Qauntum Ling adder
논문 리뷰

https://youtu.be/tpDg04WaIPo

Ling adder

Reducing Depth of Quantum Adder using Ling Structure

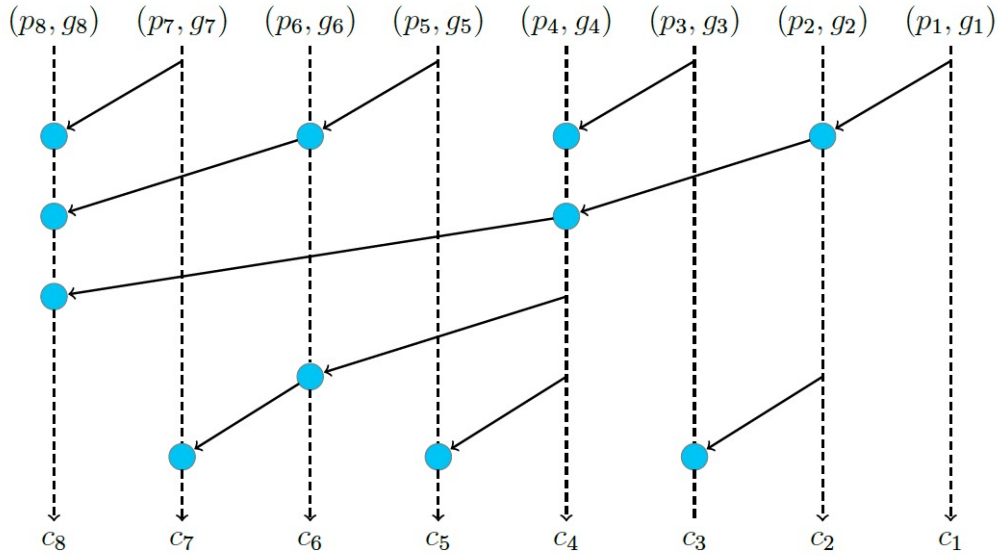
***Abstract*—Improving the performance of quantum adder is an important technical challenge with major impact on the implementation of efficient, large-scale quantum computing. Continuing along this research direction, we propose a novel parallel-prefix quantum adder based on Ling expansion. We systematically explored classical structures for parallel-prefix adders assessing their suitability to be realized in quantum domain. Furthermore, Ling adder enforces Logical OR and large fan-out, which require innovative solutions. We addressed these challenges to realize the quantum Ling adder, which results in a T-depth of only $O(\log \frac{n}{2})$. This represents a substantial improvement over the previous quantum adders based on parallel prefix structure, which require $O(\log n)$ T-depth. We present extensive theoretical and simulation-based studies to establish our claims.**

***Index Terms*—Quantum Computing, Quantum Arithmetic, T-depth Optimization, Carry-lookahead Adder, Ling Adder**

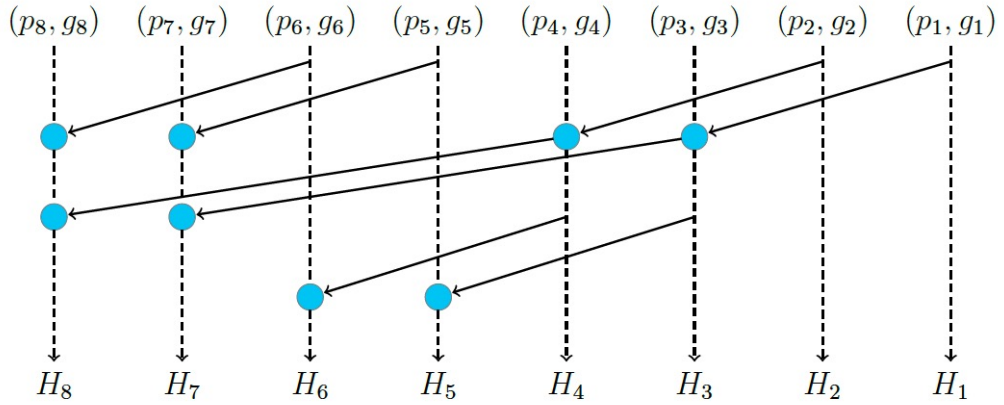
rectly translating Ling structure into quantum circuits. Hence, we modified the classical Ling structure, maximizing parallel structures while considering quantum circuit properties. Moreover, our building blocks are refined based on the parallel prefix adder design to address the strict requirements of OR logic in Ling structure.

Our main contribution is the exploration of the effectiveness and potential advantages of Ling structure in the quantum world, compared to existing adder designs. Through a comprehensive evaluation of the proposed design by using T-depth (TD), Qubit Count (QC), and T-count (TC), we provide several valuable insights into the strengths and limitations of quantum Ling adder, thereby contributing to the continuous progress of quantum computing field.

The rest of this paper is organized as follows: Section II



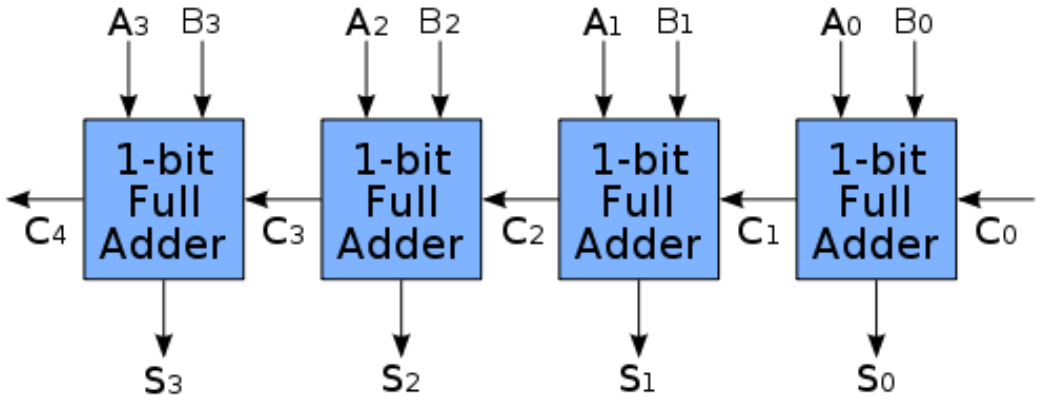
(a) Brent-Kung

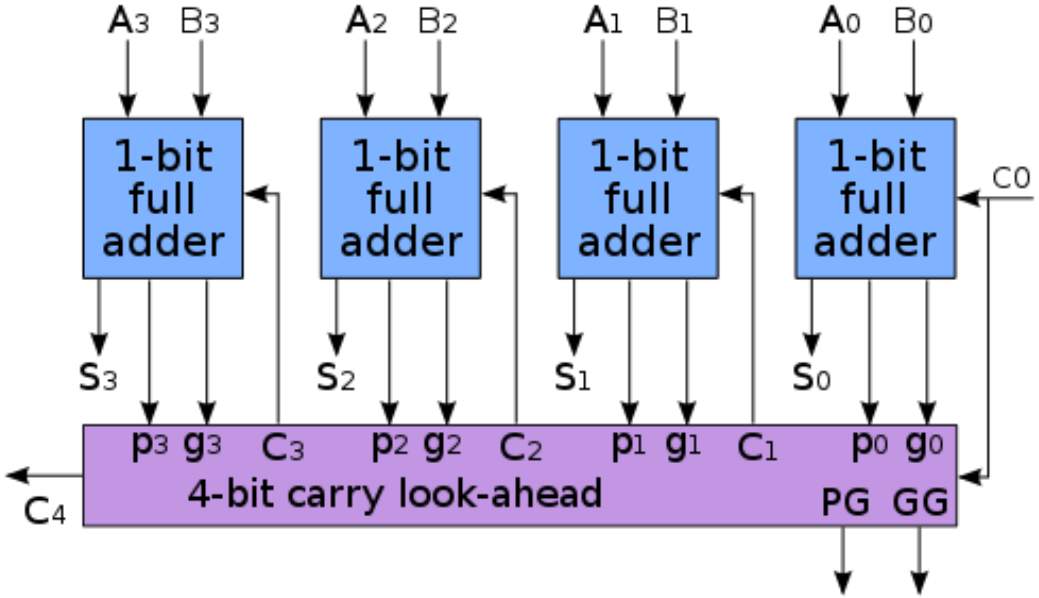


(b) Ling-based Brent-Kung

https://sites.google.com/view/vlsi-soc2023/submission?authuser=0

Carry look ahead adder





$$G(A,B) = A \cdot B$$

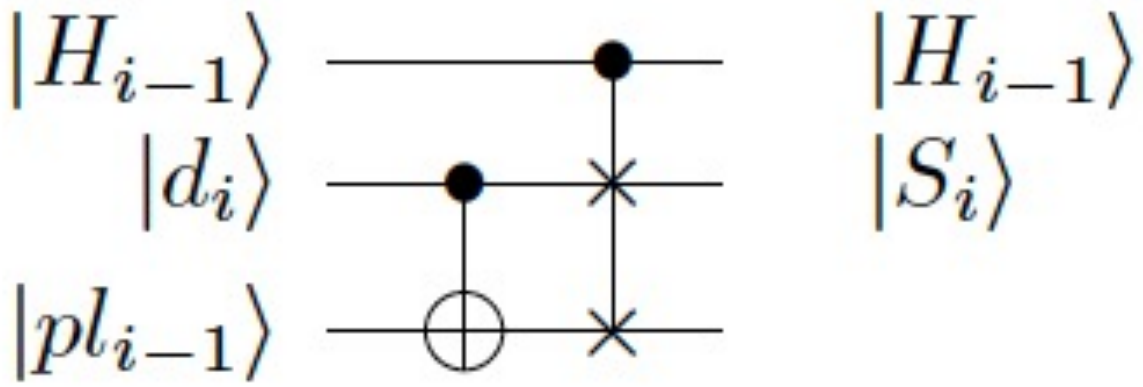
$$P(A,B) = A \oplus B$$

G는 "자리올림수 생성"이라고 불리며, 기존 연산과 관계 없이 '반드시' 자리올림수가 생성됨을 확인하는 출력 값이다.

P는 "자리올림수 전달"라고 불리며, 추가적으로 자리올림수가 발생할 가능성을 검사한다.

CSWAP gate (Fredkin gate)

$$s_i = \overline{H}_{i-1} \cdot d_i + H_{i-1} \cdot (d_i \oplus p_{i-1}),$$



4비트 양자 회로

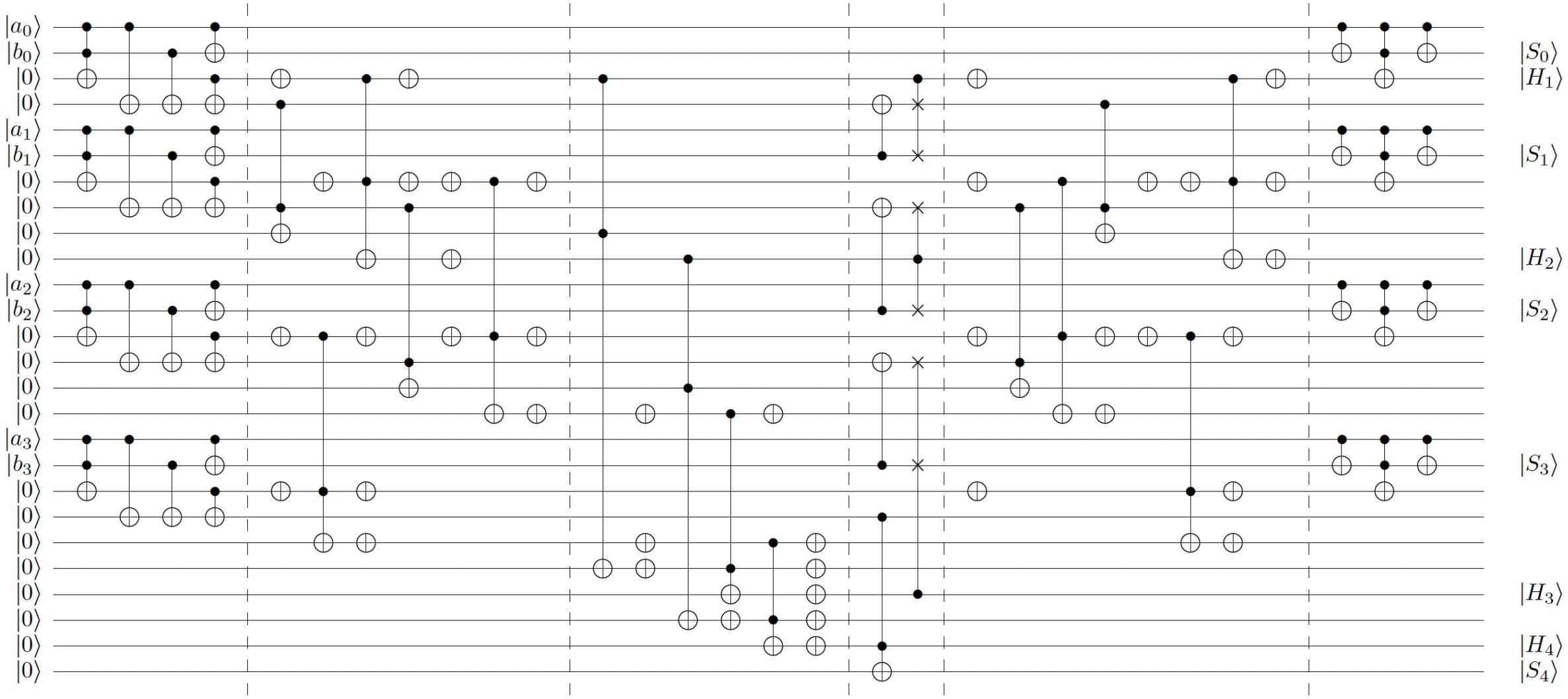
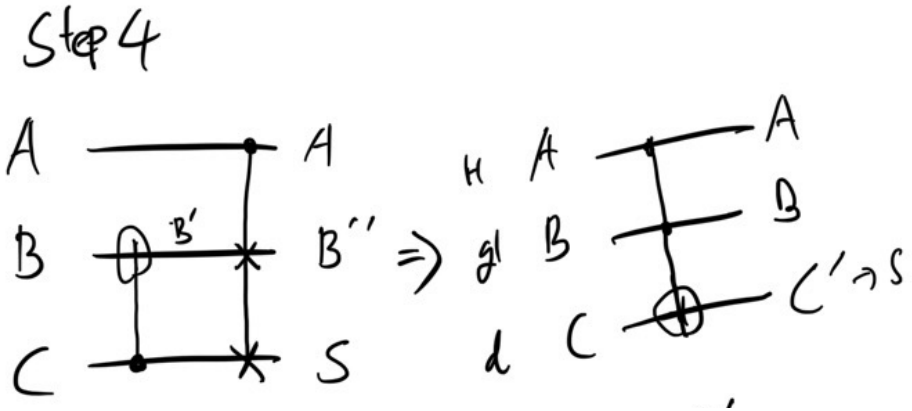


Fig. 10. 4-bit Quantum Ling Adder

SWAP gate → Toffoli gate



A	B	C	B′	B″	S	C′
0	0	0	0	0	0	0
0	0	1	1	1	1	1
0	1	0	1	1	0	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	1	0	1	1	0	0
1	1	1	0	0	1	1

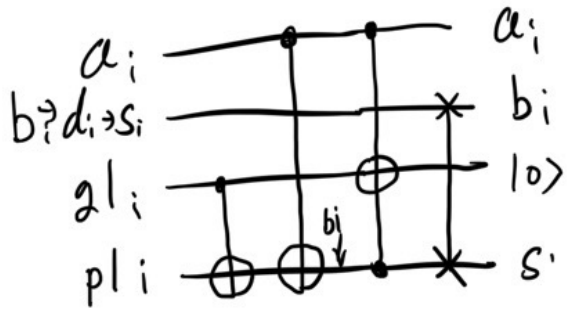
$$s_i = d_i \oplus c_{i-1} = d_i \oplus (p_{i-1} \cdot H_{i-1})$$

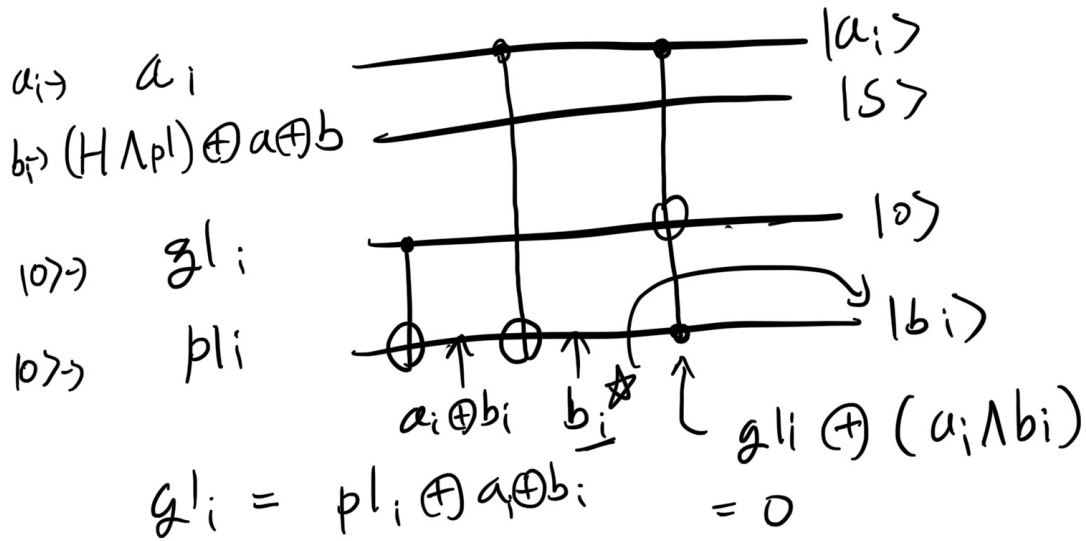
Uncomputation 회로 수정

step 5 = step 3

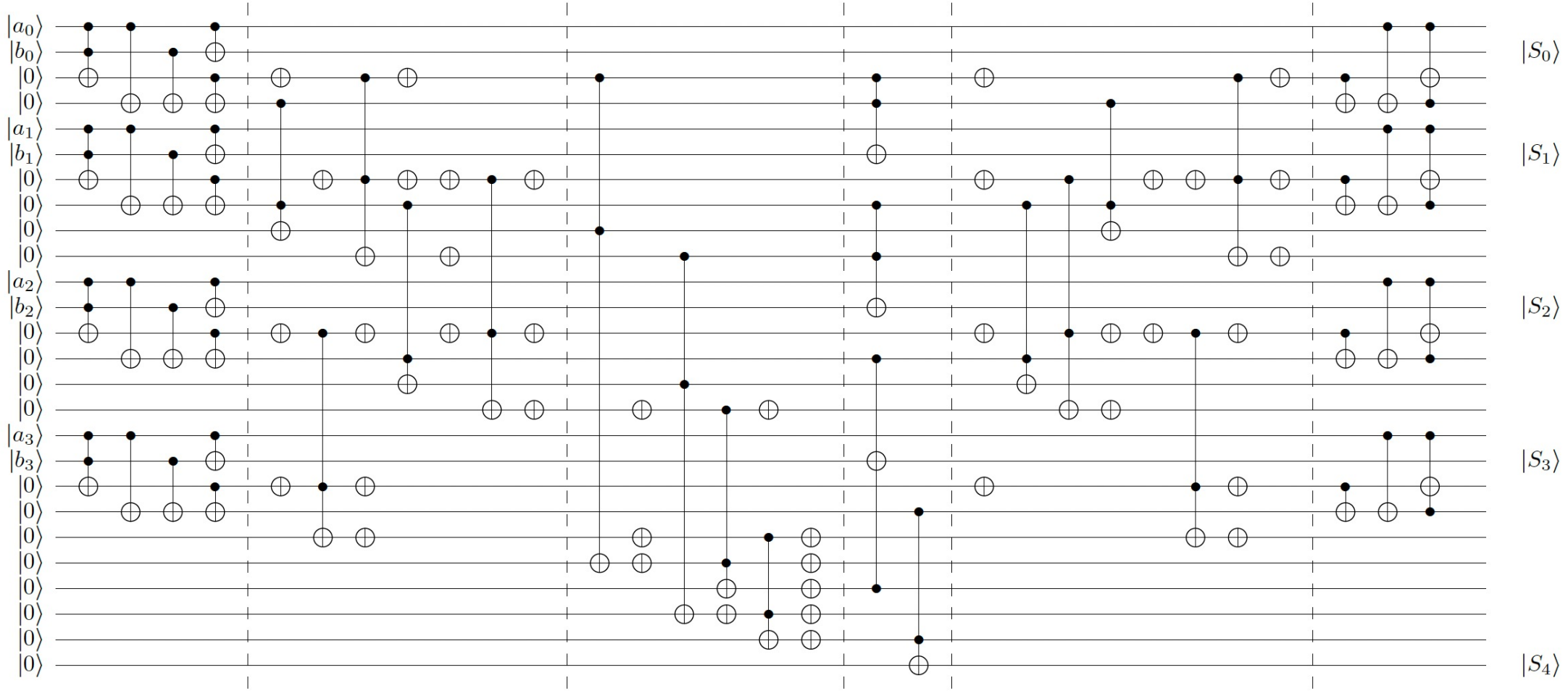
step 6 = step 2

Step 7



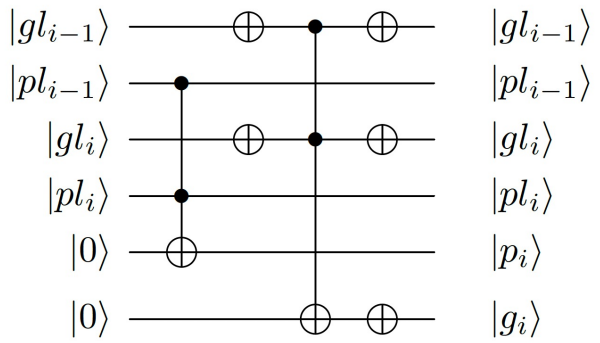


4-bit 양자 회로

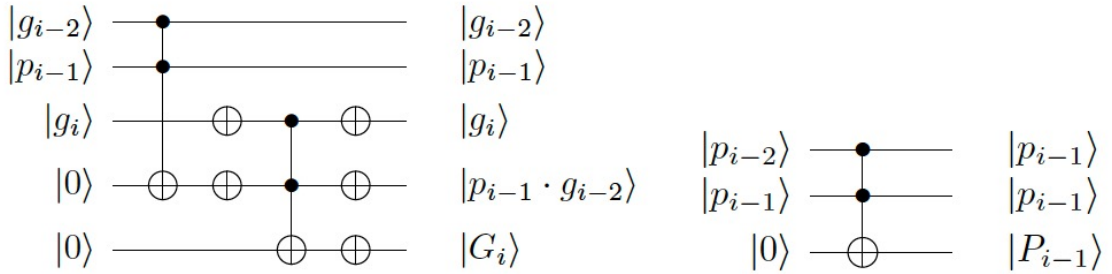

$|a_0\rangle$
$|b_0\rangle$
$|0\rangle$
$|0\rangle$
$|a_1\rangle$
$|b_1\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|a_2\rangle$
$|b_2\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|a_3\rangle$
$|b_3\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|S_0\rangle$
$|S_1\rangle$
$|S_2\rangle$
$|S_3\rangle$
$|S_4\rangle$

H 설명

$$pl_i = a_i + b_i$$
$$gl_i = a_i \cdot b_i$$
$$d_i = a_i \oplus b_i$$



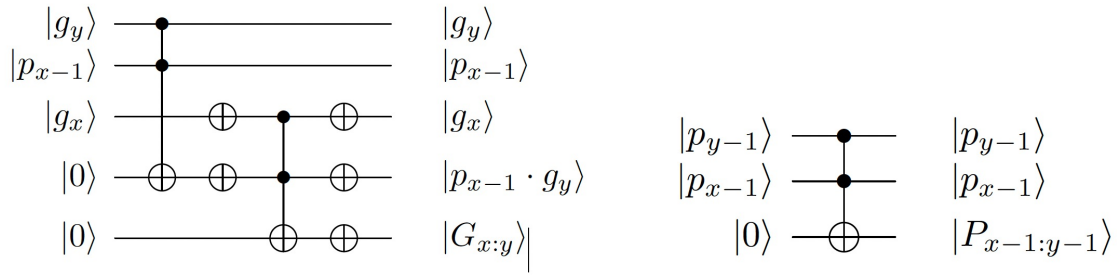
$$g_i = gl_i + gl_{i-1}$$
$$p_i = pl_i \cdot pl_{i-1}$$



(a) Propagation of G (b) Propagation of P

$$(g_x, p_x) \circ (g_y, p_y) = (g_x + p_x \cdot g_y, p_x \cdot p_y)$$
$$H_0 = g_0$$
$$H_i = G_i + P_{i-1} \cdot G_{i-2}$$

$$g_x, p_{x-1} \circ g_y, p_{y-1} = g_x + p_{x-1} \cdot g_y, p_{x-1} \cdot p_{y-1}$$
$$H_0 = g_0$$
$$H_i = G_i + P_{i-1} \cdot G_{i-2}$$



(a) Propagation of G (b) Propagation of P

Fig. 6. Ling Propagation

H 정의

$$G(A,B) = A \cdot B$$
$$P(A,B) = A \oplus B$$

$$s_i = d_i \oplus c_{i-1} = d_i \oplus (p_{i-1} \cdot H_{i-1}).$$

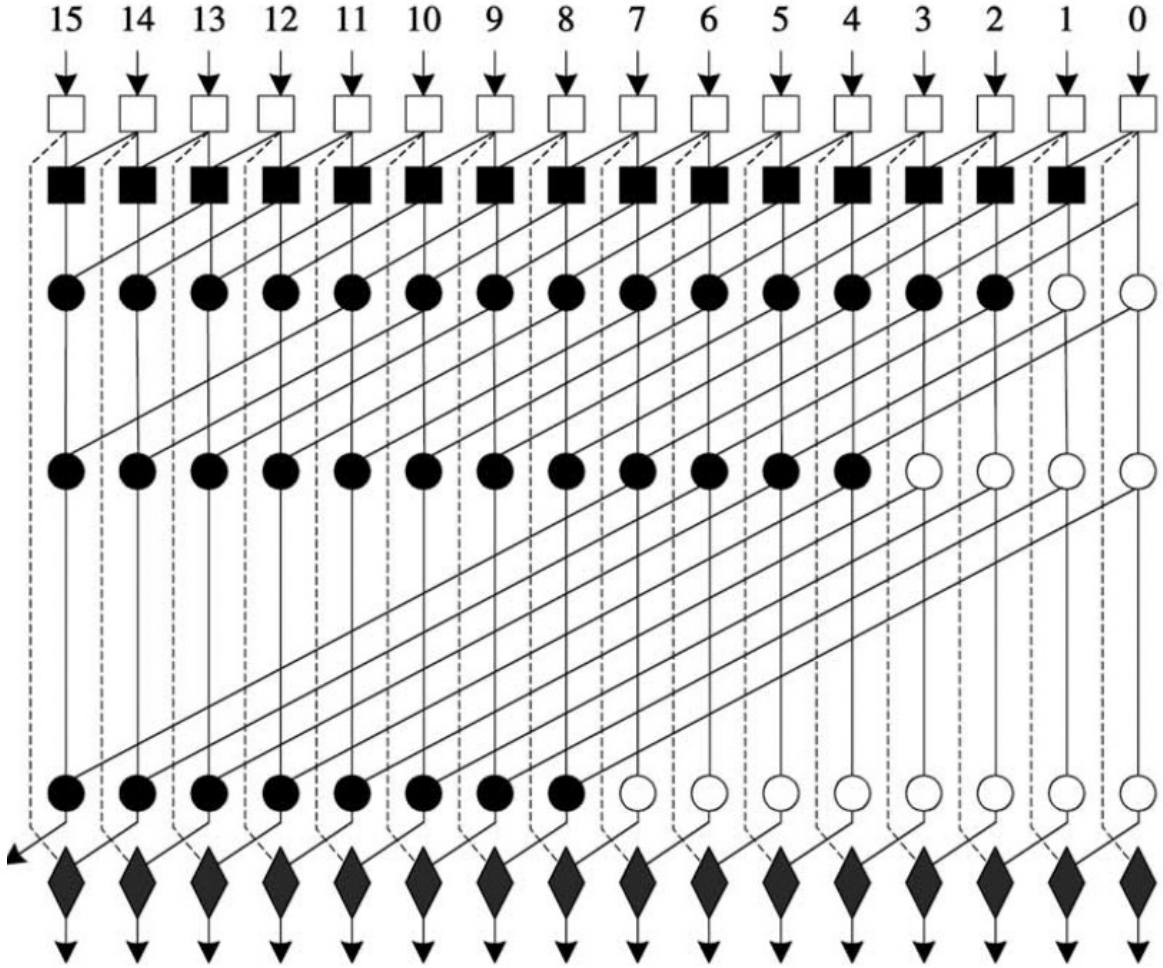
$$G_i^* = g_i + g_{i-1} \qquad \text{and} \qquad P_i^* = p_i \cdot p_{i-1},$$

$$H_i = (G_i^*, P_{i-1}^*) \circ (G_{i-2}^*, P_{i-3}^*) \circ \ldots \circ (G_0^*, P_{-1}^*) \qquad (12)$$

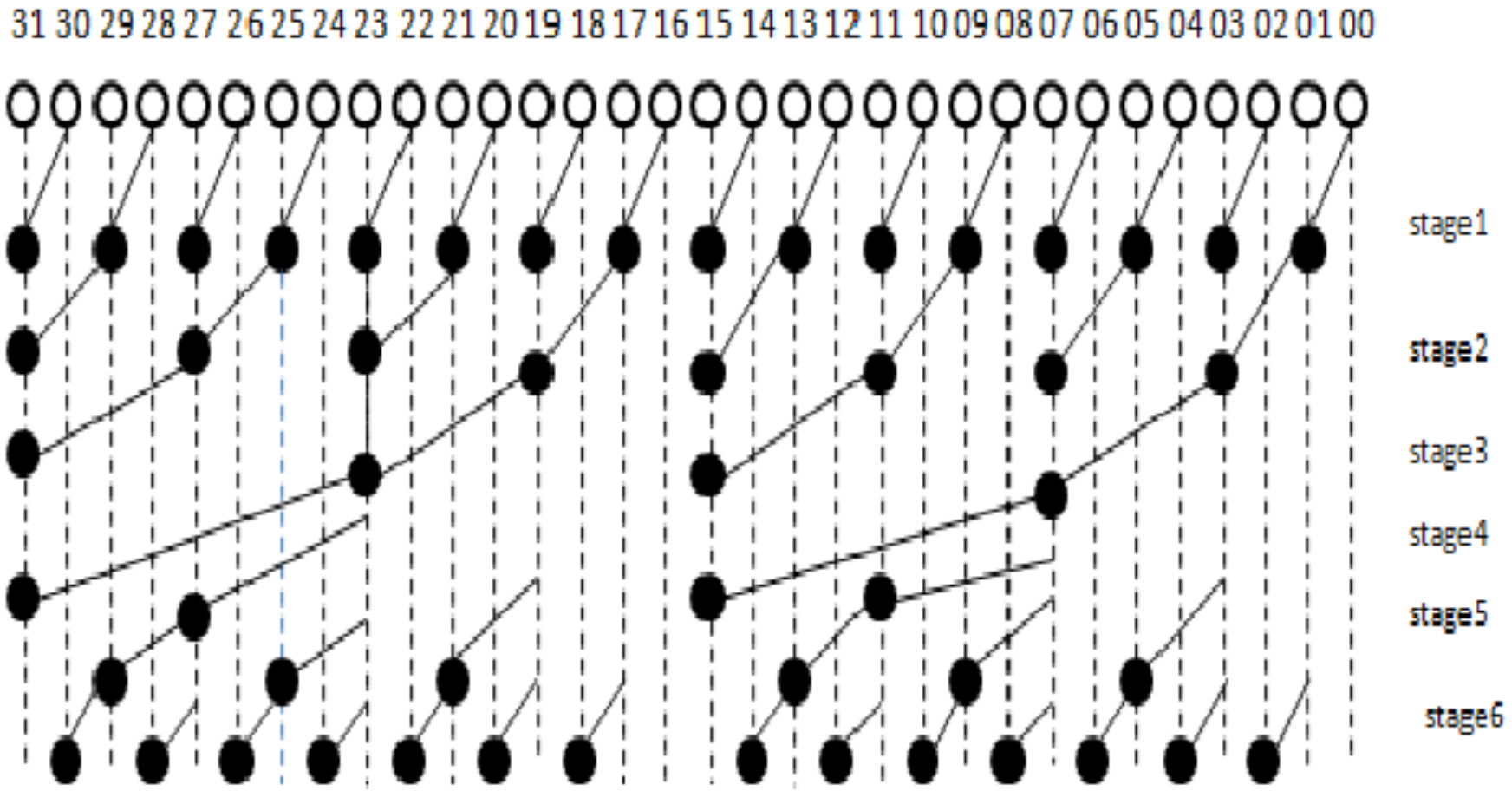
$$H_{i+1} = (G_{i+1}^*, P_i^*) \circ (G_{i-1}^*, P_{i-2}^*) \circ \ldots \circ (G_1^*, P_0^*). \qquad (13)$$

병렬 연산

$$
\begin{aligned}
H_0 &= (G_0^*, P_{-1}^*),\\
H_2 &= (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*),\\
H_4 &= (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*),\\
H_6 &= (G_6^*, P_5^*) \circ (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*),\\
H_1 &= (G_1^*, P_0^*),\\
H_3 &= (G_3^*, P_2^*) \circ (G_1^*, P_0^*),\\
H_5 &= (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \circ (G_1^*, P_0^*),\\
H_7 &= (G_7^*, P_6^*) \circ (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \circ (G_1^*, P_0^*).
\end{aligned}
$$

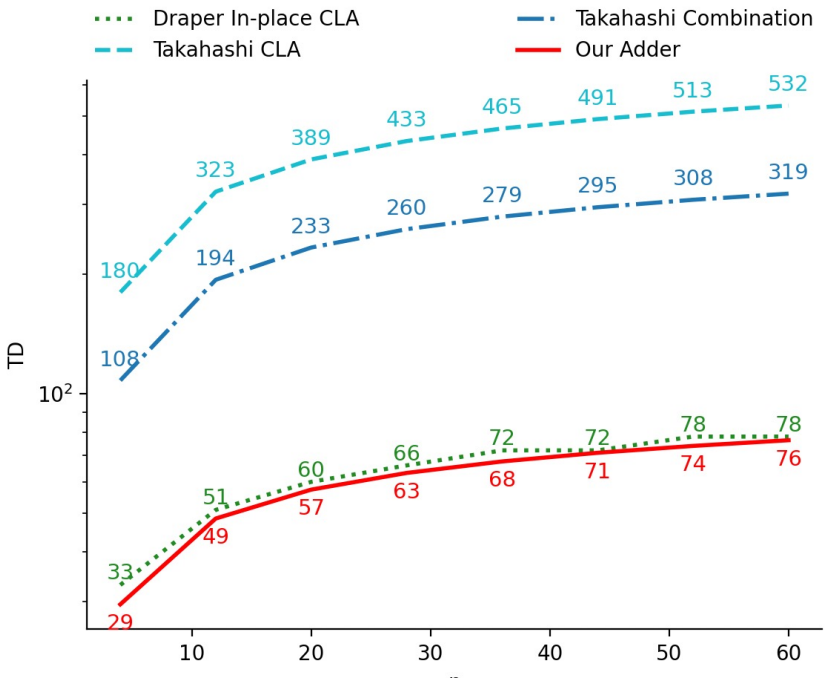


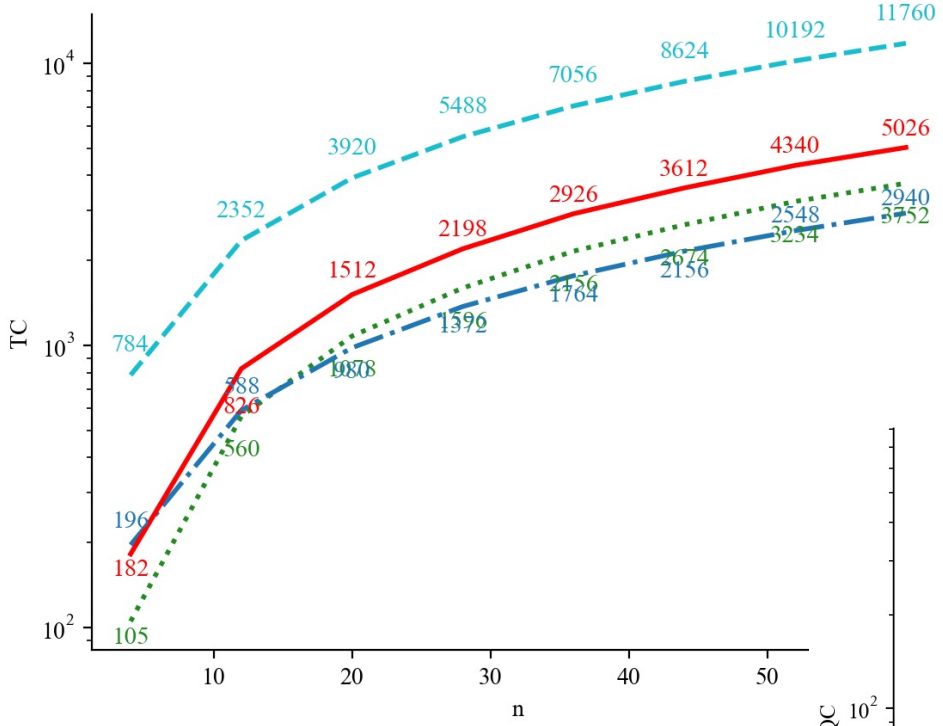
brent-kung

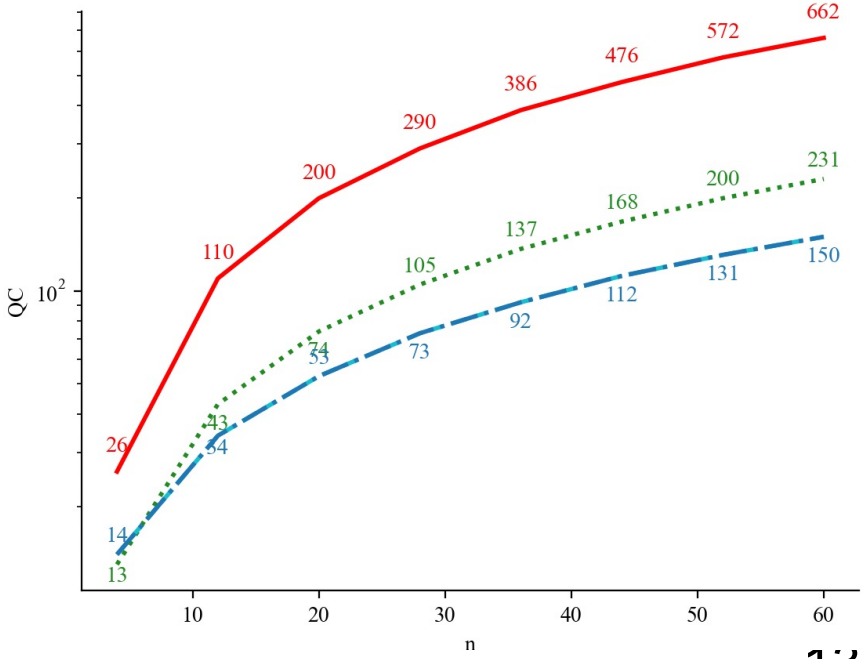

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
stage1
stage2
stage3
stage4
stage5
stage6

Discussion

- 비교

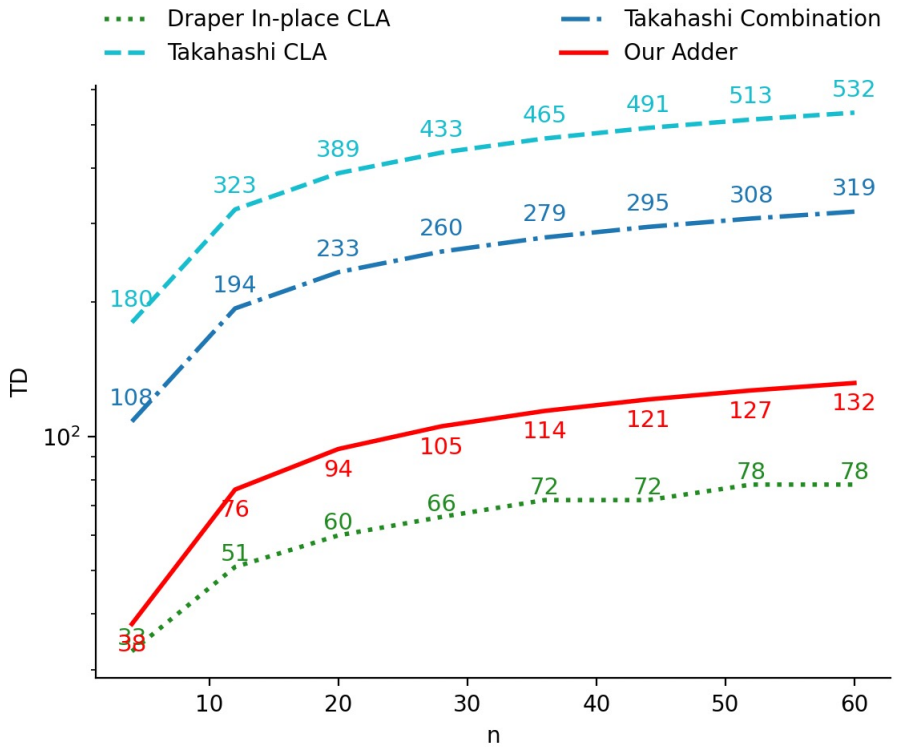


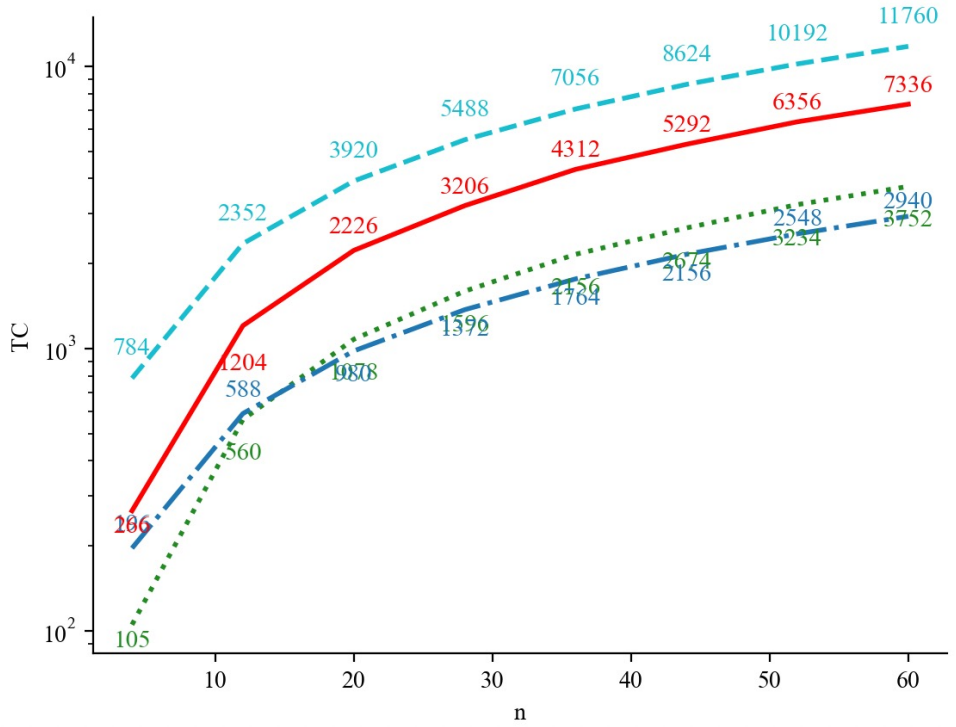




Discussion

- 언컴퓨팅 추가

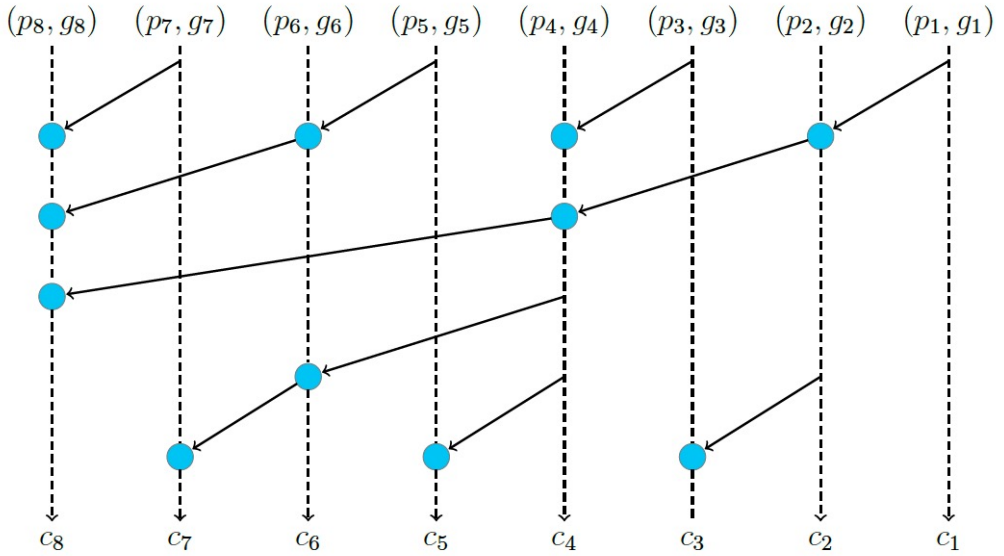




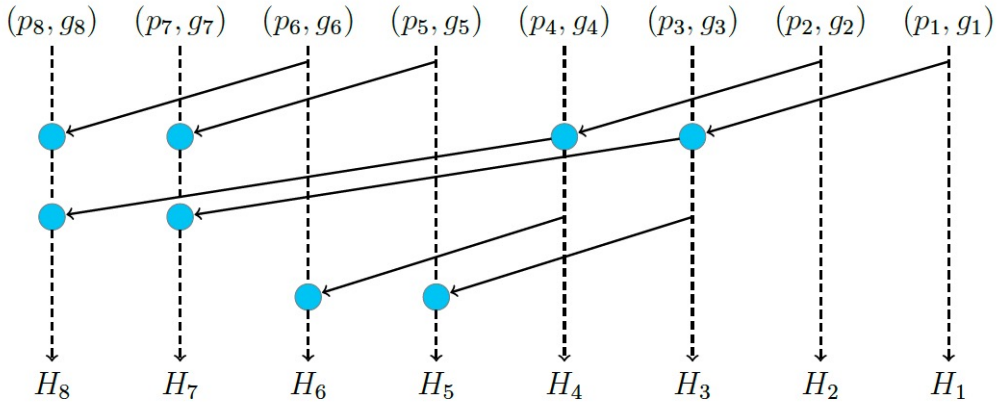
Discussion

- Brent-Kung 기반의 Draper adder는 캐리로 생성된 값을 출력 으로 사용하면 언컴퓨트하지 않아도 되므로 효율적
- 그러나 Ling base 구조는 불가능 하기 때문에 구조적인 한계가 있어 보임
- 클래식 컴퓨터의 AND와 양자컴퓨터의 AND 연산차이

$$s_i = d_i \oplus c_{i-1} = d_i \oplus (p_{i-1} \cdot H_{i-1})$$



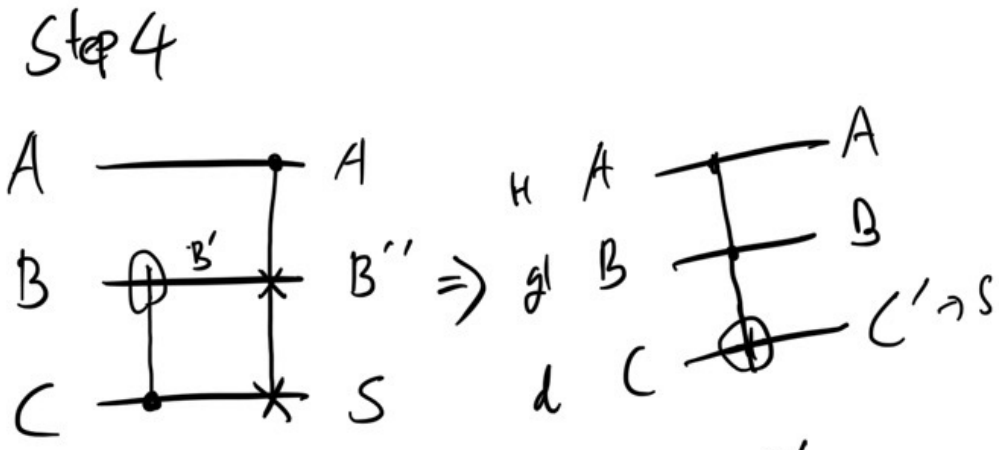
(a) Brent-Kung



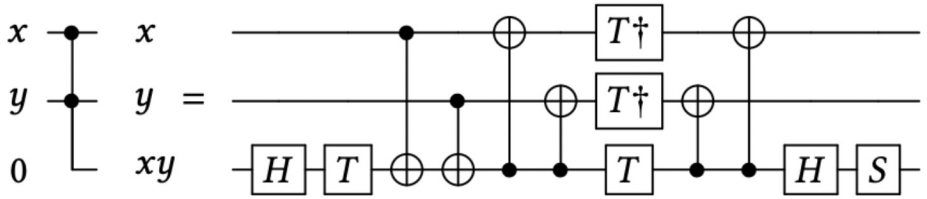
(b) Ling-based Brent-Kung

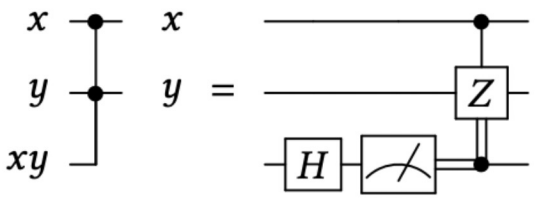
Discussion

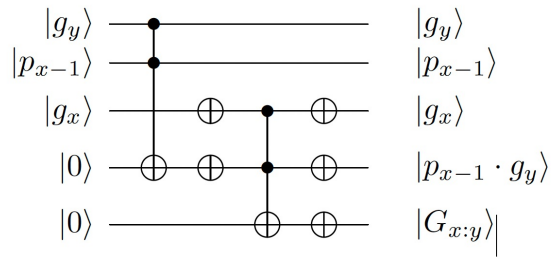
- Inplace 는 불가능한것으로 보임.
  - d xor (p and H) 때문



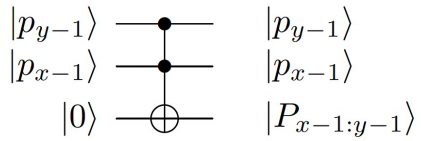
Logical and 사용







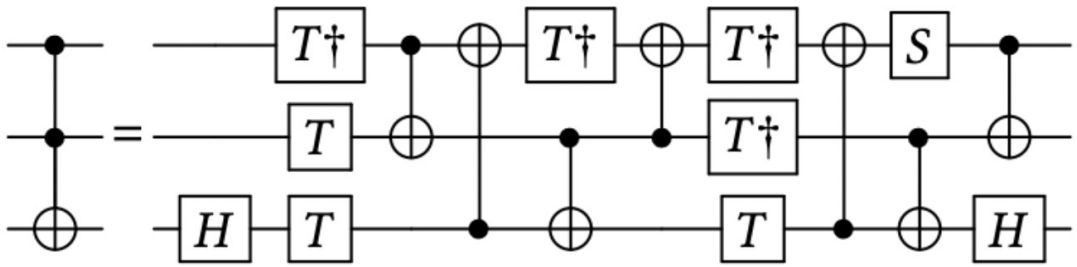
(a) Propagation of G



(b) Propagation of P

Fig. 6. Ling Propagation

$$s_i = d_i \oplus c_{i-1} = d_i \oplus (p_{i-1} \cdot H_{i-1})$$

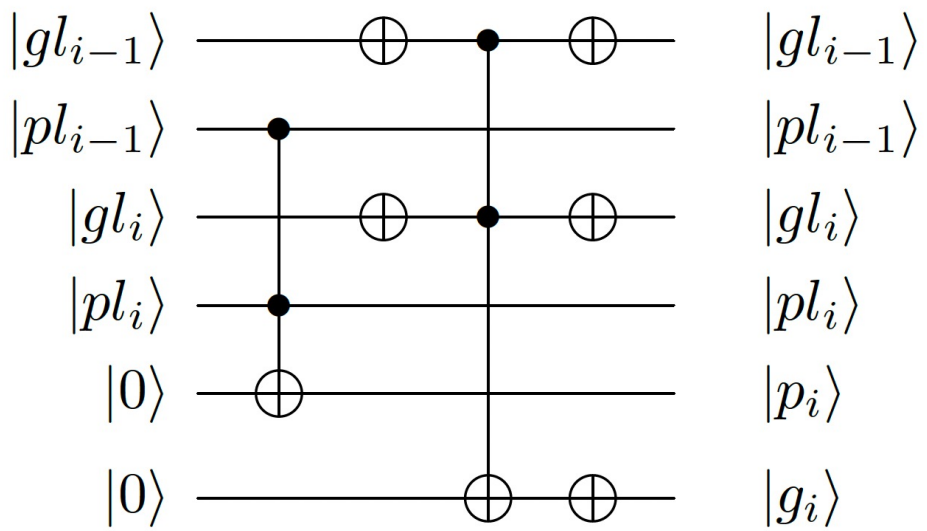


gl 제거

$$pl_i = a_i + b_i$$

$$gl_i = a_i \cdot b_i$$

$$d_i = a_i \oplus b_i$$



Q & A