EDA final project presentation

Problem B: Power & Timing optimization using MBFF

Presented by B11007048 李兆翔
B11007050 李子杰

Agenda

01. **Introduction**

02. **Data structures & Algorithms**

03. **Results**

1. Introduction

- **Multi-bit FF decomposition / composition (map)**
- **Minimize PPA, #violations of bin density, and TNS**

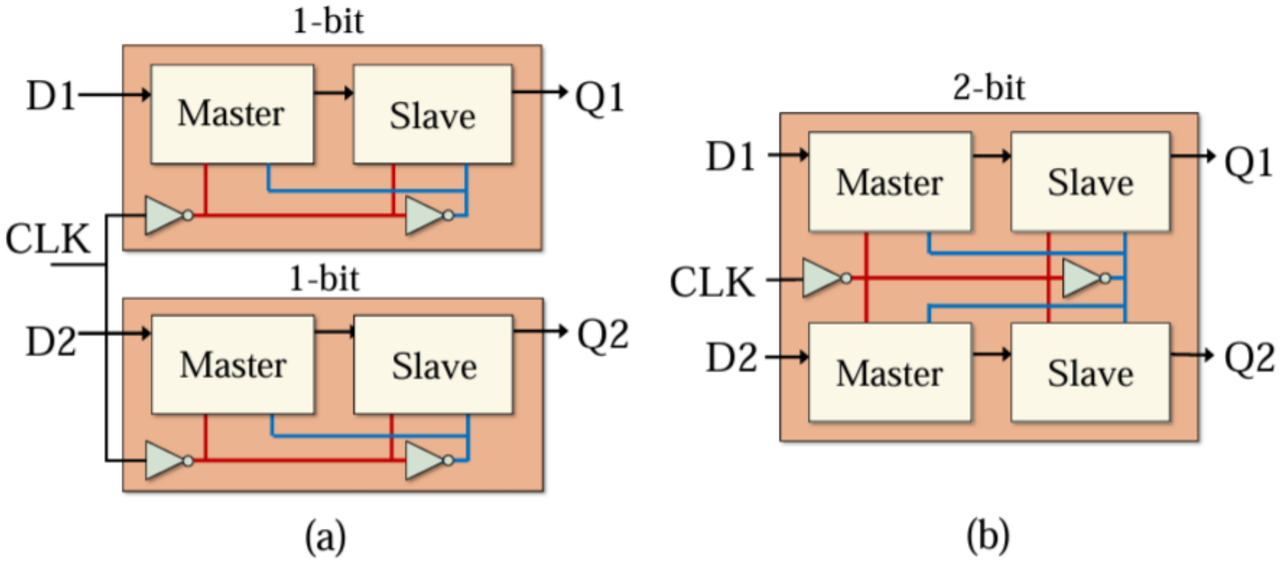

Figure 1.1: (a) Structure of two 1-bit flip-flops. (b) Structure of 2-bit MBFF merging the two 1-bit flip-flops in (a).

ref: https://s-space.snu.ac.kr/bitstream/10371/196453/1/000000177339.pdf

Ideation process

01

Parser

02

ILP...?

03

Simulated Annealing

Parsing input data

Global parameters: treat as global variables

sampleCase

```
1  Alpha 10
2  Beta 10
3  Gamma 0.0000002
4  Lambda 10
5  DieSize 0 0 23475 23280
```

->

```
11  double Alpha_, Beta_, Gamma, Lambda;
12  double die_Lx, die_Ly, die_Rx, die_Ry;
13  double BinWidth, BinHeight, BinMaxUtil;
14  double DisplacementDelay;
```

⋮

Parsing input data

Cell library, Instances, Nets : using “unordered map” to save particular data structures

```
NumInput 2
Input in 8344 22840
Input clk 0 1970
NumOutput 1
Output out 23075 11410
```

⋮

input Data

->

```
24   unordered_map<string, Pin> Inputs;
25   double NumInput;
26   unordered_map<string, Pin> Outputs;
27   double NumOutput;
```

```
while (buffer == "GatePower") {
    fin >> name >> buffer;
    FlipFlop_Cells[name].GatePower = stod(buffer);
    fin >> buffer;
}
```

e.g.: accessing FF cell library using unordered map

Data structures

(1) Pin: Store pin’s name, and x, y coordinates

```
class Pin {
public:
    Pin(){};
    Pin(string name, double x, double y) : name(name), x(x), y(y){};
    ~Pin(){};
    string name;
    double x;
    double y;
};
```

Data structures

(2) FFs: “FlipFlop” is for cell library, and “Inst_FlipFlop” is for storing instances

```
class Inst_FlipFlop {
    public:
     string name;
     FlipFlop* type;
     double clktype;
     double x, y;
     unordered_map<string, double> TimingSlack;
     Inst_FlipFlop() {}
     Inst_FlipFlop(string name, string type_in, double x, double y)
         : name(name), x(x), y(y) {
         type = &FlipFlop_Cells[type_in];
     };
};
```

```
53  class FlipFlop {
54      public:
55       FlipFlop(){};
56       FlipFlop(double bits, string name, double width,
57                double height, double pincount)
58           : bits(bits), name(name), width(width),
59             height(height), pincount(pincount) {
60           Qpindelay = 0;
61           GatePower = 0;
62       };
63       ~FlipFlop(){};
64       double bits;
65       string name;
66       double width, height;
67       double pincount;
68       unordered_map<string, Pin> pins;
69       double Qpindelay;
70       double GatePower;
71       void addPin(const Pin& pin) {
72           pins.insert_or_assign(pin.name, pin);
73       }
74  };
```

Data structures

(3) Gates: “Gate” is for cell library, and
“Inst_Gate” is for storing instances

```
class Inst_Gate {
    public:
     string name;
     Gate* type;
     double x, y;
     Inst_Gate() {}
     Inst_Gate(string name, string type_in, double x, double y)
         : name(name), x(x), y(y) {
         type = &Gate_Cells[type_in];
     };
};
```

```
class Gate {
    private:
    public:
     Gate(){};
     Gate(string name, double width,
          double height, double pincount)
         : name(name), width(width),
           height(height), pincount(pincount) {
         GatePower = 0;
     };
     ~Gate(){};
     string name;
     double width;
     double height;
     double pincount;
     vector<Pin> pins;
     double GatePower;
     void addPin(const Pin& pin) {
         pins.emplace_back(pin);
     }
};
```

Data structures

(4) Nets & Placement Rows:

```
class PlacementRows {
public:
    PlacementRows(){};
    PlacementRows(double x, double y, double w, double h, double num)
        : x(x), y(y), siteW(w), siteH(h), NumOfSites(num){};
    ~PlacementRows(){};
    double x;
    double y;
    double siteW;
    double siteH;
    double NumOfSites;
};
```

```
class Net {
public:
    Net(){};
    Net(string name, double pincount) :
    name(name), pincount(pincount){};
    ~Net(){};
    string name;
    double pincount;
    vector<string> connect;
    void addPin(string& pinname) {
        // stringstream ss(pinname);
        // string a, b;
        // getline(ss, a, '/');
        // getline(ss, b);
        // cout << a << " " << b << "\n";
        connect.emplace_back(pinname);
    }
};
```

Using Cplex

- **Set objective function**

Minimize $\sum_{\forall i \in FF} \left(\alpha \cdot TNS(i) + \beta \cdot Power(i) + \gamma \cdot Area(i)\right) + \lambda \cdot D$

- **How to set Constraints ? ...**

only same clks can bank together

nets are corrected connect

Instances are placed on-site

Instances are placed without overlap

more and more ...

```
int main(int argc, char** argv) {
    try {
        IloModel model(env);
        IloNumVarArray x(env, NumFlipFlops, die_Lx, die_Rx, ILOFLOAT);
        IloNumVarArray y(env, NumFlipFlops, die_Ly, die_Ry, ILOFLOAT);
        IloNumArray clk_type(env,NumFlipFlops);
        IloArray<IloBoolVarArray> isMerged(env, NumFlipFlops);
        IloArray<IloBoolVarArray> isClkDifferent(env, NumFlipFlops);

        for (int i = 0; i < NumFlipFlops; ++i) {
            model.add(x[i] ⩾ die_Lx);
            model.add(x[i] ⩽ die_Rx);
            model.add(y[i] ⩾ die_Ly);
            model.add(y[i] ⩽ die_Ry);
        }

        int index = 0;
        for (auto &&i : FlipFlops){
            clk_type[index] = i.second.clktype;
        }

        for (int i = 0; i < NumFlipFlops; ++i) {
            isMerged[i] = IloBoolVarArray(env, NumFlipFlops);
            isClkDifferent[i] = IloBoolVarArray(env, NumFlipFlops);
            for (int j = 0; j < NumFlipFlops; ++j) {
                isMerged[i][j] = IloBoolVar(env);
                isClkDifferent[i][j] = IloBoolVar(env);
            }
        }

        for (int i = 0; i < NumFlipFlops; ++i) {
            for (int j = 0; j < NumFlipFlops; ++j) {
                if (i ≠ j) {
                    model.add(isClkDifferent[i][j] == (clk_type[i] ≠ clk_type[j]));
                    model.add(IloIfThen(env,isClkDifferent[i][j] == 1,isMerged[i][j] == 0));
                    model.add(IloIfThen(env,isMerged[i][j] == 1,x[i] == x[j] && y[i] == y[j]));
                }
            }
        }

        for (int i = 0; i < NumFlipFlops; ++i) {
            IloExpr sumMerged(env);
            for (int j = 0; j < NumFlipFlops; ++j) {
                sumMerged += isMerged[i][j];
            }
            model.add(sumMerged ⩽ 2);
        }
```

Simulated Annealing

1.cost

- The cost is determined by following criteria:

$$\sum_{\forall i \in FF} \left(\alpha \cdot TNS(i) + \beta \cdot Power(i) + \gamma \cdot Area(i)\right) + \lambda \cdot D$$

```cpp
// reg1 reg2 SVT_FF_2 reg3 SVT_FF_3 ...
inline double FF_expression::cost() {
    double TNS = 0;
    double Power = 0;
    double Area = 0;
    int numOverflow = 0;

    TNS = calcTNS();
    for (int i = 0; i < this->expr.size(); i++) {
        string t = expr[i];
        if (Is_FlipFlop_Cell(t)) {
            Area += FlipFlop_Cells[t].height * FlipFlop_Cells[t].width;
            Power += FlipFlop_Cells[t].GatePower;
            numOverflow += checkOverflow(t, expr[i-1]); // type, inst
        }
    }
    return TNS * Alpha_ + Power * Beta_ + Area * Gamma + numOverflow * Lambda;
}
```

Simulated Annealing

2.Expression

- contains idea from PA2
- An expression with “instance” and “type”
- Easier to do perturbation

```
class FF_expression {
public:
    FF_expression(){};
    ~FF_expression(){};
    void set(string&);
    void Perturbation(int mode);
    bool Is_valid();
    vector<string> expr;
};
```

```
364   // reg1 reg2 SVT_FF_2 reg3 SVT_FF_1 ...
```

Expression validation

- To maintain the correctness of expression, we require validation in the expression.

```
// reg1 reg2 FF2 reg3 FF1
inline bool FF_expression::Is_valid() {
    double type_bits = 0, count = 0, clk_type = -1;
    for (auto&& i : this→expr) {
        if (Is_FlipFlop_Cell(i)) {
            if ((FlipFlop_Cells[i].bits ≠ type_bits)
            || (type_bits ≠ count) || count == 0) {
                return false;
            }
            type_bits = 0;
            count = 0;
            clk_type = -1;
        } else if (Is_FlipFlop_Inst(i)) {
            if (type_bits ≠ 0) {
                if ((FlipFlops[i].type→bits ≠ type_bits)
                || (FlipFlops[i].clktype ≠ clk_type)) {
                    return false;
                }
            } else {
                type_bits = FlipFlops[i].type→bits;
                clk_type = FlipFlops[i].clktype;
            }
            count++;
        } else {
            cout << "Error in valid ⇒ " << i;
            exit(1);
        }
    }
    return true;
}
```

Simulated Annealing

3.Perturbation

- Randomly choose a FF type from cell library, then merge(bank) or decompose(debank) the chosen instances

e.g.: Expr = reg1, reg2, SVT_FF_2, reg3, SVT_FF_1, reg4, SVT_FF_1

type = SVT_FF_2 (randomly choose)

the chosen instance(s) = reg3, reg4 (banking)

Expr = reg1, reg2, SVT_FF_2, reg3, reg4, SVT_FF_2

Simulated Annealing

4. Experimental results

```
zijie@LAPTOP-O6EDFFF4:/mnt/c/Users/zijie/Desktop/Code/EDA/proj$ ./sanity sampleCase test.out
Read testcase...
Read output...
Check Inst <name> <libcell> <x> <y> format...
Check From/pin map To/pin  format...
Start checking...
check libType
check pins are in design
check pins are correct connected
check same CLK for banking
check netlist... step1
check netlist... step2
Checking pass!
```

test.out

```
1   CellInst 4
2   Inst reg5 SVT_FF_1 3615 3600
3   Inst reg6 SVT_FF_1 1278 3600
4   Inst reg7 SVT_FF_1 1278 6000
5   Inst reg8 SVT_FF_1 5952 3600
6   reg4/Q map reg5/Q
7   reg4/CLK map reg5/CLK
8   reg4/D map reg5/D
9   reg2/Q map reg6/Q
10  reg2/CLK map reg6/CLK
11  reg2/D map reg6/D
12  reg3/Q map reg7/Q
13  reg3/CLK map reg7/CLK
14  reg3/D map reg7/D
15  reg1/Q map reg8/Q
16  reg1/CLK map reg8/CLK
17  reg1/D map reg8/D
18
```

Division of labor

	B11007048 李兆翔	B11007050 李子杰
文獻研究、討論	✓	✓
Parser		✓
ILP		✓
SA	✓	✓
製作簡報	✓	✓
報告	✓	

Thank you very much!