CS613200 Advanced Logic Synthesis
Final Project

Name: 小山翼 (Koyama Tsubasa)
Student ID: 110062526

Part Ⅰ： How to compile and execute my program.

- To compile the program, go to directory **"./src"** and execute the following command, then the executable file named **"map"** will be generated in the same directory.

Command: $ make

- To execute the program, it can be executed with the following command.

Command: $ ./map –k <LUT num> <input blif file> <output blif file>

e.g. $ ./map –k 4 ./blif/10aoi_9symml.blif ./output/10aoi_9symml.blif

Part Ⅱ： data structure

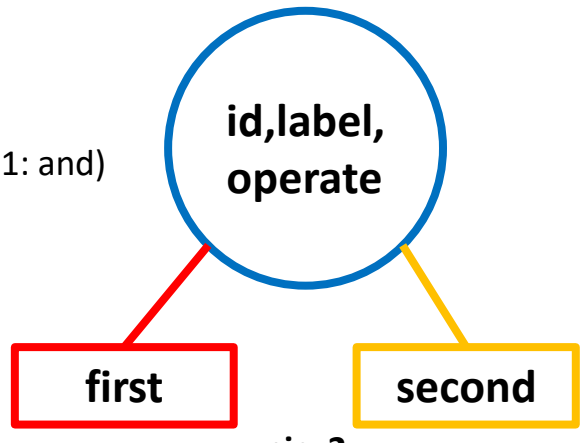
```
/* -----      tree node      ----- */
class Node {

    public:
        string id;
        int label , operate = -2; // -1_not , 0_+ , 1_*
        Node* first = NULL;
        Node* second = NULL;

};
```

pic_1

string id:	name of the node
int label:	label of the node
int operate:	store the operation (-1: not, 0: or, 1: and)
Node* first:	point at the left child
Node* second:	point at the right child

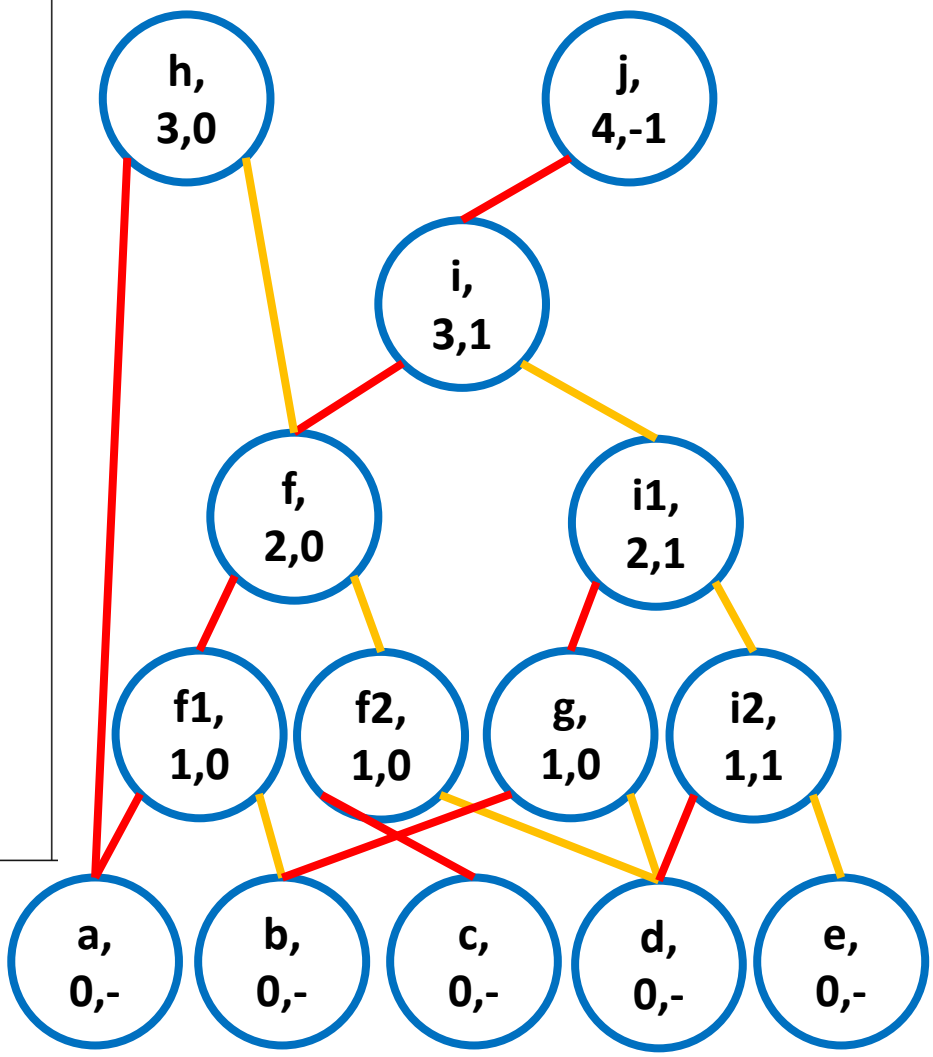


pic_2

Example:

```
.model sample01
.inputs a b c d e
.outputs h j
.names a b f1
1- 1
-1 1
.names c d f2
1- 1
-1 1
.names f1 f2 f
1- 1
-1 1
.names b d g
1- 1
-1 1
.names a f h
1- 1
-1 1
.names d e i2
11 1
.names g i2 i1
11 1
.names f i1 i
11 1
.names i j
0 1
.end
```

pic_3



pic_4

Part Ⅲ: algorithm pseudo code

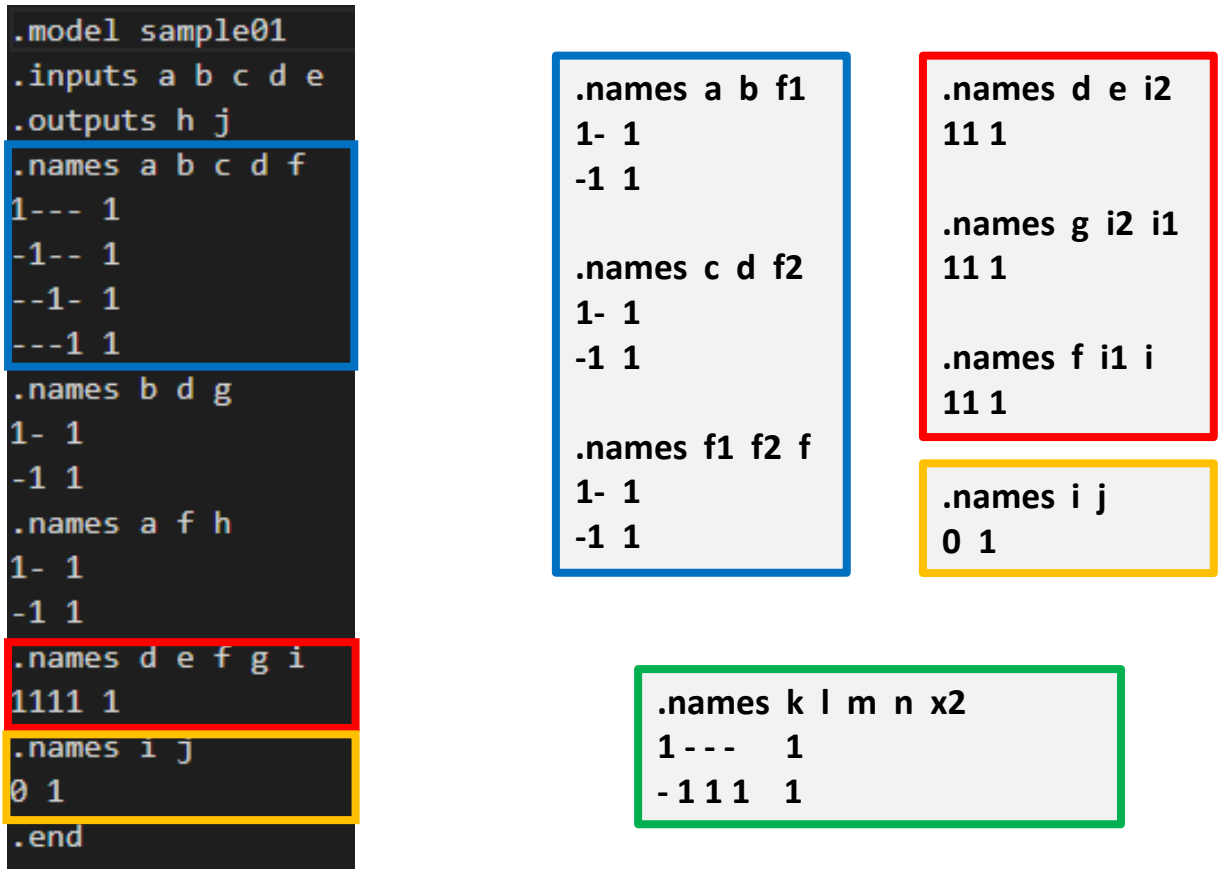
Input : blif_file, LUT_num
Output : decomposed_blif_file, output_file

1. **begin**
2. read blif_file, LUT_num
3. construct decomposed_blif_file from original blif_file (1)
4. construct LUT graph from decomposed_blif_file (2)
5. **end**

partial algorithm (1):

- I only consider the blue, red and orange part first since the input blif files can be done with these three situations, which I have checked by the ABC command (cec).

- I will consider the mixed files like the green one after the whole algorithm finished.



pic_5

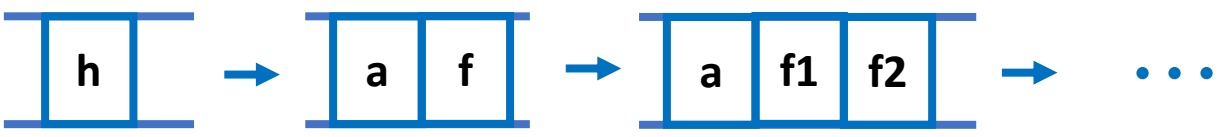
partial algorithm (2):

- In this part, I did not use LEDA but by my own algorithm since I have no idea how to use the command max flow to construct the LUTs.

- I first construct LUTs one by one, from each nodes (pic_6). Then I construct function of each LUT (pic_7) and convert it into postorder. After converting, I use [3] to construct the truth table of LUTs.

Example (use pic_4):

stack



pic_6

```
void construct_func(string original , vector<string> temp , vector<string> pi) {

    if ( find(temp.begin(),temp.end(),original)!=temp.end() or
         find(pi.begin(),pi.end(),original)!=pi.end() ) return;

    Node *n = find_Node.find(original)->second;
    if (n->operate==-1) {

        func = regex_replace(func,regex(original),"( ! "+n->first->id+" ) ");
        construct_func(n->first->id,temp,pi);

    }
    else {

        if (n->operate==0) func = regex_replace(func,regex(original),"( "+n->first->id+" || "+n->second->id+" ) ");
        if (n->operate==1) func = regex_replace(func,regex(original),"( "+n->first->id+" * "+n->second->id+" ) ");
        construct_func(n->first->id,temp,pi);
        construct_func(n->second->id,temp,pi);

    }

}
```

pic_7

- After finish constructing truth tables, I will call the void function, cal_level(), to calculate the whole level of the output blif file (pic_8).

```
void cal_level(vector<string> v) {

    int n_id = -1 , max_level;
    string temp;
    map<string,int> mapping;
    vector<string> funcs[out_size];
    for (auto i: inputs) mapping[i] = 0;

    for (auto s: v) {

        istringstream ss(s);
        while (ss >> temp);
        for (auto o: out) if (temp==o) n_id++;
        funcs[n_id].push_back(s);

    }

    for (auto f: funcs) {

        for ( auto sub_f = f.rbegin(); sub_f!=f.rend() ; ++sub_f ) {

            max_level = 0;
            vector<string> cal;
            istringstream s(sub_f->c_str());
            while (s >> temp) cal.push_back(temp);
            cal.pop_back();
            for (auto c: cal) max_level = max(mapping.find(c)->second,max_level);
            mapping[temp] = max_level + 1;
            // cout<<temp<<" "<<max_level + 1<<endl;
        }

    }
    // for (auto m: mapping) cout<<m.first<<" "<<m.second<<endl;
    max_level = 0;
    for (auto o: out) max_level = max(max_level , mapping.find(o)->second);
    circuit_level = max_level;

}
```

pic_8

Part Ⅳ: results

I have run some samples and compare the output file and the original input file by ABC command cec, and the results are all correct. Although I have finished the whole functions, but there are some parts which I think it can be further improved. The first one is in decomposed part, my codes can only deal with the independent one but can not convert mixed one. The second is the number of LUTs and circuit level, since my algorithm is very simple, these two might not be minimal. Through this final project, I learned how to read the blif file, the method to construct functions which are used in decomposition and technology mapping.

Part Ⅴ: references

[1]: J. Cong, Y. Ding, An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table based FPGA Designs, IEEE Trans. on CAD, Vol. 13, No. 1, Jan. 1994, pp. 1-12
[2]: Berkeley Logic Interchange Format (BLIF), University of California, Berkeley, July 28, 1992
[3]: Boolean Truth Table C++, https://stackoverflow.com/questions/52505340/boolean-truth-table-c