Integrated Systems and Quantum Computing
Mathias Soeken and Giovanni De Micheli

Integrated Systems Laboratory, EPFL, Switzerland

✉️ mathias.soeken@epfl.ch  🌐 msoeken.github.io  🌀 msoeken/cirkit
Logic synthesis

- Represent (complex) functionality in terms of logic networks

Example: implement floating-point operation in terms of the majority-of-three and complement operations

Tasks involved:

- Synthesizing high-level code (C, Verilog, VHDL, ...) into logic networks
- Optimize logic networks wrt. a cost function
- Map logic networks into a specific technology

Teaching:

- Design Technologies for Integrated Systems (this semester)
- Semester projects (contact me)
Logic synthesis

- Represent (complex) functionality in terms of logic networks
- **Example:** implement floating-point operation in terms of the majority-of-three and complement operations
Logic synthesis

▷ Represent (complex) functionality in terms of logic networks

▷ **Example**: implement floating-point operation in terms of the majority-of-three and complement operations

▷ **Tasks involved:**
Logic synthesis

- Represent (complex) functionality in terms of logic networks
- **Example:** implement floating-point operation in terms of the majority-of-three and complement operations
- **Tasks involved:**
  - **Synthesizing** high-level code (C, Verilog, VHDL, ...) into logic networks
Logic synthesis

- Represent (complex) functionality in terms of logic networks
- **Example:** implement floating-point operation in terms of the majority-of-three and complement operations
- **Tasks involved:**
  - Synthesizing high-level code (C, Verilog, VHDL, ...) into logic networks
  - Optimize logic networks wrt. a cost function
Logic synthesis

- Represent (complex) functionality in terms of logic networks
- **Example:** implement floating-point operation in terms of the majority-of-three and complement operations
- **Tasks involved:**
  - **Synthesizing** high-level code (C, Verilog, VHDL, ...) into logic networks
  - **Optimize** logic networks wrt. a cost function
  - **Map** logic network into a specific technology

Teaching:

- Design Technologies for Integrated Systems (this semester)
- Semester projects (contact me)
Logic synthesis

- Represent (complex) functionality in terms of logic networks
- **Example:** implement floating-point operation in terms of the majority-of-three and complement operations
- **Tasks involved:**
  - Synthesizing high-level code (C, Verilog, VHDL, ...) into logic networks
  - Optimize logic networks wrt. a cost function
  - Map logic network into a specific technology
- **Teaching:**
Logic synthesis

- Represent (complex) functionality in terms of logic networks
- **Example:** implement floating-point operation in terms of the majority-of-three and complement operations
- **Tasks involved:**
  - **Synthesizing** high-level code (C, Verilog, VHDL, ...) into logic networks
  - **Optimize** logic networks wrt. a cost function
  - **Map** logic network into a specific technology
- **Teaching:**
  - **Design Technologies for Integrated Systems** (*this semester*)
Logic synthesis

- Represent (complex) functionality in terms of logic networks
- **Example:** implement floating-point operation in terms of the majority-of-three and complement operations
- **Tasks involved:**
  - Synthesizing high-level code (C, Verilog, VHDL, ...) into logic networks
  - Optimize logic networks wrt. a cost function
  - Map logic networks into a specific technology
- **Teaching:**
  - Design Technologies for Integrated Systems (*this semester*)
  - Semester projects (*contact me*)
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 $\approx 50$ qubits (45-qubit simulation has been performed classically)
- Smallest practical problems require $\approx 100$ (logical) qubits
Challenges in logic synthesis for quantum computing

1. Quantum computers process qubits not bits

Classical half-adder

Quantum half-adder

\[
\begin{align*}
|x\rangle & \rightarrow |x\rangle \\
|y\rangle & \rightarrow |s\rangle \\
|0\rangle & \rightarrow |c\rangle
\end{align*}
\]
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
Challenges in logic synthesis for quantum computing

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

\[
\begin{align*}
\text{NOT} & \quad x_1 \oplus \overline{x}_1 \\
\text{CNOT} & \quad x_1 \quad x_2 \quad x_1 \oplus x_2 \\
\text{Toffoli} & \quad x_1 \quad x_2 \quad x_3 \quad x_1 \oplus x_1 x_2
\end{align*}
\]
Reversible gates

NOT: $x_1 \oplus \bar{x}_1$

CNOT: $x_1 \oplus x_1 \oplus x_2$

Toffoli: $x_1 \oplus x_1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_1 x_2$

Single-target: $x_1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus f(x_1, x_2)$
Reversible gates

NOT

\[\bar{x}_1 \oplus x_1\]

CNOT

\[x_1 \oplus (x_1 \oplus x_2)\]

Toffoli

\[x_1 \oplus (x_1 \oplus x_2) \oplus x_3 \oplus x_1 x_2\]

Single-target

\[x_1 \oplus f(x_1, x_2)\]

Multiple-controlled Toffoli

\[x_5 \oplus (x_1 \bar{x}_2 x_3 \bar{x}_4)\]
Reversible gates

\[ \bar{x}_1 \oplus x_1 \]

**NOT**

\[ x_1 \oplus x_1 \oplus x_2 \]

**CNOT**

\[ x_1 \oplus x_1 \oplus x_1 x_2 \]

**Toffoli**

\[ x_3 \oplus f(x_1, x_2) \]

**Single-target**

\[ x_1 \oplus x_1 \]

**Multiple-controlled Toffoli**

\[ x_1 \oplus x_1 \oplus x_2 x_3 \bar{x}_4 \]

**Full adder**

\[ x_1 \oplus x_2 \oplus x_3 \]

\[ 0 \oplus (x_1 x_2 x_3) \]
Quantum gates

- Qubit is vector $|\varphi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ with $|\alpha|^2 + |\beta|^2 = 1$.
- Classical 0 is $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$; Classical 1 is $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.
Quantum gates

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

\[
\begin{align*}
|\varphi_1\rangle &\quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad |\varphi_1\varphi_2\rangle \\
|\varphi_2\rangle &\quad CNOT \\
\end{align*}
\]

\[
\begin{align*}
|\varphi\rangle &\quad H \quad \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} |\varphi\rangle \\
|\varphi\rangle &\quad T \quad \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} |\varphi\rangle \\
\end{align*}
\]
Quantum gates

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

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

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

$|\varphi\rangle \rightarrow T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} |\varphi\rangle$

Clifford+$T$ circuit [Amy et al., TCAD 32, 2013]
Quantum gates

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

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

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

$|\varphi\rangle \xrightarrow{T} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} |\varphi\rangle$

CNOT

Hadamard

T

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

- Parallel composition using Kronecker product
- Sequential composition using matrix product

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
4. Map single-target gates into multiple-controlled Toffoli gates
5. Map multiple-controlled Toffoli gates into Clifford+$T$ networks

Applications: Creating a floating-point library
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 multiple-controlled Toffoli gates

Applications: Creating a floating-point library
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 multiple-controlled Toffoli gates
5. Map multiple-controlled Toffoli 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)
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 multiple-controlled Toffoli gates
5. Map multiple-controlled Toffoli gates into Clifford+ $T$ networks

**Applications:** Creating a floating-point library
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 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)

$\vdash$ fast mapping that generates a fixed-space skeleton for subnetwork synthesis
$k$-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

Diagram showing the relationship between $k$-LUTs and reversible gates.
$k$-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

>>> fast mapping that generates a fixed-space skeleton for subnetwork synthesis
$k$-LUT network to reversible network

- $k$-LUT corresponds to $k$-controlled single-target gate
  - non-output LUTs need to be uncomputed
$k$-$\text{LUT}$ network to reversible network

\[ y_1 \rightarrow 3 \rightarrow y_2 \rightarrow 5 \rightarrow 1 \rightarrow y_1 \]

\[ x_1 \rightarrow 2 \rightarrow x_2 \rightarrow 4 \rightarrow x_3 \rightarrow 2 \rightarrow x_4 \rightarrow 4 \rightarrow x_5 \rightarrow 2 \rightarrow x_4 \]

$\text{! } k$-$\text{LUT}$ 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 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
Direct mapping

\[ f(x_1, x_2, x_3, x_4) = [(x_4 x_3 x_2 x_1)_2 \text{ is prime}] \]
\[ = \bar{x}_4 x_3 x_2 \lor \bar{x}_4 x_3 x_1 \lor x_4 \bar{x}_3 x_2 x_1 \lor x_4 x_3 \bar{x}_2 x_1 \]

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

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

\[ = \bar{x}_4 x_3 x_2 \lor \bar{x}_4 x_3 x_1 \lor x_4 \bar{x}_3 x_2 x_1 \lor x_4 x_3 \bar{x}_2 x_1 \]

\[ = x_4 x_2 x_1 \oplus x_3 x_1 \oplus \bar{x}_4 \bar{x}_3 x_2 \]

Each multiple-controlled Toffoli gate is mapped to Clifford+T \[ \dashrightarrow \]

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

\[
f(x_1, x_2, x_3, x_4) = [(x_4x_3x_2x_1)_2 \text{ is prime}]
\]

\[
= \bar{x}_4\bar{x}_3x_2 \lor \bar{x}_4x_3x_1 \lor x_4\bar{x}_3x_2x_1 \lor x_4x_3\bar{x}_2x_1 \\
= x_4x_2x_1 \oplus x_3x_1 \oplus \bar{x}_4\bar{x}_3x_2
\]

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

\[ f(x_1, x_2, x_3, x_4) = [(x_4 x_3 x_2 x_1)_2 \text{ is prime}] \]
\[ = \bar{x}_4 \bar{x}_3 x_2 \lor \bar{x}_4 x_3 x_1 \lor x_4 \bar{x}_3 x_2 x_1 \lor x_4 x_3 \bar{x}_2 x_1 \]
\[ = x_4 x_2 x_1 \oplus x_3 x_1 \oplus \bar{x}_4 \bar{x}_3 x_2 \]

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

☆ ESOP minimization tools (e.g., exorcism) optimize for cube count
Integrated Systems and Quantum Computing

Mathias Soeken and Giovanni De Micheli

Integrated Systems Laboratory, EPFL, Switzerland

✉️ mathias.soeken@epfl.ch 🌐 msoeken.github.io ☑️ msoeken/cirkit