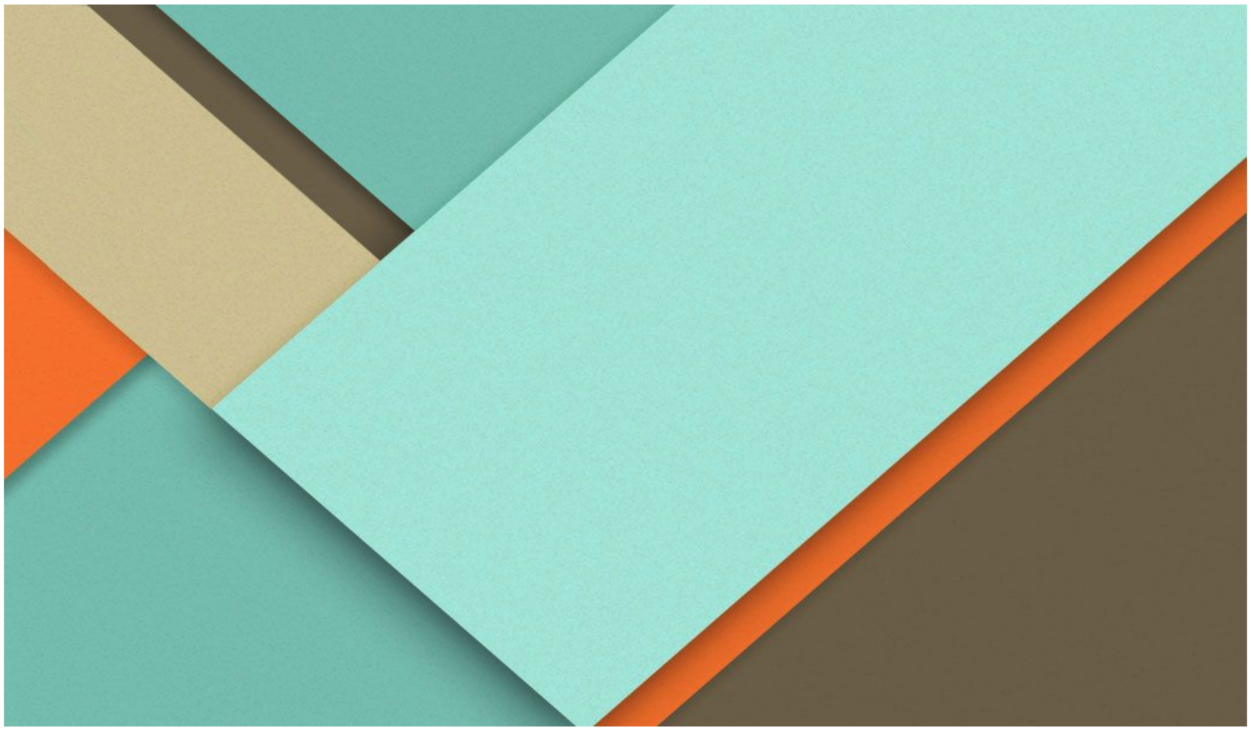


EE 677 COURSE PROJECT

LUT MAPPING

INSTRUCTOR : Prof. Sachin B. Patkar

2016
REPORT

Abhishek G. Konale
14D070010

OBJECTIVE:

To transform a given logic circuit into its equivalent realization using several 6-LUTs

INPUT/OUTPUT SPECIFICATIONS:

Input:

Logic expression(Representation of logic circuit) to implemented using 6-LUTs with following constraints:

1.Accepted literals in expressions ‘a’, ‘b’, ... ‘z’ , ‘A’, ... ‘Z’ (not ‘a’ denoted as ‘~a’ ...)

2.Supported operators ‘and’ , ‘or’ , ‘xor’ , ‘not’

3.Not operator should be at the leaf nodes

Output:

1.Mappings of binary gates to 6-LUTs

2.Graphical visualization of obtained partitions of the network (in png format)

3.Bits to be filled in the 6-LUTs

4.Connection of inputs with the LUTs and interconnection of the LUTs

DESCRIPTION:

The LUT Mapping Problem has been formulated as Tree Covering Problem, hence the objective becomes to minimize the number of partitions required to cover the tree representation(Logic Circuit) for the given logic expression.

Basic Idea:

Cost of covering a node - Minimum number of 6-LUTs required to cover the tree rooted at the given node

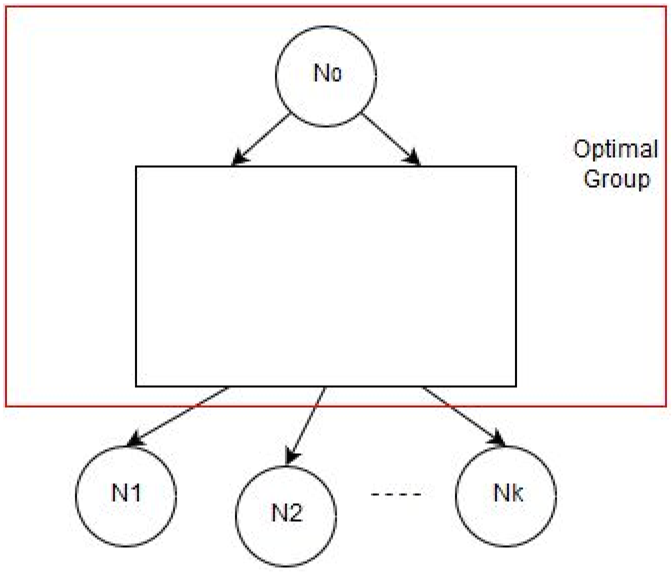
Optimal Group -A maximal group of nodes (consisting of a root node) that can be covered by a single 6-LUT (may not be unique)

Observations:

1. **For leaf nodes the Cost is 1**
2. **For higher nodes :**

$$\text{Cost(N0)} = 1 + \text{Cost(N1)} + \text{Cost(N2)} \ldots + \text{Cost(Nk)}$$

$$= 1 + \Sigma\,\text{Cost(j)} \qquad \text{s.t } j \text{ is in fanout of Optimal Group}$$



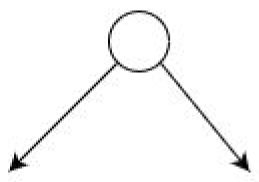
The above observation naturally leads to a Dynamic Programming approach for the given problem due it's recursive characteristics and overlapping nature.

But the nodes which fanout to multiple nodes could possibly create conflicts,hence binary tree representation (i.e converting given logic expression into a circuit consisting of only binary gates) is suitable.

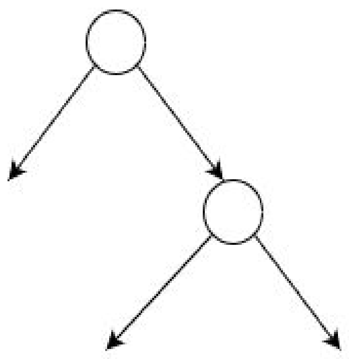
Now we may choose suitable evaluation ordering i.e postorder for this problem.

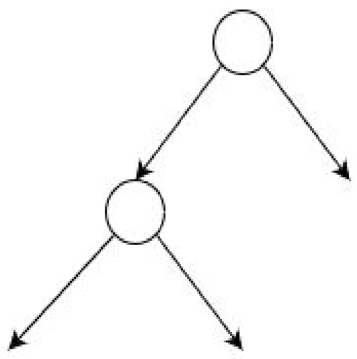
Possible Structures of Optimal Groups:

1. #Nodes = 1 [1]

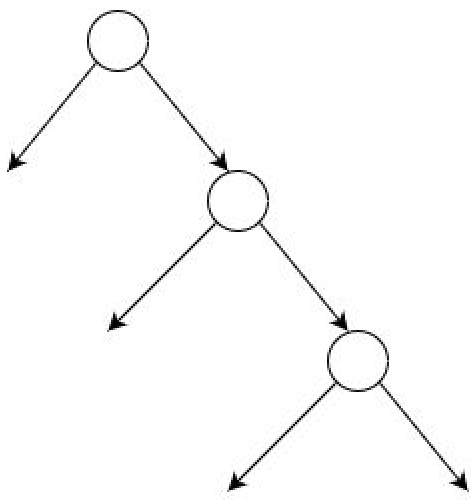


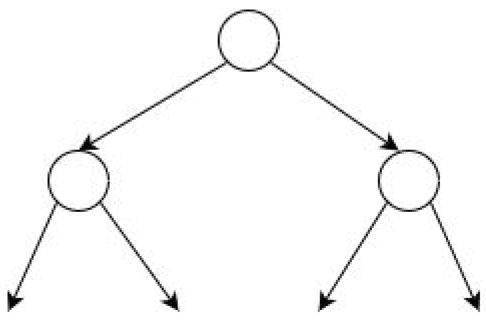
2. #Nodes = 2 [1,1]



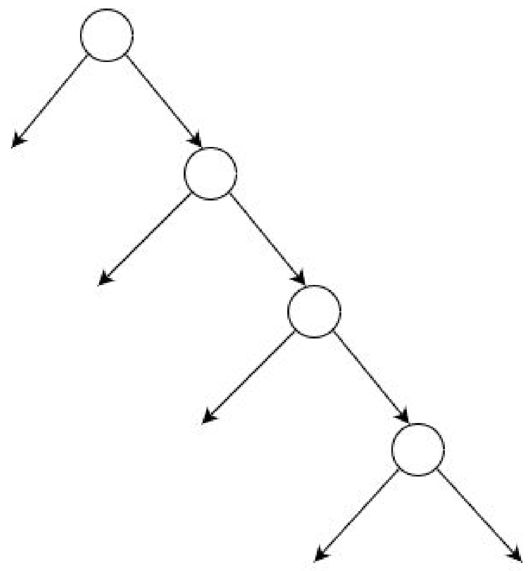


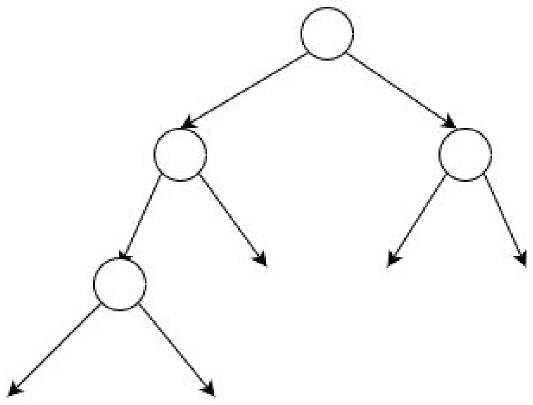
3. #Nodes = 3 [1,1,1] ,[1,2]

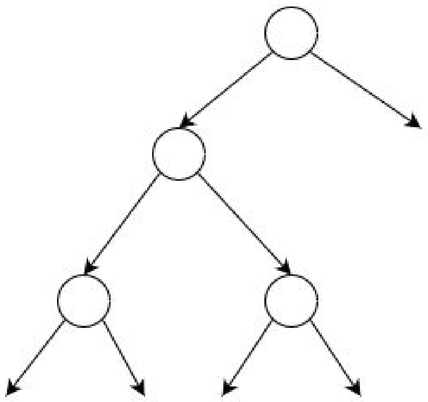




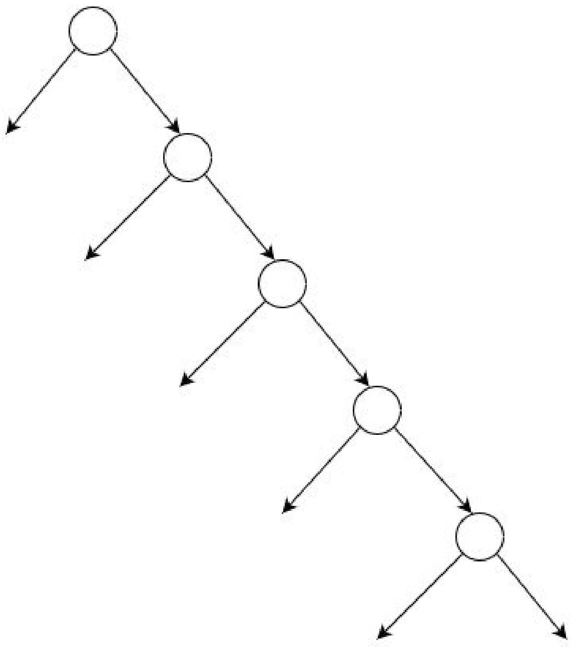
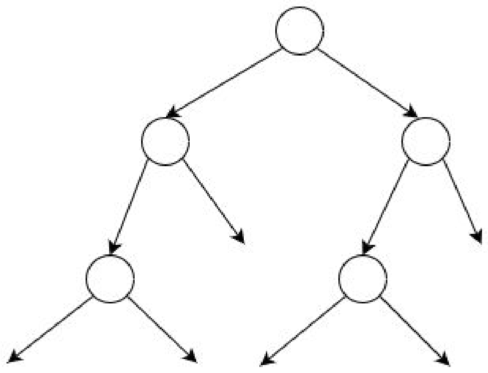
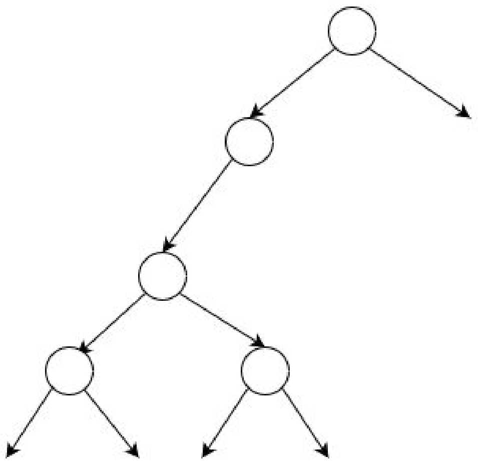
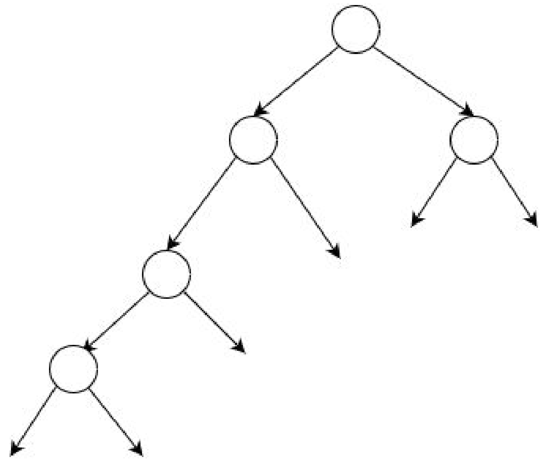
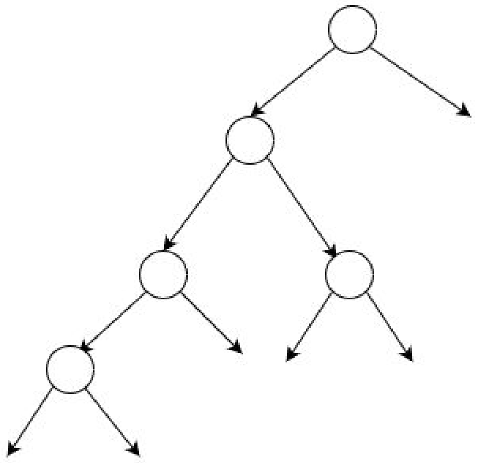
4. #Nodes = 4 [1,1,1,1] ,[1,2,1] ,[1,1,2]



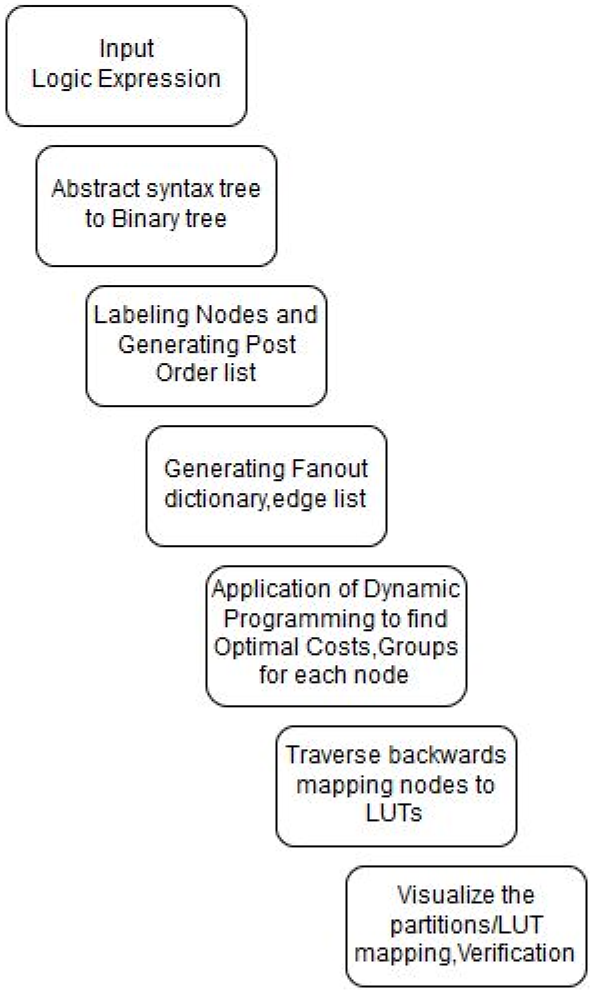




5. #Nodes = 5 [1,1,1,1,1] , [1,2,2] ,[1,2,1,1] ,[1,1,,1,2] ,[1,1,2,1]

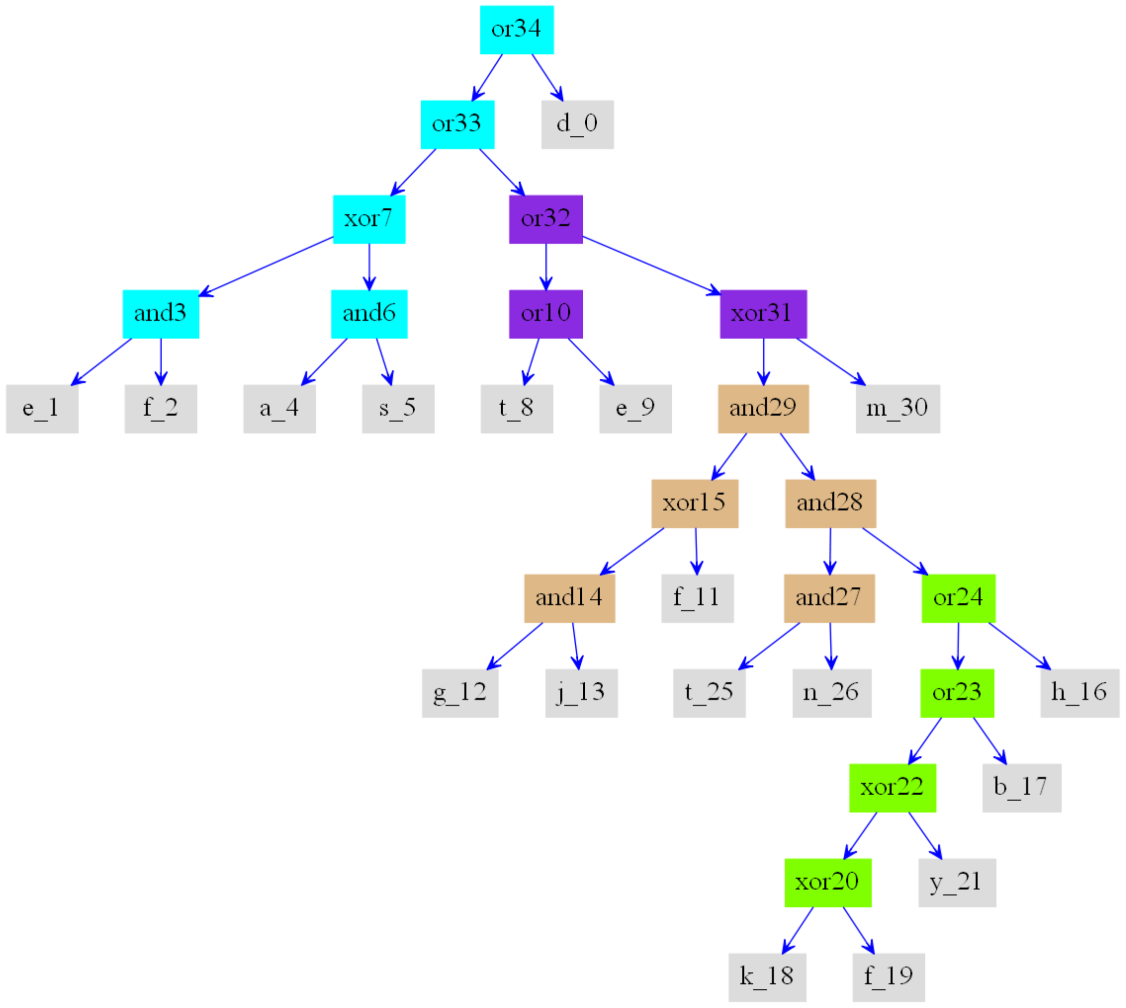
FLOW CHART :(describing major steps involved in solving the problem)


Input
Logic Expression
Abstract syntax tree to Binary tree
Labeling Nodes and Generating Post Order list
Generating Fanout dictionary,edge list
Application of Dynamic Programming to find Optimal Costs,Groups for each node
Traverse backwards mapping nodes to LUTs
Visualize the partitions/LUT mapping,Verification

RESULTS: (Test Cases)

1. Input Expression:

'd|(e&f)^(a&s)|(t|e)|(f^g&j)&(h|b|(k^f)^y)&(t&n)^m'



OUTPUT:

Total Cost : 4

LUT Mappings:

LUT1: ['or34', 'or33', 'xor7', 'and3', 'and6']

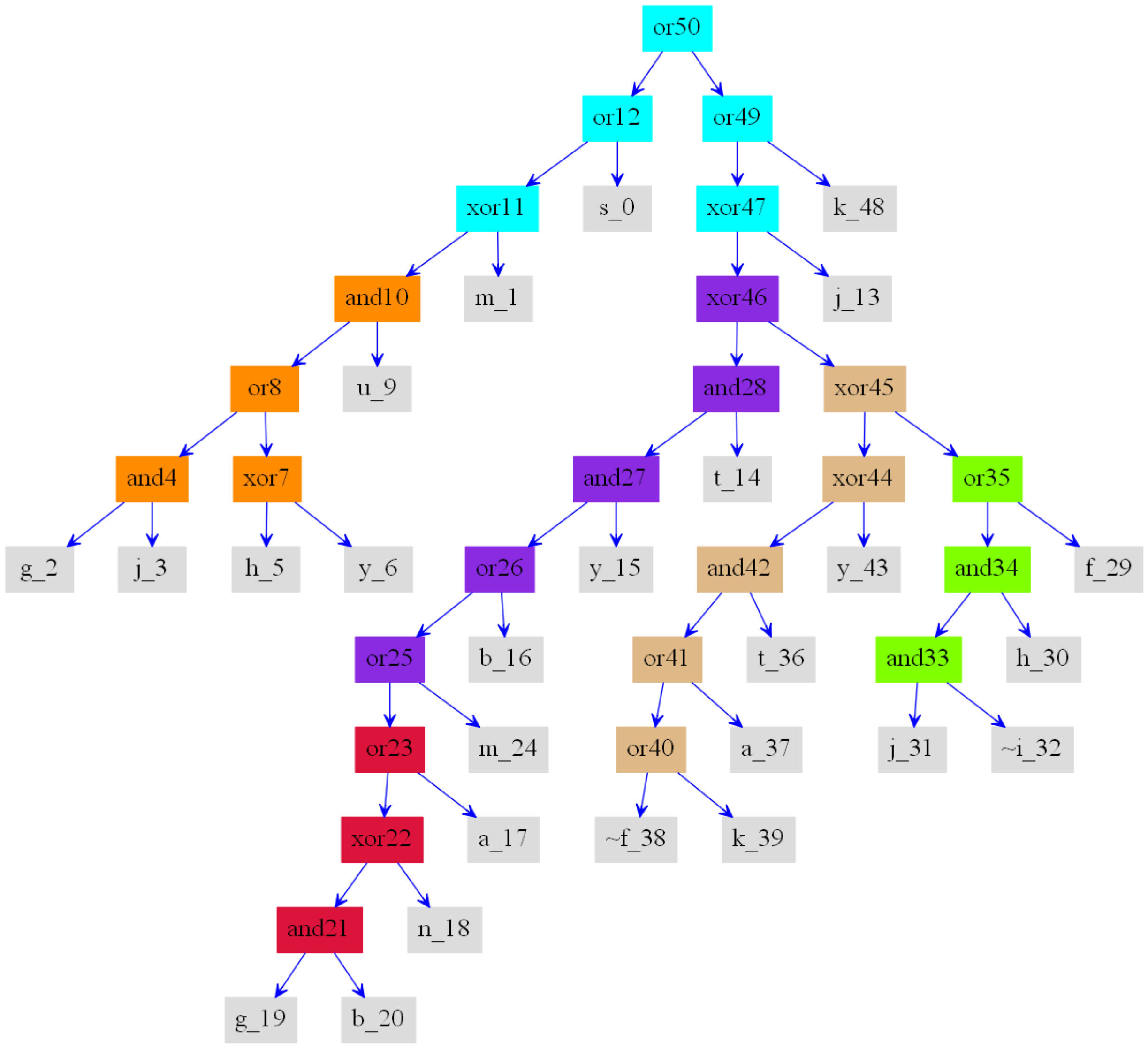
LUT2: ['or32', 'or10', 'xor31']

LUT3: ['and29', 'xor15', 'and28', 'and14', 'and27']

LUT4: ['or24', 'or23', 'xor22', 'xor20']

2. Input Expression:

'(s|m^(g&j|h^y)&u)|j^t&y&(b|(a|n^g&b)|m)^(f|h&j&~i)^t&(a|~f|k)^y|k'



OUTPUT:

Total Cost : 6

LUT Mappings :

LUT1: ['or50', 'or12', 'or49', 'xor11', 'xor47'] **LUT2: ['xor46', 'and28', 'and27', 'or26', 'or25']**

LUT3: ['xor45', 'xor44', 'and42', 'or41', 'or40'] **LUT4: ['or35', 'and34', 'and33']**

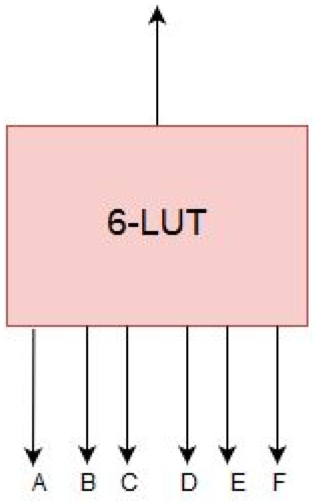
LUT5: ['or23', 'xor22', 'and21'] **LUT6: ['and10', 'or8', 'and4', 'xor7']**

Also , the output contains (with reference to the following diagram):

1.Logic expression implemented by each LUT

2.Bits to be filled in each LUT

3.Connection of inputs with the LUTs and interconnection of the LUTs



SOURCE CODE:

https://github.com/agkonale/EE677

REFERENCES:

1. Hachtel & Somenzi :Logic Synthesis and Verification Algorithms

2. http://interactivepython.org/runestone/static/pythonds/index.html