Exploiting Wavelength Division Multiplexing for Optical Logic Synthesis

Zheng Zhao, Derong Liu, Zhoufeng Ying, Biying Xu, Chenghao Feng, Ray T. Chen, David Z. Pan
Department of Electrical and Computer Engineering, University of Texas at Austin

***Abstract*—Photonic integrated circuit (PIC), as a promising alternative to traditional CMOS circuit, has demonstrated the potential to accomplish on-chip optical interconnects and computations in ultra-high speed and/or low power consumption. Wavelength division multiplexing (WDM) is widely used in optical communication for enabling multiple signals being processed and transferred independently. In this work, we apply WDM to optical logic PIC synthesis to reduce the PIC area.**

I. Introduction

As Moore's law is approaching the limits, photonic integrated circuits (PICs) have emerged as a promising alternative to CMOS in ultra-high speed and power efficient on-chip interconnects and computing [1]. Taking the advantage of optics, there are several recent works on optical logic synthesis. [2] introduces a synthesis framework based on optical virtual gates, where each literal in a logic function is implemented by a VG. The major problem is the significant number of optical switches and splitters. In the same paper, the authors also indicate the possibility of using binary decision diagrams (BDD) to build optical logics. The idea is to replace each BDD node by an optical crossbar switch. The light is sourced from the BDD terminal to the functional output. However, a critical problem is caused by the garbage outputs related to the switches due to the light flow direction, which further lead to complicated re-routing and/or the application of optical terminators. As a solution, [3] proposes to reverse the framework in the sense that the light streams from the BDD top node to the BDD terminal. By reversing the signal flow, the garbage outputs can be greatly reduced. A more recent work [4] further improves the optical power efficiency of the reverse-BDD architecture. However, the reverse-BDD architecture has the limitation that logic functions of multiple primary outputs (POs) are required to be separately built. Otherwise, it is impossible to differentiate the light received at the PD.

Wavelength division multiplexing (WDM), at this point, naturally emerges as a promising solution. WDM is a technology that enables multiple optical signals to be transmitted independently and simultaneously in a single waveguide [5]. Although WDM has been widely adopted in optical interconnects, its application to optical computing has scarcely been considered. In this work, we present an optical synthesis flow that exploits WDM to reduce the number of optical switches.

II. Background and Motivations

A. Optical Components and WDM

Optical crossbar switch is the basic optical computational unit, which can be implemented by resonator-based micro-rings or micro-disks, Mach-Zehnder interferometers, etc. Our following discussions are based on micro-rings. Yet, the high-level model is general enough for other realizations. Fig. 1a shows a schematic of a typical optical switch using micro-ring and the high-level notations of 1×2 and 1×1 switches. The micro-ring has one light input and two light outputs: the through output and drop output. One electrical input E_a can be applied as a control signal. A continuous wave light of wavelength λ is sourced into the switch. Part of the light is coupled into the micro-ring, and then coupled back to the waveguides with a certain phase shift. The combination of the light of a certain phase difference leads to a wavelength selective behavior depicted in Fig. 1b. The output ports can either have no light, where the transmission is close to 0, or have the input light transferred, where the transmission is close to 1. The behavior is dependent on the phase difference, which is further dependent on the voltage E_a. By using either both the output ports or one of the output ports, one can build either 1×2 switches or 1×1 switches. We define that if the controlling signal a of the switch is logical 1, the light is passed to the through port (marked as 1), otherwise to the drop port (marked as 0).

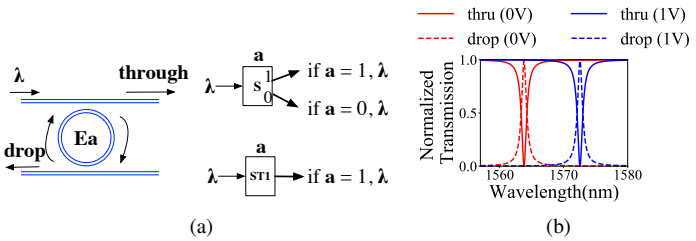


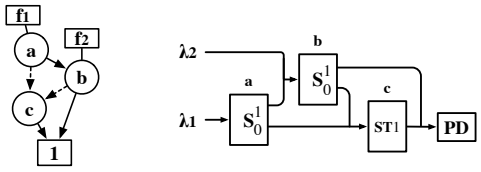
Fig. 1: (a) Schematic diagram of a micro-resonator and the 1×2 and 1×1 switch notations (b) optical transmission spectra measured at through and drop ports.

The micro-ring switch can also be used to realize WDM. A WDM system employs a multiplexer at the waveguide input port to join the signals, and a demultiplexer at the receiver to distinguish them. The multiplexer can be implemented by optical combiners, which are basically joint waveguides; and the demultiplexer can be implemented by micro-rings. One can choose a specific voltage bias to adjust the transmission curve, so that a specific wavelength can be filtered out. Generally, due to the limitation of integrated lasers on wavelength spacing, on-chip WDM has a capacity limit varying from 2 to 16 [6], [7].

B. BDD-based Optical Logic Synthesis

BDD is the fundamental data structure for optical logic synthesis A BDD is a directed acyclic graph that can represent a multi-primary output (PO) Boolean function. Fig. 2 shows a BDD of two POs: f_1 and f_2. There are two types of BDD nodes, decision nodes (e.g., circular node a, b and c) and terminal nodes

(e.g., square node 1). A decision node is functionally a crossbar switch with two outputs, marked by solid and dashed lines. The node is controlled by a decision variable v: if $v = 1$ (0), the solid (dashed) output path is selected. The control signals are primary inputs of the Boolean function, which are also the only electrical signals required by the BDD architecture. A terminal node has a value of 1 or 0, representing the functional output evaluation. If there exists a path from a PO to the 1-terminal (0-terminal), the PO is evaluated as logic 1 (0). Without modifying the functionality, we can keep the 1-terminal and delete the edges to the 0-terminal as shown in Fig. 2a. The POs' functions in the example can thus be written as $f_1 = a'c + ab'c + ab$ and $f_2 = b'c + b$. Common functional structures can be shared among different POs so that the number of BDD nodes is reduced.



(a) (b)

Fig. 2: WDM principle ; BDD and the optical implementation.

A BDD can be directly mapped to an optical implementation. Fig. 2b shows the implementation of Fig. 2a. The synthesis simply replaces each BDD node by an optical crossbar switch. Waveguides and optical combiners are employed to connect signals. The light (λ) from a laser (or from the output of the previous stage) is sourced from the BDD PO port. A photodetector (PD) (or optical amplifier to the next computation stage) is located at the 1-terminal. If there is light detected at the PD, the output of the optical network is logical 1, otherwise logical 0. Fig. 2b shows two wavelength λ_1 and λ_2 are sourced to represent the two POs. However, in the previous works, as WDM is not exploited, in order to distinguish the evaluation at the 1-terminal for different PO functions, the authors attempt to split the multi-PO BDD to several single-PO BDDs. As structure sharing is discouraged, it generally leads to a bigger BDD and hence bigger optical implementation. It can be calculated that the separation of Fig. 2b leads to two more optical switches. The difference would be non-trivial for bigger logic functions.

III. Synthesis Algorithms

The general flow of our proposed synthesis methods is shown in Fig. 3. Given an arbitrary multi-primary output (PO) logic function and WDM capacity constraint, the BDD reordering engine is called to build a BDD with all the PO functions. Given this information, the synthesis problem is first modeled to a hypergraph partitioning problem and solved to minimize the partitioning cost (`HyPart`). As the result of hypergraph partitioning contains infeasible partitions, a second resolving step is introduced (`ReFlow`). The resolving is achieved by modeling and solving a min-cost max-flow problem. The final result is hence guaranteed to be feasible. During the whole procedure, BDD reordering engine is called for multiple times for either the complete PO set or some PO subset in order to produce further improvement: at the beginning, at the end of `HyPart` and `ReFlow`.

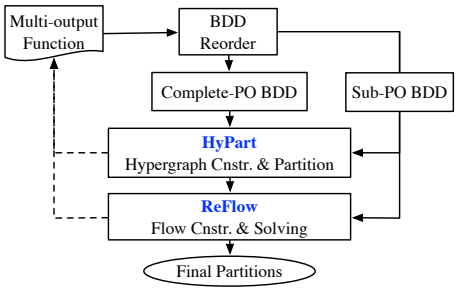


Fig. 3: Proposed Synthesis Flow.

A. HyPart: Hypergraph Partitioning

Based on the equivalence of BDD and optical implementation discussed in Section II, the BDD-based optical synthesis problem under WDM capacity constraint is presented in the following BDD-partitioning problem. We denote the WDM capacity by Cap.

Problem III.1. BDD partitioning problem (BPP). Given a BDD composed of $|PO|$ primary outputs and a constant Cap, the BDD partitioning problem is to find a partition of the BDD that has the minimum total number of BDD nodes. The number of functions of each partition is no greater than Cap.

In the following, we approximate BPP by a hypergraph partitioning problem (HPP). A hypergraph is a generalization of a graph in which a hyperedge can join any number of vertices. Formally, a hypergraph G_h is a pair $G_h = (V_h, E_h)$ where V_h is a set of vertices, and E_h is a set of non-empty subsets of V_h called hyperedges. As an example, Fig. 4a shows a hypergraph with four vertices marked by f_1 to f_4, three 2-vertex hyperedges represented by solid lines: (f_1, f_2), (f_2, f_3), (f_3, f_4); and one 3-vertex hyperedge represented by a curved area: (f_1, f_2, f_3). Each hyperedge weight w^* is indexed by the indices of the nodes connected to the hyperedge. For example, the weights of the hyperedge (f_2, f_3) and the hyperedge (f_1, f_2, f_3) are w^*_{23} and w^*_{123} respectively. The HPP problem is defined as follows:

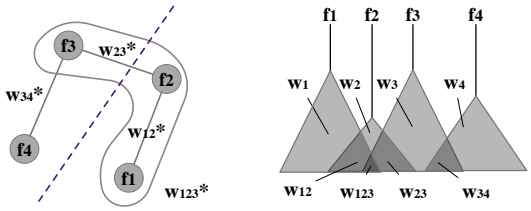
Problem III.2. Hypergraph partitioning problem (HPP). Given a hypergraph $G_h = (V_h, E_h)$, the hypergraph partitioning problem is to assign the vertices V into disjoint non-empty partitions so that the hyperedge cut weight is minimum.

Note that HPP is known to be NP-hard [8]. Given a multi-PO BDD, we construct the hypergraph by Algorithm 1. For each primary output f, a hypergraph node is created and the fan-out cone $Cone$ in the BDD is collected (Line 1-3). Each BDD node is then classified based on which PO cone it is contained. If multiple BDD nodes are shared by multiple PO's, they are merged to the same group G which is indexed by the corresponding PO set s (Line 4-7). A hyperedge is created for each such group whose s has more than 1 PO's. Let k be the actual number of partitions in a set s. The edge weight w_s is set to be the number of BDD nodes contained in the group ($|G_s|$) times a scaling factor $\alpha = k^* - 1$, where $k^* := \lceil |s|/Cap \rceil$ as a straightforward approxiamtion of k. If $k^* = 1$, it is set to 1 to avoid zero weights (Line 8-11).

Fig. 4b and 4a show a 4-PO BDD and its corresponding hypergraph respectively. There are four hypergraph nodes rep-

Algorithm 1 Hypergraph Construction for BPP.

1: **for** each primary output function f_i of BDD **do**
2: Create a hypergraph vertex marked by f_i
3: $Cone_i \leftarrow$ {BDD nodes in its fanout cone}
4: **for** each BDD node n_i **do**
5: $s_{cur} \leftarrow \{f_j$'s: $n_i \in Cone_j\}$, $G_{s_{cur}} \leftarrow \{n_i\}$
6: **if** s_{cur} is identical to an existing group G_s's index s **then**
7: Merge $G_{s_{cur}}$ and G_s to $G_{s \leftarrow s_{cur} \cup s} = G_{s_{cur}} \cup G_s$
8: **for** each group G_{s_i} whose $|s_i| > 1$ **do**
9: Create a hyperedge connecting all f's $\in s_i$, whose
10: weight $w^*_{s_i} := w_{s_i} \cdot \alpha_i$, where
11: $w_{s_i} := |G_{s_i}|$, $\alpha_i := k^*_{s_i} - 1$, $k^*_{s_i} := \lceil |s_i|/Cap \rceil$



(a) (b)

Fig. 4: Hypergraph Modeling Example.

resenting POs and four hyperedges, each corresponding to a shared region. The area of the regions in terms of the BDD nodes is denoted as w_{12}, w_{123}, w_{23}, w_{34}, respectively. Each hyperedge cost is marked by a corresponding weight $w^* := w \cdot \alpha$, which is w weighed by the scaling factor α described above. In the hypergraph, the edge cost is the region area weighted by a constant $\alpha_{s_i} = k^*_{s_i} - 1$. $k^*_{s_i} = \lceil |s_i|/Cap \rceil$ can be viewed as a lower-bound approximation of k_{s_i}: the actual number of partitions in set s_i. The fundamental idea of linking HPP and BPP is that, for HPP the lower cut cost meaning the fewer shared nodes are separated to different partitions, then for BPP, the less area overhead is introduced by a certain partition solution. Once the partitions are computed, BDD reordering engine is called again as a post-optimization.

B. ReFlow: Resolving Infeasible Partitions

`HyPart` does not always produce perfectly balanced partitions, in the sense that the numbers of elements in some partitions are smaller than the specified capacity of WDM and others are greater. The latter case is infeasible in the context of optical implementation. To resolve the problem, the following algorithm `ReFlow` is proposed to balance the element allocation in each partition. Basically, the above partition methodology has achieved a near-balanced solution but with a small number of capacity violations. The key idea of `ReFlow` is to transfer POs from the partitions over the WDM capacity to those under the capacity. Ultimately, we obtain a solution which satisfies the capacity constraints. We will show how to solve the problem through a min-cost max-flow model, which demonstrated effective application in physical design problems [5].

Fig. 5 is a pseudo-example illustrating how the network flow model works. Suppose in the partition solution given by the previous step, P_1 and P_2 are the over-capacity partitions that require re-allocation and P_3 and P_4 are the under-capacity partitions that can accommodate extra elements. To construct the flow graph, we require the following flow nodes: (1) node P_1 and P_2, corresponding to over-capacity partitions (marked in Column-1); (2) node f_1 to f_3, corresponding to the PO's contained in the over-capacity partitions (marked in Column-2); (3) node P_3, P_4, P'_1 and P'_2 corresponding to the target partitions (marked in Column-3); (4) node s and t representing the pseudo starting and terminating nodes. In Column-3, P_3 and P_4 are the under-capacity partitions and P'_1 and P'_2 mirror the original over-capacity partitions P_1 and P_2, which allows the possibility that nodes in P_1 and P_2 can still stay in the original partition as long as the flow is feasible, i.e., the capacity constraint is satisfied. As for the flow edges, one flow edge is created: (1) from an over-capacity P node to an f node, if $f \in P$ given by `HyPart`; (2) from each f node to each under-capacity P node; (3) from an f node to an mirroring P' node if $f \in P$ given by `HyPart`; (4) from s to each over-capacity P node; and from each under-capacity P node and mirroring P' node to t.

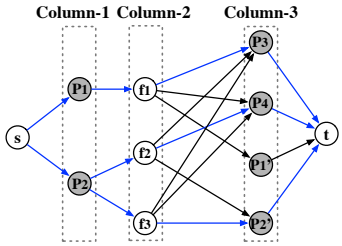


Fig. 5: `ReFlow` Example.

Intuitively, if we observe a flow from f_i to partition P or P' in Column-3, it means f_i's eventual destination is the partition P or P'. For the example in Fig. 5, the flow solution marked by the blue edges means $f_1 \in P_1$ is moved to P_3, $f_2 \in P_2$ is moved to P_4, and $f_3 \in P_2$ stays in P_2. To guarantee a feasible re-assignment, the edge capacities are set as follows. Each edge from s to a Column-1 P node has a capacity equivalent to the number of f's in the partition P to accommodate for the required flow in the next step. Each edge from a Column-1 over-capacity P node to an f node has a capacity of 1, which constrains that f can only be reallocated *from* one partition; similarly, each edge from f to a Column-3 under-capacity P or to a mirroring node P' has a capacity of 1, which constrains that f can only be reallocated *to* one partition. Finally, each edge from a Column-3 over-capacity P node to t has a capacity of the availability, i.e., the difference between the WDM capacity and the number of the existing POs; each edge from a Column-3 mirroring P' node to t has a capacity of the WDM capacity, to represent the maximum allowable capacity.

As for the edge cost, noting that our objective is to reassign the elements in a way that minimizes the reassignment cost, how to estimate the reassignment cost is critical. In our method, the cost for a flow edge from a f node to an under-capacity node P is evaluated in a local perspective, i.e., by the number of BDD nodes resulted by adding f to P. BDD reordering engine is called to obtain this value. The higher cost will discourage the reassignment greater. For all the other edges, the cost is 0 so that flowing is maximumly encouraged. as long as the flow edge capacity is satisfied. Hence, it is also encouraged that an f node stays in the original partition. In this way, the original partition solution, which is achieved by a more global perspective, are observed to the greatest extent.

TABLE I: Experimental results with different WDM capacities.

		# of BDD nodes		wdm cap.=4			wdm cap.=8			wdm cap.=16		
benchmark	#po	sep.	all	ours	rand.	time	ours	rand.	time	ours	rand.	time
cht	36	154	89	107	133.6	0.2	98	123.7	0.1	93	112.6	0.03
apex7	37	458	316	306	413.0	1.0	283	377.0	0.5	280	352.7	0.1
stpmotor	29	491	243	358	461.0	0.9	283	385.1	0.1	246	310.2	0.2
x4	71	602	366	457	582.0	3.6	402	527.0	1.1	373	463.1	0.2
example2	66	645	269	379	494.2	1.4	306	416.0	0.2	283	358.9	0.1
i5	66	672	133	221	480.4	0.3	175	385.2	0.2	151	301.0	0.2
x3	99	851	590	674	813.2	6.1	631	770.7	0.3	623	839.8	0.3
pdc	40	960	603	680	874.0	4.1	672	798.6	0.8	610	732.8	0.2
spla	46	977	592	708	837.3	6.0	660	778.8	1.3	593	712.5	1.0
vda	39	1117	549	721	912.8	1.3	644	794.2	0.2	601	694.6	0.2
apex5	85	1410	1084	1247	1379.9	24.3	1140	1338.8	4.2	1107	1269.2	0.5
simple_spi	144	1473	983	1282	1498.5	15.2	1151	1428.0	7.6	1027	1329.4	11.6
i2c	140	1836	1131	1379	1690.2	36.8	1286	1565.2	14.7	1220	1441.0	11.4
frg2	139	1981	1402	1577	1965.7	14.0	1347	1897.5	5.1	1178	1836.6	13.0
i9	63	2079	1691	1912	2062.4	4.7	1855	2011.0	3.6	1793	1935.6	3.2
k2	43	2113	1295	1670	1956.1	3.8	1483	1824.1	0.4	1372	1637.8	0.4
cps	102	2224	1027	1288	1653.7	26.6	1087	1498.5	9.0	1011	1373.2	0.5
i8	81	2368	2120	2096	2316.3	5.0	1900	2211.8	3.0	2150	2033.6	0.5
seq	35	2468	1823	1790	2346.4	3.5	1943	2263.9	0.6	1850	2174.8	0.6
average	71.63	1309.42	858.21	992.21	1203.72	8.80	912.95	1126.10	2.80	871.63	1047.86	2.3
# of sw		1309.42	906.93	1063.84	1275.35		984.58	1197.69		943.26	1119.49	

IV. Experimental Results

We implemented the proposed flow in C++ with CUDD package [9], and tested it on a 3.4GHz Linux machine. The following experiments were conducted on International Workshop on Logic and Synthesis and Microelectronics Center of North Carolina benchmarks [10], [11]. The hMetis binary [12] was applied as the hypergraph partitioning solver and the open library LEMON [13] was applied as the min-cost max-flow network solver. We apply `CUDD_REORDER_SYMM_SIFT`, a common BDD reordering heuristic available in CUDD.

Table I listed the number of BDD nodes and runtime under different WDM capacity constraints. The averages of the benchmarks are listed in the last 2 rows. The first two columns list the benchmark name and the number of primary outputs (#PO). The next two columns denoted with sep and all show the number of BDD nodes for the two extreme cases that all POs are separated (which is also the method of the previous work), and that all POs are grouped in one, respectively. It can be seen that the number of BDD nodes for the sep case is 52% greater than the all case on average. Column 5-7, 8-10, 11-13 for the benchmark list the results for three WDM capacity settings: 4, 8 and 16. The columns named as $ours$ correspond to the number of BDD nodes given by our method. The rows named by $\#\ of\ sw$ compute the actual number of optical switches by adding the number of filtering switches, which in the worst case, is equal to the number of POs. The ratios of the switch number of our method compared to that of the previous $sep.$ method are 81.2%, 75.2% and 72.0% on average for WDM capacity of 4, 8 and 16, respectively. To make a fair comparison, we also compute 10 random partition solutions following the procedure described above, whose averages are listed in the $rand.$ columns. The ratio compared to the random solutions are 83.4%, 82.2% and 84.3% on average. We note that, as to obtain the optimal BDD (i.e., with the optimal variable order) is known to be NP-hard [14] and BDD reordering algorithms are essentially heuristic, the optimality of the computed BDD is targeted but is not guaranteed. As a result, in some cases, the BDD size under capacity constraints can even be smaller than those without the constraint. We attribute this phenomenon to the heuristic nature of BDD reordering engine. Finally, the total runtime in seconds for our methods is listed under the $time$ columns. The average is about 8.8s, 2.8s and 2.3s, for the three capacities, respectively. The reason for the decrease is that, when the capacity of WDM increases, the chance of infeasible partition solutions produced by hMetis, thus the need for `ReFlow`, also decreases.

Acknowledgement

We acknowledge the Multidisciplinary University Research Initiative program through the Air Force Office of Scientific Research, contract No.FA 9550-17-1-0071, monitored by Dr. Gernot S. Pomrenke.

References

[1] D. A. Miller, "Attojoule optoelectronics for low-energy information processing and communications," *Journal of Lightwave Technology*, 2017.

[2] C. Condrat, P. Kalla, and S. Blair, "Logic Synthesis for Integrated Optics," in *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*.

[3] R. Wille, O. Keszocze, C. Hopfmuller, and R. Drechsler, "Reverse BDD-based Synthesis for Splitter-free Optical Circuits," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, 2015.

[4] Z. Zhao, Z. Wang, Z. Ying, S. Dhar, R. T. Chen, and D. Z. Pan, "Logic synthesis for energy-efficient photonic integrated circuits," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, 2018.

[5] D. Liu, Z. Zhao, Z. Wang, Z. Ying, R. T. Chen, and D. Z. Pan, "Operon: optical-electrical power-efficient route synthesis for on-chip signals," in *Proceedings of the 55th Annual Design Automation Conference*, 2018.

[6] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. M. Shainline, R. R. Avizienis, S. Lin *et al.*, "Single-chip microprocessor that communicates directly using light," *Nature*, 2015.

[7] J. M. Kahn and K.-P. Ho, "Spectral efficiency limits and modulation/detection techniques for dwdm systems," *IEEE Journal of selected topics in quantum electronics*, 2004.

[8] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems*, 2006.

[9] F. Somenzi, "CUDD: Colorado university decision diagram package," http://vlsi.colorado.edu/~fabio/CUDD/, accessed: 2015-09-30.

[10] "IWLS 2005 benchmarks," http://iwls.org/iwls2005/benchmarks.html, accessed: 2015-09-30.

[11] "MCNC benchmarks," https://s2.smu.edu/~manikas/Benchmarks/MCNC_Benchmark_Netlists.html, accessed: 2015-09-30.

[12] hMETIS, "Hypergraph and circuit partitioning." http://glaros.dtc.umn.edu/gkhome/metis/hmetis, accessed: 2017-05-30.

[13] LEMON, "LEMON: Minimum cost flow algorithms," http://lemon.cs.elte.hu/trac/lemon, accessed: 2017-01-30.

[14] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM Computing Surveys (CSUR)*, 1992.