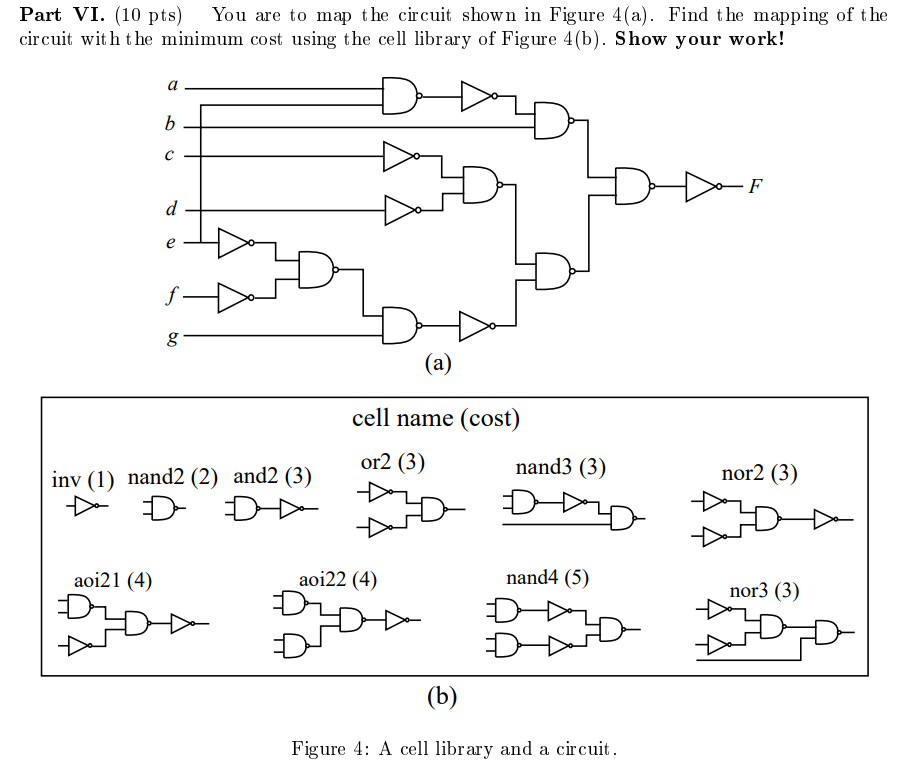
**Project #3**

1. An exercise of technology mapping (I moved it to project to allow you working together)

**20**



1. (**80 points**) Please bring up a BDD package (such as CUDD)

<https://github.com/ivmai/cudd>

Please download that package, find all the source codes and makefiles to compile it and run the binary. If a group successfully does it, it is fine to share. However, please go through the steps by yourself (i.e., everyone to go through it to get a feel).

In **./cudd** directory, you can find **cudd.h, cuddInt.h**, … These are header files that declare the data structures: **DdNode, DdManager, DdChildren**. Glance through these data-structures. You will extensively use pointers to DdManager and DdNode structures. You can also go through the #define macros declared.

Now just list all the \*.c files in the cudd directory: ’ls -al \*.c’. NOTE: the CUDD package implements BDDs, ADDs (called Algebraic Decision Diagrams) as well as ZDDs (called Zero-Suppressed BDDs). You should just ignore ADD and ZDD related files and routines; and concentrate on BDD related files. The file names are self explanatory. For example, **cuddCof.c** contains routines for performing cofactor computations. You don’t need to study the routines - just get introduced to the package.

Now go through the **nanotrav** directory. Go through the README file. Also, glance through the “**main**.c” file. You may not understand much, but don’t worry. We will overwrite this main.c file with (y)our own somewhat simpler “main.c” file. •

Come back to the top directory. You will see two files: Makefile.am and Makefile.in. But no worry. Let’s examine file “README”, we can build this package by

./configure

It will create a new file: Makefile;

then, type

make

make check

Then, the package will be compiled and libraries will be created and linked and an executable named ’nanotrav’ will be created in the ./nanotrav directory. (You will also see all the tests “PASS”.)

Just run a test program as given in the ./nanotrav/README file. For example, type ’./nanotrav -p 1 -cover C17.blif > C17.out’ just to check that the program compiled properly and is indeed running fine.

(Note: blif: Berkeley Logic Interchange Format)

The ./nanotrav/main.c file is too complicated. Let’s write a simple program that:

1. initializes a BDD manager;
2. creates variables;
3. performs some simple ITE computation;
4. prints out the resulting BDD.

A new file ’main.c’ can be found at this link <https://my.ece.utah.edu/~kalla/ECE5740/main.c>

Or you can copy this main.c files; please go to

/disk1/for\_jtli/test/cudd-release/nanotrav;

you can see two files:

main.c.orig 🡪 this comes with GitHub package; in “make check”, all the tests pass.

main.c 🡪 this is a simplified one and for us to modify; some tests in “make check” fail.

First, save the original ./nanotrav/main.c to another file (by renaming it, such as main.c.orig); Then, download a new main.c mentioned in the above link and go through it properly (now in ./nanotrav you have a new main.c) and in ./nanotrav, please (rm all the .o files), go back to top directory, then, type “make check”, which will re-create a new executable “nanotrav”. Just run the program. It is ok to see some tests “failed”.

Relate the output to the code in this file.

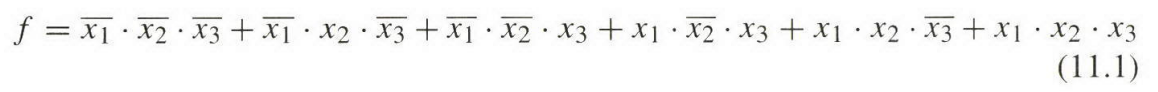
(Note: the CUDD documentation (online) or the .ps file in

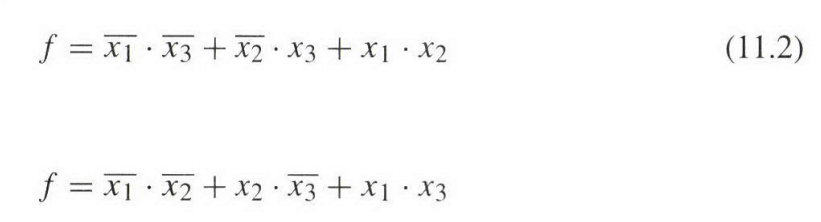
the cudd-2.4.1/cudd/doc/doc.ps will tell you all about these routines. In the .ps file, the index from page 43 onwards gives a list of all the BDD manipulation routines that you can use. (I found this <https://git8.cs.fau.de/software/nusmvf/-/blob/master/cudd-2.4.1.1/cudd/doc/cudd.ps>

But need a postscript reader.))

Now here are the assignments: In this main.c file, you will write code that does the following:

1. Create ROBDD for ***f = ab + ac + bc***. Use any variable order
2. Also run a few more examples, such as hw#3 – problem (1), whose function has 4 variables (w, x, y, z). Please use the program you build to show its BDD.
3. There are 3 other functions below. Welcome to run your program to get their BDDs.





Exercising BDD is helpful. It can be used in technology mapping to verify a pattern graph matching the circuit subject graph. For instance, how do we know we can do this mapping?

Meanwhile, from what I was told, “Boolean Matching” is used in tech mapping, since its method is more efficient than BDD. It is because the input variable ordering is unknown. BDD will take longer time to run, to compare.

