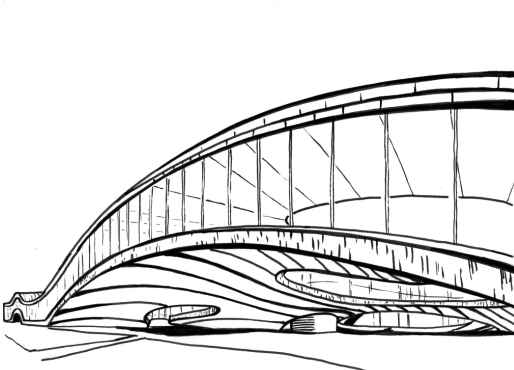
Hierarchical Reversible Logic Synthesis

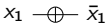
Mathias Soeken

Integrated Systems Laboratory, EPFL, Switzerland

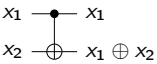
mathias.soeken@epfl.ch msoeken.github.io msoeken/cirkit download slides



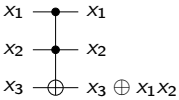
Reversible gates



NOT

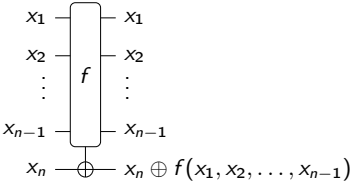


CNOT



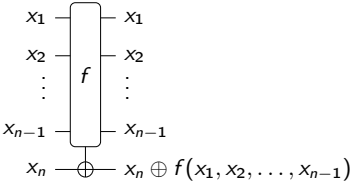
Toffoli

Reversible gates

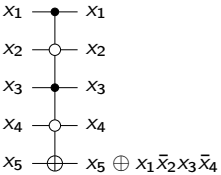


Single-target

Reversible gates

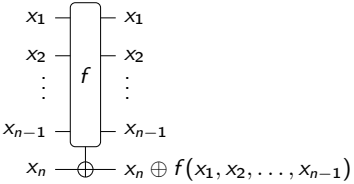


Single-target

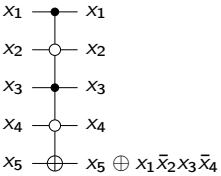


Multiple-controlled Toffoli

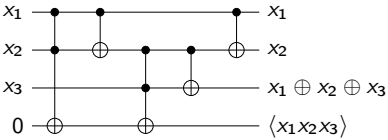
Reversible gates



Single-target

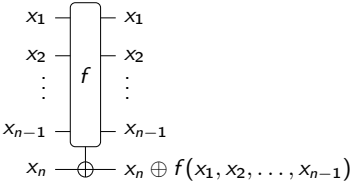


Multiple-controlled Toffoli

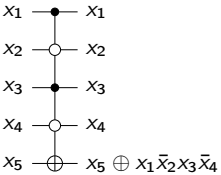


Full adder

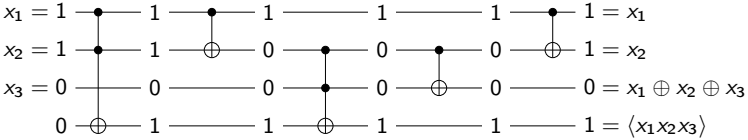
Reversible gates



Single-target



Multiple-controlled Toffoli



Full adder

Reversible synthesis classification

	line opt.	*gate opt.*	**nonreversible func.**	**reversible func.**
functional	✔	✔		SAT-based Enumerative
functional	✔	✖		Transformation-based Cycle-based Decomposition-based Metaheuristic Greedy
structural	✖	✖	ESOP-based Hierarchical Building block	

ESOP-based synthesis

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

RevKit: `esopbs`

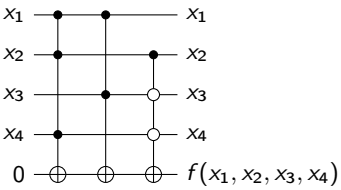
ESOP-based synthesis

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

RevKit: `esopbs`

ESOP-based synthesis

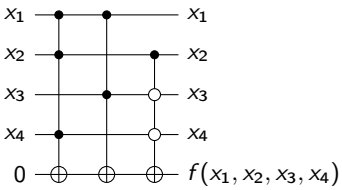
$$
\begin{aligned}
f(x_1, x_2, x_3, x_4) &= [(x_4 x_3 x_2 x_1)_2 \text{ is prime}] \\
&= \bar{x}_4 \bar{x}_3 x_2 \vee \bar{x}_4 x_3 x_1 \vee x_4 \bar{x}_3 x_2 x_1 \vee x_4 x_3 \bar{x}_2 x_1 \\
&= x_4 x_2 x_1 \oplus x_3 x_1 \oplus \bar{x}_4 \bar{x}_3 x_2
\end{aligned}
$$



RevKit: `esopbs`

ESOP-based synthesis

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



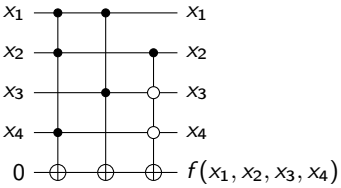
$$
\begin{aligned}
f_1 &= [3 \mid (x_4 x_3 x_2 x_1)_2] \\
&= \bar{x}_1 \bar{x}_2 x_3 \oplus \bar{x}_1 \bar{x}_4 \oplus x_1 \bar{x}_3 x_4 \oplus \\
&\quad x_1 x_2 x_4 \oplus x_2 \bar{x}_3 \bar{x}_4
\end{aligned}
$$

$$f_2 = [(x_4 x_3 x_2 x_1)_2 \text{ is prime}]$$

RevKit: `esopbs`

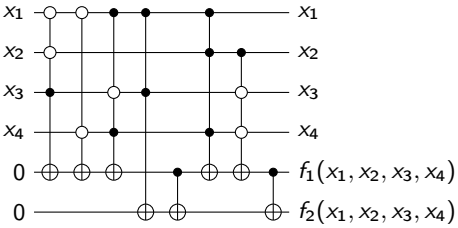
ESOP-based synthesis

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



$$
\begin{aligned}
f_1 &= [3 \mid (x_4x_3x_2x_1)_2] \\
&= \bar{x}_1\bar{x}_2x_3 \oplus \bar{x}_1\bar{x}_4 \oplus x_1\bar{x}_3x_4 \oplus \\
&\quad x_1x_2x_4 \oplus x_2\bar{x}_3\bar{x}_4
\end{aligned}
$$

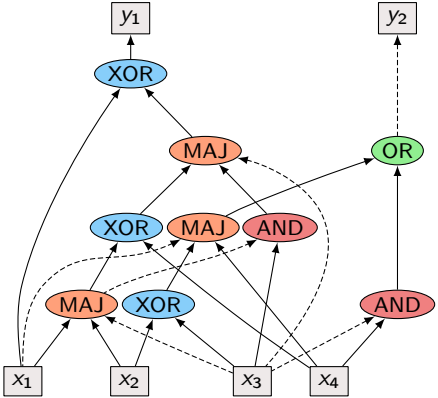
$$f_2 = [(x_4x_3x_2x_1)_2 \text{ is prime}]$$



RevKit: `esopbs`

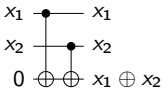
XMG-based synthesis

- XMG consists of **XOR** gates with 2 inputs and **MAJ** gates with 3 inputs
- MAJ gates with constant input can represent **AND** and **OR** gates
- Edges can be complemented (dashed lines in graph)

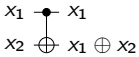


RevKit: `dxs`

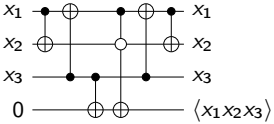
XMG-based synthesis



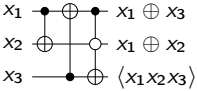
XOR



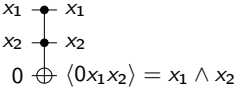
XOR (in-place)



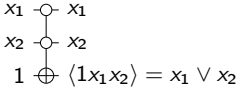
MAJ



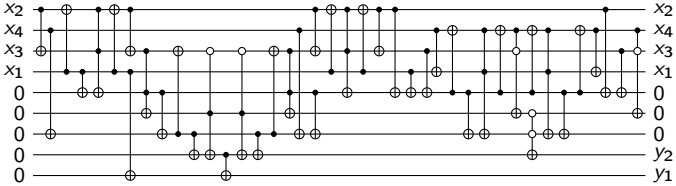
MAJ (in-place)



AND



OR



Soeken, Chattopadhyay: Unlocking efficiency and scalability of reversible logic synthesis using conventional logic synthesis, *DAC* (2016)

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

RevKit: `lhrs`

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

RevKit: `lhrs`

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

RevKit: `lhrs`

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

RevKit: `lhrs`

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

RevKit: `lhrs`

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

RevKit: `lhrs`

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

RevKit: `lhrs`

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.

new 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

affects #qubits

affects #T gates

RevKit: `lhrs`

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

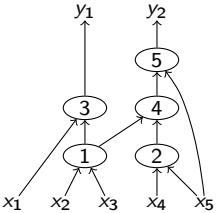
LUT mapping

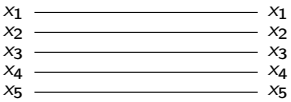
- Realizing a logic function or logic circuit in terms of a k-LUT logic network
- A k-LUT is any Boolean function with at most k inputs
- One of the most effective methods used in logic synthesis
- Typical objective functions are size (number of LUTs) and depth (longest path from inputs to outputs)
-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

LUT mapping

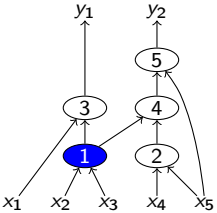
- Realizing a logic function or logic circuit in terms of a k-LUT logic network
- A k-LUT is any Boolean function with at most k inputs
- One of the most effective methods used in logic synthesis
- Typical objective functions are size (number of LUTs) and depth (longest path from inputs to outputs)
-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

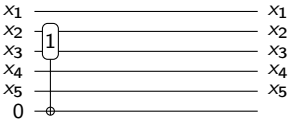
k-LUT network to reversible network





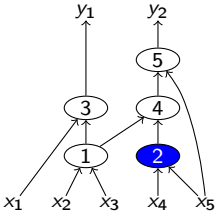
k-LUT network to reversible network





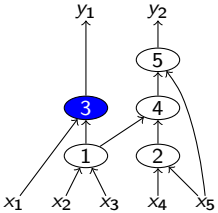
! k-LUT corresponds to k-controlled single-target gate

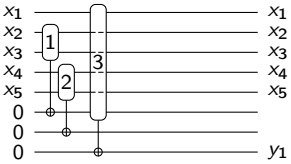
k-LUT network to reversible network



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

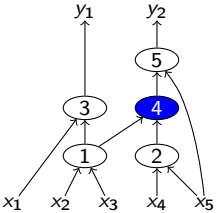
k-LUT network to reversible network

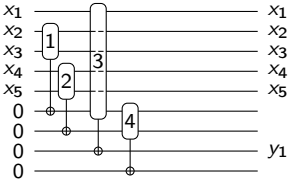




! k-LUT corresponds to k-controlled single-target gate

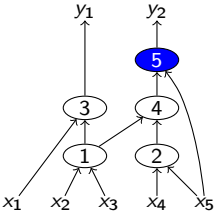
k-LUT network to reversible network

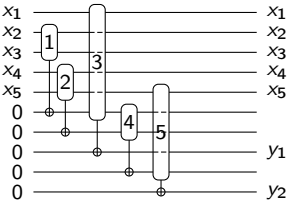




! k-LUT corresponds to k-controlled single-target gate

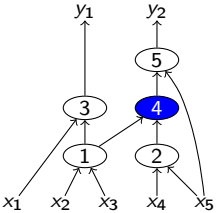
k-LUT network to reversible network

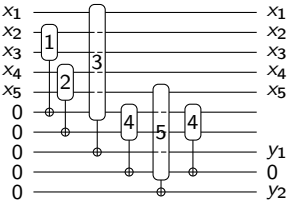




! k-LUT corresponds to k-controlled single-target gate

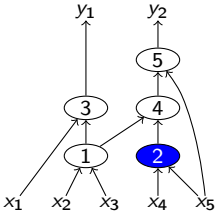
k-LUT network to reversible network

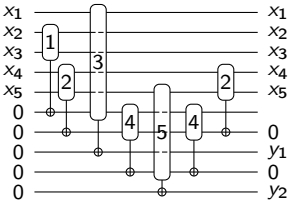




- ! k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed

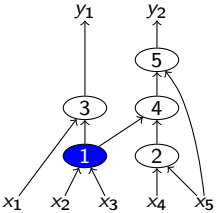
k-LUT network to reversible network

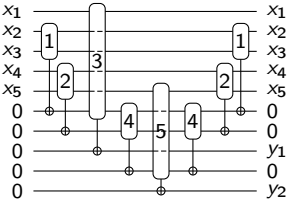




- ! k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed

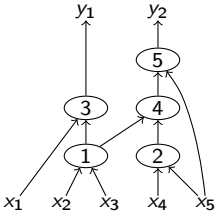
k-LUT network to reversible network

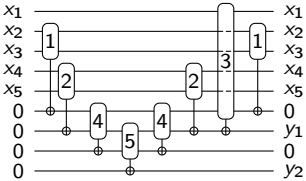




- **!** k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed

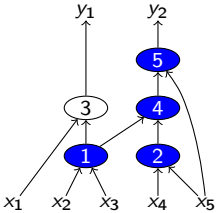
k-LUT network to reversible network

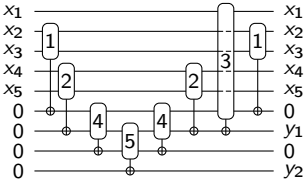




- ! k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed
- order of LUT traversal determines number of ancillas

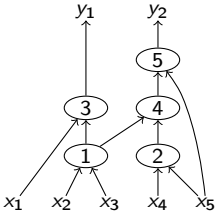
k-LUT network to reversible network

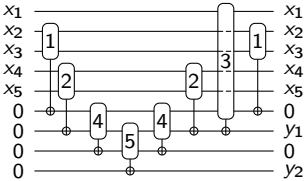




- ! k-LUT corresponds to k-controlled single-target gate
- non-output LUTs need to be uncomputed
- order of LUT traversal determines number of ancillas
- maximum output cone determines minimum number of ancillas (if we use at most 2 single-target gates per LUT)

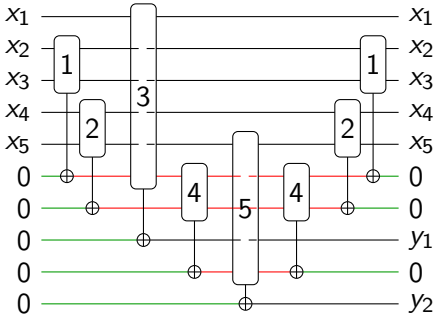
k-LUT network to reversible network





- ! 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 (if we use at most 2 single-target gates per LUT)
- ☺ 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

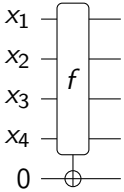
- Direct
  - Map each control function using ESOP based synthesis
  - Does not use ancillae
- LUT-based
  - Map control function into smaller LUT network

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

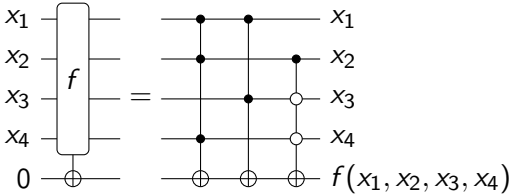
Direct mapping

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



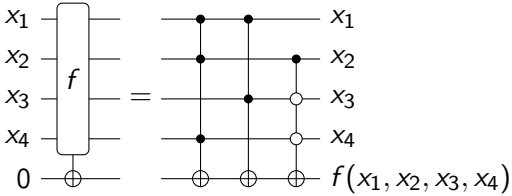
Direct mapping

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



Direct mapping

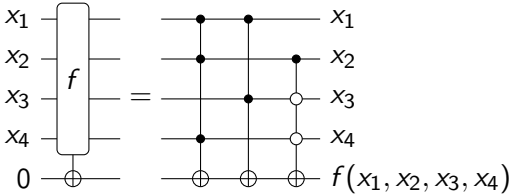
$$
\begin{aligned}
f(x_1, x_2, x_3, x_4) &= [(x_4 x_3 x_2 x_1)_2 \text{ is prime}] \\
&= \bar{x}_4 \bar{x}_3 x_2 \vee \bar{x}_4 x_3 x_1 \vee x_4 \bar{x}_3 x_2 x_1 \vee x_4 x_3 \bar{x}_2 x_1 \\
&= x_4 x_2 x_1 \oplus x_3 x_1 \oplus \bar{x}_4 \bar{x}_3 x_2
\end{aligned}
$$



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

Direct mapping

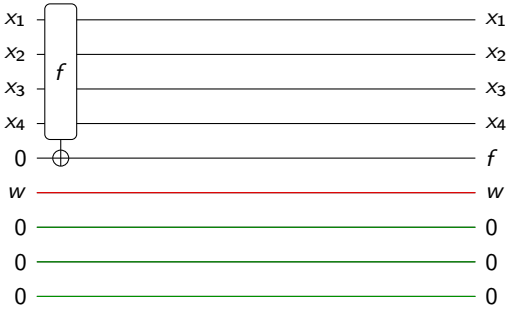
$$
\begin{aligned}
f(x_1, x_2, x_3, x_4) &= [(x_4 x_3 x_2 x_1)_2 \text{ is prime}] \\
&= \bar{x}_4 \bar{x}_3 x_2 \vee \bar{x}_4 x_3 x_1 \vee x_4 \bar{x}_3 x_2 x_1 \vee x_4 x_3 \bar{x}_2 x_1 \\
&= x_4 x_2 x_1 \oplus x_3 x_1 \oplus \bar{x}_4 \bar{x}_3 x_2
\end{aligned}
$$



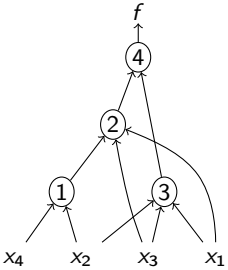
- Each multiple-controlled Toffoli gate is mapped to Clifford+T

☹ ESOP minimization tools (e.g., exorcism) optimize for cube count

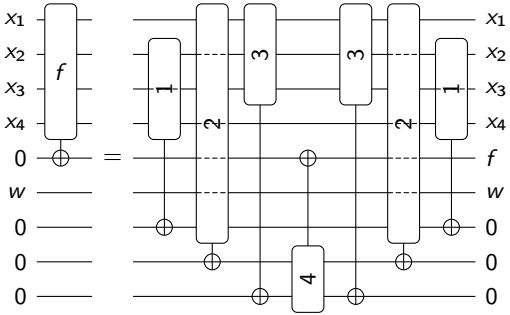
LUT-based single-target gate mapping

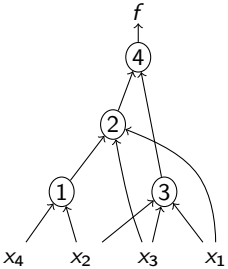


LUT-based single-target gate mapping

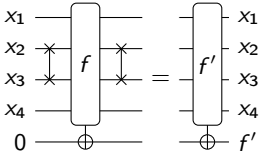


LUT-based single-target gate mapping

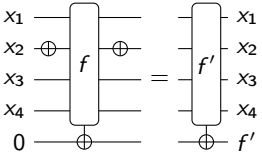




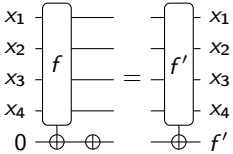
Exploiting Boolean function classification



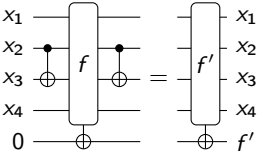
swap inputs



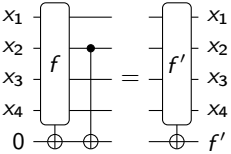
invert inputs



invert output



spectral translation



disjoint spectral translation

- Operations do not influence T-count of the quantum circuit

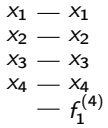
☺ All optimum circuits in an equivalence class have the same T-count

Classification of all 4-input functions

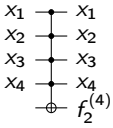
- All 65,356 4-input functions collapse into only 8 equivalence classes
- Classification simple by comparing coefficients in the function's Walsh spectrum

Classification of all 4-input functions

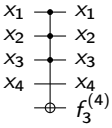
- All 65,356 4-input functions collapse into only 8 equivalence classes
- Classification simple by comparing coefficients in the function's Walsh spectrum



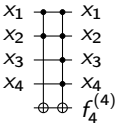
$\perp$



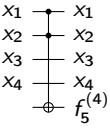
$x_1x_2x_3x_4$



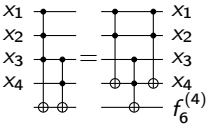
$x_1x_2x_3$



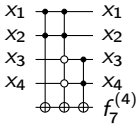
$x_1x_2 \oplus x_1x_2x_3x_4$



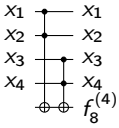
x_1x_2



$x_1x_2x_3 \oplus x_3x_4$



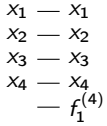
$x_1x_2 \oplus x_1x_2\bar{x}_3\bar{x}_4 \oplus x_3x_4$



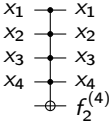
$x_1x_2 \oplus x_3x_4$

Classification of all 4-input functions

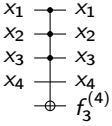
- All 65,356 4-input functions collapse into only 8 equivalence classes (all 4,294,967,296 5-input functions collapse into 48 classes)
- Classification simple by comparing coefficients in the function's Walsh spectrum (and auto-correlation spectrum)



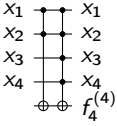
$\perp$



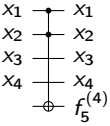
$x_1x_2x_3x_4$



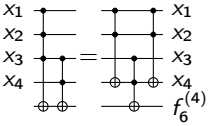
$x_1x_2x_3$



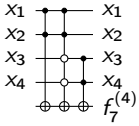
$x_1x_2 \oplus x_1x_2x_3x_4$



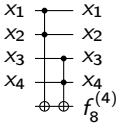
x_1x_2



$x_1x_2x_3 \oplus x_3x_4$



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



$x_1x_2 \oplus x_3x_4$

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

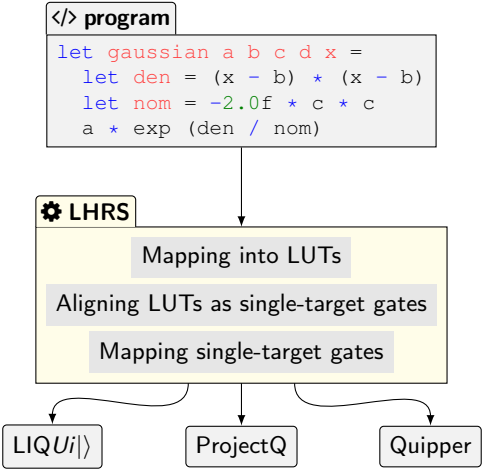
Mapping into LUTs

Aligning LUTs as single-target gates

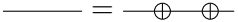
Mapping single-target gates

The LHRS ecosystem

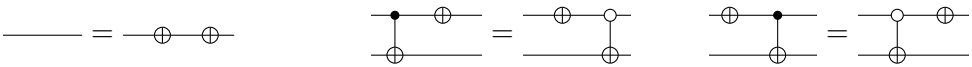
arxiv.org/abs/1706.02721



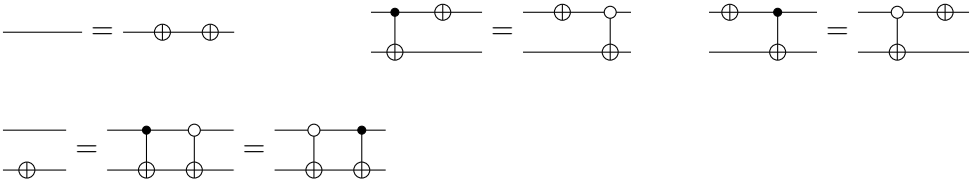
Circuit rewriting



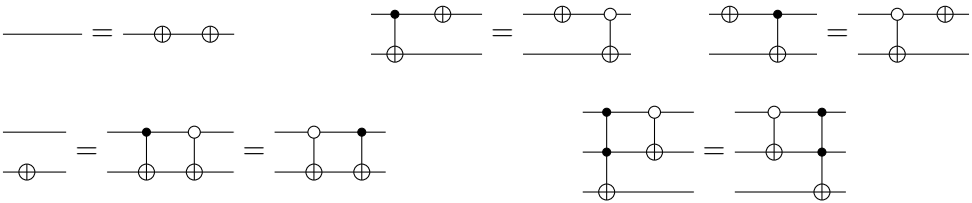
Circuit rewriting



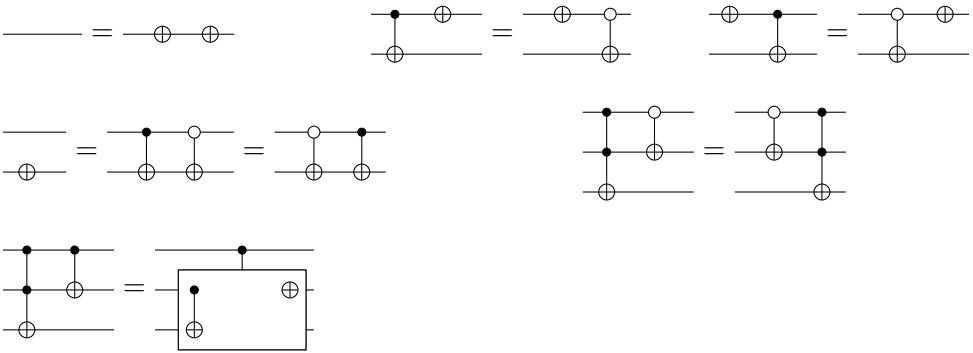
Circuit rewriting



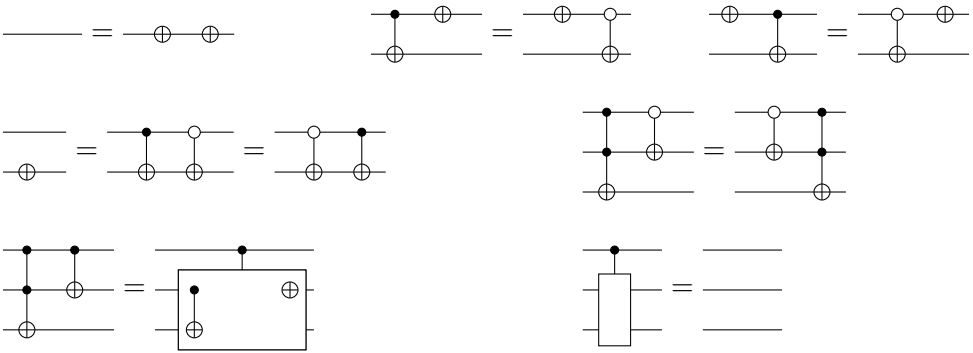
Circuit rewriting



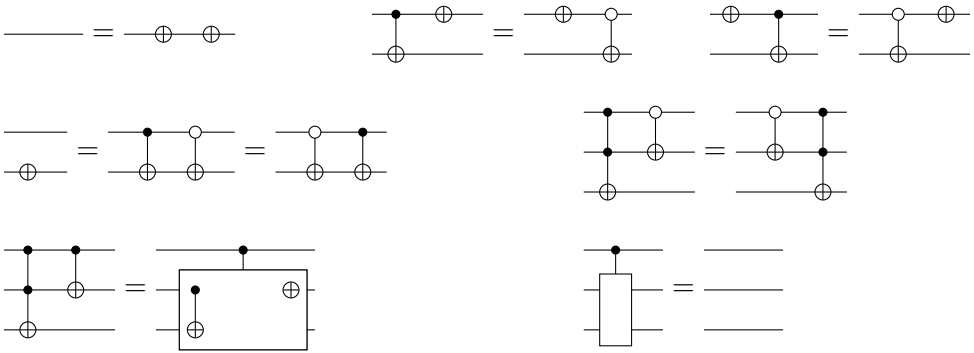
Circuit rewriting



Circuit rewriting

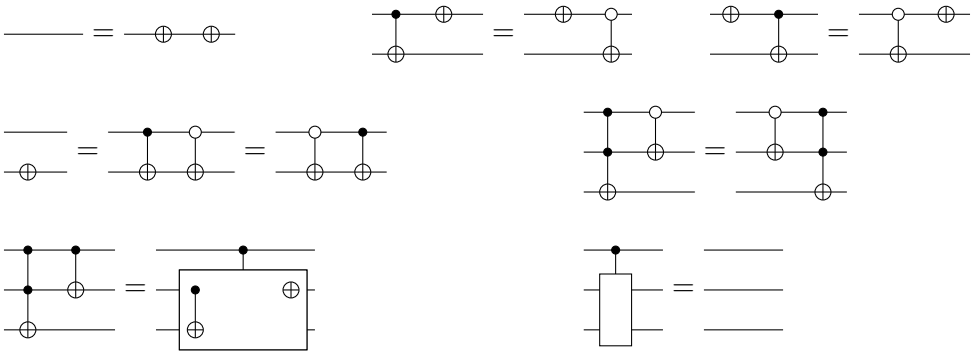


Circuit rewriting



? **Open problem:** These six rules (plus SWAP rule) are complete, i.e., one can rewrite any circuit realizing some function into any other circuit realizing the same function

Circuit rewriting



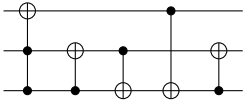
? **Open problem:** These six rules (plus SWAP rule) are complete, i.e., one can rewrite any circuit realizing some function into any other circuit realizing the same function

- Rule set has been extended to consider ancillae

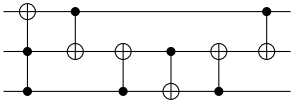
M. Soeken, M.K. Thomsen: White dots *do* matter: rewriting reversible logic circuits, *RC* 5, (2013), 196–208

Circuit rewriting: example

Circuit G_1



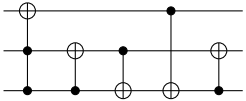
Circuit G_2



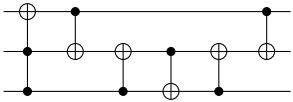
M. Soeken, M.K. Thomsen: White dots *do* matter: rewriting reversible logic circuits, *RC* 5, (2013), 196–208

Circuit rewriting: example

Circuit G_1



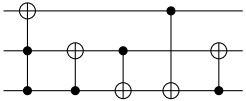
Circuit G_2



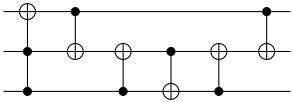
- Can be used for equivalence checking to check $G_1 \equiv G_2$
- Construct circuit $G = G_2^{-1} \circ G_1$
- Rewrite G to identity

Circuit rewriting: example

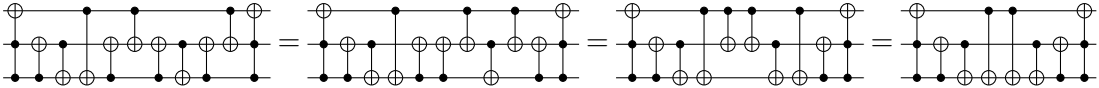
Circuit G_1



Circuit G_2

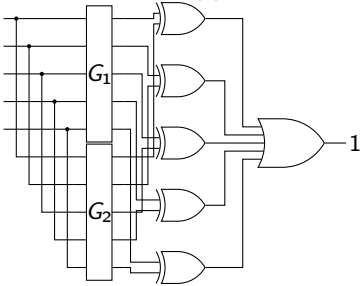


- Can be used for equivalence checking to check $G_1 \equiv G_2$
- Construct circuit $G = G_2^{-1} \circ G_1$
- Rewrite G to identity



Equivalence checking of reversible circuits

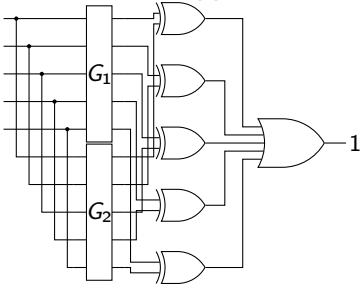
Conventional approach



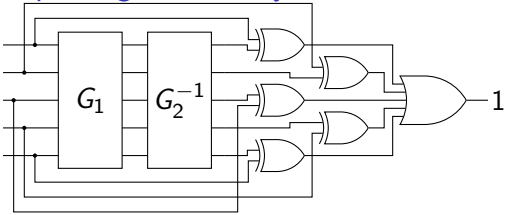
Exploiting reversibility

Equivalence checking of reversible circuits

Conventional approach



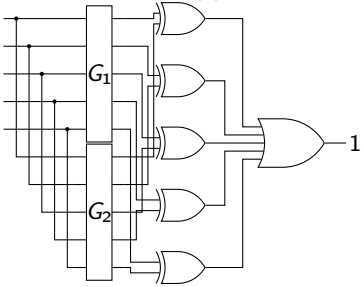
Exploiting reversibility



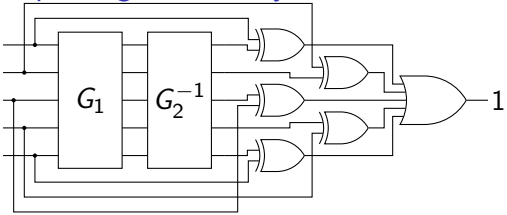
RevKit: `rec`

Equivalence checking of reversible circuits

Conventional approach



Exploiting reversibility



RevKit: `rec`

- Circuit is translated into a SAT formula and solved with a SAT solver
- A satisfying solution is witnessing a counter-example
- Solvers with support for XOR clauses allow for more natural encoding and better runtimes

Hierarchical Reversible Logic Synthesis

Mathias Soeken

Integrated Systems Laboratory, EPFL, Switzerland

mathias.soeken@epfl.ch msoeken.github.io msoeken/cirkit download slides