A PROJECT REPORT

ON

Binary Decision Diagrams in Designing Digital Logic Circuits using Optical Components

By

Deepansh Nagaria(Enroll No. 17114024), Abhishek Kumar(Enroll No. 17114005) and Hemant Singh(Enroll No. 17114038)
(B.Tech 1st Year, Computer Science & Engineering, Indian Institute of Technology Roorkee)

Dr. Sudip Roy
(Assistant Professor, Department of CSE, Indian Institute of Technology Roorkee, Uttarakhand)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
UTTARAKHAND - INDIA

08 MAY- 10 JUNE, 2018

CONTENTS

CERTIFICATE

This is to certify that **Deepansh Nagaria** (B.Tech 1st Year, CSE , IIT Roorkee, Enroll No. 17114024), **Hemant Singh** (B.Tech 1st Year, CSE , IIT Roorkee, Enroll No. 17114038) and **Abhishek Kumar** (B.Tech 1st Year, CSE , IIT Roorkee, Enroll No. 17114005) did a Summer Research project on **Binary Decision Diagrams in Designing Digital Logic Circuits using Optical Components** (BDD Data Structure, its optimization techniques and its implementation using CUDD library in C) during the Summer break(08 May 2018 – 10 June 2018) under the guidance of **Dr. Sudip Roy** (Assistant Professor, Department of Computer Science, Indian Institute of Technology Roorkee).

SIGNATURE

(Dr. Sudip Roy)

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude to professor **Dr. Sudip Roy** who guided us through the project and made it quite a great learning experience. We would also like to thank him for providing us with required resources and giving access to CoDA Lab. We would also like to thank **Mr. Arun Kant Dwivedi** and **Ms. Sumit Sharma** for his guidance, support and encouragement throughout the project. At last we would like to thank all other people who directly or indirectly helped us during this project .

Deepansh Nagaria (Enroll.No. – 17114024)

Hemant Singh (Enroll. No. – 17114038)

Abhishek Kumar (Enroll. No. – 17114005)

-B.Tech 1st Year, Computer Science and Engineering

-Indian Institute of Technology Roorkee

ABSTRACT

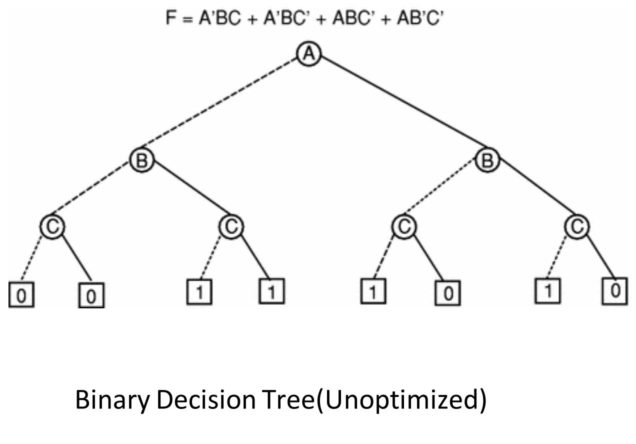
A Binary Decision Diagram (BDD) is a directed acyclic graph that consists of nodes and edges. It deals with boolean functions consisting of a set of decision nodes starting with root node at the top. Each decision nodes consists of two outgoing branches represented by solid(high) and dotted(low) lines. its optimization is based on several steps that includes sharing of nodes, removing of redundant nodes and merging duplicate nodes. CUDD library in C provides us with tools to visualize boolean functions as binary and algebraic decision diagrams (BDD and ADD). Graphviz is a software which works in sync with CUDD to provide us tools to view a DOT format file produced as result of CUDD's Optimization of boolean functions. We have worked on both combinational and optical circuits implementing BDD optimization using CUDD. In domain of combinational circuits which is quite a saturated field we implemented single and multiple output functions with both optimized and unoptimized BDD. In the domain of optical circuits, this is particularly interesting for the number of gates and the effect of so-called splitters to the signal strength. In this work, we investigate this relation by considering a variety of (existing as well as proposed) synthesis approaches for optical circuits. Our investigations show that reducing the number of gates and reducing the number of splitters are contradictory optimization objectives. Furthermore, the performance of synthesis guided with respect to gate efficiency as well as synthesis guided with respect to splitter freeness is evaluated and an overhead factor between the contradictory metrics is experimentally determined.

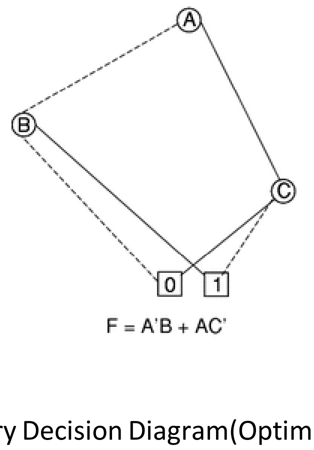
Key Words- BDD ,optimization, CUDD , combinational logic circuits, optical circuits, splitters, splitter free circuits.

INTRODUCTION

A binary decision diagram (BDD) is a mean to represent, analyse, test, and implement Boolean functions. It is a directed acyclic graph that consists of nodes and edges . Although other methods may be used to complete such tasks, e.g. the Karnaugh map, binary decision diagrams offer some useful advantages . Karnaugh maps and truth tables are suitable methods that may be used to describe functions consisting of a small number of variables . However, the problem with such methods is that the size of these structures increases dramatically as the number of variables increase, i.e., 2^n rows in a truth table or squares in a Karnaugh map are required for a function of *n* variables. Although a binary decision diagram contains 2^n nodes for *n* variables, it is possible to reduce the size of these structures by following certain algorithms. For instance, the order in which variables are evaluated within a binary decision diagram can significantly affect the size of the structure. By implementing ordering restrictions algorithms, a more efficient manipulation of the diagrams is possible . This additional ordering also allows for a reduction technique to be applied to the ordered binary decision diagram to remove redundancies from the data structure and therefore produce a more compact representation of the Boolean expression. Such structures are known as reduced ordered binary decision diagrams (ROBDD). In addition, the reduced ordered binary decision diagrams provide a unique representation of a Boolean function . By implementing such ordering restrictions algorithms, a more efficient manipulation of the diagrams is possible . This additional ordering also allows for a reduction technique to be applied to the ordered binary decision diagram to remove redundancies from the data structure and therefore produce a more compact representation of the Boolean expression. Such structures are known as reduced ordered binary decision diagrams (ROBDD). In addition, the reduced ordered binary decision diagrams provide a unique representation of a Boolean function.

Optical Circuits are quite relatable with the combinational circuits, using MZI switch for implementation of boolean logic. The correspondence of both the circuits can be seen in the following figure where a BDD's conversion to a optical circuit is shown:





Binary Decision Tree(Unoptimized)

Binary Decision Diagram(Optimized)

In optical circuits the challenge always remains to keep the cost as low as possible. In this sense, particularly the number of gates as well as the number of splitters are of utmost importance.
While the first metric corresponds to the area of the resulting chip, splitters have a significant impact on the signal strength.

We try to address this issue. For this purpose, we consider a variety of (complementary) synthesis approaches for optical circuits that are explicitly based on different function representations—including solutions reviewed above as well as two proposed synthesis methods. As a result of our observations, we can conclude that reducing the number of gates and reducing the number of splitters are contradictory optimization objectives.

More precisely, the performance of synthesis guided with respect to gate efficiency as well as synthesis guided with respect to splitter freeness is evaluated and an overhead factor between these contradictory metrics is determined.

BDD and its OPTIMIZATION

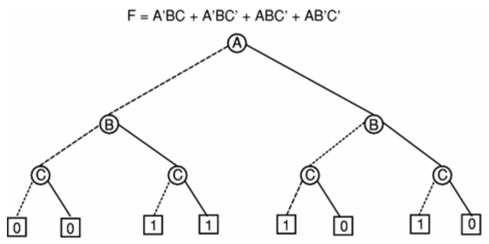
Binary Decision Tree(BDT) –

Binary Decision Trees are trees whose non-terminal nodes are labeled with boolean variables x, y, z, …. and whose terminal nodes are labeled with either 0 or 1.

Binary Decision Diagram (BDD) –

A Binary Decision Diagram (BDD) is a finite DAG with an unique initial node, where

- all terminal nodes are labeled with 0 or 1.
- all non-terminal nodes are labeled with a boolean variable.
- Each non-terminal node has exactly two edges from that node to others; one labeled 0 and one labeled 1; represent them as a dashed line and solid line respectively.

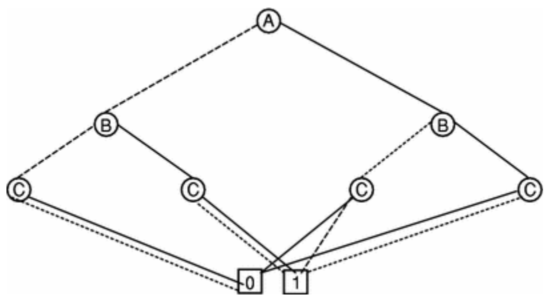


Unoptimized BDD or BDT

BDD Optimization rules

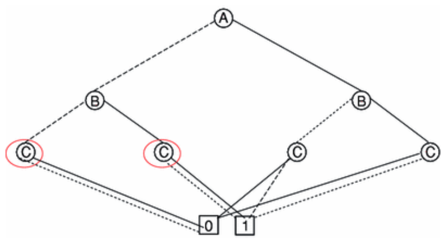
1. **Eliminate duplicate terminals**

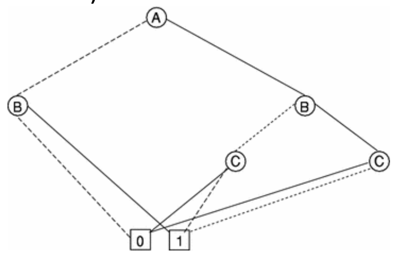
   If a BDD contains more than one terminal 0-node, then we redirect all edges which point to such a0-node to just one of them.
   Similarly, we proceed for nodes labeled with 1.



2. **Eliminate redundant nodes**

   (with both edges pointing to same node)

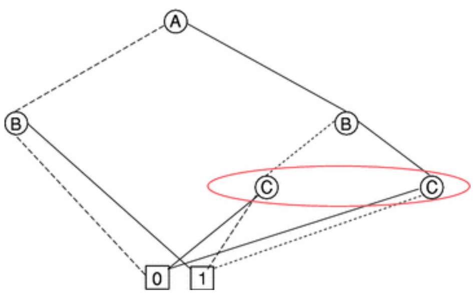




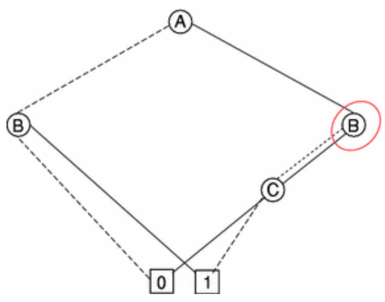
Nodes to be eliminated are marked by *red circles*

Resulting diagram after nodes are eliminated

3. Merge duplicate nodes



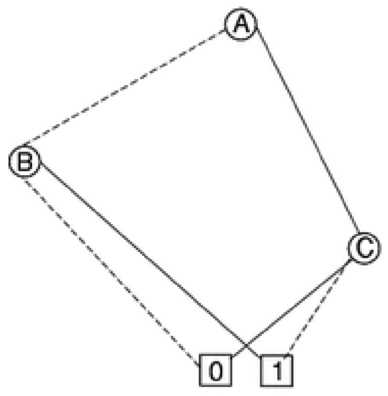
Nodes to be merged are identified



Merging of the two nodes results in another node to be eliminated

Reduced BDD –

A BDD is said to be reduced if no more reductions are possible or no more optimization rules can be applied.



F = A'B + AC'

Reduced or Optimized BDD

Implementation of Boolean Logic functions

Boolean Logic Function –

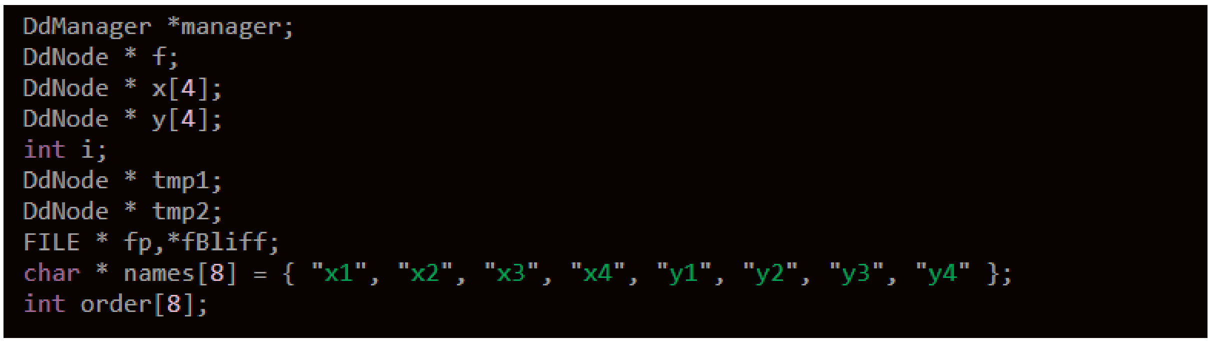
A Boolean function is a mathematical function that maps arguments to value where the allowable values of range(the function arguments) and domain(the function value) are just one of the two values- true and false(0 or 1). The study of boolean function is known as boolean logic. In other words mapping from boolean variables to a boolean value is called Boolean function.

Every boolean function can be represented as a BDD. Although other methods may be used to complete such tasks, e.g., the Karnaugh map, binary decision diagrams offer some useful advantages . Karnaugh maps and truth tables are suitable methods that may be used to describe functions consisting of a small number of variables . However, the problem with such methods is that the size of these structures increases dramatically as the number of variables increase, i.e., 2^n rows in a truth table or squares in a Karnaugh map are required for a function of *n* variables. Although a binary decision diagram contains 2^n nodes for *n* variables, it is possible to reduce the size of these structures by following certain algorithms.

Single Output Functions(1-Output)

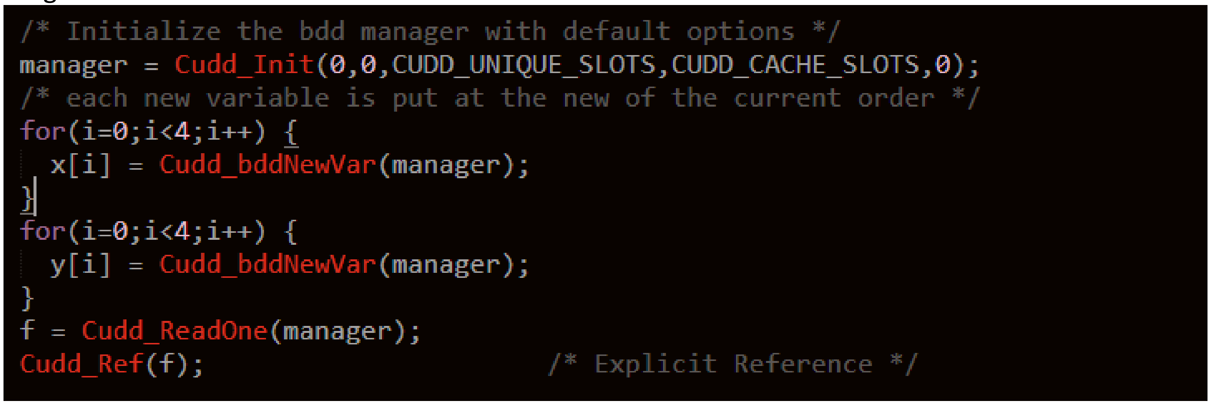
1. For 8 input Xor

Declaration:



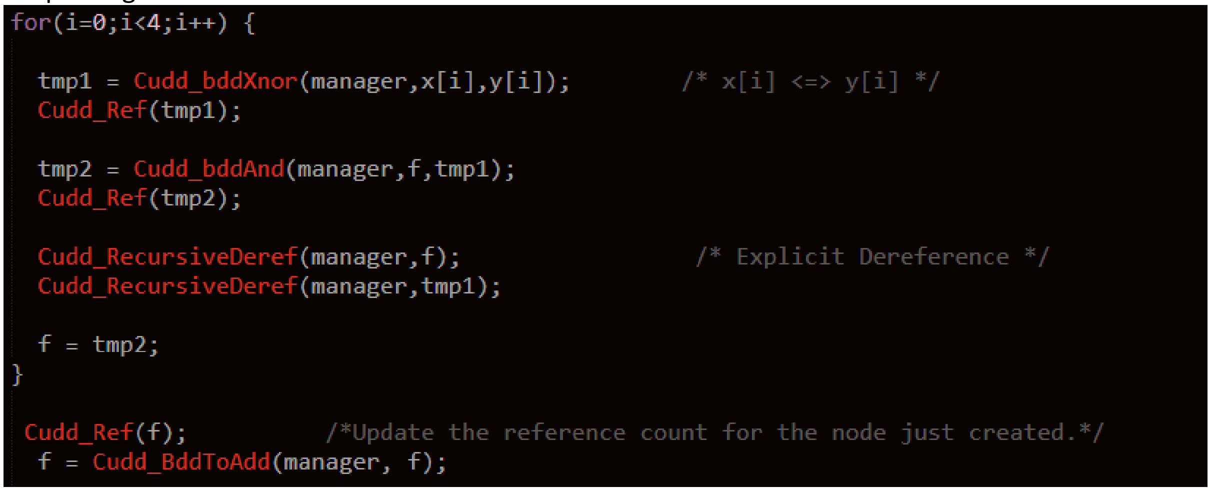
```
DdManager *manager;
DdNode * f;
DdNode * x[4];
DdNode * y[4];
int i;
DdNode * tmp1;
DdNode * tmp2;
FILE * fp,*fBliff;
char * names[8] = { "x1", "x2", "x3", "x4", "y1", "y2", "y3", "y4" };
int order[8];
```

Referencing:



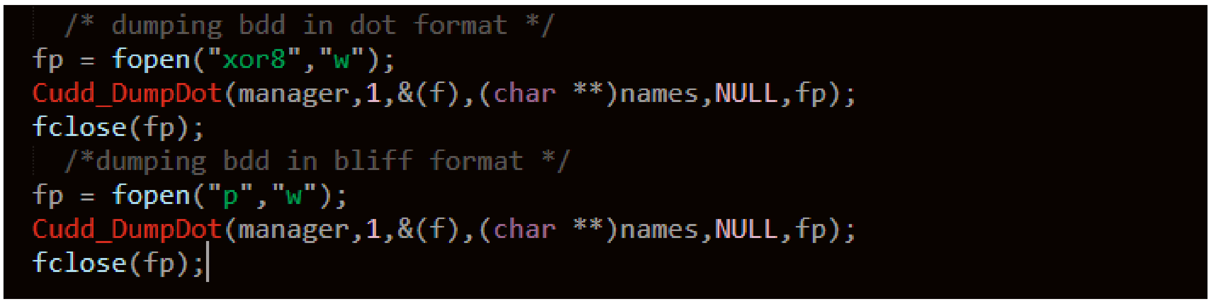
```
/* Initialize the bdd manager with default options */
manager = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
/* each new variable is put at the new of the current order */
for(i=0;i<4;i++) {
  x[i] = Cudd_bddNewVar(manager);
}
for(i=0;i<4;i++) {
  y[i] = Cudd_bddNewVar(manager);
}
f = Cudd_ReadOne(manager);
Cudd_Ref(f);                    /* Explicit Reference */
```

Logic and Updating Reference:

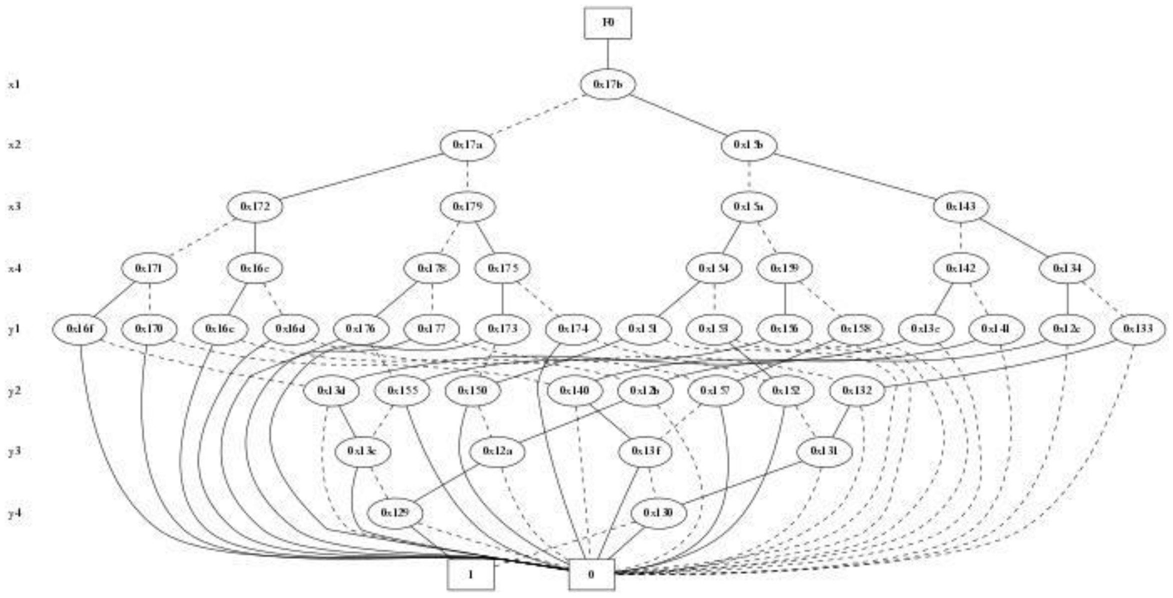


```
for(i=0;i<4;i++) {

  tmp1 = Cudd_bddXnor(manager,x[i],y[i]);        /* x[i] <=> y[i] */
  Cudd_Ref(tmp1);

  tmp2 = Cudd_bddAnd(manager,f,tmp1);
  Cudd_Ref(tmp2);

  Cudd_RecursiveDeref(manager,f);                  /* Explicit Dereference */
  Cudd_RecursiveDeref(manager,tmp1);

  f = tmp2;
}

 Cudd_Ref(f);          /*Update the reference count for the node just created.*/
  f = Cudd_BddToAdd(manager, f);
```

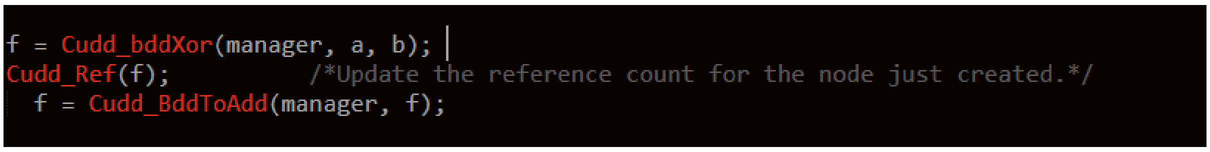
Writing bdd to dot file:



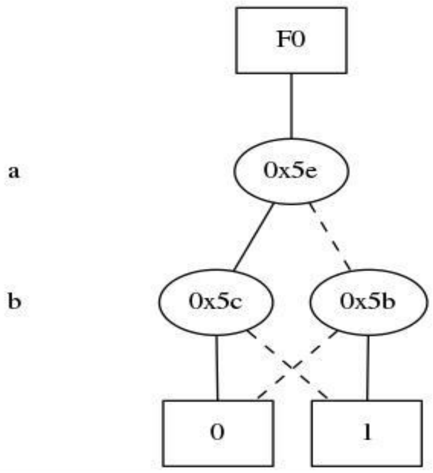
```
   /* dumping bdd in dot format */
 fp = fopen("xor8","w");
 Cudd_DumpDot(manager,1,&(f),(char **)names,NULL,fp);
 fclose(fp);
   /*dumping bdd in bliff format */
 fp = fopen("p","w");
 Cudd_DumpDot(manager,1,&(f),(char **)names,NULL,fp);
 fclose(fp);
```



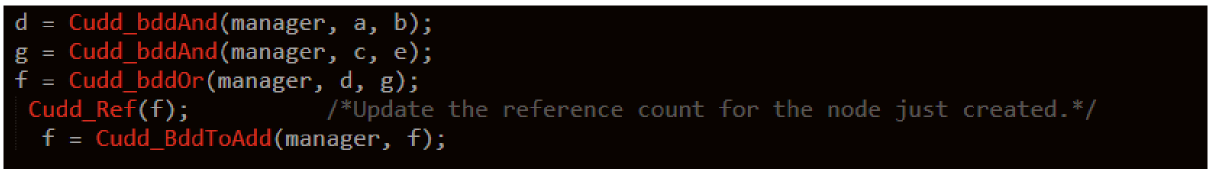
2. For function f=a^b



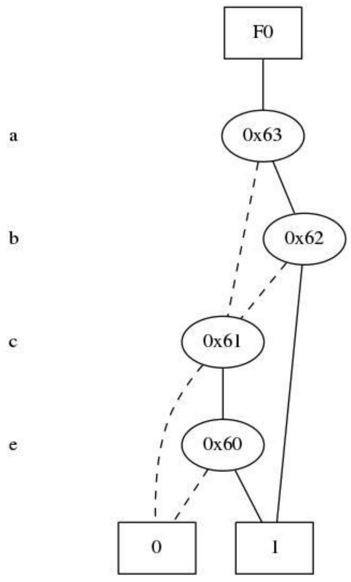
```
f = Cudd_bddXor(manager, a, b);
Cudd_Ref(f);          /*Update the reference count for the node just created.*/
  f = Cudd_BddToAdd(manager, f);
```



3. For function ab+ce



```
d = Cudd_bddAnd(manager, a, b);
g = Cudd_bddAnd(manager, c, e);
f = Cudd_bddOr(manager, d, g);
 Cudd_Ref(f);          /*Update the reference count for the node just created.*/
  f = Cudd_BddToAdd(manager, f);
```

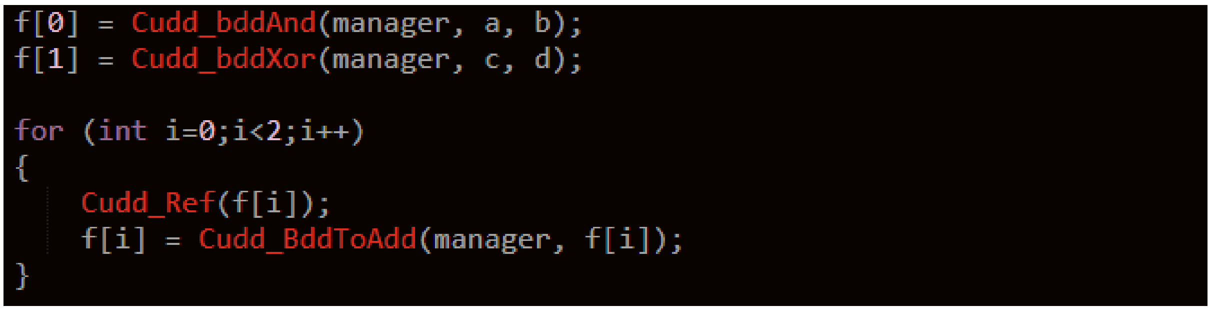


Multiple Output Functions

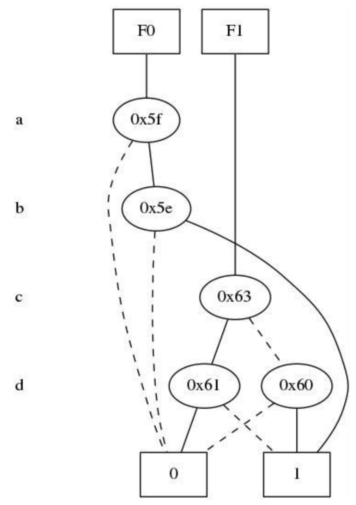
In case of multiple output functions, BDDs constitute an efficient representation for Boolean functions as they represent redundant sub-functions by the same sub-graph. This eventually leads to *shared nodes*, that is, nodes with more than one predecessor. In other words, sometimes we encounter a condition in which there are repetition of nodes not within same function but there is a common node in two or more functions, removal of this node further optimizes the BDD hence sharing of such nodes is done while multiple output functions are implemented.

This property is being showcased in the examples below and the shared nodes are highlighted.

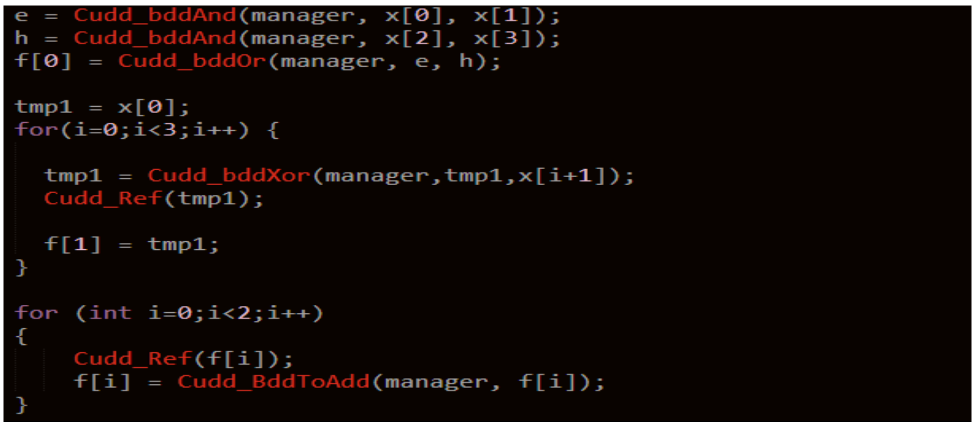
1. f1=a&b and f2=c^d



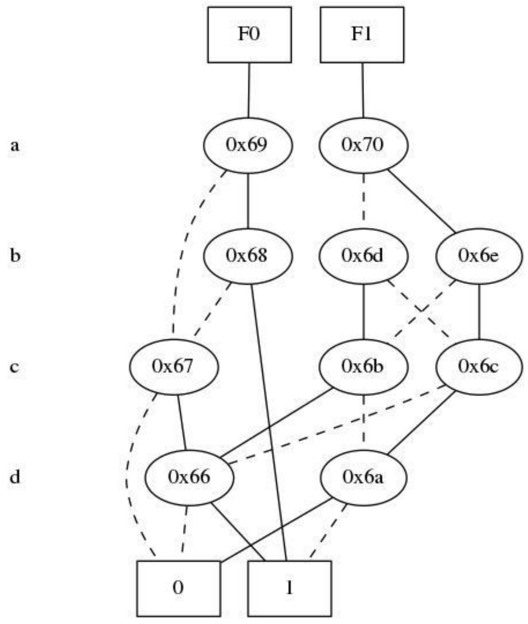
```
f[0] = Cudd_bddAnd(manager, a, b);
f[1] = Cudd_bddXor(manager, c, d);

for (int i=0;i<2;i++)
{
    Cudd_Ref(f[i]);
    f[i] = Cudd_BddToAdd(manager, f[i]);
}
```



2. f0=x0&x1+x2&x3 and f1=x0^x1^x2^x3



```
e = Cudd_bddAnd(manager, x[0], x[1]);
h = Cudd_bddAnd(manager, x[2], x[3]);
f[0] = Cudd_bddOr(manager, e, h);

tmp1 = x[0];
for(i=0;i<3;i++) {

   tmp1 = Cudd_bddXor(manager,tmp1,x[i+1]);
   Cudd_Ref(tmp1);

   f[1] = tmp1;
}

for (int i=0;i<2;i++)
{
    Cudd_Ref(f[i]);
    f[i] = Cudd_BddToAdd(manager, f[i]);
}
```



OPTICAL CIRCUITS

An optical circuit can be derived by traversing the decision diagram in a depth-first fashion and substituting each node with a corresponding crossbar gate. If a shared node occurs, then the optical signal has to be split accordingly. Combining all gates and connecting the respective signals eventually leads to an optical circuit realizing the desired function.

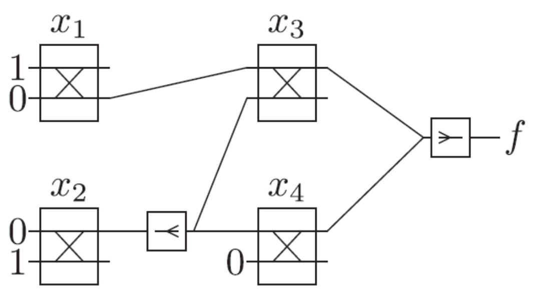
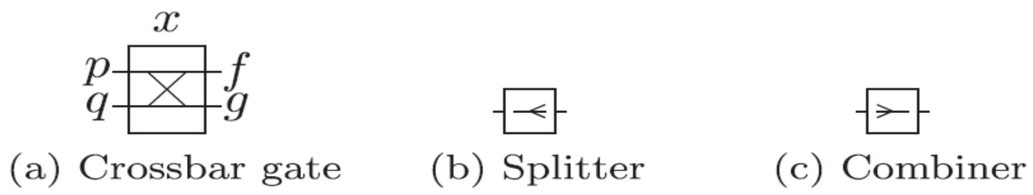


Fig. 2. Optical circuit.

CROSSBAR GATES

The main logic element of an optical circuit is a *crossbar gate*, which routes the optical signals between two parallel paths. The inputs of both paths can be sourced either by light (representing the logical value *true*) or darkness (representing the logical value *false*).2 Furthermore, the routing of both paths is controlled by an electrical signal. The output of each optical signal can be read using optical receivers. Note that, in the above definition, an optical signal cannot directly be used to switch the electrical input of the crossbar gate. For this purpose, an opto-electrical interface would be required that, however, is considered expensive as well as slow. As a result, electrical and optical signals are never assumed to interact with each other except for the crossbar gate.

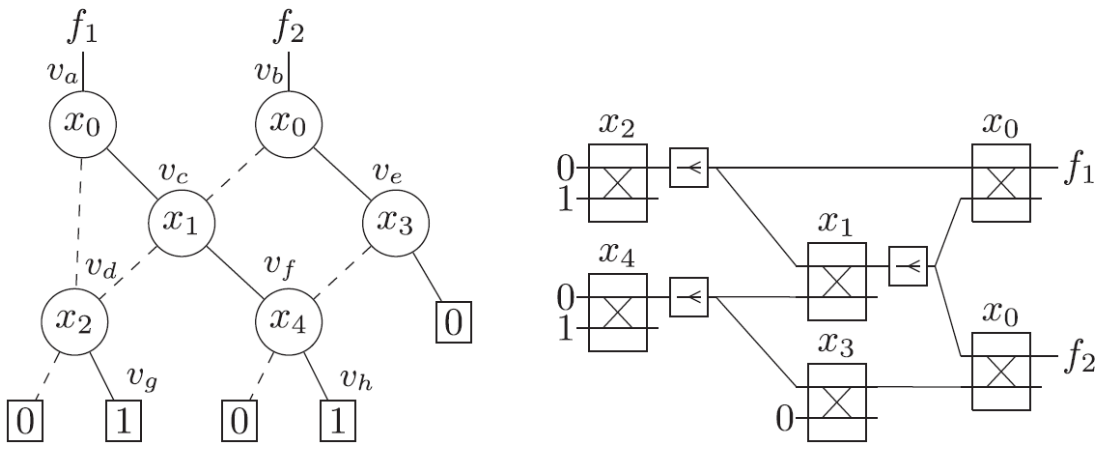


SPLITTERS and COMBINERS

A *splitter* divides an optical signal into two signals—each with only half of the incoming signal power. In contrast, a *combiner* merges two optical signals into a single one and, by this, inherently realizes the OR function. A splitter may have more than two outputs and a combiner may have more than two inputs. Then, in case of a splitter, the strength of the signal is divided by the number of outputs. Furthermore, to keep the model simple, combiners are assumed not to change the signal strength. This makes the worst-case fraction a pessimistic approximation, that is, the actual signal strength may be better than the computed one.

In BDD-based synthesis, the need for splitters is an obvious drawback. Considering practically relevant functions, the BDD representation usually includes a large amount of shared nodes. This directly corresponds to the amount of splitters. Hence, applying this synthesis method leads to optical circuits where certain output signals have a barely noticeable signal strength. On the other hand, sharing allows for a rather compact realization with respect to the number of required gates.

The number of splitters: A splitter causes a considerable decrease in the optical signal strength. Hence, keeping their number as small as possible is an important objective.



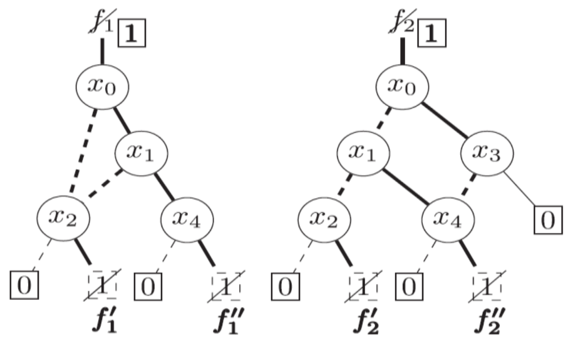
(a) BDD (b) Obtained circuit

SPLITTER FREE SYNTHESIS

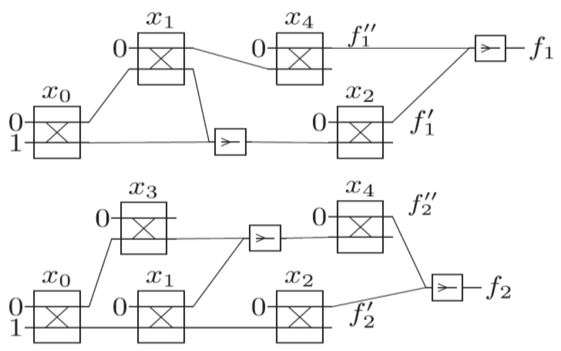
The synthesis approaches reviewed and proposed above allow for an automatic synthesis of rather complex Boolean functionality. Redundancies are kept small (by means of shared nodes in the BDD-based approach or by means of shared products in the synthesis based on two-level descriptions). Hence, the resulting circuits are efficient with respect to the number of gates. But, in contrast, a significant amount of splitters is required. Hence, the resulting circuits may have severe issues with respect to the signal strength. Accordingly, in a second series of investigations, we consider approaches aiming for avoiding splitters at all.

It does not matter whether the paths for a given BDD are traversed from the root to the leaves of the BDD or vice versa. So for splitter free synthesis paths will be traversed from root to leaf with the difference that now combiners rather than splitters are added for shared nodes. This scheme indeed enables BDD-based synthesis without the need for a single splitter. However, problems occur when multi-output functions are considered. Here, it remains unclear whether a terminal corresponds, for example, to a sub-function *f*1, a sub-function *f*2, or both. As a consequence, shared nodes in sub-trees that affect more than one sub-function cannot be handled and have to be explicitly realized. Hence in order to realize multi-output functions we have to consider separate BDD's for each sub function.

Obviously, such a splitter-free synthesis leads to additional redundancy and hence an increase in the number of gates.



(a) BDD(s) for $f_1 = f_1' \vee f_1''$ and $f_2 = f_2' \vee f_2''$



(b) Obtained circuit(s)

Implementation of boolean logic through PLA files

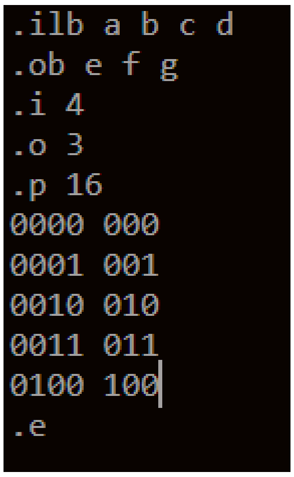
Practically sometimes it is not possible to give input directly to code. To overcome that problem input is given indirectly through files known as benchmarks. Benchmark usually contain edif, fsm, tv ,pla etc. PLA format files are one those files or benchmarks. It usually contains data in the following format:

.lib a b c d e …..	name of the variables
.ob a1 b1 c1 …..	name of the output functions
.i n	n=number of input variables
.o m	m=number of output variables
.p q	q=number of lines containing the input

Here the 'q' lines contains the given output.

After 'q' lines

.e	end



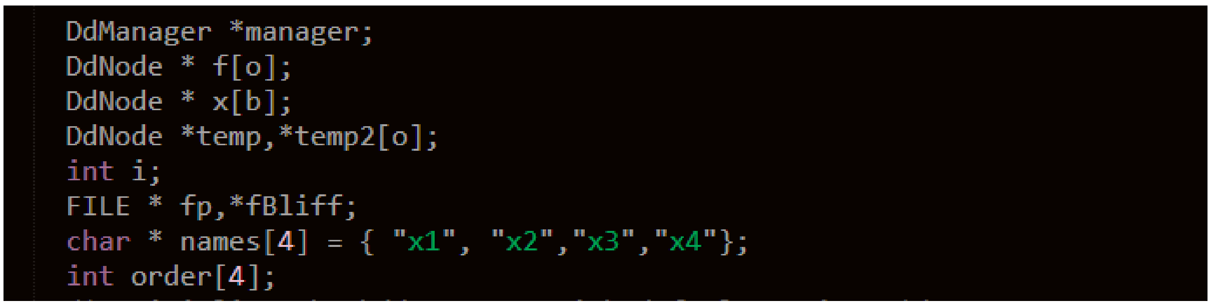
```
.ilb a b c d
.ob e f g
.i 4
.o 3
.p 16
0000 000
0001 001
0010 010
0011 011
0100 100
.e
```

BENCHMARKS

Benchmarks are the files which contain input that is to be given to the CUDD program for displaying the BDD. There can be many format like PLA, EDIF, TV, FSM etc. Following are some of the Benchmarks including both single and multiple output functions, we have implemented:-

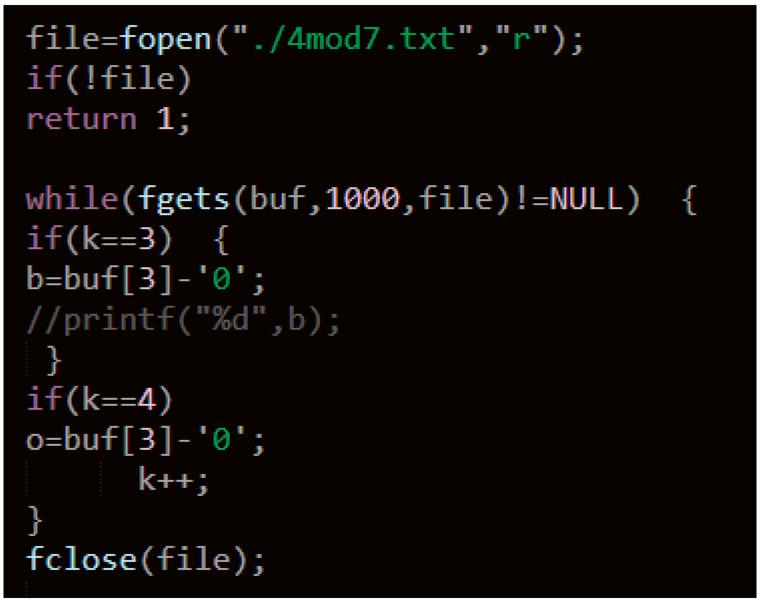
How we operate with the PLA files and the program for that:

Declaration:



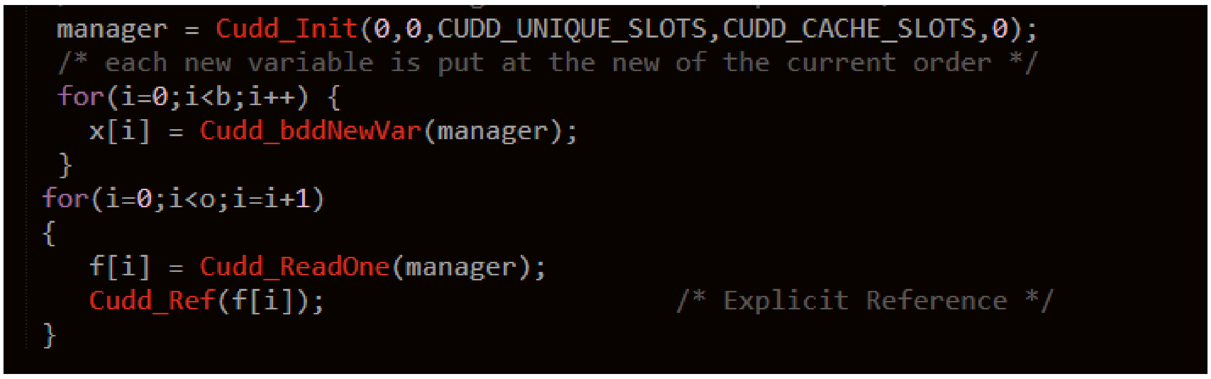
```
DdManager *manager;
DdNode * f[o];
DdNode * x[b];
DdNode *temp,*temp2[o];
int i;
FILE * fp,*fBliff;
char * names[4] = { "x1", "x2","x3","x4"};
int order[4];
```

Taking input from PLA:



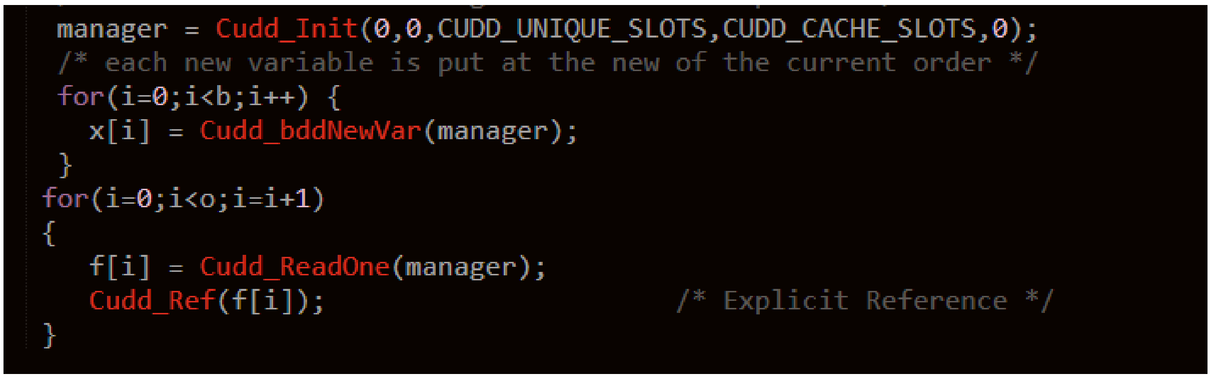
```
file=fopen("./4mod7.txt","r");
if(!file)
return 1;

while(fgets(buf,1000,file)!=NULL)  {
if(k==3)  {
b=buf[3]-'0';
//printf("%d",b);
 }
if(k==4)
o=buf[3]-'0';
          k++;
}
fclose(file);
```

Referencing:



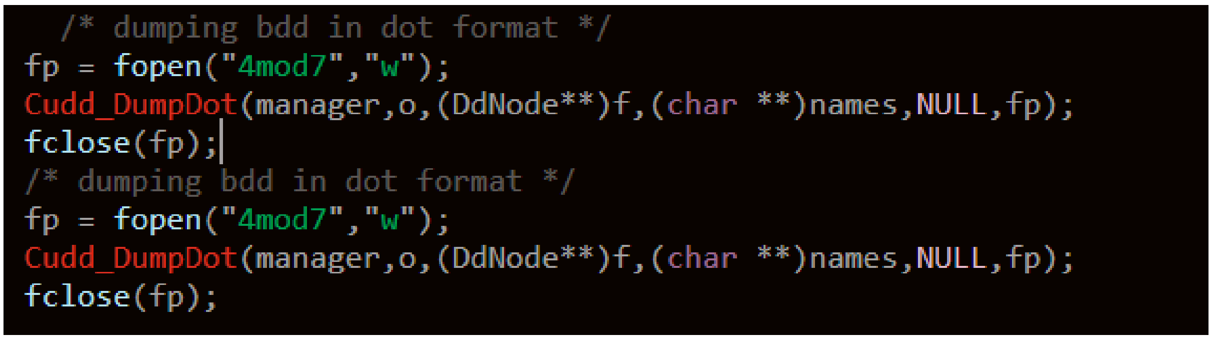
```
  manager = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
  /* each new variable is put at the new of the current order */
  for(i=0;i<b;i++) {
    x[i] = Cudd_bddNewVar(manager);
  }
 for(i=0;i<o;i=i+1)
 {
    f[i] = Cudd_ReadOne(manager);
    Cudd_Ref(f[i]);                     /* Explicit Reference */
 }
```

Logic and Updating Reference:



```
  manager = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
  /* each new variable is put at the new of the current order */
  for(i=0;i<b;i++) {
    x[i] = Cudd_bddNewVar(manager);
  }
 for(i=0;i<o;i=i+1)
 {
    f[i] = Cudd_ReadOne(manager);
    Cudd_Ref(f[i]);                     /* Explicit Reference */
 }
```

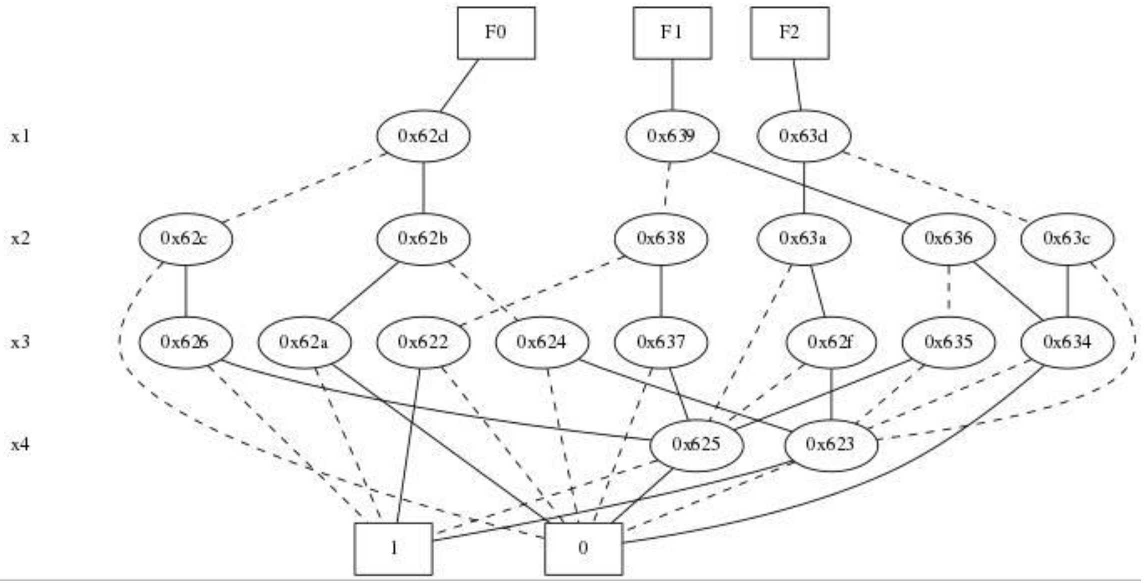
Writing output bdd to dot file:



```
  /* dumping bdd in dot format */
fp = fopen("4mod7","w");
Cudd_DumpDot(manager,o,(DdNode**)f,(char **)names,NULL,fp);
fclose(fp);
/* dumping bdd in dot format */
fp = fopen("4mod7","w");
Cudd_DumpDot(manager,o,(DdNode**)f,(char **)names,NULL,fp);
fclose(fp);
```

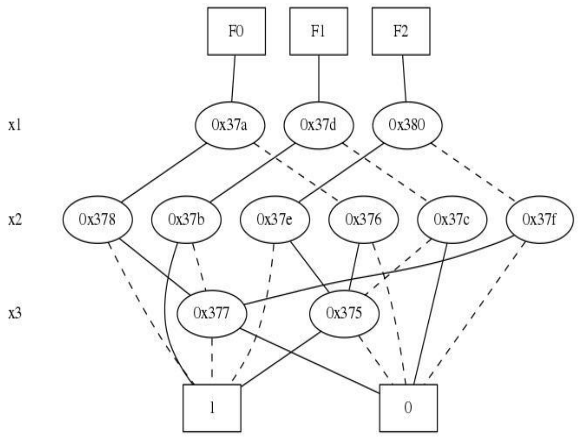
1. **4mod7:** .i = 4 .o = 3 .p = 16

BDD:



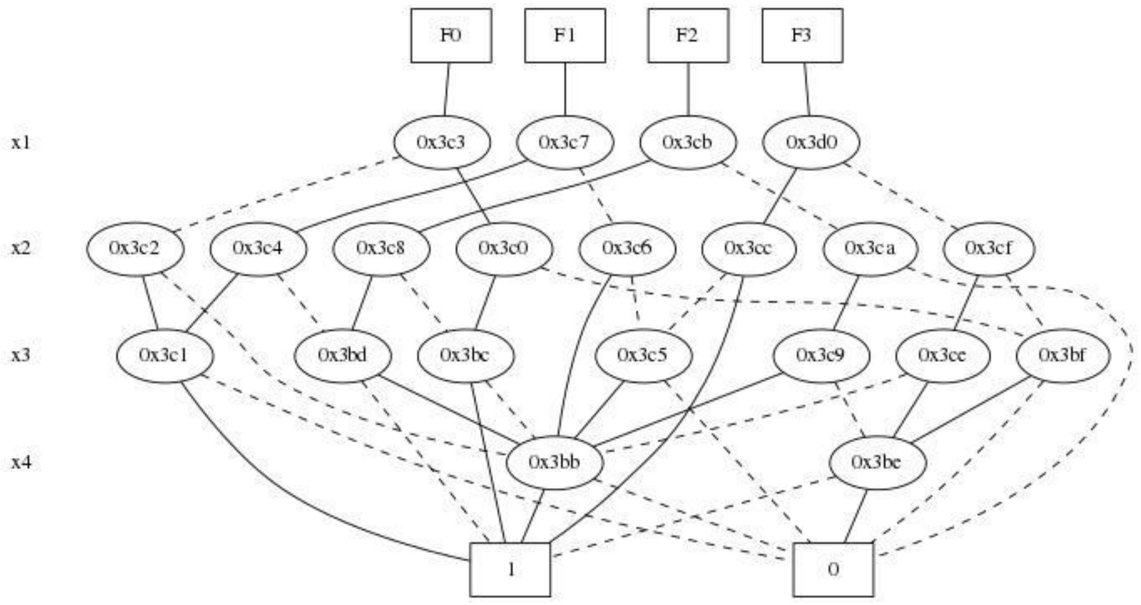
2. **ham3_28:** .i = 3 .o = 3 .p = 8

BDD:



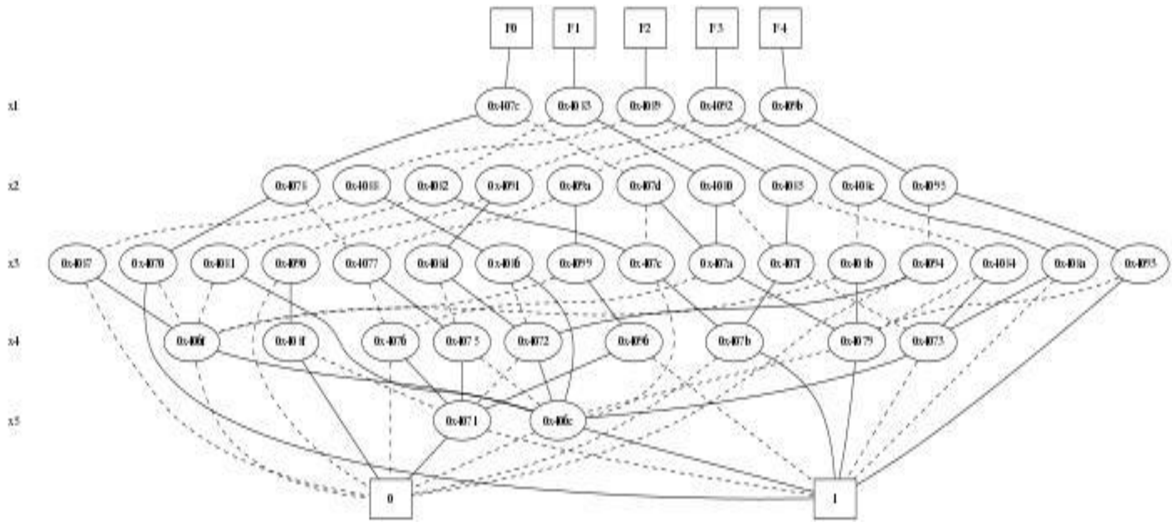
3. **hwb4_12:** .i = 4 .o = 4 .p = 16

BDD:



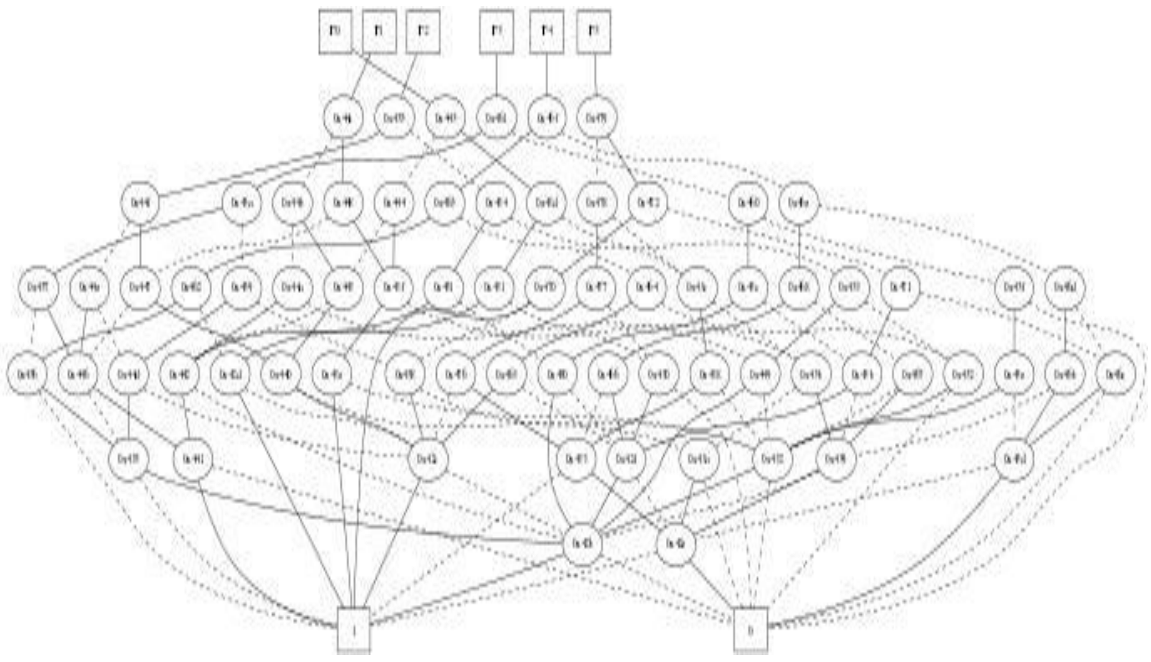
4. **hwb5_13:** .i = 5 .o = 5 .p = 33

BDD:



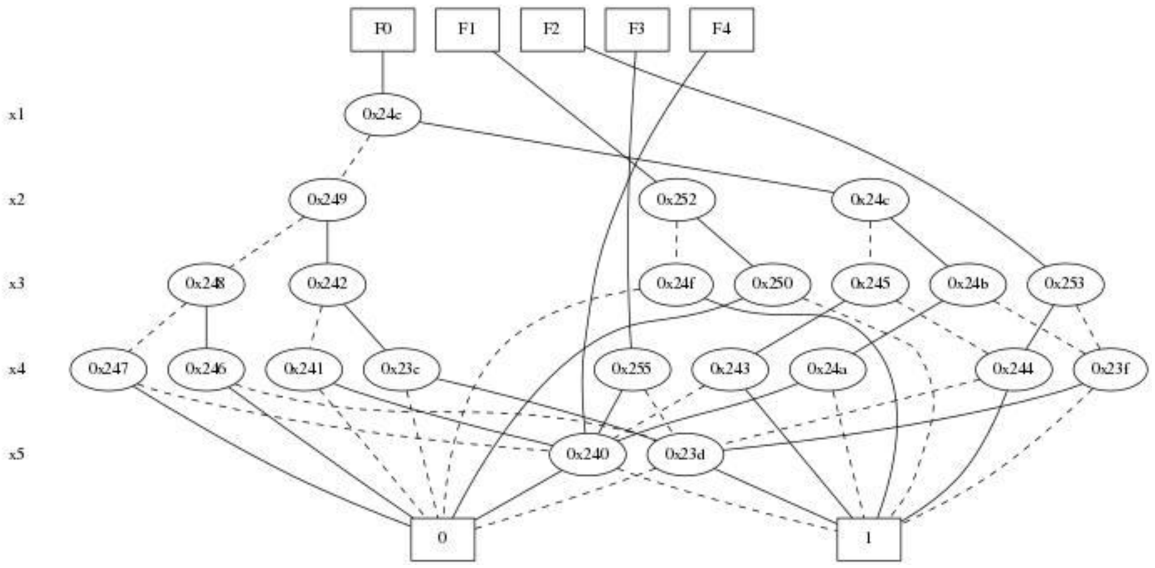
5. **hwb6_14:** .i = 6 .o = 6 .p = 66

BDD:



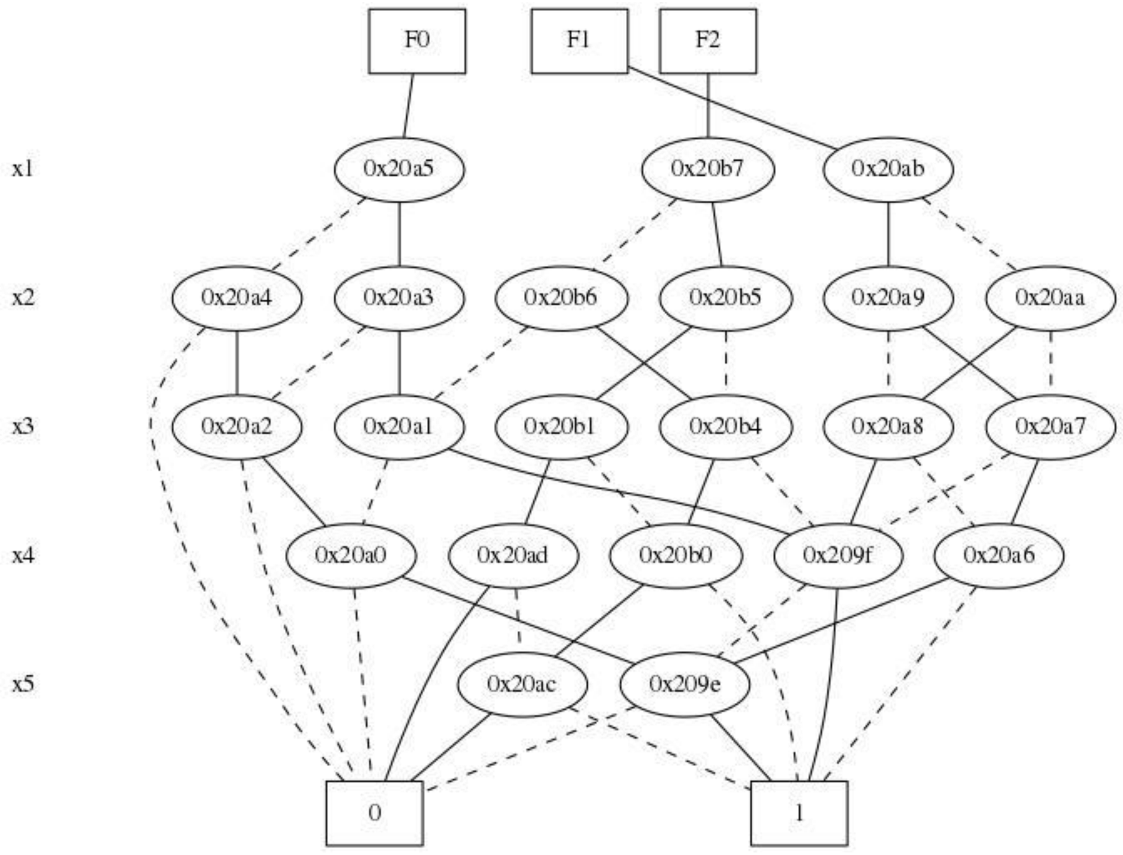
6. **mod5d2_17:** .i = 5 .o = 5 .p = 32

BDD:



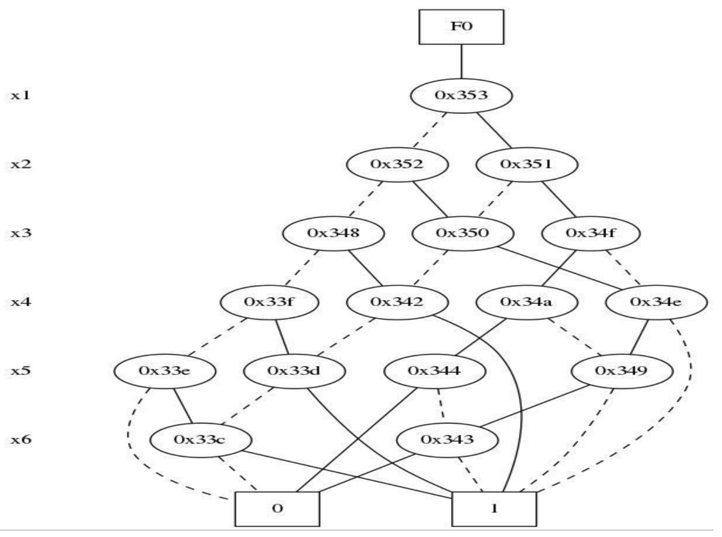
7. **rd53_68:** .i = 5 .o = 3 .p = 32

BDD:



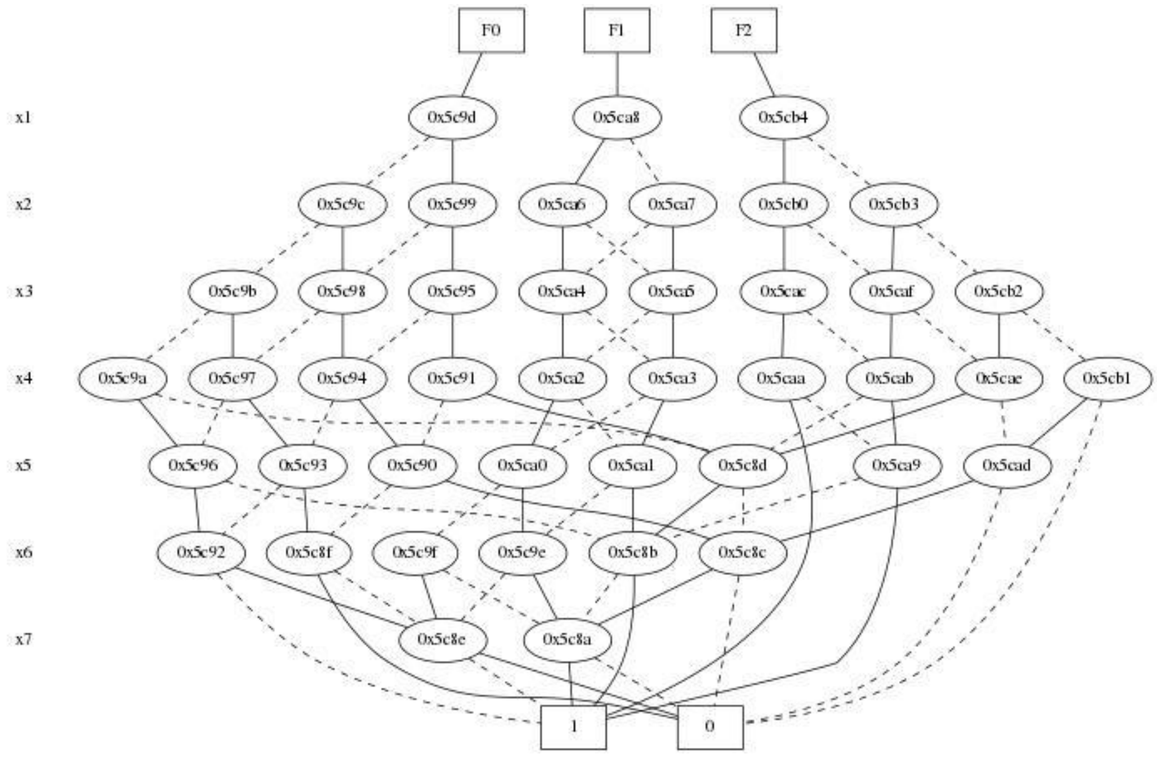
8. **sym6_63:** .i = 6 .o = 1 .p = 50

BDD:



9. **rd73_69:** .i = 7 .o = 3 .p = 141

BDD:



S.No.	Benchmark Name	Node without Splitters	Nodes with Splitters
1	4mod7	30	21
2	ham3_28	21	13
3	hwb4_12	39	23
4	hwb5_13	82	44
5	hwb6_14	148	73
6	mod5d2_17	38	24
7	rd53_68	34	24
8	sym6_63	18	18
9	rd73_69	54	48

Conclusion and Future Possibilities

Our investigations show that reducing the number of gates and reducing the number of splitters are contradictory optimization objectives. Furthermore, the performance of synthesis guided with respect to gate efficiency as well as synthesis guided with respect to splitter freeness is evaluated and an overhead factor between the contradictory metrics is experimentally determined.

The field is open to quite a lot of possibilities, a balance between reducing the number of gates and reduction of splitter such that both optimisation and minimum signal strength is maintained is quite a possibility, but till now it remains untouched. Also optimisation of combinational logic BDDs have not yet reached a dead end either, even though the field is quite saturated but there is a room for still further improvement. Further, optical domain is still a new field and needs to be exploited much more in coming future. Optimisation of optical circuits by reducing the number of gates and splitters is another topic to be explored further. Hence, as a whole the field is quite new and a lot is to follow.

REFERENCES

[1] Arighna Deb, Robert Wille, Oliver Kesz¨ocze, Stefan Hillmich, and Rolf Drechsler. 2016. Gates vs. splitters: Contradictory optimization objectives in the synthesis of optical circuits. J. Emerg. Technol. Comput. Syst. 13, 1, Article 11 (June 2016), 13 pages.
DOI: http://dx.doi.org/10.1145/2904445

[2] L.G. Amaru, New Data Structures and Algorithms for Logic Synthesis and Verification,
DOI 10.1007/978-3-319-43174-1_1

[3] 2017_iccad_synthesis_optical_circuits_aig
http://www.informatik.unibremen.de/agra/doc/konf/2017_iccad_synthesis_optical_circuits_aig.pdf

[4] M. Mohamed, Z. Li, X. Chen, L. Shang, and A. R. Mickelson,
"Reliability-Aware Design Flow for Silicon Photonics On-Chip Interconnect,"
IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 8, pp. 1763–1776, 2014.

[5] T. Sato, K. Takeda, A. Shinya, M. Notomi, K. Hasebe, T. Kakitsuka, and S. Matsuo, "Photonic Crystal Lasers for Chip-to-Chip and On-Chip Optical Interconnects," IEEE Journal of Selected Topics in Quantum Electronics, vol. 21, no. 6, pp. 728–737, 2015.

[6] Optalysys– revolutionary optical processing technology,
www.optalysys.com.

[7] C. E. Shannon. 1938. A symbolic analysis of relay and switching circuits. *Trans. AIEE* 57, 12 (1938), 713–723.DOI:10.1109/T-AIEE.1938.5057767

[8] Fabio Somenzi at the Dept. of Electrical and Computer Engineering, University of Colorado at Boulder,author of the package at http://davidkebo.com/cudd

[9] http://davidkebo.com/source/cudd_versions/cudd-3.0.0.tar.gz

[10] R. Wille, O. Kesz¨ocze, C. Hopfmuller, and R. Drechsler. 2015. Reverse BDD-based synthesis for splitter-free optical circuits. In *Design Automation Conference (ASP-DAC), Asia and South Pacific*. 172–177.

[11] http://www.async.ece.utah.edu/~myers/nobackup/ee5740_98/cudd/cuddExtAbs.html

[12] http://web.mit.edu/sage/export/tmp/y/usr/share/doc/polybori/cudd/node3.html

[13] http://www.async.ece.utah.edu/~myers/nobackup/ee5740_98/cudd/cuddAllDet.html

[14] https://stackoverflow.com/questions/1494492/graphviz-how-to-go-from-dot-to-a-graph?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa