本次作業為讀 verilog combinational 電路，透過所附的 lib 檔求 output capacitance、timing、最長路徑。分成了五個檔案寫：分別是 graph.h、graph.cpp、func.h、func.cpp、0710880.cpp，分別敘述各檔功能：

1. graph.h、graph.cpp:

   .h 檔宣告函式，因為本題需要探討每個 node 之間的連接情形，故宣告一個 graph node 的結構，裡面放會用到的資訊:inout wire、node、logic、node delay、total delay、transition time 等等，相關會用到的函式也在此處宣告。

0710880.cpp | func.cpp | func.h | graph.h | graph.cpp

```h
#ifndef GRAPH_H
#define GRAPH_H

using namespace std;
#include<map>
#include<vector>
#include<string>
#include<iostream>
/*--------------------function initialized----------------------------------*/


struct GraphNode{
    int id;
    string type;
    vector<string> in_wire;
    vector<string> out_wire;
    vector<string> in_graph;
    map<string,int> input_pin; //0:a1,1:a2
    vector<string> out_graph;
    map<string,int> output_pin; //0:a1,1:a2

    double outcap;

    int logic_result;
    double total_delay,delay,transition_time;
    bool run_through;
    string path_choose_in;
};
typedef struct GraphNode GraphNode;

void print_graph(map<string,GraphNode> &circuit);
vector<string> search_outgraph(map<string,GraphNode> &circuit,string wire,map<string,int> &input_pin);
vector<string> search_ingraph(map<string,GraphNode> &circuit,vector<string> &wire,map<string,int> &output_pin);

#endif
```

   下圖為兩個重要的查找連接點函式(inout pin 連接情形)

```cpp
//return the connection situation
vector<string> search_outgraph(map<string,GraphNode> &circuit,string wire,map<string,int> &input_pin)
{
    vector<string> return_id;
    for(const auto& m : circuit){
        for(int i=0;i<m.second.in_wire.size();++i){
            if(wire == m.second.in_wire[i]){
                input_pin[m.first] = i;
                return_id.push_back(m.first);
            }
        }
    }
    return  return_id;
}

vector<string> search_ingraph(map<string,GraphNode> &circuit,vector<string> &wire,map<string,int> &output_pin){
    vector<string> return_id;

    for(const auto& m : circuit){
        for(int i=0;i<m.second.out_wire.size();++i){
            if(wire[0] == m.second.out_wire[i]){
                return_id.push_back(m.first);
                output_pin[m.first] = 0;
            }
            //nand nor
            if(wire.size() > 1){
                if((wire[1] == m.second.out_wire[i])){
                    return_id.push_back(m.first);
                    output_pin[m.first] = 1;
                }
            }
        }
    }
    return  return_id;
}
```

2. func.h、func.cpp:

所有本程式用到的函式宣告，新增一個 struct 叫 libdata，存放讀 lib 檔讀出來的資料(pin capicitance、建立的 look up table)。

0710880.cpp | func.cpp | func.h | graph.h | graph.cpp

```
#ifndef FUNC_H
#define FUNC_H

#include<iostream>
using namespace std;
#include <sstream>
#include<map>
#include<vector>
#include<string>
#include "graph.h"

struct data{
    double a1;
    double a2;
    double zn;
    //need to store four look up table
    vector<double> index_outcap;
    vector<double> index_transiton_time;
    vector< vector<double> > table_cell_rise,table_cell_fall; //caculate delay
    vector< vector<double> > table_rise_transition,table_fall_transition; //calculate transtion time
};

typedef struct data libdata;
typedef pair<string,GraphNode> Pair;

/*--------------------function initialized----------------------------------*/
void splitStr2Vec(string str, vector<string>& buf);
string RemoveSpaces(const std::string& str);
void print_lib(map<string,libdata> &lib_data);
bool compare(const Pair &p1,const Pair &p2);
double calculation(double cap1,double cap2,double in_trans1,double in_trans2,double n1,double n2,double n3,double n4,double cap,double trans);
void cal_logic(map<string,GraphNode> &circuit,string id,map<string,int> index);
void cal_delay(map<string,GraphNode> &circuit,string id,map<string,int> index, map<string,libdata> lib_data);

#endif
```

Calculation:功用為查詢讀檔進來之 look up table，運用內插法求相對應 delay 或 transition time。

```
double calculation(double cap1,double cap2,double in_trans1,double in_trans2,double n1,double n2,double n3,double n4,double cap,double trans){
    double ans1,ans2,result;
    if((cap>=cap1) && (cap <= cap2)){
        if((trans >= in_trans1) && (trans <= in_trans2)){
            ans1 =  (cap2-cap)*n1/(cap2-cap1)+(cap-cap1)*n2/(cap2-cap1);
            ans2 = (cap2-cap)*n3/(cap2-cap1)+(cap-cap1)*n4/(cap2-cap1);
            result = (trans-in_trans1)*ans2/(in_trans2-in_trans1) + (in_trans2-trans)*ans1/(in_trans2-in_trans1);
        }
        else if(trans < in_trans1){
            ans1 =  (cap2-cap)*n1/(cap2-cap1)+(cap-cap1)*n2/(cap2-cap1);
            ans2 = (cap2-cap)*n3/(cap2-cap1)+(cap-cap1)*n4/(cap2-cap1);
            result = ans2 - (ans2-ans1)*(in_trans2-trans)/(in_trans2-in_trans1);
        }
        else if(trans > in_trans2){
            ans1 =  (cap2-cap)*n1/(cap2-cap1)+(cap-cap1)*n2/(cap2-cap1);
            ans2 = (cap2-cap)*n3/(cap2-cap1)+(cap-cap1)*n4/(cap2-cap1);
            result = ans1 + (ans2-ans1)*(trans-in_trans1)/(in_trans2-in_trans1);
        }
    }
    else if(cap < cap1){
        if(trans < in_trans1){
            ans1 = n2-((n2-n1)*(cap2-cap)/(cap2-cap1));
            ans2 = n4-((n4-n3)*(cap2-cap)/(cap2-cap1));
            result = ans2 - (ans2-ans1)*(in_trans2-trans)/(in_trans2-in_trans1);
        }
        else if((trans >= in_trans1) && (trans <= in_trans2)){
            ans1 = n2-((n2-n1)*(cap2-cap)/(cap2-cap1));
            ans2 = n4-((n4-n3)*(cap2-cap)/(cap2-cap1));
            result = (trans-in_trans1)*ans2/(in_trans2-in_trans1) + (in_trans2-trans)*ans1/(in_trans2-in_trans1);
        }
    }
    else if(cap > cap2){
         if(trans > in_trans2){
            ans1 = n1+(n2-n1)*(cap-cap1)/(cap2-cap1);
            ans2 = n3+(n4-n3)*(cap-cap1)/(cap2-cap1);
            result = ans1 + (ans2-ans1)*(trans-in_trans1)/(in_trans2-in_trans1);
        }
         else if((trans >= in_trans1) && (trans <= in_trans2)){
            ans1 = n1+(n2-n1)*(cap-cap1)/(cap2-cap1);
            ans2 = n3+(n4-n3)*(cap-cap1)/(cap2-cap1);
            result = (trans-in_trans1)*ans2/(in_trans2-in_trans1) + (in_trans2-trans)*ans1/(in_trans2-in_trans1);
        }
    }
    return result;
}
```

下圖兩個函式的功能為處理 input 字串、分割字串，其中 str.find_first_of 字串放的內容為分割字元

```
//deal with string
void splitStr2Vec(string str, vector<string>& buf)
{
  int current = 0; //initial position
  int next;
  while (1)
  {
    next = str.find_first_of(" ,\";\{()\\}:" , current);
    if (next != current)
    {
      string tmp = str.substr(current, next - current);
      if (tmp.size() != 0) //忽略空字串
        buf.push_back(tmp);
    }
    if (next == string::npos) break;
    current = next + 1; //下次由 next + 1 的位置開始找起。
  }
}

string RemoveSpaces(const std::string& str) {
    string out;                 // the result
    string word;                // used to extract words from str
    istringstream ss(str);      // create an istringstream from str
    while(ss >> word) {             // extract a word
        if(!out.empty()) out += ' '; // add a space between words
        out += word;                // add the extracted word
    }
    return out;
}

```

Cal_logic:依照 node 的 type 對應到不同計算求邏輯值

Cal_delay:依照 node 的 type 對應到不同的 LUT 進行查表計算 timing

```
void cal_logic(map<string,GraphNode> &circuit,string id,map<string,int> index){
    //cout << id <<":"<<endl;

    if(circuit[id].type == "INVX1"){
        if(!circuit[id].in_graph.empty()){
            circuit[id].logic_result = !( circuit[circuit[id].in_graph[0]].logic_result);
            //cout <<itr->first<<":"<< itr->second.logic_result << endl;
        }
        else{  //read in graph
            circuit[id].logic_result = !( index[circuit[id].in_wire[0]]);

        }
    }
    else if(circuit[id].type == "NANDX1"){
```

3. 0710880.cpp:

處理讀寫檔的部分，在這邊我們利用 stl 的 map 來儲存 circuit 及 libdata 資訊，下圖為一開始變數的宣告及讀入檔案，我們總共會輸出三個檔案內容。

```
    ifstream infile,infile2,infile3;
    ofstream outfile,outfile2,outfile3;
    vector<string> input_node,output_node,wire_node;
    string instr;

    //combinational circuit
    map<string,GraphNode> circuit;

    //lib data
    map<string,libdata> lib_data;

    deque<string> undo_queue; //to calculate delay

    infile.open(argv[1]);
    infile2.open(argv[3]);
    infile3.open(argv[5]);

    string new_argv = argv[1];

    string sub_str = new_argv.substr(0,new_argv.size()-2);

    outfile.open("0710880_"+sub_str+"_load.txt");
    outfile2.open("0710880_"+sub_str+"_delay.txt");
    outfile3.open("0710880_"+sub_str+"_path.txt");
```

下圖程式的部分為讀完所有 input file 資訊後，由連接情形計算 step 1 之 output capacitance

```
/*go through the graph(n2) and build the adjancy list*/
    for(auto itr=circuit.begin();itr != circuit.end() ;++itr){
        itr->second.out_graph = search_outgraph(circuit,itr->second.out_wire[0],itr->second.input_pin);
        itr->second.in_graph = search_ingraph(circuit,itr->second.in_wire,itr->second.output_pin);
        if(itr->second.in_graph.empty()){
            undo_queue.push_back(itr->first);
            //cout <<undo_queue.back()<<" ";
        }
    }

    for(auto itr=circuit.begin();itr != circuit.end() ;++itr){
        for(int i=0;i<itr->second.out_graph.size();++i){
            if(circuit[itr->second.out_graph[i]].type == "NOR2X1"){
                if(itr->second.input_pin.at(itr->second.out_graph[i]) == 0){
                    itr->second.outcap  += lib_data["NOR2X1"].a1;
                }
                else if(itr->second.input_pin.at(itr->second.out_graph[i]) == 1){
                    itr->second.outcap  += lib_data["NOR2X1"].a2;
                }
                //cout << itr->second.outcap <<endl;
            }
            else if(circuit[itr->second.out_graph[i]].type == "INVX1"){
                if(itr->second.input_pin.at(itr->second.out_graph[i]) == 0){
                    itr->second.outcap  += lib_data["INVX1"].a1;
                }
            }
            else if(circuit[itr->second.out_graph[i]].type == "NANDX1"){
                if(itr->second.input_pin.at(itr->second.out_graph[i]) == 0){
                    itr->second.outcap  += lib_data["NANDX1"].a1;
                }
                else if(itr->second.input_pin.at(itr->second.out_graph[i]) == 1){
                    itr->second.outcap  += lib_data["NANDX1"].a2;
                }
            }
        }
        for(int i=0;i<output_node.size();++i){
            if(itr->second.out_wire[0] == output_node[i]){
                itr->second.outcap += 0.03;
                break;
            }
```

406~472 行主要使用 deque 在處理計算 timing 及 logic 時之順序問題，當我們計算一個 queue 時它前面的 input node 一定已經被計算完，我們計算完要 pop 的時候再把它的 output 連接的點都 push 進來。

```
401         //deal with queue
402         deque<string> undo_queue2 = undo_queue;
403         bool in_queue = false;
404 
405         //queue end when run through all nodes
406         while(!undo_queue2.empty()){
407 
408             //if is still input node at front,re push
409             if(circuit[undo_queue2.front()].in_graph.empty()){
410                 cal_logic(circuit,undo_queue2.front(),input_data[i]);
411                 cal_delay(circuit,undo_queue2.front(),input_data[i],lib_data);
412 
413                 circuit[undo_queue2.front()].run_through = true;
414                 for(int j=0;j<circuit[undo_queue2.front()].out_graph.size();++j){
415                     //prevent redundant add
416                     for(int k=0;k<undo_queue2.size();++k){
417                         if(undo_queue2[k] == circuit[undo_queue2.front()].out_graph[j]){
418                             in_queue = true;
419                         }
420                     }
421                     if(!in_queue)  undo_queue2.push_back(circuit[undo_queue2.front()].out_graph[j]);
422                     in_queue = false;
423                 }
424                 undo_queue2.pop_front();
425             }
```

下圖為遇到 queue 中的值前面接點還沒計算過之情形，此時我們會先跳過，itr 為計算最後要計算之結點，最後計算該編號值並刪除它

```
        while(1){
            if(circuit[undo_queue2[pop_index]].in_graph.size() == 1){
                if(!circuit[circuit[undo_queue2[pop_index]].in_graph[0]].run_through){
                    ++pop_index;
                    ++itr;
                }
                else{
                    break;
                }
            }
            else if(circuit[undo_queue2[pop_index]].in_graph.size() == 2){
                if((circuit[circuit[undo_queue2[pop_index]].in_graph[0]].run_through) && (circuit[circuit[undo_queue2[pop_index]].in_graph[1]].run_through)){
                    break;
                }
                else{
                    ++pop_index;
                    ++itr;
                }
            }
        }
        cal_logic(circuit,undo_queue2[pop_index],input_data[i]);
        cal_delay(circuit,undo_queue2[pop_index],input_data[i],lib_data);

        circuit[undo_queue2[pop_index]].run_through = true;



        for(int j=0;j<circuit[undo_queue2[pop_index]].out_graph.size();++j){
            for(int k=0;k<undo_queue2.size();++k){
                if(undo_queue2[k] == circuit[undo_queue2[pop_index]].out_graph[j]){
                    in_queue = true;
                }
            }
            if(!in_queue)  undo_queue2.push_back(circuit[undo_queue2[pop_index]].out_graph[j]);
            in_queue = false;
        }
        undo_queue2.erase(itr);
    }
```

最後透過 stack FILO 的特性(path)由 output total delay 最小的點追蹤回去，再 pop stack，就得到走過的最長路徑

```
500         path.push(circuit[max_node].out_wire[0]);
501
502         path_node = circuit[max_node].path_choose_in;
503
504         while(circuit[path_node].path_choose_in != " "){
505             path.push(circuit[path_node].out_wire[0]);
506             path_node = circuit[path_node].path_choose_in;
507         }
508         path.push(circuit[path_node].out_wire[0]);
509
510         if(circuit[path_node].type == "INVX1"){
511             path.push(circuit[path_node].in_wire[0]);
512         }
513         else if(circuit[path_node].type == "NANDX1"){
514             if(input_data[i][circuit[path_node].in_wire[0]]==0){
515                 path.push(circuit[path_node].in_wire[0]);
516             }
517             else if(input_data[i][circuit[path_node].in_wire[1]]==0){
518                 path.push(circuit[path_node].in_wire[1]);
519             }
520             else{
521                 path.push(circuit[path_node].in_wire[0]);
522             }
523         }
524         else if(circuit[path_node].type == "NOR2X1"){
525             if(input_data[i][circuit[path_node].in_wire[0]]==1){
526                 path.push(circuit[path_node].in_wire[0]);
527             }
528             else if(input_data[i][circuit[path_node].in_wire[1]]==1){
529                 path.push(circuit[path_node].in_wire[1]);
530             }
531             else{
532                 path.push(circuit[path_node].in_wire[0]);
533             }
534         }
535
```