8 variables in the input equations. This might mean that our algorithm may be flawed for input variables greater than 7, or, it might also mean that the minterms selected by us for our test cases could be one of such that they cannot be grouped. Further analysis and verification with a trusted working source is required to confirm that our algorithm executes correctly.

Figures 2-14, displays the execution trace of the test cases when it was run on our machine. As can be seen from the figures, the following data is displayed during the execution trace:

1) Minterms entered by the user
2) Reduction by grouping at each pass, i.e., the intermediate grouping tables
3) Prime Implicants at the end of grouping
4) Prime Implicant table
5) Essential Prime Implicants
6) The Prime Implicant table after removing essential Prime Implicants
7) The Prime Implicant table after applying Row and column dominance reduction
8) Literal variables used in the function
9) Final Simplified minterms

V. CONCLUSIONS

We have successfully implemented the Quine McKluskey algorithm and executed several test cases. We have delved in depth into Boolean logic minimization and have come out more knowledgeable with regards to this subject matter. There is still a lot of scope for improvement code wise, but the expertise gained through working on this assignment will help us a lot in improving ourselves.

REFERENCES

[1] https://www.eecs.umich.edu/courses/eecs270/lectures/270L19NotesF14.pdf

[2] http://www.async.ece.utah.edu/ myers/nobackup/ee5740_98/lec8_2.pdf
[3] https://en.wikipedia.org/wiki/Quine%E2%80%93McCluskey_algorithm
[4] https://cseweb.ucsd.edu/classes/su09/cse140/Quine.pdf

APPENDIX

QM.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.lang.*;

public class QM {

  public static int getInt() throws
      IOException {
    String s = getString();
    return Integer.parseInt(s);
  }

  public static char getChar() throws
      IOException {
    String s = getString();
    return s.charAt(0);
  }

  public static String getString() throws
      IOException {
    InputStreamReader isr = new
        InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(
        isr);
    String s = br.readLine();
    return s;
  }

  public static void fillMatrixWithVal(int x
      [][], int val) {
    for (int i=0; i<x.length; i++) {
      for (int j=0; j<x[i].length; j++) {
        x[i][j] = val;
      }
    }
  }

  public static void fillRowWithVal(int x[],
      int val) {
    for (int i = 0; i < x.length; i++)
      x[i] = val;
  }

  public static boolean getPIfromMinterms(int
      min, int a[][], int row, int
      numOfVariables) {

    int b[]=new int[numOfVariables], i=
        numOfVariables-1, c=0;
    fillRowWithVal(b, 0);
    while (min>0) {
      b[i]= min%2;
      min = min/2;
      i--;
    }
    for (i=0; i<numOfVariables; i++) {
      if (a[row][i] == 9) {
        continue;
      }
      if (a[row][i] != b[i]) {
        c++;
      }
    }
    if (c == 0) {
      return true;
    }
    return false;
  }

  public static String decode(int x[][], int
      row, int numOfVariables, char bitvar[])
       {
    String S = "";
    for (int i = 0; i < x[row].length; i++) {
      if (x[row][i] == 9)
        continue;
      else if (x[row][i] == 1)
        S += bitvar[i];
      else
        S += bitvar[i] + "'";
    }
    return S;
  }

  public static void main(String args[])
      throws IOException {

    System.out.print("Enter the number of
        variables: ");
    int numOfVariables = getInt();
    System.out.print("Enter the number of
        minterms present in the expression (
        max "+(int)(Math.pow(2,numOfVariables
        )-1)+"): ");
    int numOfMinterms = getInt();
    int minTerms[] = new int[numOfMinterms];

    for(int i=0; i<numOfMinterms; i++) {
      System.out.print("Enter minterm in
          ascending order (Remaining Minterms
           - "+(numOfMinterms-i)+"): ");
      minTerms[i] = getInt();
    }

    int matrixA[][] = new int[numOfMinterms*(
        numOfMinterms+1)/2][numOfVariables];
    int matrixB[][] = new int[numOfMinterms*(
        numOfMinterms+1)/2][numOfVariables];
    int matrixPrimeImpl[][] = new int[
        numOfMinterms*(numOfMinterms+1)/2][
        numOfVariables];
    int checkerRow[] = new int[numOfMinterms
        *(numOfMinterms+1)/2];

    fillMatrixWithVal(matrixA, -1);

    for (int i=0; i<numOfMinterms; i++) {
      for (int j=0; j<numOfVariables; j++) {
        matrixA[i][j] = 0;
      }
    }

    int position=0;
    for (int i=0; i<numOfMinterms; i++) {
      int temp = minTerms[i];
      position = numOfVariables - 1;
      while(temp>0) {
        matrixA[i][position] = temp % 2;
```

```
        temp = temp/2;
        position--;
      }
    }

    System.out.println("\nThe following
        minterms have been entered: ");
    for (int i = 0; i < numOfMinterms; i++) {
      System.out.print(minTerms[i] + " ");
    }
    System.out.println("\n");

    System.out.println("\nThe following shows
        the grouping happenning at each pass
       :\n");
    int count=0;
    int flag=0, flag1=0, flag2=0;
    int i, j, k;
    int c, c1, c2, c3;
    int x, y;
    while(true){
      count = 0;
      flag = 0;
      fillMatrixWithVal(matrixB, -1);
      fillRowWithVal(checkerRow, -1);
      for (i=0; i<matrixA.length; i++) {
        if (matrixA[i][0] == -1) {
          break;
        }
        for (j=i+1; j<matrixA.length; j++) {
          c = 0;
          if (matrixA[j][0] == -1) {
            break;
          }
          for (k=numOfVariables-1; k>=0; k
              --) {
            if (matrixA[i][k] != matrixA[j][
                k]) {
              position = k;
              c++;
            }
          }
          if (c==1) {
            count++;
            checkerRow[i]++;
            checkerRow[j]++;
            for (k=numOfVariables-1; k>=0; k
                --) {
              matrixB[flag][k] = matrixA[i
                  ][k];
            }
            matrixB[flag][position] = 9;
            flag++;
          }
        }
      }
      for (j=0; j<i; j++) {
        if (checkerRow[j] == -1) {
          for (k=0; k<numOfVariables; k++) {
            matrixPrimeImpl[flag2][k] =
                matrixA[j][k];
          }
          c3 = 0;
          for (x=(flag2-1); x>=0; x--) {
            c1 = 0;
            for(y=0; y<numOfVariables; y++)
                {
              if(matrixPrimeImpl[x][y] !=
                  matrixPrimeImpl[flag2][y])
                  {
                c1++;
              }
            }
            if (c1 == 0) {
              c3++;
              break;
            }
          }
          if(c3==0) {
            flag2++;
          }
        }
      }
      if(count==0) {
        break;
      }
      for (i=0; i<matrixB.length; i++) {
        if (matrixB[i][0] == -1) {
          break;
        }
        for(j=0; j<numOfVariables; j++) {
          if (matrixB[i][j] == 9) {
            System.out.print("_");
          }
          else {
            System.out.print(matrixB[i][j]);
          }
        }
        System.out.println();
      }
      System.out.println();
      for (i=0; i<matrixB.length; i++) {
        for (j = 0; j < matrixB[i].length; j
            ++) {
          matrixA[i][j] = matrixB[i][j];
        }
      }
      flag1++;
    }

    System.out.println("\nThe binary forms of
        the Prime Implicant are: ");
    for (i=0; i<flag2; i++) {
      for (j=0; j<numOfVariables; j++) {
        if (matrixPrimeImpl[i][j] == 9)
          System.out.print("_");
        else
          System.out.print(matrixPrimeImpl[i
              ][j]);
      }
      System.out.println();
    }
    System.out.println();

    int dash[]=new int[numOfVariables];
    fillRowWithVal(dash, -1);
    matrixA=new int[flag2][numOfMinterms];
    fillMatrixWithVal(matrixA, 0);
    for (i = 0; i < flag2; i++) {
      for (j = 0; j < numOfMinterms; j++) {
        boolean check = getPIfromMinterms(
            minTerms[j], matrixPrimeImpl, i,
             numOfVariables);
        if (check == true)
          matrixA[i][j] = 1;
      }
```

```
    }


    System.out.println("The Prime Implicant table : ");
    for (i=0; i<numOfMinterms; i++) {
      System.out.print(minTerms[i]+"\t");
    }
    System.out.println();
    for (i = 0; i < matrixA.length; i++) {
      for (j = 0; j < numOfMinterms; j++) {
        if (matrixA[i][j] == 1) {
          System.out.print((char)(matrixA[i][j]+87)+"\t");
        }
        else {
          System.out.print(" "+"\t");
        }
      }
      System.out.println();
    }

    checkerRow = new int[flag2];
    dash = new int[numOfMinterms];
    count = 0;
    fillRowWithVal(checkerRow, -1);
    fillRowWithVal(dash, -1);
    for (j = 0; j < numOfMinterms; j++) {
      count = 0;
      for (i = 0; i < flag2; i++) {
        if (matrixA[i][j] == 1) {
          position = i;
          count++;
        }
      }
      if (count == 1)
        checkerRow[position]++;
    }

    System.out.println("\nThe binary forms of the ESSENTIAL Prime Implicant are: ");
    for (i=0; i<flag2; i++) {
      if (checkerRow[i] != -1) {
        for (j=0; j<numOfVariables; j++) {
          if (matrixPrimeImpl[i][j] == 9) {
            System.out.print("_");
          }
          else {
            System.out.print(matrixPrimeImpl[i][j]);
          }
        }
        System.out.println();
      }
    }
    System.out.println();

    for (i = 0; i < flag2; i++) {
      if (checkerRow[i] != -1) {
        for (j=0; j<numOfMinterms; j++) {
          if (matrixA[i][j] == 1)
            dash[j]++;
        }
        for (j=0; j<numOfMinterms; j++)
          matrixA[i][j] = -1;
      }
    }
    for (j=0; j<numOfMinterms; j++) {
      if (dash[j] != -1) {
        for (i=0; i<flag2; i++) {
          matrixA[i][j] = -1;
        }
      }
    }


    System.out.println("The PI table after removing essential PI : ");
    for (i=0; i<numOfMinterms; i++) {
      System.out.print(minTerms[i]+"\t");
    }
    System.out.println();
    for (i=0; i<matrixA.length; i++) {
      for (j=0; j<numOfMinterms; j++) {
        if (matrixA[i][j] == 1) {
          System.out.print((char)(matrixA[i][j]+87)+"\t");
        }
        else {
          System.out.print(" "+"\t");
        }
      }
      System.out.println();
    }

    System.out.println();

    int nonine;
    while(true) {
      count = 0;
      for(j=0; j<numOfMinterms; j++) {
        for(k=j+1; k<numOfMinterms; k++) {
          c1 = 0; c2 = 0; c3 = 0;
          for(i=0; i<flag2; i++) {
            if(matrixA[i][j]==1 && matrixA[i][k]==1) {
              c1++;
            }
            if(matrixA[i][j]==1 && matrixA[i][k]==0) {
              c2++;
            }
            if(matrixA[i][j]==0 && matrixA[i][k]==1) {
              c3++;
            }
          }
          if(c2>0 && c3>0) {
            break;
          }
          if(c1>0 && c2>0 && c3==0) {
            for(nonine=0; nonine<flag2; nonine++) {
              matrixA[nonine][j] = -1;
            }
            count++;
          }
          if(c1>0 && c3>0 && c2==0) {
            for(nonine=0; nonine<flag2; nonine++) {
              matrixA[nonine][k] = -1;
            }
            count++;
          }
```

```
            if(c1>0 && c2==0 && c3==0) {
              for(nonine=0; nonine<flag2;
                  nonine++) {
                matrixA[nonine][j] = -1;
              }
              count++;
            }
          }
        }

        for(i=0; i<flag2; i++) {
          for(j=i+1; j<flag2; j++) {
            c1 = 0; c2 = 0; c3 = 0;
            for(k=0; k<numOfMinterms; k++) {
              if(matrixA[i][k]==1 && matrixA[j
                  ][k]==1) {
                c1++;
              }
              if(matrixA[i][k]==1 && matrixA[j
                  ][k]==0) {
                c2++;
              }
              if(matrixA[i][k]==0 && matrixA[j
                  ][k]==1) {
                c3++;
              }
            }
            if(c2>0 && c3>0) {
              break;
            }
            if(c1>0 && c2>0 && c3==0) {
              for(nonine=0; nonine<
                  numOfMinterms; nonine++) {
                matrixA[j][nonine] = -1;
              }
              count++;
            }
            if(c1>0 && c3>0 && c2==0) {
              for(nonine=0; nonine<
                  numOfMinterms; nonine++) {
                matrixA[i][nonine] = -1;
              }
              count++;
            }
            if(c1>0 && c2==0 && c3==0) {
              for(nonine=0; nonine<
                  numOfMinterms; nonine++) {
                matrixA[j][nonine] = -1;
              }
              count++;
            }
          }
        }
        if (count==0) {
          break;
        }
      }

      System.out.println("PI table after
          applying both Row and Column
          Dominance : ");
      for (i=0; i<numOfMinterms; i++) {
        System.out.print(minTerms[i]+"\t");
      }
      System.out.println();
      for (i=0; i<matrixA.length; i++) {
        for (j=0; j<numOfMinterms; j++) {
          if (matrixA[i][j] == 1) {
            System.out.print((char)(matrixA[i
                ][j]+87)+"\t");
          }
          else {
            System.out.print(" "+"\t");
          }
        }
        System.out.println();
      }
      System.out.println();

      for (i=0; i<matrixA.length;i++) {
        for (j=0; j<numOfMinterms; j++) {
          if (matrixA[i][j]==1) {
            checkerRow[i]++;
          }
        }
      }

      char bitvar[] = new char[numOfVariables];
      for (i=0; i<numOfVariables; i++) {
        bitvar[i] = (char)(65+i);
      }

      System.out.print("\nThe variables used in
          this expression are: ");
      for (i = 0; i < numOfVariables; i++) {
        System.out.print(bitvar[i] + " ");
      }
      System.out.println();
      System.out.println();

      System.out.println("The final simplified
          minterms are :");
      for(i=0; i<flag2; i++) {
        if(checkerRow[i] != -1) {
          System.out.println(decode(
              matrixPrimeImpl, i,
              numOfVariables, bitvar));
        }
      }
    }
}
```