# Taking One-to-one Mappings for Granted: Advanced Logic Design of Encoder Circuits

Alwin Zulehner[1] Robert Wille[1,2]
[1]Institute for Integrated Circuits, Johannes Kepler University, Linz, Austria
[2]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
alwin.zulehner@jku.at robert.wille@jku.at

***Abstract*—Encoders play an important role in many areas such as memory addressing, data demultiplexing, or for interconnect solutions. However, design solutions for the automatic synthesis of corresponding circuits suffer from various drawbacks, e.g. they are often not scalable, do not exploit the full degree of freedom, or are applicable to realize certain codes only. All these problems are caused by the fact that existing design solutions have to explicitly guarantee a one-to-one mapping. In this work, we propose an alternative design approach which relies on dedicated description means for both, the specification of an encoder as well as its circuit. Based on that, synthesis can be conducted without the need to explicitly take care of guaranteeing one-to-one mappings. Experiments show that this indeed overcomes the drawbacks of current design solutions and leads to an improvement in the resulting number of gates by up to 92%.**

## I. Introduction

Encoding devices represent a vital part of numerous applications realized in today's electronic systems such as addressing memories and caches, data demultiplexing, etc. (see [13], [3]). With the rise of *System on Chip* and *Network on Chip* architectures, they gained further importance by the fact that those architectures usually rely on a rather sophisticated interconnect solutions [2], [7], [12], [8]: In order to address the underlying limitations with respect to performance and/or power efficiency, communications between components of those architectures is usually conducted by dedicated encodings instead of the originally provided raw data. All these applications eventually motivate the design automation of corresponding encoders, i.e. the automatic synthesis of circuits which transform an $n$-bit word into another $n$-bit word while, at the same time, guarantee a one-to-one mapping from inputs to outputs.

However, how to specify and, afterwards, synthesize the corresponding encoders is a non-trivial task. In contrast to other circuit designs, encoders require a more or less explicit consideration of all possible patterns. In its most straightforward fashion, this can be conducted by a *complete* specification in terms of a truth-table as shown in Table Ia. This however results in an exponential complexity for the specification alone (not to mention the actual synthesis) – a rather unsuitable solution for practically relevant encoders.

As a compromise, *incomplete encodings* (also known as *discretized* or *approximate* encodings; see [1]) have been considered where the desired input/output mapping is specified for a (non-exponential) selection of the patterns only (e.g. the most important ones). For all other input patterns, the corresponding output patterns are considered don't care. Fig. Ib shows an example of such an encoding. While this allows for a compact specification of the desired encoder, it still poses significant challenges to the synthesis process: Although all non-specified patterns are don't care, a valid one-to-one mapping has to be guaranteed for them as well (otherwise a code would result that cannot be decoded anymore). As this, again, leads to an exponential complexity, often rather simple solutions are enforced for this purpose (e.g. enforcing the realization of the identity function for the remaining patterns whenever possible).[1] This, in turn, significantly reduces the degree of freedom to be exploited during the synthesis (suddenly a circuit has to be synthesized where, again, *all* patterns are restricted) and, hence, yields rather expensive circuits.

[1]This requires that the specified mapping from inputs to outputs is cyclic.

A completely different approach to deal with this problem is based on *clustering* [2], [1]. Here, the $n$-bit specification of the desired encoder is provided by several clusters of encoders with significantly smaller size. Table Ic illustrates the main idea: Instead of defining the encoder for all $n = 3$ input/output combinations, the mapping of the first input/output (i.e. $x_3$) and the last two inputs/outputs (i.e. $x_2x_1$) are provided separately. While this addresses the exponential complexity, this scheme is not generically applicable to all possible encodings. For example, if e.g. a 2-bit encoder realizing the mapping $\{(00,10),(01,00),(10,01),(11,11)\}$ is desired, no clustering can be conducted since the relation between the two bits is entangled.

Besides the problems reviewed above, many synthesis methods for encoders do not exploit the full potential provided by the given design task. In many domains such as the design of low-power encoders, not an explicit input/output mapping is desired, but only one which satisfies a dedicated objective. For example, specifications derived from probability-based mapping [14] only require that certain inputs are getting mapped to outputs of a dedicated Hamming weight (as illustrated at the left-hand side of in Table Id). But since, again, existing synthesis methods require an explicit specification of the input/output mapping, those specifications are usually mapped to an explicit specification prior to synthesis (as illustrated at the right-hand side of in Table Id). By this, all the shortcomings reviewed above are inherited, i.e. also these solutions have a limited scalability and/or do not exploit the full degree of freedom yielding to rather expensive circuits.

Overall, state-of-the-art methods for the design of encoders suffer from the fact that

- they are not scalable (if complete specifications are considered),
- they do not exploit the full degree of freedom (because fix input/output mappings are enforced initially if incomplete encoders or encoders based on Hamming weights are considered), or
- they are not generically applicable (if clustering is to be applied to encoders where inputs/outputs are entangled).

All these problems are actually caused by the fact that existing solutions rely on rather unsatisfactory methods for guaranteeing the needed one-to-one mapping.

In this work, we provide a solution which overcomes these problems. To this end, we first utilize an alternative description means in Section II that allows for a compact specification of the desired encoders while, at the same time, considers the full degree of freedom (e.g. provided by don't care cases, more flexible descriptions e.g. in terms of Hamming weight, etc.). Based on that, a synthesis method is sketched in Section III and detailed in Section IV which realizes the desired encoder while, at the same time, inherently guarantees a one-to-one mapping. To this end, circuit descriptions based on so-called *reversible logic* are utilized. This way, the actual design objective can be specified (including don't care cases, Hamming

TABLE I: Specifications of encoders

| (a) Complete | | (b) Incomplete | | (c) Clustering | | | | (d) H. weight | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_3x_2x_1$ | $x_3x_2x_1$ | $x_3x_2x_1$ | $x_3x_2x_1$ | $x_3$ | $x_2x_1$ | $x_3$ | $x_2x_1$ | $x_3x_2x_1$ | Hw | | $x_3x_2x_1$ |
| 000 | 000 | 000 | 000 | 0 | 00 | 1 | 01 | 000 | 1 | | 001 |
| 001 | 101 | 001 | – | 0 | 01 | 1 | 11 | 001 | 0 | | 000 |
| 010 | 011 | 010 | 010 | 0 | 10 | 1 | 00 | 010 | 2 | | 011 |
| 011 | 111 | 011 | – | 0 | 11 | 1 | 10 | 011 | 3 | | 111 |
| 100 | 010 | 100 | – | 1 | 00 | 0 | 01 | 100 | 2 | $\Rightarrow$ | 110 |
| 101 | 100 | 101 | 001 | 1 | 01 | 0 | 11 | 101 | 1 | | 010 |
| 110 | 110 | 110 | – | 1 | 10 | 0 | 00 | 110 | 2 | | 101 |
| 111 | 001 | 111 | – | 1 | 11 | 0 | 10 | 111 | 1 | | 100 |

weights, etc.) without the need to explicitly take care of guaranteeing one-to-one mappings. Experimental evaluations summarized in Section V show that this solution indeed allows for the design of encoders, for which the number of gates in the resulting circuits is reduced by up to 92% (approx. 60% on average) compared to conventional approaches.

## II. A Decision Diagram for the Specification of Encoders

In this section, we show how to efficiently represent the specification of the desired encoders without the need to explicitly take care of guaranteeing one-to-one mappings. To this end, we introduce a description means which has been inspired from so-called *Quantum Multiple-Valued Decision Diagrams* (QMDDs) introduced in [9], [11]. QMDDs allow for the efficient representation and manipulation of quantum computations – a future way of doing computations which relies on quantum-mechanical characteristics such as dedicated quantum bits, superposition, entanglement, etc. (see [10] for details). Moreover, since all quantum computations are inherently reversible, all descriptions of quantum functionality (and, hence, also QMDDs) do have to guarantee one-to-one mappings.

In this work, we are exploiting this characteristic in order to address the problem of how to represent a one-to-one mapping when designing coders. More precisely, we took the original definition of QMDDs and removed all quantum-related issues such as working with quantum values, superposition, etc. This eventually led to a decision diagram – called *1:1DD* in the following – in which the specification of an encoder is represented as a one-to-one mapping matrix $M$ rather than a truth table. In this matrix, the $n$-bit inputs and $n$-bit outputs are represented by columns and rows of the matrix, respectively. A 1-entry in the matrix indicates that an input (column) is mapped to an output (row). In contrast, a 0-entry indicates that there is no relation between the input and the output. The following example illustrates the idea.

**Example 1.** *Consider the specification of a fully specified encoder as shown in Table Ia. The corresponding matrix representation $M$ is shown in Fig. 1a. Since the encoder maps e.g. input $x_3x_2x_1 = 010$ to output $x'_3x'_2x'_1 = 011$, the third column of the matrix contains its 1-entry in the fourth row.*[2]

To efficiently represent $M$, 1:1DDs provide a decision diagram structure where each node partitions the matrix according to a variable $x_i$ ($n \geq i \geq 1$). More precisely, let's assume $x_n$ is the most significant variable of $M$. Then, the matrix can be decomposed into four sub-matrices, where each of them represents one of the four possible mappings of $x_n$, i.e.

- from 0 to 0 (left upper sub-matrix; denoted $M_{0\to 0}$),
- from 1 to 0 (right upper sub-matrix; denoted $M_{1\to 0}$),
- from 0 to 1 (left lower sub-matrix; denoted $M_{0\to 1}$), and
- from 1 to 1 (right lower sub-matrix; denoted $M_{1\to 1}$).

Each of these sub-matrices are again represented in terms of a node in the decision diagram. By recursively continuing

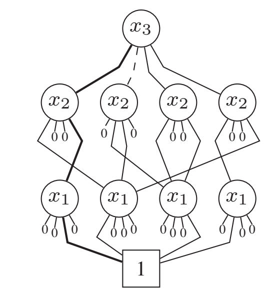
[2]Since we are considering input/output mappings with equal bit-widths in the following, we will not distinguish e.g. between an input $x_i$ and output $x'_i$ anymore. Instead, we consistently denote that by the mapping of the variable $x_i$.

Inputs

$$\begin{array}{c|cccc:cccc}
x_3x_2x_1 & \mathbf{0}00 & \mathbf{0}01 & \mathbf{0}10 & \mathbf{0}11 & \mathbf{1}00 & \mathbf{1}01 & \mathbf{1}10 & \mathbf{1}11 \\
\hline
\mathbf{0}00 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{0}01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{0}10 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
\mathbf{0}11 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
\hdashline
\mathbf{1}00 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
\mathbf{1}01 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{1}10 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\
\mathbf{1}11 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0
\end{array}$$

Outputs

(a) Matrix



(b) 1:1DD

Fig. 1: Representations for the encoder specified in Table Ia

this partition, smaller sub-matrices result until a single value (i.e. a terminal) is reached. Since the resulting sub-matrices often include a significant amount of redundancies or are even identical, sharing is possible (similar to other decision diagrams such as BDDs [5]). Moreover, zero sub-matrices (i.e. matrices which are solely composed of 0's) frequently occur which, independently of their dimension, can simply be represented by a 0-stub. This eventually allows for a rather compact representation.

**Example 1** (continued)**.** *Fig. 1b shows the resulting 1:1DD representation of the matrix shown in Fig. 1a. Note that the successors of a node, i.e. the first, second, third, and fourth edge, represent $M_{0\to 0}$, $M_{1\to 0}$, $M_{0\to 1}$, and $M_{1\to 1}$, respectively.*

*As an example, the bold path to the 1-terminal*[3] *denotes that the mapping of the variables $x_3$, $x_2$, and $x_1$ from 0, 1, and 0 to 0, 1, and 1, respectively, is a valid input/output mapping of the function represented by the 1:1DD. On the other side, no mapping e.g. of the variables $x_3$ and $x_2$ from 1 and 0 to 0 and 0 exists, respectively (as represented by the path to the 0-stub highlighted with dashed lines).*

Using 1:1DDs as introduced above allows for the compact specification of *complete* encoders and, by this, addresses one of the main problems discussed in Section I. Moreover, 1:1DDs can also be utilized in order to represent incomplete specifications or specifications based on alternative objectives such as Hamming weights. More precisely:

- An incomplete specification can be represented by a 1:1DD where the columns for which no output is specified contain 0-entries only. As an example, Fig. 2a shows a 1:1DD representation of the specification from Table Ib.
- A specification based on Hamming weights can be represented by a 1:1DD following the convention that a Hamming weight of $h$ is represented by output $2^h - 1$. As an example, Fig. 2b shows a corresponding 1:1DD representation of the specification from Table Id.

Note that, in both cases, these 1:1DDs do not technically represent one-to-one mappings (in the former case, mappings are missing; in the latter case several mappings to the same output exist). However, they provide a description means which compactly represents all the relevant information that is needed for synthesis. How this can be utilized during the synthesis process itself is described next.

## III. Synthesis of the Encoders

The 1:1DDs introduced above allow for a compact specification of the desired encoders. Based on this, also their synthesis can be conducted. This requires an inherent guarantee that a one-to-one mapping is realized. For the synthesis step, we propose to accomplish that by using *reversible logic* [6], [15] – a circuit model which is composed of reversible gates and,

[3]For brevity, paths to the 1-terminal are called 1-paths in the following.

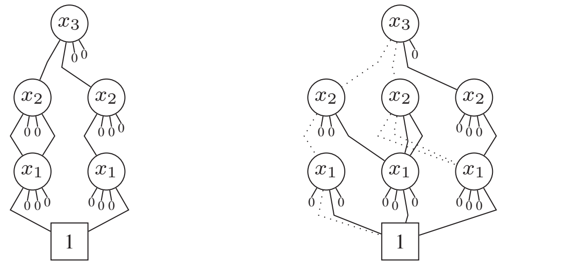

(a) Incomplete (c.f. Table Ib) (b) H. weight (c.f. Table Id)

Fig. 2: 1:1DDs for other encoder specifications

by this, inherently allows for the realization of one-to-one mappings only.[4] In this section, we first review the main concepts of this circuit model. Afterwards, the general idea of the proposed synthesis scheme which utilizes the description means introduced in Section II together with the reversible logic circuit model in order to realize the desired encoder are described. Details on the implementation are then provided in Section IV.

## A. Reversible Logic

Circuits realized in reversible logic are structurally different to conventional ones. They do not directly allow feedback and fanout and, hence, need to be described by $n$ circuit lines which are passed through a cascade of gates [6], [15]. Moreover, in order to ensure reversibility, each gate is allowed to realize a reversible function only. In the domain of reversible logic, the Toffoli gate is one of the most established gate types as it is reversible and universal, i.e. each reversible function can be realized with Toffoli gates only.

This eventually leads to the definition of a *reversible circuit* as a cascade $G = g_1g_2g_3\ldots g_k$ of $k$ reversible gates $g_i$ over a set of circuit lines $X = \{x_n, \ldots, x_2, x_1\}$. A *reversible gate* (here: *Toffoli gate*) $g_i = TOF(C_i, t_i)$ consists of a set $C_i \subseteq \{x_j \mid x_j \in X\} \cup \{\overline{x}_j \mid x_j \in X\}$ of positive ($x_j$) and negative ($\overline{x}_j$) *control lines* and a *target line* $t \in X$ with $\{t_i, \overline{t}_i\} \cap C_i = \emptyset$. A circuit line cannot be used as positive and negative control line at the same time. The value of the target line $t_i$ is inverted iff the value of all positive control lines $x_j \in C_i$ evaluate to 1 and all negative control lines $\overline{x}_j \in C_i$ evaluate to 0. All other lines pass through the gate unchanged. In the following, positive control lines, negative control lines, and the target line of a Toffoli gate are visualized with symbols ●, ○, and $\oplus$, respectively.

**Example 2.** *Fig. 3 shows a reversible circuit composed of three circuit lines and three Toffoli gates. Furthermore, the circuit is labeled with the values on the circuit lines for input $x_3x_2x_1 = 001$. The first gate $g_1 = TOF(\{x_1\}, x_3)$ inverts the value of the target line $x_3$ since the control line $x_1$ is initialized 1. Because of the same reason (control lines are accordingly initialized), the second gate $g_2 = TOF(\{x_3, \overline{x}_2\}, x_1)$ inverts target line $x_1$. In contrast, the third gate $g_3 = TOF(\{x_1\}, x_2)$ does not invert target line $x_2$, because the control line $x_1$ is 0.*

Because of their dedicated structure as well as gate types, reversible circuits allow for the realization of one-to-one mappings only. By this, they are inherently addressing the main problem discussed above: They do not require to explicitly take care of guaranteeing one-to-one mappings. At the same time, once a reversible circuit realizing the desired encoder is available, it can easily be converted into a conventional circuit description (since, after all, Toffoli gates realize conjunctions

[4]Note that, in the past, reversible logic has already been considered for designing encoders as e.g. in [16] or, more recently, in [17]. However, in both cases the design has been conducted based on descriptions such as truth-tables, i.e. only encoders of rather small size (up to 12 bits) have been considered.

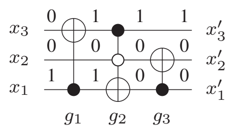


Fig. 3: Reversible circuit

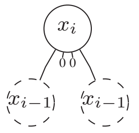


Fig. 4: Identity structure

of positive/negaive control signals exclusively-ORed with the target signal). This way, encoders can be synthesized by focusing on the objectives only (not on how to guarantee one-to-one mappings), while the result is still suitable for the further design steps.

## B. Resulting Synthesis Scheme

Summarizing the considerations from above, the synthesis task is reformulated as follows: For a 1:1DD representing the desired encoder in terms of a matrix $M$ (as introduced in Section II), determine a circuit $G = g_1g_2g_3\ldots g_k$ based on the model of reversible logic (as reviewed in Section III-A) which realizes $M$. This can be conducted by applying reversible gates $g_i$ to $M$ so that, eventually, the identity matrix $I$ results. In other words, let's assume a cascade of reversible gates $G^{-1} = g_kg_{k-1}\ldots g_2g_1$ applied to $M$ transforms $M$ into $I$. Then, due to the reversibility, the inverse cascade $G = g_1g_2g_3\ldots g_k$ realizes $M$.

This leaves the question how to efficiently determine the gates needed in order to transform $M$ to $I$. Since the identity $I$ only represents mappings from 0 to 0 and from 1 to 1, each node in the 1:1DD has to be transformed such that its second edge and third edge point to a 0-stub (as shown in Fig. 4 for a node $x_i$). Hence, for a given 1:1DD $M$, the task remains how to apply reversible gates so that eventually this structure results.

This task is addressed by successively transforming the 1:1DD towards the identity. To this end, the nodes of the 1:1DD are considered in a breadth-first traversal from the top to the bottom. In each step, the currently considered node (representing the partition according to variable $x_i$) is transformed into the desired structure. This is accomplished by applying Toffoli gates which move all 1-paths of the second and third edge to the first and fourth edge (eventually leading to nodes as shown in Fig. 4).

Obviously, the sequence of gates required to accomplish the desired structure for the currently considered node of the 1:1DD depends on the represented objective, i.e. whether a complete, incomplete, or Hamming weight-based specification is provided. Hence, details on this are separately provided later in Section IV. But since the main principles are similar in all these cases, how Toffoli gates affect the 1:1DD nodes (and how this can be utilized during synthesis) is briefly shown before.

A main principle is that Toffoli gates swap 1:1DD-paths. More precisely, applying e.g. a gate $TOF(C, x_i)$ to a given 1:1DD inverts the input of the mapping of variable $x_i$ for all paths represented by $C$. This way Toffoli gates may be used to swap e.g. a mapping from 0 to 1 to a mapping from 1 to 1 and, by this, moving paths from the third edge to the fourth edge – bringing it closer to the identity structure. Again, an example illustrates the idea.

**Example 3.** *The simplest gate $TOF(\emptyset, x_2)$ inverts the value, from which variable $x_2$ maps, for all paths and, therefore, simply exchanges the first (third) and the second (fourth) edge of the 1:1DD node (as illustrated in the left part of Fig. 5). The gate $TOF(\{x_1\}, x_2)$ inverts the value, from which variable $x_2$ maps, for all paths where variable $x_1$ maps from 1 to anything (as illustrated in the right part of Fig. 5). This already yields the identity structure for the top node of the 1:1DD in Fig. 5. In a similar fashion, various other modifications on the 1:1DD can be conducted.*

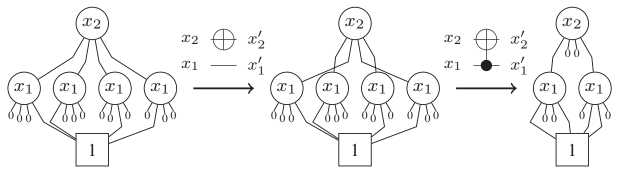


Fig. 5: Effects of applying Toffoli gates to 1:1DDs

In addition to that, we have to make sure that applying Toffoli gates does not affect previously traversed nodes. This can be accomplished by adding control lines describing the path to the currently considered node to each Toffoli gate.

**Example 4.** *Consider the 1:1DD in Fig. 5 and assume that the rightmost 1:1DD node with label $x_1$ is currently processed. Each gate applied for processing this node has to include a positive control line $x_2$. By this, only paths with $x_2 = 1$ are changed, i.e. paths which run through the fourth edge of the top node (representing a mapping from 1 to 1).*

## IV. Detailed Synthesis Schemes

Based on the general ideas discussed above, this section provides details on how the synthesis is conducted when either a complete, incomplete, or Hamming weight-based specification is provided. More precisely, we show how to determine a sequence of gates that transforms a node of the respectively given 1:1DD to the desired identity structure. Following the synthesis flow outlined in Section III and without loss of generality, we assume that the currently considered node is labeled with variable $x_i$ ($n \geq i \geq 1$) and that all previously traversed nodes (i.e. all nodes labeled with variable $x_l$, where $n \geq l > i$) already established the identity structure.

### *A. Case 1: Complete Specification*

The key idea is to swap paths such that all 1-paths are moved from the second to the first edge while, at the same time, all 1-paths from the third edge are moved to the fourth edge. To this end, we determine the sets of 1-paths for each edge of the currently considered node (denoted by $P_1$, $P_2$, $P_3$, and $P_4$, respectively) as well as the corresponding sets of 0-paths of the currently considered node (i.e. paths that terminate in a 0-stub; denoted by $\overline{P}_1$, $\overline{P}_2$, $\overline{P}_3$, and $\overline{P}_4$, respectively). A path represents an input and, hence, contains a literal for each variable $x_j$ with $1 \leq j < i$ which either occurs in positive phase ($x_j$) or negative phase ($\overline{x}_j$). Variable $x_i$ is neglected, because it is inherently known ($\overline{x}_i$ for paths in $P_1$ and $P_3$, as well as $x_i$ for paths in $P_2$ and $P_4$). Since the 1:1DD node describes a fully specified function, the sets $P_1$ and $P_3$ are disjoint (both represent a mapping with input $x_i = 0$), i.e. $P_1 \cap P_3 = \emptyset$. Moreover, the set of 0-paths $\overline{P}_1$ in the first edge is equal to the set of 1-paths $P_3$ in the third edge. Finally, since a completely specified encoder describes a reversible function, the cardinalities of the sets $P_2$ and $\overline{P}_1 = P_3$ have to be equal.

**Example 5.** *Consider the top node of the 1:1DD depicted in Fig. 1b (representing the complete encoder specification from Table Ia). The sets of inputs with a 1-path are defined by $P_1 = \{\overline{x}_2\overline{x}_1, x_2\overline{x}_1\}$, $P_2 = \{\overline{x}_2\overline{x}_1, x_2x_1\}$, $P_3 = \{\overline{x}_2x_1, x_2x_1\}$, and $P_4 = \{\overline{x}_2x_1, x_2\overline{x}_1\}$. Note that $P_1 \cap P_3 = P_2 \cap P_4 = \emptyset$ and that $\overline{P}_1 = \{\overline{x}_2x_1, x_2x_1\} = P_3$ are the 0-paths of $P_1$.*

Because of the relation between 1-paths and 0-paths discussed above, each 1-path of the second edge can be swapped with a 0-path of the first edge. To keep the number of required Toffoli gates as small as possible, we swap a 1-path $p \in P_2$ with its most similar 0-path $p' \in \overline{P}_1$. In fact, if there exist corresponding paths $p$ and $p'$ which are identical ($p = p'$), only a Toffoli gate with target line $x_i$ and a set of control lines that represent $p$ is required.[5] If $p \neq p'$, $p$ has to be adjusted to match $p'$ before the paths can be swapped as described above. To this end, a Toffoli gate with target line $x_j$ is added for each variable $x_j$ that occurs in $p$ and $p'$ in different phases. Note that all these Toffoli gates contain a positive control line $x_i$, since only the paths in the second and fourth edge shall be changed. Swapping all 1-paths of the second edge with the 0-paths of the first edge inherently swaps the 1-paths of the third edge with the 0-paths of the fourth edge and, hence, transforms the currently considered node to the identity structure.

**Example 5** (continued)**.** *Consider again the top node of the 1:1DD depicted in Fig. 1b. As can be seen, there exists a 1-path $p \in P_2$ which is identical to a 0-path $p' \in \overline{P}_1$, namely $p = x_2x_1 = p'$. To swap the 1-path with the 0-path, a gate $TOF(\{x_2, x_1\}, x_3)$ is applied. The remaining 1-path $p = \overline{x}_2\overline{x}_1$ of $P_2$ has to be adjusted to match the 0-path $p' = \overline{x}_2x_1$. This requires a gate $TOF(\{x_3, \overline{x}_2\}, x_1)$. Since afterwards $p = p'$, the paths can be swapped using a gate $TOF(\{\overline{x}_2, x_1\}, x_3)$ – resulting in the 1:1DD shown in Fig 6a where the currently considered node assumes the identity structure.*

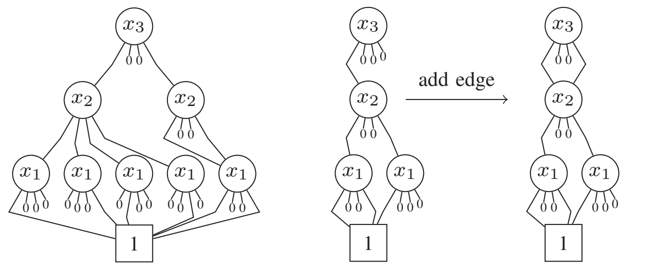
### *B. Case 2: Incomplete Specification*

The synthesis of incompletely specified encoders is similar to the completely specified case described above. However the 1:1DD contains fewer 1-paths (and, hence, contains more 0-paths), i.e. less paths have to be considered in order to accomplish the identity structure. Consequently, the number of 0-paths $p' \in \overline{P}_1$ is larger than the number of 1-paths $p \in P_2$ for most nodes (i.e. $|\overline{P}_1| > |P_2|$) which introduces some degree of freedom. This degree of freedom allows for choosing the subset of $\overline{P}_1$ containing the 0-paths that require the fewest number of Toffoli gates when swapped with the 1-paths of $P_2$. Note that we have to consider the 1-paths in the third edge separately, because swapping all 1-paths of the second edge with 0-paths of the first edge does not necessarily swap all 1-paths of the third edge with 0-paths of the fourth edge. In the case that $|\overline{P}_1| = |P_2|$ for the currently considered node, synthesis is conducted as described for the completely specified case.

Once the second and third edge are transformed to 0-stubs, the currently considered 1:1DD node does not necessarily have the desired identity structure – the first or the fourth edge may be a 0-stub as well. In this case, we insert an edge to accomplish the identity structure (exploiting the degree of freedom that some one-to-one mappings can arbitrarily be defined). In other words, inserting an edge is nothing but adding missing 1-paths in a way that suits best to derive the desired identity structure.

**Example 6.** *Consider the top node of the 1:1DD shown in Fig. 2a (representing the incomplete encoder specification from Table Ib), which contains only three 1-paths. The respective set of paths are $P_1 = \{\overline{x}_2\overline{x}_1, x_2\overline{x}_1\}$, $P_2 = \{\overline{x}_2x_1\}$, and $P_3 = P_4 = \emptyset$. The degree of freedom allows for arbitrarily choosing 0-paths from $\overline{P}_1 = \{\overline{x}_2x_1, x_2x_1\}$ that should be swapped with the 1-paths $p \in P_2$. There is only one path $p = \overline{x}_2x_1 \in P_2$. Furthermore, this path is also contained in $\overline{P}_1$. To swap the two paths, gate $TOF(\{\overline{x}_2, x_1\}, x_3)$ is applied as discussed above. The resulting 1:1DD is shown on the left hand side of Fig. 6b. Since the fourth edge of the top node of the 1:1DD is a 0-stub, we insert an edge such that the top node assumes the identity structure (shown on the right hand side of Fig.6b). Inserting the edge doubles the number of 1-paths of the currently considered node from 3 to 6.*

[5]As discussed in Section III, the set of control lines is additionally be enriched with literals that represent the path to the currently considered node.



(a) Completely specified (b) Incompletely specified

Fig. 6: Synthesis of encoders

### C. Case 3: Hamming Weight-based Specification

Synthesis of encoders described by 1:1DDs based on Hamming weights requires different transformations to generate the desired identity structure. Since only the Hamming weight of the outputs is fixed, we have a certain degree of freedom which can be exploited. However, to get a starting point, we first initialize each output with a fix value: Each output which is supposed to assume a Hamming weight $h$ is initialized with $2^h - 1$ (as already discussed at the end of Section II by means of Fig. 2b). At the same time, we keep the flexibility to change the permutation of the corresponding assignments later in the process (as long as they keep the Hamming weight). Besides that, we utilize a key observation that, for an encoder with $n$ variables, exactly $\binom{n}{h}$ different patterns with Hamming weight $h$ have to be defined. Finally, the Hamming weights can easily be derived from the paths of the 1:1DD, since the first and the second edge describe a mapping to 0, while the remaining two edges represent a mapping to 1. In the following, this is explicitly highlighted by representing edges mapping to 0 (1) with dotted (solid) lines in the 1:1DD.

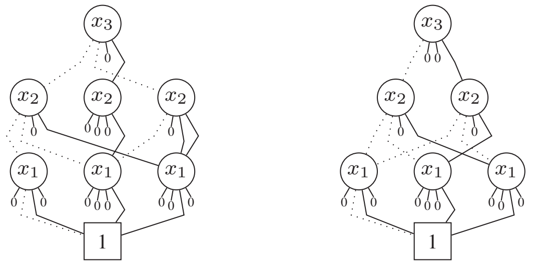
**Example 7.** *Consider again the specification based on Hamming weights as discussed before in Table Id. In order to synthesize an encoder for that, we first create a 1:1DD in which* all *mappings to a Hamming weight $h = 0$ are mapped to $2^0 - 1 = 0$, all mappings to a Hamming weight $h = 1$ are mapped to $2^1 - 1 = 1$, all mappings to a Hamming weight $h = 2$ are mapped to $2^2 - 1 = 3$, and so on. This yields the 1:1DD as shown in Fig. 2b. Now, for each input, the desired Hamming weight can easily be determined by simply following the respective 1-path and counting the number of 1-edges (i.e. solid edges). As an example, the 1:1DD has exactly $\binom{3}{2} = 3$ entries with Hamming weight 2, i.e. three paths from the root to the* 1-terminal *with one dotted and two solid edges.*

As before, the goal is to transform the currently considered node such that an identity-structure results. This also influences the respectively considered Hamming weights. While, overall, exactly $\binom{i}{h}$ different patterns with Hamming weight $h$ exists, this number changes to *exactly* $\binom{i-1}{h}$ in case $x_i$ is mapped from 0 to 0 and to $\binom{i-1}{h-1}$ in case $x_i$ is mapped from 1 to 1. Since we are not interested in the precise output patterns (as long as they assume the desired Hamming weight), this is the only requirement we have to ensure when performing the transformation. Hence, in a first step, we analyze how many 1-paths assume the Hamming weight $h$ when the input of the currently considered node is 0 (represented by the first and the third edge) and how many 1-paths assume the Hamming weight $h$ when the input of the currently considered node is 1 (represented by the second and fourth edge).

**Example 8.** *Consider the top node of the 1:1DD shown in Fig. 2b. From the first and third edge, we can easily obtain how many 1-paths with Hamming weight $h$ exists when setting $x_3 = 0$ (to this end, simply count the number of 1-paths from the currently considered node that include exactly $h$ solid edges). Similarly, we can obtain from the second and fourth edge how many 1-paths with Hamming weight $h$ exists when setting $x_3 = 1$. Table II provides the respective numbers. Now, these numbers are compared to the actually desired values, namely $\binom{2}{h}$ for case $x_3 = 0$ and $\binom{2}{h-1}$ for case $x_3 = 1$. As can be seen, we have one path too much with Hamming weight 3 (for $x_3 = 0$) and one too much with weight 1 (for $x_3 = 1$).*

TABLE II: Distribution of the Hamming weights

| | input $x_3 = 0$ | | | input $x_3 = 1$ | | |
|---|---|---|---|---|---|---|
| Hw. | $1^{st}$ e. | $3^{rd}$ e. | desired | $2^{nd}$ e. | $4^{th}$ e. | desired |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 2 | 2 | 0 | 1 |
| 2 | 1 | 0 | 1 | 2 | 0 | 2 |
| 3 | 0 | 1 | 0 | 0 | 0 | 1 |



(a) Move paths (b) Rotate outputs

Fig. 7: Synthesis specifications based on Hamming weights

In order to consolidate the respective paths so that the desired distribution of Hamming weights is established, we have to swap paths – similar as we did above for the complete and incomplete case. To this end, we determine the set of paths $P_{1-3}$ and $P_{2-4}$ for Hamming weights that occur too often when setting $x_i = 0$ and $x_i = 1$, respectively. The paths in these sets have to be swapped (as discussed above) to obtain the desired distribution of Hamming weights.

**Example 8** (continued). *When considering $x_3 = 0$ (i.e. the first and the third edge), one path with Hamming weight 3 occurs too often. Since there is only one such path (namely $x_2x_1$), $P_{1-3} = \{x_2x_1\}$. Furthermore, when considering $x_3 = 1$, one path with Hamming weight 1 occurs too often. The degree of freedom allows to choose one of the two paths with Hamming weight 1 (namely $\overline{x}_2x_1$ or $x_2x_1$) that has to be swapped. We chose $P_{2-4} = \{x_2x_1\} = P_{1-3}$, which results in a smaller cascade of Toffoli gates to swap the paths in $P_{1-3}$ and $P_{2-4}$. In fact, only gate $TOF(\{x_2, x_1\}, x_3)$ is required. The resulting 1:1DD is shown in Fig. 7a.*

After establishing the desired distribution of 1-paths with Hamming weight $h$, the second edge may still contain 1-paths. However, they can be easily shifted to the fourth edge by permuting their outputs. More precisely, we rotate the outputs of the not yet considered variables (including $x_i$) to the right by one position, i.e. the outputs of $x_i, x_{i-1}, \ldots, x_1$ are changed to the outputs of $x_1, x_i, \ldots, x_2$. Since we chose to represent Hamming weight $h$ with $2^h - 1$ the output of $x_1$, which becomes the new output of $x_i$ after the rotation, is 1. Therefore, the mappings of $x_1$ from 1 to 0 are changed to mappings from 1 to 1 – leading to the desired identity structure. Permuting outputs does not require to apply gates and, since only the Hamming weights of the outputs matter, is a valid modification of the function to be synthesized.

**Example 8** (continued). *Consider again the 1:1DD in Fig. 7a. The 1-paths of the second edge have outputs $x_3x_2x_1 = 001$ (i.e. Hamming weight 1) once and outputs $x_3x_2x_1 = 011$ (i.e. Hamming weight 2) twice. After rotation these 1-paths, they have outputs* 100 *and* 101*, respectively, and are therefore* *moved to the fourth edge. The resulting 1:1DD is shown in Fig. 7b. This yields the desired identity structure for the considered node while the objective is still satisfied.*

## V. Experimental Evaluation

We have implemented the proposed approach in C. As basis for the implementation of the 1:1DDs, the QMDD package provided in [11] has been utilized. By this, a tool resulted which realizes Toffoli circuits for a given encoder specification (without the need to explicitly take care of guaranteeing one-to-one mappings). Afterwards, this circuit has been re-synthesized for a conventional technology using ABC [4], a synthesis tool based on rewriting of And-Inverter Graphs. This allows to determine the number of 2-input gates, required to synthesize the circuit.

To compare the proposed approach to conventional methodologies, several encoding specifications to be realized have been considered which are frequently used in related work – including probability density functions as found in DSP applications with an increasing, decreasing, or Gaussian shape as well as probability functions obtained from analysis of real-life Linux programs. For all these specifications, the desired Hamming weight for 1000 input patterns are provided. In order to realize these specifications with the conventional design methodology (as e.g. applied in [7], [2], [14], [1] and denoted by *Conventional* in the following), the Hamming weights are translated to precise output patterns. Moreover, for all remaining input patterns the necessary one-to-one mapping is specified by enforcing the realization of the identity function for them. Like for the proposed approach, we use ABC [4] to synthesize the specification for conventional technology. As already discussed in Section I, these steps are inevitable in the current design of encoders but significantly reduce the degree of freedom to be exploited during the synthesis. In contrast, the method proposed here allows to fully exploit these freedoms.

This is also confirmed by the obtained numbers which are summarized in Table III. The first columns provide thereby the bit-width of the desired encoder as well as its name. Afterwards, the number of required 2-input gates is provided which is required when the conventional approach is applied and when the method proposed in this work is applied. The last column of the table lists the reduction of 2-input gates of the proposed approach compared to the conventional methodology.

The experimental results clearly show the benefit of the proposed approach. We observe a substantial reduction of the number of gates in the encoder circuits. Taking the one-to-one mapping for granted allows to exploit the full degree of freedom during synthesis and results in smaller circuits. In the case of having an incompletely specified function combined with the Hamming weight objective, this degree of freedom leads to a reduction of up to 92% (approx. 60% on average).

## VI. Conclusions

In this paper, we proposed a novel approach for the synthesis of encoders which takes one-to-one mappings for granted and, by this, overcomes main problems of conventional solutions with respect to scalability, applicability, and the exploitation of the degree of freedom. The proposed approach utilizes description means which is explicitly suited for corresponding specifications and a circuit description which inherently ensures a one-to-one mapping. This allows for exploiting the full degree of freedom (e.g. for incompletely specified encoders or encoders based on Hamming weights) during synthesis. Experimental evaluations show that the proposed approach allows for synthesizing encoder circuits with up to 92% less gates compared to conventional design methodologies. This confirms the promises of the proposed approach for the synthesis of encoders and motivates more detailed evaluations to be conducted in future work.

TABLE III: Experimental Evaluation

| Width | Function | Conventional | Proposed | Reduction [%] |
|---|---|---|---|---|
| 16 | inc. | 21649 | 9869 | 54.41 |
| 16 | dec. | 10595 | 9590 | 9.49 |
| 16 | Gauss | 16716 | 9731 | 41.79 |
| 16 | inv. Gauss | 17712 | 9726 | 45.09 |
| 16 | bison | 31050 | 13588 | 56.24 |
| 16 | espresso | 32257 | 14369 | 55.45 |
| 16 | flex | 31507 | 13178 | 58.17 |
| 16 | gcc | 29019 | 12841 | 55.75 |
| 16 | gnuplot | 30651 | 13848 | 54.82 |
| 16 | gopher | 33074 | 14418 | 56.41 |
| 16 | gzip | 32273 | 14471 | 55.16 |
| 32 | inc. | 48091 | 15689 | 67.38 |
| 32 | dec. | 15984 | 13270 | 16.98 |
| 32 | Gauss | 35022 | 15774 | 54.96 |
| 32 | inv. Gauss | 36147 | 15775 | 56.36 |
| 32 | bison | 159999 | 34615 | 78.37 |
| 32 | espresso | 159834 | 37004 | 76.85 |
| 32 | flex | 200829 | 38323 | 80.92 |
| 32 | gcc | 170687 | 37073 | 78.28 |
| 32 | gnuplot | 146023 | 34249 | 76.55 |
| 32 | gopher | 155483 | 36286 | 76.66 |
| 32 | gzip | 171997 | 38739 | 77.48 |
| 64 | inc. | 47177 | 26232 | 44.40 |
| 64 | dec. | 24593 | 18549 | 24.58 |
| 64 | Gauss | 63029 | 29900 | 52.56 |
| 64 | inv. Gauss | 69732 | 20930 | 69.99 |
| 64 | bison | 928848 | 89194 | 90.40 |
| 64 | espresso | 976491 | 93197 | 90.46 |
| 64 | flex | 1289836 | 95041 | 92.63 |
| 64 | gcc | 1020144 | 87045 | 91.47 |
| 64 | gnuplot | 846739 | 83831 | 90.10 |
| 64 | gopher | 830333 | 90780 | 89.07 |
| 64 | gzip | 902930 | 91655 | 89.85 |

## Acknowledgements

This work has partially been supported by the European Union through the COST Action IC1405.

## References

[1] L. Benini, A. Macii, M. Poncino, and R. Scarsi. Architectures and synthesis algorithms for power-efficient businterfaces. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(9):969–980, 2000.
[2] L. Benini, G. D. Micheli, E. Macii, M. Poncino, and S. Quer. Power optimization of core-based systems by address bus encoding. *Trans. VLSI Syst.*, 6(4):554–562, 1998.
[3] L. Benini, G. D. Micheli, D. Sciuto, E. Macii, and C. Silvano. Address bus encoding techniques for system-level power optimization. In *Design, Automation and Test in Europe*, pages 861–866, 1998.
[4] R. K. Brayton and A. Mishchenko. ABC: an academic industrial-strength verification tool. In *Computer Aided Verification*, pages 24–40, 2010.
[5] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.
[6] R. Drechsler and R. Wille. From truth tables to programming languages: Progress in the design of reversible circuits. In *International Symposium on Multiple-Valued Logic, ISMVL*, pages 78–85, 2011.
[7] A. García-Ortiz, D. Gregorek, and C. Osewold. Optimization of interconnect architectures through coding: A review. In *Electronics, Communications and Photonics Conference*, pages 1–6, April 2011.
[8] K. Lee, S. Lee, and H. Yoo. Low-power network-on-chip for high-performance soc design. *IEEE Trans. VLSI Syst.*, 14(2):148–160, 2006.
[9] D. M. Miller and M. A. Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *Int'l Symp. on Multi-Valued Logic*, page 6, 2006.
[10] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.
[11] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler. QMDDs: Efficient quantum function representation and manipulation. *IEEE Trans. on CAD*, 35(1):86–99, 2016.
[12] A. G. Ortiz, L. S. Indrusiak, T. Murgan, and M. Glesner. Low-power coding for networks-on-chip with virtual channels. *J. Low Power Electronics*, 5(1):77–84, 2009.
[13] P. R. Panda and N. D. Dutt. Reducing address bus transitions for low power memory mapping. In *European Design and Test Conference, ED&TC*, pages 63–71, 1996.
[14] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj. A coding framework for low-power address and data busses. *IEEE Trans. VLSI Syst.*, 7(2):212–221, 1999.
[15] M. Saeedi and I. L. Markov. Synthesis and optimization of reversible circuits - a survey. *ACM Comput. Surv.*, 45(2):21, 2013.
[16] R. Wille, R. Drechsler, C. Osewold, and A. Garcia-Ortiz. Automatic design of low-power encoders using reversible circuit synthesis. In *Design, Automation and Test in Europe*, pages 1036–1041, 2012.
[17] R. Wille, O. Keszocze, S. Hillmich, M. Walter, and A. G. Ortiz. Synthesis of approximate coders for on-chip interconnects using reversible logic. In *Design, Automation and Test in Europe*, 2016.