**Logo, company name

Description automatically generated  Pak-Austria Fachhochschule**

  Institute of Applied Sciences and Technology

Mang, Haripur, Khyber Pakhtunkhwa, Pakistan

# **PROJECT REPORT**

**GROUP MEMBERS:**

* MUHAMMAD ASHARIB REHAN (B21F0219AI023)
* HAFSA WAHEED (B21F0386AI040)

**DEPARTMENT:** ARTIFICIAL INTELLIGENCE

**DATED:** 26 – 01 – 22

**COURSE:**DIGITAL AND LOGIC DESIGN

**SUBMITTED TO:** DR. SYED ALI ASADULLAH BUKHARI

**QUINE–MCCLUSKEY & PETRICK’S METHOD**

**INTRODUCTION:**

The Quine–McCluskey algorithm is functionally identical to Karnaugh mapping, but the tabular form makes it more efficient for use in computer algorithms, and it also gives a deterministic way to check that the minimal form of a Boolean function has been reached. It is sometimes referred to as the tabulation method.

**PROCEDURE OF QM METHOD:**

Petrick's method is a systematic method for finding all the possible minimum Sum of Products (SoP) forms for a Boolean function with a given number of variables. The method was first introduced by R. T. Petrick in his paper "A Method for Finding All Minimum Sum-of-Product Forms of a Boolean Function" in 1978. In this report, we will discuss Petrick's method for finding all the possible minimum SoP forms for a Boolean function with 5 variables.

The first step in Petrick's method is to create a table of all the prime implicants of the Boolean function. A prime implicant is a product term that cannot be further reduced or simplified without losing some of the minterms (the Boolean function values that are 1). For example, for a Boolean function with 5 variables, there will be 32 minterms, and the table of prime implicants will have 32 rows and 32 columns, with each column representing a minterm and each row representing a prime implicant.

The next step is to find the essential prime implicants. These are the prime implicants that cover at least one minterm that is not covered by any other prime implicant. These essential prime implicants are then added to the SoP form of the Boolean function.

After finding the essential prime implicants, we can then use Petrick's method to find the remaining non-essential prime implicants. The method involves creating a matrix of all the minterms that are covered by each prime implicant and then using Boolean algebra to find the possible combinations of these minterms. The resulting combination of minterms that covers all the minterms is the SoP form of the Boolean function.

In the case of a Boolean function with 5 variables, there may be multiple SoP forms, as there may be multiple ways to combine the prime implicants to cover all the minterms. Petrick's method can be used to find all the possible minimum SoP forms for the Boolean function.

In conclusion, Petrick's method is a systematic and efficient method for finding all the possible minimum SoP forms for a Boolean function with a given number of variables. The method involves creating a table of prime implicants, finding the essential prime implicants, and using Boolean algebra to find the possible combinations of non-essential prime implicants. With Petrick's method, we can find all the possible minimum SoP forms for a Boolean function with 5 variables.

**ALGORITHM FOR IMPLEMENTATION:**

**Objectives:**

1. Read in (and validate) a Boolean function using its minterms and don’t care terms (as decimal numbers).
2. Generate and print all prime implicants using the Quine-McCluskey tabulation method.
3. Using the prime implicants generated, obtain, and print all the essential prime implicants.
4. Print the optimized function based on the least number of literals and the least number of inversions.

**Code Description:**

**List of the functions used in this algorithm:**

def multiply\_minterms(): # Multiply 2 minterms

def multiply\_exp(): # Multiply 2 expressions

def remv\_dontcare(): # Removes don't care terms from the list

def fEPI(): # Find essential prime implicants

def findVar(): # Function to find variables in a minterm

def flatten(): # Flattens a list

def findmin(): # Finding out which minterms are grouped

def compare(): # Checking if minterms differ by 1 bit only

def removeTerms(): # Removes minterms which are in chart

**Input:**

The user inputs the minterms as well as don’t cares (if any) and press enter to start executing

**Procedures of QM (step 1):**

All the minterms are converted into their binary form and arranged into different groups according to the number of 1s in the binary representation. These groups are put in ascending order in the table and displayed the table.

CODE:

|  |
| --- |
| minterms.sort()  size = len(bin(minterms[-1]))-2  groups,all\_pi = {},set()  for minterm in minterms:      try:          groups[bin(minterm).count('1')].append(bin(minterm)[2:].zfill(size))      except KeyError:          groups[bin(minterm).count('1')] = [bin(minterm)[2:].zfill(size)]  print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='\*50))  for i in sorted(groups.keys()):      print("%5d:"%i)      for j in groups[i]:          print("\t\t    %-20d%s"%(int(j,2),j))      print('\_'\*50) |

**Procedures of QM (step 2):**

After making all possible groups, from all columns, the terms (which are not ticked) are PI’s

Before clearing columns and working on the terms generated from the comparison we extracts the unticked minterms as they are the PIs. While doing that the PIs are stored as a new form in a new variable

CODE:

|  |
| --- |
| print("Unmarked elements(Prime Implicants) of this table:",None if len(local\_unmarked)==0 else ', '.join(local\_unmarked))      if should\_stop:          print("\n\nAll Prime Implicants: ",None if len(all\_pi)==0 else ', '.join(all\_pi))          break      print("\n\n\n\nGroup No.\tMinterms\tBinary of Minterms\n%s"%('='\*50))      for i in sorted(groups.keys()):          print("%5d:"%i)          for j in groups[i]:              print("\t\t%-24s%s"%(','.join(findmin(j)),j))          print('\_'\*50) |

Then After taking all the PI’s draw the Prime implicant chart

CODE:

|  |
| --- |
| sz = len(str(mint[-1]))  chart = {}  print('\n\n\nPrime Implicants chart:\n\n    Minterms    |%s\n%s'%(' '.join((' '\*(sz-len(str(i))))+str(i) for i in mint),'='\*(len(mint)\*(sz+1)+16)))  for i in all\_pi:      merged\_minterms,y = findmin(i),0      print("%-16s|"%','.join(merged\_minterms),end='')      for j in remv\_dontcare(merged\_minterms,dontc):          x = mint.index(int(j))\*(sz+1)          print(' '\*abs(x-y)+' '\*(sz-1)+'X',end='')          y = x+sz          try:              chart[j].append(i) if i not in chart[j] else None          except KeyError:              chart[j] = [i]      print('\n'+'\_'\*(len(mint)\*(sz+1)+16)) |

**Procedures of QM (step 3):**

An ‘PI chart’ will be made. Each column represents a minterm (excluding don’t cares) and each row represents a PI. Column with only a single **X** indicates a minterm covered by only one PI, which means this PI is EPI.

Algorithm implementation:

For checking if PI is an EPI or not the function

1) It loops over each minterms, checking only the not-dontcare-minterms for each minterm it loops over the PIs. Calling fEPIchecking if it exists in how many primes.

2) Loops over all the minterms if it's only included in one prime

3) Then prints all the essential prime implicants

|  |
| --- |
| EPI = fEPI(chart)  print("\nEssential Prime Implicants: "+', '.join(str(i) for i in EPI))  removeTerms(chart,EPI) |

**Procedures of QM (step 4):**

Convert these EPI’s in Alphabetical form, the EPI with 0 in it should be printed in complemented form.

|  |
| --- |
| if(len(chart) == 0):      final\_result = [findVar(i) for i in EPI]  else:      P = [[findVar(j) for j in chart[i]] for i in chart]      while len(P)>1:          P[1] = multiply\_exp(P[0],P[1])          P.pop(0)      final\_result = [min(P[0],key=len)]      final\_result.extend(findVar(i) for i in EPI)  print('\n\nSolution: F = '+' + '.join(''.join(i) for i in final\_result)) |

This is considered as the reduced minimal form of a Boolean function F.

**RESULT:**

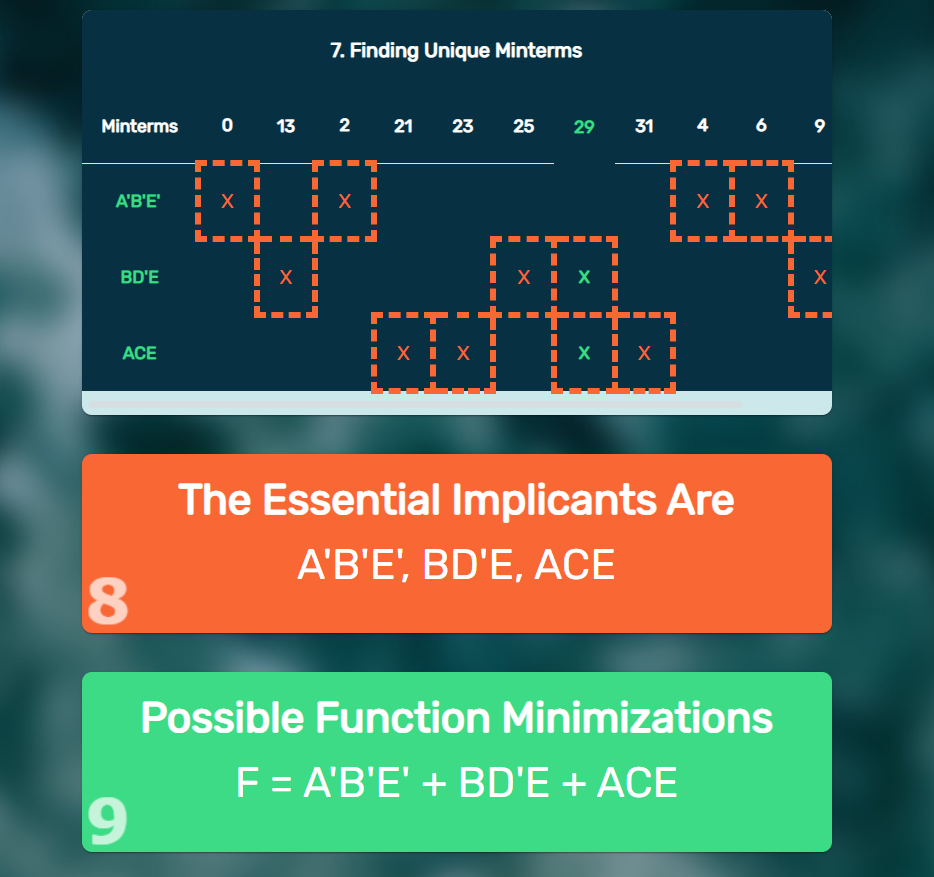
DEMO OUTPUTS:

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

**Note:**

The given output is correct and checked from:

reference link: (<https://geeekyboy.github.io/Quine-McCluskey-Solver/#/>)

**Examples:**

Text

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated