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
†‡Cyber-Physical Systems, DFKI GmbH, 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 $\mathbb{B}^2 \rightarrow \mathbb{B}^2$ composed of two optical inputs p and q as well as two optical outputs f and g. In the presence of both input

(a) MZI gate (b) Splitter (c) Combiner

Fig. 1: Optical gates

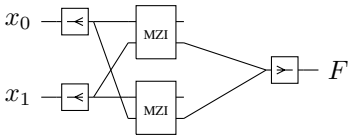


Fig. 2: Optical circuit

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

$$f = p \wedge q \qquad \text{and} \qquad g = p \wedge q'$$

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.

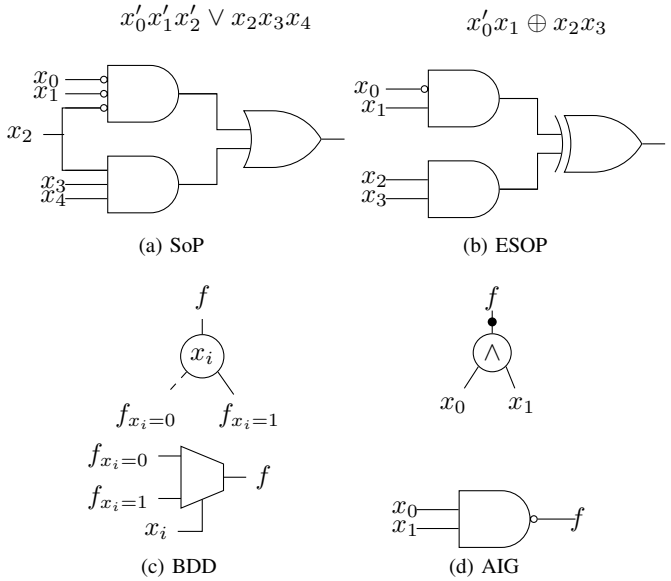
Example 1: Fig. 2 shows an optical circuit composed of two MZI gates, two splitters, and one combiner.

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}, {AND, XOR, NOT}, {MUX}, or {NAND}.

Quite often, an obvious relation between the respective function representation and the used gate library exists. For



(a) SoP (b) ESOP

(c) BDD (d) AIG

Fig. 3: Mapping function representations to conv. circuits

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,

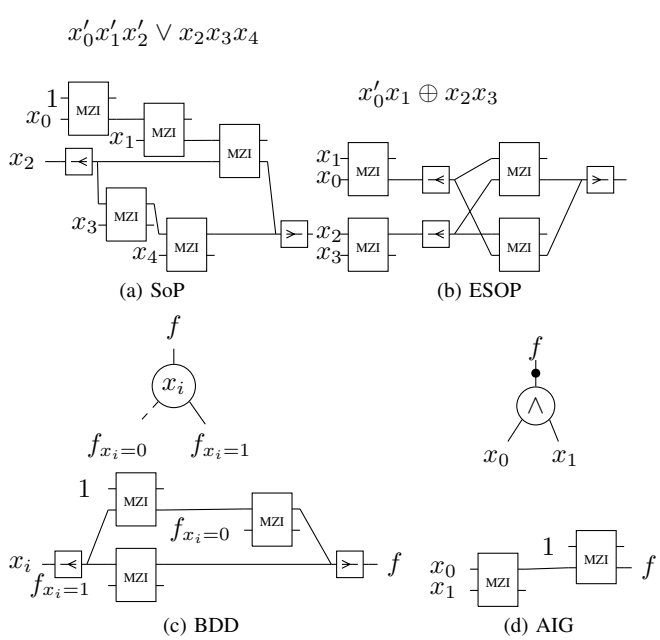


Fig. 4: Mapping function representations to optical circ.

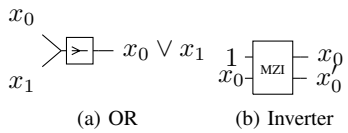


Fig. 5: Proposed library

we additionally consider MZI gates in order to realize the NOT (Inverter) operation. OR and NOT constitute a universal gate library which allows for realizing all Boolean functions. At the same time, both operations are already available as 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 outgoing 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:

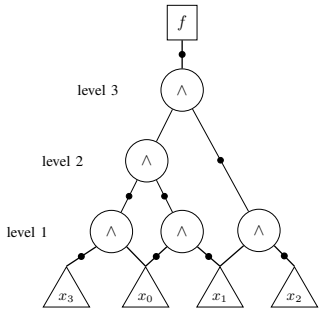


Fig. 6: AND-Inverter graph for function f

Definition 3: An *AND-Inverter graph* (AIG) over the primary input variables $X = \{x_1, x_2, \cdots, x_n\}$ and with the primary output variables $Y = \{y_1, y_2, \cdots, y_m\}$ is a directed acyclic graph $G = (V(= \{V_X \cup V_g \cup V_Y\}), E)$ 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 $y_j \in Y$ and has no outgoing edges.
- Each non-terminal node $v \in V_g$ 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 $f = (x'_0 \wedge x'_1) \vee (x_1 \wedge x'_2) \vee (x_0 \wedge x'_3)$. Using DeMorgan's theorem, the function can be written as $f = ((x'_0 \wedge x'_1)' \wedge (x_1 \wedge x'_2)' \wedge (x_0 \wedge x'_3)')'$. 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, both differ with respect to the logic primitive. Instead of the AND 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 $X = \{x_1, x_2, \cdots, x_n\}$ and with the primary output variables $Y = \{y_1, y_2, \cdots, y_m\}$ is a directed acyclic graph $H = (V, E)$ with

- a finite set of nodes $V = V_X \cup V_h \cup V_Y$, where $V_x = \{v_{x_1}, v_{x_2}, \cdots, v_{x_n}\}$ are primary input nodes, $V_h = \{v_{h_1}, v_{h_2}, \cdots, v_{h_k}\}$ are non-terminal nodes representing the logical OR operations in the graph, and $V_y = \{v_{y_1}, v_{y_2}, \cdots, v_{y_m}\}$ 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.

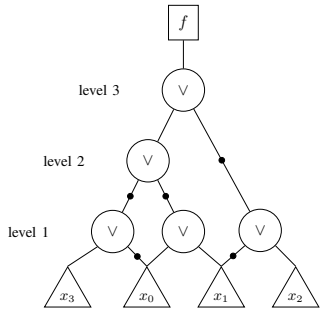


Fig. 7: OR-Inverter graph for function f

Example 3: Re-consider the function f used in Example 2. The function is expressed in terms of OR operators as $((x_0' \vee x_3)' \vee (x_0 \vee x_1)' \vee (x_1' \vee x_2)')$. 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:

$$S \begin{cases} (f_i' \wedge f_j') \equiv (f_i \vee f_j)' \\ (f_i' \wedge f_j')' \equiv (f_i \vee f_j) \\ (f_i' \wedge f_j) \equiv (f_i \vee f_j')' \\ (f_i' \wedge f_j)' \equiv (f_i \vee f_j') \\ (f_i \wedge f_j') \equiv (f_i' \vee f_j)' \\ (f_i \wedge f_j')' \equiv (f_i' \vee f_j) \\ (f_i \wedge f_j) \equiv (f_i' \vee f_j')' \\ (f_i \wedge f_j)' \equiv (f_i' \vee f_j') \end{cases}$$

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 $\{f_i' f_j', f_i' f_j, f_i f_j', f_i f_j\}$. 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. ■

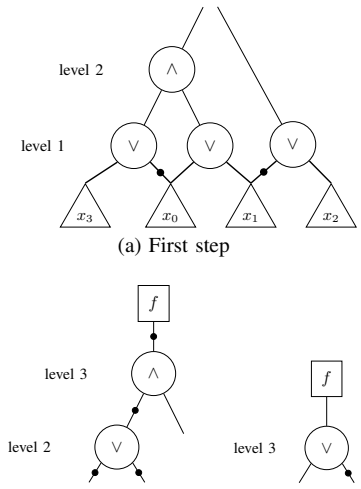
Lemma 1: Let f be a Boolean function defined over $\{\wedge, '\}$. This function can always be converted into an expression composed of $\{\vee, '\}$ by applying S in a recursive manner.

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

$$f = (f_i \wedge f_j)$$

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:

$$\begin{aligned} f &= (x_i \wedge x_j) \\ &= (x_i' \vee x_j')' \end{aligned}$$



(a) First step

(b) Second step (c) Third step

Fig. 8: Transformation of AIG into OIG

If at least one input e.g. f_i, represents a logical conjunction i.e. $f_i = (f_{i_p} \wedge f_{i_q})$, then we can substitute them by applying S as follows:

$$\begin{aligned} f &= (f_i \wedge f_j) \\ &= (f_i' \vee f_j')' \text{ [applying substitution } S\text{]} \\ &= ((f_{i_p} \wedge f_{i_q})' \vee f_j')' \\ &= ((f_{i_p}' \vee f_{i_q}') \vee f_j')' \text{ [applying substitution } S\text{]} \end{aligned}$$

This shows that applying rules from S in a recursive manner, an expression composed of $\{\wedge, '\}$ transforms into an expression composed of $\{\vee, '\}$. Hence, the proof. ■

Theorem 2: Any AIG, $G = (V, E)$ can be transformed into an OIG, $H = (V, E)$ using the substitution set S.

Proof: The proof follows from Lemma 1. ■

Example 4: Consider the AIG shown in Fig. 6 to be translated into a functionally equivalent OIG. The translation begins with traversing the AIG in a breadth-first manner and applying the substitution rules S. For the first level, this results in OR nodes as shown in the same level in Fig. 8(a). In the next step, the substitution rules S are further applied on the AND nodes at level 2 – leading to the graph as depicted in Fig. 8(b). In a similar fashion, the AND node at level 3 is handled – leading to the structure shown in Fig. 8(c). As a result of these transformations, an OIG is generated which is depicted in Fig. 7.

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

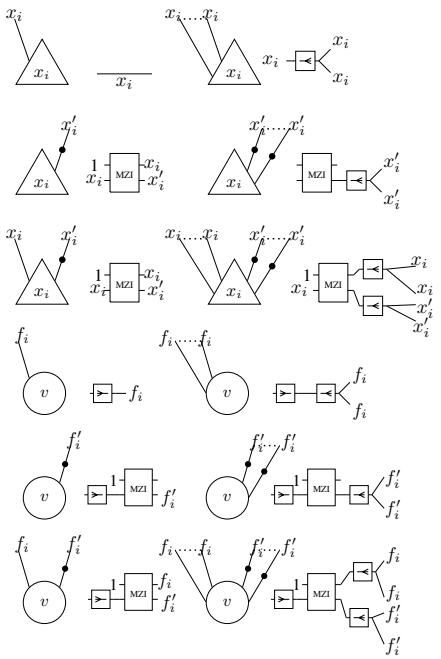


Fig. 9: Substitution of OIG nodes to optical circuits

- 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][1] as

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

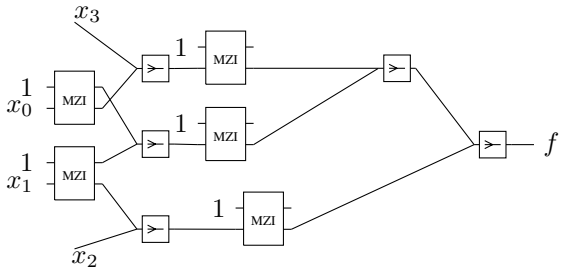


Fig. 10: Resulting optical circuit for function f

well as SoP-, ESoP-, and AIG-based approaches following the concepts discussed in Section III. As benchmarks, functions from the LGSynth library have been applied. All experiments have been carried out on a Linux machine with a 2.8 GHz Intel Core i7 processor and 8 GB memory. All circuits have been obtained in negligible run-time (i.e. not more than one CPU minute) which is why we omit a detailed run-time discussion in the following.

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 ($\sum$) 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 $\sum$. 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

TABLE I: Experimental Evaluation

Function		MZI					Combiner				Splitter					$\sum$					% Reduc. in MZI w.r.t.			
name	PI/PO	SoP	ESoP	BDD	AIG	OIG	SoP	ESoP	BDD	OIG	SoP	ESoP	BDD	AIG	OIG	SoP	ESoP	BDD	AIG	OIG	SoP	ESoP	BDD	AIG
apex5	117/88	5879	8171	3410	1443	723	80	1142	1740	826	112	2396	243	220	220	6071	11709	5393	1663	1769	87.7%	91.2%	78.8%	49.9%
cps	24/108	6502	7610	3122	2101	950	66	552	1638	1243	23	1127	171	319	319	6591	9289	4931	2420	2512	85.4%	87.5%	69.6%	54.8%
ex4	128/28	3784	4996	1192	768	378	14	606	638	403	84	1296	210	16	16	3882	6898	2040	784	797	90.0%	92.4%	68.3%	50.8%
soar	83/94	2881	3792	1928	860	405	66	435	1031	508	78	948	219	696	696	3025	5175	3178	1556	1609	85.9%	89.3%	79.0%	52.9%
apex1	45/45	1533	3653	3008	3565	1563	37	1060	1562	2002	214	2334	340	392	392	1784	7047	4910	3957	3957	-2.0%	57.2%	48.0%	56.2%
mish	94/43	73	248	262	148	117	26	57	144	87	27	141	38	14	14	126	446	444	162	218	-60.3%	52.8%	55.3%	20.9%
seq	41/35	16364	19213	3642	3214	1467	33	1424	1905	1777	41	2889	380	367	367	16438	23526	5927	3581	3611	91.0%	92.4%	59.7%	54.4%
ti	47/72	1955	3771	1980	1520	704	58	908	1054	866	235	2051	209	255	255	2248	6730	3243	1775	1825	64.0%	81.3%	64.4%	53.7%
x2dn	82/56	346	499	512	333	158	29	73	283	173	74	220	89	54	54	449	792	884	387	385	54.3%	68.3%	69.1%	52.6%
x7dn	66/15	4398	5612	1354	762	359	15	607	699	390	66	1280	238	82	82	4479	7499	2291	844	831	91.8%	93.6%	73.5%	52.9%
xparc	41/73	10605	24254	4750	5352	2259	67	6824	2451	3142	580	14228	354	686	686	11252	45306	7555	6038	6087	78.7%	90.7%	52.4%	57.8%
apex2	39/3	13418	35725	1108	536	246	3	1746	493	283	78	3622	260	90	90	13499	41093	1861	626	619	98.2%	99.3%	77.8%	54.1%
cordic	23/2	17163	12939	103	114	47	2	1544	28	60	23	3881	43	20	20	17188	18364	174	134	127	99.7%	99.6%	54.4%	58.8%
pdc	16/40	30613	4196	1099	1431	605	40	744	575	859	2403	1704	245	229	229	33056	6644	1919	1660	1693	98.0%	85.6%	44.9%	57.7%
spla	16/46	32651	4362	1050	1378	524	44	768	568	896	2289	1760	198	253	253	34984	6890	1816	1631	1673	98.4%	88.0%	50.1%	62.0%
0410184	14/14	212979	1354	184	195	93	14	257	57	103	16383	589	63	43	43	229376	2200	304	238	239	100.0%	93.1%	49.5%	52.3%
4mod5	4/1	13	11	8	17	11	1	3	2	7	4	10	5	0	0	18	24	15	17	18	15.4%	0.0%	-37.5%	35.3%
4mod7	4/3	39	31	29	39	19	3	10	8	19	10	27	17	9	9	52	68	54	48	47	51.3%	38.7%	34.5%	51.3%
5xp1	7/10	221	185	68	188	89	9	50	22	100	7	126	35	29	29	237	361	125	217	218	59.7%	51.9%	-30.9%	52.7%
add6	12/7	1841	936	140	96	50	7	201	47	43	12	460	52	15	15	1860	1597	239	111	108	97.3%	94.7%	64.3%	47.9%
adr4	8/5	265	175	38	60	32	5	46	13	27	8	112	27	9	9	278	333	78	69	68	87.9%	81.7%	15.8%	46.7%
alu	5/1	65	15	12	20	11	1	3	4	9	5	9	9	2	2	71	27	25	22	22	83.1%	26.7%	8.3%	45.0%
alu1	12/8	22	55	34	59	28	7	11	12	30	12	35	12	11	11	41	101	58	70	69	-27.3%	49.1%	17.6%	52.5%
alu2	10/6	1745	507	451	765	338	5	75	131	428	10	170	194	78	78	1760	752	776	843	844	80.6%	33.3%	25.1%	55.8%
alu3	10/8	213	320	197	129	60	7	60	66	69	12	144	68	24	24	232	524	331	153	153	71.8%	81.3%	69.5%	53.5%
alu4	14/8	6847	4484	1823	2049	921	8	508	639	1122	14	1094	251	215	215	6869	6086	2713	2264	2258	86.5%	79.5%	49.5%	55.1%
bw	5/28	176	583	353	256	101	26	266	90	150	40	559	111	45	45	242	1408	554	301	296	42.6%	82.7%	71.4%	60.5%
decod24-enable	3/4	8	14	8	10	7	0	5	4	8	3	16	5	5	5	11	35	17	15	20	12.5%	50.0%	12.5%	30.0%
ex5p	8/63	1793	1891	430	845	325	48	792	214	519	264	1656	80	143	143	2105	4339	724	988	987	81.9%	82.8%	24.4%	61.5%
ham15	15/15	458738	246	147	361	165	15	108	57	181	32767	247	100	54	54	491520	601	304	415	400	100.0%	32.9%	-12.2%	54.3%
ham3	3/3	14	13	15	26	13	3	6	4	12	7	17	4	5	5	24	36	23	31	30	7.1%	0.0%	13.3%	50.0%
hwb4	4/4	45	40	42	56	24	4	14	11	28	15	38	22	12	12	64	92	75	68	64	46.7%	40.0%	42.9%	57.1%
hwb5	5/5	124	124	99	149	65	5	44	31	79	31	110	14	22	22	160	278	144	171	166	47.6%	47.6%	34.3%	56.4%
hwb6	6/6	315	297	181	348	155	6	104	58	187	63	246	32	46	46	384	647	271	394	388	50.8%	47.8%	14.4%	55.5%
hwb8	8/8	1785	2022	512	1548	677	8	631	156	863	255	1434	79	202	202	2048	4087	747	1750	1742	62.1%	66.5%	-32.2%	56.3%
mod5d2	5/5	125	29	24	39	24	5	11	6	19	31	30	13	8	8	161	70	43	47	51	80.8%	17.2%	0.0%	38.5%
One-two-three	3/3	14	21	16	21	10	2	7	4	10	3	21	9	4	4	19	49	29	25	24	28.6%	52.4%	37.5%	52.4%
plus127mod8192	13/13	98293	132	51	212	79	13	23	19	125	8191	65	36	53	53	106497	220	106	265	257	99.9%	40.2%	-54.9%	62.7%
plus63mod4096	12/12	45046	119	48	185	75	12	23	17	103	4095	64	33	41	41	49153	206	98	226	219	99.8%	37.0%	-56.3%	59.5%
plus63mod8192	13/13	98293	137	53	199	73	13	24	18	116	8191	66	36	51	51	106497	227	107	250	240	99.9%	46.7%	-37.7%	63.3%
rd53	5/3	112	61	46	88	41	3	16	12	44	5	41	24	17	17	120	118	82	105	102	63.4%	32.8%	10.9%	53.4%
rd73	7/3	699	219	80	214	97	3	55	26	116	7	135	29	34	34	709	409	135	248	247	86.1%	55.7%	-21.3%	54.7%
rd84	8/4	1785	352	107	301	136	3	69	37	165	156	156	37	57	57	1944	577	181	358	358	92.4%	61.4%	-27.1%	54.8%
urf1	9/9	4088	4857	2171	3864	1622	9	1351	621	2233	511	3071	441	438	438	4608	9279	3233	4302	4293	60.3%	66.6%	25.3%	58.0%
urf2	8/8	1785	1881	1154	1780	757	8	559	325	1015	255	1283	323	206	206	2048	3723	1802	1986	1978	57.6%	59.8%	34.4%	57.5%
urf5	9/9	4088	1002	949	1730	729	9	256	281	993	511	593	365	224	224	4608	1851	1595	1954	1946	82.2%	27.2%	23.2%	57.9%

* $\sum$ provides sum of numbers of MZI, combiner and splitter
** AIG always generates zero combiners

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 design solution 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

[1] L. Chrostowski and M. Hochberg, *Silicon Photonics Design: From Devices to Systems*. Cambridge University Press, 2015.

[2] R. Ho, P. Amberg, E. Chang, P. Koka, J. Lexau, G. Li, F. Liu, H. Schwetman, I. Shubin, H. Thacker, X. Zheng, J. Cunningham, and A. Krishnamoorthy, "Silicon Photonic Interconnects for Large-Scale Computer Systems," *IEEE Micro*, vol. 33, no. 1, pp. 68–78, 2013.

[3] T. Sato, K. Takeda, A. Shinya, M. Notomi, K. Hasebe, T. Kakitsuka, and S. Matsuo, "Photonic Crystal Lasers for Chip-to-Chip and On-Chip Optical Interconnects," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 21, no. 6, pp. 728–737, 2015.

[4] D. Jansen, *The Electronic Design Automation Handbook*. Kluwer Academic Publishers, 2003.

[5] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[6] T. Sasao, *Switching Theory for Logic Synthesis*. Kluwer, Dordrecht, 1999.

[7] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 677–691, 1986.

[8] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, 2002.

[9] A. K. Cherri and A. S. Al-Zayed, "Circuit Designs of Ultra-fast All-Optical Modified Signed-Digit Adders using Semiconductor Optical Amplifier and Mach-Zehnder Interferometer," *Optik - International Journal for Light and Electron Optics*, vol. 121, no. 17, pp. 1577 – 1585, 2010.

[10] K. Datta, T. Chattopadhyay, and I. Sengupta, "All Optical Design of Binary Adders using Semiconductor Optical Amplifier Assisted Mach-Zehnder Interferometer," *Microelectronics Journal*, vol. 46, no. 9, pp. 839–847, 2015.

[11] K. Datta and I. Sengupta, "All Optical Reversible Multiplexer Design using Mach-Zehnder Interferometer," in *International Conference on VLSI Design*, 2014, pp. 539–544.

[12] Y. Aikawa, S. Shimizu, and H. Uenohara, "Demonstration of All-Optical Divider Circuit Using SOA-MZI-Type XOR Gate and Feedback Loop for Forward Error Detection," *Journal of Lightwave Technology*, vol. 29, no. 15, pp. 2259–2266, 2011.

[13] C. Condrat, P. Kalla, and S. Blair, "Logic Synthesis for Integrated Optics," in *Great lakes symposium on VLSI*. ACM, 2011, pp. 13–18.

[14] E. Schönborn, K. Datta, R. Wille, I. Sengupta, H. Rahaman, and R. Drechsler, "BDD-Based Synthesis for All-Optical Mach-Zehnder Interferometer Circuits," in *International Conference on VLSI Design*, 2015, pp. 435–440.

[15] R. Wille, O. Keszocze, C. Hopfmuller, and R. Drechsler, "Reverse BDD-based Synthesis for Splitter-free Optical Circuits," in *Asia and South Pacific Design Automation Conference*, 2015, pp. 172–177.

[16] A. Deb, R. Wille, O. Keszöcze, S. Hillmich, and R. Drechsler, "Gates vs. splitters: Contradictory optimization objectives in the synthesis of optical circuits," *Journal on Emerging Technologies in Computing Systems*, vol. 13, no. 1, p. 11, 2016.

[17] M. P. Scaffardi, P. Ghelfi, E. Lazzeri, L. Poti, and A. Bogoni, "Photonic Processing for Digital Comparison and Full Addition based on Semiconductor Optical Amplifiers," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 14, no. 3, pp. 826–833, 2008.

[18] Q. Wang, G. Zhu, H. Chen, J. Jaques, J. Leuthold, A. B. Piccirilli, and N. K. Dutta, "Study of All-Optical XOR using Mach-Zehnder Interferometer and Differential Scheme," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 40, no. 6, pp. 703–710, 2004.

[19] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-strength Verification Tool," in *Proc. of International Conference on Computer Aided Verification*, 2010, pp. 24–40.