Practical Quantum Computing

Lecture 09

Quantum Arithmetic: Binary and Fourier

Week	**Tuesday** (3h)			**Wednesday** (3h)			**Deadlines**	
1. The Basics	Introduction	Gates	Circuit Identities	Qiskit	Cirq/Qualtran	Q&A		
	Programming Assignment 1: The basics of a quantum circuit simulator			**Programming Assignment 1:** The building blocks of a quantum circuit simulator				
2. Entanglement and its Applications	Teleportation	Superdense Coding	Quantum Key Distribution	Qualtran/ Assignment2	Terminology of Projects	Q&A		
	Programming Assignment 2: The basics of a quantum circuit optimizer			**Programming Assignment 2:** The building blocks of a quantum circuit optimizer				
3. Computing	Phase Kickback and Toffoli	Distinguishing quantum states	The First Algorithms	Invited TBA		Q&A		11 May 2024
4. Advanced Topics*	Arithmetic Circuits*	Fault-Tolerance*	Surface QEC* Grover's Alg*	Invited TBA	Invited TBA	Q&A	18 May 2024	

* not evaluated

Programming Assignment 2 - Quantum Circuit Optimizer

Theory

- Changing the structure of quantum circuits by applying local transformations (circuit identities) leaves the computation unchanged
- The width and depth of a quantum circuit
- The parallel execution of quantum gates

Practice

- Writing Python code for applying circuit identities for reducing:
    - depth of quantum circuit
    - number of quantum gates
- Benchmarking the execution time of the quantum circuit simulator with the optimized circuit

Learning goals - 10 Arithmetic Circuits (Advanced)

1. What you have learned by now
   a. Quantum circuits: mathematics, diagrams and circuit identities
   b. Entanglement: teleportation, quantum games, QKD
   c. Superpositions, Phase Kickback and finding hidden strings
2. **Classical addition using Toffoli gates**
3. **Quantum addition with bits**
   a. Translating classical circuits into quantum circuits
   b. Ripple-Carry Addition
4. **Quantum addition with phases**
   a. Quantum Fourier Transformation - Encoding bits into phases
   b. Adding phases by rotating qubit states
5. **Modular arithmetic**
   a. Implementation using adders and subtractors
   b. Building block for Shor's algorithm

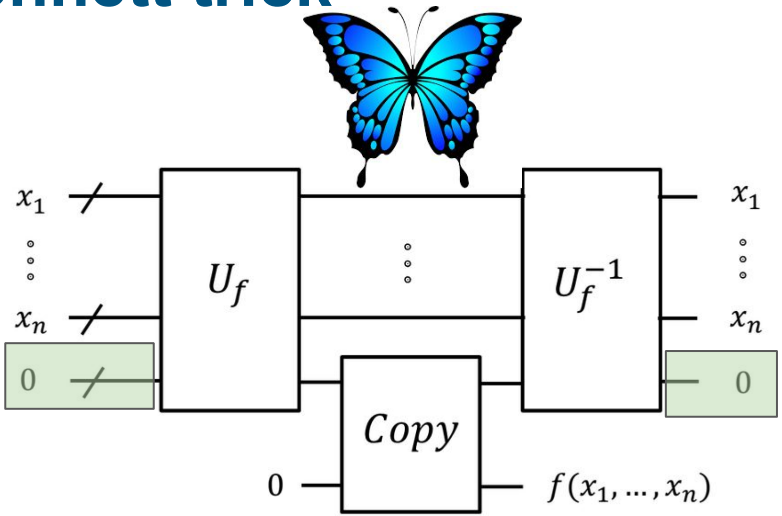
- Deadline for programming Assignment 2
- 18 May 2024

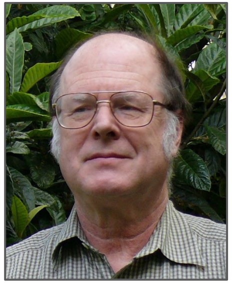
Reversibility and the Bennett trick

Ancilla qubit - “Scratch work”

- used to support the computation
- usually initialised in |0>
- before end of circuit its state to be |0>

Uncomputation and Reversibility





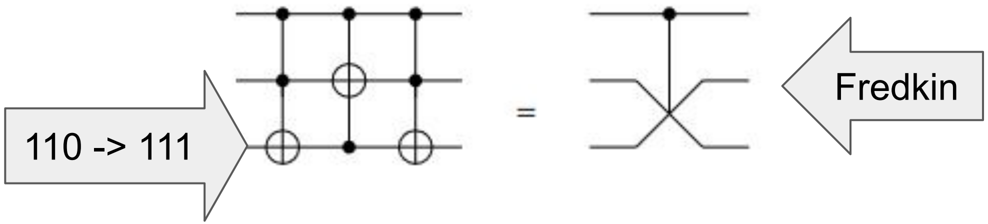
By gar (x), we mean garbage depending on x: that is, “scratch work” that a reversible computation generates along the way to computing some desired function $f(x)$. Typically, the garbage later needs to be *uncomputed*. Uncomputing, a term introduced by Bennett [7], simply means running an entire computation in reverse, after the output $f(x)$ has been safely stored.

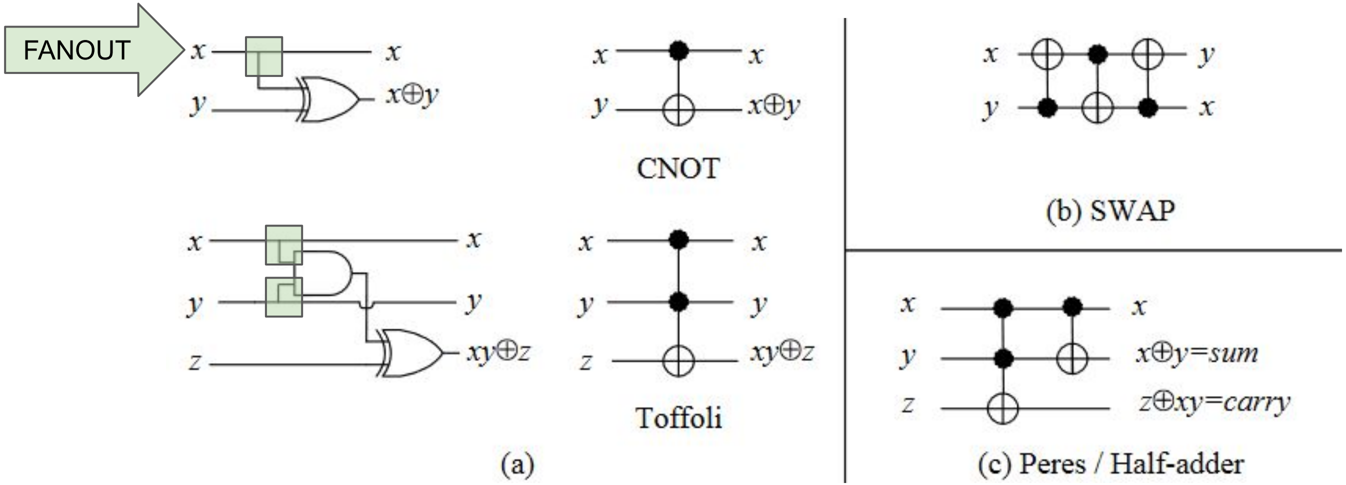
[7] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

Fredkin and Toffoli

Fredkin gates preserve the number of 1s

Toffoli gates do not preserve the number of 1





https://arxiv.org/pdf/1110.2574.pdf

One’s complement

Obtained by inverting all the bits in the binary representation of the number

Negative numbers are represented by the inverse of the binary representations of their corresponding positive numbers

An N-bit ones' complement numeral system

- represent integers in the range $-(2^{N-1}-1)$ to $2^{N-1}-1$
- two's complement can express -2^{N-1} to $2^{N-1}-1$

8-bit ones'-complement integers

Bits	Unsigned value	Ones' complement value
0111 1111	127	127
0111 1110	126	126
0000 0010	2	2
0000 0001	1	1
0000 0000	0	0
1111 1111	255	−0
1111 1110	254	−1
1111 1101	253	−2
1000 0001	129	−126
1000 0000	128	−127

Two's complement

Defined as its complement with respect to 2^N

- **calculated by inverting the bits and adding one**
- For example,
  - the two's complement of 110 is 010
  - because 010 + 110 = 8

Take the ones' complement and add one:

- the sum of a number and its ones' complement is all '1' bits, or $2^N - 1$;
- the sum of a number and its two's complement is 2^N

Subtraction: The advantage of using two's complement is the elimination of examining the signs of the operands to determine whether addition or subtraction is needed

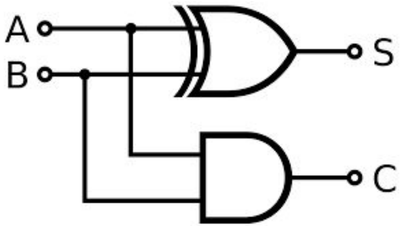
Eight-bit signed integers

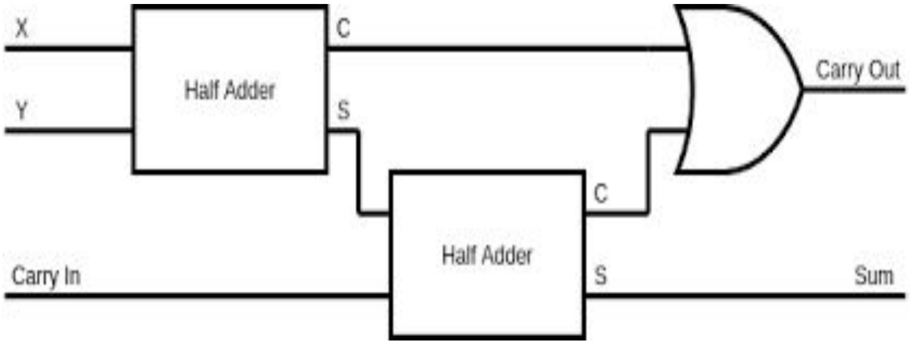
Decimal value	Two's-complement representation
0	0000 0000
1	0000 0001
2	0000 0010
126	0111 1110
127	0111 1111
−128	1000 0000
−127	1000 0001
−126	1000 0010
−2	1111 1110
−1	1111 1111

Half and Full Adder

The half adder

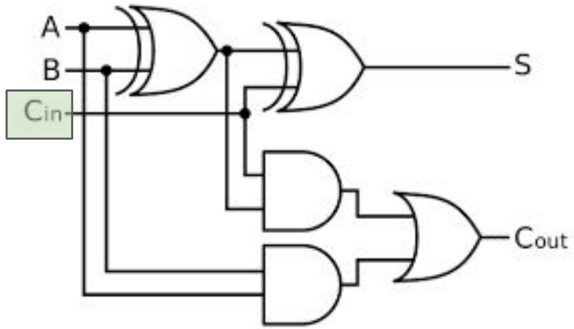
- adds two single binary digits A and B
- has two outputs, sum (S) and carry (C)



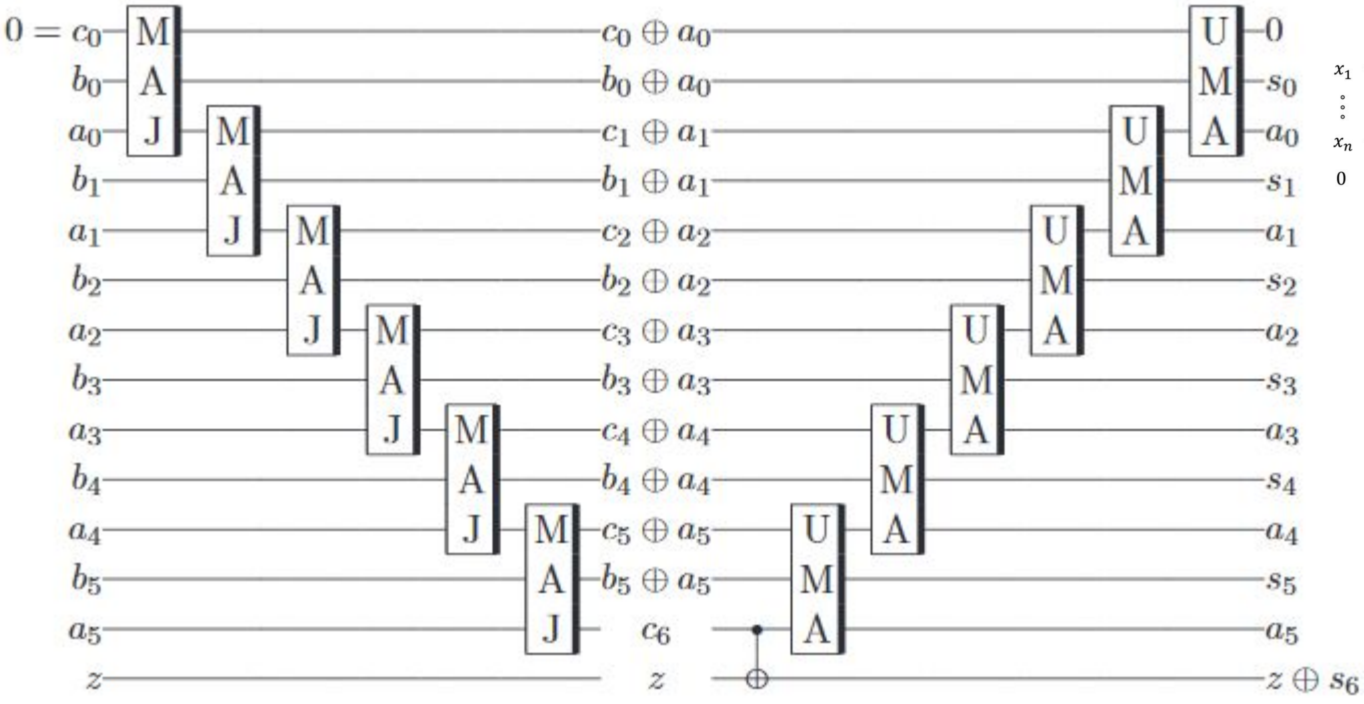


A one-bit full-adder

- adds three one-bit numbers
- A and B are the operands, and Cin is a bit carried in from the previous less-significant stage
- has two outputs, sum (S) and carry (C)



Ripple-Carry Addition



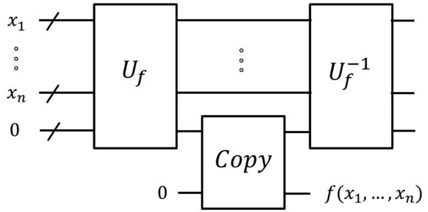
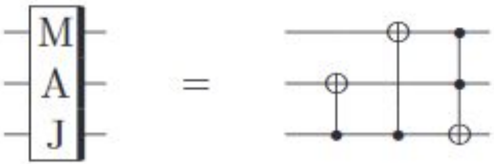


Figure 4: A simple ripple-carry adder for $n = 6$.

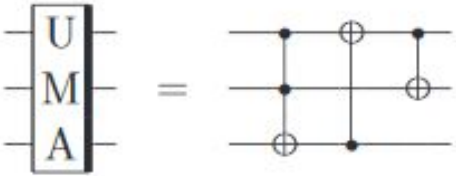
https://arxiv.org/pdf/quant-ph/0410184.pdf

Ripple-Carry Addition

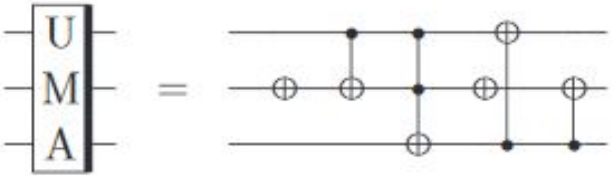


$$\mathrm{MAJ}(a_i, b_i, c_i) = a_i b_i \oplus a_i c_i \oplus b_i c_i.$$

Figure 1: The in-place majority gate MAJ



(a) 2-CNOT version



(b) 3-CNOT version

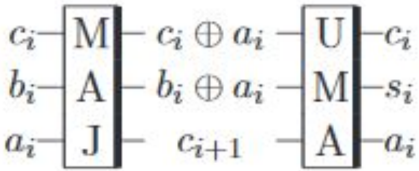


Figure 3: Combining the MAJ and UMA gates

https://arxiv.org/pdf/quant-ph/0410184.pdf

Learning goals - 10 Arithmetic Circuits (Advanced)

1. What you have learned by now
   a. Quantum circuits: mathematics, diagrams and circuit identities
   b. Entanglement: teleportation, quantum games, QKD
   c. Superpositions, Phase Kickback and finding hidden strings
2. **Classical addition using Toffoli gates**
3. **Quantum addition with bits**
   a. Translating classical circuits into quantum circuits
   b. Ripple-Carry Addition
4. **Quantum addition with phases**
   a. Quantum Fourier Transformation - Encoding bits into phases
   b. Adding phases by rotating qubit states
5. **Modular arithmetic**
   a. Implementation using adders and subtractors
   b. Building block for Shor's algorithm

- Deadline for programming Assignment 2
- 18 May 2024

Quantum Fourier Transform

Binary representation of **a** is $a_n a_{n-1} \cdots a_2 a_1$

$$a = a_n 2^{n-1} + a_{n-1} 2^{n-2} + \cdots + a_2 2^1 + a_1 2^0$$

The Fourier transformation of **a** is generating an unentangled state

$$|a\rangle \xrightarrow{F_{2^n}} \frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} e(ak/2^n)|k\rangle.$$

which using

$$|\phi_k(a)\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e(a/2^k)|1\rangle).$$

can be expressed as

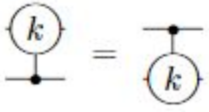
$$\sum_{k=0}^{2^n-1} e(ak/2^n)|k\rangle = |\phi_n(a)\rangle \otimes \cdots \otimes |\phi_2(a)\rangle \otimes |\phi_1(a)\rangle.$$

Quantum Fourier Transformation

$$|\phi_k(a)\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e(a/2^k)|1\rangle).$$

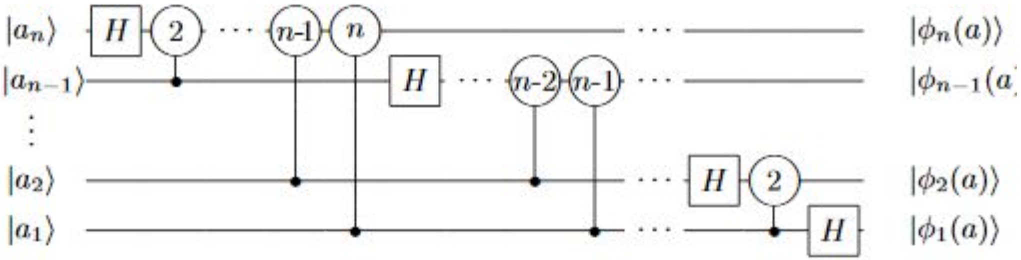
Conditional Rotation

Hadamard Transform

$R_k =$ 

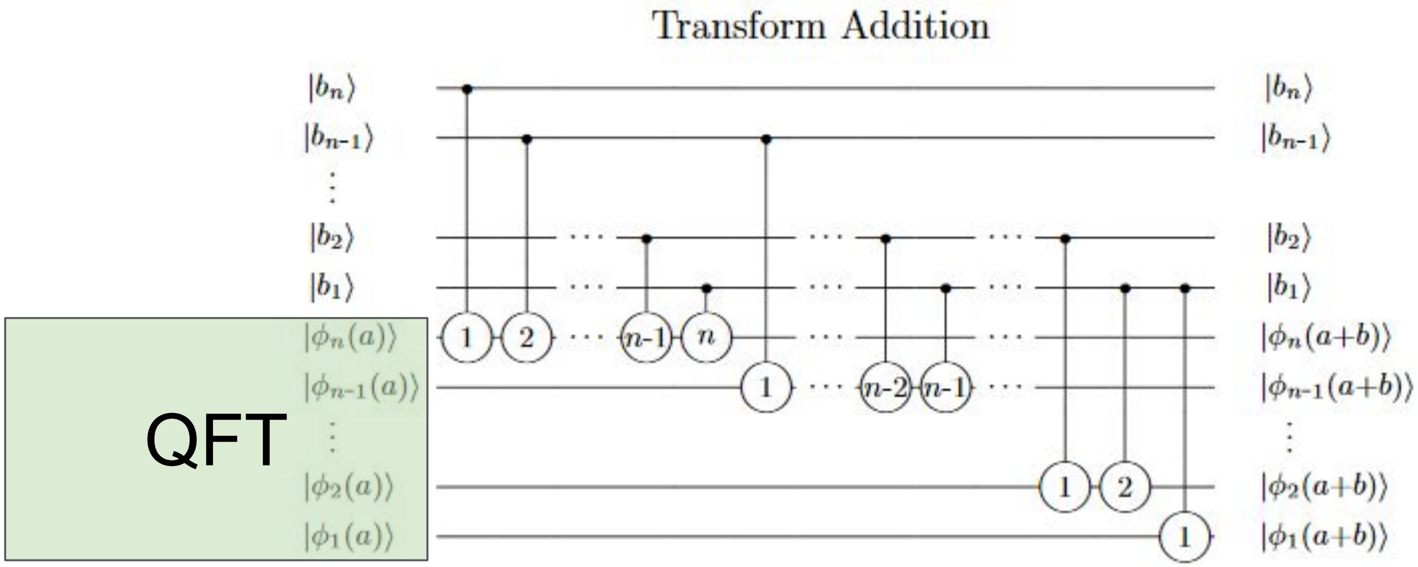
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e(\frac{1}{2^k}) \end{bmatrix} \quad \text{and} \quad \boxed{\mathrm{H}} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Quantum Fourier Transform



https://arxiv.org/pdf/quant-ph/0008033.pdf

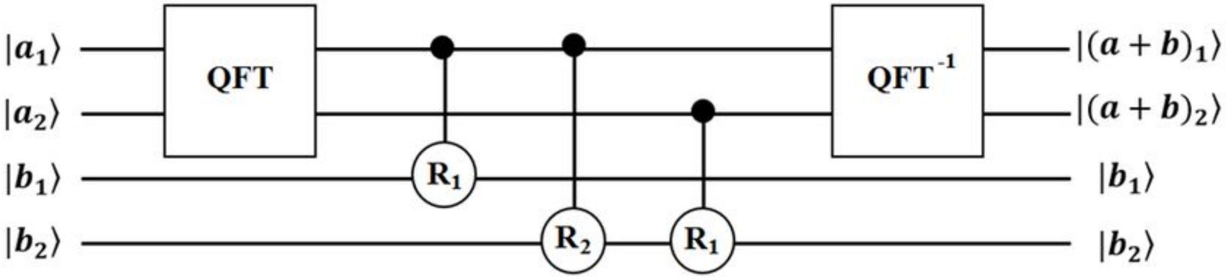
Quantum Fourier Addition

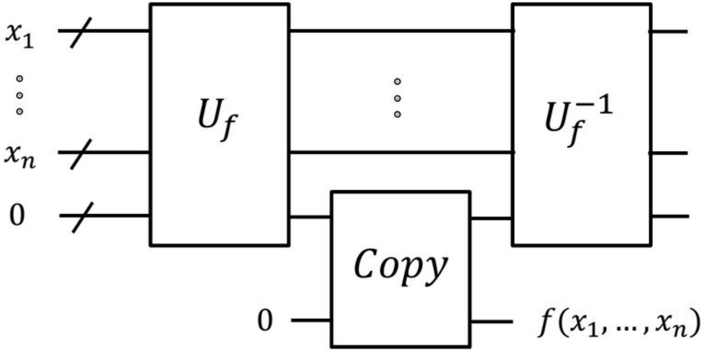


$$
\begin{aligned}
|\phi_n(a)\rangle &\longrightarrow \tfrac{1}{\sqrt{2}}(|0\rangle + e(0.a_n a_{n-1}\ldots a_1 + 0.b_n)|1\rangle) && R_1 \text{ rotation from } b_n \\
&\longrightarrow \tfrac{1}{\sqrt{2}}(|0\rangle + e(0.a_n a_{n-1}\ldots a_1 + 0.b_n b_{n-1})|1\rangle) && R_2 \text{ rotation from } b_{n-1} \\
&\qquad\vdots && \qquad\vdots \\
&\longrightarrow \tfrac{1}{\sqrt{2}}(|0\rangle + e(0.a_n a_{n-1}\ldots a_1 + 0.b_n b_{n-1}\ldots b_1)|1\rangle) && R_n \text{ rotation from } b_1 \\
&= |\phi_n(a+b)\rangle
\end{aligned}
$$

https://arxiv.org/pdf/quant-ph/0008033.pdf

Quantum Fourier Addition vs. Bennett trick





Learning goals - 10 Arithmetic Circuits (Advanced)

1. What you have learned by now
    a. Quantum circuits: mathematics, diagrams and circuit identities
    b. Entanglement: teleportation, quantum games, QKD
    c. Superpositions, Phase Kickback and finding hidden strings
2. **Classical addition using Toffoli gates**
3. **Quantum addition with bits**
    a. Translating classical circuits into quantum circuits
    b. Ripple-Carry Addition
4. **Quantum addition with phases**
    a. Quantum Fourier Transformation - Encoding bits into phases
    b. Adding phases by rotating qubit states
5. **Modular arithmetic**
    a. Implementation using adders and subtractors
    b. Building block for Shor's algorithm

- Deadline for programming Assignment 2
- 18 May 2024

Modular Addition

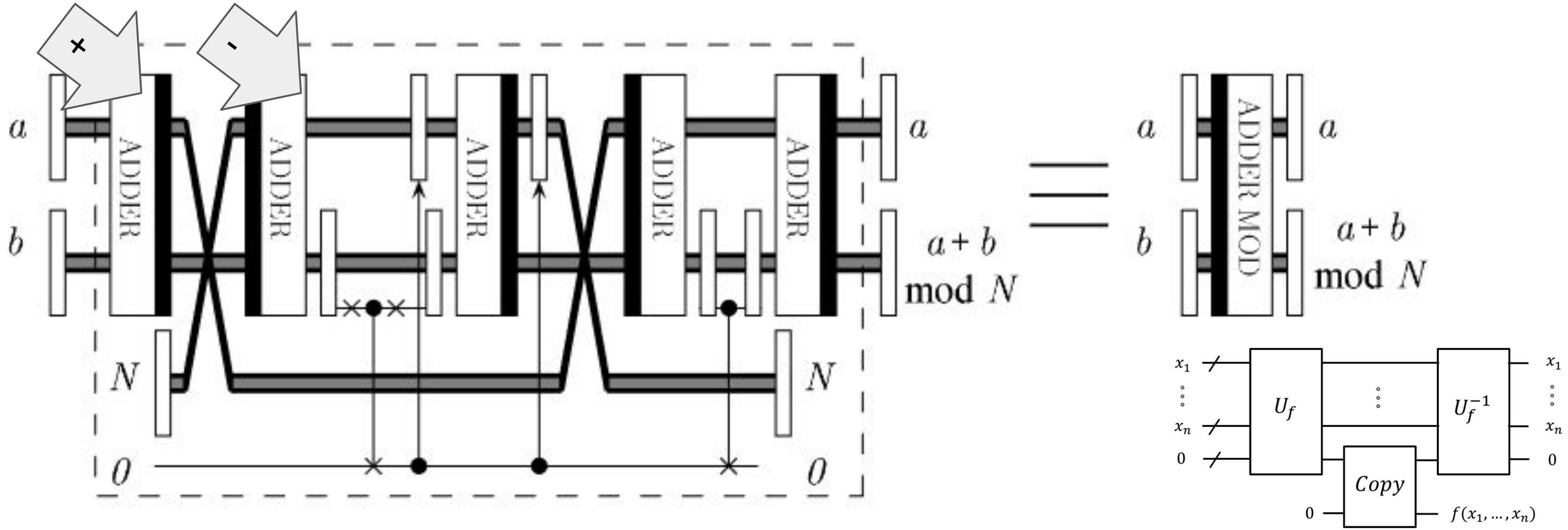


FIG. 4. Adder modulo N. The first and the second network add a and b together and then subtract N. The overflow is recorded into the temporary qubit $|t\rangle$. The next network calculates $(a + b) \bmod N$. At this stage we have extra information about the value of the overflow stored in $|t\rangle$. The last two blocks restore $|t\rangle$ to $|0\rangle$. The arrow before the third plain adder means that the first register is set to $|0\rangle$ if the value of the temporary qubit $|t\rangle$ is 1 and is otherwise left unchanged (this can be easily done with Control–NOT gates, as we know that the first register is in the state $|N\rangle$). The arrow after the third plain adder resets the first register to its original value (here $|N\rangle$). The significance of the thick black bars is explained in the caption of Fig. 2.

https://arxiv.org/pdf/quant-ph/9511018.pdf

Modular Addition

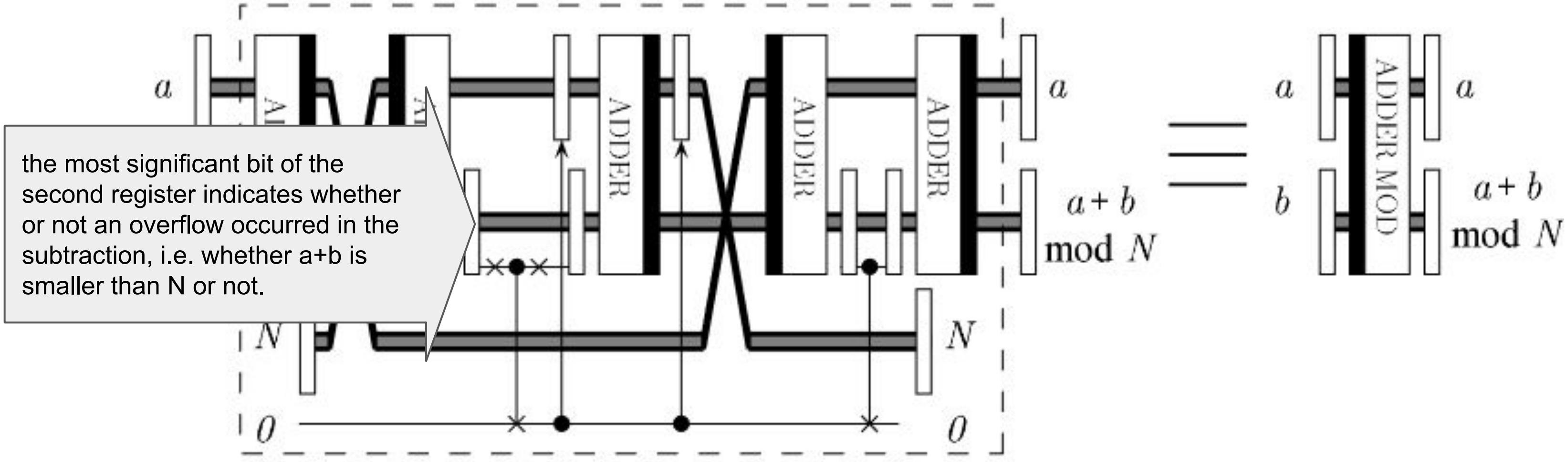
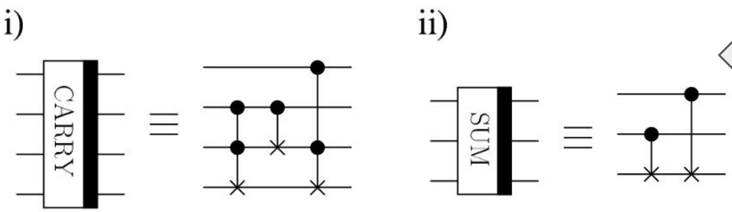


FIG. 4. Adder modulo N. The first and the second network add a and b together and then subtract N. The overflow is recorded into the temporary qubit $|t\rangle$. The next network calculates $(a+b) \bmod N$. At this stage we have extra information about the value of the overflow stored in $|t\rangle$. The last two blocks restore $|t\rangle$ to $|0\rangle$. The arrow before the third plain adder means that the first register is set to $|0\rangle$ if the value of the temporary qubit $|t\rangle$ is 1 and is otherwise left unchanged (this can be easily done with Control–NOT gates, as we know that the first register is in the state $|N\rangle$). The arrow after the third plain adder resets the first register to its original value (here $|N\rangle$). The significance of the thick black bars is explained in the caption of Fig. 2.

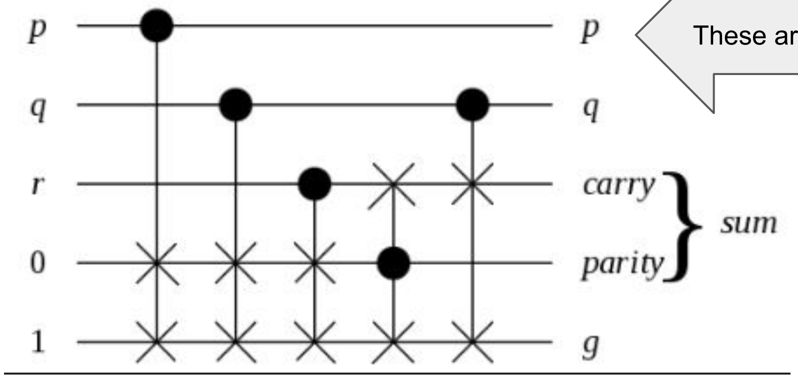
https://arxiv.org/pdf/quant-ph/9511018.pdf

Appendix

CNOT, Toffoli and Fredkin gates for addition



X symbol is for the X-gate

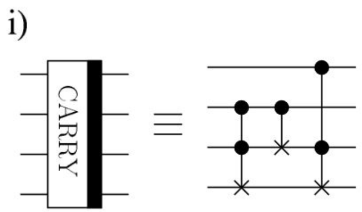


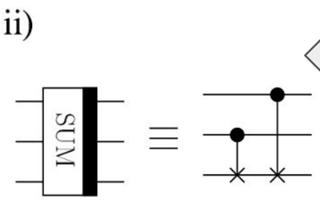
These are Fredkin gates

			parity	g			carry	
p	**q**	**r**	**0**	**1**	**r**	**1**	**r**	**g**
0	0	0	0	1	0	1	0	1
0	0	1	**1**	**0**	**0**	**1**	0	1
0	1	0	**1**	**0**	0	0	0	0
0	1	1	0	1	1	1	1	1
1	0	0	**1**	**0**	0	0	0	0
1	0	1	0	1	1	1	1	1
1	1	0	0	1	0	1	**1**	**0**
1	1	1	**1**	**0**	**0**	**1**	**1**	**0**

The "g" garbage output bit is (p NOR q) if r=0, and (p NAND q) if r=1.

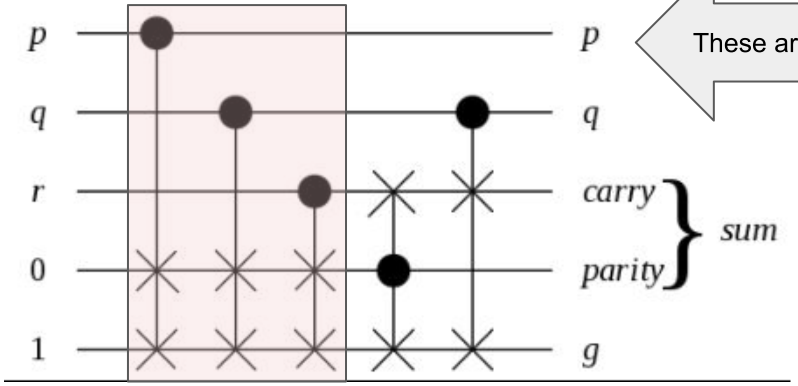
CNOT, Toffoli and Fredkin gates for addition





X symbol is for the X-gate

QRAM

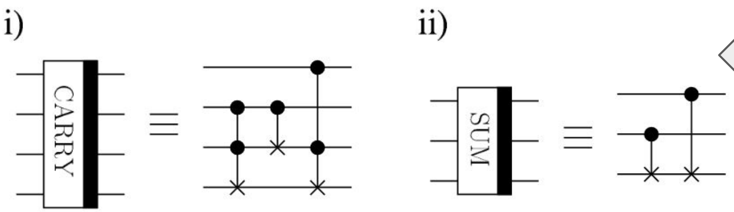


These are Fredkin gates

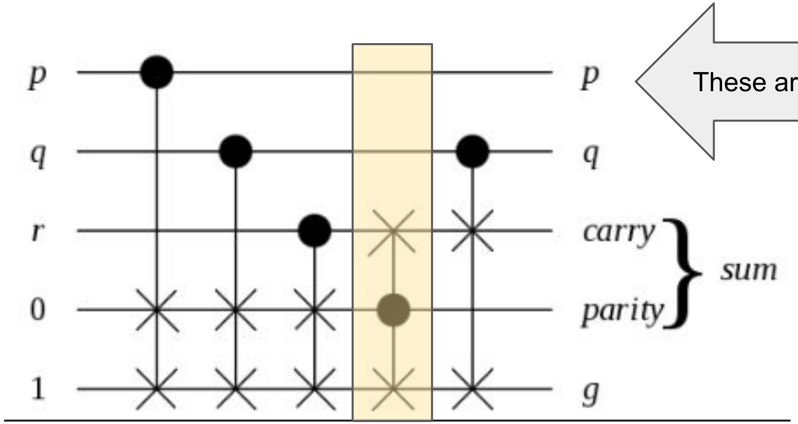
			parity	g			carry	
p	**q**	**r**	**0**	**1**	**r**	**1**	**r**	**g**
0	0	0	0	1	0	1	0	1
0	0	1	**1**	**0**	**0**	**1**	0	1
0	1	0	**1**	**0**	0	0	0	0
0	1	1	0	1	1	1	1	1
1	0	0	**1**	**0**	0	0	0	0
1	0	1	0	1	1	1	1	1
1	1	0	0	1	0	1	**1**	**0**
1	1	1	**1**	**0**	**0**	**1**	**1**	**0**

The "g" garbage output bit is (p NOR q) if r=0, and (p NAND q) if r=1.

CNOT, Toffoli and Fredkin gates for addition



X symbol is for the X-gate

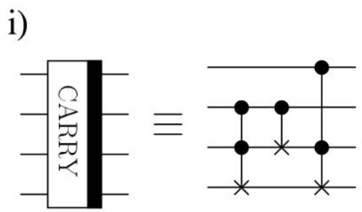


These are Fredkin gates

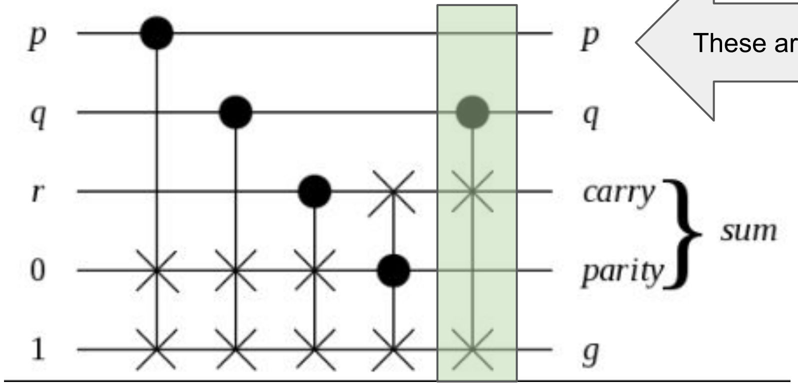
			parity	g			carry	
p	q	r	0	1	r	1	r	g
0	0	0	0	1	0	1	0	1
0	0	1	**1**	**0**	**0**	**1**	0	1
0	1	0	**1**	**0**	0	0	0	0
0	1	1	0	1	1	1	1	1
1	0	0	**1**	**0**	0	0	0	0
1	0	1	0	1	1	1	1	1
1	1	0	0	1	0	1	**1**	**0**
1	1	1	**1**	**0**	**0**	**1**	**1**	**0**

The "g" garbage output bit is (p NOR q) if r=0, and (p NAND q) if r=1.

CNOT, Toffoli and Fredkin gates for addition



X symbol is for the X-gate



These are Fredkin gates

			parity	g			carry	
p	**q**	**r**	**0**	**1**	**r**	**1**	**r**	**g**
0	0	0	0	1	0	1	0	1
0	0	1	**1**	**0**	**0**	**1**	0	1
0	1	0	**1**	**0**	0	0	0	0
0	1	1	0	1	1	1	1	1
1	0	0	**1**	**0**	0	0	0	0
1	0	1	0	1	1	1	1	1
1	1	0	0	1	0	1	**1**	**0**
1	1	1	**1**	**0**	**0**	**1**	**1**	**0**

The "g" garbage output bit is (p NOR q) if r=0, and (p NAND q) if r=1.