Factoring using 2n+2 qubits with Toffoli based modular multiplication

Thomas Häner[1,2] **Martin Roetteler**[2] Krysta M. Svore[2]

[1]Institute for Theoretical Physics
ETH Zürich, Switzerland

[2]Quantum Architectures and Computation Group
Microsoft Research, Redmond, U.S.A.

arXiv:1611.07995

19th Annual SQuInT Workshop
Baton Rouge, Louisiana
February 24, 2017

Outline

- Improved constant increment "$+1$"
- Toffoli networks for modular multiplication
- Testable circuit for Shor on $2n+2$ qubits
- Simulations

How to increment a quantum register?

Realizing a cyclic shift

How to realize $x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1: Recursive

Good: needs only $n+1$ qubits
Bad: needs $O(n^2)$ gates

Solution 2: Fourier-style

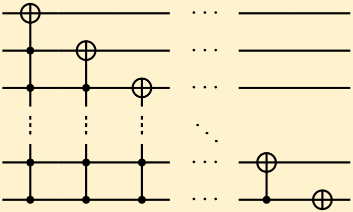
Good: needs only n qubits
Bad: needs $O(n^2)$ gates
Ugly: needs rotations

How to increment a quantum register?

Realizing a cyclic shift

How to realize $x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1: Recursive



Good: needs only $n{+}1$ qubits
Bad: needs $O(n^2)$ gates

Solution 2: Fourier-style

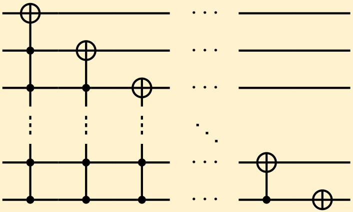
Good: needs only n qubits
Bad: needs $O(n^2)$ gates
Ugly: needs rotations

How to increment a quantum register?

Realizing a cyclic shift

How to realize $x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1: Recursive



Good: needs only $n{+}1$ qubits
Bad: needs $O(n^2)$ gates

Solution 2: Fourier-style

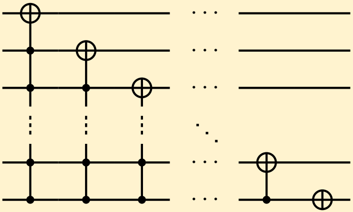
Good: needs only n qubits
Bad: needs $O(n^2)$ gates
Ugly: needs rotations

How to increment a quantum register?

Realizing a cyclic shift

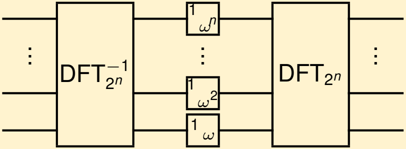
How to realize $x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1: Recursive



Good: needs only $n+1$ qubits
Bad: needs $O(n^2)$ gates

Solution 2: Fourier-style



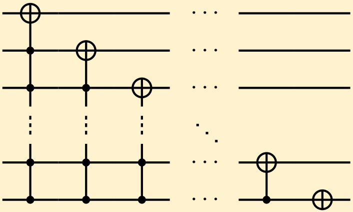
Good: needs only n qubits
Bad: needs $O(n^2)$ gates
Ugly: needs rotations

How to increment a quantum register?

Realizing a cyclic shift

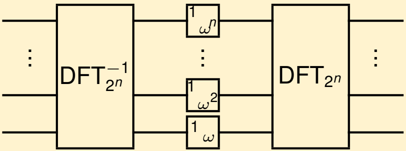
How to realize $x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1: Recursive



Good: needs only $n+1$ qubits
Bad: needs $O(n^2)$ gates

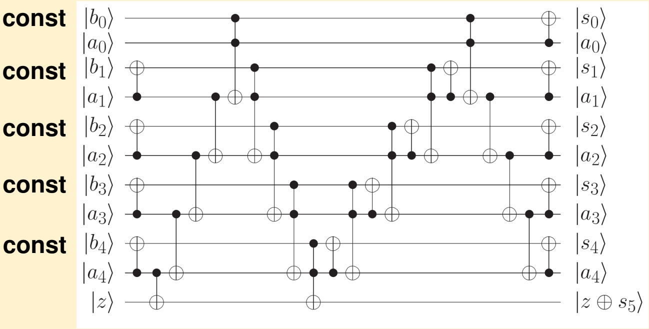
Solution 2: Fourier-style



Good: needs only n qubits
Bad: needs $O(n^2)$ gates
Ugly: needs rotations

How to increment a quantum register?

Solution 3: Constant folding



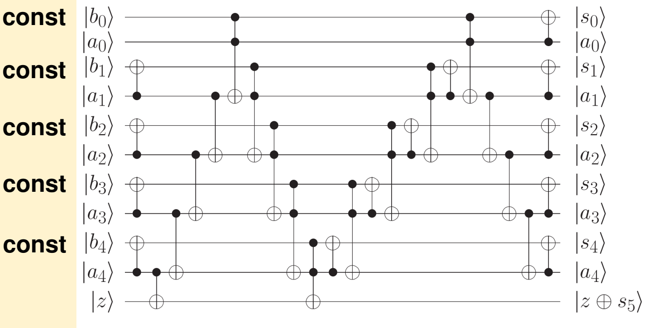
Good: needs $O(n)$ gates. Bad: needs $2n$ qubits

Solution 4 ???

Is there a circuit with n qubits, that needs only $O(n)$ gates?
Or even just one with $n + const$ qubits but $O(n)$ gates?

How to increment a quantum register?

Solution 3: Constant folding



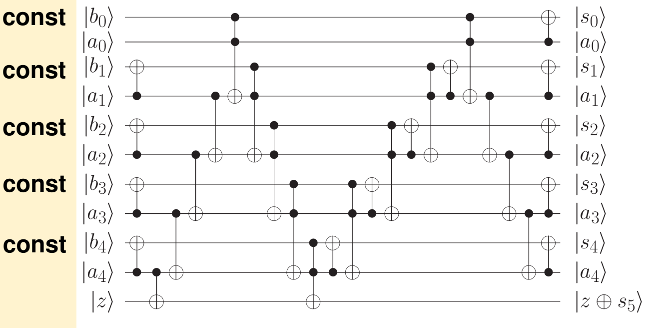
Good: needs $O(n)$ gates. Bad: needs $2n$ qubits

Solution 4 ???

Is there a circuit with n qubits, that needs only $O(n)$ gates?
Or even just one with $n + const$ qubits but $O(n)$ gates?

How to increment a quantum register?

Solution 3: Constant folding



Good: needs $O(n)$ gates. Bad: needs $2n$ qubits

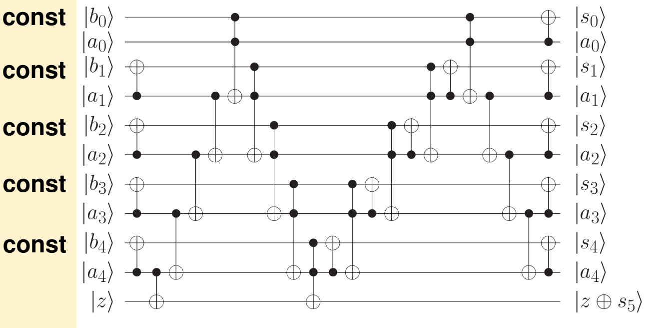
Solution 4 ???

Is there a circuit with n qubits, that needs only $O(n)$ gates?

Or even just one with $n + const$ qubits but $O(n)$ gates?

How to increment a quantum register?

Solution 3: Constant folding

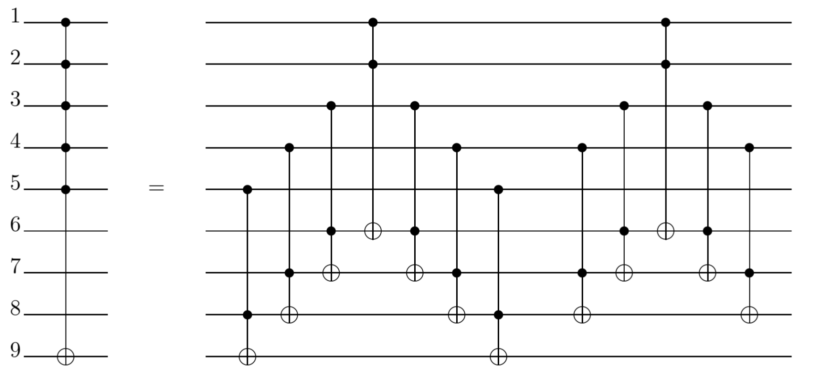


Good: needs $O(n)$ gates. Bad: needs $2n$ qubits

Solution 4 ???

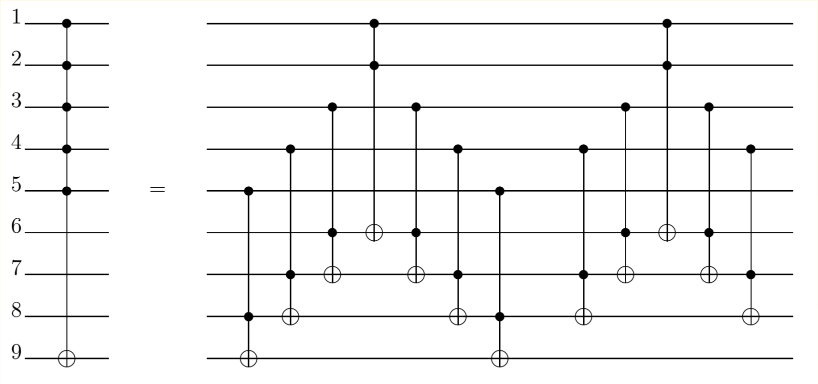
Is there a circuit with n qubits, that needs only $O(n)$ gates?
Or even just one with $n + const$ qubits but $O(n)$ gates?

Trick from Barenco et al (PRA'95)



Note: this uses $n/2$ "dirty" ancillas qubits.

Trick from Barenco et al (PRA'95)



Note: this uses $n/2$ "dirty" ancillas qubits.

Incrementer "$+1$" by Craig Gidney

- Based on the following trick:

$$\begin{aligned} |x\rangle\,|g\rangle &\mapsto |x-g\rangle\,|g\rangle \\ &\mapsto |x-g\rangle\,\left|g'-1\right\rangle \\ &\mapsto \left|x-g-g'+1\right\rangle\,\left|g'-1\right\rangle \\ &\mapsto |x+1\rangle\,|g\rangle \end{aligned}$$

  (Note that $\overline{g}+1=g'$, where g' denotes two's complement and $\overline{g}$ denotes one's complement, and that $g+g'=0$).

- If n dirty ancillas are available, this allows to implement $+1$ increment using only $O(n)$ Toffoli gates.

- If only 1 dirty ancilla is available, precompute final carry, apply a split and recurse. Leads to $O(n\log n)$ Toffoli gates.

Incrementer "$+1$" by Craig Gidney

- Based on the following trick:

$$\begin{aligned} |x\rangle\,|g\rangle &\mapsto |x-g\rangle\,|g\rangle \\ &\mapsto |x-g\rangle\,\big|g'-1\big\rangle \\ &\mapsto \big|x-g-g'+1\big\rangle\,\big|g'-1\big\rangle \\ &\mapsto |x+1\rangle\,|g\rangle \end{aligned}$$

  (Note that $\overline{g}+1=g'$, where g' denotes two's complement and $\overline{g}$ denotes one's complement, and that $g+g'=0$).

- If n dirty ancillas are available, this allows to implement $+1$ increment using only $O(n)$ Toffoli gates.

- If only 1 dirty ancilla is available, precompute final carry, apply a split and recurse. Leads to $O(n\log n)$ Toffoli gates.

Incrementer "$+1$" by Craig Gidney

- Based on the following trick:

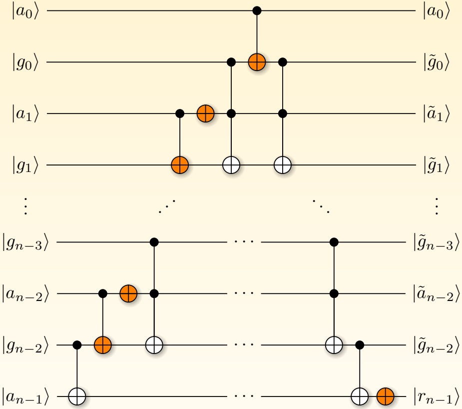
$$\begin{aligned} |x\rangle\,|g\rangle &\mapsto |x-g\rangle\,|g\rangle \\ &\mapsto |x-g\rangle\,\left|g'-1\right\rangle \\ &\mapsto \left|x-g-g'+1\right\rangle\,\left|g'-1\right\rangle \\ &\mapsto |x+1\rangle\,|g\rangle \end{aligned}$$

  (Note that $\overline{g}+1=g'$, where g' denotes two's complement and $\overline{g}$ denotes one's complement, and that $g+g'=0$).

- If n dirty ancillas are available, this allows to implement $+1$ increment using only $O(n)$ Toffoli gates.

- If only 1 dirty ancilla is available, precompute final carry, apply a split and recurse. Leads to $O(n\log n)$ Toffoli gates.

Carry precomputation w/dirty qubits

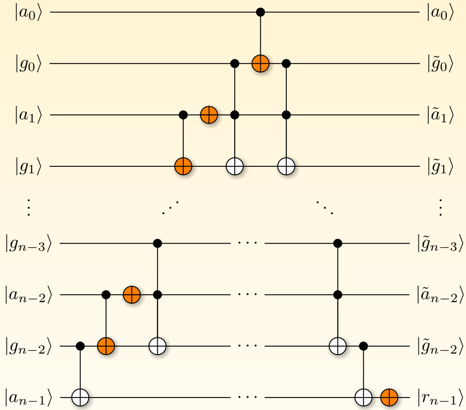


Works for any constant addition "$+c$" (not just "$+1$").

Bits are encoded in the presence/absence of orange gates.

This needs $4(n-2)$ Toffoli and $4wt(c)$ Clifford gates.

Carry precomputation w/dirty qubits

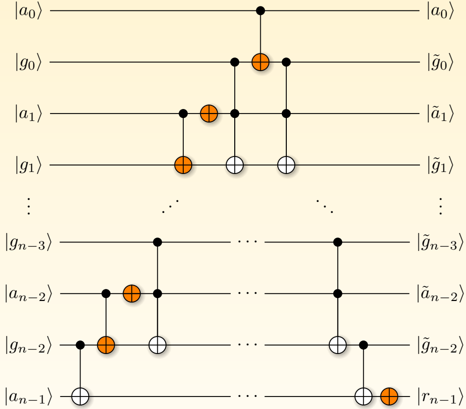


Works for any constant addition "$+c$" (not just "$+1$").

Bits are encoded in the presence/absence of orange gates.

This needs $4(n-2)$ Toffoli and $4wt(c)$ Clifford gates.

Carry precomputation w/dirty qubits

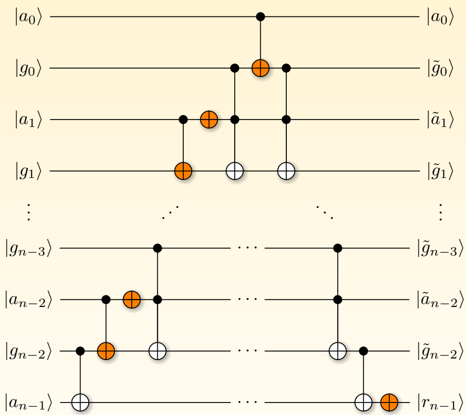


Works for any constant addition "$+c$" (not just "$+1$").

Bits are encoded in the presence/absence of orange gates.

This needs $4(n-2)$ Toffoli and $4wt(c)$ Clifford gates.

Carry precomputation w/dirty qubits

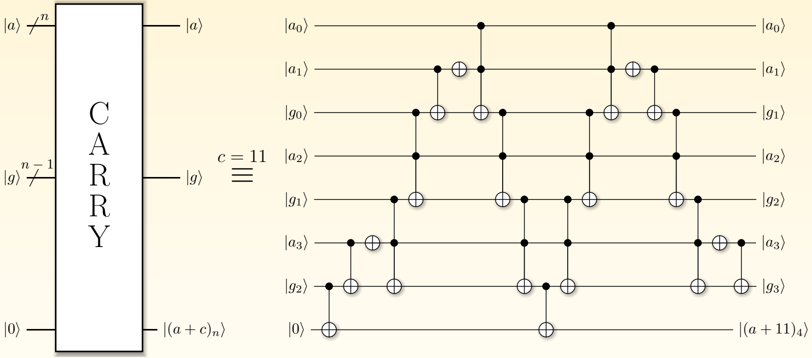


Works for any constant addition "$+c$" (not just "$+1$").

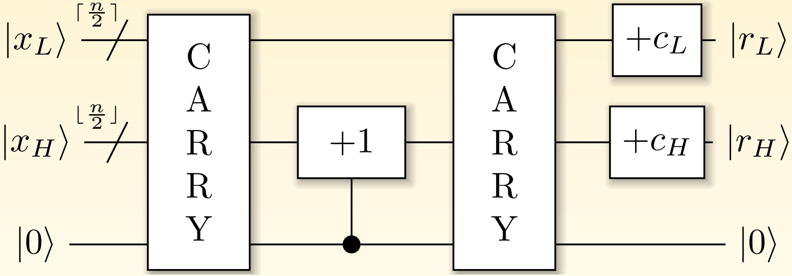
Bits are encoded in the presence/absence of orange gates.

This needs $4(n-2)$ Toffoli and $4wt(c)$ Clifford gates.

Carry precomputation w/dirty qubits



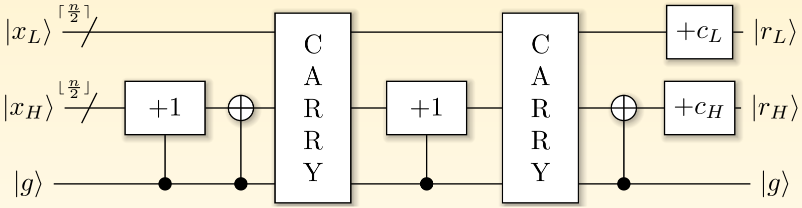
Putting it all together: addition-by-constant



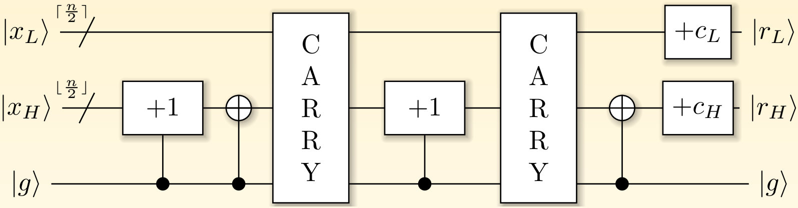
Note that this circuit uses a clean ancilla to detect if the final overflow happened.

However, it is not necessary to use a clean qubit, a dirty qubit suffices as shown next.

Carry computation using garbage only



Carry computation using garbage only



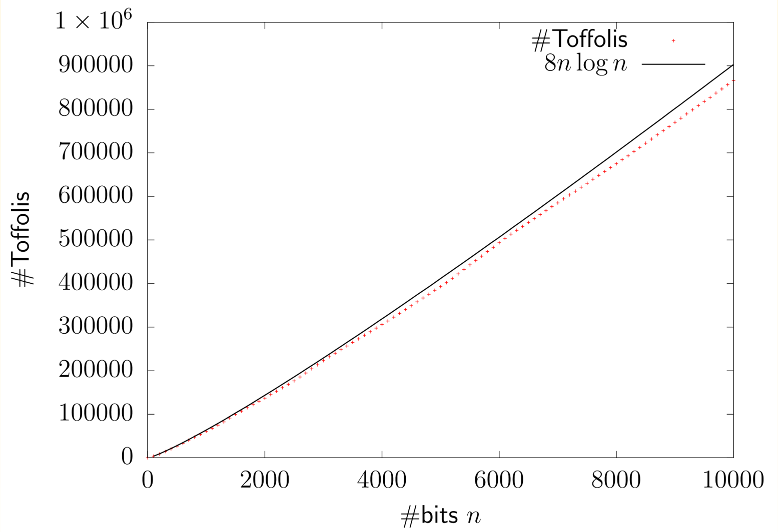
$$T_A(n) = 2T_A(\frac{n}{2}) + 2(\underbrace{2 \cdot 2\frac{n}{2}}_{\text{incr}} + \underbrace{4\frac{n}{2}}_{\text{carry}})$$

$$= 2T_A(\frac{n}{2}) + 8n$$

$$\vdots$$

$$= 8n \log_2 n$$

Experimental results (addition)



Toffoli circuits implemented and simulated in LIQ$Ui|\rangle$.

Addition-by-constant: comparison

- Fourier-based adder, Draper-style:
  Advantage: Ancilla-free
  Disadvantage: $\Theta(n^2)$ gates, not exact

- Cuccaro et al adder, with folded constants:
  Advantage: $O(n)$ runtime
  Disadvantage: Requires $n+1$ extra (clean) qubits

- Takahashi et al adder, with folded constants:
  Advantage: $O(n)$ runtime
  Disadvantage: Requires n extra (clean) qubits

Addition-by-constant: comparison

- Fourier-based adder, Draper-style:
  Advantage: Ancilla-free
  Disadvantage: $\Theta(n^2)$ gates, not exact

- Cuccaro et al adder, with folded constants:
  Advantage: $O(n)$ runtime
  Disadvantage: Requires $n+1$ extra (clean) qubits

- Takahashi et al adder, with folded constants:
  Advantage: $O(n)$ runtime
  Disadvantage: Requires n extra (clean) qubits

- **Our adder:**
  Advantage: Toffoli-based, only 1 extra (dirty) qubit
  Disadvantage: $\Theta(n \log n)$ runtime

Application: modular exponentiation

Shor's algorithm

Finds period r of $f(x) = a^x \bmod N$, where a, N constant.

$$\begin{aligned} a^x \bmod N &= a^{x_m \cdot 2^m} \cdots a^{x_1 \cdot 2^1} \cdot a^{x_0 \cdot 2^0} \bmod N \\ &= (a^{2^m} \bmod N)^{x_m} \cdots (a^{2^1} \bmod N)^{x_1} \cdot (a^{2^0} \bmod N)^{x_0} \end{aligned}$$

Application: modular exponentiation

Shor's algorithm

Finds period r of $f(x) = a^x \bmod N$, where a, N constant.

$$\begin{aligned} a^x \bmod N &= a^{x_m \cdot 2^m} \cdots a^{x_1 \cdot 2^1} \cdot a^{x_0 \cdot 2^0} \bmod N \\ &= (a^{2^m} \bmod N)^{x_m} \cdots (a^{2^1} \bmod N)^{x_1} \cdot (a^{2^0} \bmod N)^{x_0} \end{aligned}$$

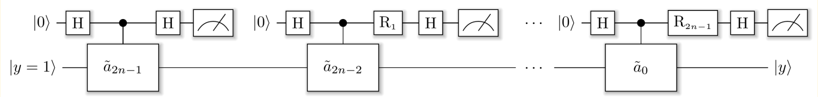
- Start with $|y = 1\rangle |x_0\rangle$
- For $i = 0, ..., m$, apply the operator

$$\sum_y \left|(a^{2^i} y) \bmod N\right\rangle \langle y| \otimes |1\rangle\langle 1| + \mathbb{1} \otimes |0\rangle\langle 0|$$

  to $|y\rangle |x_i\rangle$.
- Final state:

$$|a^x \bmod N\rangle |x\rangle$$

Shor's algorithm, PE style



Note that computing a modular multiplication

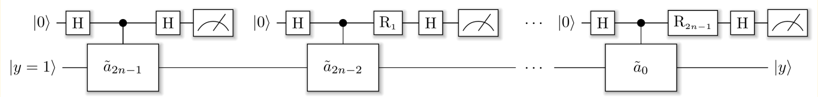
$$\begin{aligned} ax \bmod N &= a \cdot (2^m x_m + \cdots + 2x_1 + x_0) \bmod N \\ &= ((2^m a) \bmod N) x_m \oplus \cdots \oplus ((2a) \bmod N) x_1 \oplus a x_0 \end{aligned}$$

can be done using $m + 1$ controlled modular additions

Controlled multiplication $\Rightarrow$ Doubly-controlled modular additions

How to perform modular additions?

Shor's algorithm, PE style



Note that computing a modular multiplication

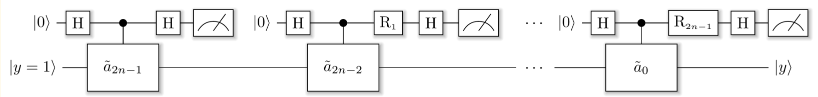
$$
\begin{aligned}
ax \bmod N &= a \cdot (2^m x_m + \cdots + 2x_1 + x_0) \bmod N \\
&= ((2^m a) \bmod N) x_m \oplus \cdots \oplus ((2a) \bmod N) x_1 \oplus a x_0
\end{aligned}
$$

can be done using $m + 1$ controlled modular additions

Controlled multiplication $\Rightarrow$ Doubly-controlled modular additions

How to perform modular additions?

Shor's algorithm, PE style



Note that computing a modular multiplication

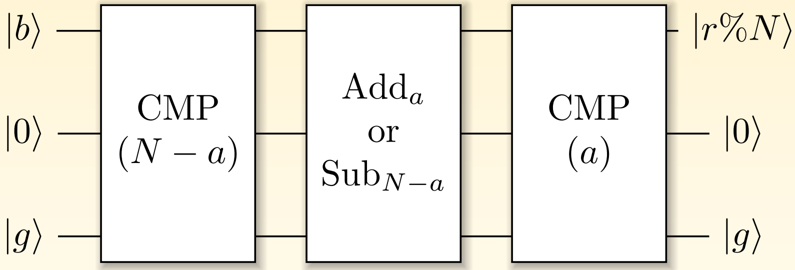
$$\begin{aligned} ax \bmod N &= a \cdot (2^m x_m + \cdots + 2x_1 + x_0) \bmod N \\ &= ((2^m a) \bmod N) x_m \oplus \cdots \oplus ((2a) \bmod N) x_1 \oplus a x_0 \end{aligned}$$

can be done using $m + 1$ controlled modular additions

Controlled multiplication $\Rightarrow$ Doubly-controlled modular additions

How to perform modular additions?

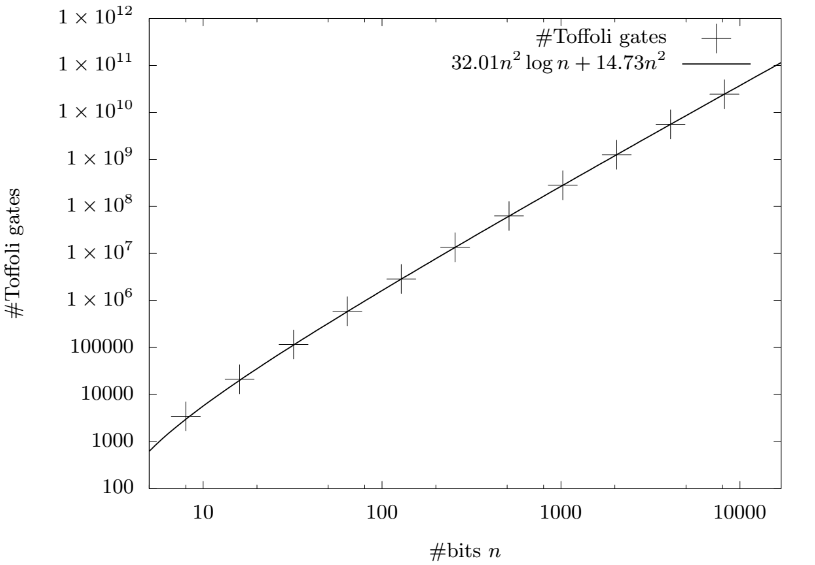
Modular addition: requires 3 integer additions



Ultimately, the *b* register holds the value $r = a + b$ mod *N*, where *a* is the constant to be added.

This method was used in van Meter and Itoh [4] and Takahashi and Kunihiro [5].

Scaling results (modular multiplication)



Resource estimates for Shor's algorithm

	Takahashi et al	Our implementation
Runtime (exact)	$\Theta(n^4 \log \frac{1}{\varepsilon})$	$\Theta(n^3 \log n)$
Runtime (approx.)	$\Theta(n^3 \log \frac{n}{\epsilon} \log \frac{1}{\varepsilon})$	n/a
Depth	$\Theta(n^3 \log \frac{1}{\varepsilon})$	$\Theta(n^3)$
Space	$2n + 2$	$2n + 2$

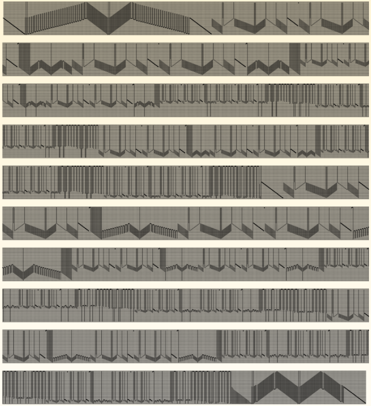
Bonus feature: Our modular multiplication circuit can be tested and debugged efficiently!

Resource estimates for Shor's algorithm

	Takahashi et al	Our implementation
Runtime (exact)	$\Theta(n^4 \log \frac{1}{\varepsilon})$	$\Theta(n^3 \log n)$
Runtime (approx.)	$\Theta(n^3 \log \frac{n}{\epsilon} \log \frac{1}{\varepsilon})$	n/a
Depth	$\Theta(n^3 \log \frac{1}{\varepsilon})$	$\Theta(n^3)$
Space	$2n + 2$	$2n + 2$

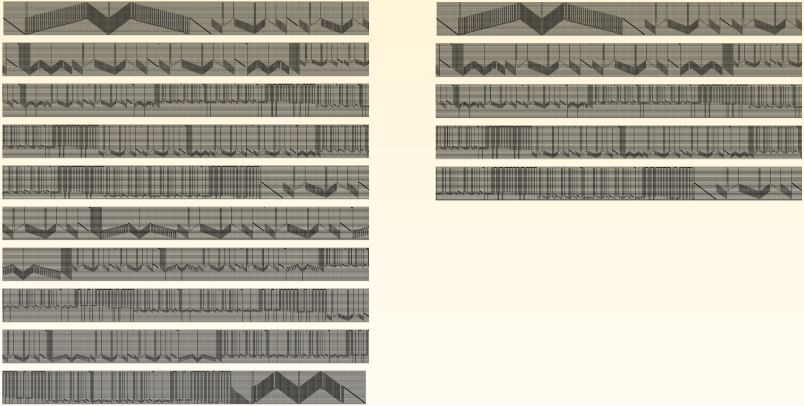
Bonus feature: Our modular multiplication circuit can be tested and debugged efficiently!

Toffoli networks: debugging



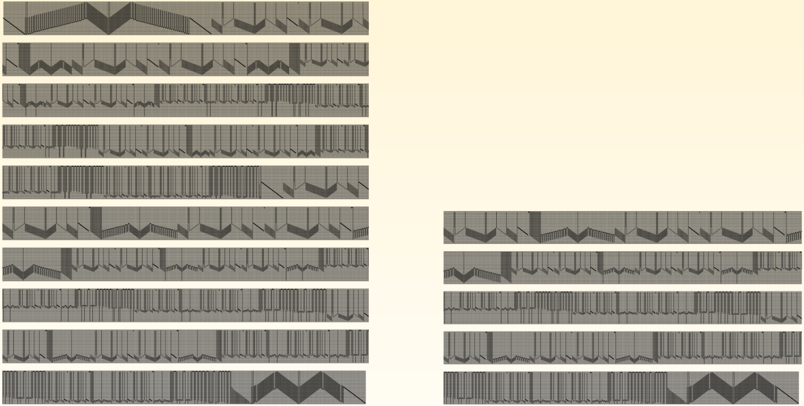
Toffoli network to implement modular addition of the constant value $65,521$ to a 16-qubit register.

Toffoli networks: debugging



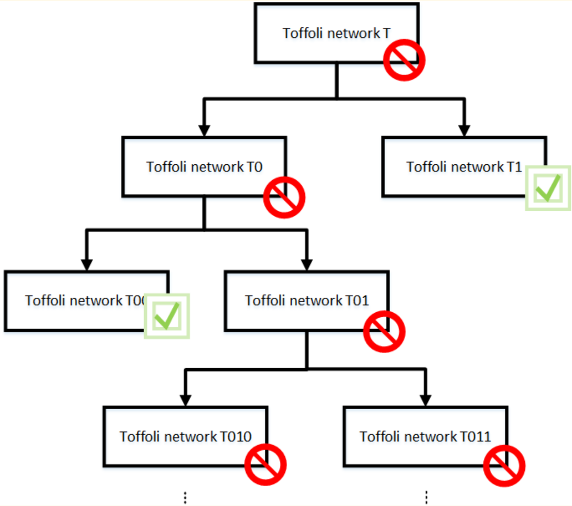
Toffoli network to implement modular addition of the constant value $65,521$ to a 16-qubit register. Partial executions.

Toffoli networks: debugging



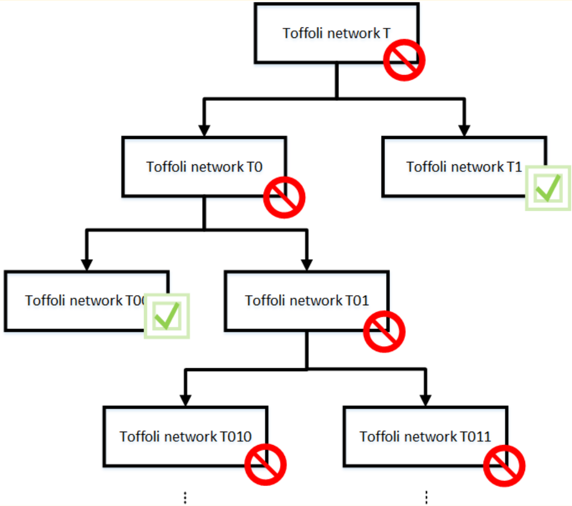
Toffoli network to implement modular addition of the constant value $65,521$ to a 16-qubit register. Partial executions.

Hierarchical debugging of Toffoli circuits



Remark: If chosen test vectors trigger all faults that might be present in the circuit, this method allows to localize of all faults.

Hierarchical debugging of Toffoli circuits



Remark: If chosen test vectors trigger all faults that might be present in the circuit, this method allows to localize of all faults.

References

Thomas G Draper.
Addition on a quantum computer.
arXiv preprint quant-ph/0008033, 2000.

Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro.
Quantum addition circuits and unbounded fan-out.
arXiv preprint arXiv:0910.2530, 2009.

Craig Gidney.
Creating bigger controlled nots from Toffoli gates, without workspace.
`http://cs.stackexchange.com/questions/40933/`, 2015.

Rodney van Meter and Kohei M. Itoh.
Fast quantum modular exponentiation.
Physical Review A, 71:052320, 2005.

Yasuhiro Takahashi and Noboru Kunihiro.
A quantum circuit for Shor's factoring algorithm using 2n+ 2 qubits.
Quantum Information & Computation, 6(2):184–192, 2006.