A Project Report on:

“Displaying the cofactors and the ROBDDs of any given SOP form of Boolean Expression using Python”

Submitted by:

Shruti Masand(181EC245)

Vishal Shukla (171EC252)

Under the guidance of

Prof. M S Bhat

Department of ECE, NITK Surathkal

in partial fulfilment for the award of the degree
of
Bachelor of Technology
in
Electronics and Communications
at

Department of in Electronics and Communications National Institute of Technology Karnataka, Surathkal

November 2020

TABLE OF CONTENTS

INTRODUCTION

Given a Boolean function F of n variables x1, x2, ..., xn,
F : {0,1}n -> {0,1}.
Suppose we define new Boolean functions of n-1 variables as follows:
Fx1 (x2, ..., xn) = F(1, x2, x3, ..., xn)Fx1'(x2, ..., xn) = F(0, x2, x3, ...,xn)
Fx1and Fx1'are called the **cofactors of F**.
They can also be found with respect to more number of variables.

A Binary Decision Diagram(BDD) is a data structure that is used to represent a Boolean function, directly derivable from Shannon's Expression.
A Reduced Ordered Binary Decision Diagram is called a ROBDD.
The ROBDD is a *canonical* form, which means that given an identical ordering of input variables, equivalent Boolean functions will always reduce to the same ROBDD. This is a very desirable property for determining formal equivalence.

PROBLEM STATEMENT/ OBJECTIVES

Given a SoP form of Boolean expression, write a program (in python/C/Java/TclTk/Pearl) to

1. Find the cofactors of the function with respect to any given variable or a set of variables (a, ab ab' etc) and display the results.

2. For a given variable ordering, find and display the ROBDD of the function.

Assumption:

1. You are free to assume the way you want to input the Boolean expression – from keyboard, file etc. Product terms can also be given as one per line. It is left to you to decide how to feed the input SoP.

2. Display of ROBDD also can be either in a graph format or in any other way which you specify. Like, display the nodes in Depth First Search (DFS) or Breadth First Search (BFS) etc format. Again, left to your choice.

3. Whichever program takes more variables and gives correct results and displays correctly will be the benchmark for evaluation

METHODOLOGY

Q1) In order to find the Cofactors, we use the following cofactors() function related to the iter_cofactors function:

```
  def cofactors(self, vs=None):
    r"""Return a tuple of the cofactors of a function over N variables.

    The *vs* argument is a sequence of :math:`N` Boolean variables.

    The *cofactor* of :math:`f(x_1, x_2, \dots, x_i, \dots, x_n)`
    with respect to variable :math:`x_i` is:
    :math:`f_{x_i} = f(x_1, x_2, \dots, 1, \dots, x_n)`

    The *cofactor* of :math:`f(x_1, x_2, \dots, x_i, \dots, x_n)`
    with respect to variable :math:`x_i'` is:
    :math:`f_{x_i'} = f(x_1, x_2, \dots, 0, \dots, x_n)`
    """
    return tuple(cf for cf in self.iter_cofactors(vs))

def iter_cofactors(self, vs=None):
    r"""Iterate through the cofactors of a function over N variables.

    The *vs* argument is a sequence of :math:`N` Boolean variables.

    The *cofactor* of :math:`f(x_1, x_2, \dots, x_i, \dots, x_n)`
    with respect to variable :math:`x_i` is:
    :math:`f_{x_i} = f(x_1, x_2, \dots, 1, \dots, x_n)`

    The *cofactor* of :math:`f(x_1, x_2, \dots, x_i, \dots, x_n)`
    with respect to variable :math:`x_i'` is:
    :math:`f_{x_i'} = f(x_1, x_2, \dots, 0, \dots, x_n)`
    """
    vs = self._expect_vars(vs)
    for point in iter_points(vs):
      yield self.restrict(point)
```

Q2) In order to form the Reduced Ordered BDDs we use the following function

```
def bdd2expr(bdd, conj=False):
      """Convert a binary decision diagram into an expression.

      This function will always return an expression in two-level form.
      If conj is ``False``, return a sum of products (SOP).
```

```
    Otherwise, return a product of sums (POS).

    For example::

    >>> a, b = map(bddvar, 'ab')
    >>> bdd2expr(~a | b)
    Or(~a, And(a, b))
    """
    if conj:
    outer, inner = (And, Or)
    paths = _iter_all_paths(bdd.node, BDDNODEZERO)
    else:
    outer, inner = (Or, And)
    paths = _iter_all_paths(bdd.node, BDDNODEONE)
    terms = list()
    for path in paths:
    expr_point = {exprvar(v.names, v.indices): val
        for v, val in _path2point(path).items()}
    terms.append(boolfunc.point2term(expr_point, conj))
    return outer(*[inner(*term) for term in terms])
```

The program for graph class is as follows:

```
class Source(File):
    @classmethod
    def from_file(cls, filename, directory=None,
        format=None, engine=None, encoding=ENCODING):
    """Return an instance with the source string read from the given file.

    Args:
    filename: Filename for loading/saving the source.
    encoding: Encoding for loading/saving the source.
    """
    filepath = os.path.join(directory or '', filename)
    if encoding is None:
    encoding = locale.getpreferredencoding()
    log.debug('read %r with encoding %r', filepath, encoding)
    with io.open(filepath, encoding=encoding) as fd:
    source = fd.read()
    return cls(source, filename, directory, format, engine, encoding)


    def _init_(self, source, filename=None, directory=None,format=None, engine=None,
encoding=ENCODING):
```

```
super(Source, self)._init_(filename, directory,format, engine, encoding)
self.source = source  #: The verbatim DOT source code string.

def _kwargs(self):
result = super(Source, self)._kwargs()
result['source'] = self.source
return result
```

SIMULATION/RESULTS

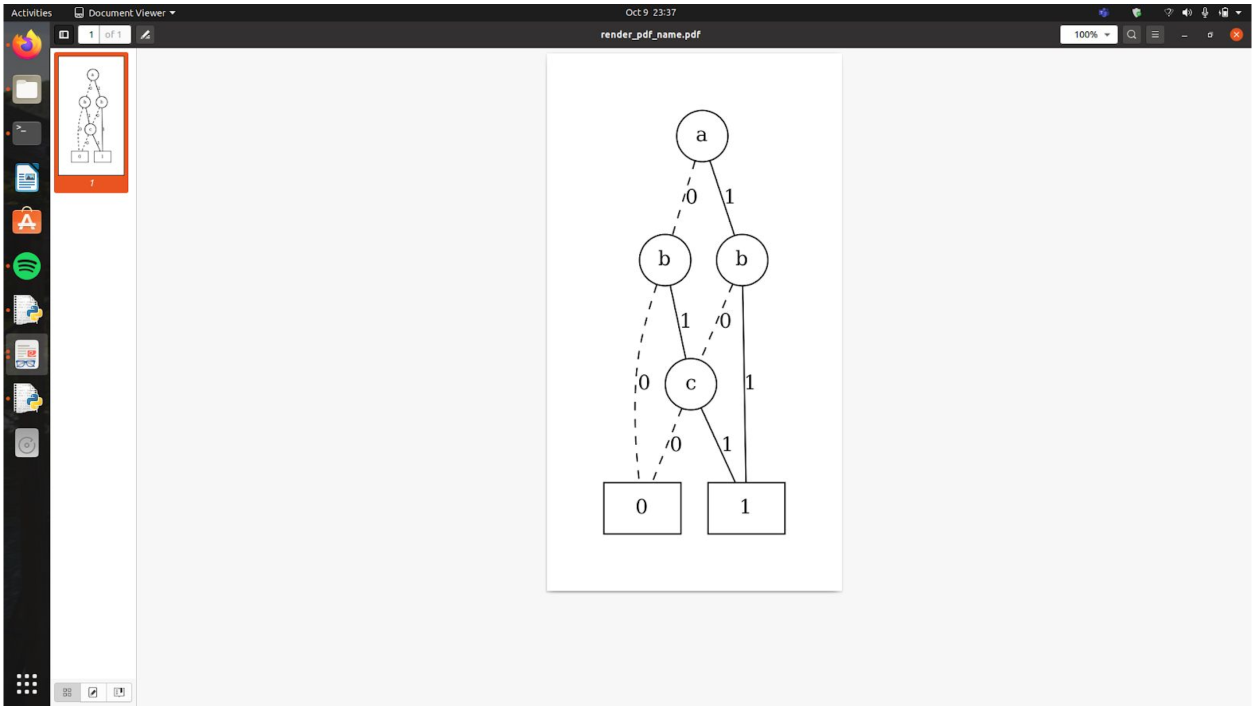
PYTHON CODE -

```
######################LST Project - SHRUTI MASAND-181EC245, VISHAL
SHUKLA-171EC252######################

#Import all libraries required for the code to function properly

#Pyeda library for implementing the data structures and algorithms
#necessary for performing logic synthesis and verification

import pyeda
from pyeda.boolalg.boolfunc import *
from pyeda.boolalg.expr import exprvar
from pyeda.inter import *
from pyeda.boolalg.bfarray import exprvars

#Graphviz library for using the graphviz tool in order to
#visualise the constructed ROBDD

from graphviz import Source

#cofactQ1 function for calculating cofactors

def cofactQ1(expression, variable):
    tmp = Function
    t = tmp.cofactors(expression, variable)
    return t

#bdd function for calculating and displaying the ROBDD

def bdd(expression):
    # a, b, c, d = map(bddvar, 'abcd')
    #f = expression
    f = expr2bdd(expression)
    gv = Source(f.to_dot())
    gv.render('render_pdf_name',view=True)

#define_var function for defining the variables as boolean variables

def define_var(var_array):
    for i in var_array:
        #r = var_array[i]
        #print(exprvar(r))
        #s = tuple(i)
        #print(s)
        r = exprvar(str(i))
        print(r)


f = Function()

######################################### INPUT THE FUNCTION HERE
###########################################

expression = '(a&b)|(b&c)|(a&c)'
f = expr(expression)

######################################## DEFINE THE VARIABLES HERE
##########################################

a, b, c = map(exprvar, 'abc')

#####################################################################################
########################

varlist = sorted(list(f.support))
print(varlist,'\n',f)

var_new = []
var_dict = dict()
for i in varlist:
    var_new.append(str(i))
#print(var_new)

for i in var_new:
    var_dict[i] = str(i)
```

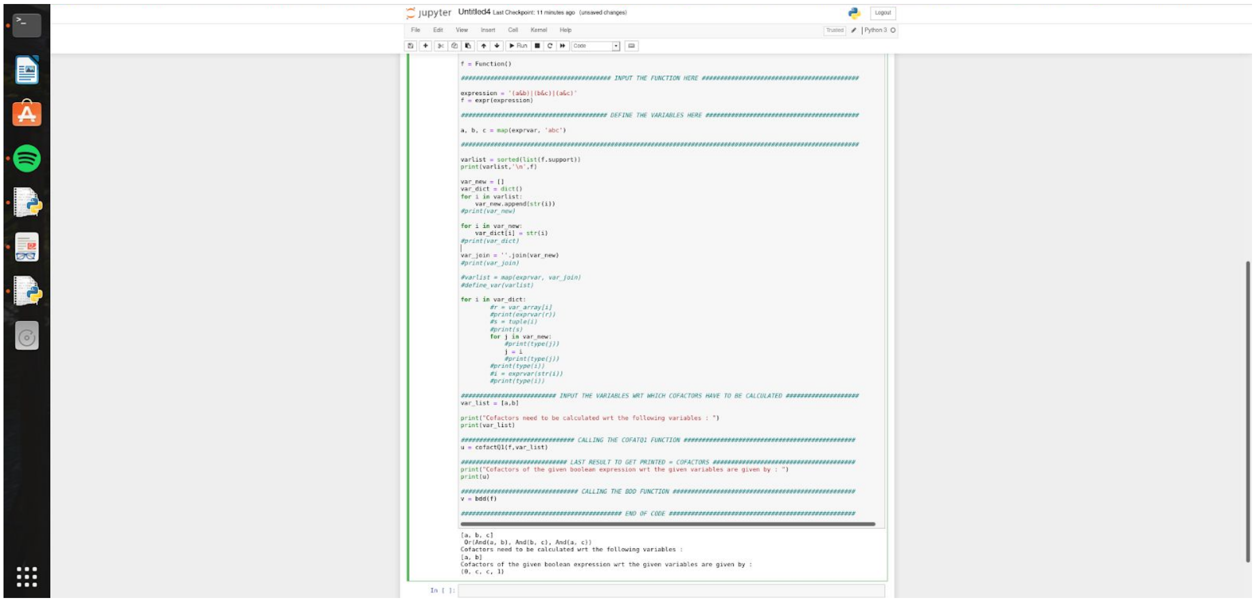
```
#print(var_dict)

var_join = ''.join(var_new)
#print(var_join)

#varlist = map(exprvar, var_join)
#define_var(varlist)

for i in var_dict:
    #r = var_array[i]
    #print(exprvar(r))
    #s = tuple(i)
    #print(s)
    for j in var_new:
      #print(type(j))
      j = i
      #print(type(j))
    #print(type(i))
    #i = exprvar(str(i))
    #print(type(i))

############################ INPUT THE VARIABLES WRT WHICH COFACTORS HAVE TO
BE CALCULATED #####################

var_list = [a,b]

print("Cofactors need to be calculated wrt the following variables : ")
print(var_list)

################################# CALLING THE COFATQ1 FUNCTION
###################################################
u = cofactQ1(f,var_list)

############################### LAST RESULT TO GET PRINTED = COFACTORS
##########################################
print("Cofactors of the given boolean expression wrt the given variables are given by : ")
print(u)

################################### CALLING THE BDD FUNCTION
######################################################
v = bdd(f)

################################################ END OF CODE
#######################################################
```

RESULTS OBTAINED FOR THE GIVEN INPUTS -

ROBDD FOR (A&B)|(B&C)|(C&A) :



CODE RUNNING WITHOUT ERRORS GIVING (0,C,C,1) AS THE COFACTORS OF THE ABOVE FUNCTION WITH RESPECT TO THE VARIABLES [A,B] -


jupyter Untitled4 Last Checkpoint: 11 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
Run
Code

```
f = Function()

########################################## INPUT THE FUNCTION HERE ##########################################

expression = '(a&b)|(b&c)|(a&c)'
f = expr(expression)

######################################### DEFINE THE VARIABLES HERE #########################################

a, b, c = map(exprvar, 'abc')

#############################################################################################################

varlist = sorted(list(f.support))
print(varlist,'\n',f)

var_new = []
var_dict = dict()
for i in varlist:
    var_new.append(str(i))
#print(var_new)

for i in var_new:
    var_dict[i] = str(i)
#print(var_dict)

var_join = ''.join(var_new)
#print(var_join)

#varlist = map(exprvar, var_join)
#define_var(varlist)

for i in var_dict:
        #r = var_array[i]
        #print(exprvar(r))
        #s = tuple(i)
        #print(s)
        for j in var_new:
            #print(type(j))
            j = i
            #print(type(j))
        #print(type(i))
        #i = exprvar(str(i))
        #print(type(i))

######################### INPUT THE VARIABLES WRT WHICH COFACTORS HAVE TO BE CALCULATED ###################
var_list = [a,b]

print("Cofactors need to be calculated wrt the following variables : ")
print(var_list)

############################### CALLING THE COFATQ1 FUNCTION ##############################################
u = cofactQ1(f,var_list)

############################## LAST RESULT TO GET PRINTED = COFACTORS ######################################
print("Cofactors of the given boolean expression wrt the given variables are given by : ")
print(u)

################################ CALLING THE BDD FUNCTION ##################################################
v = bdd(f)

########################################## END OF CODE #####################################################
```

```
[a, b, c]
 Or(And(a, b), And(b, c), And(a, c))
Cofactors need to be calculated wrt the following variables :
[a, b]
Cofactors of the given boolean expression wrt the given variables are given by :
(0, c, c, 1)
```

In [ ]:

CONCLUSION

The Cofactors for several variables and the corresponding BDDs are calculated using several functions/utilities of the PYEDA library. PyEDA is primarily concerned with implementing the data structures and algorithms necessary for performing logic synthesis and verification. These tools form the theoretical foundation for the implementation of CAD tools for designing VLSI.
It is free software; you can use it or redistribute it under the terms of the “two-clause” BSD License. It’s source code can be seen on GitHub.

REFERENCES

https://pyeda.readthedocs.io/en/latest/install.html
https://pyeda.readthedocs.io/en/latest/bdd.html
https://pyeda.readthedocs.io/en/latest/boolalg.html?highlight=cofactor#pyeda-variable-function-base-classes
https://pyeda.readthedocs.io/en/latest/_modules/pyeda/boolalg/boolfunc.html#Function.iter_cofactors
https://pyeda.readthedocs.io/en/latest/overview.html
https://docs.python.org/2.5/lib/ctypes-pointers.html