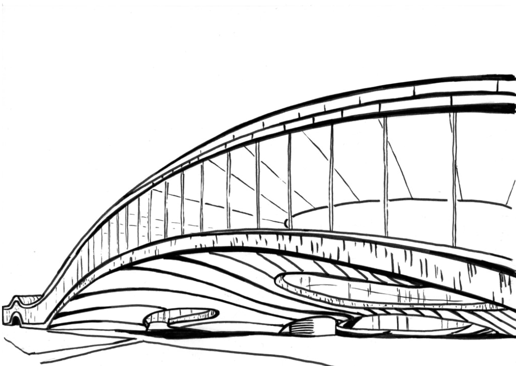
Logic Synthesis for Quantum Computing

Mathias Soeken, Alan Mishchenko, Luca Amarù, Robert K. Brayton

Integrated Systems Laboratory, EPFL, Switzerland

lsi.epfl.ch • msoeken.github.io



Quantum computing is getting real

- 17-qubit quantum computer from IBM based on superconducting qubits (16-qubit version available via cloud service)
- 9-qubit quantum computer from Google based on superconducting circuits
- 5-qubit quantum computer at University of Maryland based on ion traps
- Microsoft is investigating topological quantum computers
- Intel is investigating silicon-based qubits

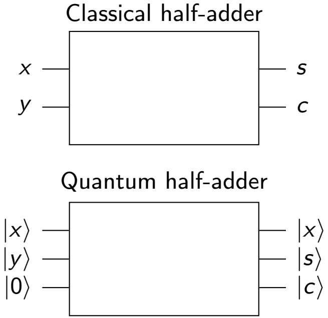
Quantum computing is getting real

- 17-qubit quantum computer from IBM based on superconducting qubits (16-qubit version available via cloud service)
- 9-qubit quantum computer from Google based on superconducting circuits
- 5-qubit quantum computer at University of Maryland based on ion traps
- Microsoft is investigating topological quantum computers
- Intel is investigating silicon-based qubits

- "Quantum supremacy" experiment may be possible with ≈50 qubits (45-qubit simulation has been performed classically)
- Smallest practical problems require ≈100 qubits

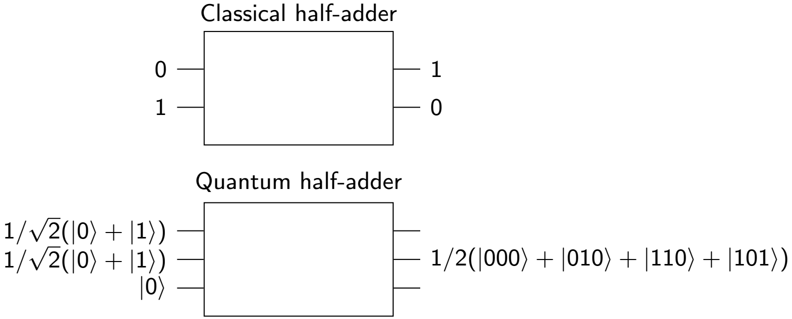
Challenges in logic synthesis for quantum computing

1. Quantum computers process qubits not bits



Challenges in logic synthesis for quantum computing

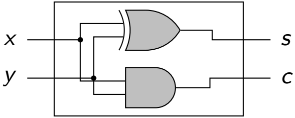
1. Quantum computers process qubits not bits



Challenges in logic synthesis for quantum computing

1. Quantum computers process qubits not bits
2. All qubit operations, called quantum gates, must be reversible

Classical half-adder

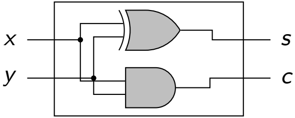


Quantum half-adder

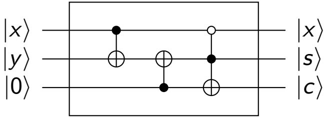
Challenges in logic synthesis for quantum computing

1. Quantum computers process qubits not bits
2. All qubit operations, called quantum gates, must be reversible

Classical half-adder



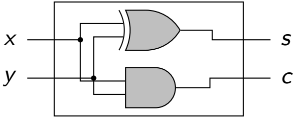
Quantum half-adder



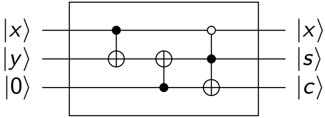
Challenges in logic synthesis for quantum computing

1. Quantum computers process qubits not bits
2. All qubit operations, called quantum gates, must be reversible
3. Standard gate library for today's physical quantum computers is non-trivial

Classical half-adder



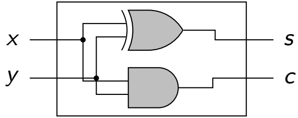
Quantum half-adder



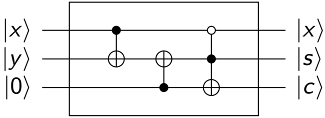
Challenges in logic synthesis for quantum computing

1. Quantum computers process qubits not bits
2. All qubit operations, called quantum gates, must be reversible
3. Standard gate library for today's physical quantum computers is non-trivial
4. Circuit is not allowed to produce intermediate results, called garbage qubits

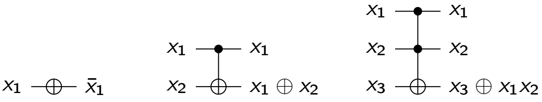
Classical half-adder



Quantum half-adder

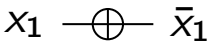


Reversible gates

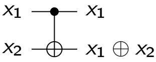


NOT CNOT Toffoli

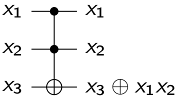
Reversible gates



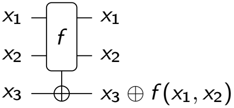
NOT



CNOT

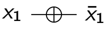


Toffoli

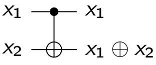


Single-target

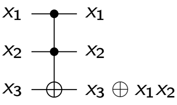
Reversible gates



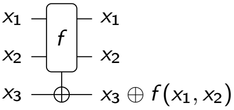
NOT



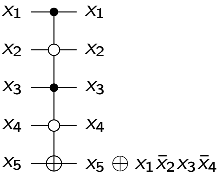
CNOT



Toffoli

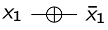


Single-target

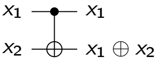


Multiple-controlled Toffoli

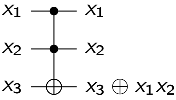
Reversible gates



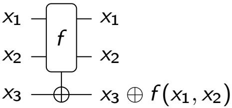
NOT



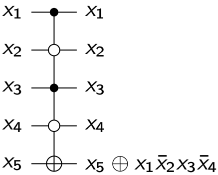
CNOT



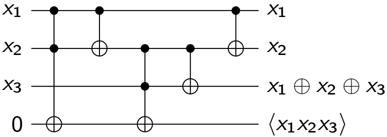
Toffoli



Single-target



Multiple-controlled Toffoli



Full adder

Quantum gates

- Qubit is vector $|\varphi\rangle = \left(\begin{smallmatrix}\alpha\\\beta\end{smallmatrix}\right)$ with $|\alpha^2| + |\beta^2| = 1$.
- Classical 0 is $|0\rangle = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$; Classical 1 is $|1\rangle = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$

Quantum gates

- Qubit is vector $|\varphi\rangle = \binom{\alpha}{\beta}$ with $|\alpha^2| + |\beta^2| = 1$.
- Classical 0 is $|0\rangle = \binom{1}{0}$; Classical 1 is $|1\rangle = \binom{0}{1}$

$$\begin{matrix} |\varphi_1\rangle \\ |\varphi_2\rangle \end{matrix} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} |\varphi_1\varphi_2\rangle$$

CNOT

$$|\varphi\rangle \; -\boxed{H}- \; \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} |\varphi\rangle$$

Hadamard

$$|\varphi\rangle \; -\boxed{T}- \; \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} |\varphi\rangle$$

T

Composing quantum gates

- Applying a quantum gate to a quantum state (matrix-vector multiplication)

$$|\varphi\rangle = \left(\begin{smallmatrix}\alpha\\ \beta\end{smallmatrix}\right) -\boxed{U}- U|\varphi\rangle$$

Composing quantum gates

- Applying a quantum gate to a quantum state (matrix-vector multiplication)

$$|\varphi\rangle = \left(\begin{smallmatrix}\alpha \\ \beta\end{smallmatrix}\right) \;-\boxed{U}-\; U|\varphi\rangle$$

- Applying quantum gates in sequence (matrix product)

$$|\varphi\rangle = \left(\begin{smallmatrix}\alpha \\ \beta\end{smallmatrix}\right) \;-\boxed{U_1}-\boxed{U_2}-\; (U_2 U_1)|\varphi\rangle$$

Composing quantum gates

- Applying a quantum gate to a quantum state (matrix-vector multiplication)

$$|\varphi\rangle = \left(\begin{smallmatrix}\alpha \\ \beta\end{smallmatrix}\right) -\boxed{U}- \; U|\varphi\rangle$$

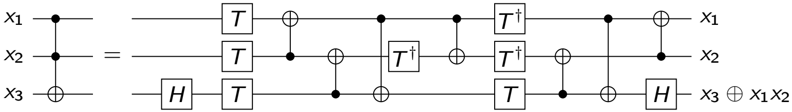
- Applying quantum gates in sequence (matrix product)

$$|\varphi\rangle = \left(\begin{smallmatrix}\alpha \\ \beta\end{smallmatrix}\right) -\boxed{U_1}-\boxed{U_2}- \; (U_2 U_1)|\varphi\rangle$$

- Applying quantum gates in parallel (Kronecker product)

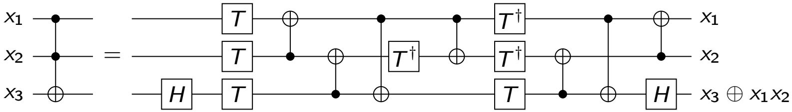
$$\left.\begin{array}{l} |\varphi_1\rangle = \left(\begin{smallmatrix}\alpha_1 \\ \beta_1\end{smallmatrix}\right) -\boxed{U_1}- \\ |\varphi_2\rangle = \left(\begin{smallmatrix}\alpha_2 \\ \beta_2\end{smallmatrix}\right) -\boxed{U_2}- \end{array}\right\} (U_1 \otimes U_2)|\varphi_1\varphi_2\rangle$$

Mapping Toffoli gates

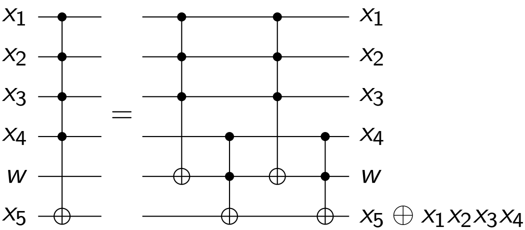


Clifford+T circuit [Amy *et al.*, *TCAD* **32**, 2013]

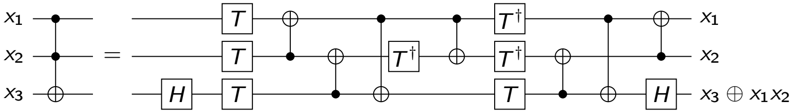
Mapping Toffoli gates



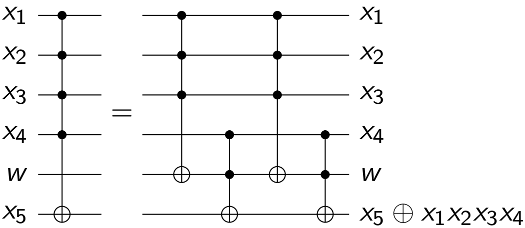
Clifford+T circuit [Amy *et al.*, *TCAD* **32**, 2013]



Mapping Toffoli gates



Clifford+T circuit [Amy *et al.*, *TCAD* **32**, 2013]



Costs are number of qubits and number of T gates

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

1. Represent input function as classical logic network and optimize it

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

1. Represent input function as classical logic network and optimize it
2. Map network into k-LUT network

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

1. Represent input function as classical logic network and optimize it
2. Map network into k-LUT network
3. Translate k-LUT network into reversible network with k-input single-target gates

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

1. Represent input function as classical logic network and optimize it
2. Map network into k-LUT network
3. Translate k-LUT network into reversible network with k-input single-target gates
4. Map single-target gates into Clifford+T networks

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

conv. alg.

1. Represent input function as classical logic network and optimize it
2. Map network into k-LUT network
3. Translate k-LUT network into reversible network with k-input single-target gates
4. Map single-target gates into Clifford+T networks

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

conv. alg.

1. Represent input function as classical logic network and optimize it
2. Map network into k-LUT network

new alg.

3. Translate k-LUT network into reversible network with k-input single-target gates
4. Map single-target gates into Clifford+T networks

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

conv. alg.

1. Represent input function as classical logic network and optimize it
2. Map network into k-LUT network

new alg.

3. Translate k-LUT network into reversible network with k-input single-target gates
4. Map single-target gates into Clifford+T networks

affects #qubits

LUT-based hierarchical reversible synthesis

Goal: Automatically synthesizing large Boolean functions into Clifford+T networks of reasonable quality (qubits and T-count)

Algorithm: LUT-based hierarchical reversible synthesis (LHRS)

conv. alg.

1. Represent input function as classical logic network and optimize it
2. Map network into k-LUT network

new alg.

3. Translate k-LUT network into reversible network with k-input single-target gates
4. Map single-target gates into Clifford+T networks

affects #qubits

affects # T gates

LUT mapping

- Realizing a logic function or logic circuit in terms of a k-LUT logic network
- A k-LUT is any Boolean function with at most k inputs
- One of the most effective methods used in logic synthesis

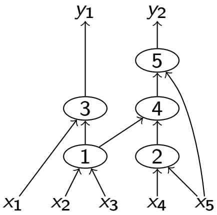
LUT mapping

- Realizing a logic function or logic circuit in terms of a k-LUT logic network
- A k-LUT is any Boolean function with at most k inputs
- One of the most effective methods used in logic synthesis
- Typical objective functions are size (number of LUTs) and depth (longest path from inputs to outputs)
- Open source software ABC can generate industrial-scale mappings

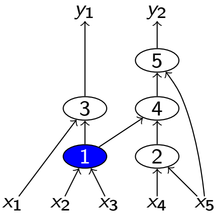
LUT mapping

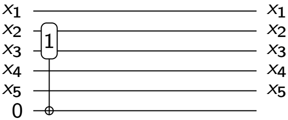
- Realizing a logic function or logic circuit in terms of a k-LUT logic network
- A k-LUT is any Boolean function with at most k inputs
- One of the most effective methods used in logic synthesis
- Typical objective functions are size (number of LUTs) and depth (longest path from inputs to outputs)
- Open source software ABC can generate industrial-scale mappings
- Can be used as technology mapper for FPGAs (e.g., when $k \leq 7$)

k-LUT network to reversible network



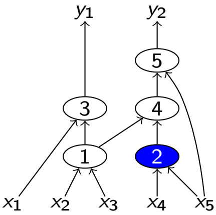
k-LUT network to reversible network





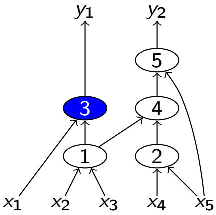
! k-LUT corresponds to k-controlled single-target gate

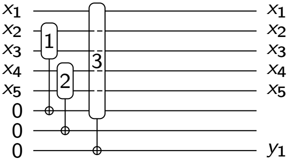
k-LUT network to reversible network



! k-LUT corresponds to k-controlled single-target gate

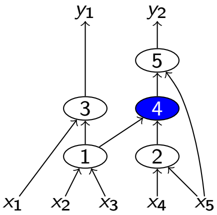
k-LUT network to reversible network

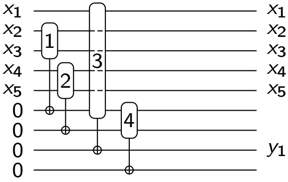




! k-LUT corresponds to k-controlled single-target gate

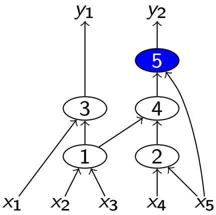
k-LUT network to reversible network

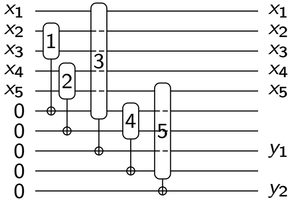




! k-LUT corresponds to k-controlled single-target gate

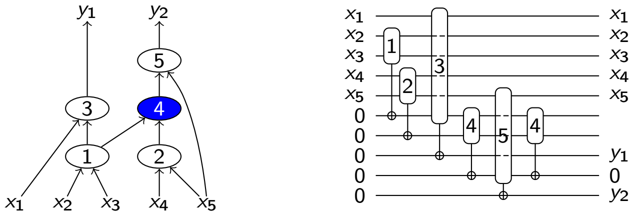
k-LUT network to reversible network





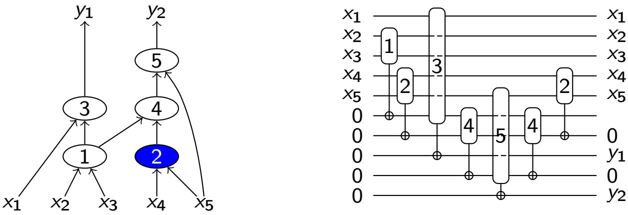
! k-LUT corresponds to k-controlled single-target gate

k-LUT network to reversible network



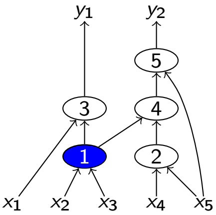
- **!** k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed

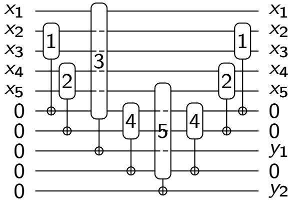
k-LUT network to reversible network



- **!** k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed

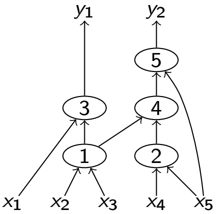
k-LUT network to reversible network

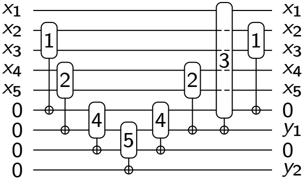




- ! k-LUT corresponds to k-controlled single-target gate
- ▶ non-output LUTs need to be uncomputed

k-LUT network to reversible network

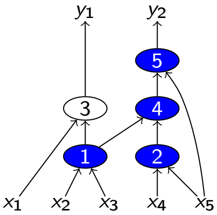


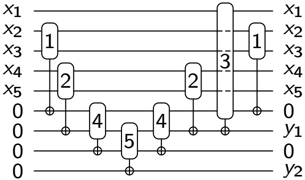


! k-LUT corresponds to k-controlled single-target gate

- non-output LUTs need to be uncomputed
- order of LUT traversal determines number of ancillas

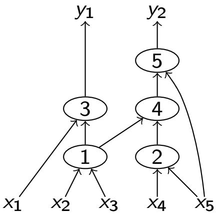
k-LUT network to reversible network

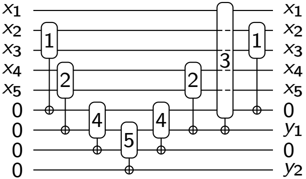




- **!** k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed
- order of LUT traversal determines number of ancillas
- maximum output cone determines minimum number of ancillas

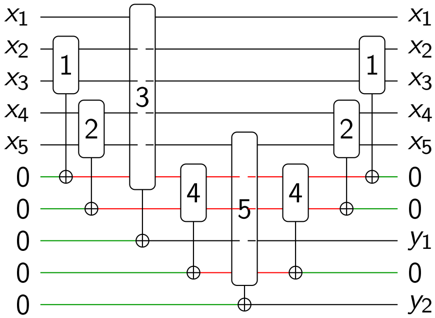
k-LUT network to reversible network





- ! k-LUT corresponds to k-controlled single-target gate
- ▶ non-output LUTs need to be uncomputed
- ▶ order of LUT traversal determines number of ancillas
- ▶ maximum output cone determines minimum number of ancillas
- ☺ fast mapping that generates a fixed-space skeleton for subnetwork synthesis

Single-target gate LUT mapping



- Mapping problem: Given a single-target gate $\mathrm{T}_f(X, x_t)$ (with control function f, control lines X, and target line x_t), a set of clean ancillas X_c, and a set of dirty ancillas X_d, find the best mapping into a Clifford+T network, such that all ancillas are restored to their original value.

Single-target gate mapping algorithms

- Direct

Single-target gate mapping algorithms

- Direct
  - Map each control function using ESOP based synthesis

Single-target gate mapping algorithms

- Direct
  - Map each control function using ESOP based synthesis
  - Does not use ancillae

Single-target gate mapping algorithms

- Direct
  - Map each control function using ESOP based synthesis
  - Does not use ancillae
- LUT-based

Single-target gate mapping algorithms

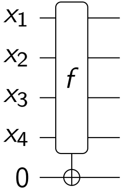
- Direct
  - Map each control function using ESOP based synthesis
  - Does not use ancillae
- LUT-based
  - Map control function into smaller LUT network

Single-target gate mapping algorithms

- Direct
  - Map each control function using ESOP based synthesis
  - Does not use ancillae
- LUT-based
  - Map control function into smaller LUT network
  - Map small LUTs into pre-computed optimum quantum circuits

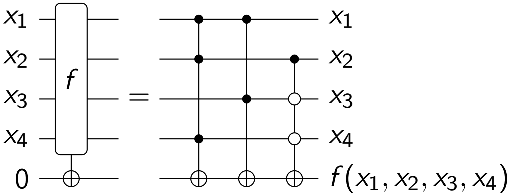
Direct mapping

$$
\begin{aligned}
f(x_1, x_2, x_3, x_4) &= [(x_4 x_3 x_2 x_1)_2 \text{ is prime}] \\
&= \bar{x}_4 \bar{x}_3 x_2 \vee \bar{x}_4 x_3 x_1 \vee x_4 \bar{x}_3 x_2 x_1 \vee x_4 x_3 \bar{x}_2 x_1
\end{aligned}
$$



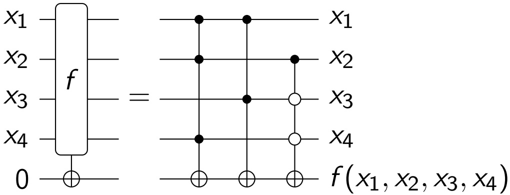
Direct mapping

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= [(x_4 x_3 x_2 x_1)_2 \text{ is prime}] \\ &= \bar{x}_4 \bar{x}_3 x_2 \vee \bar{x}_4 x_3 x_1 \vee x_4 \bar{x}_3 x_2 x_1 \vee x_4 x_3 \bar{x}_2 x_1 \\ &= x_4 x_2 x_1 \oplus x_3 x_1 \oplus \bar{x}_4 \bar{x}_3 x_2 \end{aligned}$$



Direct mapping

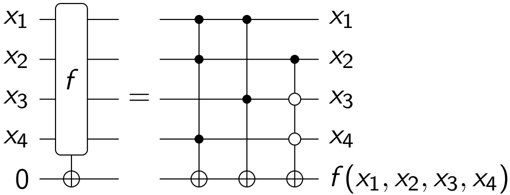
$$
\begin{aligned}
f(x_1, x_2, x_3, x_4) &= [(x_4x_3x_2x_1)_2 \text{ is prime}] \\
&= \bar{x}_4\bar{x}_3x_2 \vee \bar{x}_4x_3x_1 \vee x_4\bar{x}_3x_2x_1 \vee x_4x_3\bar{x}_2x_1 \\
&= x_4x_2x_1 \oplus x_3x_1 \oplus \bar{x}_4\bar{x}_3x_2
\end{aligned}
$$



- Each multiple-controlled Toffoli gate is mapped to Clifford+T

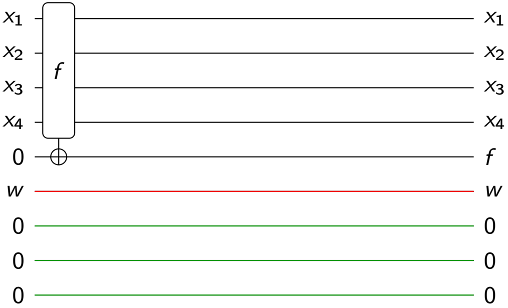
Direct mapping

$$
\begin{aligned}
f(x_1, x_2, x_3, x_4) &= [(x_4x_3x_2x_1)_2 \text{ is prime}] \\
&= \bar{x}_4\bar{x}_3x_2 \lor \bar{x}_4x_3x_1 \lor x_4\bar{x}_3x_2x_1 \lor x_4x_3\bar{x}_2x_1 \\
&= x_4x_2x_1 \oplus x_3x_1 \oplus \bar{x}_4\bar{x}_3x_2
\end{aligned}
$$

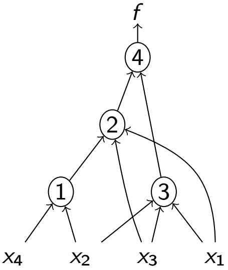


- Each multiple-controlled Toffoli gate is mapped to Clifford+T
- ☹ ESOP minimization tools (e.g., exorcism) optimize for cube count

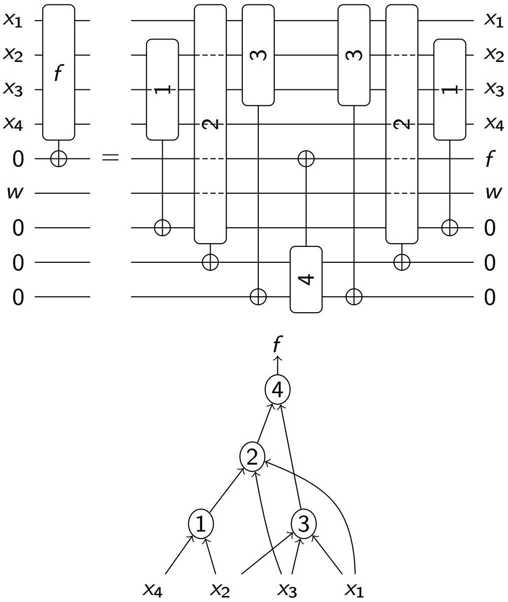
LUT-based single-target gate mapping



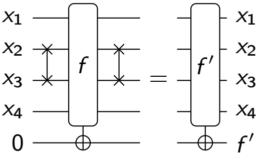
LUT-based single-target gate mapping



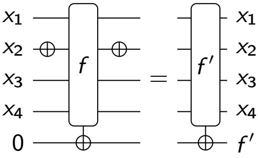
LUT-based single-target gate mapping



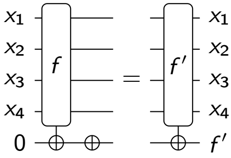
Exploiting Boolean function classification



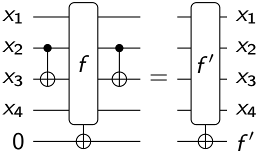
swap inputs



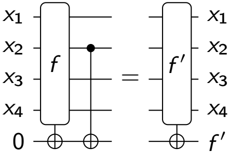
invert inputs



invert output



spectral translation



disjoint spectral translation

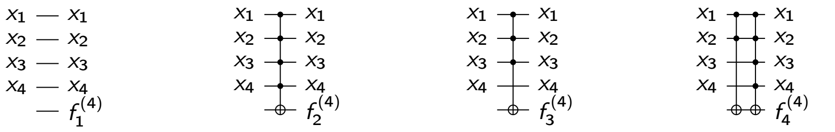
- Operations do not influence T-count of the quantum circuit
- ☺ All optimum circuits in an equivalence class have the same T-count

Classification of all 4-input functions

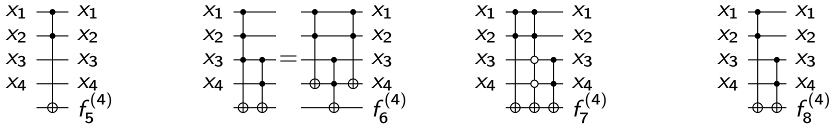
- All 65,356 4-input functions collapse into only 8 equivalence classes
- Classification simple by comparing coefficients in the function's Walsh spectrum

Classification of all 4-input functions

- All 65,356 4-input functions collapse into only 8 equivalence classes
- Classification simple by comparing coefficients in the function's Walsh spectrum



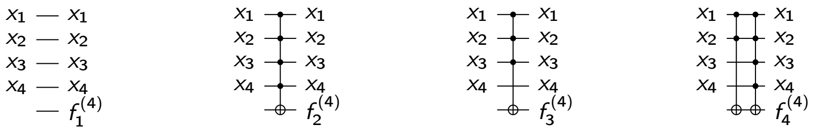
$\perp$ | $x_1x_2x_3x_4$ | $x_1x_2x_3$ | $x_1x_2 \oplus x_1x_2x_3x_4$



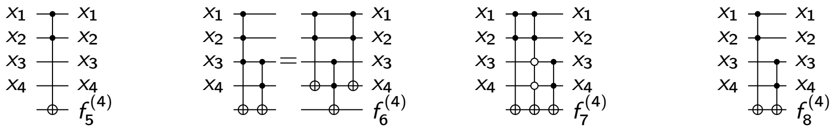
x_1x_2 | $x_1x_2x_3 \oplus x_3x_4$ | $x_1x_2 \oplus x_1x_2\bar{x}_3\bar{x}_4 \oplus x_3x_4$ | $x_1x_2 \oplus x_3x_4$

Classification of all 4-input functions

- All 65,356 4-input functions collapse into only 8 equivalence classes (all 4,294,967,296 5-input functions collapse into 48 classes)
- Classification simple by comparing coefficients in the function's Walsh spectrum (and auto-correlation spectrum)



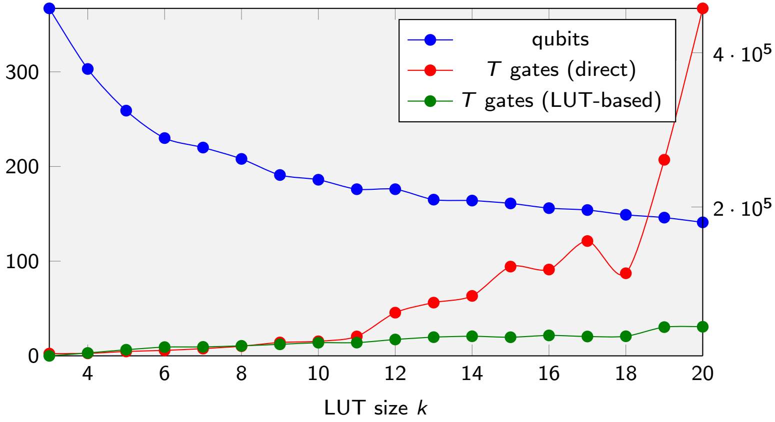
$\perp$ $\qquad$ $x_1x_2x_3x_4$ $\qquad$ $x_1x_2x_3$ $\qquad$ $x_1x_2 \oplus x_1x_2x_3x_4$



x_1x_2 $\qquad$ $x_1x_2x_3 \oplus x_3x_4$ $\qquad$ $x_1x_2 \oplus x_1x_2\bar{x}_3\bar{x}_4 \oplus x_3x_4$ $\qquad$ $x_1x_2 \oplus x_3x_4$

Trading off size and space with LUT size

Synthesizing a 16-bit floating point adder with different LUT sizes



Experimental results: Quantum floating point library

- Existing implementations of 16-bit, 32-bit, and 64-bit floating point components
- quantumfpl.stationq.com
- Optimization using ABC (academic, open source)
- LHRS implemented in RevKit (academic, open source)

Experimental results: Quantum floating point library

- Existing implementations of 16-bit, 32-bit, and 64-bit floating point components
- quantumfpl.stationq.com
- Optimization using ABC (academic, open source)
- LHRS implemented in RevKit (academic, open source)

```
revkit> read_aiger add_32.aig
revkit> ps -a
[i] add_32: i/o = 64 / 32   and = 1763   lev = 137
revkit> lhrs -k 16
[i] run-time: 9.32 secs
revkit> ps -c
Lines:         368
Gates:         6141
T-count:       256668
Logic qubits:  368
```

The LHRS ecosystem

arxiv.org/abs/1706.02721

LHRS

Mapping into LUTs

Aligning LUTs as single-target gates

Mapping single-target gates

The LHRS ecosystem

arxiv.org/abs/1706.02721

program

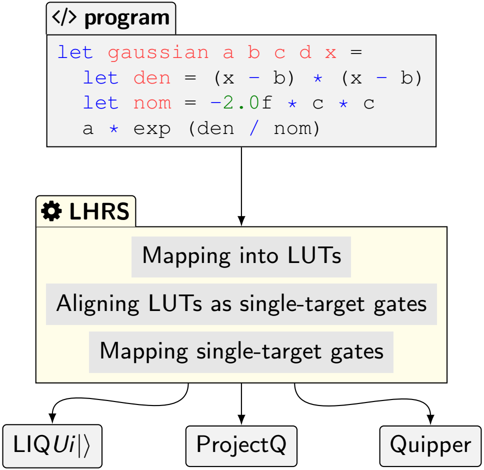
```
let gaussian a b c d x =
  let den = (x - b) * (x - b)
  let nom = -2.0f * c * c
  a * exp (den / nom)
```

LHRS

- Mapping into LUTs
- Aligning LUTs as single-target gates
- Mapping single-target gates

The LHRS ecosystem

arxiv.org/abs/1706.02721



Summary

- LHRS is the first scalable synthesis algorithm that allows reasonable space/time trade-offs

Summary

- LHRS is the first scalable synthesis algorithm that allows reasonable space/time trade-offs
- Valuable tool to estimate the cost of future quantum algorithms

Summary

- LHRS is the first scalable synthesis algorithm that allows reasonable space/time trade-offs
- Valuable tool to estimate the cost of future quantum algorithms
- Step 1: Mapping to efficiently partition large function into subfunctions (determines qubits)

Summary

- LHRS is the first scalable synthesis algorithm that allows reasonable space/time trade-offs
- Valuable tool to estimate the cost of future quantum algorithms
- Step 1: Mapping to efficiently partition large function into subfunctions (determines qubits)
- Step 2: High effort methods to map subfunctions into Clifford+T networks

Summary

- LHRS is the first scalable synthesis algorithm that allows reasonable space/time trade-offs
- Valuable tool to estimate the cost of future quantum algorithms
- Step 1: Mapping to efficiently partition large function into subfunctions (determines qubits)
- Step 2: High effort methods to map subfunctions into Clifford+T networks
- Most steps in the algorithm are (still) performed using conventional algorithms

Logic Synthesis for Quantum Computing

Mathias Soeken, Alan Mishchenko, Luca Amarù, Robert K. Brayton

Integrated Systems Laboratory, EPFL, Switzerland

lsi.epfl.ch • msoeken.github.io