**Digital Computer Fundamentals**

**Tabular Method– Opti-Circuit**

**Final Project Report**

***By***

**HeshamMedhat & Marina Zakaria**

**Tabular Method Project Report**

**CS131 Course, Faculty of Engineering, Alexandria University.**

**Computer & Systems Engineering Dpt.**

**To Professor Mustafa El-Nenaei**

**and teaching assistants:**

**• Eng/ Reham Mohamed Samir.**

**• Eng/ RanaMomtaz.**

**Index**

1. Introduction 4
2. Overview 5
3. Features 5
4. Data Structures 6
5. Functions 9
6. Test Cases 18
7. User Manual 22

**Introduction**

This project was assigned in the CS131: Digital Computer Fundamentals course in the date: Friday, April 28th, 2017.

And due : Monday, May15th, 2017.

As the final project of the course, we were assigned this final project as teams of two.

This delivery is by :

*Hesham Medhat Mahmoud Ahmed Abou-Mousa*

&

*Marina Zakaria Boules Youssef Zakaria*

The source of this project is available on github on this link:

<https://github.com/EngHesham/Opti-Circuit/>

**Overview**

This program intends to optimize a digital logic circuit for possible solutions of lowest cost.

All the program needs as input is the minterms of the function that needs optimization and the don’t-care elements (optional).

**Features**

*In our implementation,*

The program can:

• give all possible optimal solutions.

• detect the best solution eith least cost

• output all prime implicants.

• output all essential prime implicants.

• show some steps of solution.

• read input from a file.

• print output in a file.

**Data Structures**

As we are learning the course Data Structures-1 in the very same semester, we are using our own implementations of Data Structures in this project.

We are using Singly Linked Lists, Doubly Linked Lists, of our own implementation.

How we used them is described thoroughly in the respective interfaces in the “Functions” section.

Here is the ILinkedList interface:

ILinkedList {

/\*\*

\* Inserts a specified element at the specified position in the

\* list.

\*/

public void add(int index, Object element);

/\*\* Inserts the specified element at the end of the list. \*/

public void add(Object element);

/\*\* Returns the element at the specified position in this list.

\*/

public Object get(int index);

/\*\*

\* Replaces the element at the specified position in this list

\* with the specified element.

\*/

public void set(int index, Object element);

/\*\* Removes all of the elements from this list. \*/

public void clear();

/\*\* Returns true if this list contains no elements. \*/

publicbooleanisEmpty();

/\*\* Removes the element at the specified position in this list. \*/

public void remove(int index);

/\*\* Returns the number of elements in this list. \*/

publicint size();

/\*\*

\* Returns a view of the portion of this list between the

\* specified

\* fromIndex and toIndex, inclusively.

\*/

publicILinkedListsublist(intfromIndex, inttoIndex);

/\*\*

\* Returns true if this list contains an element with the same

\*value as thespecified element.

\*/

publicboolean contains(Object o);

}

**Functions**

**A. Module: PrimeImplicants:**

IPrimeImplicants {

/\*\*

\* listing minterms in groups

\* **@author**Marina

\* **@parameters**int array of minterms

\* **@return** array of linked lists

\*/

SinglyLinkedList[] listing(**int**[] minterms);

/\*\*

\* Sorts the combinations of the implicant.

\* **@author**Marina

\* **@param**implicant input

\* **@return** sorted implicant

\*/

DoublyLinkedListsortImplicantCombinations(DoublyLinkedListimplicant);

/\*\*

\* combine one haming distance minterms

\*

\* **@parameters** 2 lists return a list

\*/

SinglyLinkedListcombiningTwoGroups(SinglyLinkedList group1, SinglyLinkedList group2);

/\*\*

\* performs the method combine 2 groups \* depending on the length of the array

\* of lists

\* call itself recursively till the

\* length of the list is dropped down to

\* 1 list

\* **@author** Marina

\* **@parameter** array of singly linkedlists

\* **@return** array of singly linked lists

\*/

SinglyLinkedList []combineOneLevel (SinglyLinkedList[]list);

/\*\*

\* performs the method combining one

\* level till it generates 1 group only

\* containing the essential prime

\* implicants

\*

\* **@author**Marina

\* **@parameter** array of integers

\* containing minterms

\* **@return** one singly linked list of prime implicants ready to get

\* combinations method

\*/

DoublyLinkedList[] generatePrimeImplicants(**int**[] minterms);

**B. Module: EssentialPrimeImplicants:**

/\*\*

\* This is the interface of the class that finds the essential

\* primeimplicants.

\* **@author** H

\*

\*/

**publicinterface**IEssentialPrimeImplicants {

/\*\*

\* Gets possible minterm combinations

\* from a PI.

\* This method works recursively adding

\* combinations to a given list.

\* **@param** node of the list representation

\* of PI.

\* **@param** sum of the combinations picked

\* so far.

\* **@param**coveredMTs list where the

\* possible combinations will be added.

\*/

**publicvoid**findCombinations (**final**DLNode node, **int** sum, **final**DoublyLinkedListcoveredMTs);

/\*\*

\* This method gets the array of MTs

\* covered by given PIs.

\* **@param** primes array of PIs lists.

\* **@return**An array of lists of MTs

\* covered by the PIs.

\* Indices of this array correspond to

\* the indices of

\* the given PIs array.

\*/

**public**DoublyLinkedList[] coveredMinterms(DoublyLinkedList[] primes);

/\*\*

\* This method produces a list of given

\* prime implicants.

\* **@param**coveredMT this is the array of

\* lists of covered MTs by each PI.

\* Obtained from coveredMinterms method.

\* **@param**minterms this is the sorted

\* array of MTs.

\* **@return**An array of covering PIs for

\* each MT.

\* Indices of this array correspond to

\* the MTs.

\*/

**public**DoublyLinkedList[] coveringPIs(DoublyLinkedList[] coveredMT, **int**[] minterms);

/\*\*

\* This method produces the solution

\* formula.

\* **@param** coveringImplicants array

\* obtained from coveringImplicants.

\* **@return** String of the solution

\* formula.

\* This needs simplification to get the

\* best possible solutions

\* for optimization.

\* At this point, we need boolean algebra

\* solving techniques to

\* get the final optimal answers.

\*/

**public** String getFormula(**final**DoublyLinkedList[] coveringImplicants);

/\*\*

\* This method finds the essential PIs

\* using dominance.

\* **@param** coveringImplicants array

\* obtained from coveringImplicants.

\* **@return** The essential prime implicants

\* in the form of a

\* string as PxPyPz where (x,y,z..)

\* represent the indices

\* of these implicants in the "primes"

\* input list.

\*/

**public**DoublyLinkedListgetEssentials(**final**DoublyLinkedList[] coveringImplicants);

/\*\*

\* This function uses recursion to find

\* all possible solutions

\* of how to use the covering PIs to

\* cover the MTs we need covered.

\* **@param** solutions the returner list of

\* solutions

\* **@param** toCover index of coveringPIs MT

\* that we need covered

\* **@param** coveringPIs obtained from

\* coveringPIs method

\* **@param** thisSolution the current

\* solution that the method

\* is trying to get in the recursive

\* calls

\* **@return** the list of SLLs of solutions.

\*/

**public**DoublyLinkedListfindSolutions(DoublyLinkedList[] coveredMT, DoublyLinkedList[] powerSet, **int**[] minterms);

/\*\*

\* This method uses all methods in this

\* class to obtain all the

\* possible solutions for the given

\* primes to cover given minterms.

\* **@param** primes array of prime

\* implicants (DLLs).

\* **@param** minterms list of minterms to

\* cover.

\* **@return** a String list of solutions.

\* Each element of this list is a string

\* solution in the form

\* ofPxPyPz where (x,y,z,..) are the

\* indices of the PIs in the

\* given primes array.

\*/

**public**DoublyLinkedListgetSolutions(**final**DoublyLinkedList[] primes, **finalint**[] minterms);

/\*\*

\* This method turn all the possible

\* solutions into functions with

\* literals.

\* **@author** Marina

\* **@param** primes array of prime

\* implicants (DLLs).

\* **@param** solutions DLL of DLLs each list

\* is a solution

\* **@return** array of strings each string

\* is a possible solution

\*/

**public** String[]possibleOptimization (DoublyLinkedList[] primes, DoublyLinkedList solutions, **int**maxChar);

/\*\*

\* This method gets a power series of all

\* possible combinations

\* of the prime implicants. We will use

\* this in our complete

\* search algorithm to find the best

\* solutions.

\* **@param** numberOfPrimeImplicants length

\* of primes array

\* **@return** DLL of the power set of

\* indices supposedly, in the

\* primes array.

\*/

**public**DoublyLinkedList[] powerSet(**int**numberOfPrimeImplicants) ;

/\*\*

\* Prints a given list of prime

\* implicants.

\* **@param** list to be returned

\* **@param** primes array of prime

\* implicants

\* **@param**maxChar maximum literals in an

\* implicant

\* **@return** String array where each

\* element is an implicant

\* in the function form of literals.

\*/

**public** String printImplicants(DoublyLinkedList list, DoublyLinkedList[] primes, **int**maxChar);

/\*\*

\* Gets the best solution(s) among all possible

\* solutions obtained from the possibleOptimization.

\* **@param**possibleOptimization string array of possible solutions

\* **@return** the best solution(s).

\*/

**public** String[] bestOptimization(String[]possibleOptimization);

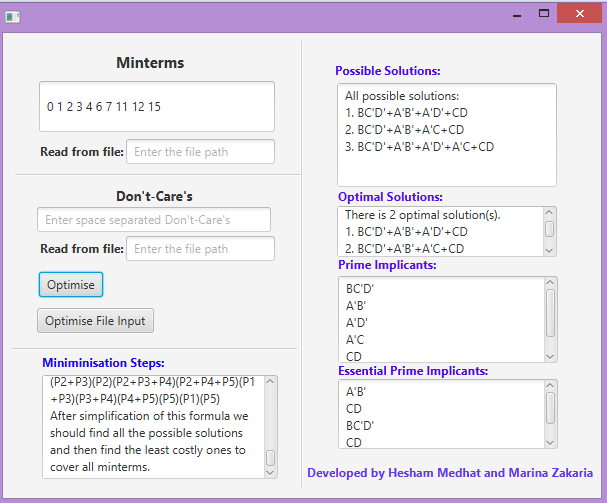
**Test cases**

**Test (1)**

**Simplify the following expression to sum of product using Tabulation Method**

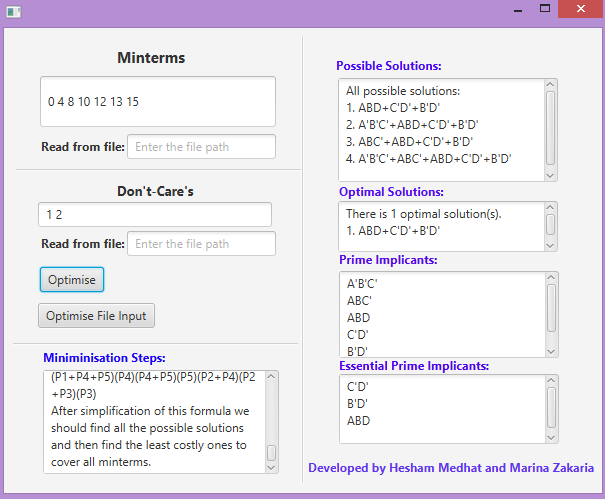
**f (a,b,c,d ) = ∑(0,1,2,3,4,6,7,11,12,15)**

**Solution: f(a,b,c,d) = ac’d’+a’b’+cd+a’d’**

****

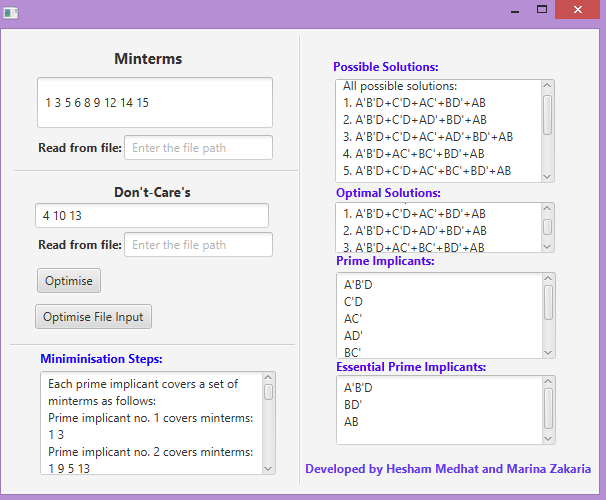
**f(a,b,c,d ) = ∑(0,4,8,10,12,13,15 ) + ∑d(1,2 )**

**Solution: f(a,b,c,d) = abd+c’d’+b’d’**

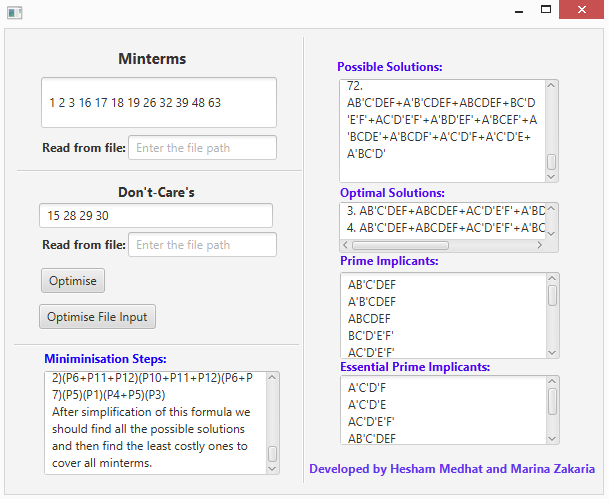
****

**f(a, b, c, d) = Σm(1, 3, 5, 6, 8, 9, 12, 14, 15)**

**+ ∑d(4, 10, 13)**

****

**f(a,b,c,d,e,f ) = ∑(1 2 3 16 17 18 19 26 32 39 48 63) + ∑d(15 28 29 30)**

****

**User Manual**

• The minterms represent the binary combinations at which the function output of your logic circuit should be one.

• You should enter the minterms and don’t-care elements separated by a space in the regular text input in the GUI.

• You should enter the minterms and don’t-care elements separated by a comma in the file when reading from a file.

• Everything else should be clear and intuitive as you are using the GUI model we created.

• The optimal solution is assumed to be the one with the least number of prime implicants used

*Thank you.*