iEDA-Tutorial 第四期

逻辑综合工具（iMAP）

可测试设计工具（iATPG）

倪利伟、刘俊锋、杨宗霖、林晓泽

2023/09/27

iEDA Tutorial 第四期议程

- Part1 iEDA-iMAP工具介绍 20min （倪利伟）

- Part2 AiMAP工艺映射算法 15min （刘俊锋）

- Part3 并行化逻辑重写算法 15min （杨宗霖）

- Part4 iEDA-iATPG工具介绍 20min （林晓泽）

01 研究内容

02 研究进展

03 未来计划

Design Flow

```
module conv;
reg [31:0] m[0:8192];
reg [12:0] pc;
reg [31:0] acc;
reg[15:0] ir;
always
  begin
    ir = m[pc];
    if(ir[15:13] == 3b'000)
       pc = m[ir[12:0]];
    else if (ir[15:13] == 3'b010)
```

Specifications

Architectural Design

Functional Design

Simulation

Simulation (Circuit)

Emulation (System)

Logic Synthesis

Logic Compiling

Logic Optimization

Tech Mapping

Equivalence Checking

Std Cell Lib

Device Modeling

Parasitic Extraction

DRC/LVS

Physical Design

Floorplan

Placement

Timing Optimization

Clock Tree Synthesis

Routing

ECO

Analysis

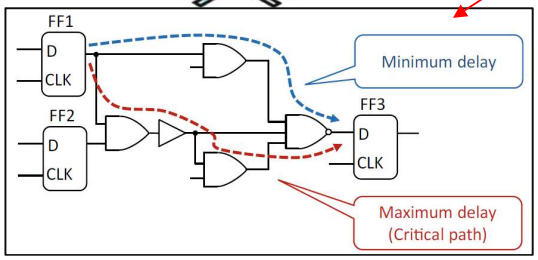
RC Extraction

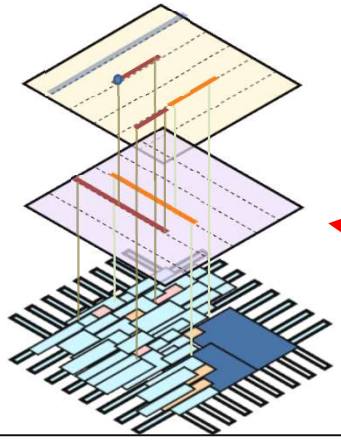
Timing Analysis

Power

Noise

IR Drop





Chip

Package and Test

Fabrication

Layout Processing

Mask Generation

Post Simulation

RET/OPC/ILT

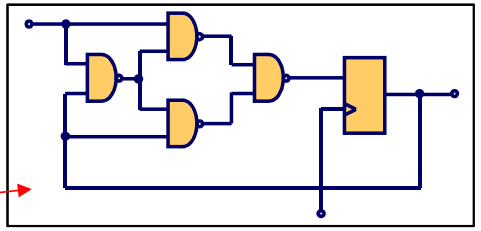
Physical Verification

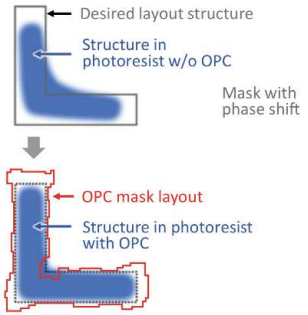
ESD/CD

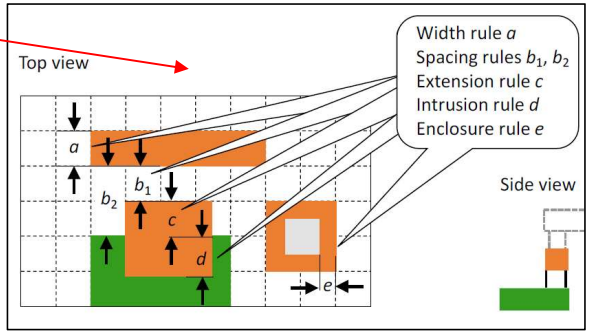
LVS

ERC

DRC (MP)







Design Flow

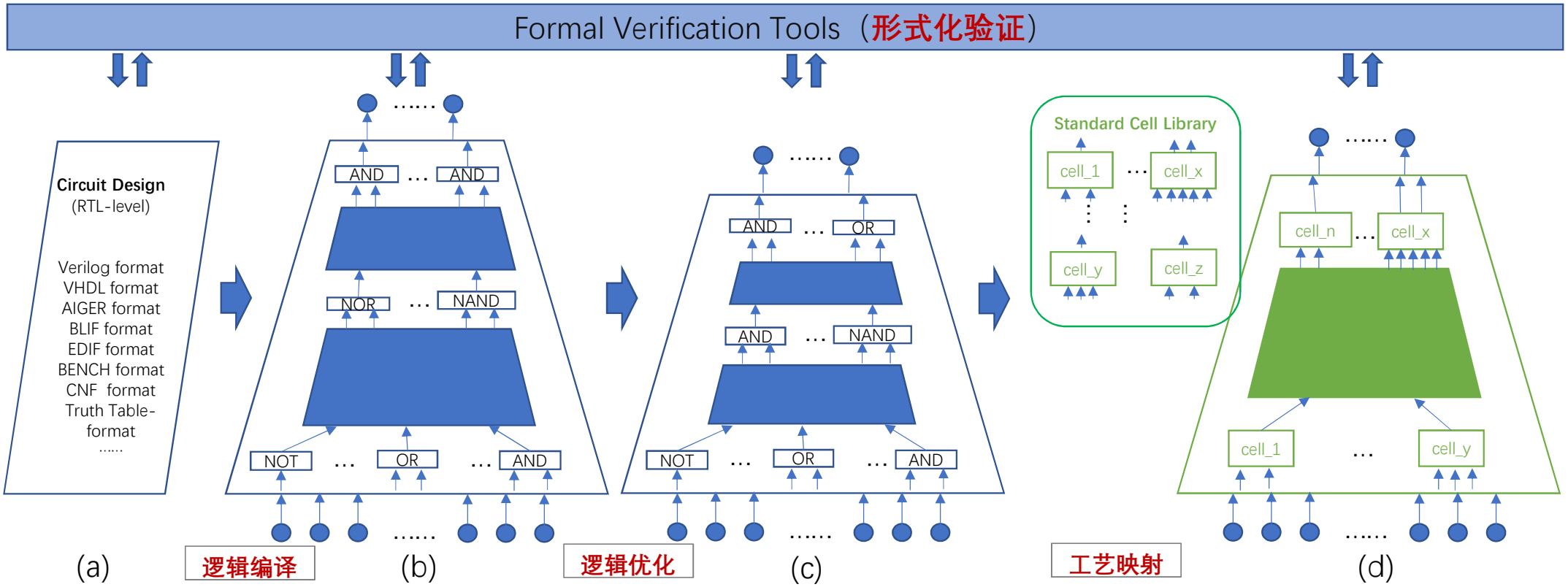


图1 逻辑综合状态流程图

Overview

- Why

  为什么要研发iMAP工具，现有开源工具存在什么问题？

  - Berkeley-abc[1]存在的问题：
    - 代码历史悠久，没有文档
    - 基于c语言编程，可扩展性较差
    - AI+逻辑综合的研究在berkeley-abc受限，研发成本较高
  - EPFL-lsils[2]存在的问题
    - lsils下的库文件主要用于学术研究
    - lsils的逻辑优化和工艺映射算法的runtime以及QoR和abc还存在差距
    - lsils对于liberty文件格式还不支持

[1] https://github.com/berkeley-abc/abc/
[2] https://github.com/lsils

Overview

- How

  怎样研发iMAP工具？需求推进工具不断演进。

  - 取精华，弃糟粕
    - 利用berkeley-abc以及EPFL-lsils的优点
    - 提高代码的可读性
    - 提升算子的可扩展性
    - 在runtime和QoR上接近berkeley-abc
    - 促进AI+逻辑综合的前沿研究

- Goal

科研+工程结合，做创新实用的逻辑综合工具，支持流片

从零开始
创造属于你的
RISC-V®处理器

```
set abc_script  "+read_constr,$SDC_FILE;strash;ifraig;retime,-D,{D},-M,6;strash;dch,-f;map,-p,-M,1,{D},-f;topo;dnsize;buffer,-p;upsize;"
```

```
#define ABC_COMMAND_LIB "strash; ifraig; scorr; dc2; dretime; retime {D}; strash; &get -n; &dch -f; &nf {D}; &put"
#define ABC_COMMAND_CTR "strash; ifraig; scorr; dc2; dretime; retime {D}; strash; &get -n; &dch -f; &nf {D}; &put; buffer; upsize {D}; dnsize {D}; stime -p"
#define ABC_COMMAND_LUT "strash; ifraig; scorr; dc2; dretime; retime {D}; strash; dch -f; if; mfs2"
#define ABC_COMMAND_SOP "strash; ifraig; scorr; dc2; dretime; retime {D}; strash; dch -f; cover {I} {P}"
#define ABC_COMMAND_DFL "strash; ifraig; scorr; dc2; dretime; retime {D}; strash; &get -n; &dch -f; &nf {D}; &put"
```

01 研究内容

02 研究进展

03 未来计划

iMAP 工具原型

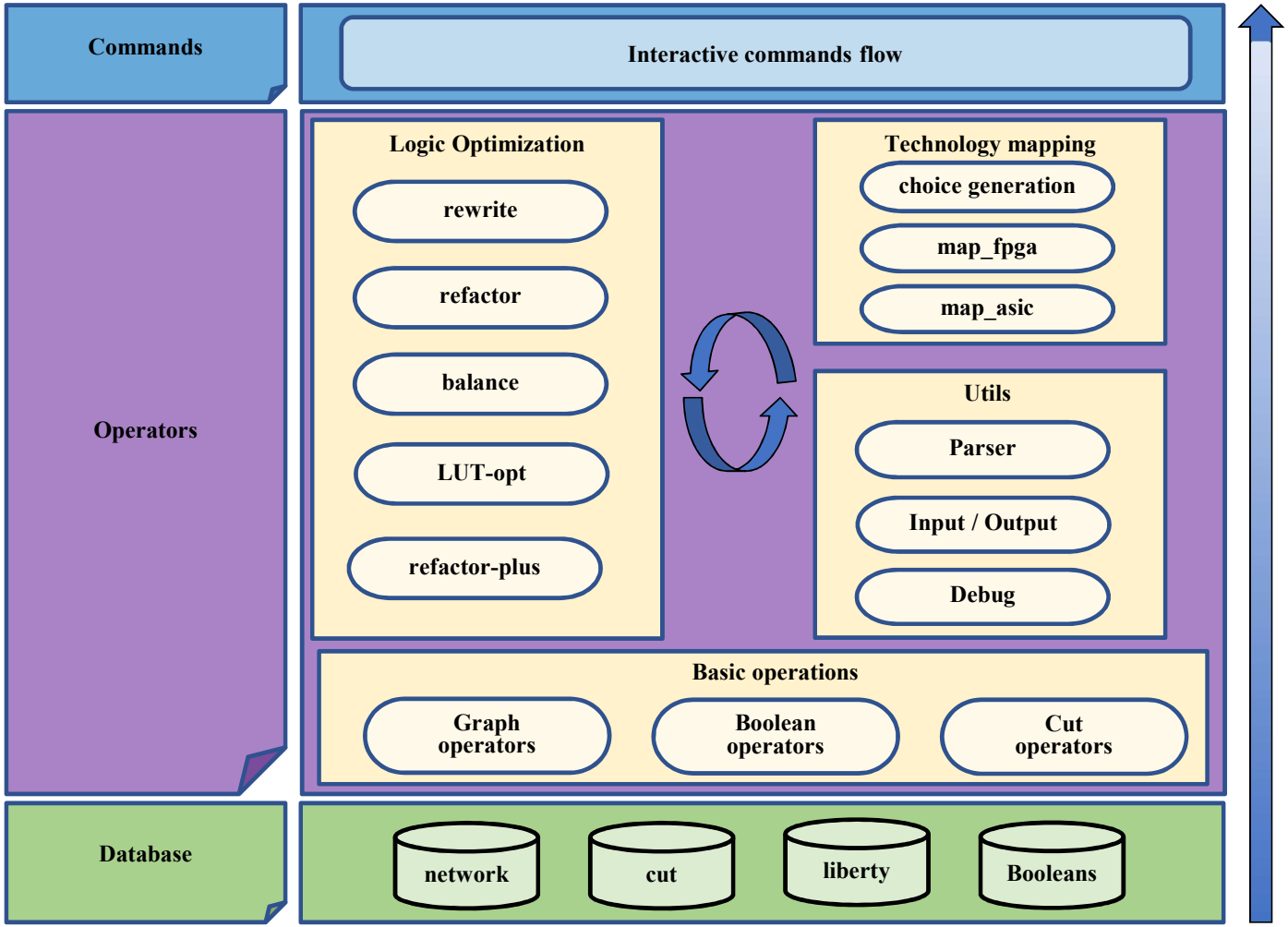
- **iMAP 0.1完成了基本工艺映射算法以及逻辑优化算子**

  当前已支持功能：

  - 数据格式支持：
    - ✓ AIG (And-Inverter Graph) 格式的输入和输出
    - ✓ Verilog / DOT 格式的输出
  - 逻辑综合算子:
    - ✓ Rewrite (基于4-输入真值表的NPN匹配)
    - ✓ Refactor (基于SOP表达式的优化)
    - ✓ Balance (基于AND-tree的平衡算法)
    - ✓ LUT-opt (基于FPGA工艺映射的优化算法)
    - ✓ Map-fpga (基于cut以及多AIG融合成choice Graph的FPGA优化算法)
    - ✓ Map-asic(基于AIG的ASIC优化算法，支持libertyfile，待release)

iMAP工具设计

- 框架设计
  - 扩展性与兼容性
    - add new features
  - 高质量
    - compare with ABC
  - 易读性
    - 文档 + APIs
  - AI + iMAP
    - Commands
    - Python APIs

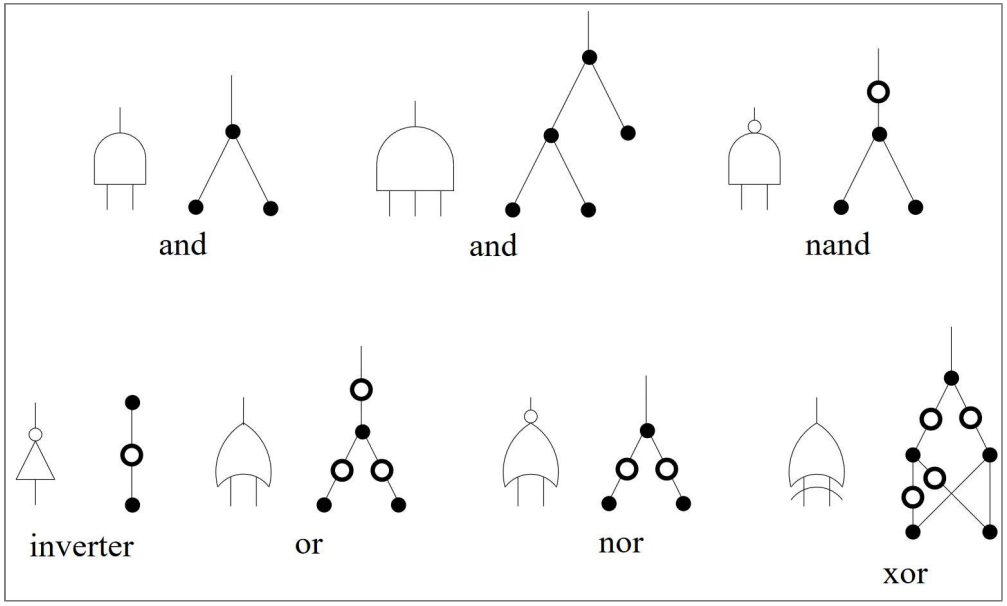


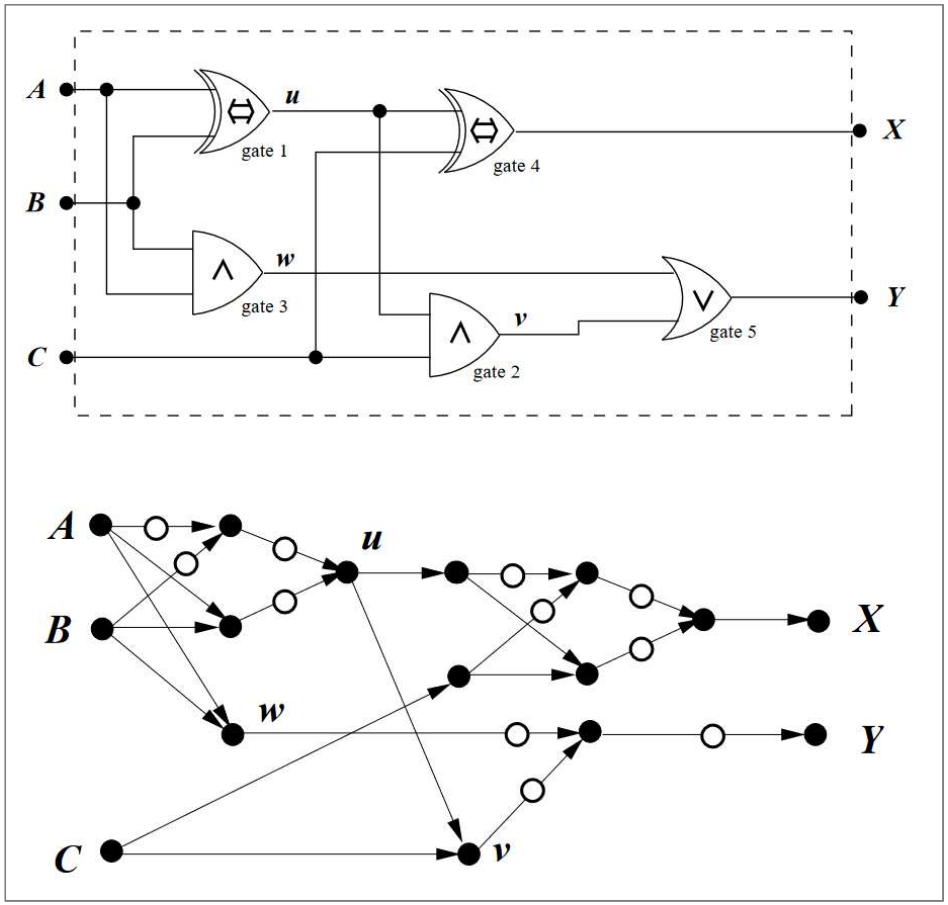
[1] https://github.com/OSCC-Project/iMAP
[2] https://gitee.com/oscc-project/iMAP

数据结构

- 逻辑电路表达
  - AIG（And-Inverter Graph）

AND和Inverter是逻辑完备的



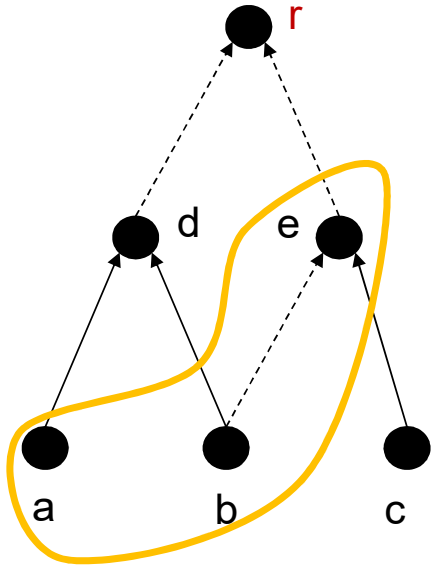


数据结构

- CUT

对于一个节点r的cut, 是一组由其输入锥上的路径组成的节点集合，且从输入到节点r一定会经过该cut

- k-feasible-cut
- k-feasible-cut computation



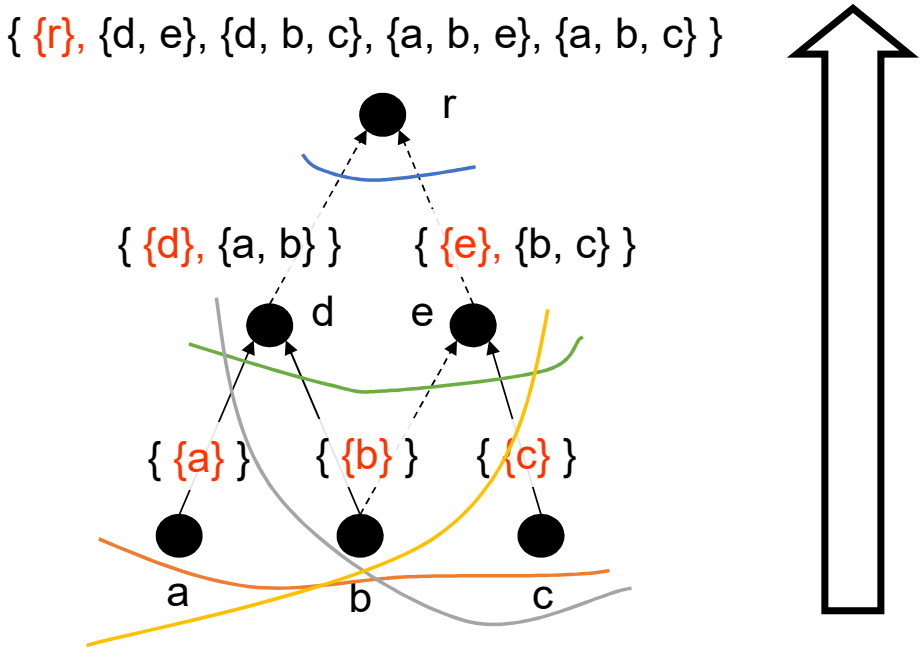
集合{a,b,e}是节点r的一个3-feasible-cut

数据结构

- CUT

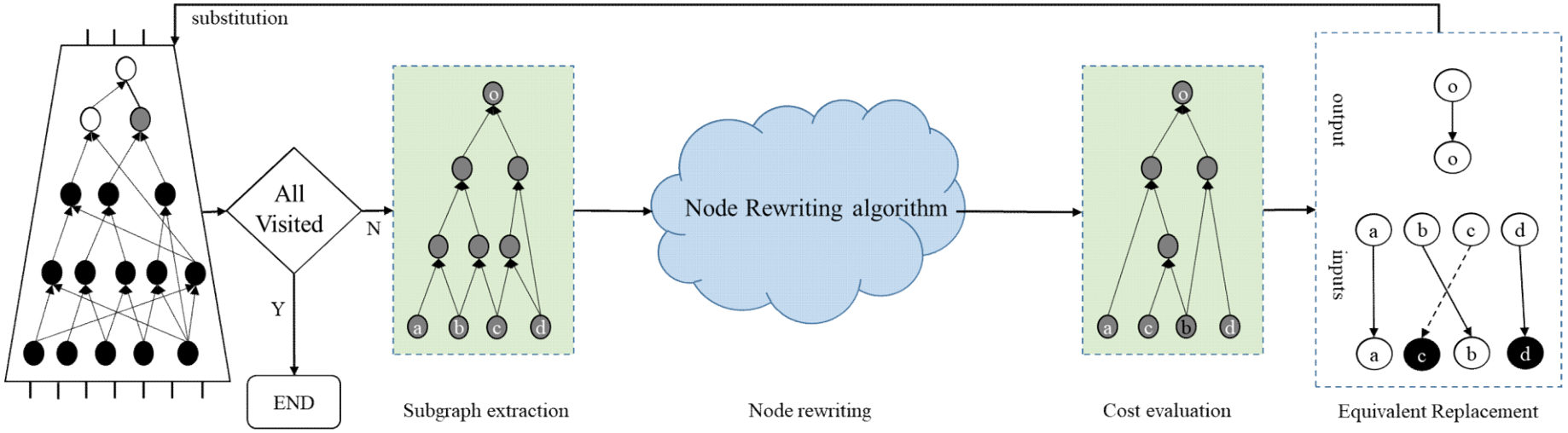
对于一个节点r的cut, 是一组由其输入锥上的路径组成的节点集合，且从输入到节点r一定会经过该cut

- k-feasible-cut
- k-feasible-cut computation



逻辑优化算子

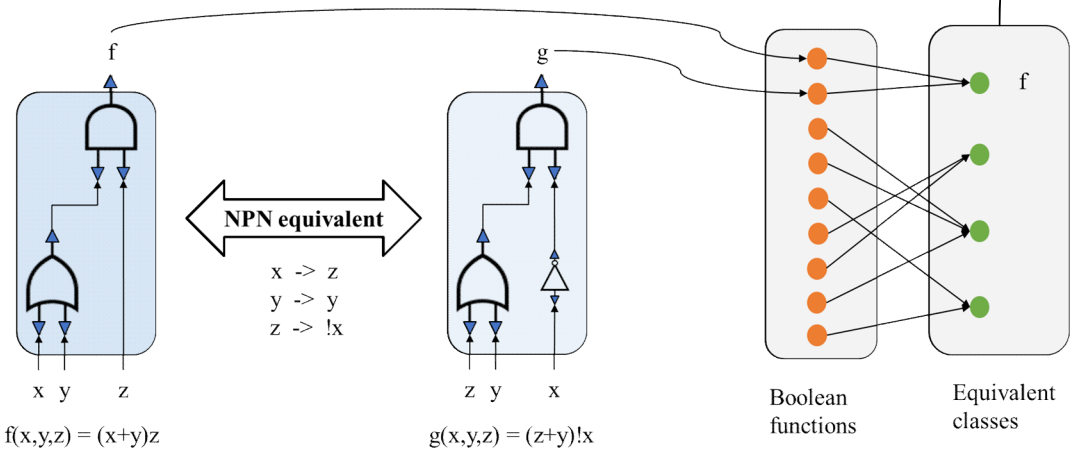
- Framework of rewriting algorithms

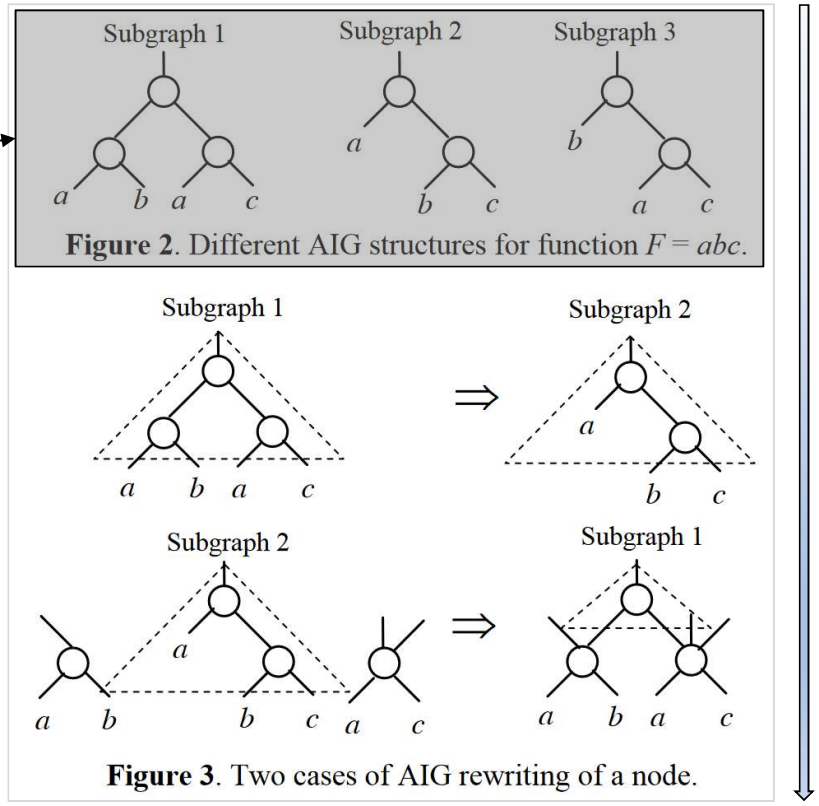


1. Subgraph extraction
2. Node rewriting
3. Cost evaluation
4. Equivalent replacement

逻辑优化算子

- Rewrite-4
  - NPN-library pre-computation
  - k-feasible-cut enumeration
  - Iteratively local replacement
    - cost evaluation
    - Substitution

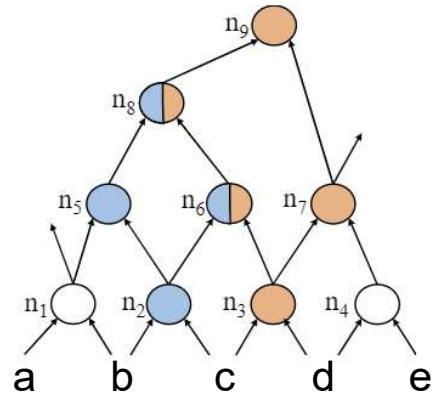


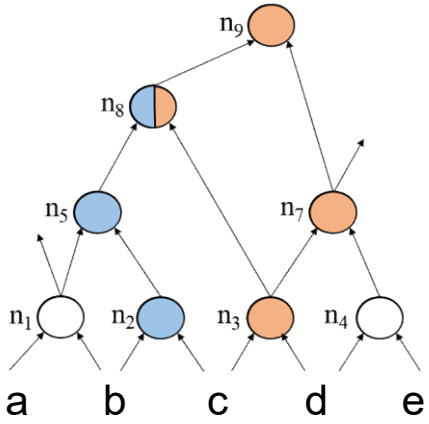


[1] A. Mishchenko, S. Chatterjee and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis", DAC06

逻辑优化算子

- Refactor
  - Reconvergence-cut computation
  - Local replacement
    - SOP minimization
    - Cost evaluation
    - Substitution





$n_1 = ab$
$n_2 = bc$
$n_3 = cd$
$n_4 = de$
$n_5 = n_1n_2 = abc$
$n_6 = n_2n_3 = bcd$
$n_7 = n_3n_4 = cde$
$n_8 = n_5n_6 = abc \ \& \ bcd = abcd$
$n_9 = n_7n_8 = cde \ \& \ abcd = abcde$

逻辑优化算子

- Balance
  - AND-tree computation
    - Tree-balance
    - replacement

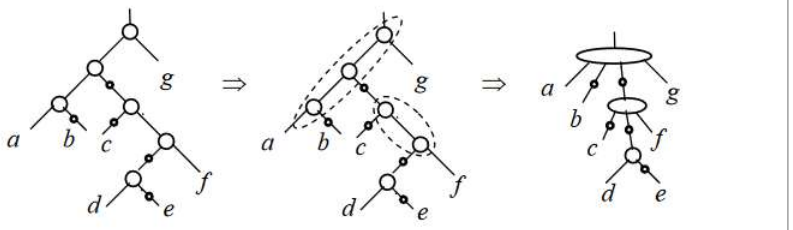


Figure 3.1.1: Illustration of the covering step.

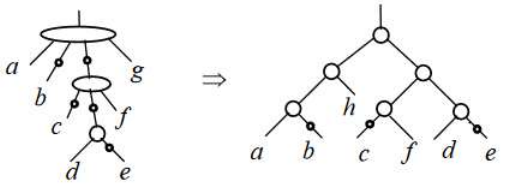


Figure 3.1.2: Illustration of the tree-balancing step.

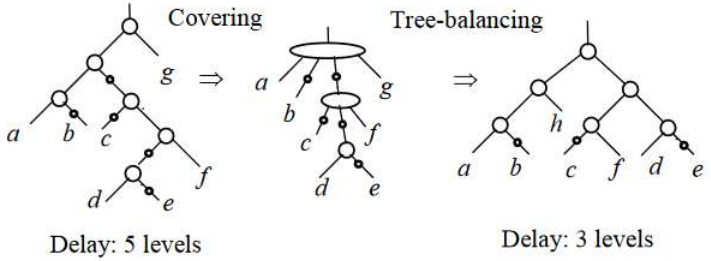
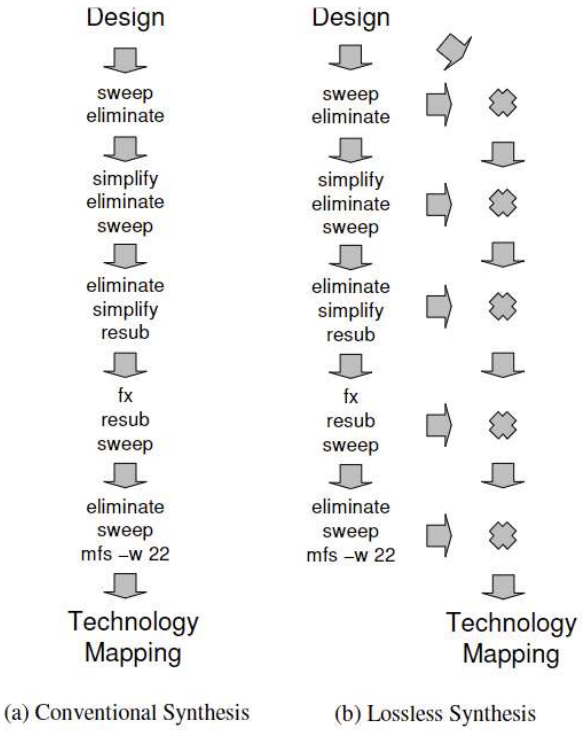


Figure 3.1.3: Illustration of AND-balancing.

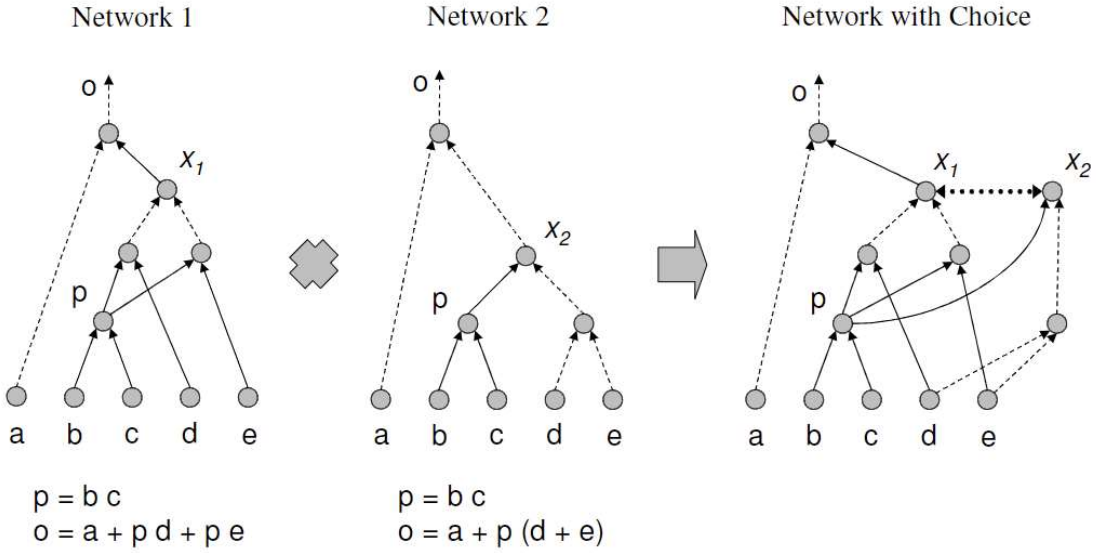
[1] A. Mishchenko, R. Brayton, et al. "Delay Optimization Using SOP Balancing ", ICCAD-2011

工艺映射算子

- choice_generation (remain structural bias)
  - Lossless synthesis



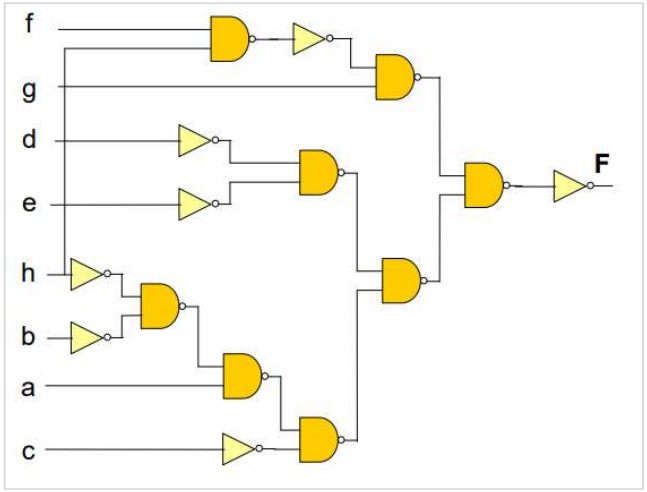
(a) Conventional Synthesis (b) Lossless Synthesis



[1] S. Chatterjee, A. Mishchenko and et al, "Reducing Structural Bias in Technology Mapping ", TCAD05

工艺映射算子

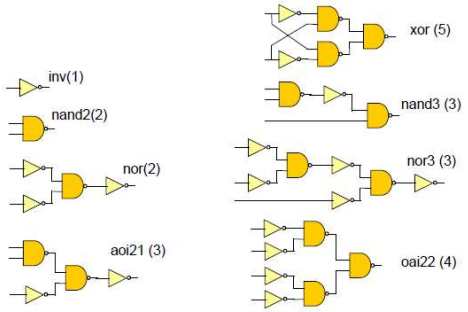
- map_fpga / map_asic



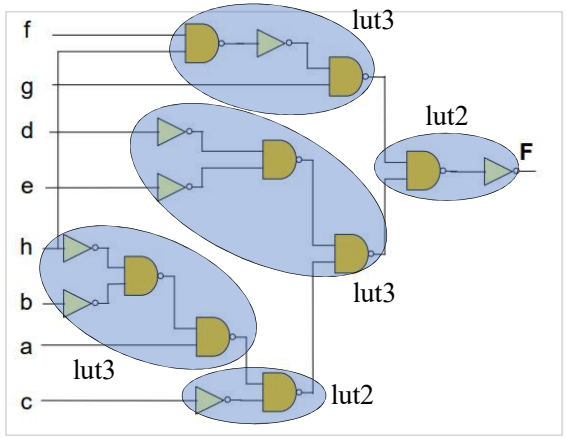
Logic-Level circuit

k-LUT

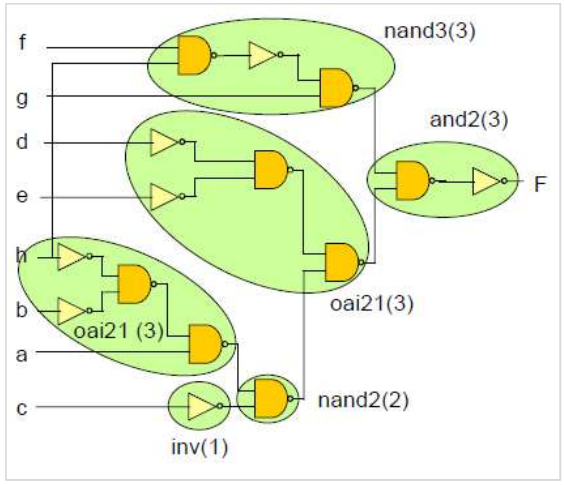
Technology mapping



standard cell library



FPGA



ASIC

iMAP使用说明

- command engine

```
username@pcl-X11DPi-N-T:<project_path>/bin$ ./imap
imap> read_aiger -f ../../../benchmark/EPFL/arithmetic/adder.aig
imap> print_stats -t 0
Stats of AIG: pis=256, pos=129, area=1020, depth=255
imap> rewrite
imap> balance
imap> refactor
imap> lut_opt
imap> map_fpga
imap> print_stats -t 1
Stats of FPGA: pis=256, pos=129, area=215, depth=75
imap> write_fpga -f adder.fpga.v
```

```
./imap -c "read_aiger -f ../../../benchmark/EPFL/arithmetic/adder.aig; print_stats -t 0; rewrite; balance; refactor; lut_opt; map_fpga; print_stats -t 1; write_fpga -f adder.fpga.v"
```

iMAP使用说明

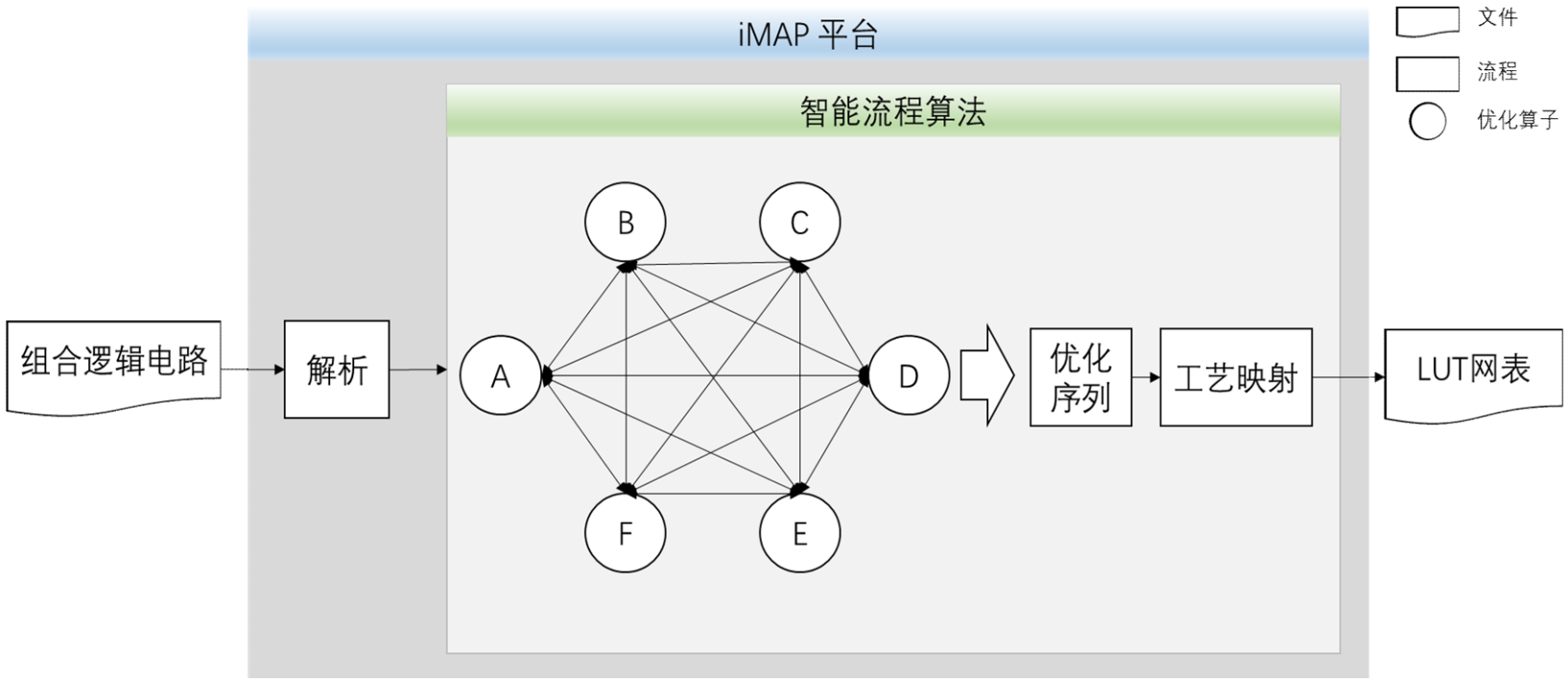
- python engine

```
class EngineIMAP():
    """ IMAP engine for python usages

    IO functions:
    read
    write

    Logic optimization functions:
    rewrite
    refactor
    balance
    lut_opt
    history

    Technology mapping functions:
    map_fpga

    Utils:
    add_sequence
    cleanup
    print_stats
```

```
from imap_engine import EngineIMAP

class Demo(object):
    '''
    A very simple demo.
    '''
    def __init__(self, input_file) -> None:
        self.input_file = input_file
        self.engine = EngineIMAP(input_file, input_file+'.seq')

    def _opt_size(self):
        self.engine.rewrite()
        self.engine.add_sequence('rewrite')
        self.engine.refactor(zero_gain=True)
        self.engine.add_sequence('refactor -z')

    def _opt_depth(self):
        self.engine.balance()
        self.engine.add_sequence('balance')

    def _opt_lut(self):
        self.engine.lut_opt()
        self.engine.add_sequence('lut_opt')

    def run(self):
        self.engine.read()
        opt_round = 5
        while opt_round > 0:
            self._opt_size()
            self._opt_depth()
            self.engine.print_stats()
            opt_round -= 1

        self.engine.map_fpga()
        self.engine.add_sequence('map_fpga')
        self.engine.print_stats(type=1)

        self.engine.write()
```

iMAP for EDA精英挑战赛

- 赛题二：组合逻辑优化与工艺映射的智能流程



iMAP for EDA精英挑战赛

- 自定义特征:
  - 路径：iMAP/ cli / command / print_stats.hpp

AIG feature

```
if( type == 0 ) {
    if( store<iFPGA::aig_network>().empty() ) {
        printf("WARN: there is no any stored AIG file, please refer to the command \"read_aiger\"\n");
        return;
    }
    iFPGA::aig_network aig = store<iFPGA::aig_network>().current()._storage;
    iFPGA::depth_view<iFPGA::aig_network> daig(aig);
    printf("Stats of AIG: pis=%d, pos=%d, area=%d, depth=%d\n", aig.num_pis(), aig.num_pos(), aig.num_gates(), daig.depth());
}
```

FPGA netlist feature

```
else if (type == 1) {
    if( store<iFPGA::klut_network>().empty() ) {
        printf("WARN: there is no any FPGA mapping result, please refer to the command \"map_fpga\"\n");
        return;
    }
    iFPGA::klut_network klut = store<iFPGA::klut_network>().current()._storage;
    iFPGA::depth_view<iFPGA::klut_network> dklut(klut);
    printf("Stats of FPGA: pis=%d, pos=%d, area=%d, depth=%d\n", klut.num_pis(), klut.num_pos(), klut.num_gates(), dklut.depth());
}
```

01 研究内容

02 研究进展

03 未来计划

未来计划

- 在runtime以及QoR上赶超berkeley-abc
- 完善功能以及接口，支持开源芯片流片
- 支持与iEDA中PR工具的交互
- 支持physical-aware的逻辑综合
- 支持更细粒度 AI+逻辑综合 的研究

iEDA Tutorial 第四期议程

- Part1 iEDA-iMAP工具介绍 20min （倪利伟）

- Part2 AiMAP工艺映射算法 15min （刘俊锋）

- Part3 并行化逻辑重写算法 15min （杨宗霖）

- Part4 iEDA-iATPG工具介绍 20min （林晓泽）

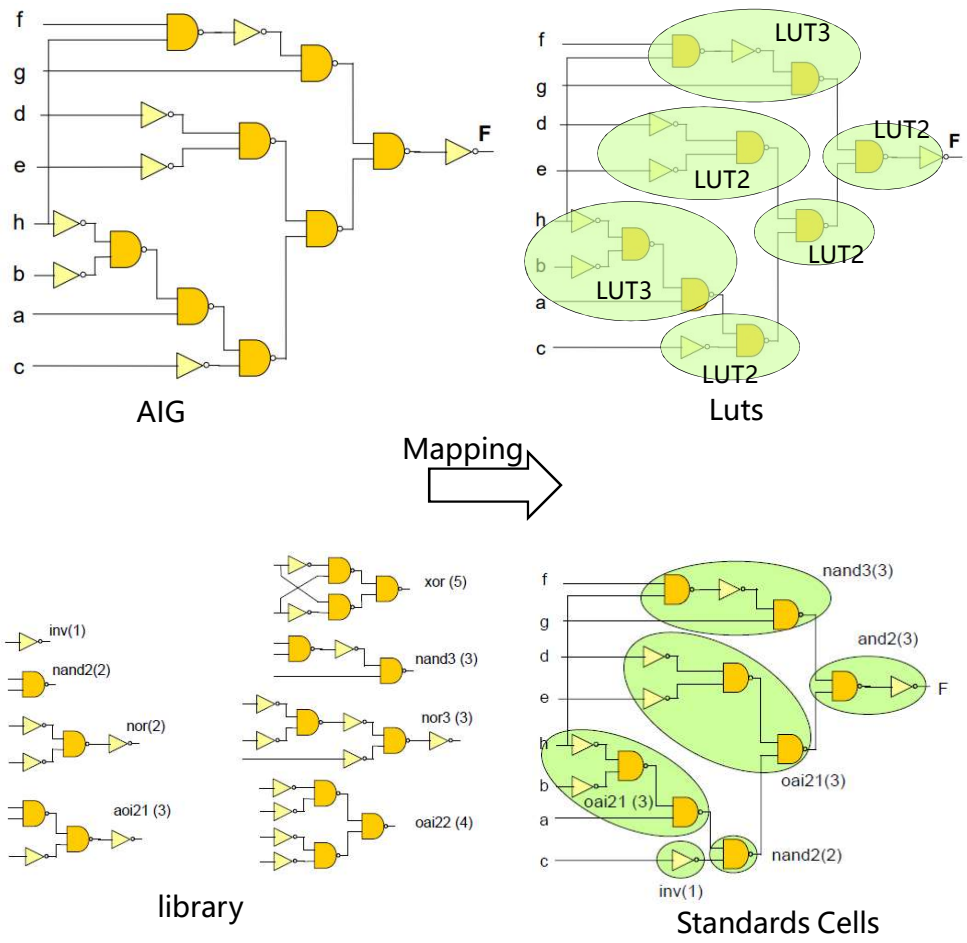
Technology Mapping Problem

Combinational Technology Mapping: Given a set of gates L, called the library, and a Boolean network G, let $\mathcal{M}$ be the set of Boolean networks constructed using gates from L that are functionally equivalent to G. These are called mapped networks.
The **goal** of mapping is to find a mapped network $\mathbf{M} \in \mathcal{M}$ that **minimizes some objective** such as **area subject to certain constraints such as timing**

- ASIC library: Standard Cells, *e.g.,* AND2, INVERTER
- FPGA library: Look-up Tables(LUT)
  look-up table with k inputs, called a k-lut is a configurable gate that can implement any Boolean function of k variables

Time Complexity:

- Minimize delay (depth of LUTs, load-independent delay): $O(n^k)$ [1]
- Minimize area (number of LUTs, area of cells): ***NP-Complete*** [2,3]



For ASIC: Standard Cell

A **standard cell library** is a collection of predefined and pre-characterized logic cells or building blocks that are used to implement digital designs

It composes a set of standard cells (predefined and reusable logic gates: e.g., NAND2, INV, OR2, and AOI21) and their electrical and physical properties (e.g., functionality, pins, timing information, and power consumption). For example ASAP7 7nm PDK contains 154 standard cells

Supergate is the composition of the standard cells into a single-out gate and contains many gates with diverse functionality, which is utilized to mitigate the structural bias problem in technology mapping

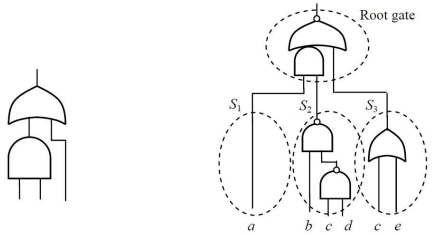


Figure 4: A supergate. Figure 5: Supergate generation.

```
cell ( AND2CLKHD1X ) {
    area                : 7.564 ;
    cell_leakage_power  : 0.344886 ;
    cell_footprint      : and2clk ;
    pin ( A ) {
    |   direction        : input ;
    |   capacitance      : 0.00226414 ;
    }
    pin ( B ) {
    |   direction        : input ;
    |   capacitance      : 0.00221157 ;
    }
    pin ( Z ) {
    |   direction        : output ;
    |   capacitance      : 0 ;
    |   max_capacitance  : 0.423245 ;
    |   function         : "(A B)" ;
    |   timing ( ) {
    |       related_pin  : "A" ;
    |       timing_sense : positive_unate ;
    |       cell_rise ( delay_template_6x6 ) {
    |       |   index_1 ( "0.0001, 0.02, 0.1, 0.2
    |       |   index_2 ( "0.033344, 0.093963, 0.
    |       |   values ( \
    |       |       "0.067736, 0.074117, 0.09274,
    |       |       "0.125151, 0.133677, 0.152551
    |       |       "0.3343, 0.337365, 0.355432,
    |       |       "0.5683, 0.578238, 0.598328,
    |       |       "0.807801, 0.819009, 0.867235
```

For ASIC: Standard Cell

Non-Linear behavior of Cell Delays:

Input transition (slew): refers to the change in voltage at the input of a logic gate or circuit from one logic state to another. typically from low (0) to high (1) or vice versa.

Output Capacitance (load): refers to the capacitance that is present at the output of a logic gate or circuit. It represents the ability of the output to store electrical charge.

If the mapped circuit is obtained, one can compute the delay using (i) *non-linear delay model, NLDM,* or (ii) *wire load model, WLM*

```
pin (OUT) {
  max_transition : 1.0;
  timing() {
    related_pin : "INP1";
    timing_sense : negative_unate;
    cell_rise(delay_template_3x3) {
      index_1 ("0.1, 0.3, 0.7"); /* Input transition */
      index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
      values (  /*   0.16      0.35      1.43 */ \
        /* 0.1 */   "0.0513,  0.1537,  0.5280", \
        /* 0.3 */   "0.1018,  0.2327,  0.6476", \
        /* 0.7 */   "0.1334,  0.2973,  0.7252");
    }
    cell_fall(delay_template_3x3) {
      index_1 ("0.1, 0.3, 0.7"); /* Input transition */
      index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
      values (  /*   0.16      0.35      1.43 */ \
        /* 0.1 */   "0.0617,  0.1537,   0.5280", \
        /* 0.3 */   "0.0918,  0.2027,   0.5676", \
        /* 0.7 */   "0.1034,  0.2273,   0.6452");
    }
```

Before mapping, both the input slew and output load are unknown!

For ASIC: Mapping Flow (Dynamic Programming)

The heuristic mapper in ABC [4]

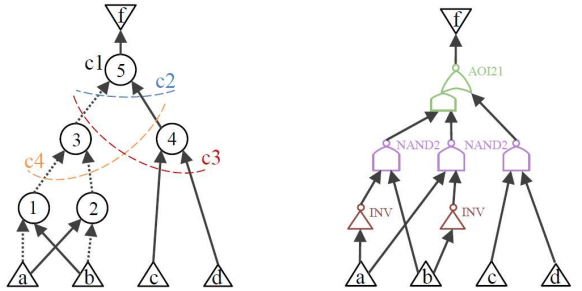
Input: And-Inverter Graph

1. Construct supergates and Compute the load-independent delay model for each supergate
2. Compute K-feasible cuts for each node
3. For each cut, the truth table is computed to check whether it can be implemented by the supergates
4. Compute best **arrival time (estimated delay)** at each node
   - In topological order (from PI to PO)
   - Compute the depth of all cuts and choose the best one
5. Perform *area recovery*
   - Using area flow
   - Using exact local area
6. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist

Existing studies focus on:

(1) Cut Ranking, filtering, merging
(2) Designing heuristic area estimation
(3) Supergate generation
(4) Delay estimation of the cells (or supergates)
   - load-independent delay estimation
   - load-dependent delay estimation



(a) AIG with function f. (b) Mapped circuit of f.

Fig. 1. An example of an AIG and its mapped circuit with the Boolean function $f = a \oplus b \wedge c \wedge d$. The dashed lines on the edges indicate the wires with inverters.

Supergate Delay Estimation in ABC

- Load-Independent Supergate Delay Estimation

$$\mathrm{d}(g) = \mathtt{LD} \times \mathtt{Ga} + \mathtt{P}$$

  - `P` is the load-independent parasitic delay w.r.t. the gate,
  - `LD` is the induced delay per unit load,
  - `Ga` is a pre-defined gain factor for load.

```
pin (OUT) {
  max_transition : 1.0;
  timing() {
    related_pin : "INP1";
    timing_sense : negative_unate;
    cell_rise(delay_template_3x3) {
      index_1 ("0.1, 0.3, 0.7"); /* Input transition */
      index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
      values (  /*  0.16      0.35     1.43 */ \
        /* 0.1 */   "0.0513,  0.1537,  0.5280", \
        /* 0.3 */   "0.1018,  0.2327,  0.6476", \
        /* 0.7 */   "0.1334,  0.2973,  0.7252");
    }
    cell_fall(delay_template_3x3) {
      index_1 ("0.1, 0.3, 0.7"); /* Input transition */
      index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
      values (  /*  0.16      0.35     1.43 */ \
        /* 0.1 */   "0.0617,  0.1537,  0.5280", \
        /* 0.3 */   "0.0918,  0.2027,  0.5676", \
        /* 0.7 */   "0.1034,  0.2273,  0.6452");
    }
```

- Large Gap Between Estimated and Actual Circuit Delay

TABLE I
MOTIVATION EXAMPLE: THE SIGNIFICANT DISPARITY BETWEEN ESTIMATED CIRCUIT DELAY AND ACTUAL CIRCUIT DELAY.

Circuits	Estimated Results		Actual Results		Δ Delay
	Area(μm^2)	LI-Delay(ps)	Area(μm^2)	Delay(ps)	
adder	898.31	2,613.78	898.13	3,770.65	44%
bar	2,681.62	152.96	2,680.39	1,114.9	629%
log2	26,556.98	3,891.66	26,561.26	6,797.77	75%
cavlc	463.27	185.07	463.29	93.2	50%
int2float	158.61	174.27	158.63	91.7	47%
ctrl	106.92	98.53	106.84	89.9	9%

LI-Delay refers to the load-independent circuit delay estimation in ABC [1].
The actual circuit delay is computed by the non-linear delay model [1].

Supergate Delay Estimation in ABC

- Load-Independent Supergate Delay Estimation

$$\mathrm{d}(g) = \mathtt{LD} \times \mathtt{Ga} + \mathtt{P}$$

  - `P` is the load-independent parasitic delay w.r.t. the gate,
  - `LD` is the induced delay per unit load,
  - `Ga` is a pre-defined gain factor for load.

```
pin (OUT) {
  max_transition : 1.0;
  timing() {
    related_pin : "INP1";
    timing_sense : negative_unate;
    cell_rise(delay_template_3x3) {
      index_1 ("0.1, 0.3, 0.7"); /* Input transition */
      index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
      values (  /*  0.16      0.35     1.43 */ \
        /* 0.1 */   "0.0513,  0.1537,  0.5280", \
        /* 0.3 */   "0.1018,  0.2327,  0.6476", \
        /* 0.7 */   "0.1334,  0.2973,  0.7252");
      }
    cell_fall(delay_template_3x3) {
      index_1 ("0.1, 0.3, 0.7"); /* Input transition */
      index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
      values (  /*  0.16      0.35     1.43 */ \
        /* 0.1 */   "0.0617,  0.1537,  0.5280", \
        /* 0.3 */   "0.0918,  0.2027,  0.5676", \
        /* 0.7 */   "0.1034,  0.2273,  0.6452");
    }
```

- Large Gap Between Estimated and Actual Circuit Delay

TABLE I
MOTIVATION EXAMPLE: THE SIGNIFICANT DISPARITY BETWEEN ESTIMATED CIRCUIT DELAY AND ACTUAL CIRCUIT DELAY.

Circuits	Estimated Results		Actual Results		Δ Delay
	Area(μm^2)	LI-Delay(ps)	Area(μm^2)	Delay(ps)	
adder	898.31	2,613.78	898.13	3,770.65	44%
bar	2,681.62	152.96	2,680.39	1,114.9	629%
log2	26,556.98	3,891.66	26,561.26	6,797.77	75%
cavlc	463.27	185.07	463.29	93.2	50%

Before mapping, both the input slew and output load are unknown!

LI-Delay refers to the load-independent circuit delay estimation in ABC [1].
The actual circuit delay is computed by the non-linear delay model [1].

AiMap: Learning to Improve Technology Mapping for ASICs via Delay Prediction

- Problem

  Given a Boolean network, its *k-feasible* cuts for each node, and the supergates matched for each cut, our goal is to make an accurate estimation delay of supergates, *w.r.t* the cut to guide the search of the mapper to achieve the better QoR of the mapped circuit.

- Challenges

  a) How to generate supergate delays *under the better circuit delay?*

  b) How to design learning model to be aware of (i) varying contributions of different pins to supergate delay estimation (ii) hierarchy of node-cut-supergate tuples

  c) How to prune the number of node-cut-supergate tuples ?

AiMap: Learning to Improve Technology Mapping for ASICs via Delay Prediction

- Problem

  Given a Boolean network, its *k-feasible* cuts for each node, and the supergates matched for each cut, our goal is to make an accurate estimation delay of supergates, *w.r.t* the cut to guide the search of the mapper to achieve the better QoR of the mapped circuit.

- Challenges

  a) How to generate supergate delays *under the better circuit delay?*

  b) How to design learning model to be aware of (i) varying contributions of different pins to supergate delay estimation (ii) hierarchy of node-cut-supergate tuples

  c) How to prune the number of node-cut-supergate tuples ?

AiMap: Learning Label Generation

Three parameterizable strategies from the perspectives of ***(i) load-independent, (ii)load-dependent*** supergate delay estimation, and ***(iii) cut sample.***

- Strategy 1: Load-Independent Supergate Delay Estimation

$$\mathrm{d}_s(g) = \mathtt{LD}_s \times \mathtt{Ga}_s + \mathtt{P}_s$$
$$\overline{\mathrm{d}}(g) = \alpha \cdot \mathrm{d}(g) + (1-\alpha) \cdot \mathrm{d}_s(g)$$

Ps	is the load-independent parasitic delay from the view of slew
LDs	is the induced delay per unit slew,
Gas	is a pre-defined gain factor for slew.
d(g)	is the Load-independent model for the gate delay estimation in ABC

AiMap: Learning Label Generation

- Strategy 2: Load-dependent Supergate Delay Estimation

$$\overline{\mathrm{d}}(g,l) = \overline{\mathrm{d}}(g) \cdot (\beta \cdot \gamma \cdot \log \mathtt{RF} + (1-\beta) \cdot \tau \cdot \log \mathtt{LF})$$

`RF` an `LF` are the fanout of the root nodes and leaf nodes of the cut, respectively.

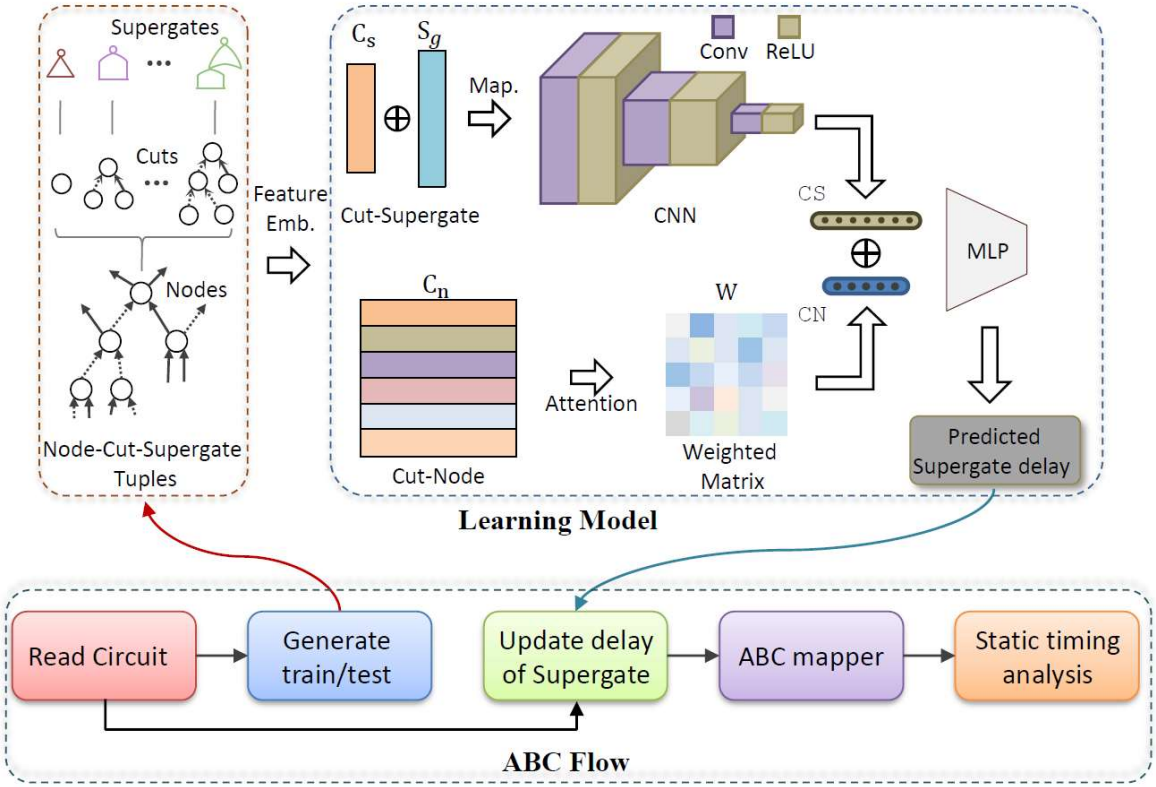
Update arrival time: $\mathrm{Arr}(v) = \max_{\forall g \in \mathrm{CutLeaves}(v)} \{\mathrm{Arr}(g) + \overline{\mathrm{d}}(g,l)\}$

- Strategy 3: Cut Sample

In practice, we indeed use five ranking criteria (with parameter t) to sort the candidate cuts and sample a relatively smaller number of cuts (with parameter r)

Finally, performing a grid search for these parameters to obtain the supergate delay *w.r.t.* the cut under the best circuit delay

AiMap: Framework Overview



AiMap: Feature Embedder

- Node Embedding

  Collect 10 features to capture the structural information from the node itself and from its two children,

TABLE II
SUMMARIZATION OF NODE FEATURES

Node Features	Child 1 Features	Child 2 Features
level(u)	level(u_1)	level(u_2)
fanout(u)	fanout(u_1)	fanout(u_2)
inverter(u)	inverter(u_1)	inverter(u_2)
re-level(u)	–	–

- Cut Embedding

  Collect 19 features for each cut from the ***cut-node*** and ***cut-structure*** aspects.

  cut-node: it includes 6 nodes as the features
  cut structure: it contains 13 features, i.e., (i) the root fanouts, (ii) the cut leaf number and the cut volume for the cut itself and its two parents, (iii) the max, min, and gap of the cut levels and fanouts,

- Supergate Embedding

  Collect 60 features, in brief ***(i) the basic descriptions,*** e.g., the area, leakage power, and the number of inputs/outputs, ***(ii) features related to delay on each pin,*** e.g., the estimation of load-independent delay for different pre-defined gain factors, and ***(iii) the overall features of the supergate,*** e.g., max and sum delay on all pins.

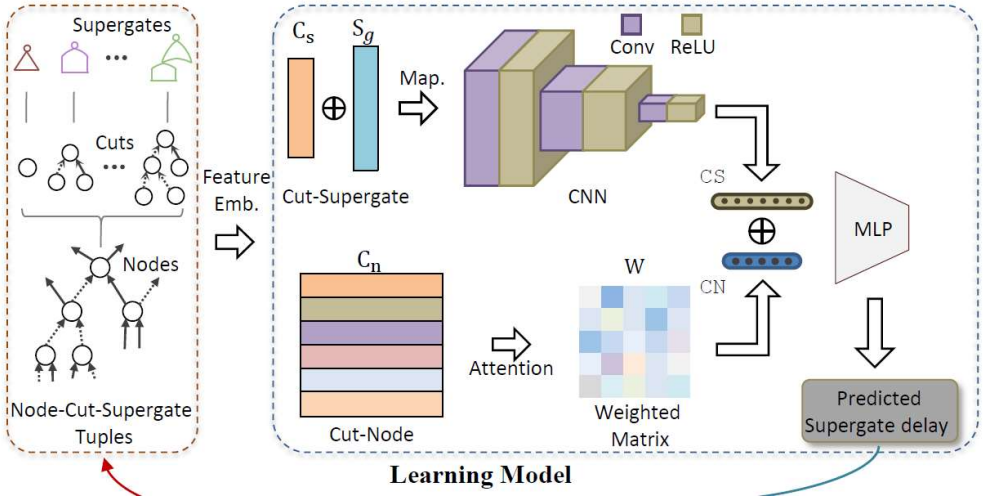
AiMap: Learning Model

- Cut-Node Attention

  To be aware of the *varying contributions* of different pins to supergate delay estimation

$$\mathbf{K} = \tanh\left(\text{mean}(\mathbf{C}_n)\mathbf{W}\right)$$

$$\text{CN} = \sigma\left(\mathbf{K}\mathbf{C}_n^{\top}\mathbf{C}_n\right)$$



- Cut-Supergate Fusion

  To *jointly learn* the physical features related to delay in the supergate and the structural features of the cut based on a CNN mode

- Training process

  Guided by minimizing MSE Loss based on our generated supergate delay labels and predicted the supergate delay

AiMap: Results

QoR of Technology Mapping

TABLE III

RESULTS COMPARISONS OF ABC, SLAP, AIMAP. WE HIGHLIGHT OUR METHOD IF IT BEATS THE COUNTERPARTS *w.r.t.* THE AREA AND DELAY. THE RESULTS OF SLAP DERIVE FROM THEIR ORIGINAL ARTICLE, WHEREAS "–" REFERS TO THE EXPERIMENT RESULTS THAT WE WERE UNABLE TO REPLICATE. COMPARED TO ABC, WE IMPROVE 10 OUT OF 15 CASES.

Circuits	ABC		SLAP		AiMap		AiMap/ABC		AiMap/SLAP	
	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area	Delay	Area	Delay
adder	898.13	3,770.65	1,031.33	3,268.67	1,060.96	3,486.11	1.18	0.92	1.03	1.07
bar	2,680.39	1,114.90	3,083.23	923.82	**2,059.40**	1,058.98	0.77	0.95	0.67	1.15
log2	26,561.26	6,797.77	–	–	**23,330.10**	6,855.81	0.88	1.01	–	–
multiplier	25,458.31	4,649.10	–	–	**20,106.40**	**4,512.38**	0.79	0.97	–	–
sin	5,207.04	3,955.57	5,087.60	3,584.79	**4,503.00**	3,599.18	0.86	0.91	0.89	1.00
sqrt	20,252.20	180,518.05	–	–	**19,918.61**	185,280.97	0.98	1.03	–	–
C6288	2,991.82	1,248.55	3,023.54	1,236.59	**2,479.53**	1,385.40	0.83	1.11	0.82	1.12
C7552	1,978.45	797.54	2,002.01	800.09	**1,758.93**	913.78	0.89	1.15	0.88	1.14
mul32-booth	9,889.44	3,229.54	–	–	**8,041.16**	3,410.47	0.81	1.06	–	–
mul64-booth	39,736.45	6,715.12	–	–	**31,087.36**	7,058.33	0.78	1.05	–	–
64b_mult	52,318.64	7,922.55	–	–	**37,275.11**	9,343.50	0.71	1.18	–	–
aes	18,190.01	656.60	16,489.63	594.64	**15,792.12**	625.55	0.87	0.95	0.96	1.05
cavlc	471.23	294.12	–	–	480.32	**259.77**	1.02	0.88	–	–
int2float	159.56	160.04	–	–	162.13	163.32	1.02	1.02	–	–
ctrl	107.54	134.04	–	–	**107.31**	**101.96**	1.00	0.76	–	–
Geomean	4,290.43	2,098.17	–	–	3,797.73	2,079.21	0.885	0.991	0.865	1.087

AiMap: Results

QoR of Delay-Oriented Technology Mapping

TABLE V
DELAY-ORIENTED MAPPING RESULTS OF ABC AND AIMAP.

Circuits	ABC		AiMap		AiMap/ABC	
	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area	Delay
adder	2,369.43	2,618.21	1,398.51	2,225.75	0.59	0.85
bar	4,198.34	1,936.90	2,779.53	1,428.83	0.66	0.74
sqrt	48,105.13	303,498.88	37,596.11	301,290.41	0.78	0.99
sin	9,784.00	6,456.01	8,691.31	6,996.77	0.89	1.08
C6288	2,991.82	1,248.55	2,479.53	1,385.40	0.83	1.11
C7552	3,748.58	1,605.98	3,733.41	1,163.79	1.00	0.72
aes	29,727.57	939.01	25,064.54	791.39	0.84	0.84
router	329.16	617.27	452.56	609.04	1.37	0.99
Geomean	5,174.28	3,219.70	4,371.54	2,914.76	0.845	0.905

AiMap achieves a remarkable delay improvement of 15% compared to ABC. Surprisingly, it even outperforms our parameterized strategy generated labels by 7% (the labels result being ***2,394.06*** ps). Proved by CEC (combinational equivalence checking)

AiMap: Results

QoR of Training Strategies

TABLE IV

TRAINING RESULTS *w.r.t.* THREE TRAINING STRATEGIES. WE ONLY RECORDED THE IMPROVEMENT IN DELAY, DISREGARDING CHANGES IN AREA.

Circuits	ABC		Strategy 1			Strategy 2			Strategy3		
	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	ΔDelay	Area(μm^2)	Delay(ps)	ΔDelay	Area(μm^2)	Delay(ps)	ΔDelay
adder	2,371.12	2,618.21	2,325.37	2,394.06	8.6%	2,325.37	2,394.06	8.6%	2,325.37	2,394.06	8.6%
bar	4,198.34	1,936.90	2,389.64	934.51	51.8%	3,169.06	876.30	54.8%	3,275.38	872.59	54.9%
max	4,092.32	4,640.24	3,448.38	4,372.64	5.8%	2,724.51	2,948.38	36.5%	2,731.99	2,939.40	36.7%
sin	9,785.40	6,456.01	10,840.20	4,498.79	30.3%	8,465.33	4,438.97	31.2%	8,467.88	4,434.73	31.3%
i2c	1,167.24	244.35	1,209.60	222.21	9.1%	1,241.32	208.55	14.7%	1,241.32	208.55	14.7%
priority	1,736.08	2,856.16	1,697.73	2,669.83	6.5%	1,697.73	2,669.83	6.5%	1,697.73	2,669.83	6.5%
router	452.56	609.04	441.56	497.60	18.3%	441.56	497.60	18.3%	464.73	488.26	19.8%
Geomean	2,323.49	1,813.76	2,113.47	1,442.44	20.5%	2,061.37	1,336.26	26.3%	2,087.21	1,331.07	26.6%

our strategies can be extended to enhance the mapped network for both area and delay. The three strategies collectively contribute ***16.6%, 18.3%, and 18.8%*** to the area and delay improvements, on average.

iEDA Tutorial 第四期议程

- Part1 iEDA-iMAP工具介绍 20min （倪利伟）

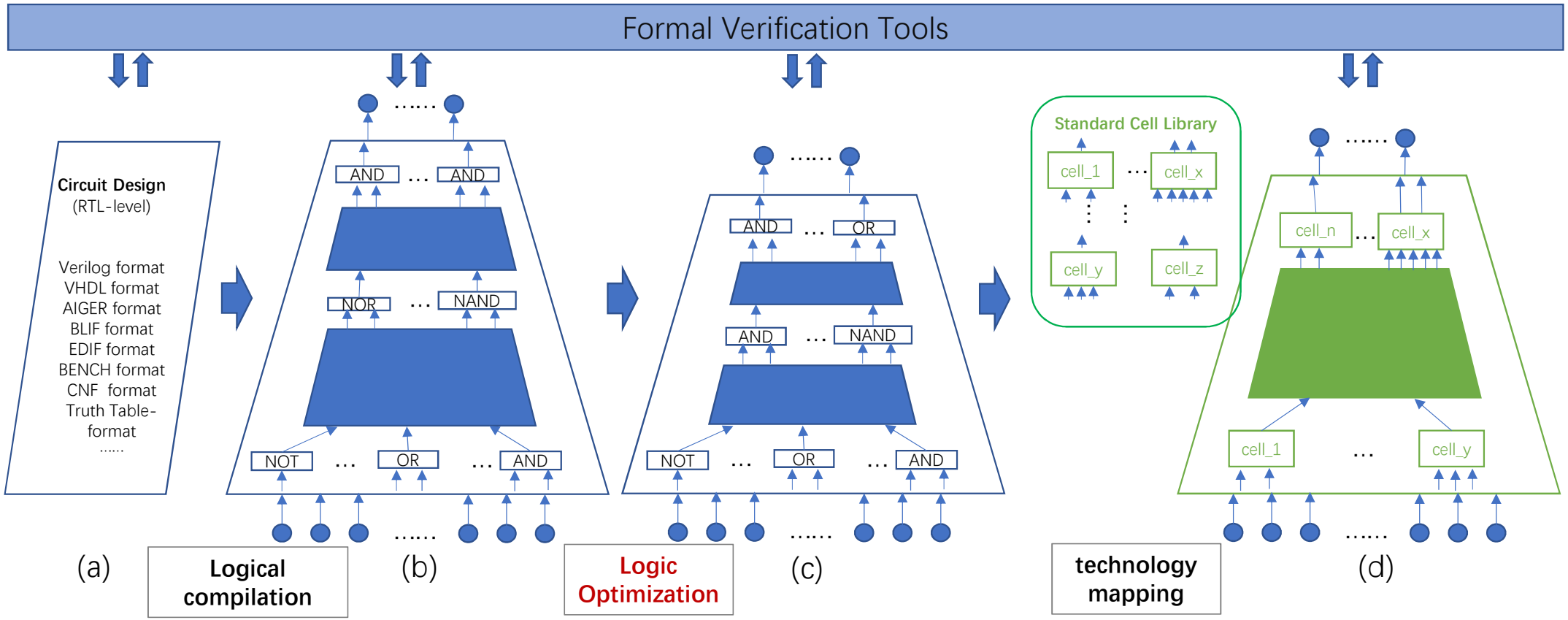
- Part2 AiMAP工艺映射算法 15min （刘俊锋）

- Part3 并行化逻辑重写算法 15min （杨宗霖）

- Part4 iEDA-iATPG工具介绍 20min （林晓泽）

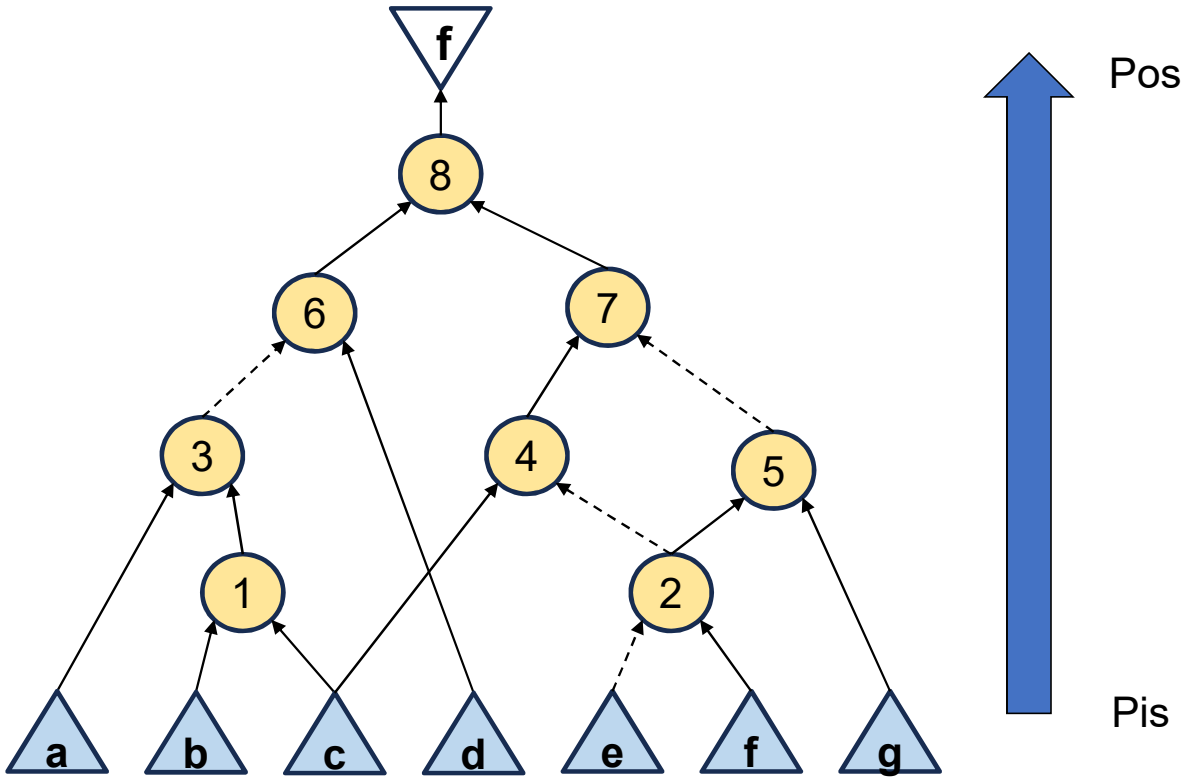
并行refactor算法研究

- 对布尔逻辑网络进行优化，获得更低area和depth。



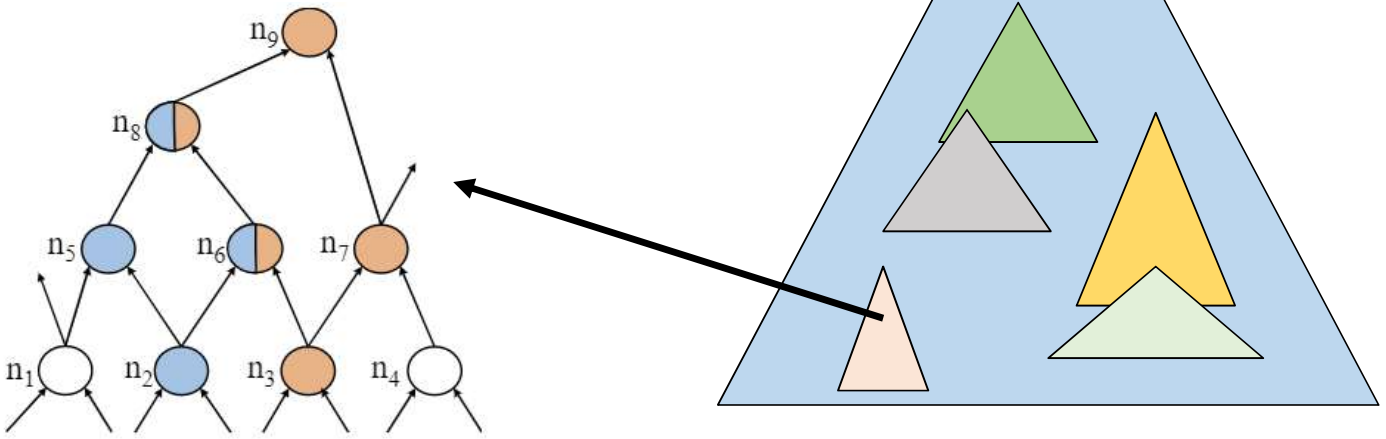
并行refactor算法研究

- Refactor
  - Topological order(Sequential)
    - Reconvergence-cut computation
    - SOP refactoring(Truth table)
    - Evaluation(MFFC)
    - Local replacement



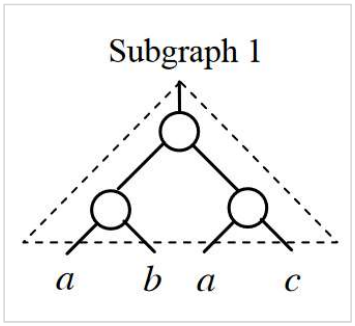
并行refactor算法研究

- Refactor
  - Topological order(Sequential)
    - Reconvergence-cut computation
    - SOP refactoring(Truth table)
    - Evaluation(MFFC)
    - Local replacement



并行refactor算法研究

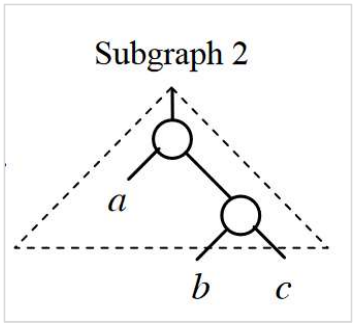
- Refactor
  - Topological order(Sequential)
    - Reconvergence-cut computation
    - SOP refactoring(Truth table)
    - Evaluation(MFFC)
    - Local replacement



F = a ∧ b ∧ b ∧ c

a	b	c	a ∧ b ∧ c
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

SOP形式化简

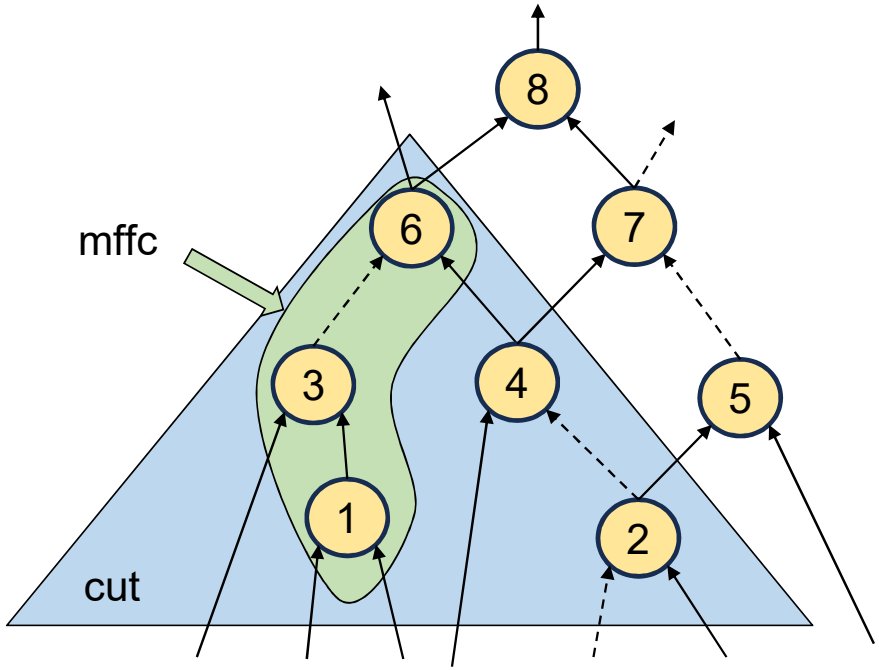


F = a ∧ b ∧ c

并行refactor算法研究

- Refactor
  - Topological order(Sequential)
    - Reconvergence-cut computation
    - SOP refactoring(Truth table)
    - Evaluation(MFFC)
    - Local replacement

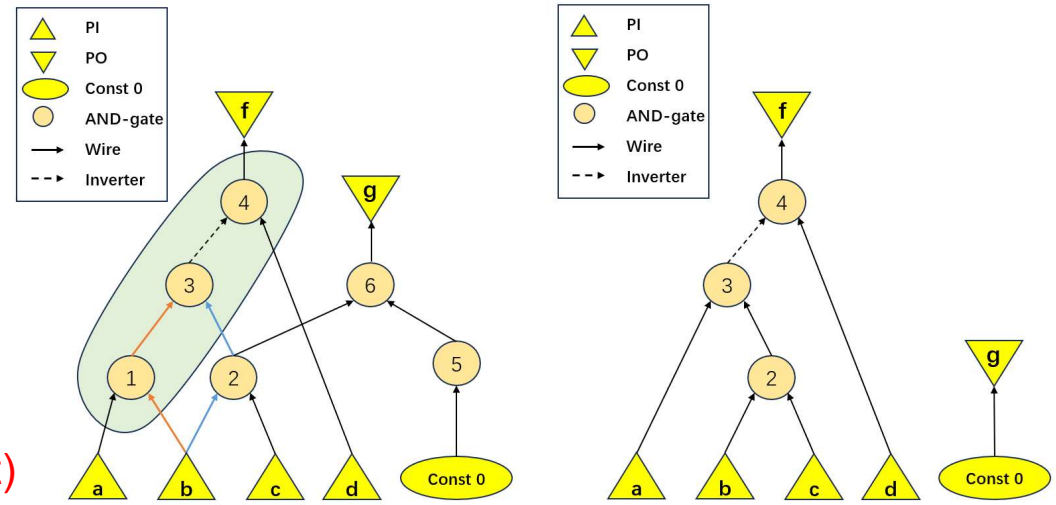
Cost = mffc.size()



并行refactor算法研究

- Refactor
  - Topological order(Sequential)
    - Reconvergence-cut computation
    - SOP refactoring(Truth table)
    - Evaluation(MFFC)
    - Local replacement

Compare(old_sub_g.cost, new_sub_g.cost)

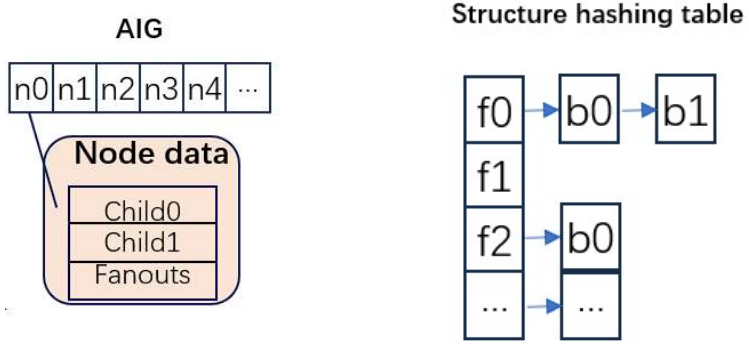


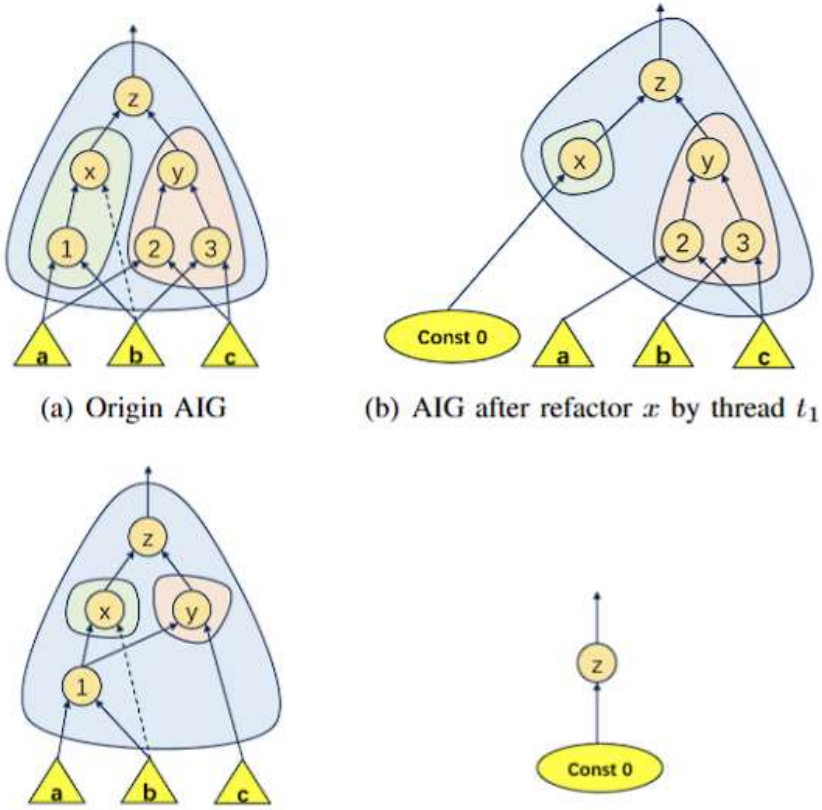
$$f = \neg(a \wedge b \wedge c) \wedge d$$

$$n3 = a \wedge b \wedge b \wedge c \Rightarrow n3 = a \wedge b \wedge c$$

并行refactor算法研究

- 拓扑序顺序执行时间开销大
  - 并行化冲突：
    - 逻辑冲突
    - 数据冲突





(a) Origin AIG (b) AIG after refactor x by thread t_1

(c) AIG after refactor y by thread t_2 (d) AIG after refactor z by thread t_3

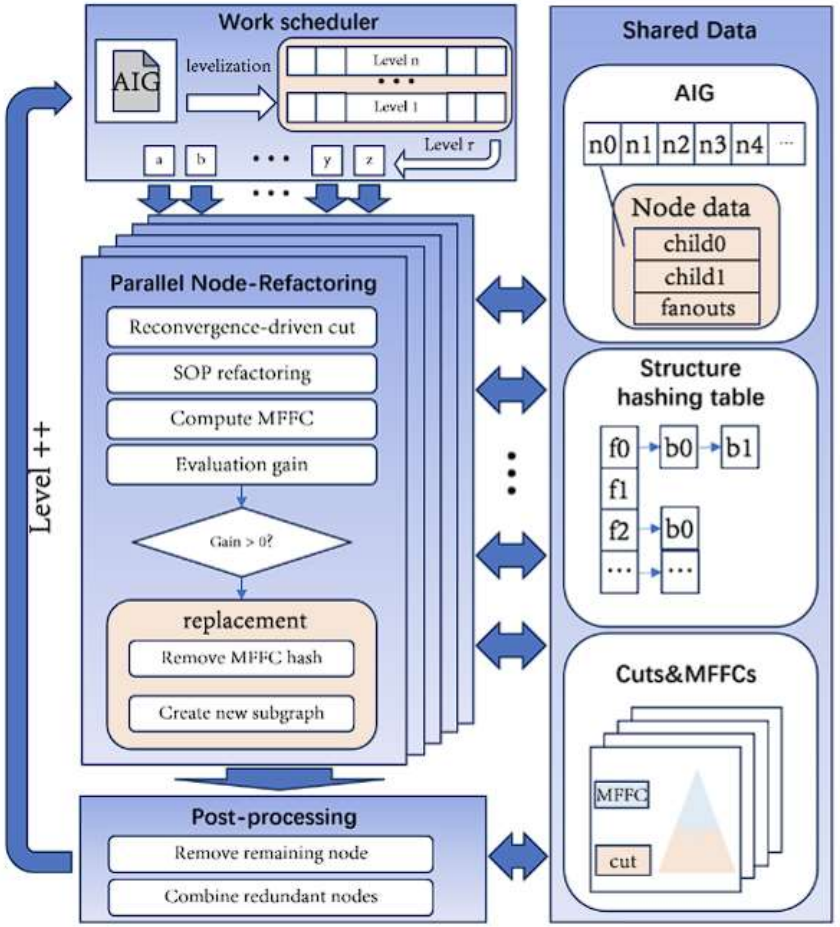
01 研究内容

02 研究进展

03 未来计划

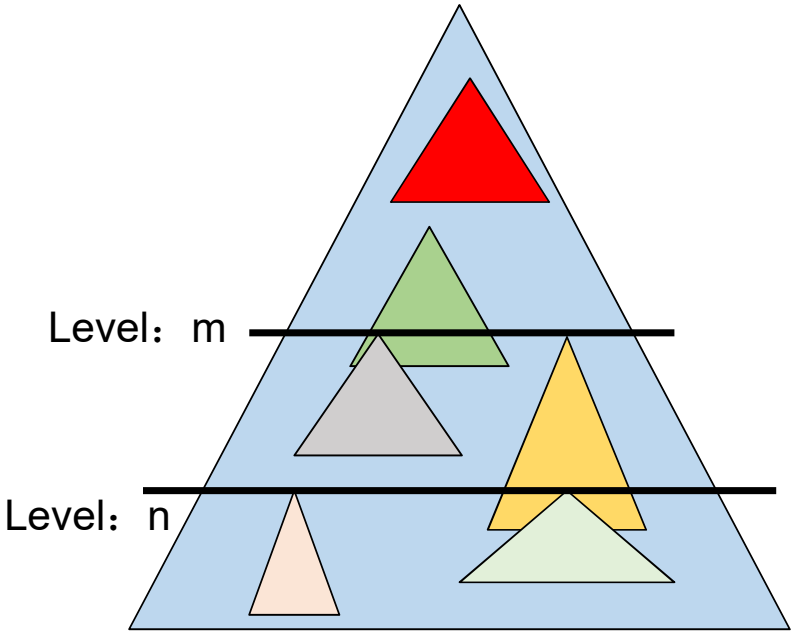
并行refactor算法研究

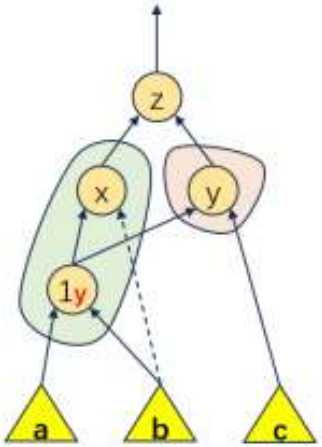
- 并行化
  - 逻辑冲突
    - 重复删除节点
    - 复用点被删除
  - 数据冲突
    - Aig nodes array
    - Structure hashing table
    - Cuts & MFFCs



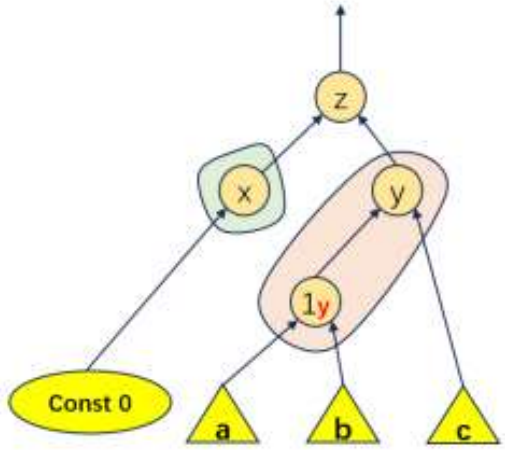
并行refactor算法研究

- 逻辑冲突
  - 重复删除节点
  - 复用点被删除





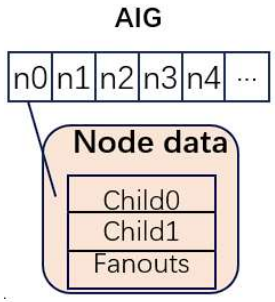
(a) AIG before deleting MFFC nodes of root node x

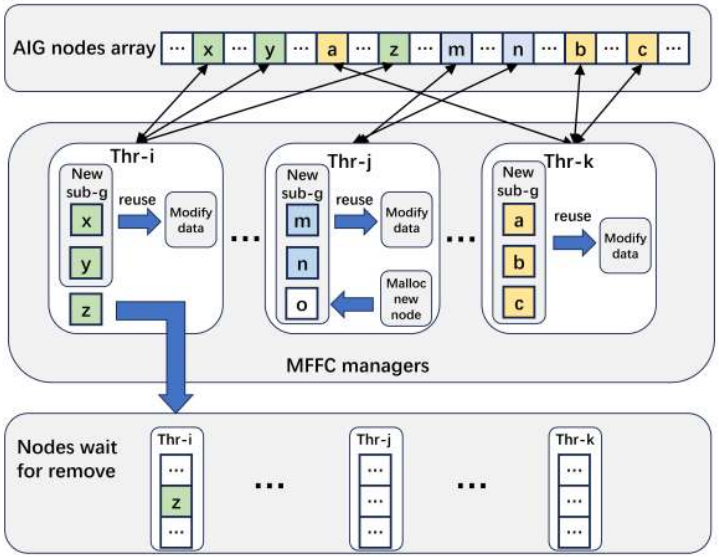


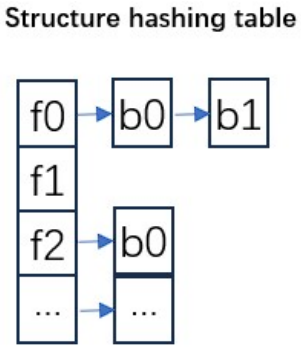
(b) AIG after deleting MFFC nodes of root node x

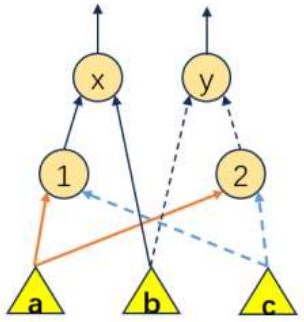
并行refactor算法研究

- 数据冲突
  - Aig nodes array
  - Structure hashing table
  - Cuts & MFFCs

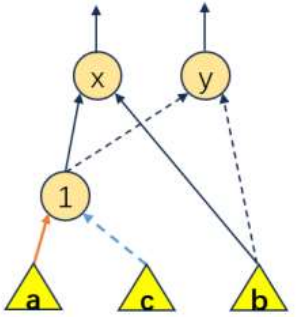








(a) AIG with function redundant nodes 1 & 2



(b) AIG after combining redundant nodes 1 & 2 to node 1

iEDA Tutorial 第四期议程

- Part1 iEDA-iMAP工具介绍 20min （倪利伟）

- Part2 AiMAP工艺映射算法 15min （刘俊锋）

- Part3 并行化逻辑重写算法 15min （杨宗霖）

- Part4 iEDA-iATPG工具介绍 20min （林晓泽）

01 研究内容

02 研究进展

03 未来计划

ATPG相关概念

- 缺陷（Defect）
  - 实际电路和预期设计之间存在的物理差异
  - 不正常的制造加工条件，工艺设计有误等原因造成
  - 引线的开路、短路等
- 故障模型（Fault Model）
  - 缺陷的逻辑层抽象级表示
  - 量化测试质量，自动化测试
  - Stuck-at故障模型，Transition故障模型，Delay故障模型等

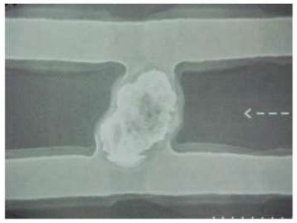
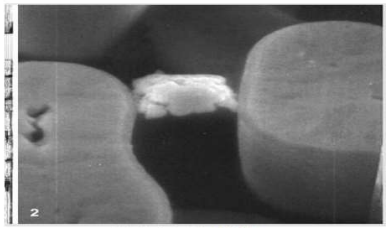
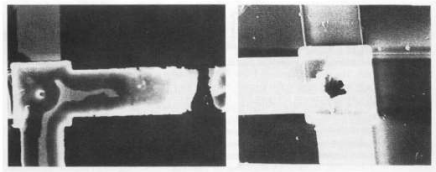


Photo 2-4 Dust-induced Wiring Short

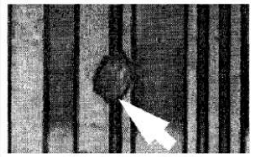


[Vallet IBM 1997]

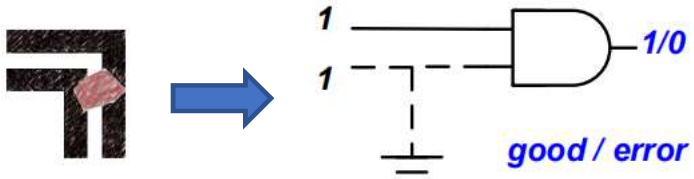


(a) Line-open failure. (b) Open failure in contact plug.

Figure 8.30 Electromigration-related failure modes (*Courtesy of N. Cheung and A. Tao, U.C. Berkeley*).

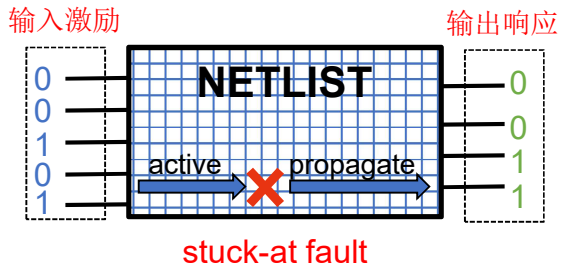


[Nigh IBM 1998]



ATPG相关概念

- 自动测试向量生成（Automatic Test Pattern generation，ATPG）
  - 针对给定的故障模型，自动地生成对应的测试向量
  - 测试向量：电路的输入激励（+ 电路的预期输出响应）
- ATPG性能的衡量标准
  - 故障覆盖率：（检测到的故障数量 / 总的故障数量）× 100%
  - 测试集的大小：测试向量的数量
  - 运行时间
- 故障模拟（Fault Simulation）
  - 通过模拟电路的行为，来比较无故障电路和故障电路在特定测试向量下的仿真结果
  - ATPG的重要组成部分
  - 评估给定的测试向量的故障覆盖率



iATPG总览

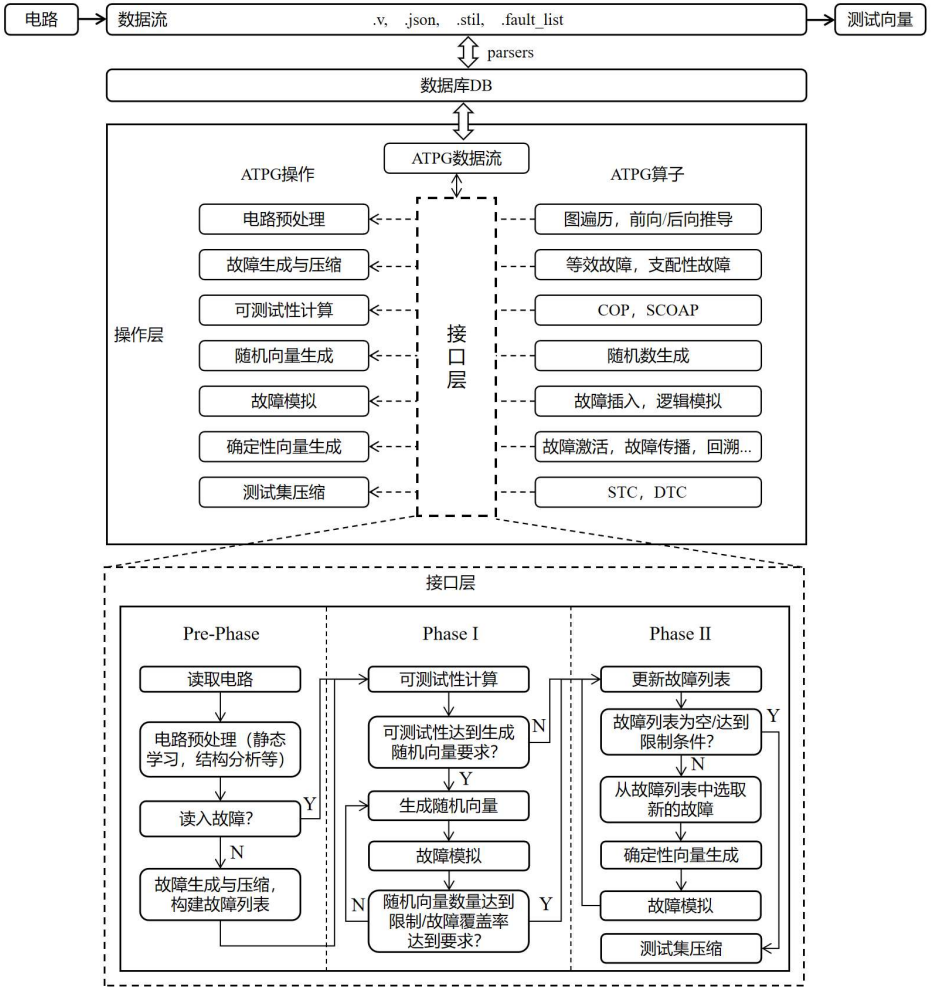
- iATPG介绍
  - 一款将在未来进行开源的测试向量生成工具
  - 当前目标
    - 支持单固定型故障模型
    - 支持全扫描链电路、部分扫描链电路
    - 支持基本扫描向量、时钟时序扫描向量、时钟输出扫描向量等结构测试向量
    - 支持功能测试向量
    - 易使用（兼容商用工具使用方式，可通过TCL脚本调用，支持STIL）
    - 追求更高的故障覆盖率，更短的运行时间，更少的测试向量数量

iATPG总览

- iATPG目前支持的功能
  - 读取与解析Verilog网表文件，JSON格式的ATPG library，STIL文件，故障列表文件
  - Standard cell网表到Primitive网表的转换与处理
  - Clock信号的声明与规则检查
  - 电路中原有扫描链的声明与规则检查
  - ATPG预处理操作
  - 单固定型故障生成、故障类别识别、故障压缩
  - 外部扫描向量读取，随机扫描向量生成
  - 基于扫描向量的逻辑模拟器
  - 基于扫描向量的故障模拟器

iATPG总览

- iATPG软件流程
  - 数据库DB
    - .v、.json、.stil、.fault_list等文件解析到自定义数据结构
  - 操作层
    - 电路预处理、故障生成与压缩、可测试性计算、故障模拟、向量生成等操作
  - ATPG算子
    - 操作层所使用的具体方法
  - 接口层
    - 通过结构层将ATPG操作和ATPG算子串通整个测试向量生成流程



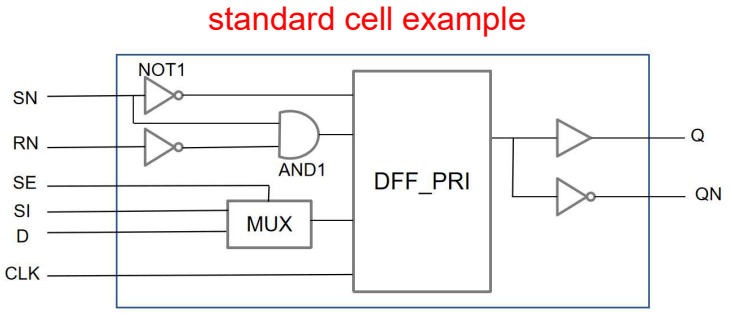
01 研究内容

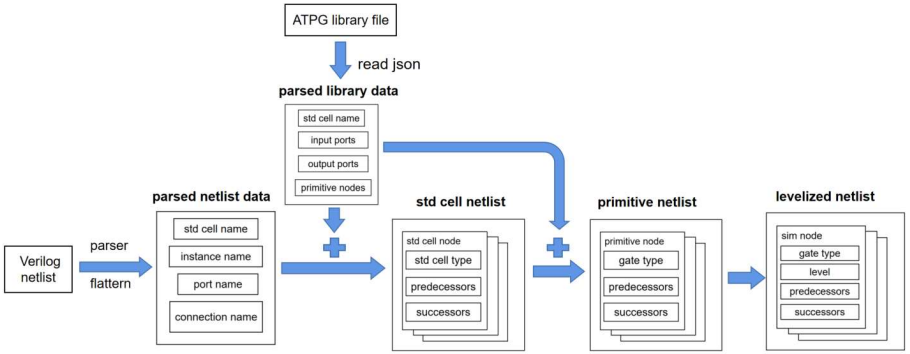
02 研究进展

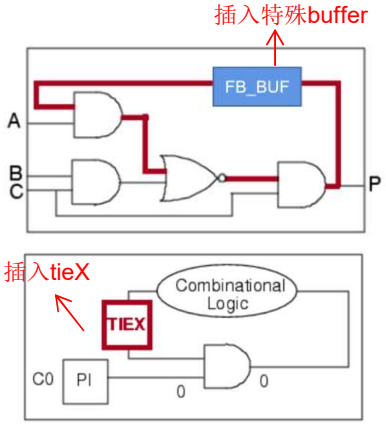
03 未来计划

iATPG关键特性

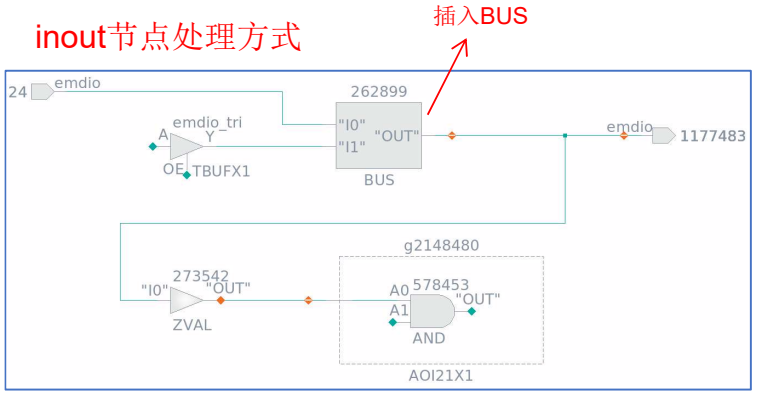
- 网表的转换与处理
  - 解析并获取Verilog网表的数据
  - 解析并获取ATPG library的cell model数据
  - 构建Standard cell网表
  - 结合cell model数据，构建Primitive网表，保存映射关系
  - Primitive网表做分级处理
  - 可调节最大扇出数量，可识别组合回路并断开回路，可处理悬空节点及inout节点





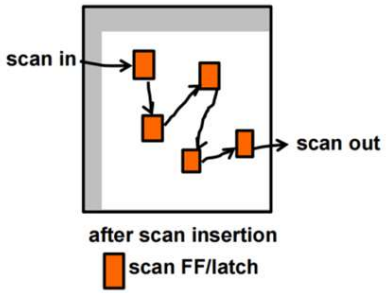


组合回路处理方式



iATPG关键特性

- Clock信号声明，电路原有的扫描链声明与规则检查
  - Clock信号的声明（port名字，关闭状态）
  - 检查电路中时序器件被Clock信号的驱动情况
  - 电路原有扫描链的声明（扫描链名字，输入引脚，输出引脚）
  - 解析STIL文件并获取相关test procedure数据
  - 结合当前的test procedure，检查扫描链的功能正确性



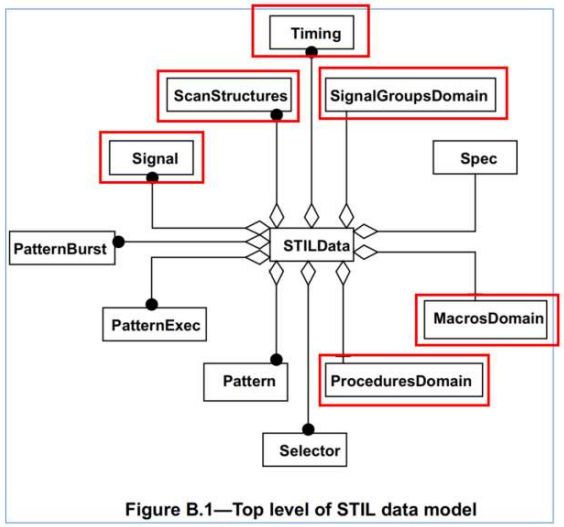


Figure B.1—Top level of STIL data model

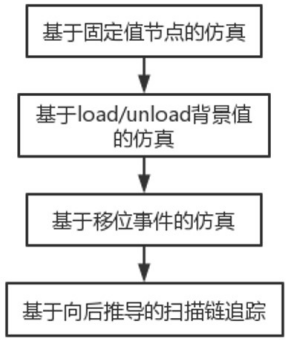
```
66  "load_unload_grp1" {
67    W tset_gen_tp1;
68    C { _po_ = \r2 X ; blif_clk_net = 0; blif_reset_net = 0; "_chain1_scan_out1_" = X; }
69    V { scan_en = 1; blif_clk_net = 0; blif_reset_net = 0; }
70    C { "_chain1_scan_in1_" = N; }
71    Shift { V { "_chain1_scan_in1_" = #; "_chain1_scan_out1_" = #; blif_clk_net = 1; blif_reset_net = 0; }
72    }
73  }
```

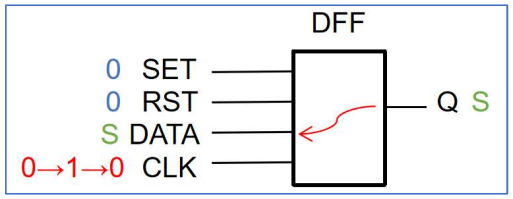
waveform table

load/unload背景值

需要检查的扫描链

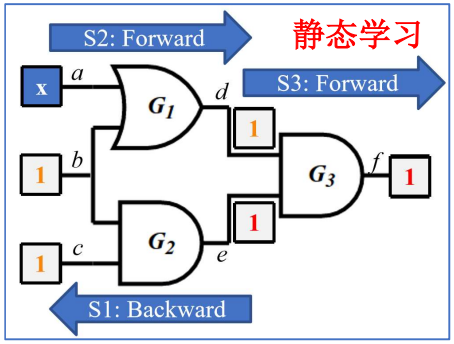
clock信号event





iATPG关键特性

- ATPG预处理操作
  - ➢ 可测试性计算（SCOAP）
  - ➢ 隐含推导关系识别、固定值节点识别（静态学习）
  - ➢ 冗余故障识别（FIRE）



	0-controllability	1-controllability	Observability
Combinational	$CC^0(N)$	$CC^1(N)$	$CO(N)$
Sequential	$SC^0(N)$	$SC^1(N)$	$SO(N)$

SCOAP

- ➢ Sandia Controllability Observability Analysis Program
- ➢ 针对每个node，计算六个值

CC0(N)：Combinational 0-controllability，**CC1(N)**：Combinational 1-controllability
Minimum number of combinational PI assignments and logic levels required to control a 0 or a 1 on node N

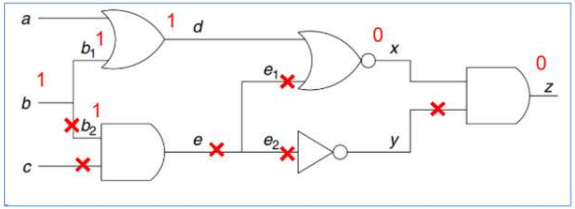
CO(N)：Combinational Observability
Minimum number of combinational PI assignments and logic levels required to propagate logical value on node N to PO

SC0(N)：Sequential 0-controllability，**SC1(N)**：Sequential 1-controllability
Minimum number of FF assignments (number of clock cycles) required to control 0 or 1 on node N

SO(N)：Sequential Observability
Minimum number of FF assignments required to propagate logical value on node N to a PO

FIRE

- ➢ Fault-Independent algorithm for REdundancy identification (FIRE)
- ➢ Traditional single-line-conflict analysis
- ➢ For each gate **g** in the circuit, the following two sets are computed
  - S0 — Set of faults not detectable when signal g = 0.
  - S1 — Set of faults not detectable when signal g = 1.
  - Faults that unexcitable or unobservable are considered as not detectable.
  - Any fault that is in the intersection of sets S0 and S1 would be untestable because it requires conflicting values on **g** as necessary conditions for its detection.



Faults unexcitable due to b = 1:
b/1, b_1/1, b_2/1, d/1, x/0, z/0

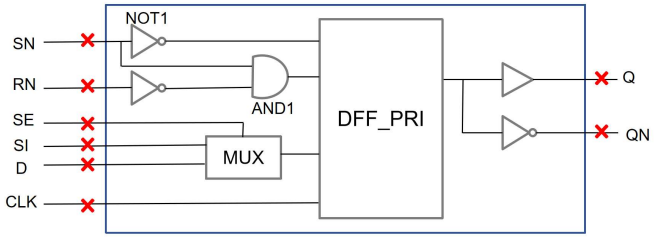
Faults unobservable due to b = 1:
a/0, a/1, e_1/0, e_1/1, y/0, y/1, e_2/0, e_2/1, e/0, e/1, c/0, c/1, b_2/0, b_2/1

Faults unexcitable and unobservable due to b = 0:
b/0, b_1/0, b_2/0, e/0, e_1/0, e_2/0, y/1, c/0, c/1

Redundant faults:
b2/0, e/0, e1/0, e2/0, y/1, c/0, c/1

iATPG关键特性

- 单固定型故障生成、故障类别识别、故障压缩、故障读取
  - 生成电路总的故障列表，记录映射关系（standard cell，primitive gate）
  - 识别等价故障，进行故障压缩
  - 识别特殊故障类别（untestable、detected by implication）
  - 外部输入故障列表读取



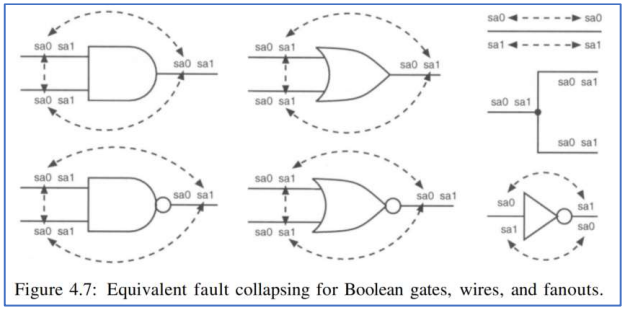
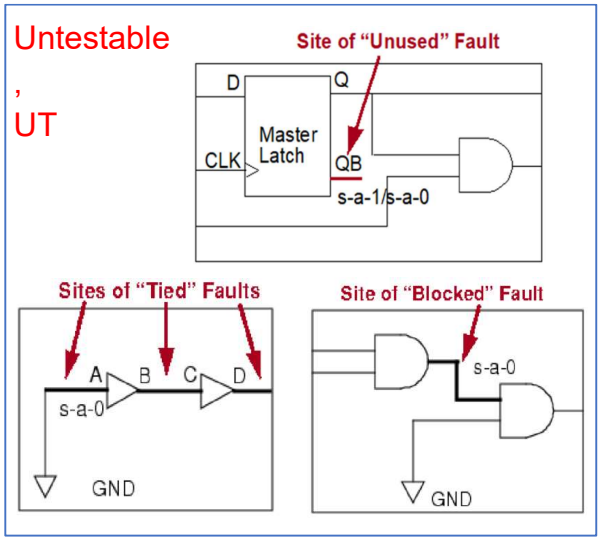


Figure 4.7: Equivalent fault collapsing for Boolean gates, wires, and fanouts.

Untestable
,
UT



Detected by implication, DI

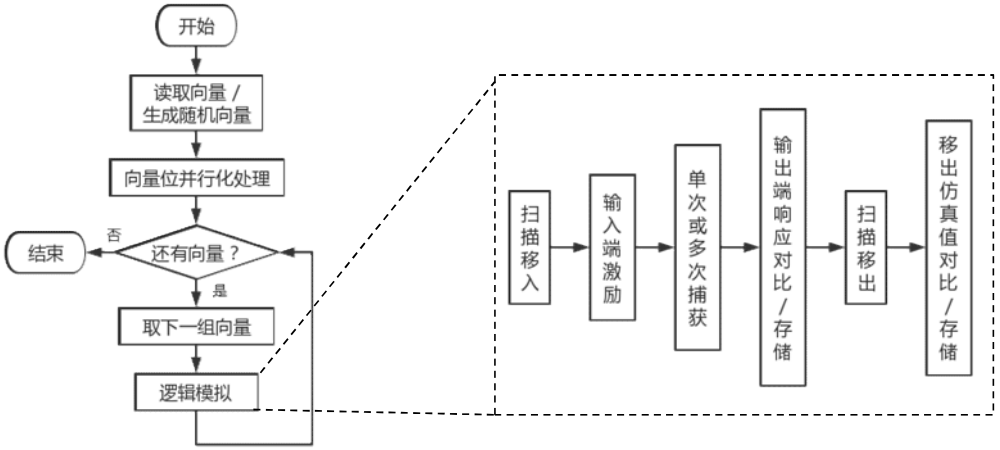
SCAN_PATH	SCAN	DI faults that are directly in the scan path.
SCAN_ENABLE	SEN	DI faults that can propagate to and disrupt scan shifting.
CLOCK	CLK	DI faults that are in either the scan or functional clock cone of state elements.

Fault list example

	type	code	pin_pathname
2			
3	----	------	----------
4	1	UC	/g137736/Y
5	0	EQ	/g137736/A
6	0	EQ	/reset
7	0	UC	/g137736/Y
8	1	EQ	/g137736/A
9	1	EQ	/reset
10	0	UC	/si[31]

iATPG关键特性

- 基于扫描向量的逻辑模拟
  - 读入并解析外部扫描测试向量（商用工具直接输出的STIL文件）
  - 支持对于全扫描链电路和部分扫描链电路的逻辑模拟
  - 支持基本扫描向量、时钟时序扫描向量、时钟输出扫描向量等向量类型
  - 支持向量的位并行化处理以加快逻辑模拟速度，支持仅对部分向量的逻辑模拟
  - 支持生成随机扫描测试向量并进行逻辑模拟
  - 支持向量以Verilog testbench格式输出，可交由商用或开源仿真器进行验证

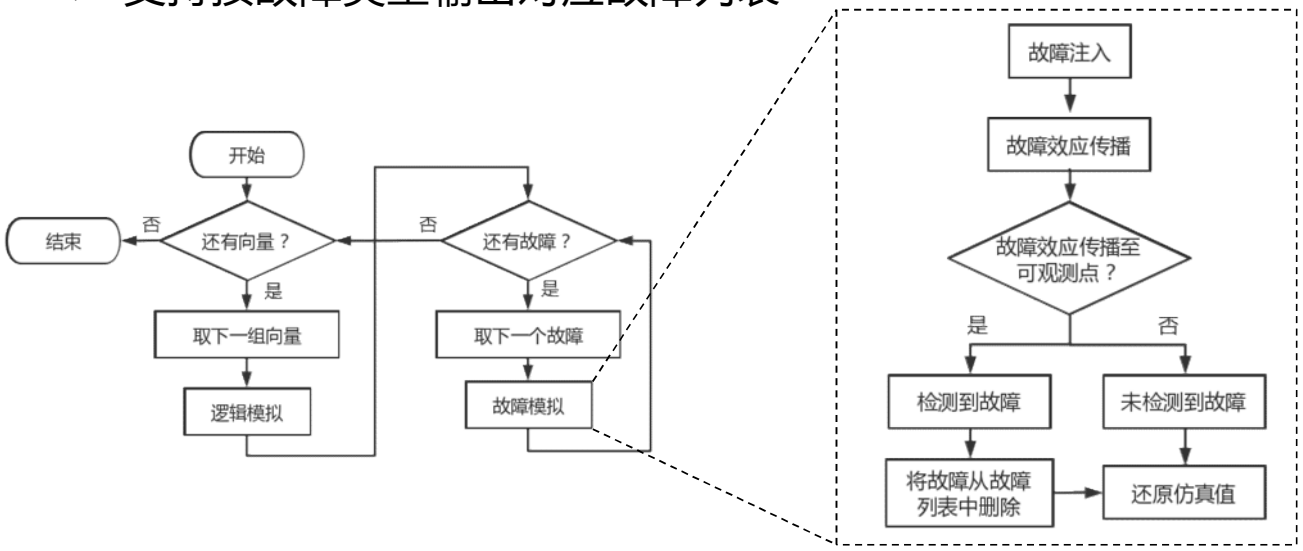


输出的Verilog testbench文件组成及内容

- .v file（testbench）
  信号和变量声明定义，实例化module，从.vec file获取pattern（激励和预期响应），激励信号，比较输出响应
- .vec file（pattern）
  按照一定格式组合force_pi，pulse_clk，load_val，unload_val，measure_po的值，附加表示vec_type等信息
- .cfg file（config）
  指出.vec file name，vec数量，pat数量（一个pat由多个vecs组成）

iATPG关键特性

- 基于扫描向量的故障模拟
  - 基于Parallel Pattern Single Fault Propagation（PPSFP）的故障模拟
  - 支持单固定型组合故障和时序逻辑故障
  - 支持以内部生成的压缩故障列表为目标故障
  - 支持以商用工具直接输出的故障列表为目标故障
  - 支持按故障类型输出对应故障列表




```
                Statistics Report
----------------------------------------
Fault Classes                    #faults

---------------------    -------------
  FU (full)                   887758
  ------------------    -------------
  DS (det_simulation)    762814 (85.93%)
  DI (det_implication)    90882 (10.24%)
  PT (posdet_testable)    14032 (1.58%)
  UT (untestable)         19742 (2.22%)
----------------------------------------
Fault Sub-classes
  ------------------
  DI (det_implication)
    SCAN (scan_path)      51952 (5.85%)
    SEN  (scan_enable)    12956 (1.46%)
    CLK  (clock)          25974 (2.93%)
  ------------------
  UT (untestable)
    UU   (unused)         11000 (1.24%)
    TI   (tied)            8742 (0.98%)
    BL   (blocked)            0 (0.00%)
----------------------------------------
Coverage
  ------------------
  test_coverage                  99.16%
  fault_coverage                 96.95%
----------------------------------------
CPU_time (secs)                  49.48
----------------------------------------
```

iATPG的API与TCL命令

- 截止目前的部分关键面向开发者的API

API	功能简述
readVerilog	解析Verilog网表文件
readLibrary	解析ATPG library文件
readSTIL	解析STIL文件
readFaultList	解析故障列表文件
buildNetlist	构建及转换网表
addClock	声明clock信号
addScanChain	声明电路原有的扫描链
checkDesignRule	clock信号和扫描链相关规则检查
preprocessBeforeATPG	ATPG相关预处理操作
readPattern	从STIL数据中获取扫描向量
createFault	构建内部故障列表
simulatePattern	逻辑模拟 / 故障模拟（取决于有没有故障）
writePattern	以Verilog testbench格式输出扫描向量

iATPG的API与TCL命令

- 截止目前的面向用户的TCL命令

TCL命令	功能说明	参数
read_verilog	读入Verilog网表文件	网表文件名
read_library	读入ATPG library文件	ATPG library文件名
read_stil	读入STIL文件	STIL文件名
read_fault_list	读入故障列表文件	故障列表文件名
add_clock	声明clock信号	port名字，-off_state [0 | 1]
add_scan_chain	声明电路原有的扫描链	扫描链名字，输入引脚，输出引脚
design_rule_checking	clock信号和扫描链相关规则检查	无
read_pattern	从STIL数据中获取扫描向量	无
set_random_pattern	设置生成随机测试向量的数量	测试向量的数量
create_fault	构建内部故障列表	无
simulate_pattern	执行逻辑模拟 / 故障模拟	-source [external | random] -start_pat_id [int] -end pat_id [int]
write_pattern	输出扫描向量文件	-source [external | random] -start_pat_id [int] -end pat_id [int]

01 研究内容

02 研究进展

03 未来计划

iATPG未来计划

- 优化基于扫描向量的故障模拟
  - 启发式方法
    - 减少故障列表中需要进行故障模拟的故障数量
    - 减少故障模拟过程中事件的数量
    - 复用已进行的故障模拟仿真结果，增量地进行故障模拟
  - CPU多线程加速
    - 发挥单核计算能力强、内存大等优点，提出可延展性强的并行算法
  - GPU并行加速
    - 规避GPU显存小、单核计算能力弱、对代码路径分支容忍度低等缺点，同时发挥GPU核多的优点，结合故障模拟的特点，提出高效的并行算法

iATPG未来计划

- 基于功能向量的故障模拟
  - 功能向量与扫描向量（结构测试向量）不同
  - 功能向量本质上是时序向量，在时间轴的不同时钟周期上有不同的向量输入值，且电路内部节点的逻辑状态依赖于前面时钟周期输入的仿真结果
  - 无法采用常用于结构测试向量模拟的位并行方法
  - 采用Parallel Fault Propagation的故障模拟
  - 功能向量可能涵盖数万甚至上百万个时钟周期的电路运行过程，这也导致了针对功能向量的故障模拟十分耗时
  - 挖掘能够提升功能向量的故障模拟性能的方法

iATPG未来计划

- 继续推进iATPG工具的开发
  - 实现ATPG系统全流程
    - 全扫描链电路的测试向量生成
    - 部分扫描链电路的测试向量生成
    - 静态测试集压缩
    - 动态测试集压缩
  - 提升iATPG的性能，挖掘具有研究价值的点
    - 启发式方法
    - 并行加速
    - 机器学习

THANK YOU!

倪利伟
nlwmode@gmail.com

iEDA开源交流群