OR-Inverter Graphs for the Synthesis of Optical Circuits

Arighna Deb†, Robert Wille‡, Rolf Drechsler†

†Computer Science & Engineering, Jadavpur University, India
‡Institute for Integrated Circuits, Johannes Kepler University, Linz, Austria
††Institute of Computer Science, University of Bremen, Bremen, Germany
arighna.deb@gmail.com, robert.wille@jku.at, drechsle@informatik.uni-bremen.de

Abstract—The advances in silicon photonics motivate the consideration of optical circuits as a promising circuit technology. Recently, synthesis for this kind of circuits received significant attention. However, neither the corresponding function descriptions nor the resulting synthesis approaches explicitly considered how optical circuits actually conduct computations – eventually leading to circuits of improvable quality. In this work, we present a synthesis flow which has explicitly been developed for this technology. To this end, we introduce and exploit OR-Inverter graphs (OIGs) – a data-structure which is particularly suited for the design of optical circuits. Experimental results confirm the efficacy of the OIG structure and the resulting synthesis approach. Compared to several alternative solutions – relying on conventional function representations – the number of gates can be reduced by half or even significantly more than that.

I. INTRODUCTION

The post-CMOS computing technology is gaining importance as the scaling of transistors is approaching its physical limits. Optical circuits emerge as an alternative to current circuit technologies thanks to the advances in silicon photonics [1]. In electronic digital systems, optical technology is already in use as ultra-fast interconnects [2], [3]. This requires back and forth conversion from the optical to the electrical domain at every interconnect interface – obviously a significant drawback. The conversion, however, can easily be avoided if the underlying systems are realized by optical technologies only. This motivates research in the area of designing full-scale optical circuits.

The design automation is one of the key driving forces behind the reduction of the circuit design time and the optimization of the circuit quality. Thus far, Electronic Design Automation (EDA) has played a crucial role in the development of complex conventional circuits [4]. The major design step usually starts with a logic level abstraction and, afterwards, moves down to the physical realization, where the desired circuit is refined with respect to the respective technological constraints. Although physical constraints in optical technology are not entirely solved yet, initial models and corresponding gate libraries already exist for the purpose of logic synthesis and optimization. At this stage, considering logic synthesis and optimization is motivated by the fact that EDA for conventional systems started with logic design automation before physical design automation was actually developed [5].

The main goal of logic synthesis is to determine an efficient realization of a Boolean function. Thus far, Boolean functions are represented by different data structures such as the two-level descriptions Sum-of-Products (SoPs) and Exclusive-Sum-of-Products (ESoPs) [5], [6], Binary Decision Diagrams (BDDs) [7], or AND-Inverter graphs (AIGs) [8]. These representations employ either AND-OR, AND-EXOR, multiplexer (MUX), or Boolean conjunction (AND) as logic primitives. However, the efficacy of any representation heavily depends on the targeted circuit type i.e. on the richness of the applied gate library and on the capability of each library element (i.e. logic gate) to implement the desired Boolean function.

Initially, optical logic synthesis was limited to dedicated functionality such as adders [9], [10], multiplexers [11], dividers [12], etc. Recently, also the synthesis of arbitrary Boolean functions has been considered – leading to approaches, e.g. based on BDDs [13]–[15] but also solutions based on SoPs, ESoPs, or AIGs have been observed [13], [16]. Although, this initiates the automatic synthesis of optical logic circuits for large Boolean functions, often, the synthesized circuits are expensive with respect to the number of required gates. This is mainly because of the fact that, during synthesis, the respective functions representations are simply mapped to corresponding optical gates without explicitly exploiting the characteristics of optical circuits.

In this paper, we propose a novel methodology to synthesize optical circuits. To this end, we introduce a data-structure called OR-Inverter Graphs (OIGs) – a logic representation which is similar to AIGs but employ OR nodes rather than AND nodes in addition to the regular/complement edges. We motivate the utilization of OIGs by considerations that clearly show the suitability of the corresponding OR and NOT operations for the common optical library. In fact, mapping an OIG into an optical circuit yields significantly smaller realizations compared to existing function representations.

Experimental evaluations confirm these findings. The proposed OIG-based synthesis is capable of realizing circuits which improve alternative solutions, e.g., based on SoPs, ESoPs, BDDs, and AIGs by 98%, 89%, 54%, and 56%, respectively (with respect to the number of required gates).

In the following, the proposed approach is introduced as follows. Section II reviews the basics on optical circuits and the commonly applied gate library. Afterwards, Section III discusses the applicability of function representations such as the above-mentioned SoPs, ESoPs, BDDs, and AIGs with respect to the considered optical domain. This provides the motivation of the approach which, afterwards, is described in detail in Section IV. Finally, Section V and Section VI summarize the obtained experimental results and conclude the paper, respectively.

II. OPTICAL CIRCUITS

To keep the paper self-contained, this section briefly reviews the common logic model and gate library used in the domain of optical logic synthesis.

Optical circuits are usually realized by means of a Mach-Zehnder Interferometer (MZI) switch which is based on Semiconductor Optical Amplifiers (SOAs). In the logic domain, the resulting structure is abstracted to a so-called MZI gate. Each MZI gate has two input ports and two output ports. The inputs can either be sourced by light (representing binary 1) or darkness (representing binary 0). Logically, an MZI gate is defined as follows [17], [18]:

Definition 1: An MZI gate realizes a Boolean function $B^2 \rightarrow B^2$ composed of two optical inputs $p$ and $q$ as well as two optical outputs $f$ and $g$. In the presence of both input...
signals, the outputs $f$ and $g$ produce 1 and 0, respectively. The presence of input signal $p$ and the absence of input signal $q$ leads to the logic value 0 and 1 at the outputs $f$ and $g$, respectively. Therefore, the functions

$$f = p \land q \quad \text{and} \quad g = p \land q'$$

are realized. Fig 1(a) provides the graphical representation of an MZI gate.

In addition, splitters and combiners are used as optical logic elements in order to realize logic functions.

**Definition 2:** A splitter divides an optical signal into two signals – each with only half of the incoming signal power. In contrast, a combiner merges two optical signals into a single one and, by this, inherently realizes the OR-function. A splitter (combiner) may have more than two outputs (inputs). Then, in case of a splitter, the strength of the signal is divided by the number of outputs. Fig. 1(b) and Fig. 1(c) provide the graphical representation of both elements.

Together these logic elements form a *gate library* that allows to realize any Boolean function.

The size of an optical circuit is determined in terms of number of MZI gates. This is motivated by the fact that each MZI gate needs to be physically realized. Sometimes, also the number of splitters and the number of combiners are considered. However, as they are significantly easier to realize than the MZI gates, they are often considered negligible. Besides that, the number of splitters has an effect to the final strength of the applied optical signals.

**Example 1:** Fig. 2 shows an optical circuit composed of two MZI gates, two splitters, and one combiner.

III. MOTIVATION

Independent from the respective technology, synthesis is the task of generating a circuit structure which realizes a given (Boolean) function to be synthesized. Corresponding algorithms approach this task from different angles i.e. they

- rely on different function representations which are used as input including Boolean Algebra and two-level representations such as *Sum of Products (SoPs)* and *Exclusive Sum of Products (ESoPs)* as well as *Binary Decision Diagrams (BDDs)*, [7] or *AND-Inverter Graphs (AIGs)*, [8] and
- realize circuits using different (universal) gate libraries allowing for the realization of all possible functions such as *{AND, OR, NOT}* (MUX), or *{NAND}*

Quite often, an obvious relation between the respective function representation and the used gate library exists. For example, Boolean algebra, SoPs, and ESoPs can directly be realized by circuits composed of *{AND, OR, NOT}* and *{AND, XOR, NOT}* respectively. The nodes of a BDD directly correspond to MUX gates, while the nodes of an AIG directly corresponds to NAND gates. Fig. 3 sketches the corresponding mappings.

Moreover, the respective relations are often exploited when technological constraints and/or physical realizations are to be considered. For example, it is known that, in current transistor technologies, a NAND gate is considered significantly cheaper than e.g. *{AND, OR, NOT}*-gates. Hence, when e.g. area or power are of significant importance, a synthesis based on AIGs (which can directly be mapped to a NAND-circuit) might be the preferred design scheme.

For the domain of optical circuits, respective considerations have not been made yet. In fact, almost all existing approaches for the synthesis of optical circuits focused on investigating whether and how established (conventional) function representations can be utilized in order to create circuits composed of MZI, splitter, and combiner gates. For example, synthesis based on Boolean Algebra, SoP, or ESoP as introduced in [13], [16] relies on so-called virtual gates i.e. sub-circuits realizing the respective AND, OR, XOR, NOT operations. Synthesis using BDDs as introduced in [13]–[15] relies on MZI sub-circuits realizing different BDD node configurations. A synthesis approach based on AIGs has, to the best of our knowledge, not been proposed yet, but would rely on sub-circuits realizing NAND operations.

Fig. 4 sketches the corresponding mappings. As can clearly be seen, for all functions representations considered thus far, no direct mapping to elementary optical gates exists — in fact, several gates are required to realize just a single building block. This obviously leads to optical circuits which are rather expensive.

Motivated by these observations and discussions, this work aims for developing an alternative function representation, and a corresponding synthesis approach, which is explicitly dedicated to the gate library reviewed in Section II. Considering the available elementary building blocks, we can see that the splitter and the combiner are the cheapest gates in this library. While the splitter only realizes a fanout and, hence, not an “actual” Boolean function, the combiner serves as a realization of an OR gate. Since the OR operation itself is not universal,
we additionally consider MZI gates in order to realize the
NOT (Inverter) operation. OR and NOT constitute a universal
elementary building blocks (the very cheap combiner and
the single MZI gate as shown in Fig. 5). In contrast to
the previously proposed synthesis approaches reviewed above,
this may provide the basis for a significantly more efficient
synthesis scheme. Hence, in the remainder of this work, we
consider the question how to develop a synthesis scheme which
relies on OR and NOT only.

IV. PROPOSED APPROACH

In order to develop a synthesis scheme for optical circuits
which relies on OR and NOT operations only, we introduce
the concept of an OR-Inverter Graph (OIG). OIGs are inspired
by the AIGs from conventional circuit design. In this section,
we first review and illustrate the background on AIGs. Based
on that, OIGs are introduced and it is shown how OIGs can
be derived from AIGs. These considerations eventually lead to
a synthesis scheme for optical circuits which is described in
detail at the end of this section.

A. AND-Inverter Graph

An AND-Inverter Graph (AIG) is a directed acyclic graph
\( G = (V, E) \) which is composed of three types of nodes.
The first type has no incoming edges and represents a unique
terminal node which serves as primary output. The second
type has no incoming edges and represents primary inputs. The
third type has two incoming edges and one outgoing edge and
represents a Boolean AND operation. These AND nodes also
have two kinds of outgoing edges: a regular edge representing
the actual functionality and a complement edge representing
the negation of this functionality. More formally, an AIG is
defined as follows:

Definition 3: An AND-Inverter graph (AIG) over the primary
input variables \( X = \{x_1, x_2, \ldots, x_n\} \) and with the
primary output variables \( Y = \{y_1, y_2, \ldots, y_m\} \) is a directed
acyclic graph \( G = (V(= \{V_X \cup V_y \cup V_Y\}, E)) \) with
the following properties:

- Each primary input (PI) node \( v \in V_X \) is labeled by
  \( x_i \in X \) and has no incoming edges.
- Each primary output (PO) node \( v \in V_Y \) is a terminal
  labeled by \( y_i \in Y \) and has no outgoing edges.
- Each non-terminal node \( v \in V_y \) represents a Boolean
  conjunction (AND) of the functions represented by the
two incoming edges.
- An edge \( e \in E \) connecting a source node \( u \in V \) to a
target node \( v \in V \) is either a regular or a complement
  edge i.e. \( e = \{(u, (v \times p))|u, v \in V, u \notin V_Y, v \notin V_X\} \)
  with \( p \) denoting whether the edge is a regular edge
  \((p = 1)\) or a complement edge \((p = 0)\).

The size of an AIG is measured in terms of the total number
of AND nodes.

Example 2: Consider the function \( f = (x_0' \land x_1') \lor (x_1 \land
x'_2) \lor (x_0 \land x'_3) \). Using DeMorgan’s theorem, the function can
be written as \( f = ((x_0' \land x_1')' \land (x_1 \land x'_2)' \land (x_0 \land x'_3)'')' \).
The corresponding AIG is shown in Fig. 6, in which, an edge with
a solid dot denotes a complement edge.

B. OR-Inverter Graph

An OR-Inverter Graph (OIG) is a directed acyclic graph
\( H = (V, E) \) which is structurally identical to an AIG. However,
the OR operation, an OR operation is employed in the non-
terminal nodes of an OIG. More formally, an OIG is defined
as follows:

Definition 4: An OR-Inverter graph (OIG) over the primary
input variables \( X = \{x_1, x_2, \ldots, x_n\} \) and with the
primary output variables \( Y = \{y_1, y_2, \ldots, y_m\} \) is a directed
acyclic graph \( H = (V, E) \) with

- a finite set of nodes \( V = V_X \cup V_h \cup V_Y \),
  where \( V_x = \{v_{x_1}, v_{x_2}, \ldots, v_{x_n}\} \) are primary input nodes,
  \( V_h = \{v_{h_1}, v_{h_2}, \ldots, v_{h_m}\} \) are non-terminal nodes
  representing the logical OR operations in the graph,
  and \( V_y = \{v_{y_1}, v_{y_2}, \ldots, v_{y_m}\} \) are terminal nodes
  representing primary outputs.
- an edge \( e \in E \) between a source node \( u \in V \) and a
target node \( v \in V \) is either a regular or a complement
  edge i.e. \( e = \{(u, (v \times p))|u, v \in V, u \notin V_Y, v \notin V_X\} \)
  where \( p \) denotes whether the edge is a regular edge
  \((p = 1)\) or a complement edge \((p = 0)\).

The size of an OIG is measured in terms of the total number
of OR nodes.
Example 3: Re-consider the function \( f \) used in Example 2. The function is expressed in terms of OR operators as \((x_0 \lor x_3) \lor (x_0 \lor x_1) \lor (x_1 \lor x_2)\). This can be represented as an OIG as shown in Fig. 7.

C. Transformation of AIG into OIG

Here, we show that an AIG representing an arbitrary Boolean function can easily be translated into a functionally equivalent OIG. This builds the basis of our proposed synthesis flow.

Given an AIG, \( G = (V, E) \), an OIG, \( H = (V, E) \) is obtained by substituting each 2-input AND node with a functionally equivalent 2-input OR node applying DeMorgan’s theorem. To this end, a total of eight substitutions (forming a substitution set \( S \)) have to be considered:

\[
S = \begin{cases} 
(f_1' \land f_2') & = (f_1 \lor f_2) \\
(f_1' \land f_2) & = (f_1 \lor f_2') \\
(f_1 \land f_2') & = (f_1' \lor f_2) \\
(f_1 \land f_2) & = (f_1' \lor f_2') \\
(f_1 \lor f_2') & = (f_1' \land f_2) \\
(f_1 \lor f_2) & = (f_1' \land f_2') \\
(f_1') & = (f_1) \\
(f_2') & = (f_2) 
\end{cases}
\]

**Theorem 1:** For any AIG, the substitution set \( S \) is complete.

**Proof:** By definition, any AND node in an AIG has two incoming edges i.e. takes 2-inputs \( f_i \) and \( f_j \). This means, any single node can realize the logical conjunction (AND) of one input combination out of 4 combinations \( \{f_1', f_1, f_2', f_2, f_1, f_1', f_2, f_2'\} \). In other words, all AND nodes realize at most 4 logical conjunctions. Further, the complement edge out of any AND node realizes the inversion of the respective logical conjunction. Therefore, in total, there are 8 possible node combinations in any AIG. Hence, the proof.

**Lemma 1:** Let \( f \) be a Boolean function defined over \( \{\lor, \land\} \). This function can always be converted into an expression composed of \( \{\lor, \land\} \) by applying \( S \) in a recursive manner.

**Proof:** Without the loss of generality, assume that the Boolean function \( f \) is expressed as follows:

\[ f = (f_1 \land f_2) \]

If \( f_i \) and \( f_j \) are two primary inputs e.g. \( x_i \) and \( x_j \), then we can substitute them by applying \( S \) as follows:

\[ f = (x_i \land x_j) \]

\[ = (x_i' \lor x_j') \]

D. Proposed Synthesis Flow

Based on the discussions above, a synthesis flow for the efficient realization of optical circuits can be formulated, which is composed of three major steps: (1) the generation of an AIG, (2) the transformation of the AIG into an OIG, and (3) the mapping of the OIG into an optical circuit. In order to generate the AIG, existing methods as e.g. employed in tools such as ABC [19] can be applied. How to transform the AIG to an OIG has been covered in the previous sub-section. Therefore, the mapping to an optical circuit remains left to be described.

To this end, we consider the functional behavior of all relevant node configurations which may occur in an OIG and for which a corresponding sub-circuits is required. This includes all OIG nodes representing an OR operation or a PI and have
Fig. 9: Substitution of OIG nodes to optical circuits

- one or more outgoing regular edges,
- one or more outgoing complement edges, or
- both, one or more regular and complement edges.

Fig. 9 shows the corresponding circuits realizing all the cases. In general, every non-terminal node is mapped to a combiner and a complement edge is realized using an MZI gate for the NOT operation. Whenever a node has multiple outgoing edges i.e. multiple successors, a splitter is added to the respective building blocks.

Eventually, this leads to a synthesis flow as follows:

1) Generate an OIG \( H = (V, E) \) representing a function \( f \) to be synthesized.
2) Traverse \( H \) in a depth-first manner.
3) For each node, apply the corresponding sub-circuit as shown in Fig. 9.
4) Connect the inputs of the sub-circuit accordingly to the corresponding outputs of the sub-circuit representing the previously traversed nodes of \( H \).

Example 5: Consider the OIG representing the function \( f \) as shown in Fig. 7. The mapping scheme traverses the OIG in a depth-first manner. Applying the substitutions shown in Fig. 9 to each node of the OIG, the optical circuit depicted in Fig. 10 results.

V. EXPERIMENTAL EVALUATION

In this section, we summarize the results obtained by the proposed method. To this end, the synthesis flow described in Section IV has been implemented in C++. First, an AIG of the function to be synthesized is created using the tool ABC [19]. Then, we convert this data-structure to an OIG and apply the mapping method as described above. In order to compare the obtained results, we additionally synthesized circuits using the BDD-based approach proposed in [14].

We would like to thank the authors of [14] for making us their tool available.

Table I shows the obtained results. The first column provides the details of the considered benchmark, i.e. their names as well as number of primary inputs (PI) and primary outputs (PO). The next three columns report the number of MZI gates (MZI), the number of combiners (Combiner), as well as the number of splitters (Splitter) for the resulting circuits obtained by the SoP-, ESoP-, BDD-, AIG-, and OIG-based synthesis approaches. Note that the AIG-based approach always yields circuits with a total of zero combiners which is why we omit a dedicated column in this case. The fourth column (\( \Sigma \)) reports the sum of all elements, i.e. MZI gates, combiners and splitters for the respective approaches. In the final column, we have reported the percentage reduction in number of MZI gates compared to existing function representations.

The results confirm that, for the synthesis of optical circuits, OIGs are indeed more suitable than alternative function representations and methods. In fact, OIG-based synthesis clearly outperforms all other synthesis approaches with respect to the number of gates – sometimes circuits with orders of magnitudes less gate result. On average, improvements of 98%, 89%, 54%, 56% with respect to the MZI gate count can be achieved compared to the SoP-, ESoP-, BDD-, and AIG-based approaches, respectively.

As mentioned in Section II, the number of MZI gates is most important metric as MZIs contribute most to the chip size, while combiners/splitters are negligible. However to evaluate their impact on overall circuit size, the numbers of MZI gates, combiners and splitters are summed up in columns denoted by \( \Sigma \). The results demonstrate that, even under this consideration, the average circuit size obtained from OIGs is still 96%, 80%, and 26% smaller compared to SoP-, ESoP-, and BDD-based approaches. Only with respect to the AIG-based approach, roughly the same circuit size results. But note that this is basically only because of the fact that AIGs need significantly fewer combiners. However, since combiners are easier to realize than MZI gates (from which AIGs still require significantly more), OIG-based synthesis clearly positions itself as a more promising design solution for the realization of efficient and compact optical circuits.

Only in one aspect (namely the number of splitters) one synthesis approach (namely SoP-based synthesis) performs better than the proposed OIG-based solution (except for a few cases). This certainly helps to avoid splitting optical signals and keeping the signal strength. But signals applied to circuits obtained by SoP-based synthesis have to pass through a tremendous amount of MZI gates – which also harms their
strength. In contrast, signals have to pass significantly less gates in total when the circuit is derived from OIG-based synthesis. Additionally considering issues such as circuit size (which also is mainly affected by MZI gates), OIG-based synthesis clearly positions itself as a promising solution design for the realization of efficient and compact optical circuits.

VI. CONCLUSION

In this work, we presented a synthesis approach for optical circuits based on OR-Inverter graphs. In contrast to conventional function descriptions such as SoPs, ESoPs, BDDs, or AIGs, this explicitly considers how optical circuits actually conduct computations: with combiners realizing OR operations and (larger) MZI gates whose number can be kept moderate by using them for NOT operations only. Experimental results validated the potential of OIGs in optical logic synthesis and yielded circuits which are reduced by half or even significantly more than that with respect to gate count compared to previously proposed methods.

ACKNOWLEDGEMENT

This research was supported by the European Commission in the framework of the Erasmus Mundus cLink project.

REFERENCES


