Adder 기초 + Gidney Adder

임세진

https://youtu.be/CkAv20lNAMc

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

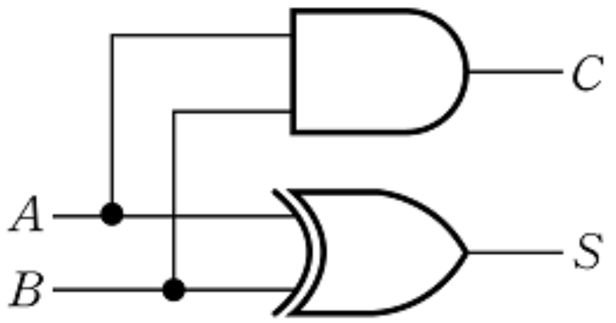
Contents

CryptoCraft LAB

01. Adder 종류 (1-bit Adder)

- Half Adder
  - 두 1-bit 입력의 합을 출력, Carry 고려 X
  - S = sum // A, B를 XOR
  - C = carry // A, B를 AND
  - LSB에서만 사용

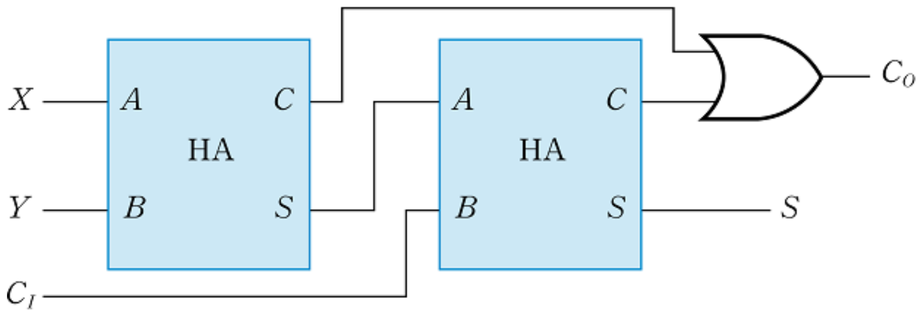
입력		출력	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



- Full Adder
  - Half Adder + Carry 고려
  - $S = A \oplus B \oplus C_{in}$
  - $C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$
  - 모든 bit에 대해 사용 가능

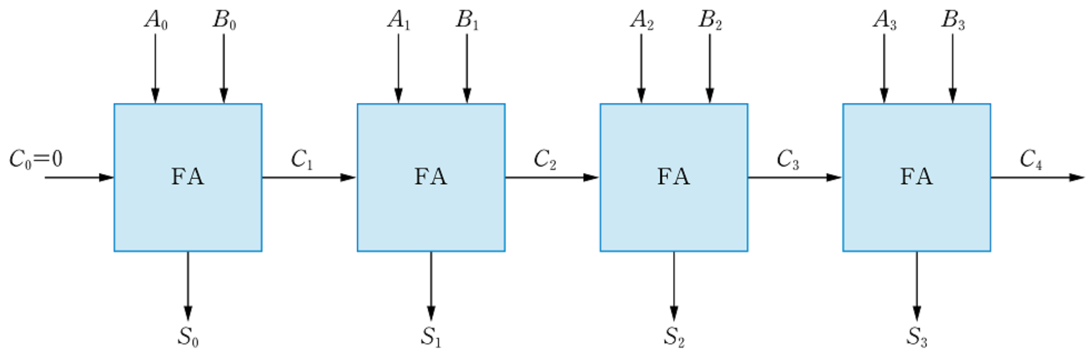
    (이전 값의 Carry에 대한 계산 가능)

입력			출력	
A	B	C_I	C_O	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



01. Adder 종류 (multi-bit Adder)

- Ripple Carry Adder
  - Full Adder를 병렬로 연결하여 구성 (간단)
  - 아래 bit의 Carry 값을 기다려야 하므로 약 O(n)의 딜레이 존재



- Carry-lookahead Adder
  - Carry를 미리 계산하여 지연 시간을 줄임

$$G(A,B) = A \cdot B$$ // A, B가 모두 1일 때만 1

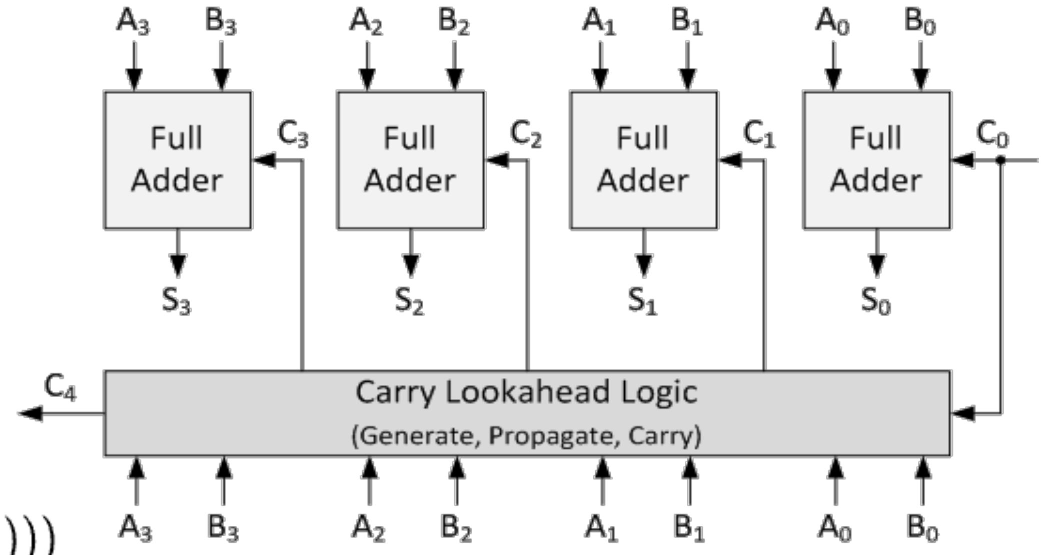
$$P(A,B) = A \oplus B$$ // 0, 1 이나 1, 0 일 때만 1

G(generate) : 하위 bit 연산에 관계없이 Carry 값이 반드시 생성되는지 여부 확인

P(propagate) : 추가적으로 Carry가 발생할 가능성 확인

$$S_i = P_i \oplus C_i \qquad C_3 = G_2 + (P_2 \cdot C_2)$$

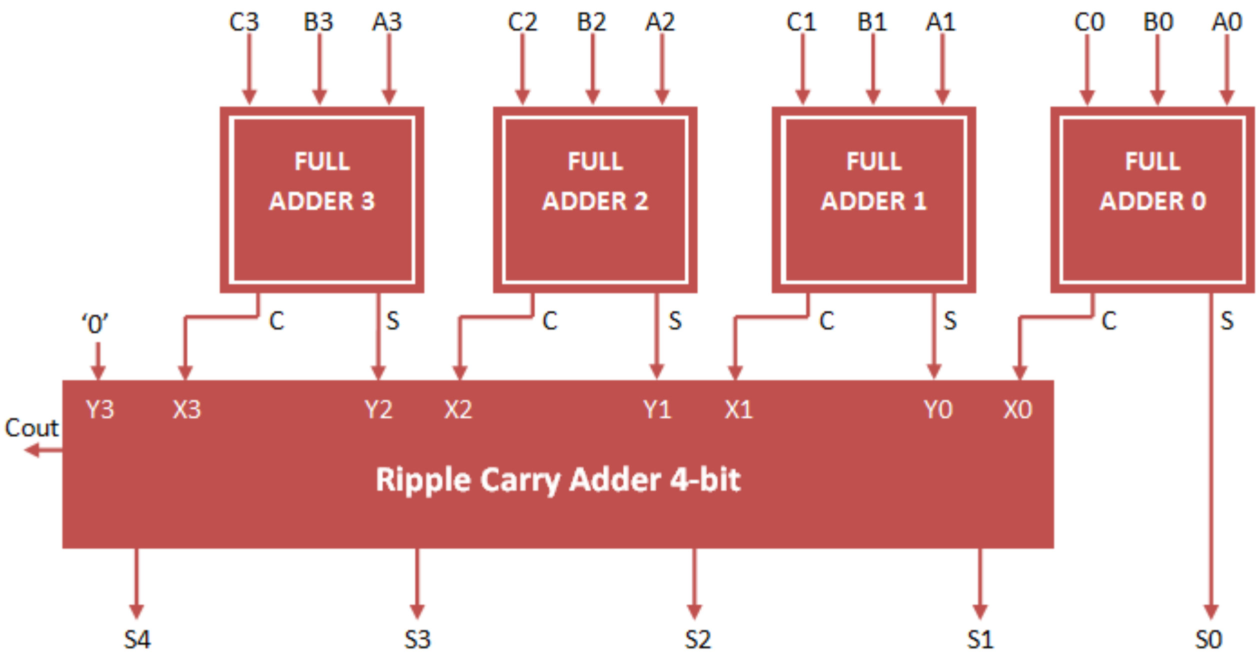
$$C_{i+1} = G_i + (P_i \cdot C_i) \qquad C_3 = G_2 + (P_2 \cdot (G_1 + (P_1 \cdot C_1)))$$



이전 자리의 Carry가 계산되기를 기다리지 않고 빠르게 처리할 수 있음

01. Adder 종류 (multi-bit Adder)

- Carry-save Adder
  - 입력된 수의 각각의 bit는 FA를 + 중간 결과는 RCA를 거쳐 최종 결과는 얻는 방식
  - **피연산자가 여러 개일 때(3개 이상)** 병렬성을 이용해서 계산 성능을 크게 향상시킬 수 있음 (곱셈기 설계 시 핵심이 되는 가산기)



02. Quantum Gate

Dirac notation (Bra-Ket notation)

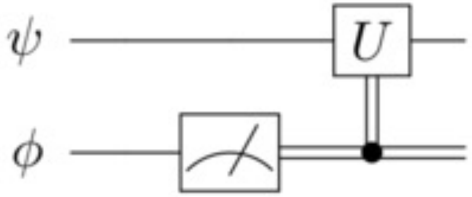
$$\begin{bmatrix}1\\0\end{bmatrix} = |0\rangle \qquad \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$$

X (NOT) gate

Input Qubit의 상태를 반전시킴

Ex) 0 → 1, 1→ 0 (NOT 연산과 동일)

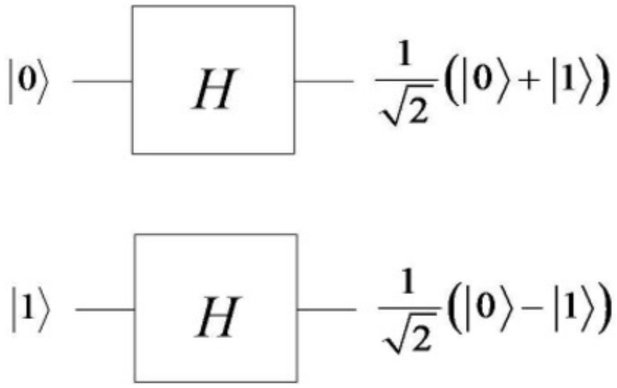
Classical control



Classical logic으로 gate를 control

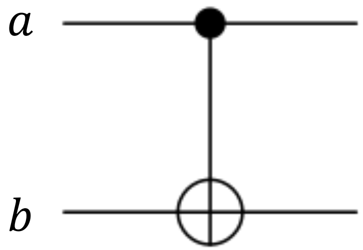
$\phi = 1$ → U gate 적용

H (Hadamard) gate



Qubit을 중첩 상태로 만듦

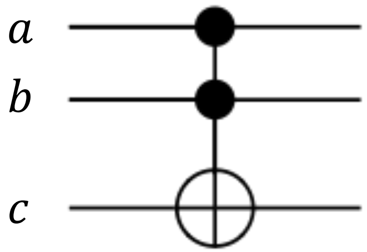
CNOT gate



a=1 → b에 X (NOT) 연산 적용

Ex) a=1, b=1 → b=0 ($a \oplus b$의 결과를 b에 저장)

Toffoli (CCNOT) gate



a=1, b=1 → c에 X (NOT) 연산 적용

Ex) a=1, b=1, c=0 → c=1 ($(a \cdot b) \oplus c$를 c에 저장)

03. Gidney Adder

Halving the cost of quantum addition

Craig Gidney

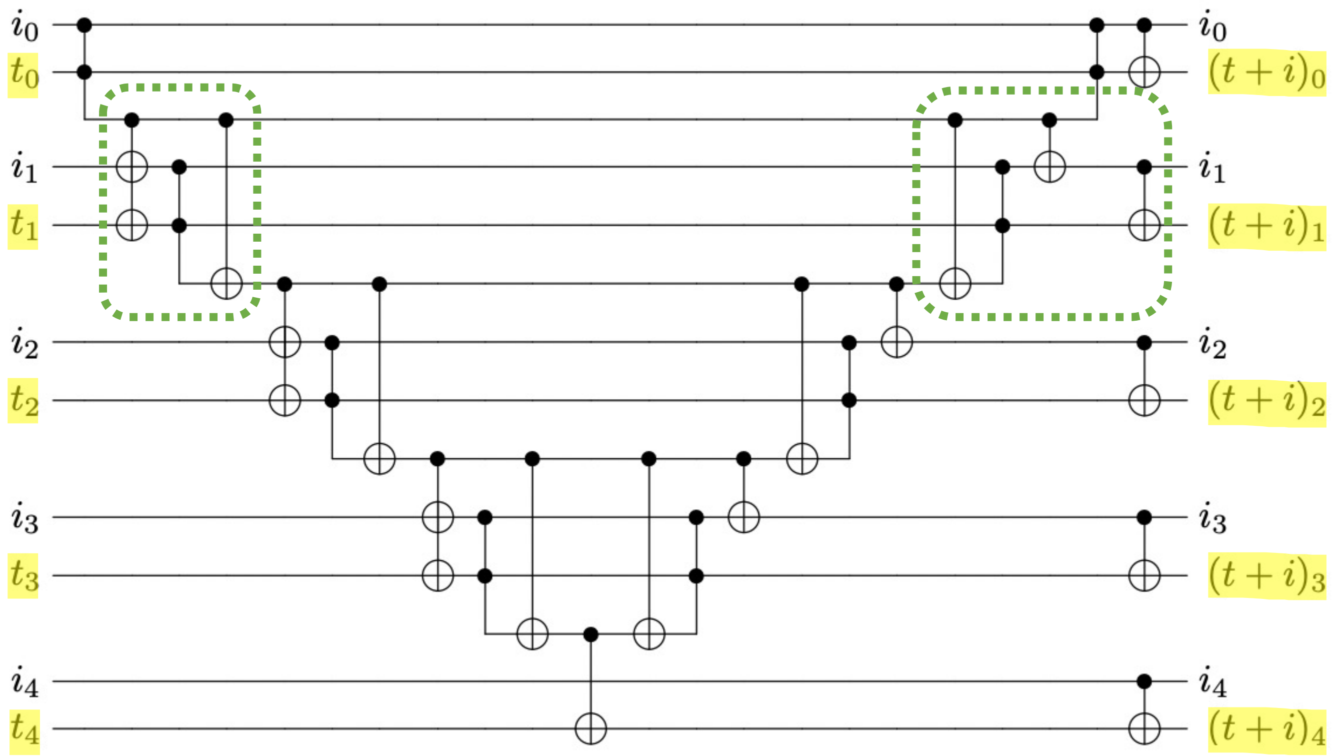
Google, Santa Barbara, CA 93117, USA

June 14, 2018

Ripple Carry Adder

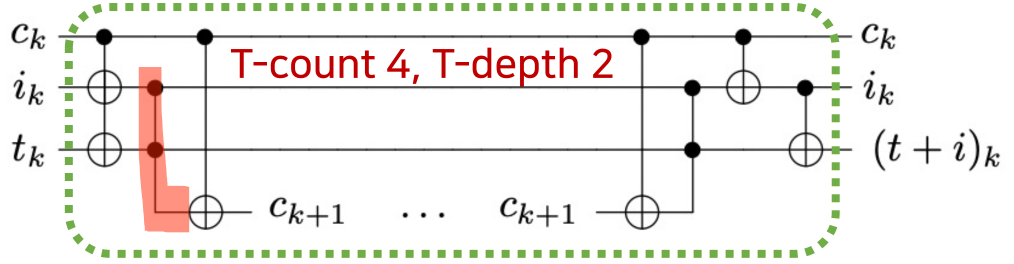
T-gate 구현에 비용이 많이 듦
→ T-count와 T-depth를 줄이는 것이 중요

Toffoli-gate는 대표적으로 T-gate가 많이 사용됨
→ 이 논문에서는 Toffoli를 Temporary logical-AND로 대체
→ T-count를 절반으로 줄임 *Compute/Uncompute 쌍으로 나타나는 Toffoli-gate

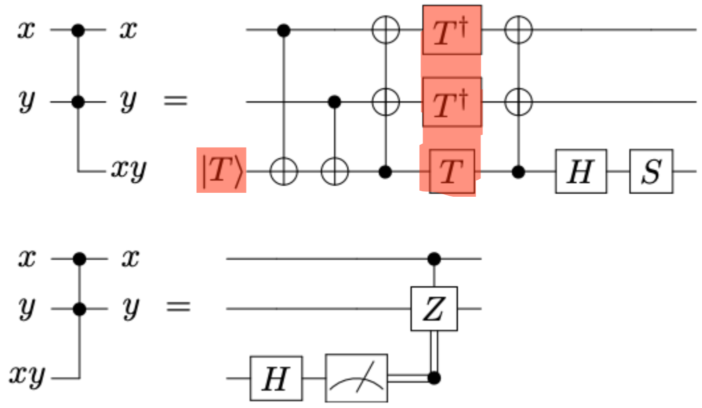


5-bit Adder

In-place



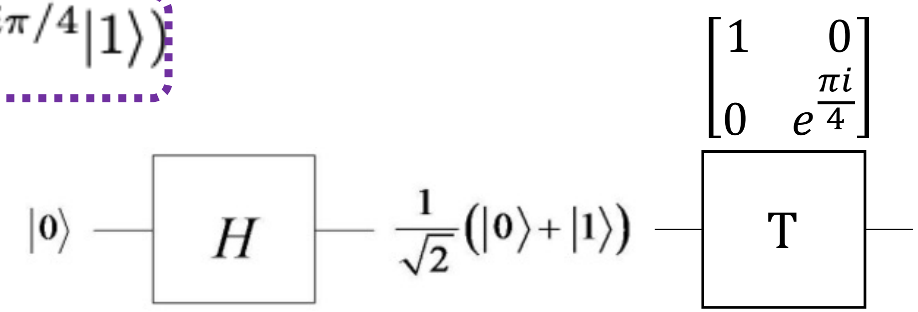
Adder circuit building-block

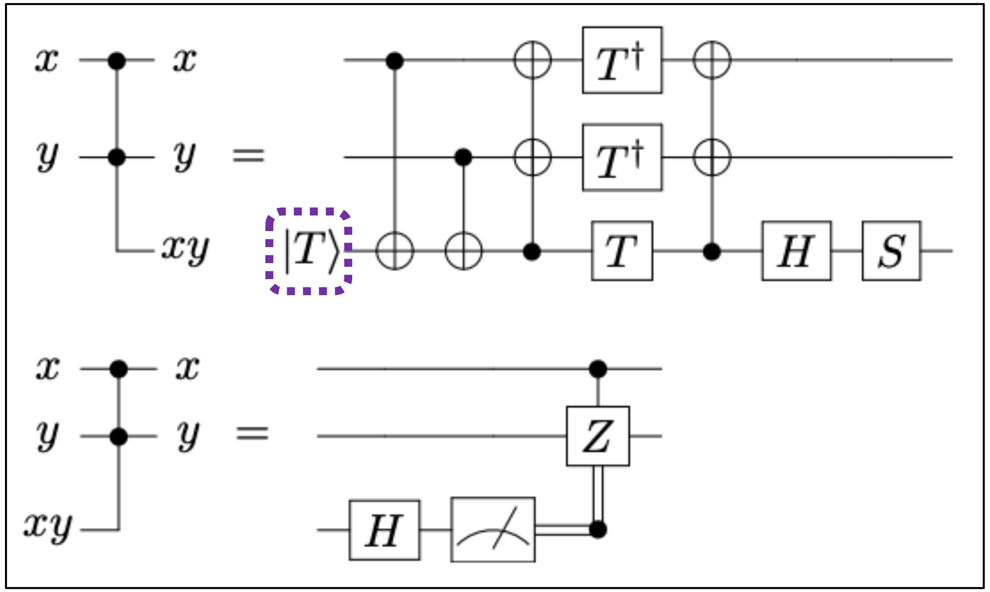


Compute(위)/Uncompute(아래) logical-AND

03. Gidney Adder

tions such as T gates. Instead, T gates are performed by distilling and consuming $\boxed{|T\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}|1\rangle)}$





Compute(위)/Uncompute(아래) logical-AND

$|T\rangle$ 는 $|0\rangle$ 에 H gate와 T gate를 수행하는 것으로 구현

$$\left(\begin{bmatrix}1 & 0\\0 & e^{\frac{\pi i}{4}}\end{bmatrix}\right) \times \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1\\0\end{bmatrix}+\begin{bmatrix}0\\1\end{bmatrix}\right)$$

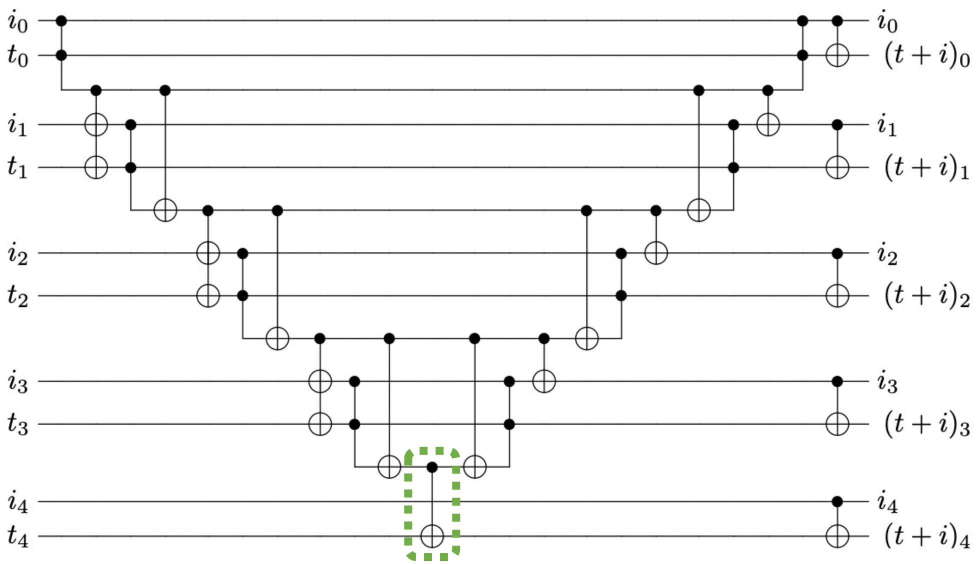
$$= \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1+0\\0+0\end{bmatrix}+\begin{bmatrix}0+0\\0+e^{\frac{\pi i}{4}}\end{bmatrix}\right)$$

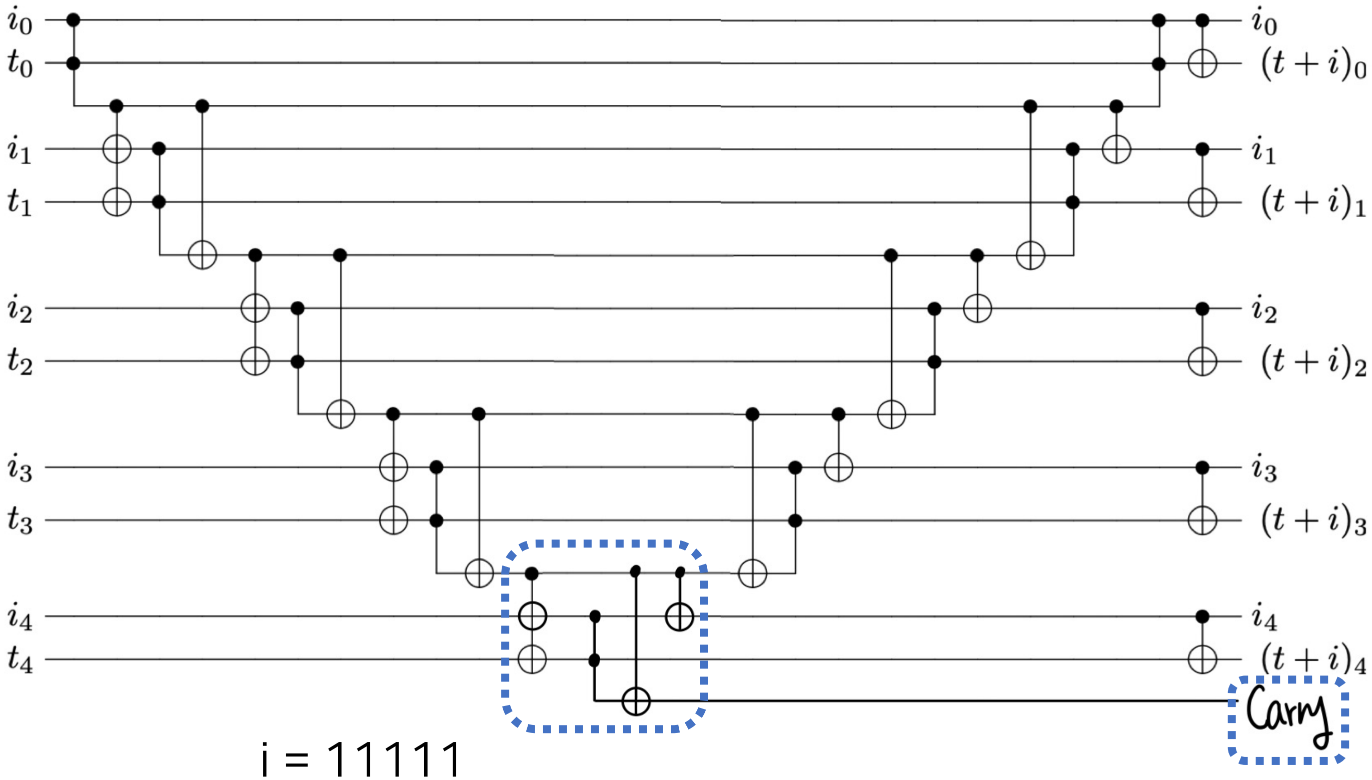
$$= \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1\\0\end{bmatrix}+\begin{bmatrix}0\\e^{\frac{\pi i}{4}}\end{bmatrix}\right)$$

$$\boxed{= \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{\pi i}{4}}|1\rangle\right)}$$

03. Gidney Adder

논문에서 제안한 Adder 회로를 살펴보면 Modular Adder임 → 일반 Adder가 필요



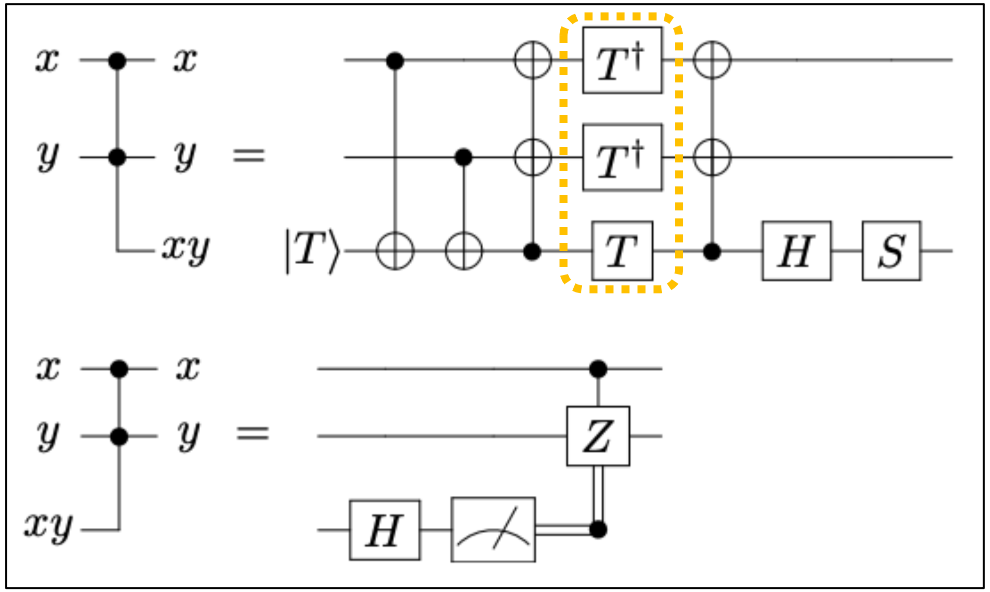


5-bit Adder 회로
수정 전 → 수정 후

i = 11111
t = 10000

01111 →101111

04. 구현 (Cirq)



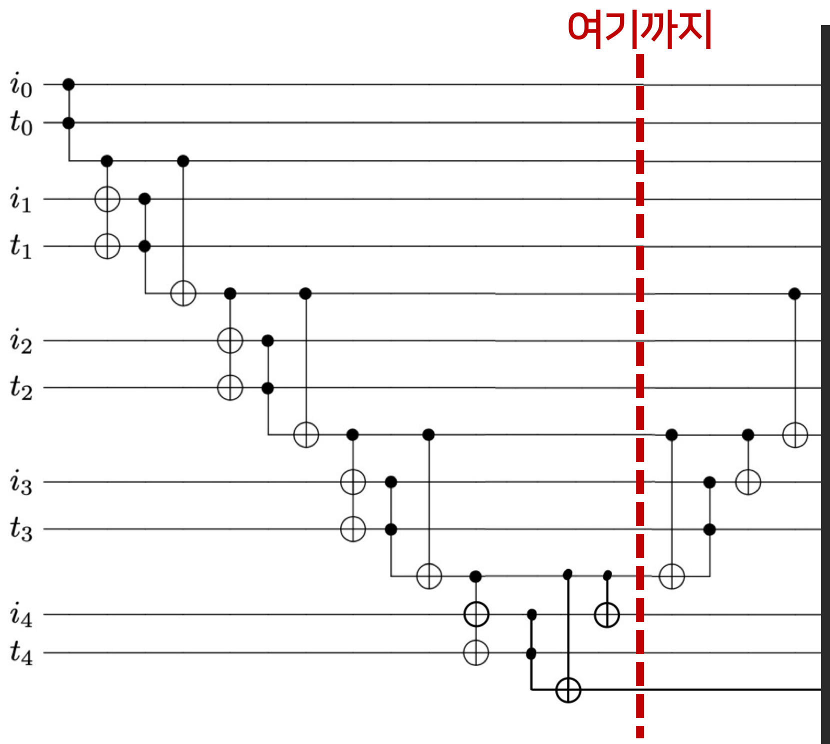

```
def logical_and(i,t,c):
    yield [H(c)]
    yield [T(c)]
    yield [CNOT(i,c), CNOT(t,c)]
    yield [CNOT(c,i), CNOT(c,t)]
    yield cirq.Moment((T ** -1)(i), (T ** -1)(t), T(c))
    yield [CNOT(c,i), CNOT(c,t)]
    yield [H(c)]
    yield [S(c)]

sejin
def logical_and_reverse(i,t,c):
    yield [H(c)]
    yield [measure(c, key=c.name)]
    yield [CZ(i,t).with_classical_controls(c.name)]
```

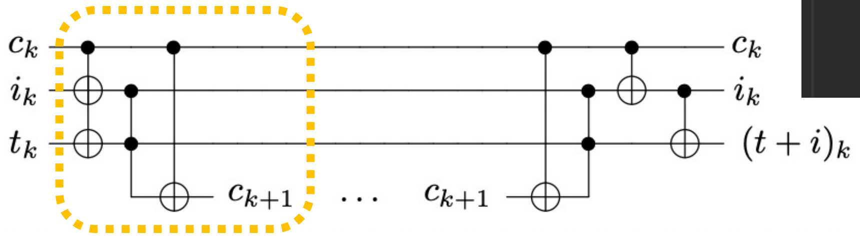
```
A0: ───X───────────@───────X───T^-1──────────X───@──────────────────────
                   │       │                 │   │
B0: ───────────────┼───@───┼───X───────T^-1──┼───X───X───M('result')───
                   │   │   │   │             │   │       │
C0: ───────H───T───X───X───@───@───────T─────@───@───H───S───M──────────
[0 1]
T_depth : 3
T_count : 4
```

```
A0: ───X───────────@───────X───────T^-1───X───────@──────────────────────
                   │       │              │       │
B0: ───────────────┼───@───┼───X───T^-1───┼───X───X───────M('result')───
                   │   │   │   │          │   │           │
C0: ───────H───T───X───X───@───@───T──────@───@───H───S───M─────────────
[0 1]
T_depth : 2
T_count : 4
```

자동으로 depth를 줄여주는 기능의 역효과로, Moment 연산을 안썼을 경우 T-depth가 늘어남

04. 구현 (Cirq)



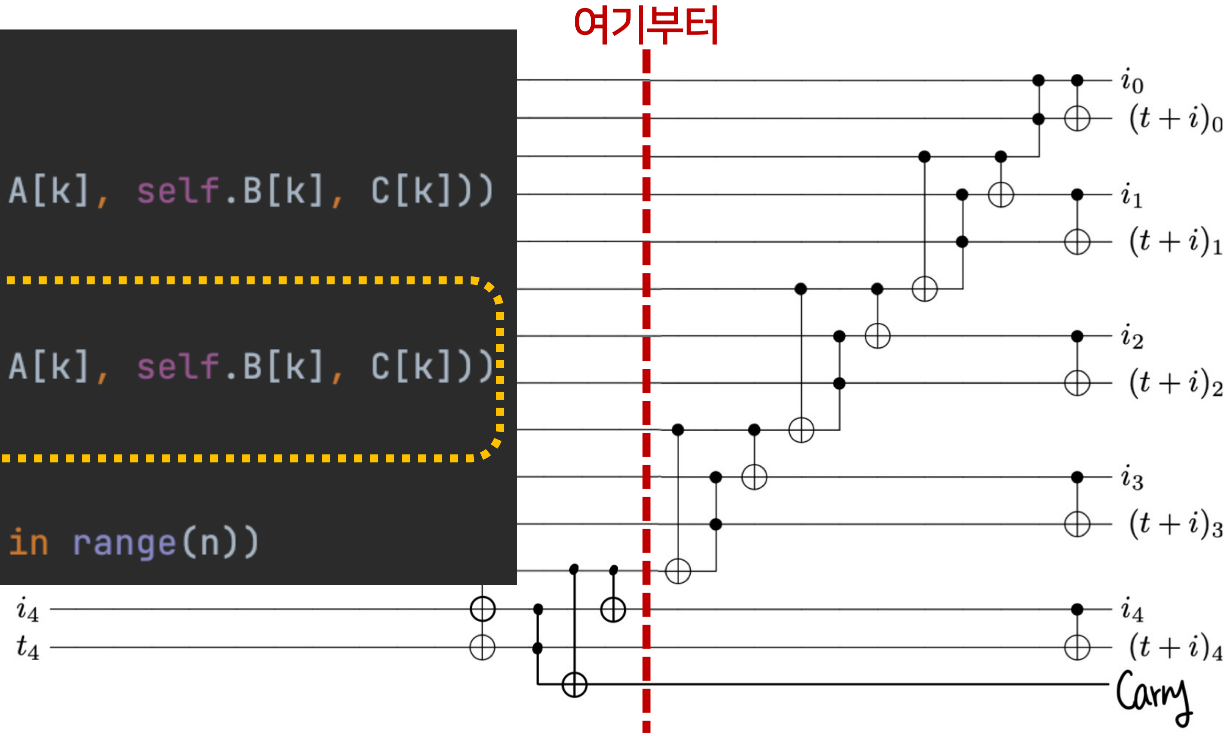


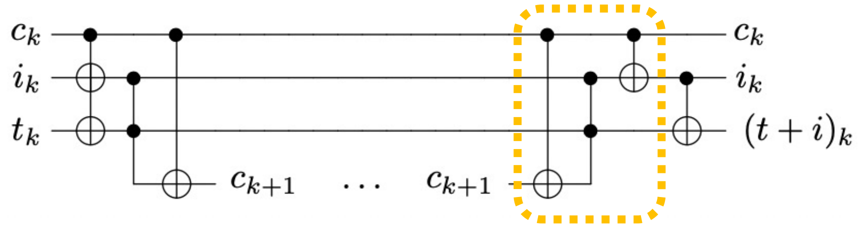

```python
def construct_circuit(self):
    n = len(self.A)
    C = [NamedQubit("C" + str(i)) for i in range(n)]
    operations = []

    for k in range(n):
        if(k==0):
            operations.append(logical_and(self.A[k], self.B[k], C[k]))

        else:
            operations.append([CNOT(C[k-1], self.A[k]), CNOT(C[k-1], self.B[k])])
            operations.append(logical_and(self.A[k], self.B[k], C[k]))
            operations.append(CNOT(C[k-1], C[k]))
            if (k == n-1):
                operations.append(CNOT(C[k-1], self.A[k]))
```

building-block

04. 구현 (Cirq)

```
for k in reversed(range(n-1)):
    if(k==0):
        operations.append(logical_and_reverse(self.A[k], self.B[k], C[k]))
    else: building-block
        operations.append(CNOT(C[k-1], C[k]))
        operations.append(logical_and_reverse(self.A[k], self.B[k], C[k]))
        operations.append(CNOT(C[k-1], self.A[k]))

operations.append(CNOT(self.A[k], self.B[k]) for k in range(n))
```





05. Demo

감사합니다