Fast Synthesis of Threshold Logic Networks with Optimization

Yung-Chih Chen, Runyi Wang, and Yan-Ping Chang*
Yuan Ze University, Taiwan

Outline

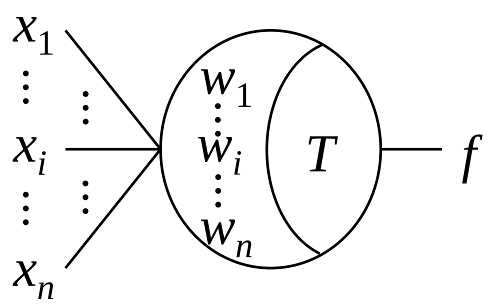
- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion

Threshold logic

- Threshold logic is an alternative representation to conventional Boolean logic

- Logical function f of a threshold logic gate is defined as follows:

$$f(x_1, x_2, \ldots, xn) = \begin{cases} 1, \text{if } \sum_{i=1}^{n} x_i w_i \geq T \\ 0, \text{otherwise} \end{cases}$$



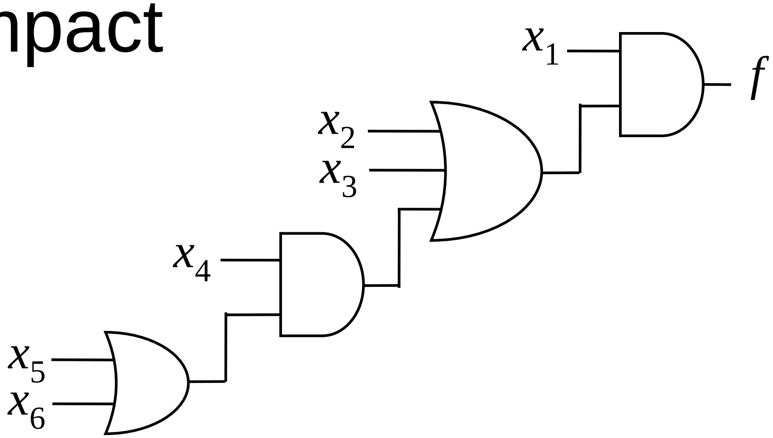
- Threshold logic network (TLN)
  - A logic network composed of threshold gates

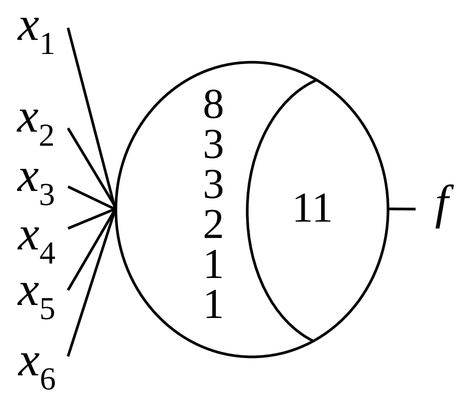
Threshold logic

- Development of threshold logic
  - Started in 1960s, but had only a little impact on today's IC designs
    - lack of effective hardware implementation
  - Re-attracted attention in recent years
    - advances in nanoscale device technology
      - Resonant tunneling diodes, quantum cellular automata, and single-electron transistor
      - They are possible devices for threshold logic implementation
  - Design automation techniques
    - synthesis, optimization, verification, static timing analysis, and automatic test pattern generation

Advantages of threshold logic (1/2)

- Compatible with nanoscale devices
- Compact

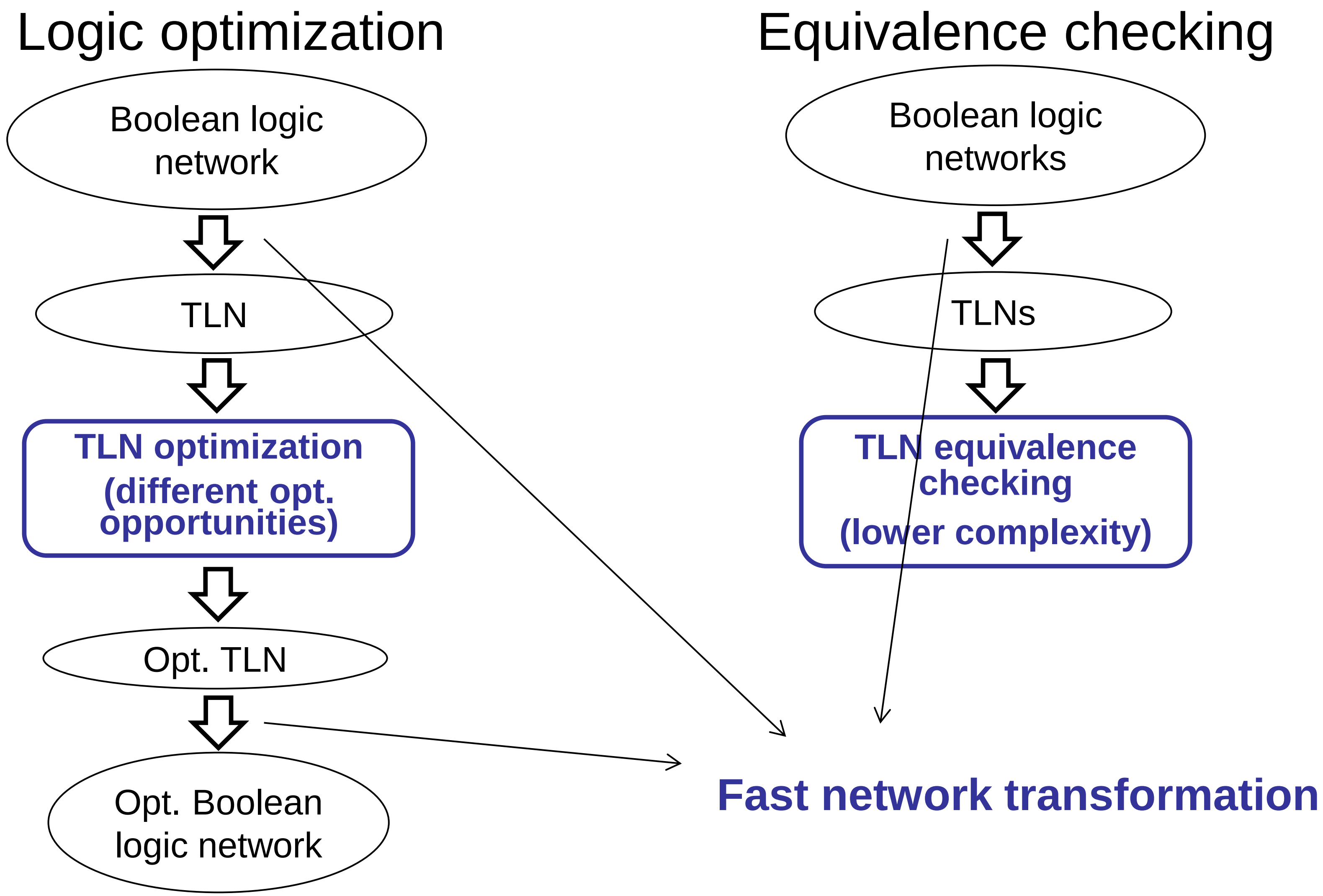




$$f = x_1(x_2+x_3+(x_4(x_5+x_6)))$$

- Could be a good intermediate representation in today's design flow
  - Used to enhance logic optimization and design verification

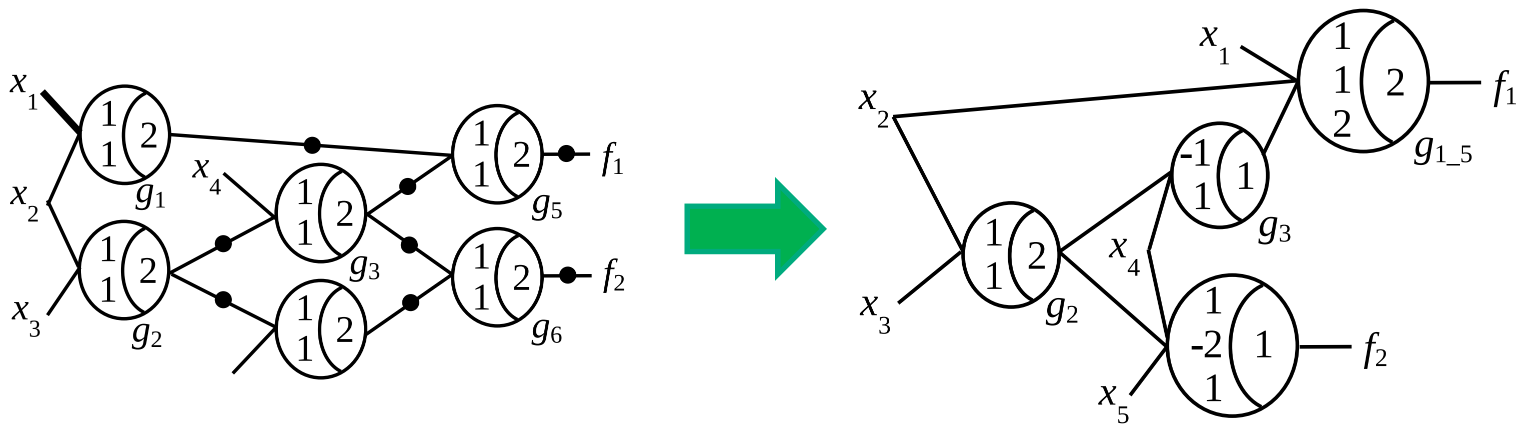
Advantages of threshold logic (2/2)



TLN synthesis

- We aim to propose a FAST TLN synthesis approach

- Problem formulation
  - Input: a conventional Boolean logic
  - Output: a TLN with minimized gate count



Previous works on TLN synthesis

- Work based on a threshold function[1] identification procedure
  - Integer linear programming (ILP)-based
  - Binary decision diagram or truth table-based
- For a Boolean function
  - A threshold function → weights and threshold value
  - Not a threshold function → function decomposition
- For a Boolean logic network to be synthesized
  - They repeatedly identify and map all the sub-functions into threshold logic gates
- Main disadvantage
  - Inefficiency

[1]Threshold function: a Boolean function which can be implemented with only one threshold logic gate

Our approach

- A fast synthesis approach without threshold function identification
  - Faster
  - Better or competitive synthesis quality

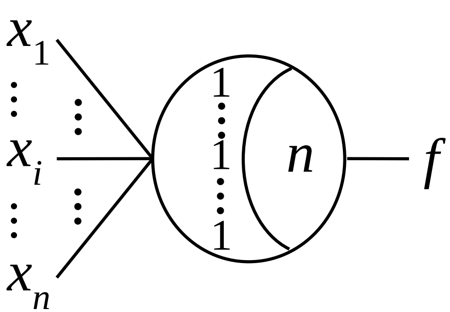
Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion

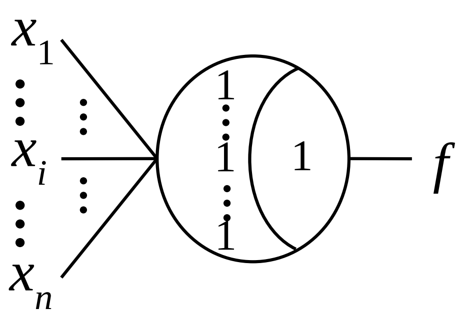
Threshold function

- A threshold function
  - A Boolean function which can be implemented with only one threshold logic gate

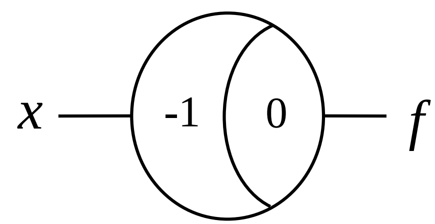
- Conventional primitive functions, such as AND, OR, and NOT, are threshold functions



n-input AND



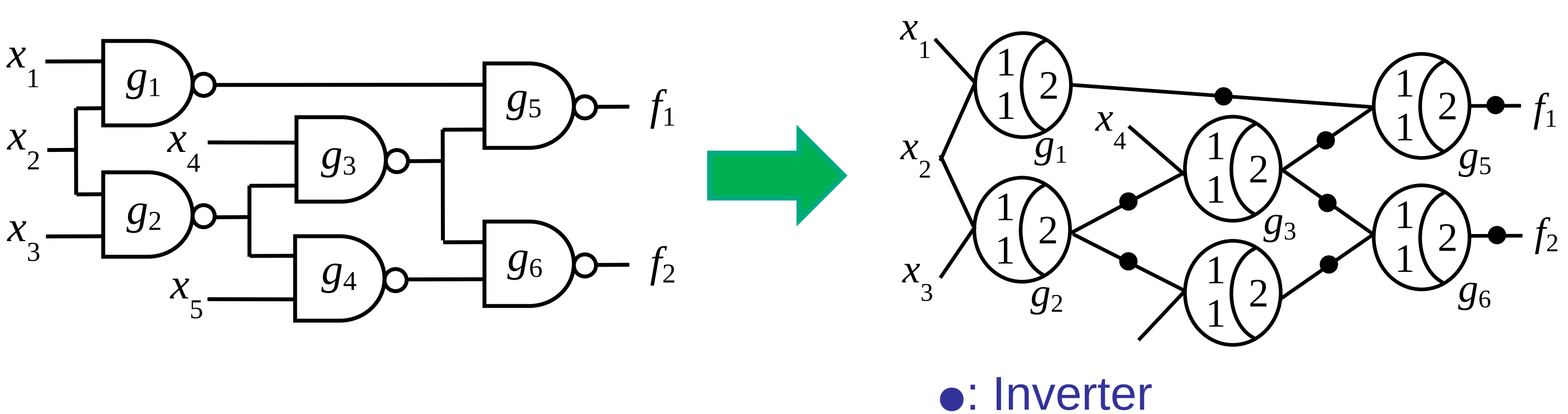
n-input OR



Inverter

One-to-one mapping

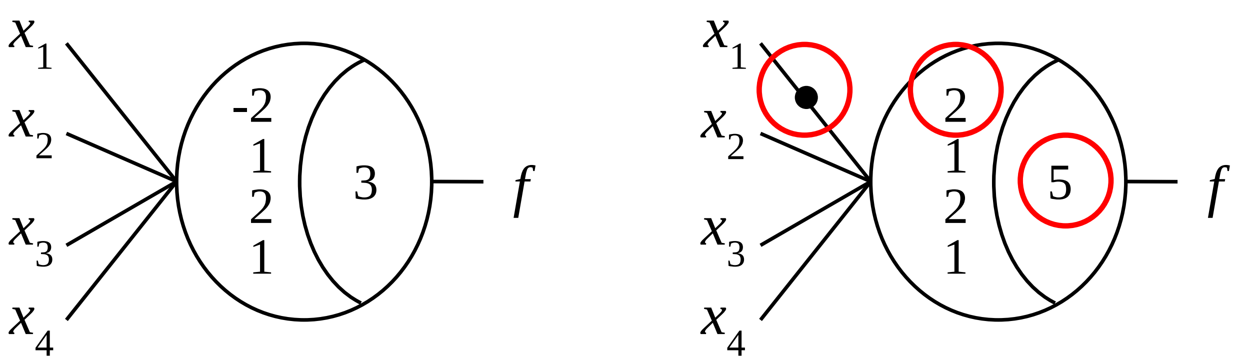
- Thus, a Boolean logic network composed of only primitive logic gates can be FAST transformed into a TLN by one-to-one mapping
  - Each logic gate is mapped to a threshold gate



- Actually, this is the first step of our approach

Positive weight transformation

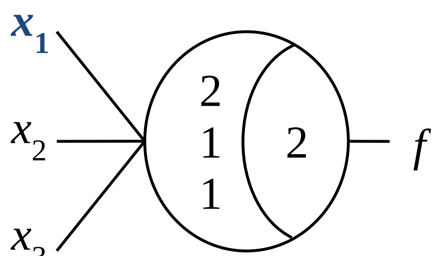
- In a threshold gate, a weight could be a positive or negative number
- For easy to manipulate a threshold gate, the negative weights can be transformed into positive weights



- We also perform this transformation to avoid negative weights during synthesis process

Controlling-1 and -0 inputs

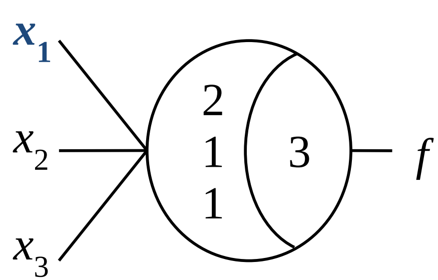
- *Controlling*-1 *input* of a threshold gate *g*
  - An input which can determine the output value of *g* to **1** regardless of the other inputs



$$w_i \geq T$$

x_1=1 implies f=1

- *Controlling*-0 *input* of a threshold gate *g*
  - An input which can determine the output value of *g* to **0** regardless of the other inputs



$$\sum_{i=1}^{n} w_i - w_i < T$$

x_1=0 implies f=0

Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion

Flowchart of the proposed method

Boolean logic network

⇩

Network transformation by one-to-one mapping

⇩

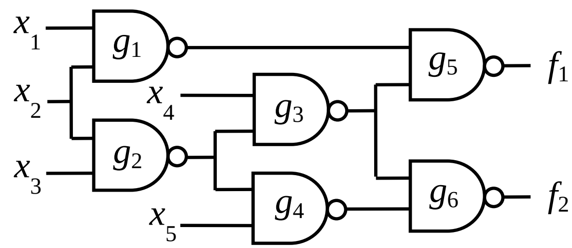
TLN

⇩

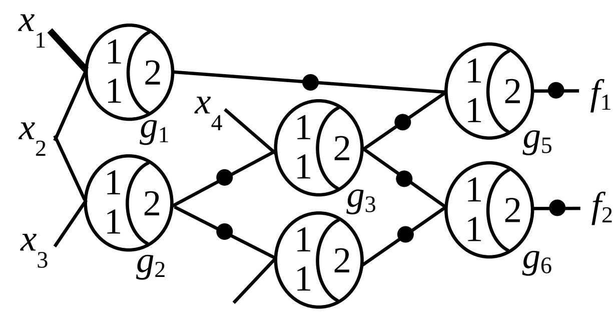
TLN optimization

⇩

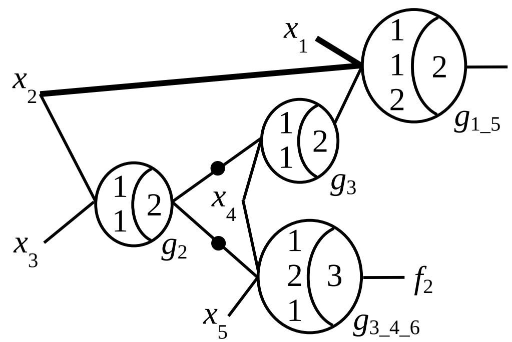
Opt. TLN



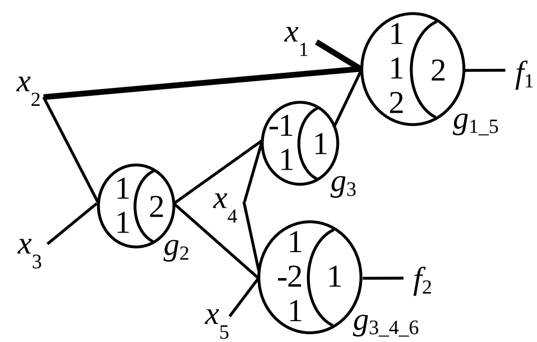
one-to-one mapping



optimization with predefined transformations



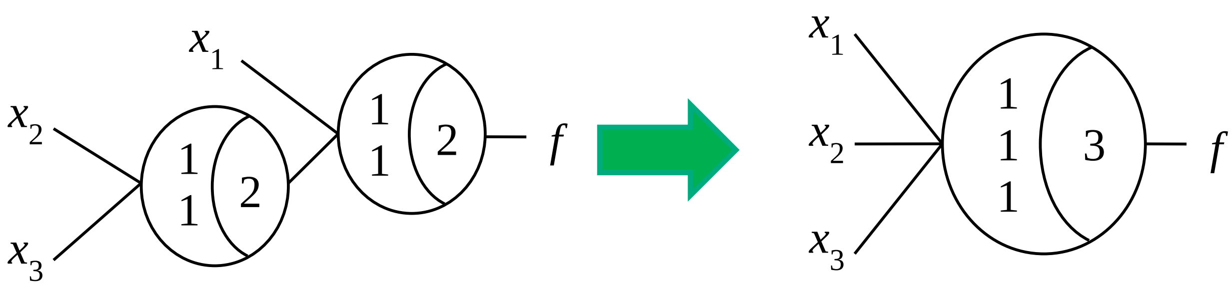
Inverter elimination with reverse positive weight transformation



TLN optimization with predefined transformations

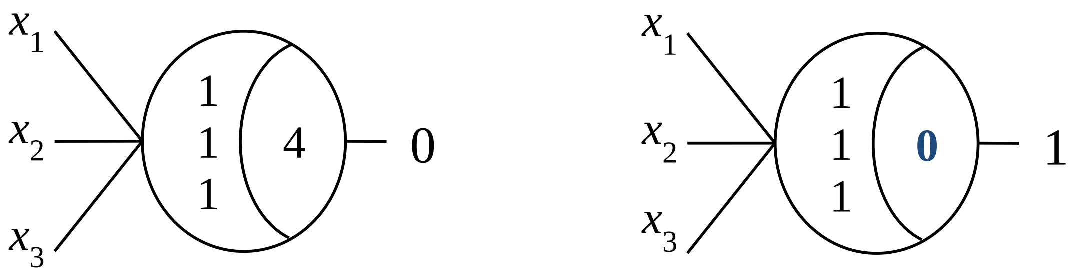
- Eight transformations for threshold logic
  - Sufficient conditions for eliminating a gate or merging two adjacent gates (i.e., one gate and one of its fanin gates)
  - Work only for threshold gates with only positive weights

- Simple example of merging two adjacent gates



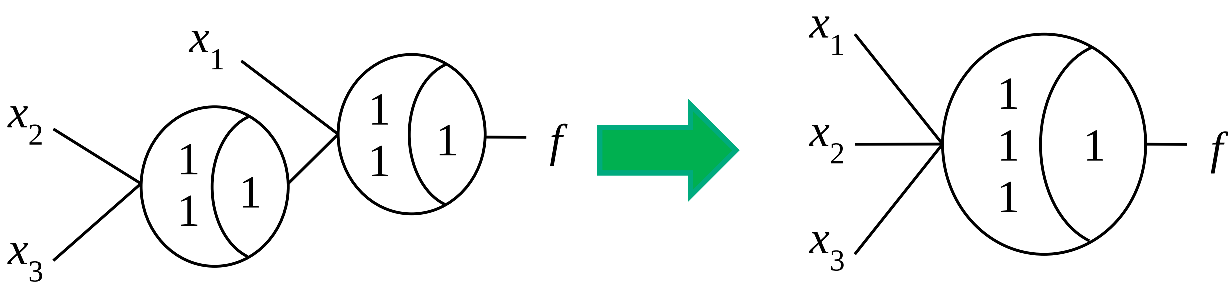
Transformations 1 & 2

- T1: Constant gate elimination



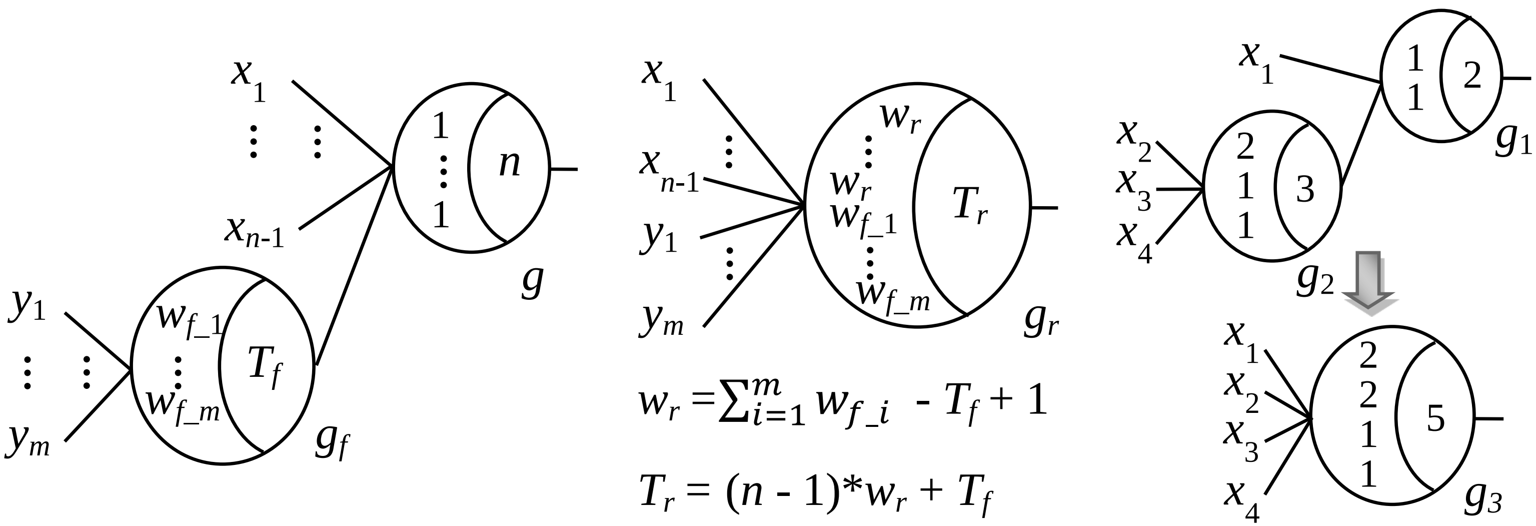
$\sum_{i=1}^{n} w_i < T$ $\qquad$ $T \leq 0$

- T2: Adjacent AND or OR gate merging



Transformation 3

- T3: AND gate-based merging (adapted from [9])
  - An AND gate can be merged with one of its fanin gates



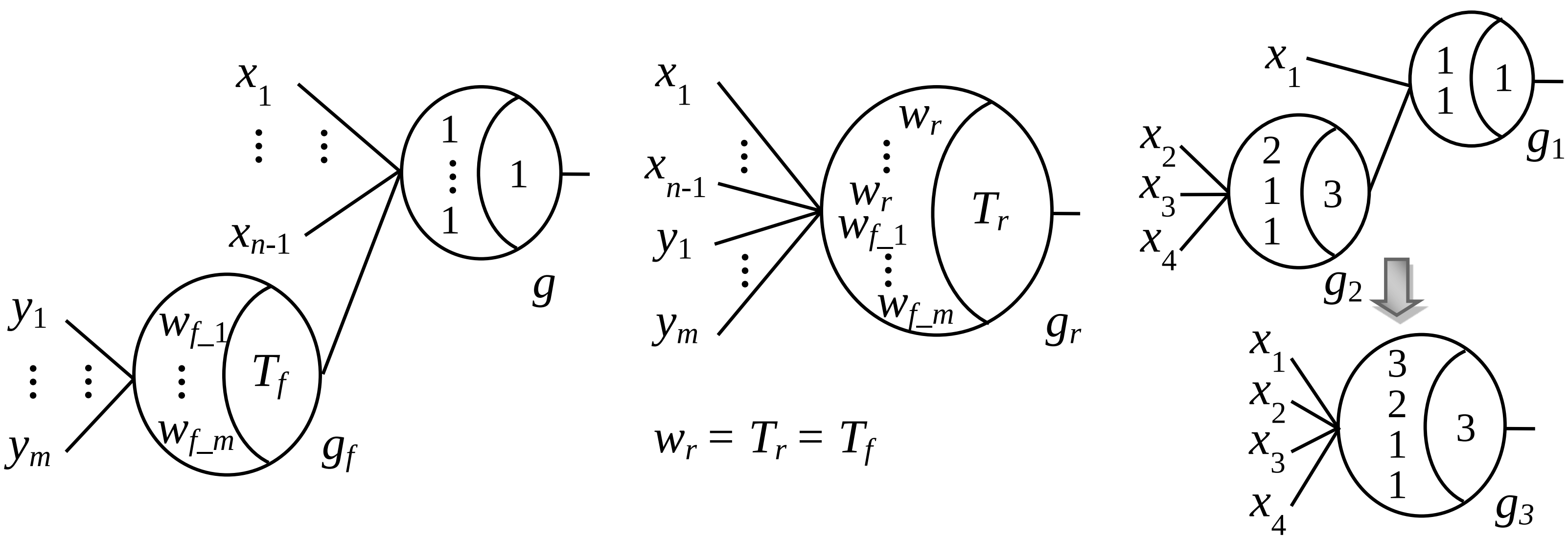
$$w_r = \sum_{i=1}^{m} w_{f\_i} - T_f + 1$$

$$T_r = (n - 1) * w_r + T_f$$

[9] S. Muroga, "*Threshold logic and its applications*," New York, NY: John Wiley, 1971.

Transformation 4

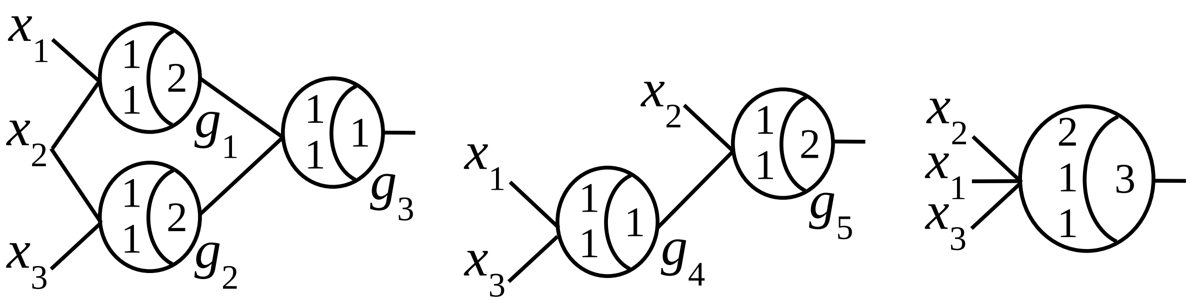
- T4: OR gate-based merging (adapted from [18])
  - An OR gate can be merged with one of its fanin gates



[18] R. Zhang *et al.*, “Threshold network synthesis and optimization and its application to nanotechnologies,” *IEEE Trans. Computer-Aided Design*, vol. 24, pp. 107-118, Jan. 2005.

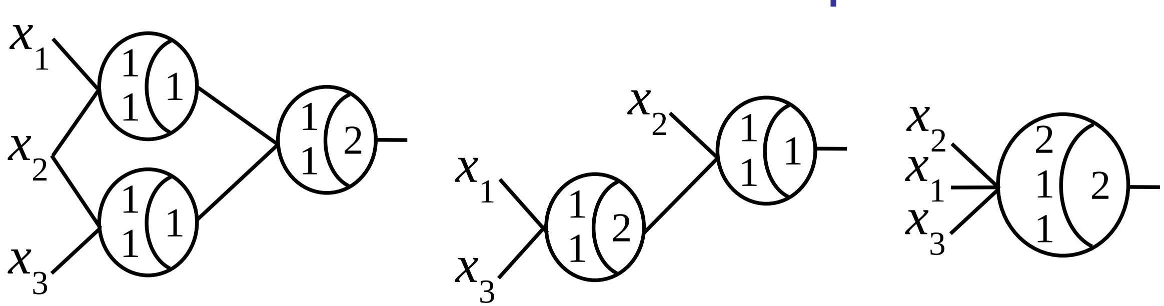
Transformations 5 & 6

- T5: Sum-of-product form to product-of-sum form conversion



$x_1 x_2 + x_2 x_3$ $x_2(x_1+x_3)$ T3: AND gate-based merging

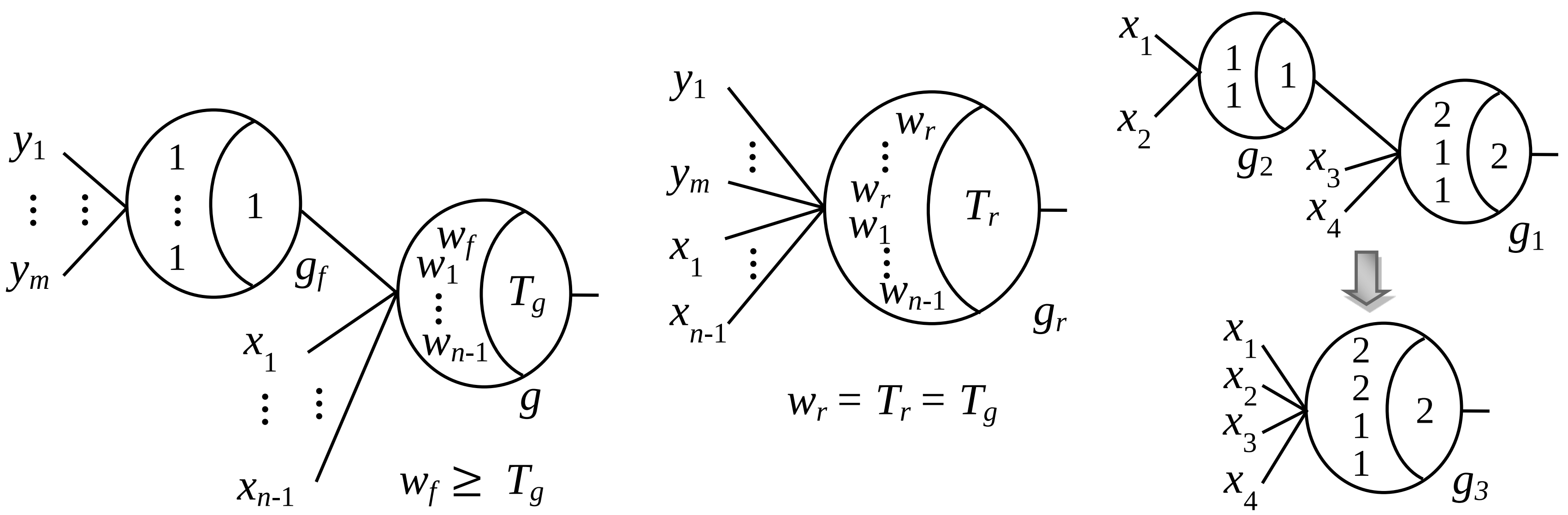
- T6: Product-of-sum form to sum-of-product form conversion



$(x_1+x_2)(x_1+x_3)$ $x_2+x_1x_2$ T4: gate-based merging

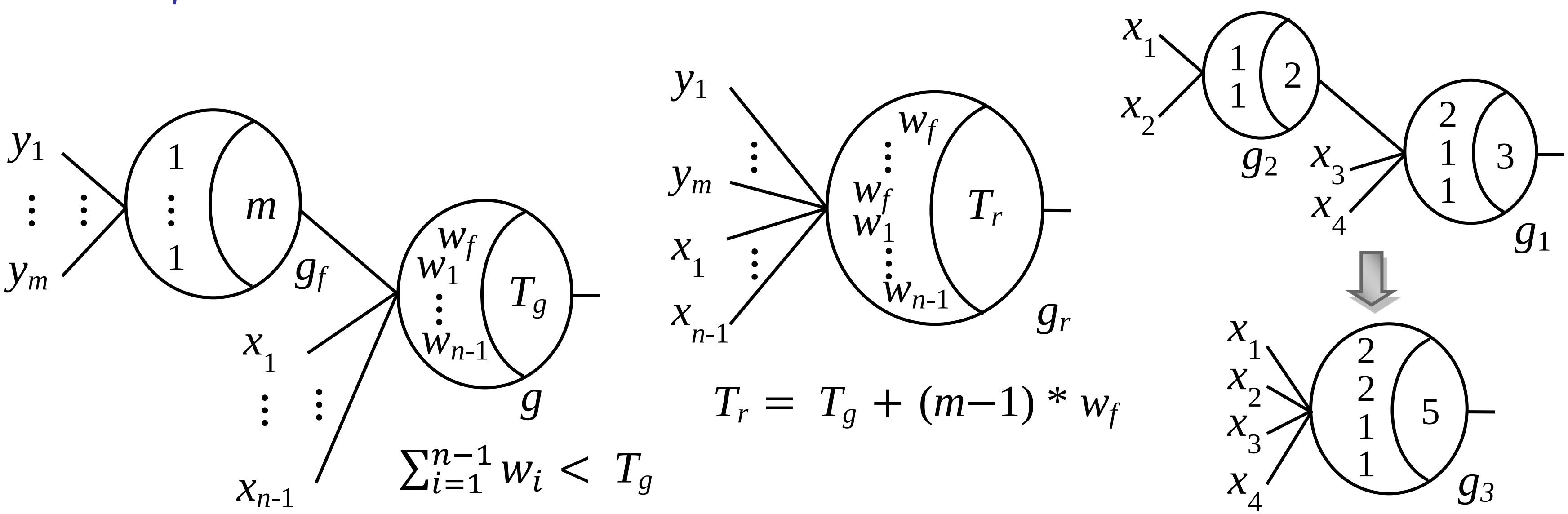
Transformation 7

- T7: Controlling-1 input-based merging
  - g_f is a controlling-1 input of g and
  - g_f is an OR gate



Transformation 8

- T8: Controlling-0 input-based merging
  - g_f is a controlling-0 input of g and
  - g_f is an AND gate



Overall flow of TLN optimization

- The are three iterations and each iteration targets certain types of transformations
  - First iteration
    - T2
  - Second iteration
    - T5 and T6
  - Third iteration
    - T3, T4, T7, T8, and T1
- At each iteration, each gate is selected as a target gate one at a time in the topological order, and we check and perform the transformation under consideration to the target gate if applicable

Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion

Experimental setup

- C language within the ABC [2] environment
- Linux platform with two 1.90GHz CPUs and 32GB memory
- Benchmarks
  - IWLS 2005 benchmark suite
  - And-Inverter Graph format
- Comparison
  - ILP-based method [18] + *lp_solve*

[2] Berkeley Logic Synthesis and Verification Group, “ABC: A System for Sequential Synthesis and Verification,” http://www.eecs.berkeley.edu/~alanmi/abc/

[18] R. Zhang *et al.*, “Threshold network synthesis and optimization and its application to nanotechnologies,” *IEEE Trans. Computer-Aided Design*, vol. 24, pp. 107-118, Jan. 2005.

Experimental results

- For fair comparison
  - Fanin count constraint
    - 6
- Save an average of 28% threshold gates
- Much more efficient

benchmark	|N|	ILP-based method		Our method		
		|N|	T(s)	|N|	*ratio*	T(s)
pci_conf.	84	91	2.2	62	0.68	0.0
stepper.	157	124	3.1	83	0.67	0.0
ss_pcm	172	173	4.4	135	0.78	0.0
usb_phy	357	287	7.2	221	0.77	0.0
sasc	563	461	12.5	333	0.72	0.0
simple_spi	775	597	16.1	436	0.73	0.0
pci_spoci.	878	559	15.6	399	0.71	0.0
i2c	941	659	18.1	482	0.73	0.0
systemcdes	2641	2018	57.7	1377	0.68	0.0
spi	3429	2421	75.6	1614	0.67	0.0
des_area	4410	2774	94.4	2011	0.72	0.0
tv80	7233	4996	191.1	3559	0.71	0.1
mem_ctrl	8815	6573	267.6	4721	0.72	0.1
systemcaes	10585	7677	334.4	5333	0.69	0.1
ac97_ctrl	10395	8326	330.0	6194	0.74	0.1
usb_funct	13320	9860	468.6	6842	0.69	0.1
pci_bridge32	17814	13595	769.9	10496	0.77	0.2
aes_core	20509	14163	761.2	10057	0.71	0.2
wb_conmax	41070	28518	2148.3	21956	0.77	0.3
ethernet	57205	47004	4978.9	35243	0.75	0.6
des_perf	71327	59886	7210.0	42719	0.71	0.8
vga_lcd	88854	74095	10918.6	55402	0.75	0.9
average					0.72	
total			28685.6			3.4

Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion

Conclusion

- We proposed a simple and fast approach for TLN synthesis and optimization
  - Much more efficient and effective than an ILP-based method

- Future work
  - Apply this compact logic representation to enhance conventional logic optimization and design verification

Thank you for attention