Re-implement “Constraint Solving for Synthesis and Verification of Threshold Logic Circuits”

LSV Final Project Presentation

r10943109 丁宣勻　　r10943089 陳妍喻

Outline

- Introduction
- Problem Formulation
- Methods
- Experiment

Outline

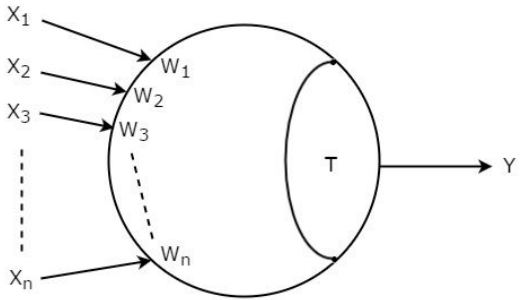
- Introduction
- Problem Formulation
- Methods
- Experiment

Introduction

- 2020 IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems
  - Nian-Ze Lee, Jie-Hong R. Jiang, National Taiwan University, Taiwan

- Threshold Logic (TL) function [w1,...,wn; T]

$$f(x_1, \ldots, x_n) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} (w_i \cdot x_i) \geq T, \\ 0, & \text{otherwise,} \end{cases}$$



- Neural network applications: [5], [10], [11], [18], [27]

[5] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016. [Online]. Available: arXiv:1602.02830.
[10] G.-B. Huang, Q.-Y. Zhu, K.-Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, "Can threshold networks be trained directly?" IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 53, no. 3, pp. 187–191, Mar. 2006.
[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 4107–4115.
[18] R. P. Lippmann, "An introduction to computing with neural nets," IEEE ASSP Mag., vol. 4, no. 2, pp. 4–22, Apr. 1987.
[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in Proc. Eur. Conf. Comput. Vis., 2016, pp. 525–542.

Introduction

Contributuion

- A formulation of the collapse operation for TL functions and a necessary and sufficient condition for collapsibility. The synthesis algorithm based on iteratively collapsing TLGs achieves an average of 18% gate count reduction on top of synthesized TL circuits.

- A conversion from a TL function to a MUX tree, which enables the use of efficient verification techniques for Boolean logic circuits and uniquely solves the equivalence checking of TL circuits for all benchmark circuits.

- A linear-time translation from a TL function to PB constraints, which is applicable to benchmarks containing TL functions with large input sizes arising from practical neural networks.

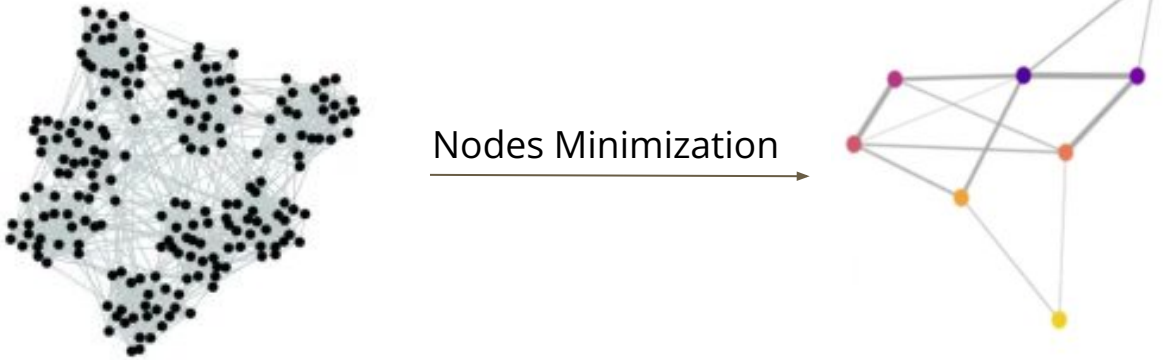
Outline

- Introduction
- Problem Formulation
- Methods
- Experiment

Problem Formulation - General Perspective

COLLAPSE A TL CIRCUIT

- Given a TL circuit, minimize its TLG counts as much as possible.
- Minimization by collapse operation.



Problem Formulation - Local Perspective

COLLAPSE TWO TCGS VIA LINEAR COMBINATION

- Given two TLGs
  - $u = [a_1 , \ldots , a_n ; T_u]$ with inputs $(x_1 , \ldots , x_n)$
  - $v = [b_1 , \ldots , b_m ; T_v]$ with inputs $(y_1 , \ldots , y_m)$
  - let u be a fanin of v, and assume $y_1 = z_u$.

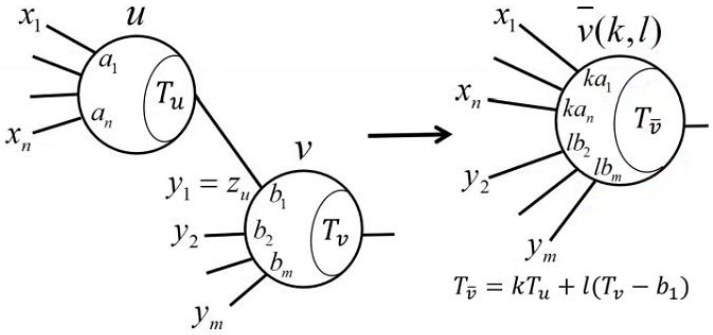


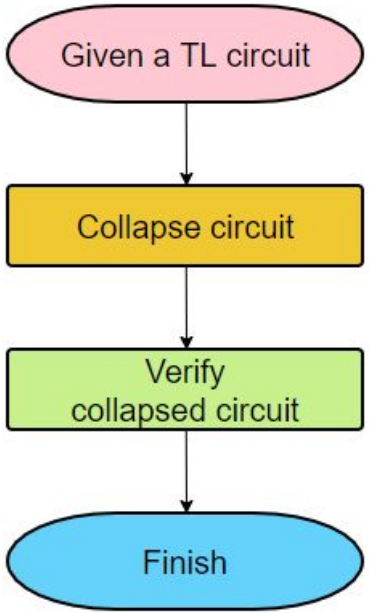
Fig. 1. Collapse of two TLGs via their linear combination.

- Whether there exists two positive parameters k and l such that
  - $TLG\ \bar{}\ v (k , l) = [ka_1 , \ldots , ka_n , lb_2 , \ldots , lb_m ; kT_u + l (T_v - b_1)]$ with inputs $(x_1 , \ldots , x_n , y_2 , \ldots , y_m)$ satisfies $f\ \bar{}\ v (x_1 , \ldots , x_n , y_2 , \ldots , y_m) = fv (fu (x_1 , \ldots , x_n), y_2 , \ldots , y_m)$ for all truth assignments to variables $(x_1 , \ldots , x_n , y_2 , \ldots , y_m)$.

Outline

- Introduction
- Problem Formulation
- Methods
- Experiment

Flow


Given a TL circuit
Collapse circuit
Verify
collapsed circuit
Finish

Methods - Synthesis

- Define 3 functions

$$f_u^+(x_1, \ldots, x_n) \overset{\text{def}}{=} \sum_{i=1}^{n} a_i x_i$$

$$f_v^-(y_2, \ldots, y_m) \overset{\text{def}}{=} \sum_{j=2}^{m} b_j y_j$$

$$f_{v \circ u} \overset{\text{def}}{=} f_v(f_u(x_1, \ldots, x_n), y_2, \ldots, y_m)$$

- Define 4 truth assignments

$$\phi_1 \overset{\text{def}}{=} \{(\beta_2, \ldots, \beta_m) \mid f_v^-(\beta_2, \ldots, \beta_m) \leq T_v - b_1 - 1\}$$

$$\phi_2 \overset{\text{def}}{=} \{(\beta_2, \ldots, \beta_m) \mid f_v^-(\beta_2, \ldots, \beta_m) \geq T_v\}$$

$$\phi_3 \overset{\text{def}}{=} \{(\beta_2, \ldots, \beta_m) \mid T_v - b_1 \leq f_v^-(\beta_2, \ldots, \beta_m) \leq T_v - 1\}$$

$$\phi_4 \overset{\text{def}}{=} \{(\alpha_1, \ldots, \alpha_n) \mid f_u^+(\alpha_1, \ldots, \alpha_n) \leq T_u - 1\}$$

for $(\alpha_1, \ldots, \alpha_n) \in \mathbb{B}^n$ and $(\beta_2, \ldots, \beta_m) \in \mathbb{B}^{m-1}$.

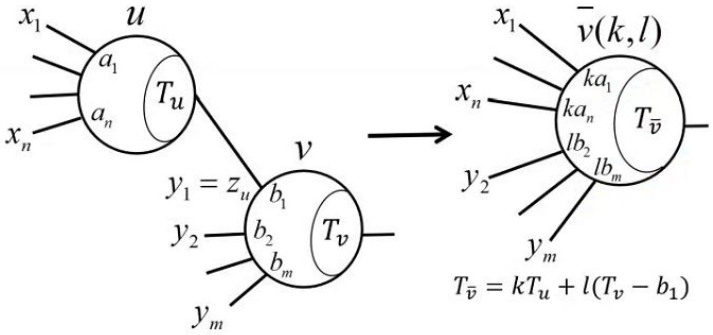


Fig. 1. Collapse of two TLGs via their linear combination.

Methods - Synthesis

- Theorem 1
  - Necessary and sufficient conditions
  - Difficult to compute : Subset Sum Problem (NP-complete)
  -
$$l(T_v - b_1 - \max_{\beta\in\phi_1}\{f_v^-\}) \geq k(\max\{f_u^+\} - T_u) + 1,\ \phi_1 \neq \emptyset$$
$$l(\min_{\beta\in\phi_2}\{f_v^-\} + b_1 - T_v) \geq k(T_u - \min\{f_u^+\}),\ \phi_2 \neq \emptyset$$
$$k(T_u - \max_{\alpha\in\phi_4}\{f_u^+\}) \geq l(\max_{\beta\in\phi_3}\{f_v^-\} + b_1 - T_v) + 1.$$

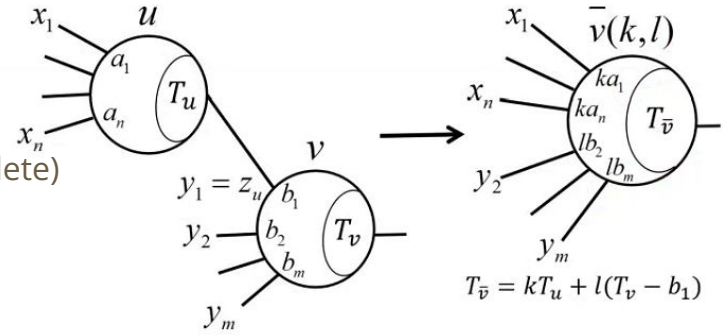
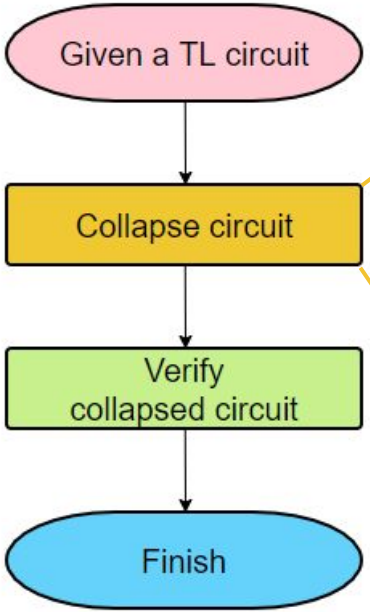


Fig. 1. Collapse of two TLGs via their linear combination.

- Theorem 2
  - Sufficient conditions
  - Efficient to compute : Time linear to the fanin size of the TLG
  -
$$l \geq k(\max\{f_u^+\} - T_u) + 1,\ \phi_1 \neq \emptyset$$
$$lb_1 \geq k(T_u - \min\{f_u^+\}),\ \phi_2 \neq \emptyset$$
$$k \geq l(b_1 - 1) + 1.$$

Flow



CollapseNtk
input: a TL circuit $G = (V, E)$ and a bound B
output: a collapsed TL circuit $G' = (V', E')$
begin
01 unmark every $v \in V$;
02 **while** some $v \in V$ is unmarked
03 **foreach** $v \in V$
04 **foreach** fanin u of v
05 **if** $|fanouts(u)| \leq B$
06 **if** u can be collapsed to all of its fanouts
07 **foreach** fanout t of u
08 w := *CollapseNode*(u,t);
09 unmark w;
10 $V := V \backslash \{t\} \cup \{w\}$;
11 $V := V \backslash \{u\}$;
12 **continue**; `//to next v`
13 **if** u is the last fanin of v `//no collapse`
14 mark v;
15 **return** (V, E);
end

Fig. 3. Algorithm: Collapse a TL circuit.

Implementation

CollapseNtk

input: a TL circuit $G = (V, E)$ and a bound B
output: a collapsed TL circuit $G' = (V', E')$
begin

```
01  unmark every v ∈ V;
02  while some v ∈ V is unmarked
03     foreach v ∈ V
04        foreach fanin u of v
05           if |fanouts(u)| ≤ B
06              if u can be collapsed to all of its fanouts
07                 foreach fanout t of u
08                    w := CollapseNode(u,t);
09                    unmark w;
10                    V := V\{t} ∪ {w};
11                 V := V\{u};
12                 continue; //to next v
13           if u is the last fanin of v //no collapse
14              mark v;
15  return (V, E);
```

end

Fig. 3. Algorithm: Collapse a TL circuit.

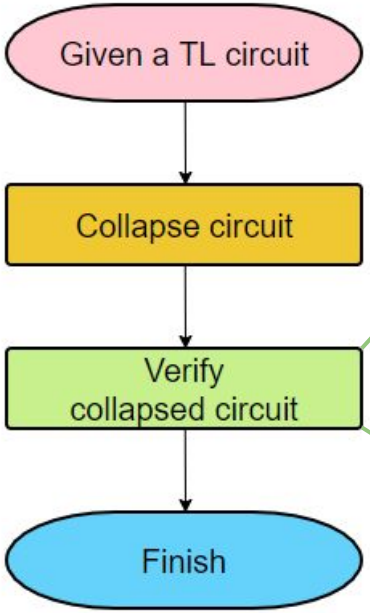
```cpp
void Lsv_collapse(int max_bound) {
    Th_Node* v;
    Th_Node* u;
    for (int bound = 1; bound <= max_bound; ++bound) {
        for (int i = 0; i < th_list.size(); ++i) {
            v = th_list[i];
            v->ref = 1 - globalref;
        }
        bool f_has_collapsed;
        do {
            f_has_collapsed = false;
            for (int i = 0; i < th_list.size(); ++i) {
                v = th_list[i];
                if (Lsv_skip_node(v))  continue;
                for (int j = 0; j < v->fanins.size(); ++j) {
                    u = v->fanins[j];
                    if (Lsv_skip_node(u))                continue;
                    if (u->fanouts.size() >= bound)      continue;
                    int v_ori_size = v->fanins.size() - 1;
                    if (Lsv_collapse2fanouts(u, bound)) {
                        f_has_collapsed = true;
                        for (int k = 0; k < th_list.size(); k++) {
                            if (th_list[k] == u) {
                                th_list.erase(th_list.begin()+k);
                                delete u;
                                break;
                            }
                        }
                        for (int k = v_ori_size; k < v->fanins.size(); k++) {
                            for (int l = 0; l < v_ori_size; l++) {
                                if (v->fanins[k] == v->fanins[l]) {
                                    v->weights[l] += v->weights[k];
                                    v->weights.erase(v->weights.begin()+k);
                                    v->fanins.erase(v->fanins.begin()+k);
                                    k--;
                                    break;
                                }
                            }
                        }
                    }
                    if (j == v->fanins.size()- 1)
                        v->ref = globalref;
                }
            }
        } while(f_has_collapsed);
    }
}
```

Methods - Verification

- MUX-based Verification Method
  - More efficient for benchmarks synthesized from conventional circuits
  - Small fanin sizes of TLGs (roughly less than 20)
  - Steps:
    - Convert TL circuits to MUX circuits
    - Use “cec” command in abc for equivalence check

- PB-based Verification Method
  - Applicable to benchmarks with fanin size up to 128 (relax the input size restriction)
  - Neural network applications
  - Steps:
    - Convert TL circuits to PB constraints
    - Equivalence check

Flow



ConvertTLGMUX
input: a TLG v and a Boolean logic circuit C
output: a Boolean logic circuit C' equivalent to v
begin
01 **if** *IsConstOne*(v)
02 **return** `CONST1`;
03 **if** *IsConstZero*(v)
04 **return** `CONST0`;
05 x_{i^*} := *SelectMaxAbsWeigthInput*(v);
06 m := *CreateMuxGate*(C);
07 *m.SetControlInput*(x_{i^*});
08 *m.SetDataZeroInput*(*ConvertTLGMux*($v_{\neg x_{i^*}}$, C).`PO`);
09 *m.SetDataOneInput*(*ConvertTLGMux*($v_{x_{i^*}}$, C).`PO`);
10 *C.SetPrimaryOutput*(m);
11 **return** C;
end

Fig. 2. Algorithm: Convert a TLG into a MUX tree.

Implementation

ConvertTLGMUX

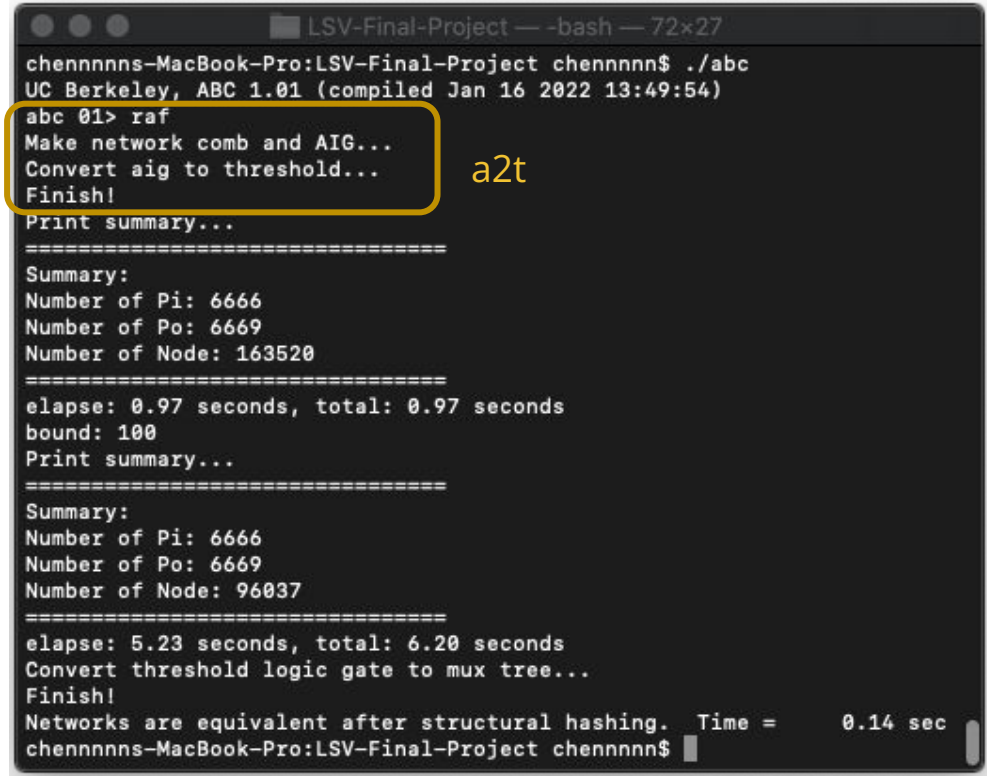
input: a TLG v and a Boolean logic circuit C
output: a Boolean logic circuit C' equivalent to v
begin
01 **if** *IsConstOne*(v)
02 **return** CONST1;
03 **if** *IsConstZero*(v)
04 **return** CONST0;
05 x_{i^*} := *SelectMaxAbsWeigthInput*(v);
06 m := *CreateMuxGate*(C);
07 m.*SetControlInput*(x_{i^*});
08 m.*SetDataZeroInput*(*ConvertTLGMux*($v_{\neg x_{i^*}}$, C).PO);
09 m.*SetDataOneInput*(*ConvertTLGMux*($v_{x_{i^*}}$, C).PO);
10 C.*SetPrimaryOutput*(m);
11 **return** C;
end

Fig. 2. Algorithm: Convert a TLG into a MUX tree.

```
void Lsv_th2mux() {
    int i, j;
    vector<Th_Node*> th_PO_list;
    // sort th_list in topologocal order
    sort_th();

    // create mux network
    Abc_Ntk_t * pNtk_th2mux;
    char buf[1000];
    pNtk_th2mux = Abc_NtkAlloc( ABC_NTK_STRASH , ABC_FUNC_AIG , 1 );
    sprintf(buf , "th2mux");
    pNtk_th2mux->pName = Extra_UtilStrsav(buf);

    // for each PI/PO/const1 create gate
    th2aigNode[th_list[0]] = Abc_AigConst1(pNtk_th2mux);
    for (i = 0; i < th_list.size(); i++) {
        if (th_list[i]->type == TH_PI) {
            th2aigNode[th_list[i]] = Abc_NtkCreatePi(pNtk_th2mux);
        }
        else if (th_list[i]->type == TH_PO) {
            th_PO_list.push_back(th_list[i]);
            th2aigNode[th_list[i]] = Abc_NtkCreatePo(pNtk_th2mux);
        }
    }

    // for each TLG: call convertTLGMUX
    for (i = 0; i < th_list.size(); i++) {
        if (th_list[i]->type == TH_NODE) {
            th2aigNode[th_list[i]] = thg2mux_recur(th_list[i], pNtk_th2mux);
        }
    }

    // line 10 : set primary output (connect Po)
    for (i = 0; i < th_PO_list.size(); i++) {
        for (j = 0; j < th_PO_list[i]->fanins.size(); j++) {
            if (th_PO_list[i]->weights[j] == 1) { // buffer
                Abc_ObjAddFanin(th2aigNode[th_PO_list[i]],
                                th2aigNode[th_PO_list[i]->fanins[j]]);
            } else if (th_PO_list[i]->weights[j] == -1) { // inverter
                Abc_ObjAddFanin(th2aigNode[th_PO_list[i]], A
                                bc_ObjNot(th2aigNode[th_PO_list[i]->fanins[j]]));
            } else { assert(0); }
        }
    }

}
```

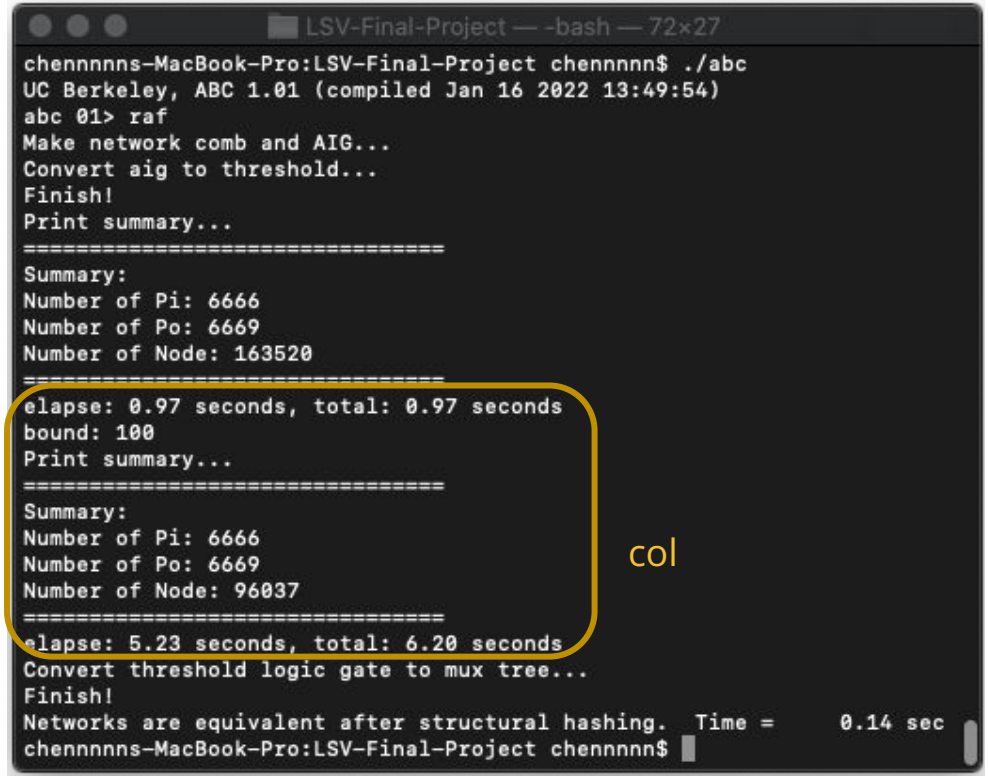
Commands

- Convert aig to threshold logic
  - lsv_aig2th (or a2t)
- Collapse
  - lsv_collapse (or col)
- Convert threshold logic to mux trees
  - lsv_th2mux (or t2m)
- Print function
  - lsv_print_th (or pth)



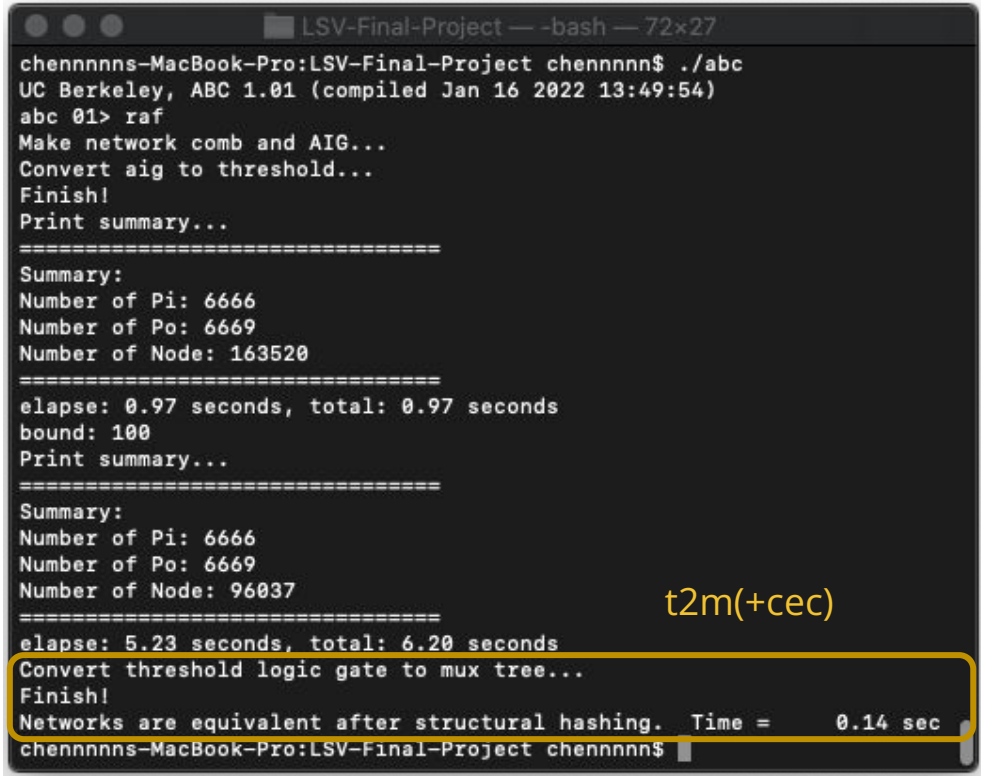
Commands

- Convert aig to threshold logic
  - lsv_aig2th (or a2t)
- Collapse
  - lsv_collapse (or col)
- Convert threshold logic to mux trees
  - lsv_th2mux (or t2m)
- Print function
  - lsv_print_th (or pth)



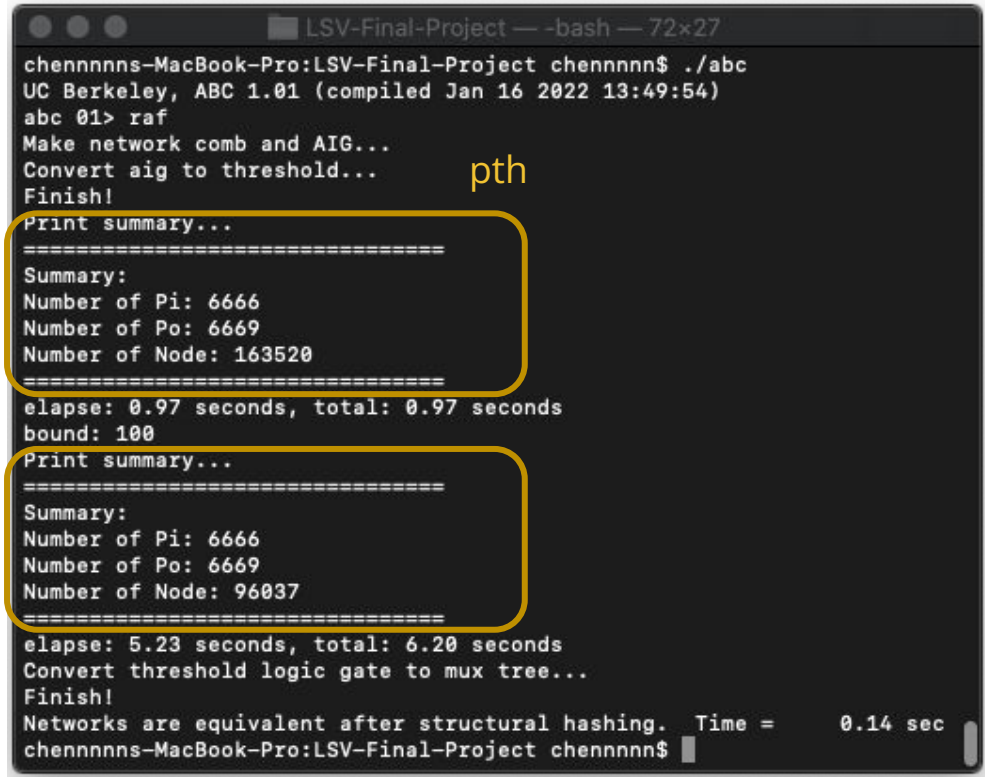
Commands

- Convert aig to threshold logic
  - lsv_aig2th (or a2t)
- Collapse
  - lsv_collapse (or col)
- Convert threshold logic to mux trees
  - lsv_th2mux (or t2m)
- Print function
  - lsv_print_th (or pth)



Commands

- Convert aig to threshold logic
  - lsv_aig2th (or a2t)
- Collapse
  - lsv_collapse (or col)
- Convert threshold logic to mux trees
  - lsv_th2mux (or t2m)
- Print function
  - lsv_print_th (or pth)



Outline

- Introduction
- Problem Formulation
- Methods
- Experiment

Experimental Results

- Implemented in ABC environment
- The combinational logic of circuits in ISCAS benchmark suites
- Fanout bound parameter B increases from 1 up to 100 iteratively
- **Do not have limitation on fanin size and weight/threshold value limitation !**

benchmarks			AIG	before collapsed	OUR after collapsed		collapse rate	PAPER after collapsed		collapse rate
circuit	#pi	#po	#AIG	#TLG	#TLG	time		#TLG	time	
c6288	32	32	2334	2337	2322	0.1	99%	1404	0.12	60%
c7552	207	108	1961	2074	1712	0.1	87%	846	0.07	43%
s13207	700	790	2605	2719	2162	0.1	83%	1190	0.06	46%
s15850	611	684	3330	3560	2875	0.14	86%	1479	0.07	44%
s35932	1763	2048	10124	11948	11948	0.36	118%	4758	0.13	47%
s38417	1664	1742	9062	9219	7889	0.4	87%	4388	0.25	48%
s38584	1464	1730	11646	12400	9907	0.56	85%	4639	0.23	40%
b14	277	299	5609	6070	4780	0.32	85%	2565	0.23	46%
b15	485	519	8158	8448	6340	0.49	78%	3667	0.24	45%
b17	1452	1512	26389	27567	21056	1.7	80%	12027	0.83	46%
b18	3357	3343	77757	81710	63231	5.8	81%	35343	4.58	45%
b19	6666	6669	156224	163520	127082	12.48	81%	70765	7.21	45%
b20	522	512	11552	12219	9937	0.76	86%	5284	0.48	46%
b21	522	512	11728	12782	10317	0.73	88%	5381	0.43	46%
b22	767	757	17614	18488	15000	1.05	85%	8028	0.61	46%

Experimental Results

- Different bound numbers
  - The bigger the bound is, the better the result is.

benchmarks			AIG	before collapsed	bound	after collapsed		collapse rate
circuit	#pi	#po	#AIG	#TLG		#TLG	time	
b19	6666	6669	156224	163520	10	131301	3.13	84%
					50	127836	4.58	82%
					100	127082	12.11	81%
					200	125561	21.09	80%

- Different increment of bound number per iterations
  - The bigger the increment is, the worse the result is.

benchmarks			AIG	before collapsed	bound incease amount	after collapsed		collapse rate
circuit	#pi	#po	#AIG	#TLG		#TLG	time	
b19	6666	6669	156224	163520	1	127082	12.47	81%
					5	128490	4.83	82%
					10	129260	3.49	83%
					20	130770	2.83	84%

Thank You

GitHub Link: https://github.com/mirkat1206/LSV-Final-Project