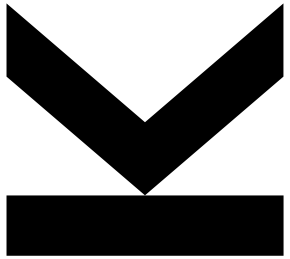
INTRODUCTION TO REVERSIBLE LOGIC SYNTHESIS



Robert Wille

Johannes Kepler University Linz, Austria

robert.wille@jku.at

FURTHER READING

- **Good overview (state-of-the-art report of the COST Action):**
  https://github.com/COST-IC1405/wg3-soar-report/blob/master/README.md
- **Embedding**
  Embedding of Large Boolean Functions for Reversible Logic, JETC 2016
  Make It Reversible: Efficient Embedding of Non-reversible Functions. DATE 2017
- **Transformation-based Synthesis**
  A Transformation Based Algorithm for Reversible Logic Synthesis, DAC 2003
  A fast symbolic transformation based algorithm for reversible logic synthesis, RC 2016.
- **BDD-based Synthesis**
  BDD-based Synthesis of Reversible Logic for Large Functions, DAC 2009
- **One-pass Synthesis**
  One-pass Design for Reversible Circuits: Combining Embedding and Synthesis for Reversible Logic, TCAD 2017
- **QMDD Data-structure**
  QMDDs: Efficient Quantum Function Representation and Manipulation, TCAD 2016
  Implementation available at http://www.informatik.uni-bremen.de/agra/eng/qmdd.php
- **SyReC HDL**
  SyReC: A Hardware Description Language for the Specification and Synthesis of Reversible Circuits, INTEGRATION 2016

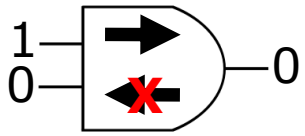
OUTLINE

- What is Reversible Logic?
- ~~Why Reversible Logic?~~ → Tomorrow (Applications)
- How to Design of Reversible Logic?
- What‘s Next?

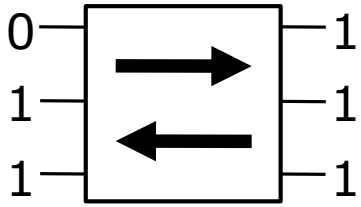
ALTERNATIVE: REVERSIBLE LOGIC

- Computations are reversible (bijective)
  - Same number of inputs/outputs
  - 1:1 mapping

AND-Gate
(irreversible)

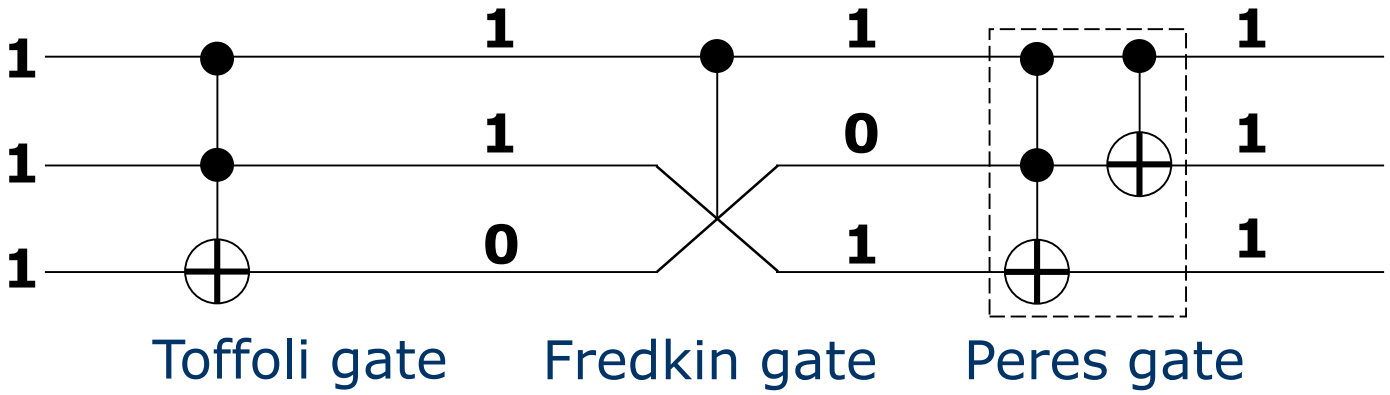


Reversible
Logic



REVERSIBLE CIRCUITS

- Fanout and feedback not directly allowed
- Cascade of reversible gates

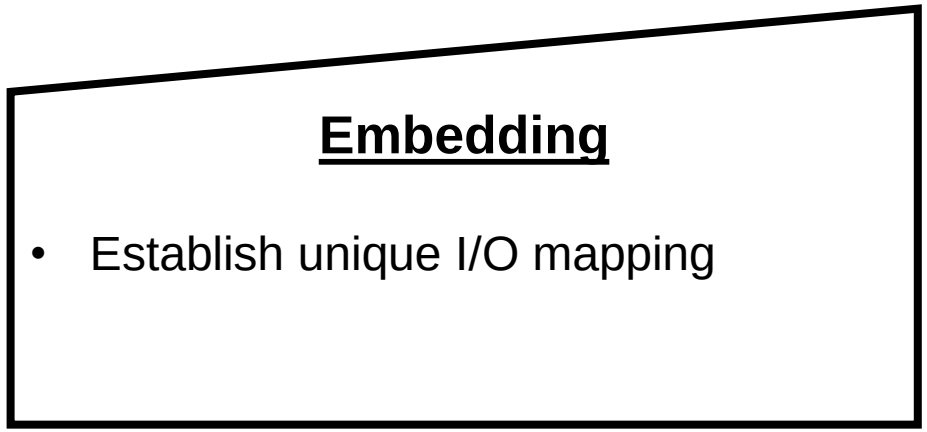


OUTLINE

- What is Reversible Logic?
- ~~Why Reversible Logic?~~ → Tomorrow (Applications)
- **How to Design of Reversible Logic?**
- What‘s Next?

HOW TO DESIGN REVERSIBLE LOGIC?

$f: B^n \rightarrow B^m$
to be synthesized



THE EMBEDDING PROBLEM

- How to describe conventional logic?

c_{in}	x	y	c_{out}	sum	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	?
1	0	1	1	0	1
1	1	0	1	0	?
1	1	1	1	1	1

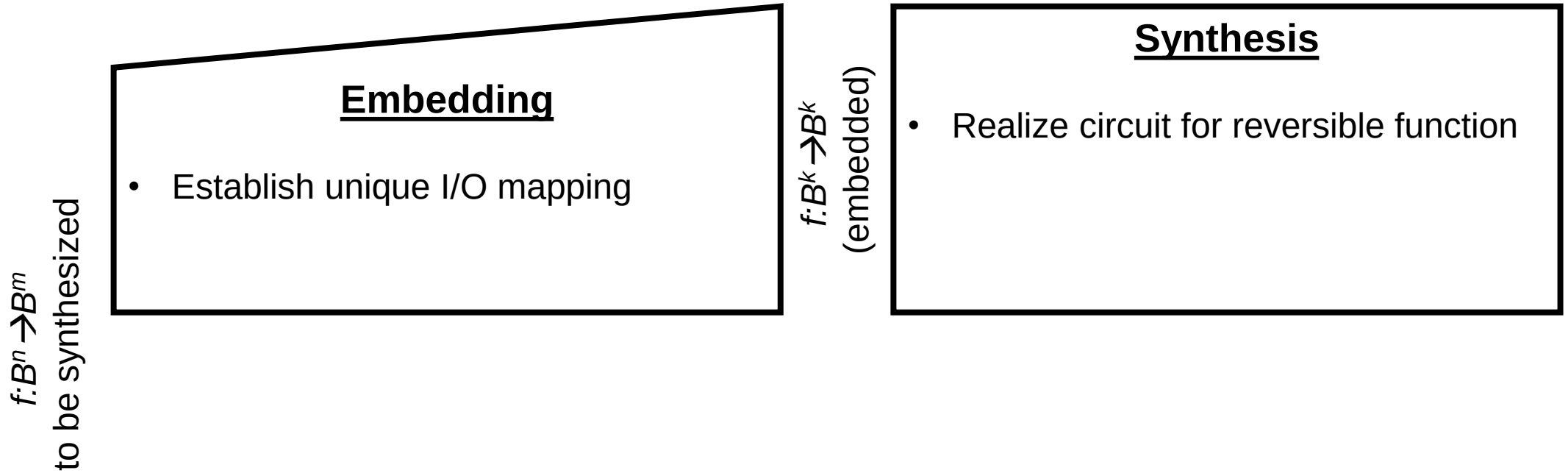
- Adding $\lceil \log M \rceil$ outputs where M is the largest number of equal output patterns

THE EMBEDDING PROBLEM

a	c_{in}	x	y	c_{out}	sum	g_1	g_2
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	1	0	0	0
0	1	0	0	0	1	1	0
0	1	0	1	1	0	0	1
0	1	1	0	1	0	1	0
0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	0
1	0	1	0	0	0	1	1
1	0	1	1	0	1	1	1
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1

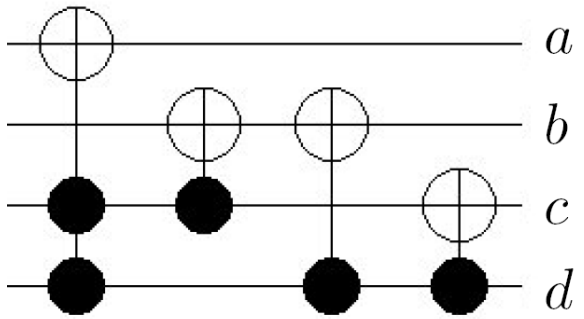
JKU

HOW TO DESIGN REVERSIBLE LOGIC?



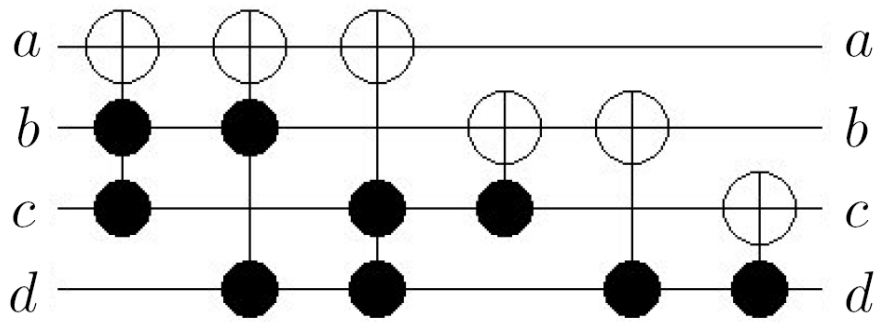
SYNTHESIS

line (i)	input abcd	output abcd	1^{st} step abcd	2^{nd} step abcd	3^{rd} step abcd
0	0000	0000	0000	0000	0000
1	0001	0111	0101	0001	0001
2	0010	0110	0110	0110	0010
3	0011	1001	1011	1111	1011
4	0100	0100	0100	0100	0100
5	0101	1011	1001	1101	1101
6	0110	1010	1010	1010	1110
7	0111	1101	1111	1011	1111
8	1000	1000	1000	1000	1000
9	1001	1111	1101	1001	1001
10	1010	1110	1110	1110	1010
11	1011	0001	0011	0111	0011
12	1100	1100	1100	1100	1100
13	1101	0011	0001	0101	0101
14	1110	0010	0010	0010	0110
15	1111	0101	0111	0011	0111

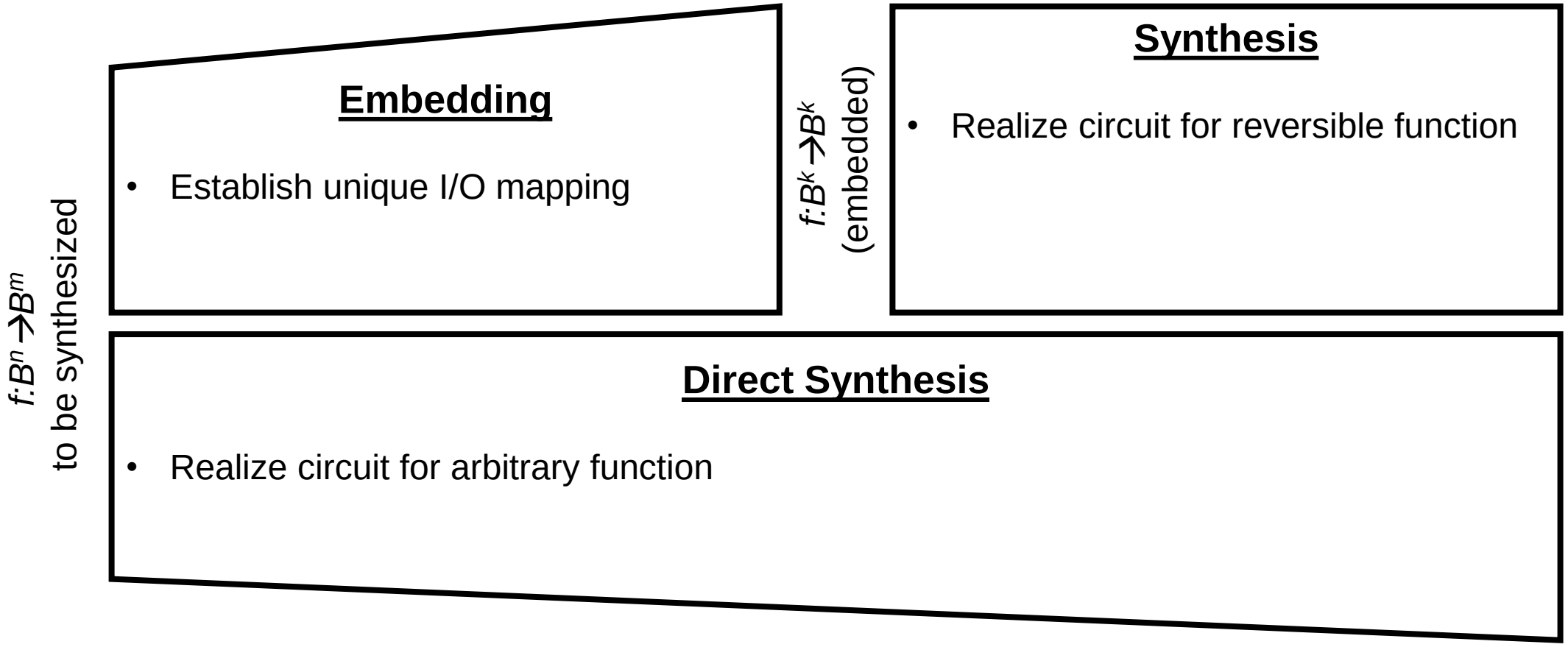


SYNTHESIS

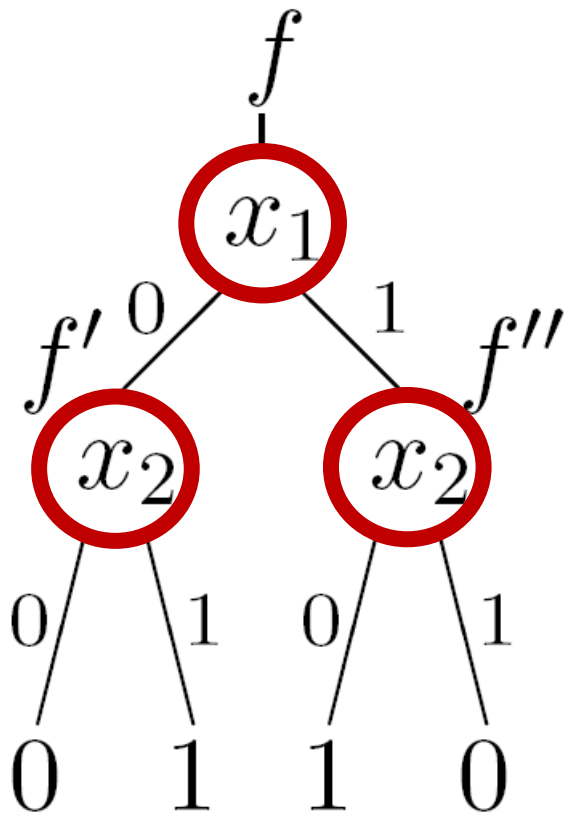
line (i)	input abcd	output abcd	6^{th} step abcd
0	0000	0000	0000
1	0001	0111	0001
2	0010	0110	0010
3	0011	1001	0011
4	0100	0100	0100
5	0101	1011	0101
6	0110	1010	**0**110
7	0111	1101	**0**111
8	1000	1000	1000
9	1001	1111	1001
10	1010	1110	1010
11	1011	0001	1011
12	1100	1100	1100
13	1101	0011	1101
14	1110	0010	**1**110
15	1111	0101	**1**111

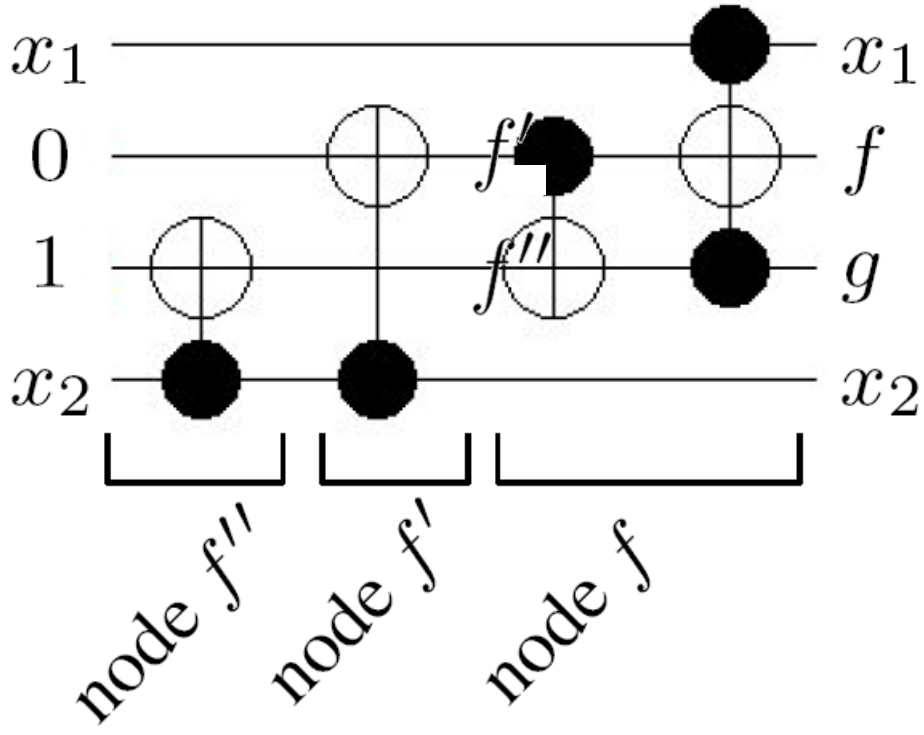


HOW TO DESIGN REVERSIBLE LOGIC?

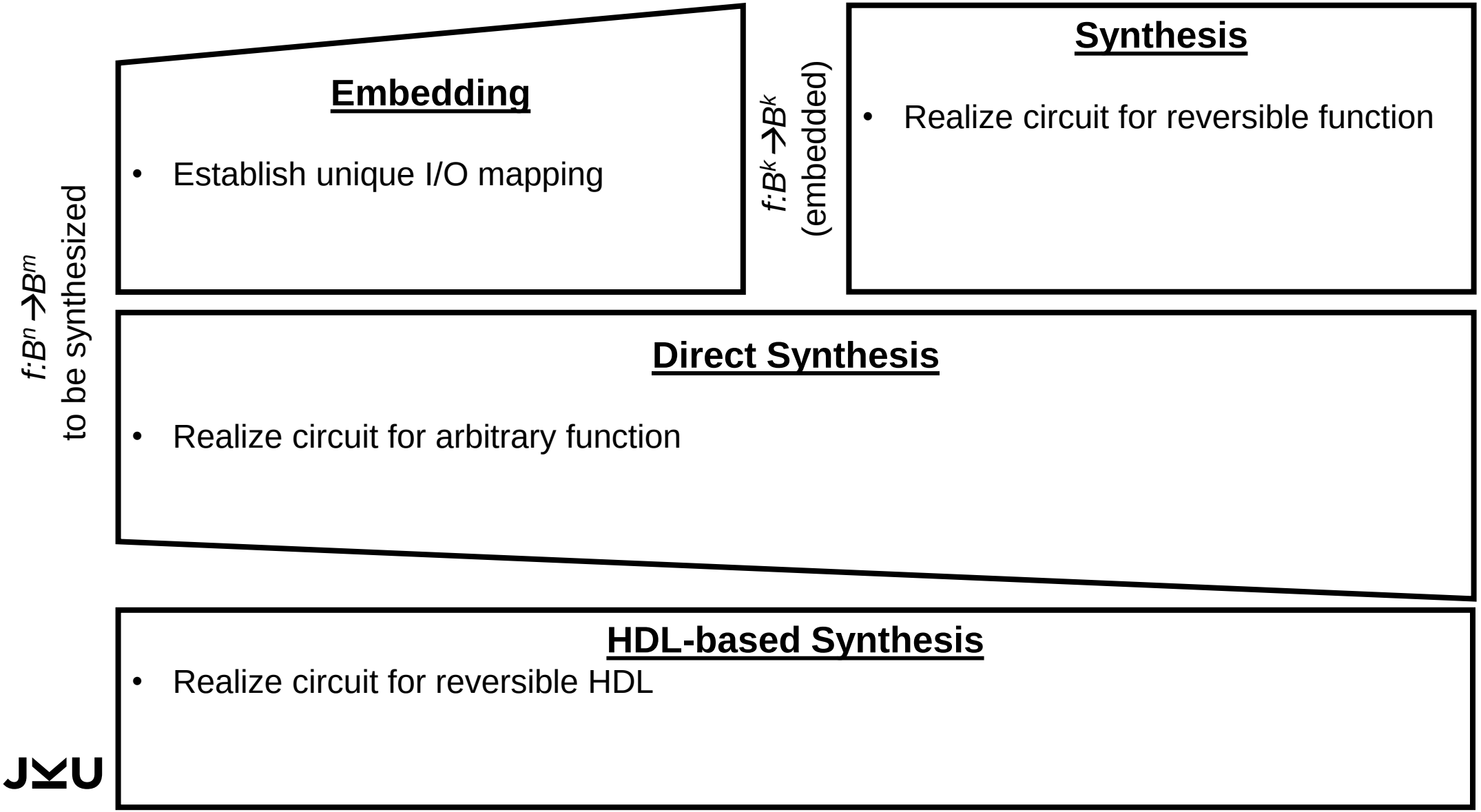


DIRECT SYNTHESIS



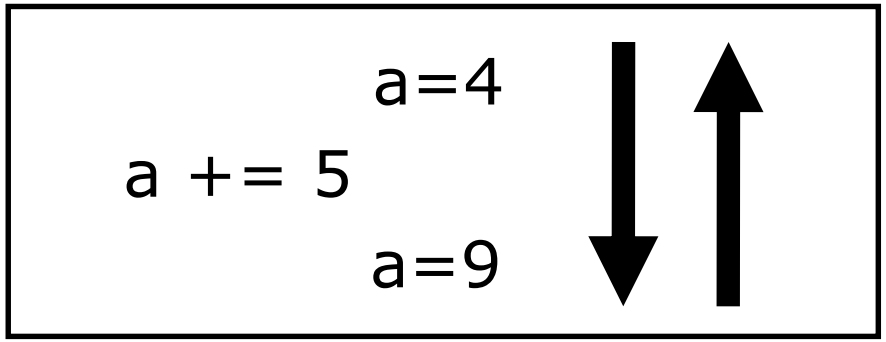


HOW TO DESIGN REVERSIBLE LOGIC?

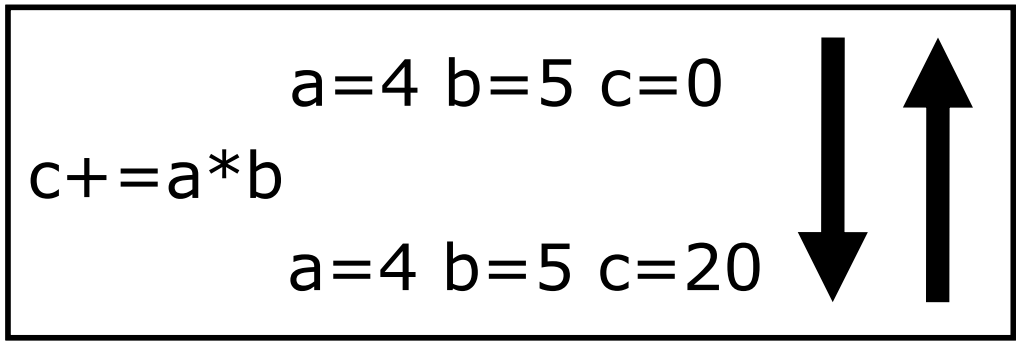


HDL-BASED SYNTHESIS

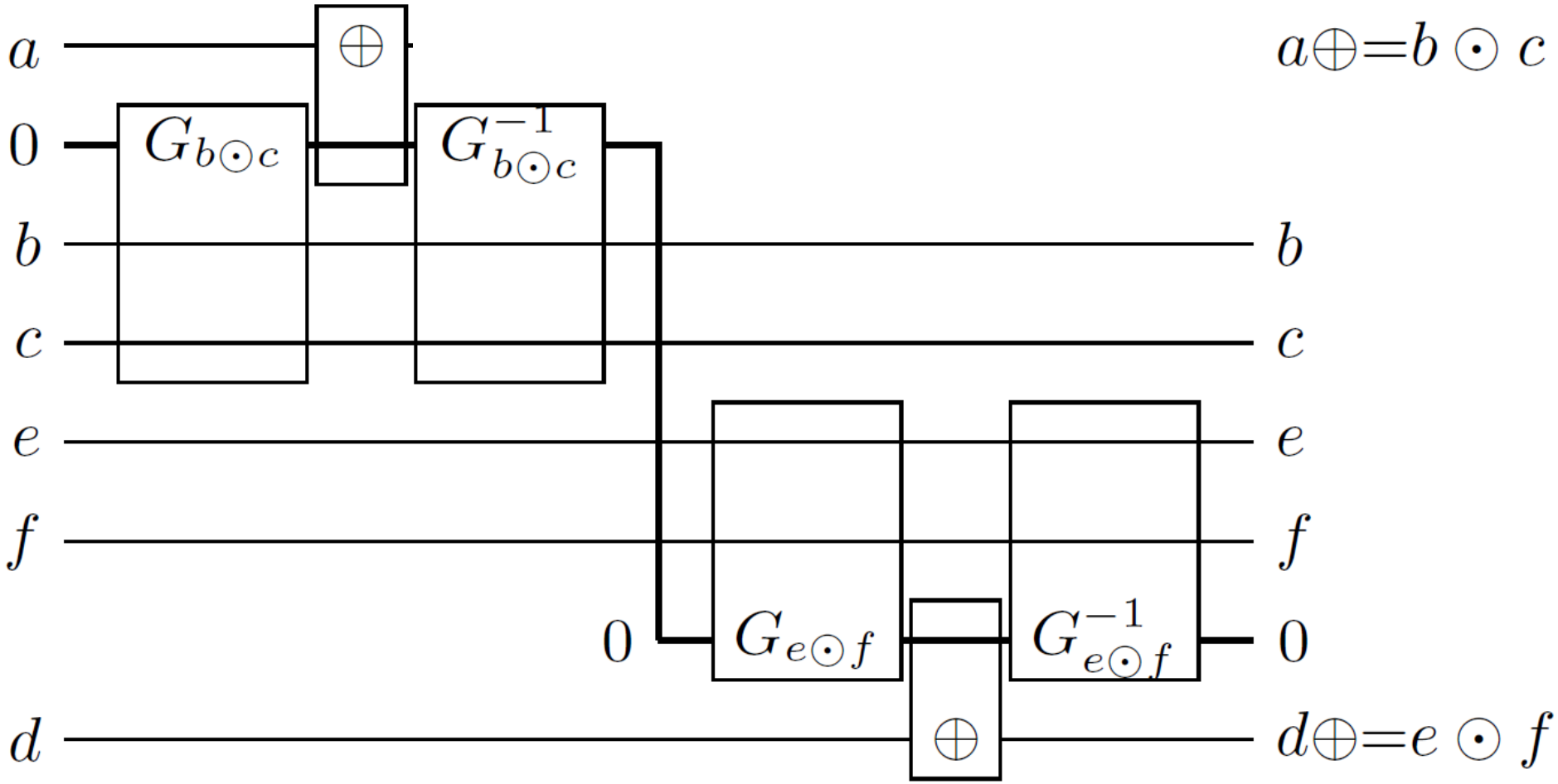
- Reversible assignment operations



- Binary operations



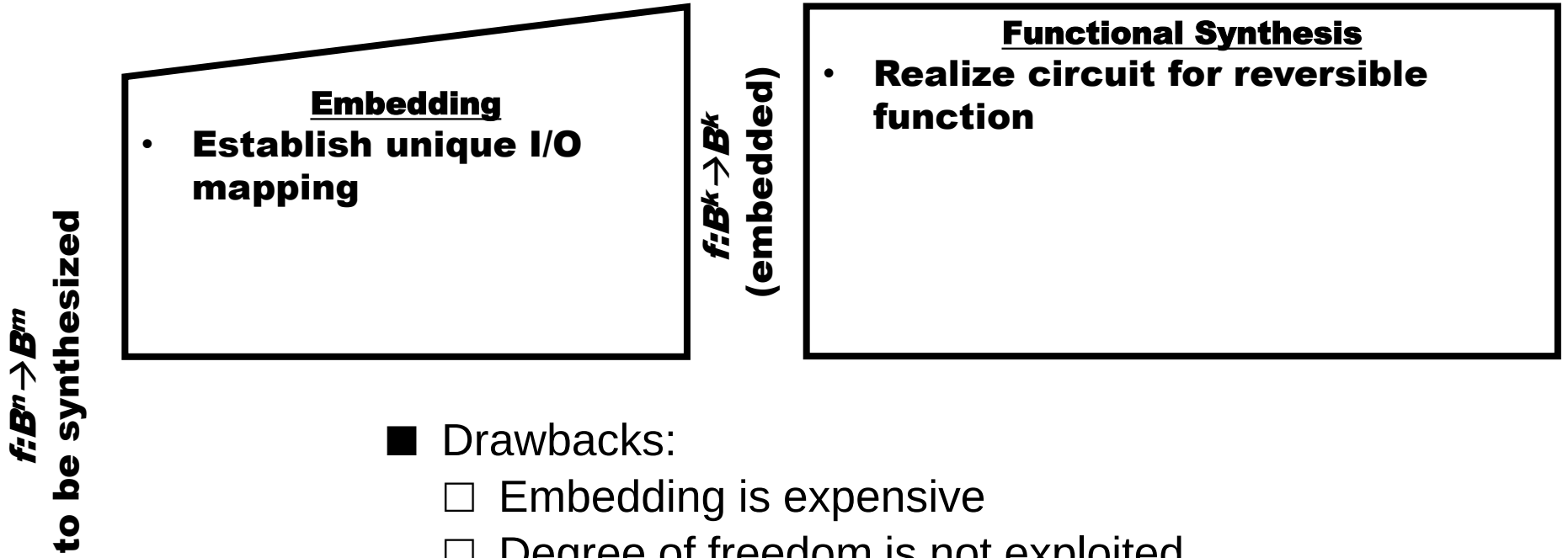
HDL-BASED SYNTHESIS



OUTLINE

- What is Reversible Logic?
- ~~Why Reversible Logic?~~ → Tomorrow (Applications)
- How to Design of Reversible Logic?
- **What‘s Next?**

SUMMARY: DESIGN OF REVERSIBLE CIRCUITS

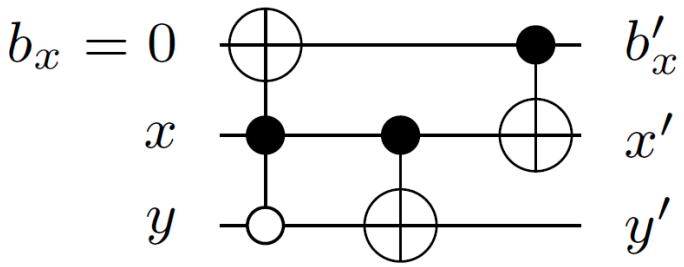


- Drawbacks:
  - Embedding is expensive
  - Degree of freedom is not exploited
  - Exponential growth of representation
- Alternative: One-pass Synthesis

ONE-PASS SYNTHESIS

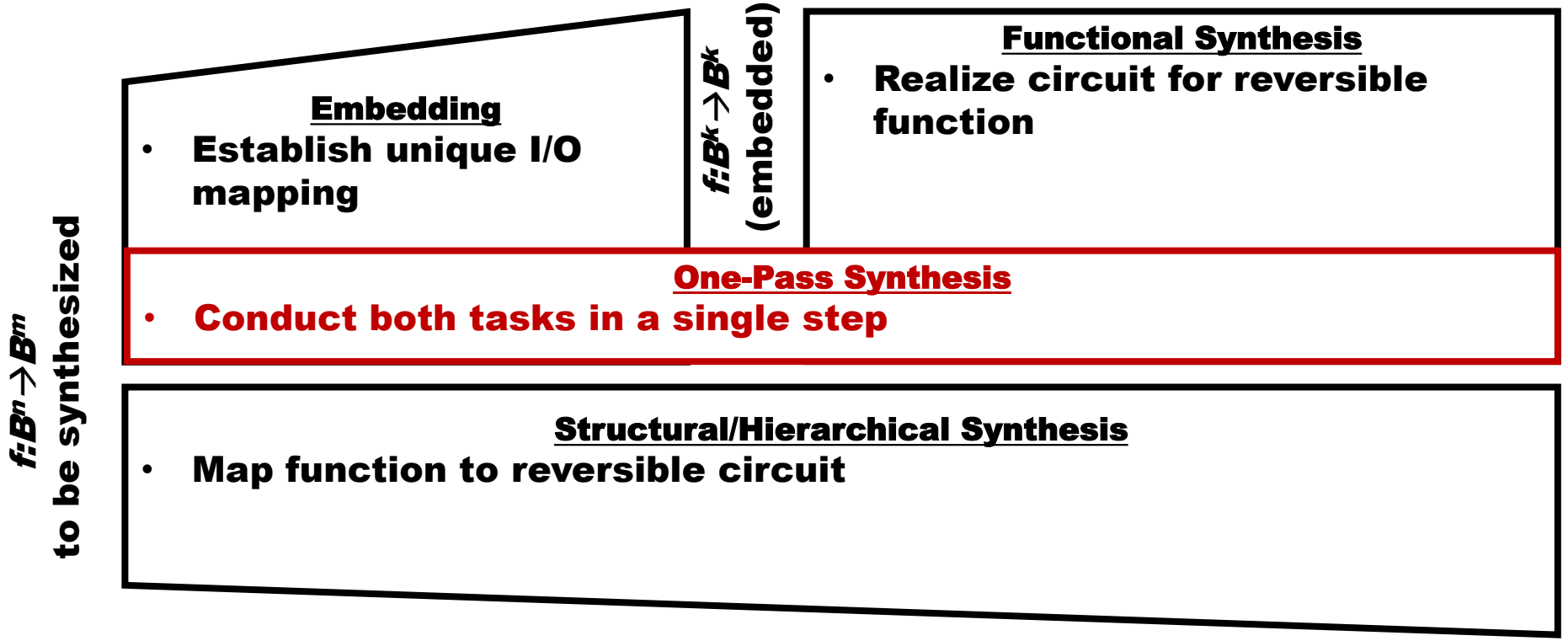
- Example: Transformation-based Synthesis

line (i)	input xy	output xy
0	00	00 ✓
1	01	01 ✓
2	10	10 ✓
3	11	11 ✓



- Start synthesis without embedding
- Modify function if problem occurs
  - Store changes on buffer line
- Revert changes after synthesis
  - One gate for each buffer line
- **Can be applied to many different synthesis approaches!**

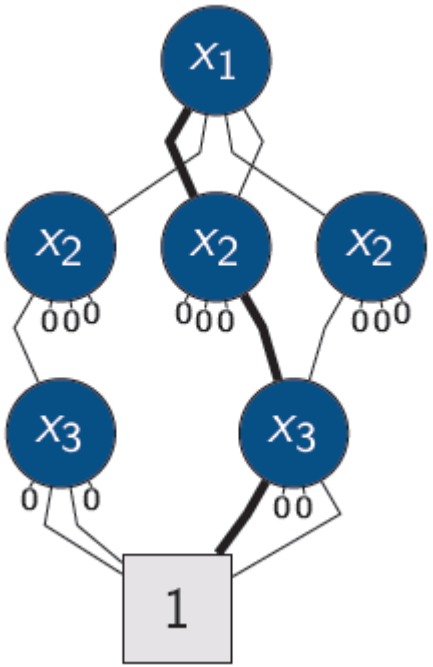
SUMMARY: DESIGN OF REVERSIBLE CIRCUITS



- Alternative data-structures?

REPRESENTATION OF REVERSIBLE FUNCTIONS

- Quantum Multiple-valued Decision Diagrams
  (publicly available at http://www.informatik.uni-bremen.de/agra/eng/qmdd.php)



ADJUSTED DESIGN OF REVERSIBLE CIRCUITS

- Work with permutation matrices!
  - Suitable representation for reversible functionality
  - Can efficiently be represented and manipulated using QMDDs

EXAMPLE: DETERMINATION OF #GARBAGE

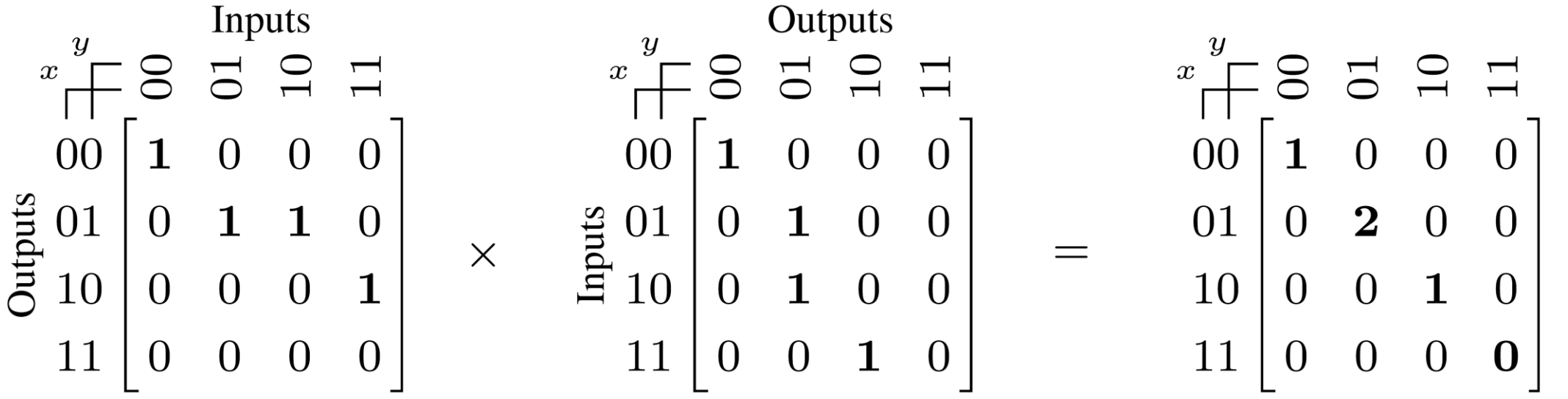
- Determine the number of garbage outputs
  - Counting the number of 1‘s in all rows
  - Equal to multiplying matrix M with transposition of M

Outputs \ Inputs (x \ y)	00	01	10	11
00	**1**	0	0	0
01	0	**1**	**1**	0
10	0	0	0	**1**
11	0	0	0	0

(a) M

EXAMPLE: DETERMINATION OF #GARBAGE

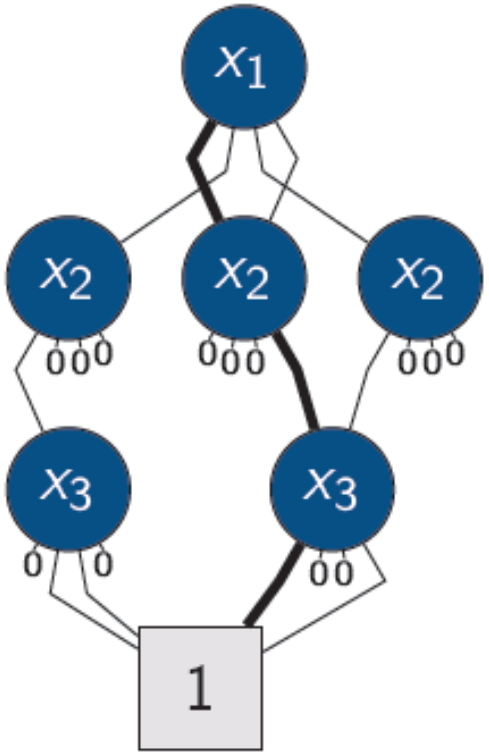
- Determine the number of garbage outputs
  - Counting the number of 1‘s in all rows
  - Equal to multiplying matrix M with transposition of M



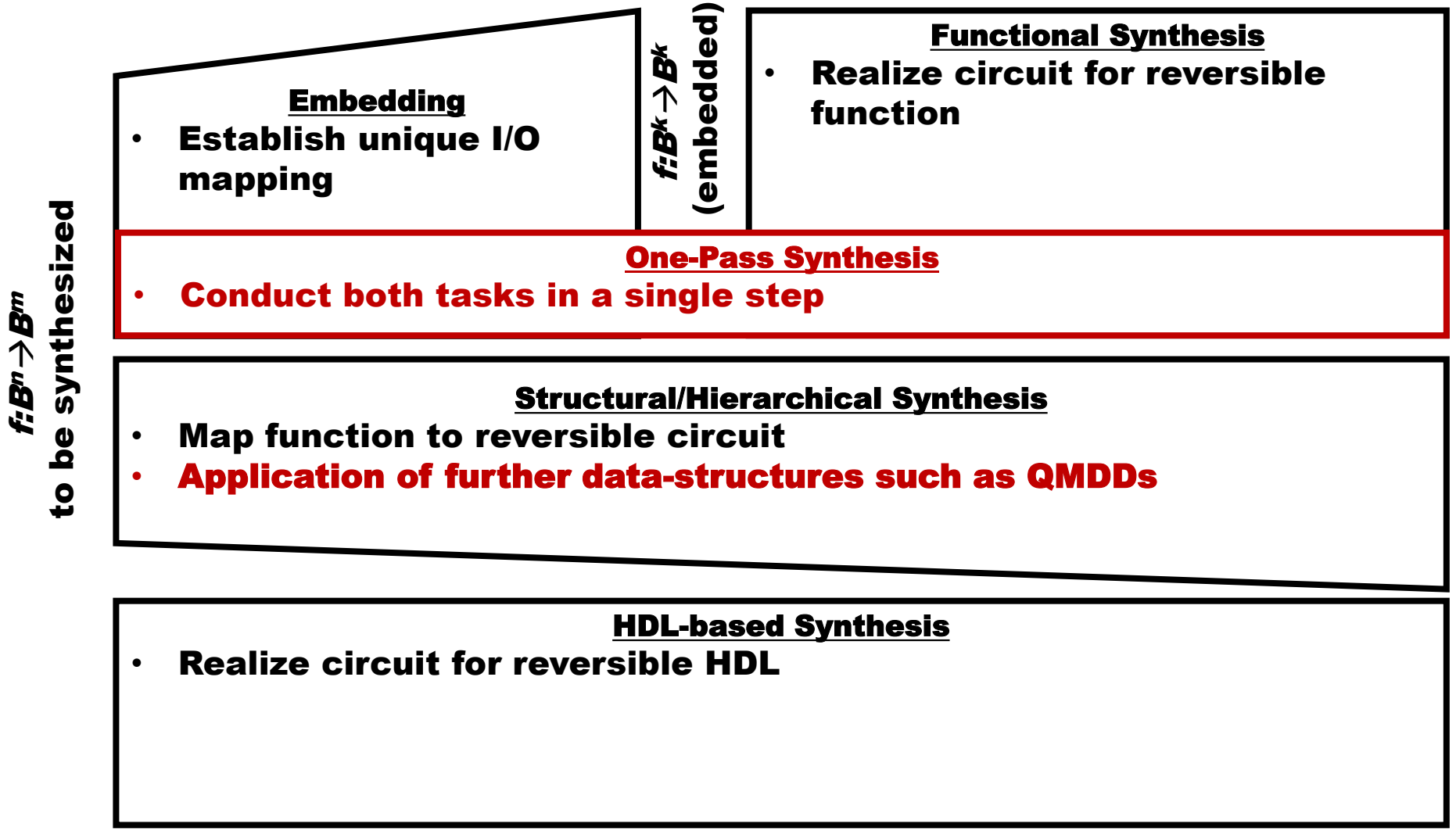
(a) M (b) M^T (c) $D = M \cdot M^T$

QMDDS: FURTHER APPLICATIONS

- Embedding
- Synthesis
- Verification
- Simulation
- …

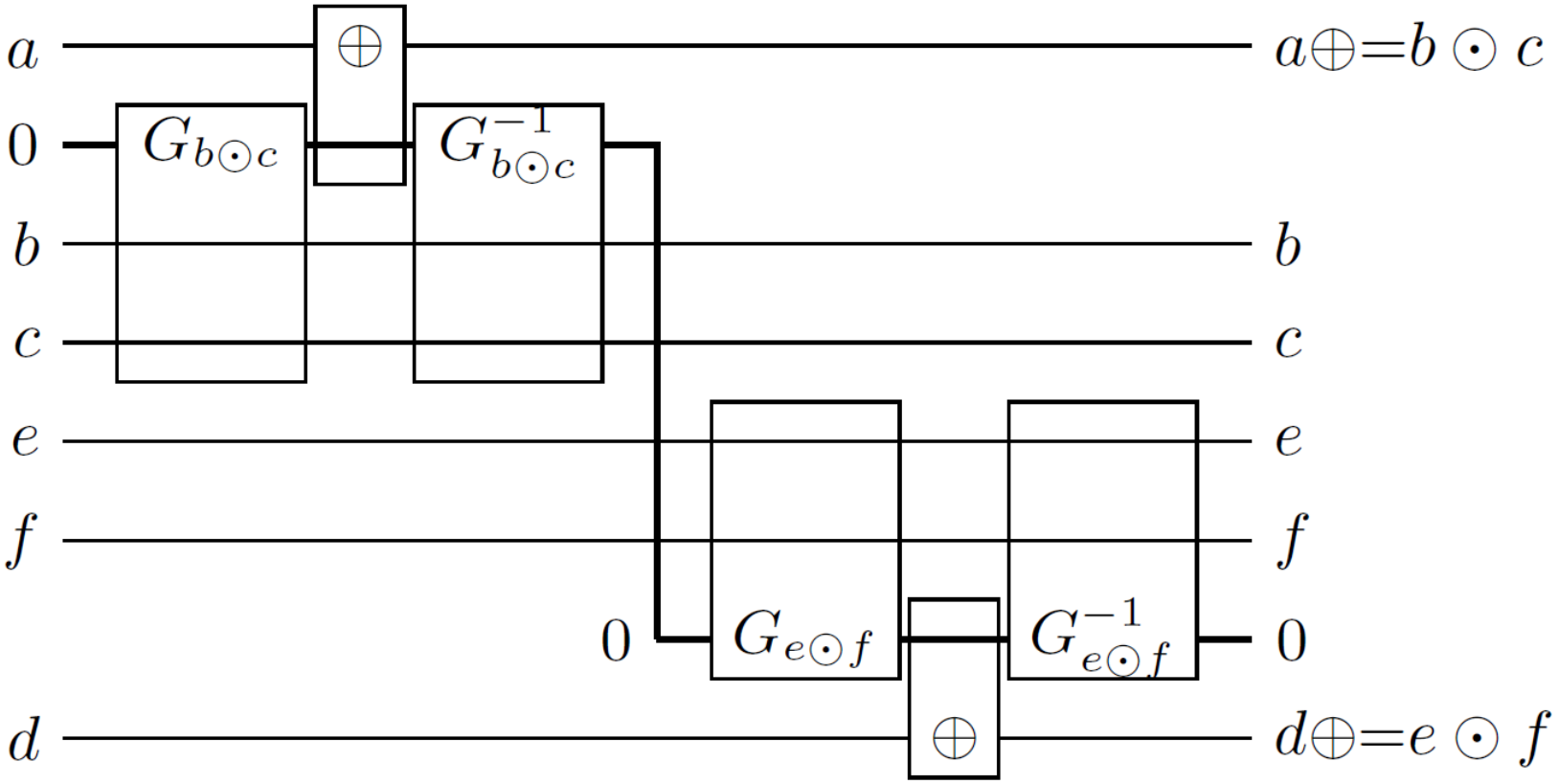


SUMMARY: DESIGN OF REVERSIBLE CIRCUITS



HDL-BASED SYNTHESIS

- Distinction between reversible assignment operations and binary operations



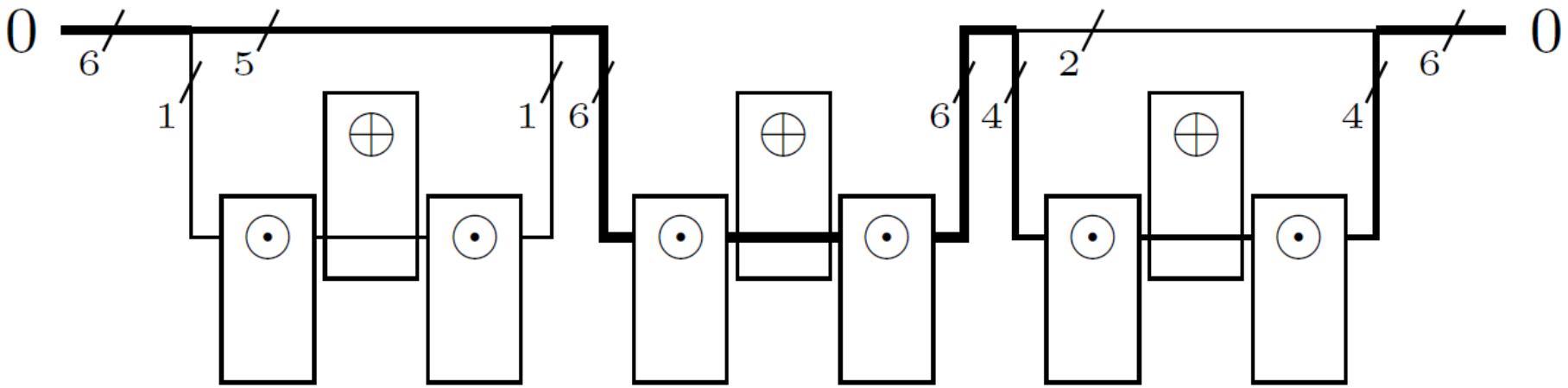
GENERAL OBSERVATIONS

- Reversible HDL descriptions are inherently reversible

➔ There must be a reversible circuit with zero additional lines realizing the described functionality

SEQUENCE OF STATEMENTS

- Consider a sequence of three statements
- Assume their realization requires 1, 6, and 4 additional lines



➔ Focus on optimizing the realization of the “largest” statement!

SIMPLE OPTIMIZATION

```
a += ((b & c) + ((d * e) - f))
```

- Intermediate results of the inner expressions must be buffered
- Considering 32-bit signals, this requires 96 circuit lines
- Plus 32 circuit lines to buffer the result of the outer expression
- → 128 circuit lines in total

```
a += (b & c);
a += (d * e);
a -= f;
```

- Binary operations are applied separately with an assignment operation.
- Hence, no more than 32 circuit lines are needed to buffer the intermediate results

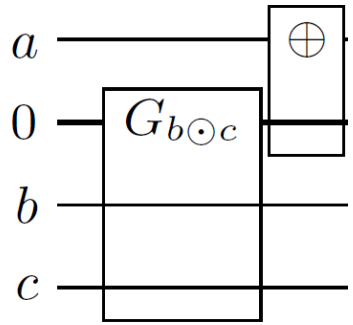
GENERAL OBSERVATIONS

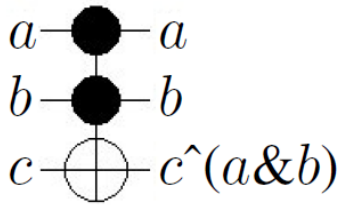
- Reversible HDL descriptions are inherently reversible

➔ There must be a reversible circuit with zero additional lines realizing the described functionality

- Thus far: The realization of the assignment and the expression is considered separately

c^= a & b





(c) c^= a & b

➔ Not possible for each combination of assignment/expr.

REVERSIBLE HDL-BASED SYNTHESIS

- **Scalable** synthesis of **efficient** circuits
  is THE big challenge in reversible circuit design
- Does not necessarily require a fully-fledged
  HDL-synthesizer
- Not so much about generating building blocks, but using/combining them in a clever fashion
- Use of techniques from compiling, term re-writing, etc.

SUMMARY: DESIGN OF REVERSIBLE CIRCUITS