FPGA Architecture & CAD

Programming Assignment - Boolean network mapper

Student ID : 113062632

Name : 吳晟光

我使用了 Greedy Algorithm 來完成，此報告將詳細介紹程式碼的結構以及相關的設計決策。

結構

1. **Node structure**：每個節點包含 ID、輸入節點列表、輸出節點列表，以及用於標記是否為 pimary input 和已被合成的標誌。
2. **Class BooleanNetwork**：此類包含添加節點、拓撲排序、LUT 的合成和 print network 的功能。

演算法

1. **拓撲排序**：首先，對節點進行拓撲排序，以確保每個節點的輸入節點在其自身之前被處理。
2. **合併節點到 LUT**：
    a. 根據拓撲排序的順序，把每個節點，從僅包含該節點本身開始創建一個 LUT。
    b. 然後嘗試將其輸入節點加入該 LUT。優先加入 input 較少的輸入節點，對於每個輸入 node，計算 currentInputCount = 兩 node 合併後的

input 數量，若 currentInputCount > K 則停止，否則換下一個輸入節點。

c. 如果某個輸入節點已被其他 LUT 合成，則跳過。

3. **創建新的 LUT 節點**：

a. 步驟 2 會找出當前 node 和其 inputnode 能組合成的最大 LUT，分配一個新的節點 ID，並標記為未合成節點。

b. 把這個 LUT(新 node)，insert 一樣按照拓樸排序的 insert 進排序好的 array 中，讓新的 node 也可以再往上找它的 input 有無合成機會

c. 把被包含在此 LUT 的任何節點的 node 標記為已被合成，避免再次被合成

d. 記錄新節點的原始 node，用來在輸出時方便追朔到原始節點

e. 把輸入節點和輸出節點在此 LUT 內部的 node 的輸入和輸出修改為此 LUT

f. 若排序好的 array 還沒跑完所有點，回到步驟 2

使用此方法的原因

1. **負擔小**：貪婪算法避免了回溯或探索所有可能的組合，而是基於當前的可用輸入直接選擇，從而減少了計算負擔

2. **局部合成**：該問題可以拆解為小範圍的局部決策，而透過每次把合成的 LUT 放回排序 array，再次跑一遍有無合成機會，我認為應該可以十分接近 optimal solution

程式執行

透過 make 指令 compile

輸入:./mapper <input file path> <output file path> <K>

code

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <unordered_map>
#include <vector>
#include <list>
#include <algorithm>

using namespace std;

struct Node {
    int id;
    vector<int> inputs;
    vector<int> outputs;
    bool isPrimary = false;
    bool used = false;

    Node(const int node, const vector<int>& inputIds = {}, bool primary = false)
        : id(node), inputs(inputIds), isPrimary(primary) {}
};

class BooleanNetwork {
public:
    unordered_map<int, Node*> nodes;
    unordered_map<int, int> outputMap;

    void addNode(const Node& node) {
        int nodeId = node.id;
        if (nodes.find(nodeId) == nodes.end()) {
            nodes[nodeId] = new Node(node.id, node.inputs, node.isPrimary);
        }
        nodes[nodeId]->inputs = node.inputs;
        nodes[nodeId]->isPrimary = node.isPrimary;

        for (int inputId : node.inputs) {
            if (nodes.find(inputId) == nodes.end()) {
                nodes[inputId] = new Node({ inputId });
            }
            nodes[inputId]->outputs.push_back(nodeId);
        }
    }

    vector<int> topologicalSort() {
        unordered_map<int, int> inDegree;
        for (const auto& pair : nodes) {
            inDegree[pair.first] = 0;
        }

        for (const auto& pair : nodes) {
            for (int output : pair.second->outputs) {
```

```
                    inDegree[output]++;
                }
            }

            list<int> zeroInDegreeQueue;
            for (const auto& pair : inDegree) {
                if (pair.second == 0) {
                    zeroInDegreeQueue.push_back(pair.first);
                }
            }

            vector<int> sortedNodes;
            while (!zeroInDegreeQueue.empty()) {
                int node = zeroInDegreeQueue.front();
                zeroInDegreeQueue.pop_front();
                sortedNodes.push_back(node);

                for (int output : nodes[node]->outputs) {
                    if (--inDegree[output] == 0) {
                        zeroInDegreeQueue.push_back(output);
                    }
                }
            }

            if (sortedNodes.size() != nodes.size()) {
                cerr << "Error: The graph has a cycle, topological sort is not possible." << endl;
            }

            return sortedNodes;
        }

        void synthesizeLUTs(int K) {
            vector<int> sortedNodes = topologicalSort();
            int newLUTId = nodes.size() + 1;
            int oldLUTsize = nodes.size();
            list<int> extendedSortedNodes(sortedNodes.begin(), sortedNodes.end());

            for (auto it = extendedSortedNodes.begin(); it != extendedSortedNodes.end(); ++it) {
                int nodeId = *it;
                Node* node = nodes[nodeId];

                if (node->isPrimary || node->used) continue;

                vector<int> currentLUT = { nodeId };
                int currentInputCount = node->inputs.size();
                vector<int> mergedLUTs;

                for (int inputId : node->inputs) {
                    Node* inputNode = nodes[inputId];

                                bool canMerge = true;
                                for (int outputId : inputNode->outputs) {
                                    if (outputId != nodeId) {
                                        canMerge = false;
                                        break;
```

```
                                }
                            }

                            if (!canMerge) continue;

                            if (inputNode->isPrimary) {
                                continue;
                            }
                            else if (!inputNode->used) {
                                int inputSize = inputNode->inputs.size();
                                for (int input : inputNode->inputs) {
                                    if (find(node->inputs.begin(), node->inputs.end(), input) != node->inputs.end()) {
                                        inputSize--;
                                    }
                                }
                                currentInputCount = currentInputCount - 1 + inputSize;
                                if (currentInputCount <= K) {
                                    currentLUT.push_back(inputId);
                                    if (inputNode->id > oldLUTsize) {
                                        mergedLUTs.push_back(inputId);
                                    }
                                    inputNode->used = true;
                                }
                                else {
                                    continue;
                                }
                            }
                        }

                        if ((currentLUT.size() > 0 && nodeId <= oldLUTsize) || (currentLUT.size() > 1 && nodeId > oldLUTsize)) {
                            node->used = true;
                            vector<int> newnode_in;
                            for (int old : currentLUT) {
                                for (int inpu : nodes[old]->inputs) {
                                    if (find(newnode_in.begin(), newnode_in.end(), inpu) == newnode_in.end() &&
                                        find(currentLUT.begin(), currentLUT.end(), inpu) == currentLUT.end()) {
                                        newnode_in.push_back(inpu);
                                    }
                                }
                            }

                            Node* newLUTNode = new Node(newLUTId, newnode_in, false);
                            newLUTNode->outputs = node->outputs;
                            nodes[newLUTId] = newLUTNode;

                            for (int oldNodeId : currentLUT) {
                                for (int outputId : nodes[oldNodeId]->outputs) {
                                    Node* outputNode = nodes[outputId];
                                    replace(outputNode->inputs.begin(), outputNode->inputs.end(), oldNodeId, newLUTId);
                                }
                            }
```

```
157.
158.                             for (int oldNodeId : currentLUT) {
159.                                 for (int inputId : nodes[oldNodeId]->inputs) {
160.                                     Node* inputNode = nodes[inputId];
161.                                     replace(inputNode->outputs.begin(),
    inputNode->outputs.end(), oldNodeId, newLUTId);
162.                                 }
163.                             }
164.
165.                             auto nextIt = next(it);
166.                             extendedSortedNodes.insert(nextIt, newLUTId);
167.                             outputMap[newLUTId] = nodeId;
168.                             newLUTId++;
169.                         }
170.                     }
171.                 }
172.
173.         bool readFromFile(const string& inputFile) {
174.             ifstream inFile(inputFile);
175.             if (!inFile.is_open()) {
176.                 cerr << "Failed to open input file: " << inputFile << endl;
177.                 return false;
178.             }
179.
180.             string line;
181.             int nodeId, numNodes, numPIs, numPOs;
182.
183.             // Read header
184.             getline(inFile, line);
185.             istringstream headerStream(line);
186.             string name;
187.             headerStream >> name >> numNodes >> numPIs >> numPOs;
188.
189.             // Read Primary Inputs
190.             for (int i = 0; i < numPIs; i++) {
191.                 inFile >> nodeId;
192.                 addNode(Node(nodeId, {}, true));
193.             }
194.
195.             // Read Primary Outputs
196.             vector<int> primaryOutputs;
197.             for (int i = 0; i < numPOs; i++) {
198.                 inFile >> nodeId;
199.                 primaryOutputs.push_back(nodeId);
200.             }
201.
202.             // Read remaining nodes
203.             while (getline(inFile, line)) {
204.                 istringstream nodeStream(line);
205.                 nodeStream >> nodeId;
206.                 int inputId;
207.                 vector<int> inputs;
208.
209.                 while (nodeStream >> inputId) {
210.                     inputs.push_back(inputId);
211.                 }
212.                 addNode(Node(nodeId, inputs, false));
213.             }
```

```
214.
215.             inFile.close();
216.             return true;
217.         }
218.
219.             int getOriginalNode(int nodeId) {
220.                 while (outputMap.find(nodeId) != outputMap.end()) {
221.                     nodeId = outputMap[nodeId];
222.                 }
223.                 return nodeId;
224.             }
225.
226.             void printNetwork(const string& outputFile) {
227.                 ofstream outFile(outputFile);
228.                 if (!outFile.is_open()) {
229.                     cerr << "Failed to open output file: " << outputFile << endl;
230.                     return;
231.                 }
232.
233.                 for (const auto& pair : nodes) {
234.                     Node* node = pair.second;
235.
236.                     if (!node->used && outputMap.find(node->id) !=
    outputMap.end()) {
237.                         int outputNodeId = getOriginalNode(node->id);
238.                         const vector<int>& inputs = node->inputs;
239.
240.                         outFile << outputNodeId;
241.                         for (int inputId : inputs) {
242.                             int inputNodeId = getOriginalNode(inputId);
243.                             outFile << " " << inputNodeId;
244.                         }
245.                         outFile << endl;
246.                     }
247.                 }
248.
249.                 outFile.close();
250.             }
251.
252.
253.             ~BooleanNetwork() {
254.                 for (auto& pair : nodes) {
255.                     delete pair.second;
256.                 }
257.             }
258.         };
259.
260.         int main(int argc, char* argv[]) {
261.             if (argc != 4) {
262.                 cerr << "Usage: " << argv[0] << " <input file> <output file>
    <K>" << endl;
263.                 return 1;
264.             }
265.
266.             string inputFile = argv[1];
267.             string outputFile = argv[2];
268.             int K = stoi(argv[3]);
269.
```

```
270.            BooleanNetwork network;
271.
272.            if (!network.readFromFile(inputFile)) {
273.                  return 1;
274.            }
275.
276.            network.synthesizeLUTs(K);
277.            network.printNetwork(outputFile);
278.
279.            return 0;
280.      }
```