CAD for VLSI Design

Project Assignment 2

Partitioning

Instructor: Andy, Yu-Guang Chen Ph.D.

TA: Yi-Ting Lin

Department/Class: Electrical Engineering 4A

Name: 陳緯亭

Student ID Number: **109501201**

Contents

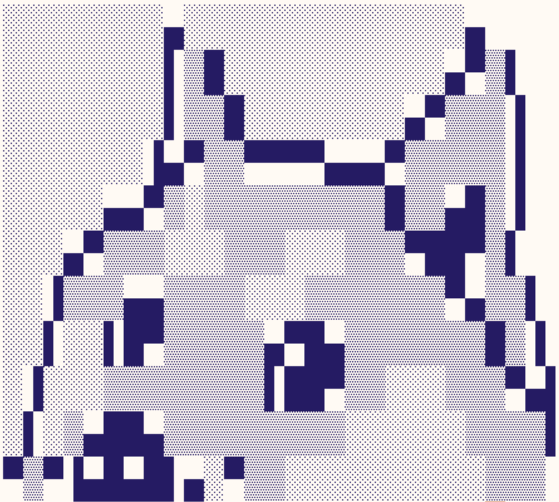
List Of Listings

1. How I compile and execute the program

```
[s109501201@cad ~/PA2]$ make all
g++ -std=c++11 -c 109501201_PA2.cpp
g++ -std=c++11 109501201_PA2.o -o exe
[s109501201@cad ~/PA2]$ make run input=case2 output=case2.out
./exe case2 case2.out
Read File
[s109501201@cad ~/PA2]$ make clean
rm -f *.o
rm -f exe
```

Fig 1: Using make to compile and execute my program

```
[s109501201@cad ~/PA2]$ ./PA2_CHECKER case2 case2.out
error count: 0
Well Done !!!
```



```
[s109501201@cad ~/PA2]$
```

Fig 2: Use the executable file to ISCAS'85 netlist into Verilog format

2. Pseudo Code

Algorithm 1 Simulated Annealing Algorithm

```
function SimulatedAnnealing
    Initialize T (temperature), T_end (ending temperature), c_now (starting configuration)
    if c_now is nullptr then
        return
    end if
    c_best → cut_size ← 0
    repeat
        repeat
            Perturb(c_now, c_next)                                          ▷ Generate a solution
            if Cost(c_next) < Cost(c_now) and IsConstraint1(c_next) then     ▷ Compare energy
                c_now ← c_next
                if Cost(c_now) < Cost(c_best) and IsConstraint1(c_now) then
                    c_best ← c_now                                           ▷ Accept the new solution
                end if
            else if Metropolis(t, Cost(c_next) - Cost(c_now)) then
                c_now ← c_next                                               ▷ Acceptance probability
            end if
        until IsConstraint1(c_best)
        T ← alpha × T
    until T ≤ T_end                                                          ▷ Update temperature
end function
```

3. PA

3.1 The degree of completion of the assignment: ALL

3.2 Code Explanation

Listing 1: Preprocessors

```
#include <iostream> // Used for standard input-output streams
#include <map>
#include <fstream> // Used for file input-output
#include <string>
#include <sstream>
#include <vector>
#include <cmath> // Provides definitions for mathematical functions
#include <ctime> // Used for obtaining system time
#include <set>
#include <chrono> //providing representations of time points and durations
#include <iomanip> // setw
#include <limits.h> // Use the INT_MAX
```

According to Listing 2, there are structures here to hold all the necessary circuit information in order to facilitate information transfer. The ckt structure is used to store the input file information, and the ab structure is used to record the partitioning of the circuit into two sub-circuits, A and B.

Listing 2: Struct and Class

```
using namespace std;

struct ckt
{
    map<int, set<int>> nets;
    int net_count;
    string name;
    int cell_count;
};

typedef struct ckt *Ckt;

struct ab
{
    vector<set<int>> cells; // an 2D array of AB seperation blocks
    set<int> cut;           // an array of nets being cut
    int cut_size;
};
```

```
typedef struct ab *AB;

class Partitioning
{
public:
    Partitioning(string input_file, string output_file) : in(std::move(input_file)),
        out(std::move(output_file)) {}
    void Run()
    {
        ABnet = new ab();
        InputFile();
        SimulatedAnnealing();
        ShrinkCut(ABnet);
        OutABNet();
    }

private:
    void FirstPass(ifstream &);
    void SecondPass(ifstream &);
    void NetCount(int);
    void CellCount(int);
    void SimulatedAnnealing();
    int Cost(AB);
    void Perturb(AB, AB);
    void Initial(AB, AB);
    map<int, set<int>> DefineConnection(AB);
    void DFS(int, set<int> &, set<int> &, map<int, set<int>> &);
    void FindConnection(set<int> &, AB, map<int, set<int>> &);
    void OutABNet();
    void StartPartition(AB);
    bool IsConstraint1(AB);
    bool Metropolis(double, int);
    void ABMerge(AB);
    bool IsFindABnet(AB, int i);
    void ShrinkCut(AB);
    bool IsRedundantCut(set<int> nets, vector<set<int>> cells);

    void InputFile();
    void Output();
    void OutAB(AB);

    int CalCutSize(AB);
    bool ABsize(vector<set<int>> now_ABnet);
    Ckt circuit;

    string in;
    string out;
    int net_count = 0;
    int cell_count = 0;
```

```
    map<int, set<int>> connections;
    AB ABnet;
};
```

Following the listing 3, I write the seeds random generator using the current time.I divide the behaviour into two branches - process file and partitioning.

Listing 3: The main function

```
int main(int argc, char *argv[])
{
    srand(time(NULL));
    if (argc != 3)
    {
        std::cout << "Can't open!" << endl;
        return 1;
    }

    // If still running, force stop the process
    Partitioning partition(argv[1], argv[2]);
    partition.Run();

    return 0;
}
```

As indicated in Listing 4, the format file is designed to be read once. The information will be collected in its entirety during the initial pass.

Listing 4: Input file process

```
void Partitioning::InputFile()
{
    circuit = new ckt;
    ifstream inFile(in);

    circuit->name = in;

    if (!inFile)
    {
        std::cout << "Input File can't be open!" << endl;
        return;
    }

    FirstPass(inFile);
    inFile.close(); // close file
    std::cout << "Read File" << endl;
}
```

As indicated in Listing 5, the output file will be generated at this location. This process would result in the outstreaming of the information stored in ABnet.

Listing 5: Output file process

```
void Partitioning::OutABNet()
{
    ofstream outFile(out);

    if (!outFile)
    {
        std::cout << "Input File can't be open!" << endl;
        return;
    }
    outFile << "cut_size ";
    outFile << ABnet->cut_size << endl;
    int A = 0;
    for (const auto &net : ABnet->cells)
    {
        if (net.size() != 0)
            outFile << char('A' + A) << endl;
        A++;
        for (int cell : net)
        {
            outFile << "c" << cell << endl;
        }
    }
    delete (ABnet);
}
```

In accordance with Listing 6, the search for information in the input file should commence, with the string being transformed into an integer. The nets are of the type map<int, set<int> >. The key value is the net name, while the set<int> is used to store the corresponding cells connected by the net.

Listing 6: First file process

```
void Partitioning::FirstPass(ifstream &inFile)
{
    string s;
    string idle;

    string net;
    while (getline(inFile, s))
    {
        istringstream iss(s);
        iss >> idle >> net;

```

```
        int netc = stoi(net.substr(1, net.size() - 1));
        NetCount(netc);

        set<int> temp;
        while (iss >> idle)
        {
            if (idle == "{")
                continue;
            if (idle == "}")
                break; // if '}' break this line
            int idlec = stoi(idle.substr(1, idle.size() - 1));
            CellCount(idlec);

            temp.insert(idlec);
        }
        circuit->nets[netc] = temp;
    }
}
```

In accordance with Listing 7, the total size of the net and the cell size can be determined.

Listing 7: Net and cell count

```
void Partitioning::NetCount(int net)
{
    if (net > net_count)
    {
        net_count = net;
    }
    circuit->net_count = net_count;
}

void Partitioning::CellCount(int cell)
{
    if (cell > cell_count)
    {
        cell_count = cell;
    }
    circuit->cell_count = cell_count;
}
```

In accordance with Listing 8, this is the method of simulated annealing (SA). Initially, the temperature is set and then decreased over time. To prevent the process from being terminated prematurely, a measurement of elapsed time is employed. The objective is to identify the optimal solution, which is believed to be provided by ABnet.

Listing 8: Simulated Annealing

```
void Partitioning::SimulatedAnnealing()
{
    double ti = 3675, tend = 1;
    double t = ti;
    auto start_time = std::chrono::steady_clock::now();

    // ABnet is not empty
    if (ABnet == nullptr)
    {
        std::cout << "ABnet pointer is null." << endl;
        return;
    }
    AB now_ABnet = new ab();
    AB next_ABnet = new ab();
    ABnet->cut_size = 0;
    Initial(now_ABnet, now_ABnet); // Initialize the partition assignment

    do
    {
        do
        {
            auto end_time = std::chrono::steady_clock::now();
            auto elapsed_time = std::chrono::duration_cast<std::chrono::minutes>(
                end_time - start_time).count();
            if (elapsed_time >= 9)
            {
                std::cout << "Time limit exceeded. Terminating program." << std::endl;
                break;
            }
            Perturb(now_ABnet, next_ABnet);

            if (Cost(next_ABnet) <= Cost(now_ABnet) && IsConstraint1(now_ABnet))
            {
                *now_ABnet = *next_ABnet;
                if (Cost(now_ABnet) <= Cost(ABnet) && IsConstraint1(now_ABnet))
                {
                    *ABnet = *now_ABnet;
                }
            }
            else if (Metropolis(t, Cost(next_ABnet) - Cost(now_ABnet)))
            {
                *now_ABnet = *next_ABnet;
            }
        } while (!IsConstraint1(ABnet));
        t = 0.9 * t;
    } while ((t > tend));

    delete now_ABnet;
    delete next_ABnet;
```

```
}
```

In accordance with Listing 9, it is necessary to ensure that the size of the A and B blocks is as uniform as possible and that the A and B blocks are present. The function would be employed in the context of both the SA and the Perturb functions.

Listing 9: Constraint1

```
bool Partitioning::IsConstraint1(AB now_ABnet)
{
    if (now_ABnet->cells.size() < 2)
    {
        return false;
    }

    if (now_ABnet->cells[0].empty() || now_ABnet->cells[1].empty())
    {
        return false;
    }
    else
    {

        int countA = now_ABnet->cells[0].size();
        int countB = now_ABnet->cells[1].size();
        if (double(abs(countA - countB)) <= double(circuit->cell_count) / 5)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

In accordance with the definition provided in Listing 10, the objective is to return the bool in order to prevent the local optimisation.

Listing 10: Metropolis

```
bool Partitioning::Metropolis(double t, int cost)
{
    double r = static_cast<double>(rand()) / (RAND_MAX + 1.0);
    return (exp(-double(cost) / t) > r);
}
```

In accordance with Listing 11, the cut_size is defined as the cost that should be used as the basis for determining whether the best solution or next solution should be refreshed in SA.

Listing 11: Cost

```
int Partitioning::Cost(AB now_ABnet)
{
    if (now_ABnet->cells.size() >= 2)
        return now_ABnet->cut.size();
    else
        return INT_MAX;
}
```

In accordance with Listing 12, if the optimal solution (ABnet) does not have the requisite value, it is permissible to simply add the cut. Otherwise, it is necessary to perform a 10% perturbation of the present cut and ascertain whether the perturbation would influence the constraint to the extent that it would not be satisfied. If the constraint is not satisfied, the perturbation should be recovered in order to prevent bias.

Listing 12: Perturbation

```
void Partitioning::Perturb(AB now_ABnet, AB next_ABnet)
{
    *next_ABnet = *now_ABnet;

    int chooseonecut = rand() % (circuit->net_count) + 1;

    if (ABnet->cut_size != 0)
    {
        int i = 0;
        while (i < chooseonecut)
        {
            int r = rand() % (circuit->net_count) + 1;
            if (i <= ABnet->cut_size * 0.1)
            {
                if (!next_ABnet->cut.count(r))
                {
                    next_ABnet->cut.insert(r);
                }
                else
                {
                    next_ABnet->cut.erase(r);
                }
                StartPartition(next_ABnet);
                if (!IsConstraint1(next_ABnet))
                {
                    if (next_ABnet->cut.count(r))
                        next_ABnet->cut.erase(r);
```

```
                    else
                        next_ABnet->cut.insert(r);
                }
                StartPartition(next_ABnet);
            }
            i++;
        }
    }
    else
    {
        int i = 1;
        while (1)
        {
            int r = rand() % (circuit->net_count) + 1;
            next_ABnet->cut.insert(r); // Mark it as cut
            StartPartition(next_ABnet);
            if (next_ABnet->cells.size() >= 2 && IsConstraint1(next_ABnet))
                break;
            i++;
        }
    }
    StartPartition(next_ABnet);
    ShrinkCut(next_ABnet);
    next_ABnet->cut_size = CalCutSize(next_ABnet);
}
```

In accordance with Listing 13, use this functions to clasify the block A and B.

Listing 13: Start partitioning

```
void Partitioning::StartPartition(AB now_ABnet)
{
    map<int, set<int>> connection = DefineConnection(now_ABnet);
    set<int> visited;
    FindConnection(visited, now_ABnet, connection);
    ABMerge(now_ABnet);
    visited.clear();
}
```

In accordance with the initial definition provided in Listing 14, this is to define the connection between cells following a cut.

Listing 14: Define connections of the cells

```
map<int, set<int>> Partitioning::DefineConnection(AB now_ABnet)
{
    map<int, set<int>> next_connections;
    next_connections.clear();
    for (const auto &i : circuit->nets) // search all nets
```

```
    {
        int j = 0;
        if (!now_ABnet->cut.count(i.first))
        {
            for (auto &j : i.second)
            {
                set<int> temp;
                for (const auto &k : i.second)
                {
                    next_connections[j].insert(k);
                    next_connections[k].insert(j);
                }
            }
        }
    }
    return next_connections;
}
```

In accordance with the definition provided in Listing 15, the objective is to implement DFS to accurately define the relationship between cells following a cut.

Listing 15: Find the connection between cells

```
void Partitioning::FindConnection(set<int> &visited, AB next_ABnet, map<int, set<int>> &next_connections)
{
    next_ABnet->cells.clear();
    for (const auto &connection : next_connections)
    {
        int current_cell = connection.first;
        if (!visited.count(current_cell))
        {
            set<int> current_net;
            DFS(current_cell, visited, current_net, next_connections);
            next_ABnet->cells.push_back(current_net);
        }
    }
}
```

In accordance with the definition provided in Listing 16, it can be demonstrated that there are some independent blocks other than A or B, and that they can be added into A or B.

Listing 16: To merge the other block into A or B

```
void Partitioning::ABMerge(AB next_ABnet)
{
    if (!next_ABnet->cells.empty())
```

```
    {
        if (next_ABnet->cells.size() > 2)
        {
            for (int i = 2; i < next_ABnet->cells.size(); i++)
            {
                if (next_ABnet->cells[0].size() < next_ABnet->cells[1].size())
                {
                    for (int cell : next_ABnet->cells[i])
                        next_ABnet->cells[0].insert(cell);
                }
                else
                {
                    for (int cell : next_ABnet->cells[i])
                        next_ABnet->cells[1].insert(cell);
                }
                next_ABnet->cells[i].clear();
            }
        }
    }
    for (int i = 1; i <= circuit->cell_count; i++)
    {
        set<int> cell;
        cell.clear();
        cell.insert(i);
        if (!IsFindABnet(next_ABnet, i))
        {
            if (next_ABnet->cells.size() < 2)
                next_ABnet->cells.push_back(cell);
            else if (next_ABnet->cells[0].size() < next_ABnet->cells[1].size())
                next_ABnet->cells[0].insert(i);
            else
                next_ABnet->cells[1].insert(i);
        }
    }
}
```

In accordance with the definition provided in Listing 17, the objective is to ascertain the existence of the cell in either the A or B block.

Listing 17: Find the cell in A or B

```
bool Partitioning::IsFindABnet(AB next_ABnet, int cell)
{
    // std::cout << "IsFindABnet ok" << endl;
    for (auto &i : next_ABnet->cells)
    {
        if (i.count(cell))
        {
            return true;
        }
```

```
    }
    // std::cout << "FindAB false ok" << endl;
    return false;
}
```

In accordance with the definition provided in Listing [S], the function is to prevent the incorrect cut-size.

Listing 18: Shrink the redundant cut

```
void Partitioning::ShrinkCut(AB now_ABnet)
{
    auto iter = now_ABnet->cut.begin();
    while (iter != now_ABnet->cut.end())
    {
        if (IsRedundantCut(circuit->nets[*iter], now_ABnet->cells))
        {
            now_ABnet->cut_size--;
            iter = now_ABnet->cut.erase(iter);
        }
        else
        {
            ++iter;
        }
    }
}

bool Partitioning::IsRedundantCut(set<int> net, vector<set<int>> cells)
{
    int countA = 0;
    int countB = 0;
    for (auto &i : net)
    {
        if (!cells[0].count(i)) // can't find in A
            countA++;
        else
            countB++;
    }
    if (countA == 0 || countB == 0)
        return true;
    else
        return false;
}
```

4. The hardness of this assignment / I overcome it

1. Segmentation fault when pointer to the struct contain container such as set and vector.
   Ans: If there are vector, set or map in structure, it should be allocate information or give their Initialization. Otherwise, it would cause segmentation. And before searching the container, it would recommand to check it is not empty.
   C++ 结构体中包含容器 push_back 异常
   使用带 map 容器的 struct 结构体指针引发的段错误

2. Makefile missing separator. Stop.
   Ans: Set tabstop = 4

5. Suggestions

No.