Technology Mapping

PA3

Revision

- 04/29
  - The output format has been changed to Verilog (.v)
  - Hints have been added
- 04/30
  - Various cells are defined in tech.lib
  - Hints -> Programming Guidance

Introduction

- Technology mapping is a step of
  - taking a (technology independent) logic netlist as the input and
  - expressing the netlist using a set of gates from a technology library
- With the library, a gate often have multiple configurations, each offering different area, delay, and power characteristics
- This assignment requires you to develop a program that performs technology mapping and determines the optimal gate configurations to optimize the area and power, while satisfying a given timing constraint

Problem Formulation

- Inputs
  - Netlist in blif format
  - Simplified technology library
  - Timing constraint
- Output
  - A functionally equivalent netlist implemented using gates from the technology library
- Objective
  - Minimize the weighted sum of total area and total power (0.5×total area+0.5×total power) while satisfying the given timing constraint

Input 1 – Netlist (.blif)

- Netlist is given in the blif format
- You can use ABC to parse the netlist or develop your own parser
- Optimization is allowed

- The timing constraint is specified on the first line as a comment
    - E.g.: # 15

Input 2 – Technology Library (tech.lib)

- **INVs**

Pattern Type	Delay	Area	Power
INV1	1	1	20.92
INV2	3.67	1.36	1
INV3	1.67	1.36	2.75

- **2-input NANDs**

Pattern Type	Delay	Area	Power
NAND1	18.4	1	1
NAND2	1	1.27	3.48
NAND3	7	1.27	3.09
NAND4	7.73	1	1.55

Output – Netlist (.v)

- Output the mapped netlist in the verilog format
- You cannot modify PI and PO names

```
// Timing constraint: xxx
// Total power: xxx
// Total area: xxx
…
…
…
NAND1 g217(.A(n260), .B(n252), .Y(G431gat)); // slack
INV1   g218(.A(n229), .Y(n262)); // slack
…
…
```

Note

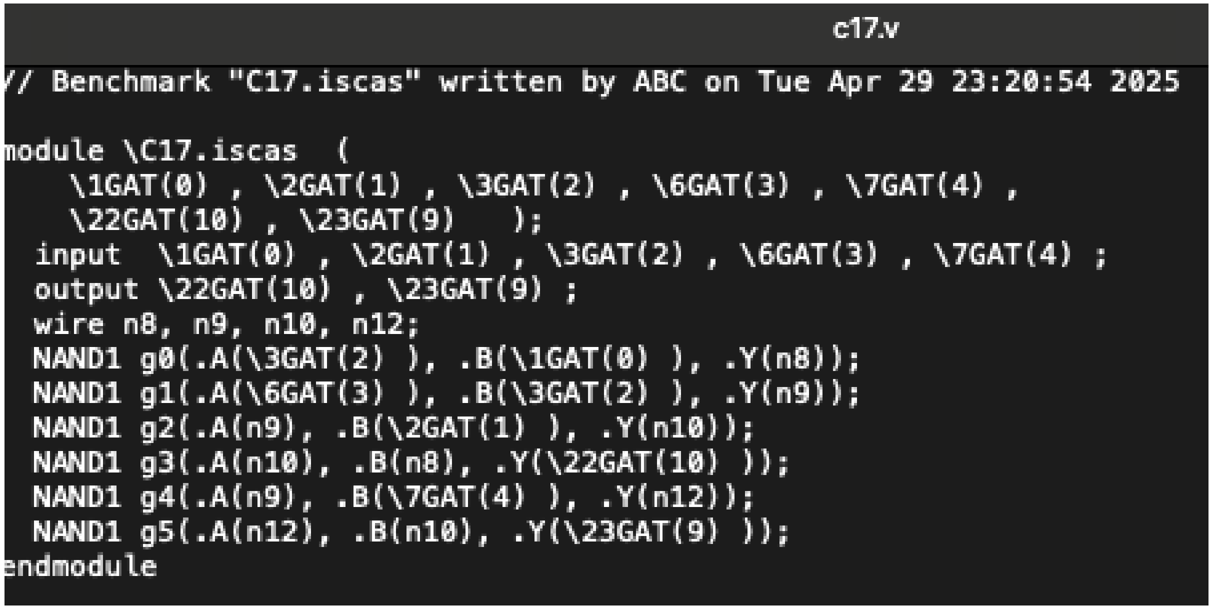
- Timing constraint is the required time of all POs
  - Use topological timing analysis to compute (arrival time, required time, slack)
- Total power
  - Sum of the power of all gates
- Total area
  - Sum of the area of all gates
- Zero-cost buffer is allowed, named BUFF

Delivery & Due Date

- A zip file including
  - Your source code and a ReadMe describing how to compile and run your program
  - Your mapping results (.v) for all the given benchmarks
- Run time limit
  - 1 hour for each benchmark
- Due on 2025/5/27 (Tue)

Programming Guidance

- How to generate an initial mapped netlist with ABC
  - abc> read_library pa3.genlib
  - abc> read_blif c432.blif
  - … (optimization script)
  - abc> map
  - abc> write_verilog c432.v



c17.v

```
// Benchmark "C17.iscas" written by ABC on Tue Apr 29 23:20:54 2025

module \C17.iscas  (
    \1GAT(0) , \2GAT(1) , \3GAT(2) , \6GAT(3) , \7GAT(4) ,
    \22GAT(10) , \23GAT(9)   );
  input  \1GAT(0) , \2GAT(1) , \3GAT(2) , \6GAT(3) , \7GAT(4) ;
  output \22GAT(10) , \23GAT(9) ;
  wire n8, n9, n10, n12;
  NAND1 g0(.A(\3GAT(2) ), .B(\1GAT(0) ), .Y(n8));
  NAND1 g1(.A(\6GAT(3) ), .B(\3GAT(2) ), .Y(n9));
  NAND1 g2(.A(n9), .B(\2GAT(1) ), .Y(n10));
  NAND1 g3(.A(n10), .B(n8), .Y(\22GAT(10) ));
  NAND1 g4(.A(n9), .B(\7GAT(4) ), .Y(n12));
  NAND1 g5(.A(n12), .B(n10), .Y(\23GAT(9) ));
endmodule
```

Programming Guidance

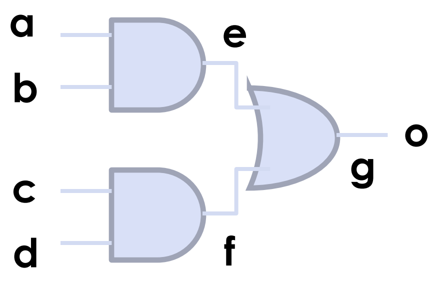
- Once the initial mapped netlist is available, the next step is technology-dependent logic optimization
  - Choose the optimal configuration for each cell

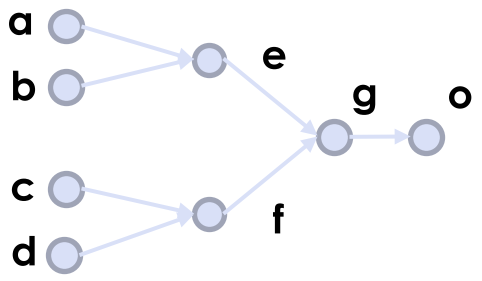
- For Boolean network manipulation, you may need to define a data structure for network representation

- Remaining main tasks
  - Parse the input Verilog file
  - Implement a topological timing analyzer
    - Compute the required time, arrival time, and slack
  - Develop an algorithm for cell selection
  - Implement a power and area evaluator
  - Generate the final Verilog output

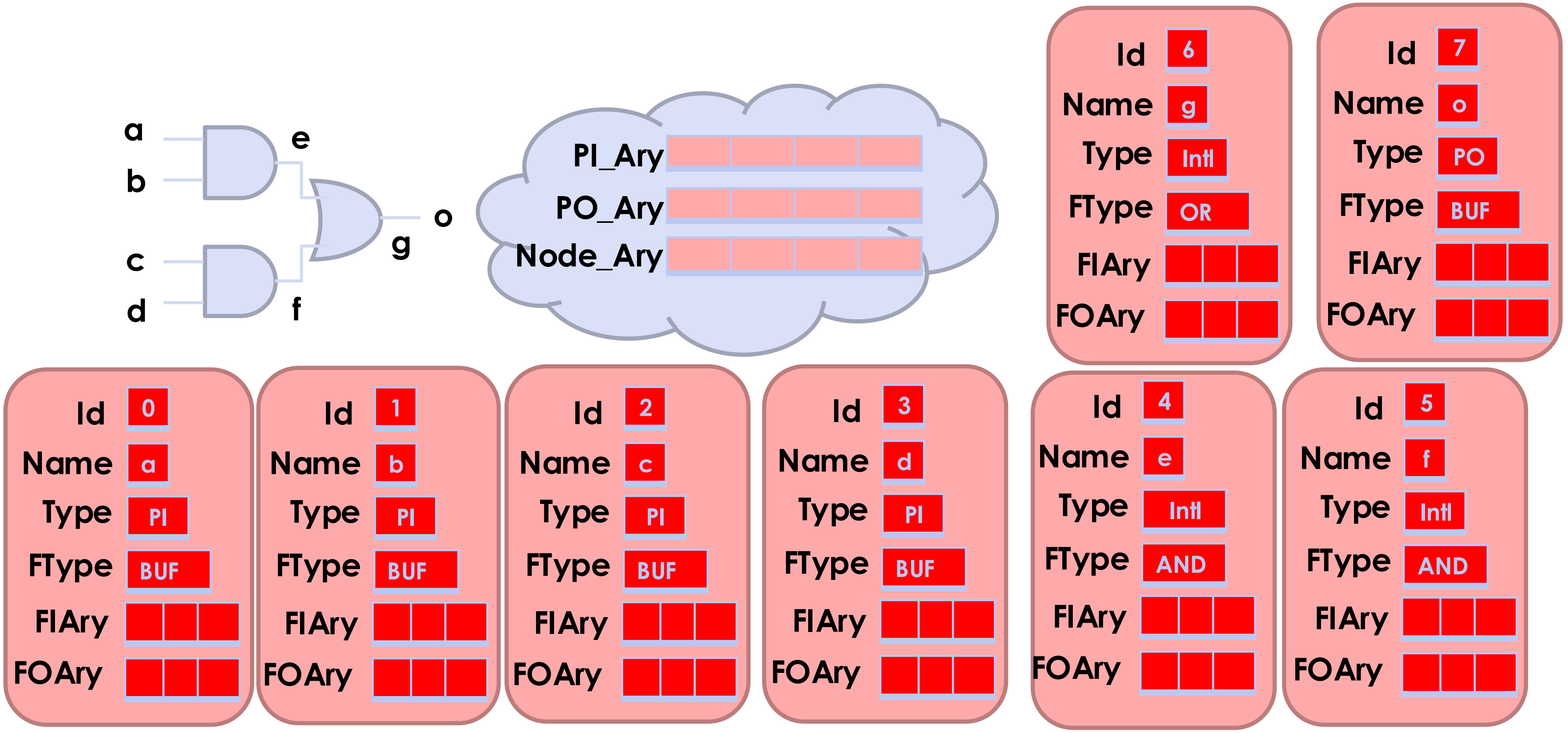
Boolean Network Representation

- A combinational circuit can be modeled as a directed acyclic graph G = {V, E}





Graph Representation



Data Structure

```
typedef struct Bn_Ntk_ Bn_Ntk;
struct Bn_Ntk_{
            Vec_Ptr_t *Node_Ary;

            Vec_Ptr_t *PI_Ary;
            Vec_Ptr_t *PO_Ary;
            Vec_Ptr_t *Key_Ary;

};
```

All the arrays are pointer arrays with the type of Bn_Node *

```
typedef struct Bn_Node_ Bn_Node;

struct Bn_Node_{

      int Id;

      char name[50];

      int Type;

      int FType;

      Vec_Ptr_t *FIAry;

      Vec_Ptr_t *FOAry;

};
```