The EPFL Logic Synthesis Libraries

Dingchao Gao

Institute of Software Chinese Academy of Sciences

October 22, 2022

libraries[1]

- alice: command shell library
- mockturtle: logic network library
- lorina: parsing library
- kitty: truth table library
- bill: reasoning library
- percy: exact synthesis library
- easy: exclusive-or sum-of-product (ESOP) library

[1] Mathias Soeken et al. "The EPFL Logic Synthesis Libraries.". In: *arXiv: Logic in Computer Science* (2022). DOI: null. URL: https://arxiv.org/abs/1805.05121.

quantum libraries

- angel: quantum state preparation library
- tweedledum: quantum compilation library
- caterpillar: quantum circuit synthesis library

initial state[2]

- target:

$$|\varphi_j\rangle = \frac{1}{\sqrt{|on(f)|}} \sum_{x \in \mathrm{on}(f)} |x\rangle \tag{1}$$

- the general idea of state preparation algorithm relies on the identity:

$$\mathrm{QSP}_f |0\rangle^{\otimes n} = \left(\mathrm{QSP}_{f_{\bar{x}_i}} \oplus \mathrm{QSP}_{f_{x_i}}\right) \left(G\left(p_f\left(\bar{x}_i\right)\right) \otimes I_{2^n-1}\right) |0\rangle \tag{2}$$

- $G\left(p_f\left(\bar{x}_i\right)\right)$ is a unitary transformation gate that satisfies:

$$G(p_f\left(\bar{x}_i\right))|0\rangle = \sqrt{p_f\left(\bar{x}_i\right)}|0\rangle + \sqrt{1 - p_f\left(\bar{x}_i\right)}|1\rangle \tag{3}$$

$$G\left(p_f\left(\bar{x}_i\right)\right) = R_y\left(2\cos^{-1}\left(\sqrt{p_f\left(\bar{x}_i\right)}\right)\right) \tag{4}$$

[2] Fereshte Mozafari et al. "Automatic Uniform Quantum State Preparation Using Decision Diagrams". In: *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)* (2020). DOI: 10.1109/ismvl49045.2020.00-10.

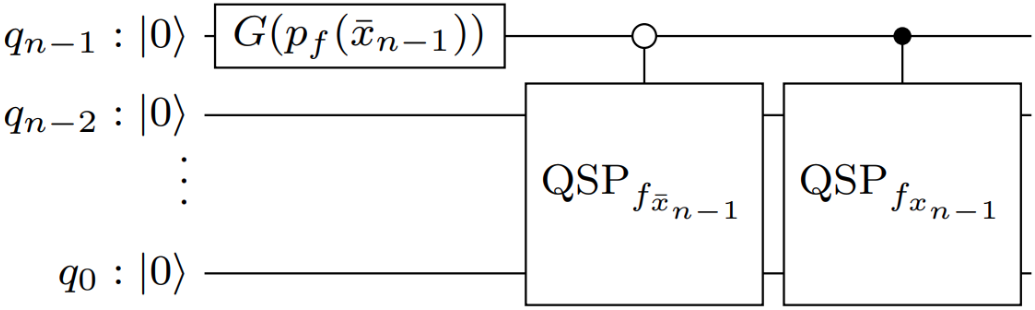


Figure: the general idea of QSP in the quantum circuit model for $i = n - 1$.

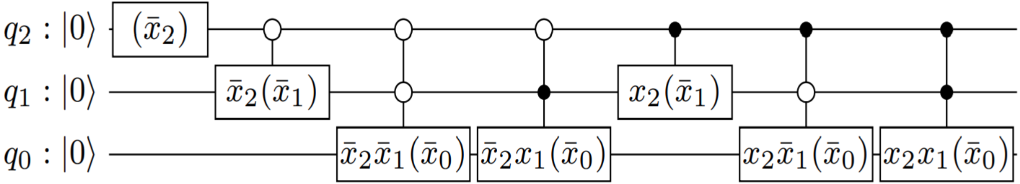


Figure: the abstract quantum gates of $QSP_{<x_0 x_1 x_2>}$

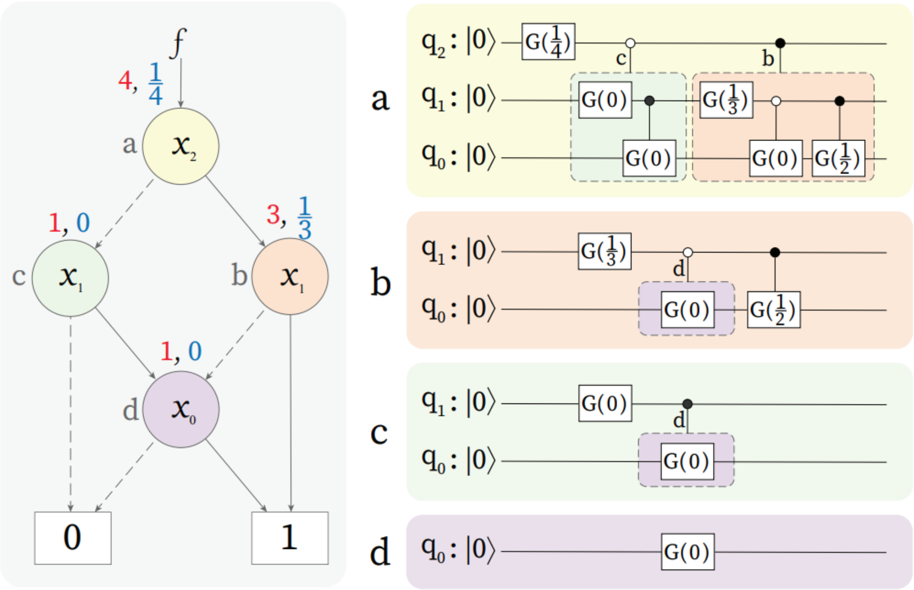


Figure: BDD for boolean function $f = x_0x_1 \vee x_1x_2 \vee x_2x_0$ and the procedure of extracting gates for each node from bottom to top

compiling oracle by XAG[3]

- representing an n-variable boolean function using a logic network over the gate basis$\{\neg, \oplus, \wedge\}$:

$$x_i = x_{j(i)} \oplus x_{k(i)} \quad \text{or} \quad x_i = x_{j(i)}^{p(i)} \wedge x_{k(i)}^{q(i)} \tag{5}$$

  where $n < i \leq n + r$

- the linear transitive fan-in of a node $x_i$using the recursive function:

$$\operatorname{ltfi}(x_i) = \begin{cases} \{x_i\} & \text{if } i < n \text{ or } \circ_i = \wedge, \\ \operatorname{ltfi}\left(x_{j(i)}\right) \triangle \operatorname{ltfi}\left(x_{k(i)}\right) & \text{otherwise} \end{cases} \tag{6}$$

[3] Giulia Meuli et al. "The Role of Multiplicative Complexity in Compiling Low T-count Oracle Circuits". In: (2019). DOI: 10.1109/iccad45719.2019.8942093.

XAG example

- for $f(x) = x_1x_2 \vee x_2x_3 \vee x_3x_1$
- can be realized by the logic network:

$$x_4 = x_1 \oplus x_2, \qquad x_5 = x_2 \oplus x_3 \tag{7}$$

$$x_6 = x_4 \wedge x_6, \qquad x_7 = x_2 \oplus x_6 \tag{8}$$

- the linear transitive fan-in of a node:

$$\operatorname{ltfi}(x_4) = \{x_1, x_2\}, \qquad \operatorname{ltfi}(x_5) = \{x_2, x_3\} \tag{9}$$

$$\operatorname{ltfi}(x_6) = \{x_6\}, \qquad \operatorname{ltfi}(x_7) = \{x_2, x_6\} \tag{10}$$

XAG

function *compute* **is**

```
for i = n+1, ..., n+r where ∘_i = ∧ do
    set p ← p(i), q ← q(i), j ← j(i), k ← k(i);
    set L_1 ← ltfi(x_j), L_2 ← ltfi(x_k);
    if L_1 ⊆ L_2 then
        swap L_1 ↔ L_2 and p ↔ q;
    end
    let t_1 be some element in L_1 \ L_2;
    let t_2 be some element in L_2;
    CNOT(x, t_1) for all x ∈ L_1 \ {t_1};
    CNOT(x, t_2) for all x ∈ L_2 \ {t_2};
    if p then NOT(t_1);
    if q then NOT(t_2);
    TOFFOLI(t_1, t_2, x_i);
    if p then NOT(t_2);
    if q then NOT(t_1);
    CNOT(x, t_2) for all x ∈ L_2 \ {t_2};
    CNOT(x, t_1) for all x ∈ L_1 \ {t_1};
end
```

end

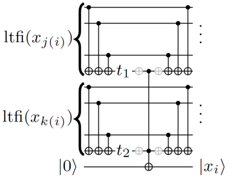


Figure: quantum circuit construction for function compute

Algorithm 1 Heuristic compilation algorithm

Input: Logic network with gates $x_{n+1}, \ldots, x_{n+r}$
Output: Quantum circuit for Uf

```
compute
CNOT(x_{n+r-1}, y)
if p then
    NOT(y)
end if
compute†
```

pebbling[4]

- target: solving iteratively the reversible pebbling game on the given network
- method: the problem is encoded as a SAT problem and addressed by state-of-the-art solvers
- result: get the trade-off between qubits and operations

[4] Giulia Meuli et al. "Reversible Pebbling Game for Quantum Memory Management". In: *arXiv: Quantum Physics* (2019). DOI: 10.23919/date.2019.8715092.

pebbling example

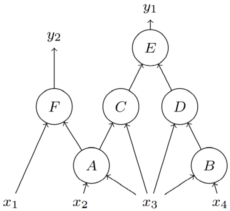


Figure: example of a directed acyclic graph

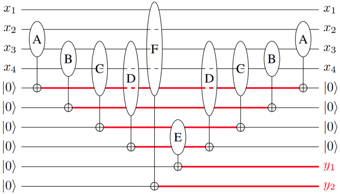


Figure: Bennet strategy

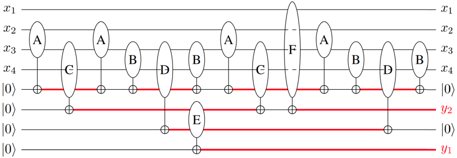


Figure: space-optimized by reordering

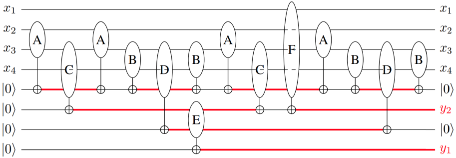


Figure: space-optimized by increasing the number of gates

tweedledum induction[5]

- narrowing the gap between high-level algorithms and physical devices
- an intuitive and flexible intermediate representation that supports different abstraction levels across the same circuit structure

[5] Bruno Schmitt and Giovanni De Micheli. "Tweedledum: A Compiler Companion for Quantum Computing". In: *2022 Design, Automation Test in Europe Conference Exhibition (DATE)* (2022). DOI: 10.23919/date54114.2022.9774510.

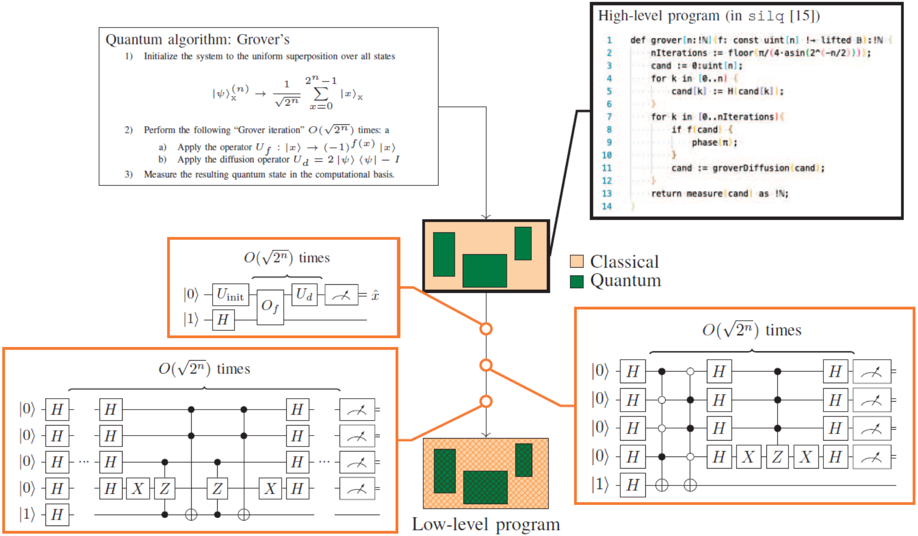


Figure: compilation flow overview

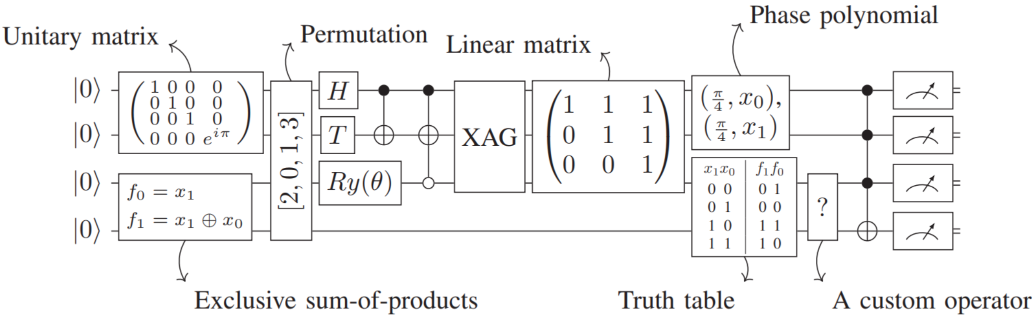


Figure: tweedledum's IR flexibility

synthesis

- pkrm_synth and pprm_synth synthesize a particular case of an exclusive-or sum-of-product (ESOP) expression for f
- spectrum_synth uses the Rademacher-Walsh spectrum of a truth table to generate a circuit
- lhrs_synth and xag_synth are examples of hierarchical synthesis
- a_star_swap_synth and star_swap_synth for circuits composed entirely of SWAP operators

synthesis

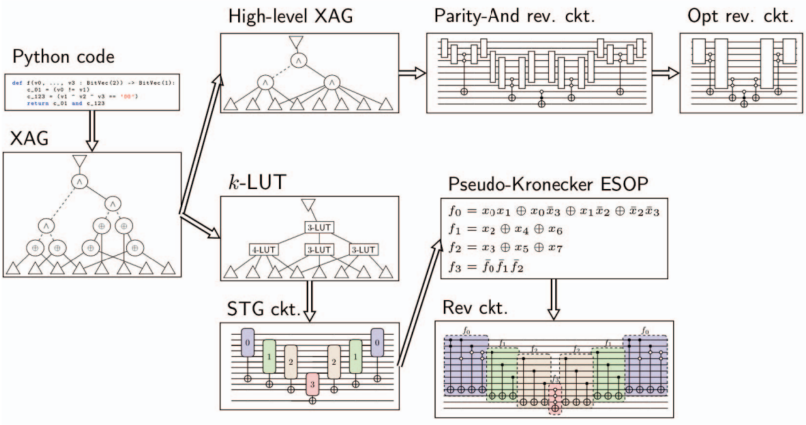


Figure: overview of possible Boolean function synthesis flows

compilation passes

- utility
- decomposition
- mapping
- optimization

problem

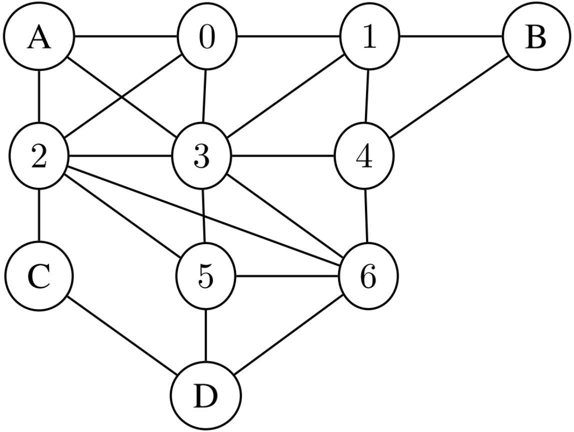


Figure: Zed city as an undirected graph

boolean function

python implementation of f

```
def f(v0, ..., v6 : BitVec(2)) ->
    BitVec(1):
  c0 = (v0 != ''00)
  c1 = (v1 != ''01) and (v1 != v0)
  c2 = (v2 != ''00) and (v2 != ''10)
       and (v2 != v0)
  c3 = (v3 != ''00) and (v3 != v0)
       and (v3 != v1) and (v3 != v2)
  c4 = (v4 != ''01) and (v4 != v1)
       and (v4 != v3)
  c5 = (v5 != ''11) and (v5 != v2)
       and (v5 != v3)
  c6 = (v6 != ''11) and (v6 != v2)
       and (v6 != v3) and (v6 != v4)
        and (v6 != v5)
  return c0 and c1 and c2 and c3
       and c4 and c5 and c6
```

hand-optimized python implementation of f

```
def f(v0, ..., v6 : BitVec(2)) ->
    BitVec(1):
  c1 = (v1[0] == v1[1]) and (v3 !=
      v1)
  c023 = ((v0 ^ v2 ^ v3) == ''00)
  c4 = (v4 != v1) and (v4 != v3)
  c5 = (v5 != v2) and (v5 != v3)
  c6 = ((v2 ^ v3 ^ v5 ^ v6) == ''00)
       and (v6 != v4)
  return c1 and c023 and c4 and c5
      and c6
```

initial state boolean function

python implementation of initial function f

```
def g(v0, ..., v6 : BitVec(2)) -> BitVec(1):
    return (v0 != ''00) and (v1 != ''01) and(v2 != ''00) and (v2 != ''10) and(v3 !=
        ''00) and (v4 != ''01) and(v5 != ''11) and (v6 != ''11)
```

result[6]

	Hand-optimized		Non-optimized	
	Qubits	cost	Qubits	cost
IBM's solution	32	5004		
Whit3z solution	32	2474		
XAG-based flow	31	2202	56	4347
XAG-based flow with pebbling	21	4497	30	7737

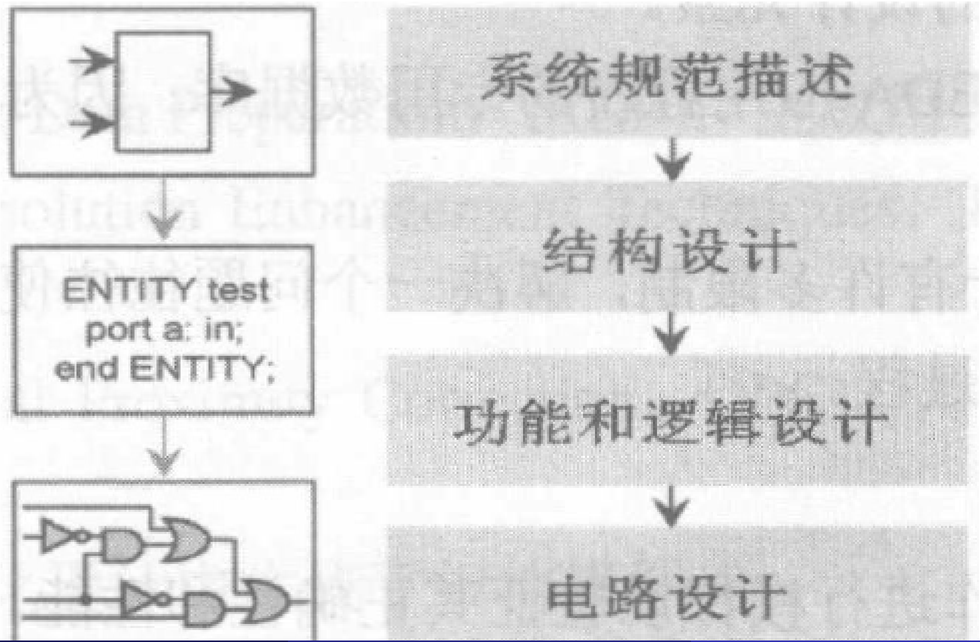
Table: quality of results for boolean function (hand-optimized and non-optimized), where $cost = q_1 + 10q_2$

[6] Bruno Schmitt et al. "From Boolean functions to quantum circuits: A scalable quantum compilation flow in C++". In: *2021 Design, Automation Test in Europe Conference Exhibition (DATE)* (2021). DOI: 10.23919/date51398.2021.9474237.

disscussion

- lack of quantum computing features
- comparing with isq etc
- the abstract level of quantum computer?

disscussion



references I

- Giulia Meuli et al. "Reversible Pebbling Game for Quantum Memory Management". In: *arXiv: Quantum Physics* (2019). DOI: 10.23919/date.2019.8715092.
- Giulia Meuli et al. "The Role of Multiplicative Complexity in Compiling Low T-count Oracle Circuits". In: (2019). DOI: 10.1109/iccad45719.2019.8942093.
- Fereshte Mozafari et al. "Automatic Uniform Quantum State Preparation Using Decision Diagrams". In: *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)* (2020). DOI: 10.1109/ismvl49045.2020.00-10.
- Bruno Schmitt and Giovanni De Micheli. "Tweedledum: A Compiler Companion for Quantum Computing". In: *2022 Design, Automation Test in Europe Conference Exhibition (DATE)* (2022). DOI: 10.23919/date54114.2022.9774510.
- Bruno Schmitt et al. "From Boolean functions to quantum circuits: A scalable quantum compilation flow in C++". In: *2021 Design, Automation Test in Europe Conference Exhibition (DATE)* (2021). DOI: 10.23919/date51398.2021.9474237.

references II

Mathias Soeken et al. "The EPFL Logic Synthesis Libraries.". In: *arXiv: Logic in Computer Science* (2022). DOI: null. URL: https://arxiv.org/abs/1805.05121.

END
Thank you