Optimization of Reversible Logic Networks with Gate Sharing

Yung-Chih Chen *and Feng-Jie Chao*

National Taiwan Univ. of Science and Tech., Taiwan

臺科大

TAIWAN TECH

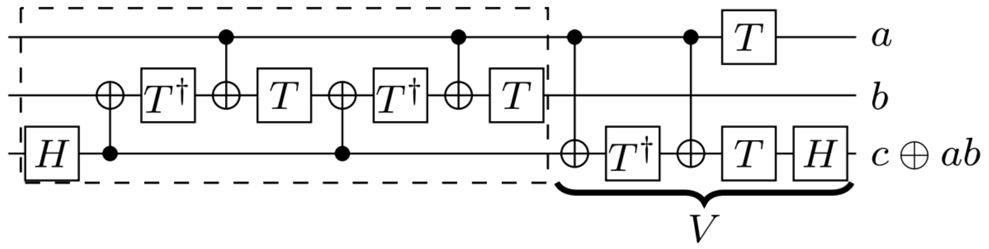
Outline

Introduction

- Logic synthesis for quantum computing is a process to map a Boolean function or network into a quantum circuit

$f = ab \oplus c \longrightarrow$

Quantum circuit [22]

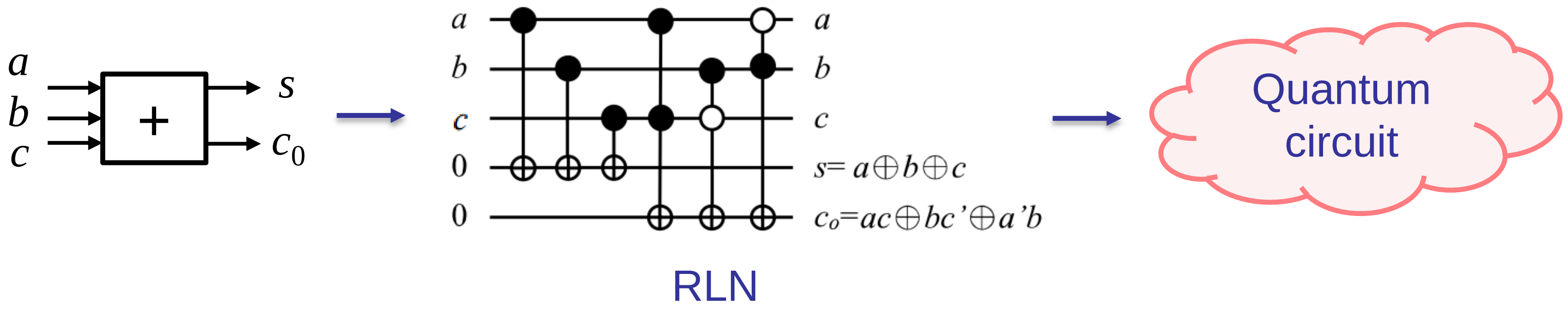


- Quantum gates: reversible and operate on qubits (e.g, 3 qubits)
  - Limited resources
- Program to be executed on quantum computers

[22] M. Soeken et al., LUT-based hierarchical reversible logic synthesis. *TCAD*, 2019.

Typical Synthesis Flow

- Two stages
  - First, map the given Boolean function into a reversible logic network (RLN)
  - Then, map each reversible gate into quantum gates



- Synthesis quality is mainly affected by the RLN
  - Many of the existing synthesis methods are dedicated to the first stage

RLN Synthesis (cont'd)

- State-of-the-art hierarchical method: lookup-table (LUT)-based
  - Take advantage of the LUT-based mapping technology for FPGAs
  - Synthesize the given Boolean network to a *k*-LUT network, and then map each LUT node to reversible gates to generate an RLN
  - Scalable, and flexible for trading off qubit count and quantum cost
  - **However, neglect that reversible gates from different LUT nodes can be shared**
    - **Quantum cost of the RLN could be further minimized**

[22] M. Soeken et al., LUT-based hierarchical reversible logic synthesis. *TCAD*, 2019.

Our Contributions

- Our objective is to optimize the RLNs generated by the LUT-based method

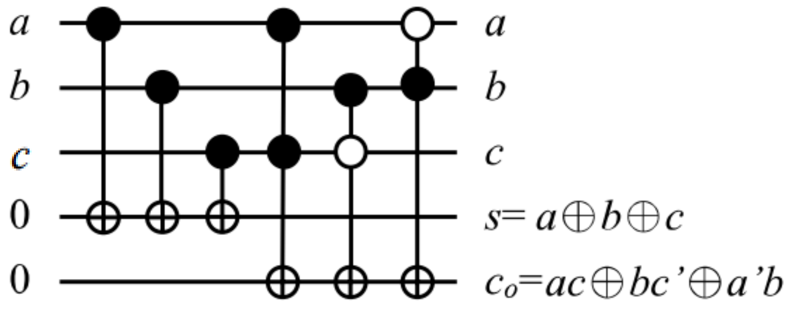
- We propose a method to extract the shareable gates to simplify the RLNs
  - We transform the extraction problem into an optimization problem of exclusive-sums-of-product (ESOP)
- Our method can reduce quantum cost without introducing extra qubits

Outline

RLN

- Reversible function: one-to-one and onto function, i.e., bijective
- An RLN realizes a reversible function
  - A number of lines
  - A cascade of reversible gates operating on the lines

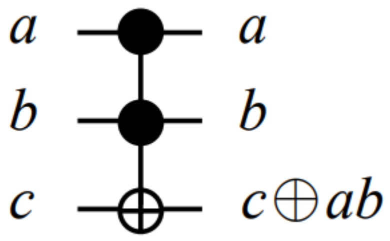
Input terminal



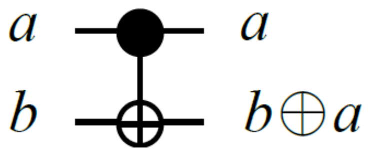
Output terminal

Multiple-Controlled Toffoli Gate

- The multiple-controlled Toffoli gate is a widely used reversible gate
  - Operate on n+1 lines
    - n lines pass through the gate unmodified, called *control lines*
    - 1 (rest) line is XORed with the conjunction of the values of the control lines, called *target line*



Two-controlled Toffoli gate



CNOT gate
Only one control line

The RLN under consideration is composed of only multiple-controlled Toffoli gates

Quantum Cost

- Synthesizing an RLN into a quantum circuit can be achieved by mapping each multiple-controlled Toffoli gate into quantum gates
  - Clifford+T quantum gate library [5]
  - T gate is sufficiently expensive
    - Reasonable to consider only the *T* gate when costing a quantum circuit

Toffoli gate	Required *T* gates [9]
2-controlled	7
k-controlled, *k* > 2	8*k* – 8*

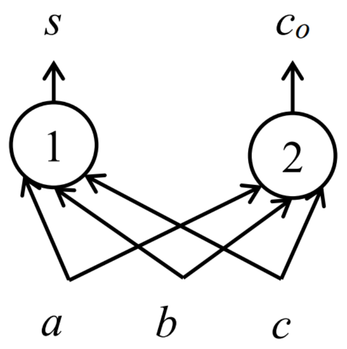
*Require more than $\left\lceil \frac{k-2}{2} \right\rceil$ free qubits

RLN optimization metric: T gate count

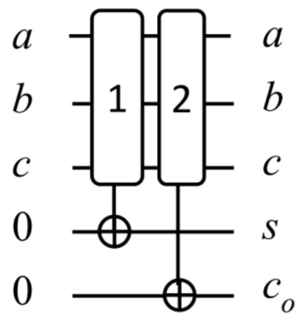
[5] P. O. Boykin et al., A new universal and fault-tolerant quantum basis. *Inform. Process. Lett.,* 2000.
[9] Maslov, Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. *ArXiv 1508.03273,* 2015.

LUT-based Hierarchical Reversible Logic Synthesis

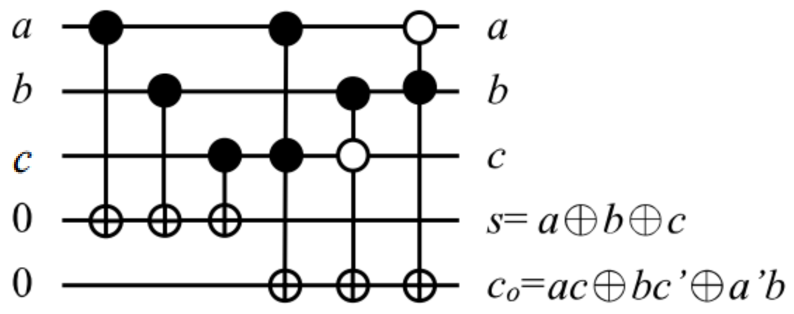
- First, synthesize the given Boolean network to an LUT network
- Then, transform each LUT node into one or two reversible single-target gates (STGs)
- Finally, map each STG into a cascade of multiple-controlled Toffoli gates, which have the same target line



LUT network



STG RLN



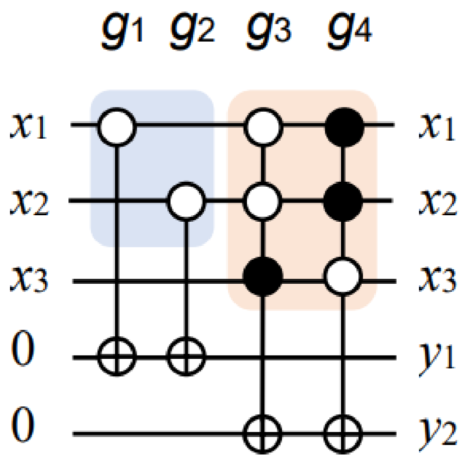
RLN

[22] M. Soeken et al., LUT-based hierarchical reversible logic synthesis. *TCAD*, 2019.

Outline

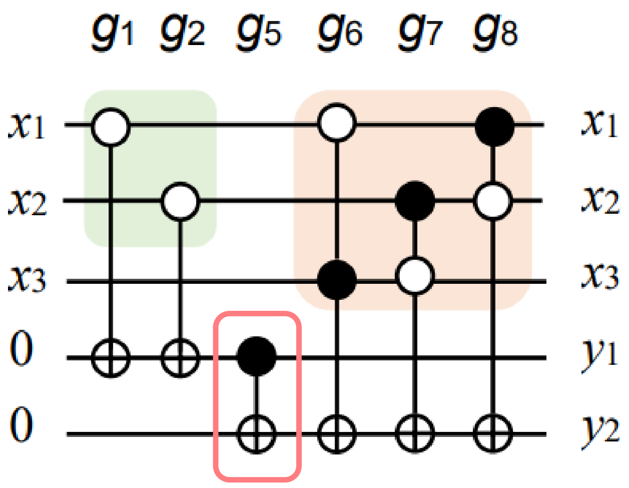
Motivation Example

Original RLN



- Two STGs
- Each STG is well optimized
- # of T gates: 32

Optimized RLN



- g1 and g2 are extracted and shared by the two STGs
- # of T gates: 21

The LUT-based method does not consider the opportunity of sharing gates among different LUT nodes

Problem Formulation

- **Input**: an RLN generated by the LUT-based method
- **Output**: an optimized RLN with minimized T gate count

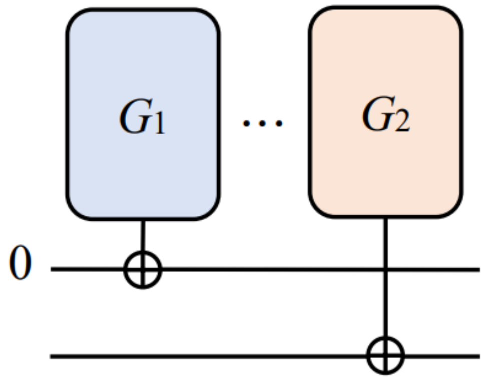
- **Goal**: Extract and share gates among different STGs to minimize T gate count without introducing extra qubits

Outline

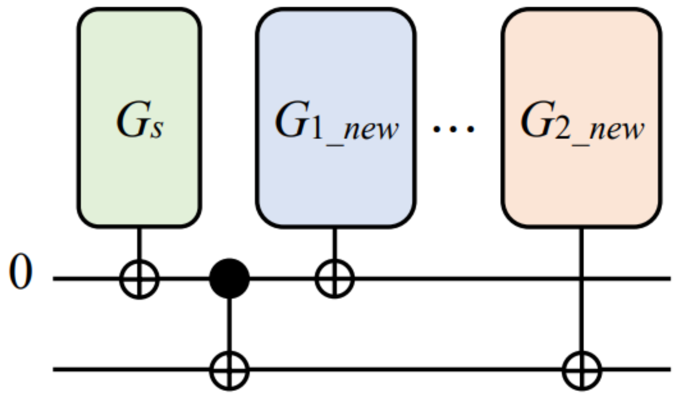
- Introduction
- Background
- Motivation example
- Reversible logic network optimization
- Experimental results
- Conclusion

Our Intention

- Main flow is to iteratively select two STGs and extract shareable multiple-controlled Toffoli gates from them



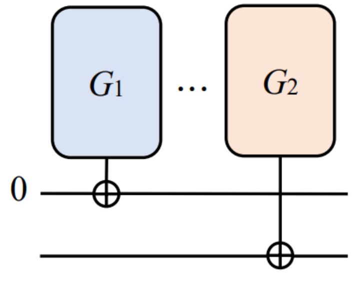
Two STGs, G_1 and G_2
Not necessary to be adjacent

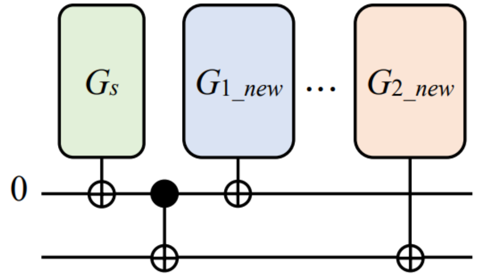


Extract $\boldsymbol{G_s}$ from G_1 and G_2,
and simplify them to $\boldsymbol{G_{1\_new}}$ and $\boldsymbol{G_{2\_new}}$

Problems

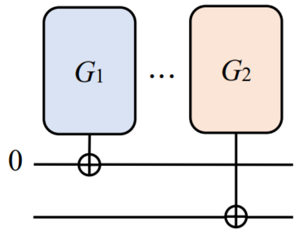
- To extract shareable gates, we need to solve two problems
  1) Given G_1 and G_2, is it valid to extract shareable gates (if any) from them?
     - Not all pairs of STGs having common sub-functions can share gates without altering the overall functionality
     - **Validity problem**
  2) How to find the shareable gate from G_1 and G_2, i.e., to compute G_s, G_{1_new} and G_{2_new}?
     - **Extraction problem**

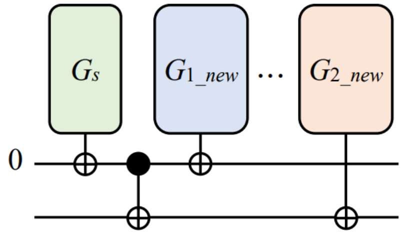




Validity Problem

- G_1 and G_2 are valid for extracting shareable gates if they satisfy the following **three conditions**
  - The target line of G_1 has an initial value 0
    - To ensure that the extracted G_s implements a desired function
  - The target line of G_2 is not a control line of G_1 and the STGs in between G_1 and G_2
    - To ensure that the control line is not altered by the newly added CNOT gate
  - A control line of G_2 and G_1 is not the target line of G_1 and not the target line of a STG in between G_1 and G_2
    - To ensure that G_s and G_{2_new} implement desired functions





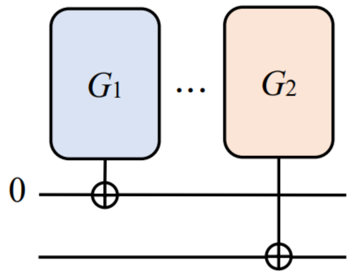
Extraction Problem

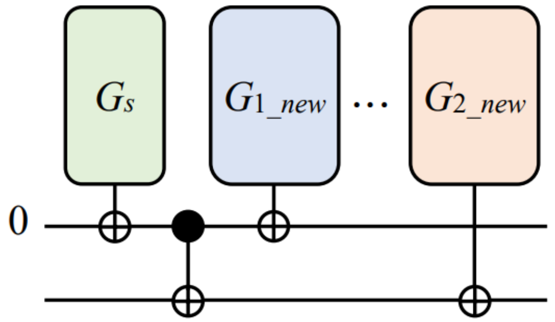
- We simplify the problem of computing G_s, G_{1_new} and G_{2_new} to that of computing G_{2_new} only
- Two types of extractions
  - Mainly different in the methods of finding G_s

Type 1 Extraction

- G_s: gates in G_1 that have common control lines with G_2
- G_{1_new}: other gates in G_1
  - G_1 is partitioned into G_s and G_{1_new}
- G_{2_new}: G_2 and G_s
  - Exclusive-sums-of-product (ESOP) optimization

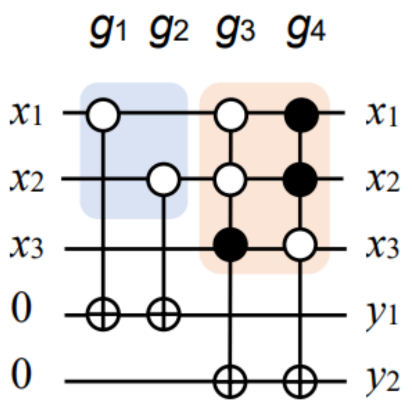
- Let f_1, f_2, f_s, and f_{2_new} denote the functions of G_1, G_2, G_s, and G_{2_new}

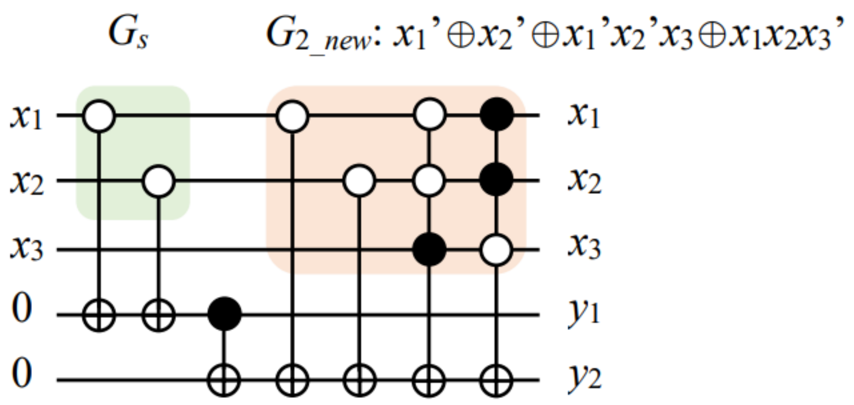




f_s
$f_{2\_new} = f_2 \oplus f_s$
G_s
G_{1_new}
…
G_{2_new}
0
f_1
$f_s \oplus f_{2\_new} = f_s \oplus f_2 \oplus f_s$
$= f_2$

Type 1 Extraction (cont'd)



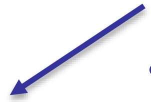


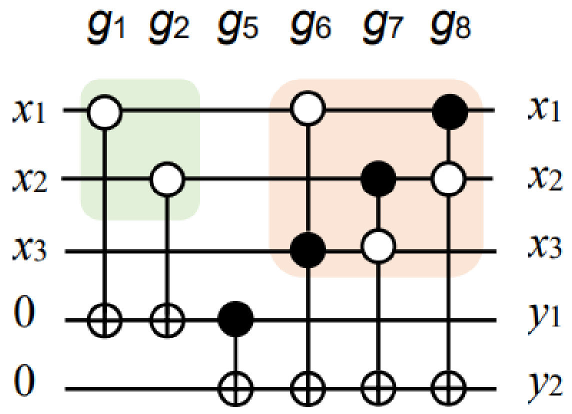
G_s
G_{2_new}: $x_1' \oplus x_2' \oplus x_1'x_2'x_3 \oplus x_1x_2x_3'$
x_1
x_2
x_3
0
0
x_1
x_2
x_3
y_1
y_2

G_1={g_1, g_2} and G_2={g_3, g_4}

G_s={g_1, g_2}, G_{1_new}=∅, and

G_{2_new}={g_1, g_2, g_3, g_4}





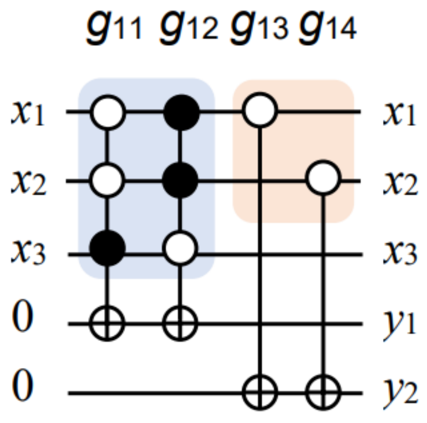
- Perform ESOP optimization to **G_{2_new}** with the ***exorcism*** technique [15]
- Accept the extraction if **simplified G_{2_new} requires fewer T gates than G_2**

Simplified G_{2_new}={g_6, g_7, g_8}

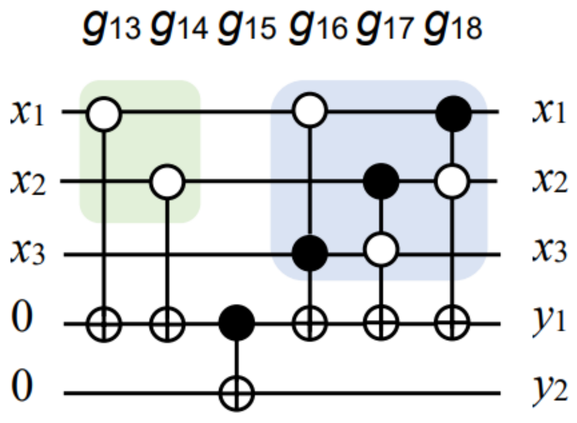
[15] A. Mishchenko and M. A. Perkowski. 2001. Fast heuristic minimization of exclusive-sums-of-products. *In Proc. Reed Muller Workshop.*

Type 2 Extraction

- In **Type 1 extraction**, if the **simplified ESOP expression** has a higher cost than G_2, but a lower cost than G_s, we then perform **Type 2 extraction** to simplify G_1 rather than G_2
- **Type 2 extraction**
  - G_s: G_2
  - G_{1_new}: other gates in G_1 and gates in **simplified ESOP expression**
  - G_{2_new}: ∅



G_1={g_{11}, g_{12}} and G_2={g_{13}, g_{14}}



Simplified ESOP expression: {g_{16}, g_{17}, g_{18}}

Overall Flow for RLN Optimization

- Given an RLN, we first identify STGs
- Then, we iteratively select two groups, G_1 and G_2, and check if they are valid
  - If yes, we compute the optimized ESOP expression and evaluate the cost
  - If the cost reduces, we apply **Type 1 Extraction** or **Type 2 Extraction** accordingly
    - Only one *exorcism* call is needed, because the two types of extractions use the same simplified ESOP expression

Outline

- Introduction
- Background
- Motivation example
- Reversible logic network optimization
- Experimental results
- Conclusion

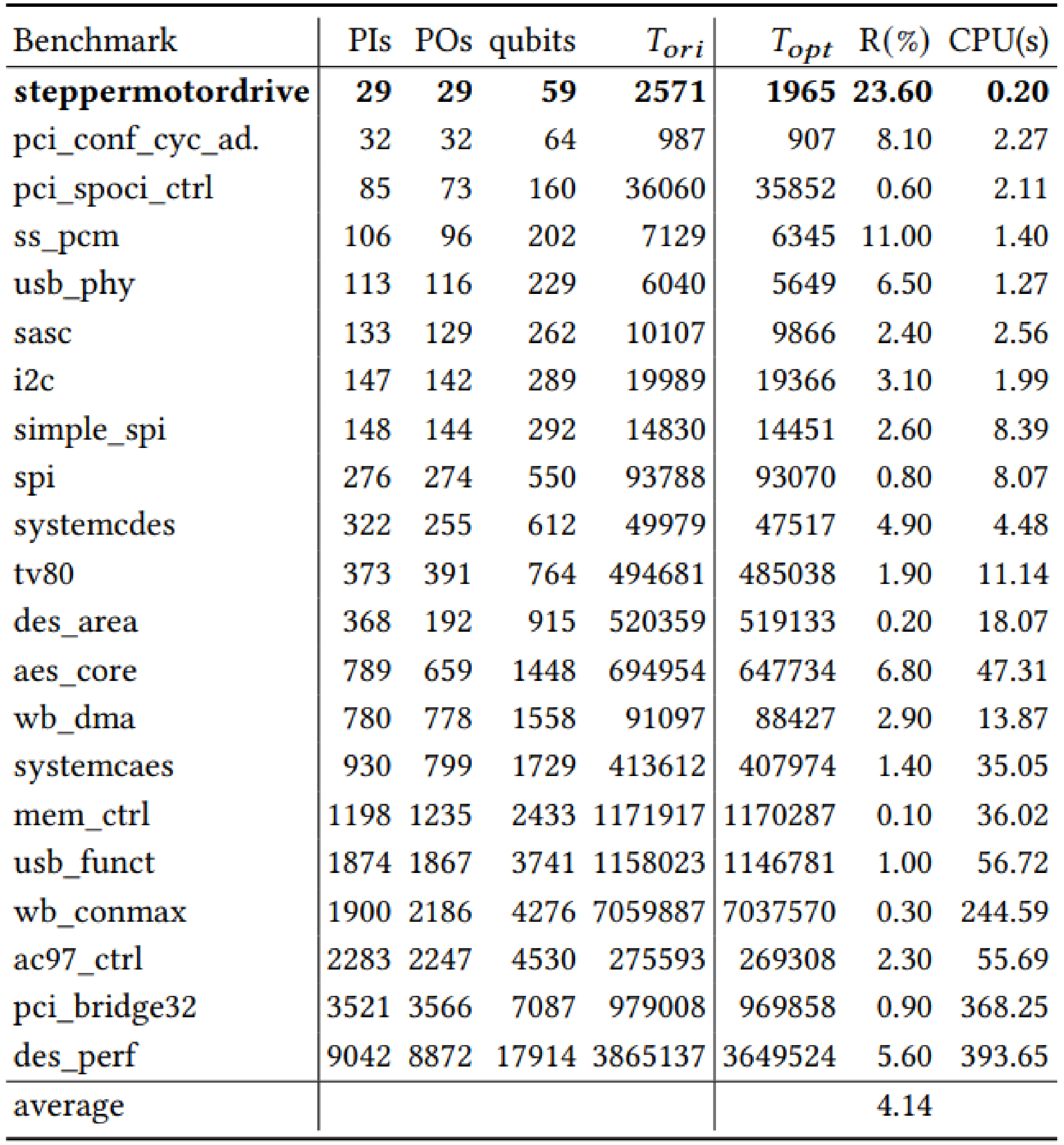
Experimental setup

- C language
- Linux workstation with an Intel Core i5 2.90GHz CPU and 32GB memory
- Benchmarks
  - IWLS 2005 benchmark suite
  - RLNs generated by [22]
- Verification
  - SAT-based combinational equivalence checker for RLNs [3]

[3] L. Amaru el al.. Exploiting inherent characteristics of reversible circuits for faster combinational equivalence checking. *In Proc. Design, Automation and Test in Europe Conf.*, 2016
[22] M. Soeken et al., LUT-based hierarchical reversible logic synthesis. *TCAD*, 2019.

Experimental results

- Achieve an average of 4.14% T gate reduction
- The best result is up to 23.60% for the *steppermotordrive* benchmark
  - Some pairs of STGs are identical
- Limited by the number of valid pairs of STGs and the optimization quality of *exorcism*



Benchmark	PIs	POs	qubits	T_{ori}	T_{opt}	R(%)	CPU(s)
steppermotordrive	**29**	**29**	**59**	**2571**	**1965**	**23.60**	**0.20**
pci_conf_cyc_ad.	32	32	64	987	907	8.10	2.27
pci_spoci_ctrl	85	73	160	36060	35852	0.60	2.11
ss_pcm	106	96	202	7129	6345	11.00	1.40
usb_phy	113	116	229	6040	5649	6.50	1.27
sasc	133	129	262	10107	9866	2.40	2.56
i2c	147	142	289	19989	19366	3.10	1.99
simple_spi	148	144	292	14830	14451	2.60	8.39
spi	276	274	550	93788	93070	0.80	8.07
systemcdes	322	255	612	49979	47517	4.90	4.48
tv80	373	391	764	494681	485038	1.90	11.14
des_area	368	192	915	520359	519133	0.20	18.07
aes_core	789	659	1448	694954	647734	6.80	47.31
wb_dma	780	778	1558	91097	88427	2.90	13.87
systemcaes	930	799	1729	413612	407974	1.40	35.05
mem_ctrl	1198	1235	2433	1171917	1170287	0.10	36.02
usb_funct	1874	1867	3741	1158023	1146781	1.00	56.72
wb_conmax	1900	2186	4276	7059887	7037570	0.30	244.59
ac97_ctrl	2283	2247	4530	275593	269308	2.30	55.69
pci_bridge32	3521	3566	7087	979008	969858	0.90	368.25
des_perf	9042	8872	17914	3865137	3649524	5.60	393.65
average						4.14	

Outline

Conclusion

- We proposed a new method for RLN optimization by sharing multiple-controlled Toffoli gates
  - Reduce the implementation cost in terms of *T* gate count without increasing the qubits

- Future work
  - Study other transformations and ESOP optimization techniques

Thank you for your attention