Logic Synthesis for Integrated Optics

Christopher Condrat
chris@g6net.com

Priyank Kalla
kalla@ece.utah.edu

Steve Blair
blair@ece.utah.edu

Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT, USA

ABSTRACT

As silicon photonics technology matures, optical devices will be available on a scale never before seen or utilized. It is therefore imperative to develop automated methods for synthesizing optical devices for large-scale designs. We present design and synthesis methodologies for implementing digital logic using conventional integrated optical components, specifically optical cross-bar routing devices based on Mach-Zehnder Interferometry. Our design methodologies utilize the unique advantages of these optical devices, while also addressing the limitations of the technology. We extend these design concepts to include technology-specific logic sharing, and provide automated techniques for logic design implementation, evaluating the efficacy of our techniques on a number of logic designs. Through the convergence of communications and computing, optical devices are utilized on scales beyond traditional optic design.

Categories: B.6.3 [**Logic Design, Automatic Synthesis**]

Terms: Algorithms, Design

Keywords: Optics, Logic Synthesis, XOR, Automation

1. INTRODUCTION

The semiconductor industry has long prospered through steady advancements in static-CMOS technology, enabling higher performance chip designs with each generation. This trend, however, is already showing its limitations [1], and chip design is moving towards multi-core processing, where communications are extremely important [2]. This is leading researchers in industry and academia to investigate *complementary* technologies for static-CMOS, and a great focus of this research has been on silicon-based integrated optics – *silicon photonics* – to leverage already available and mature silicon-based infrastructure, and to overcome the traditional barriers that have prevented silicon from being used in key integrated optic devices. A recent surge of research has yielded a number of important recent breakthroughs [3] [4] [5].

These developments and others are enabling optical devices to be deployed cheaply and on a large scale, allowing far greater flexibility in integrating such devices into systems. Wide availability of optical devices will allow chip designs to take advantage of optics for purposes beyond simply communications, and on a scale far beyond the limitations imposed by the traditional high costs of integrating optical devices. So far, the main application in mind for such devices is for use as optical interconnects [2]; however, interest in their use in optical computing has also been mentioned as an application [6].

GLSVLSI'11, May 2–4, 2011, Lausanne, Switzerland.

This convergence of communication and computation motivates the need to address the integration of photonic logic design.

This paper explores the use of optical switching devices for logic design and synthesis. A key aspect of this research is the use of conventional routing devices, specifically optical crossbar routing devices, as the *building block* for logic design. This enables us to apply logic synthesis methodologies for large-scale integration of optical devices, as well as optimize logic networks using common sub-expression sharing compatible with the technology. We describe our models, synthesis methodologies, and their applications to logic examples in the next sections.

2. BACKGROUND AND DEVICE MODELS

One of the goals of this work is to develop synthesis techniques that utilize conventional integrated optics devices that can be fabricated with current technology, while also being applicable to future design processes. We describe the basic operation of the integrated optic devices we utilize. The constraints of the physical device model are key to the logical constraints of our basic logic elements, and ultimately to the logic synthesis methodology we describe.

Integrated Optic Devices: Integrated optical devices transmit and route light using optical waveguides [7], which are created as guiding layers on the substrate using lithographic and deposition methods. In contemporary systems, light is coupled into the system from the outside using a laser, and sensed, at the destination, by optical receivers using a light-detection material such as germanium [8].

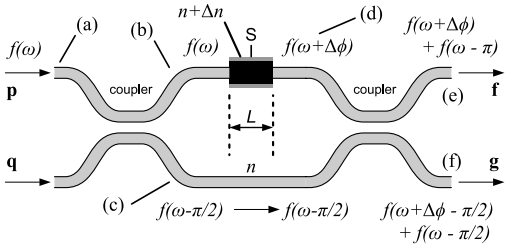
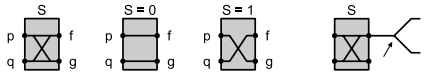


Figure 1: Operation of Mach-Zehnder Interferometer.

Our basic optical logic element is the Mach-Zehnder Interferometer (MZI). The MZI is a conventional integrated optic device found in many designs, used for modulation [3] [4], but also routing through the use of coupling and controlled interference. Consider the MZI depicted in Fig.1, with inputs **p** and **q**, and outputs **f** and **g**. Between **p** and **f** and **q** and **g** are waveguides, with an index of refraction n. Coupling occurs when two waveguides are brought within in close proximity to each other such that the electromagnetic fields in one waveguide extend over the other waveguide and vice versa, causing energy to cross over between one waveguide to the other, as a function of coupling length. The couplers in this device are 3dB couplers, tuned to divide and/or combining the signal from both inputs equally between the two outputs. In Fig.1, the signal at (a) passes through the 3dB coupler and is divided between the outputs (b) and (c), inducing a $\pi/2$ phase change in (c) (note that only the phase relationships are depicted).

In the center region, **S** is an outside input used to affect the refractive index of the upper waveguide by Δn by using methods/devices such as microheaters, carrier injection, advanced methods such as high-speed MOS-capacitors [3], or other means. This change in refractive index causes a path-length difference, and therefore a phase difference, between the signals in (b) and (d). This phase difference causes constructive or destructive interference at the second coupler when the signals from (c) and (d) are combined. Though we have only depicted one signal input at **p**, an input at **q** operates in the same manner, but its output arrives at the opposing output that the input from **p** does. A phase difference of 0 or π will route each input *completely* to one output or the other, and the device acts as the *controlled crossbar* depicted in Fig.2(a). Similarly, other MZI designs [9] [10] also operate on the same principle.



(a) Gate (b) Bar (c) Cross (d) Splitter

Figure 2: Crossbar switch, and different routing configurations.

Our Device Model: The operation of the MZI allows us to model it as a crossbar *gate* that routes light signal completely between two paths depending on the state of **S**, and depict it symbolically in Fig.2(a), with its two states Fig.2(b) and Fig.2(c) (bar and cross respectively). The waveguides are sourced by light (logical "1") or darkness ("0"), and the output of a function is read using optical receivers at the end. In our model, the switching input **S** is an *electrical* signal; it is an outside signal that controls the cross/bar configuration and cannot be switched by optical inputs. Connections to **p** and **q**, and **f** and **g** are waveguides, and for simplicity, light is assumed to move from the **p** and **q** side to **f** and **g**. In our model, *an optical signal cannot directly switch a crossbar's S input*[1]. More formally: $(\mathbf{S}=0) \Rightarrow (\mathbf{p}=\mathbf{f}) \wedge (\mathbf{q}=\mathbf{g})$; $(\mathbf{S}=1) \Rightarrow (\mathbf{q}=\mathbf{f}) \wedge (\mathbf{p}=\mathbf{g})$. These constraints affect how functions may be composed, and imply that the inputs to a crossbar are the primary inputs for that network. Waveguide connections between crossbar gates are depicted symbolically as black "wires." All designs created using the above model can be physically realized, including allowing waveguides to cross each other without interference.

In addition to MZIs, we also utilize *optical splitters,* depicted symbolically in Fig.2(d). A splitter divides the light from one waveguide into two output waveguides, each of which contain the original signal, but at half the power (a 3dB loss). In our model, splitters are the only signal degradation mechanism for a given topology, as we assume that there are no losses due to waveguide bends or insertion losses for MZI devices. Such losses can be factored into heuristics once physical layout information is available; however, this is currently beyond the scope of this work.

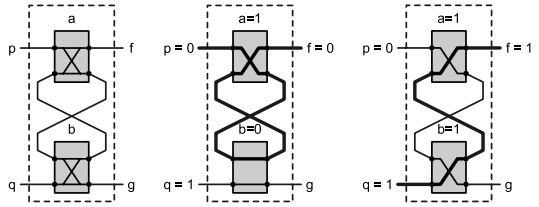
Previous work using MZIs: Optical logic design using crossbar routing devices has been investigated in literature. Shamir, Caulfield, et al. and others investigated the use of optical crossbar gates as *Fredkin gates* [11] [12]. The Fredkin gate model assumes that an optical input can *also* drive the switching input of a gate, allowing the gate to be used in a reversible logic role, but precluding its applicability to our device model. There has also been research in non-Fredkin crossbar gates [13] [14] [15] demonstrating the potential for implementing digital logic using MZIs; however, these are generally confined to small demonstrative circuits that do not scale to larger design implementations and arbitrary logic functions. More recently, techniques such as [16] investigate the integration and routing of optical interconnects; however, such work is for routing, not implementing logic. We therefore need a synthesis methodology that enables the construction of arbitrary logic functions using MZI crossbar devices, that can also scale to larger designs if necessary. To this end, we introduce the concept of *Virtual Gates*, a scalable methodology for implementing Boolean functions as a network of nested template functions constructed from interconnected MZI gates. We explore how these may be used directly, and how their limitations motivates a technique for logic sharing without violating the opto-electrical barrier.

[1] Switching a crossbar gate with an optical signal requires an opto-electrical interface comprising an optical receiver unit feeding switching hardware. This can be expensive and slow, and is currently beyond the scope of the synthesis technique applied to this device model.

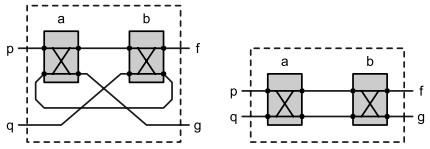
3. LOGIC DESIGN WITH VIRTUAL GATES

A *virtual gate* (VG) is – functionally and conceptually – a crossbar gate that is switched by a *function,* not necessarily a primary input. The gate is "virtual" in the sense that it is a black box for a function composed of "real" gates – those driven by primary inputs – as well as other virtual gates. A novel form of nesting can be used to compose VG function implementations, where Boolean operators are implemented by replacing child gates with other gates, a real or virtual.

A given VG implementation comprises two input waveguide ports **p** and **q** connected by waveguides and crossbar gates to two output ports **f** and **g**. The nesting operation comprises the Boolean operator forms depicted in Fig.3, and is illustrated in Fig.4(a) where two AND virtual gates are nested within an OR virtual gate, creating the final function $ab+cd$. Evaluation of a VG, given a primary input assignment, involves assigning **p** and **q** inputs logical 0 and 1 respectively, and applying *cross* or *bar* configurations to gates as defined in Fig.2. The output of the function is detected at **f**, with $\mathbf{g}=\neg\mathbf{f}$.



(a) AND(a,b) (b) AND$(1,0)$ (c) AND$(1,1)$



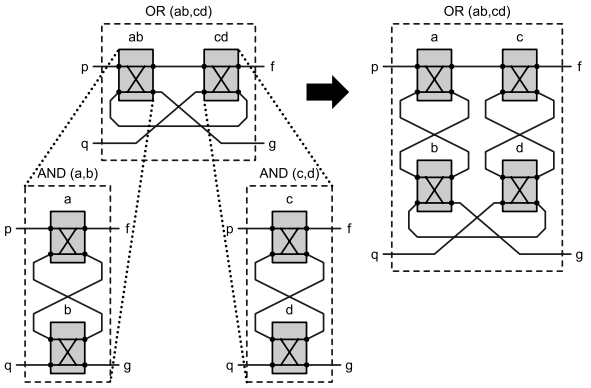
(d) OR(a,b) (e) XOR(a,b)

Figure 3: Virtual Gate templates for Boolean operators.

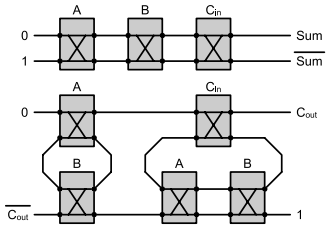
Example: Consider the VG implemented in Fig.3(a) where two inputs, a and b comprise inputs to the function $\mathbf{f}=a\cdot b$, and $\cdot$ is the Boolean AND operator. If we assign $a=1, b=0$, this evaluates $\mathbf{f}=a\cdot b|_{a=1,b=0}=0$. This is depicted on an AND-VG in Fig.3(b) where $a=1$ causes the a gate to assume a *cross* configuration, and $b=0$ causes the b gate to assume a *bar* configuration. Tracing the waveguide from the function output **f** to its optical inputs **p** or **q** reveals that the **f** connects to $\mathbf{p}=0$, the correct evaluation for the assignment. Likewise, if $a=1, b=1$, both gates assume a *cross* configuration as depicted in Fig.3(c); tracing **f** to its inputs connects it to $\mathbf{q}=1$ the correct evaluation for $\mathbf{f}=a\cdot b|_{a=1,b=1}=1$. The correctness of other AND input-assignments, and other Boolean operators can be verified by inspection.

All waveguide inputs in a VG are completely routed to outputs. Therefore the output at **g** is always the functional negation of **f**, as routing **f** to one input **p** or **q** will force the other input to be routed to **g**. Negation of primary inputs on real gates is assumed to be performed by the outside circuitry; however, *function* negation of gate outputs

(virtual or real) can be performed by swapping the two outputs **f** and **g**. For example, in Fig.4(a), the output of *ab* could be inverted by connecting **f** to the line that **g** is connected to, and vice-versa, resulting in $\neg(ab)+cd$. Note that the $\mathbf{g}=\neg\mathbf{f}$ and function negation conditions do not hold for arbitrary connections of crossbar gates, only those that are constructed as VGs.



(a) Virtual Gate nesting implementing $\mathbf{f}=ab+cd$.



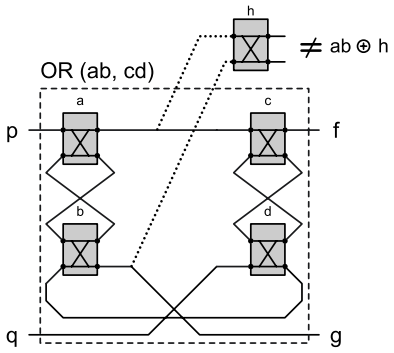
(b) Factored-form full adder implementing $sum=a\oplus b\oplus C_{in}$; $C_{out}=ab+C_{in}(a\oplus b)$.

Figure 4: Implementations using Virtual Gates.

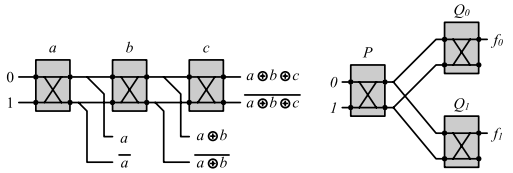
The VG topology generally reflects how actual waveguides would be implemented in an integrated optic substrate. Electrical connections to crossbar gates (MZIs) are made from above to active elements, and waveguide inputs/outputs are connected to the edge of the substrate or through waveguide gratings [17]. Devices may be oriented as needed to avoid excess waveguide congestion, such as depicted in Fig.4(b), or to meet routing constraints. Waveguides may also cross without interference by utilizing the same type of waveguide couplers found in MZIs, but tuned for complete crossover. In addition, unused waveguide outputs – deemed "garbage outputs" – must be properly accounted for. Garbage outputs cause problems because the signals, and the light/energy they carry, may interfere with the operation of the network if not properly "disposed of" at the edge of the substrate. The additional waveguides needed for garbage outputs can cause congestion and complicate the overall physical routing of the network. VG networks have the advantage of producing only a single garbage output per function if **g** is not used.

Salient features: 1) The nesting operation for constructing VG networks allows Boolean functions to be implemented in any form (SOP, factored, AND-XOR, etc.); an example of a factored-form implementation of a full-adder is depicted in Fig.4(b), where $C_{out}=ab+C_{in}(a\oplus b)$. 2) The total number of real gates is the number of literals in the expression. 3) The output **g** is always the functional complement of **f**. 4) VGs produce at most *one* garbage output per function output.

Expression Sharing with Virtual Gates: Virtual gates can implement any single-output Boolean function in a factored form. This however, has some caveats: the inability to drive gate inputs using optical inputs implies that only splitters may be used to share expressions. This places some severe constraints on VG expression sharing, namely that sharing is not possible using operators connecting operands via *loops* (e.g. AND and OR operators). Common sub-expressions connected via AND or OR operators must be *reimplemented* (replicated) wherever they appear. Consider the attempted expression sharing depicted in Fig.5(a), where an *h* gate attempts to tap the output of the left $a\cdot b$ VG as an XOR. The input to the $a\cdot b$ gate output is not, however, $a\cdot b$, rather it is functionally *undefined.* For example, if $a=b=c=d=1$ all real gates in Fig.5(a) will be in a cross configuration, which will result in the top waveguide, tapped by *h*, being isolated from any driven input. The cause of this unknown function state is the presence of loops in the design, which may or may not isolate parts of the network from the signal path.



(a) Internal functions of Virtual Gates cannot be shared.



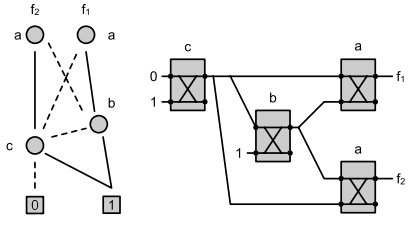
(b) XOR expression sharing is hierarchical.

(c) XOR decomposition structure.

Figure 5: Virtual Gate expression sharing.

Expression sharing using VGs is only possible through the use of the XOR operator, as it does not contain loops in its construction. This has its limitations, however: expression sharing is *hierarchical*. Consider the three VGs XORed together in Fig.5(b). If we attempt to share the output of the *b* VG, the output is not the *function b*, but rather the function formed by *b* and its inputs: $a\oplus b$. Sharing the function *b* directly would require a separate *b* gate driven only by 0 and 1 at its inputs. Therefore, common sub-expression extraction, as implemented in CMOS technologies, cannot be applied to VG networks. However, it is still possible to perform expression sharing by means of XOR decomposition, the structure of which is depicted in Fig.5(c).

A BDD structure [18], such as Fig.6, can facilitate expression sharing if one considers the crossbar as a pair of multiplexors.



(a) BDD graph (b) BDD-based design

Figure 6: BDD-based Design for $f_1=ab+c$, $f_2=\bar{a}b+c$.

However, the structure of a BDD is such that potentially *every gate* in the structure has has a garbage output. In [19], BDD-decomposition-based synthesis methods are compared against pure and hybrid BDD-VG networks methods, showing an extreme discrepancy in numbers of garbage outputs when BDD-based structures are employed. It is for this reason that we do not use a BDD-based structure for expression sharing, and choose expression sharing methods based on VG XOR operators.

XOR-based Expression Sharing: Our goal in expression sharing is to reduce the overall literal count – and therefore gate count – by sharing functionality. The network topology in Fig.5(c) depicts f_0 and f_1 sharing a common sub-expression P through the relationship in Eqn.1. Ideally, this relationship will reduce the total literal cost of the design.

$$f_0 = P \oplus Q_0 \qquad f_1 = P \oplus Q_1 \tag{1}$$

$$f_0 = (P \oplus m) \oplus (Q_0 \oplus m) \qquad f_1 = (P \oplus m) \oplus (Q_1 \oplus m) \tag{2}$$

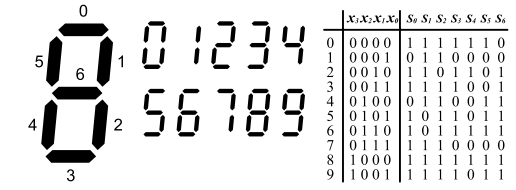
Reducing literal count utilizing the XOR operator is not as straightforward as with AND or OR operators [20] [21], and more difficult considering the feed-forward topology in Fig.5(c). We approach this problem as one of adding or removing cubes from the sub-functions P, Q_0, and Q_1.

Consider the case where an arbitrary term m is XORed with the right-hand-sides of both equations of Eqn.1. In order to balance the equations, we use the XOR identity $a \oplus a = 0$, requiring that m must be added again to each of the equations. If we group the terms as depicted in Eqn.2, what can be taken from the result is that simultaneously adding a term m to all three functions P, Q_0 and Q_1 does not change the functionality of f_0 and f_1. We can choose terms such that one or more of the functions are simplified, as a means of *optimizing* the overall expression-sharing VG network, as we will see in the following example.

Motivating Example: Consider the binary-coded-decimal (BCD) to 7-segment display in figure Fig.7(a), which converts a BCD to a visual representation of a number by turning segments on or off (1 or 0) depending on the value (0-9) of the BCD ($x_3x_2x_1x_0$). The truth table for segments 0 and 1 (S_0 and S_1) is depicted on the right side of Fig.7(a) (unlisted rows are assumed to be zero). The table is mapped to Karnaugh maps (K-maps) depicted in Fig.7(b), allowing us to derive the prime implicants for the functions, and the resulting sum-of-products (SOP) equations below the K-maps. Through this method, the total literal count for the two SOP expressions is 21 literals, requiring 21 crossbar gates if implemented as virtual gate networks.

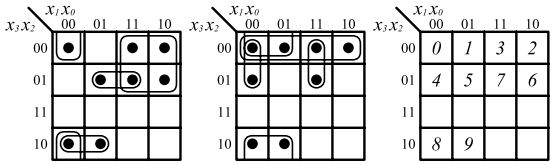
We now decompose S_0 and S_1 into functions P, Q_0 and Q_1 as depicted in Fig.5(c), where $S_0 = P \oplus Q_0$ and $S_1 = P \oplus Q_1$, and initially assign $P := 0$, $Q_0 := S_0$, and $Q_1 := S_1$. At this point, the network is essentially the same as implementing S_0 and S_1 separately. Now consider the case where we XOR an expression k with P, Q_0 and Q_1, where k is the intersection of minterms contained in Q_0 and Q_1. This operation has the effect of cancelling those minterms in P, Q_0 and Q_1 that are also contained in k, and adding them if not. This new set of functions is actually *less* optimal than the original (depicted in Fig.7(c)), because some of the larger cubes are broken up in the XOR operation. These less-optimal functions can, however, be improved by repeating the operation using minterms 1 and 6, affecting all three functions P, Q_0, and Q_1 resulting in the K-maps and functions in Fig.7(d). The final set of functions uses only 10 literals total – *11 gates less* than implementing the original functions separately. This example demonstrates the potential for good common expression sharing by "adding" and "removing" *minterms* from the decomposition functions. The same operation can also be extended to cubes in general.

Note that while the decomposition is XOR-based, the sub-functions P, Q_0 and Q_1 are implemented as VGs in any form – SOP, factored forms, etc. We choose the most optimal form for VG implementation. This is a novel feature of our decomposition technique as compared to conventional XOR-based optimization methods.

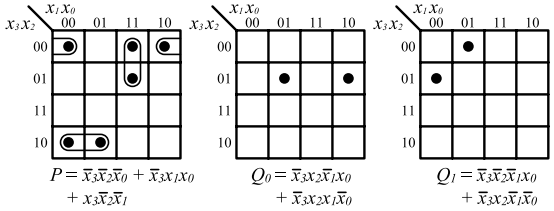


	$x_3x_2x_1x_0$	S_0	S_1	S_2	S_3	S_4	S_5	S_6
0	0 0 0 0	1	1	1	1	1	1	0
1	0 0 0 1	0	1	1	0	0	0	0
2	0 0 1 0	1	1	0	1	1	0	1
3	0 0 1 1	1	1	1	1	0	0	1
4	0 1 0 0	0	1	1	0	0	1	1
5	0 1 0 1	1	0	1	1	0	1	1
6	0 1 1 0	1	0	1	1	1	1	1
7	0 1 1 1	1	1	1	0	0	0	0
8	1 0 0 0	1	1	1	1	1	1	1
9	1 0 0 1	1	1	1	1	0	1	1

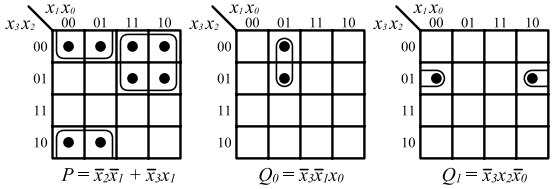
(a) 7-segment display and truth table for segments 0 and 1.



(b) Karnaugh maps for S_0 and S_1 (unmarked grids = offset).



(c) The original P, Q_0, and Q_1 after XORing with k.



(d) Optimized P, Q_0, and Q_1.

Figure 7: BCD-to-7 Segment Display

Limitations of contemporary XOR-based synthesis techniques: XOR-based optimization is well studied in literature. Techniques such as [20] [21] are designed to minimize Exclusive-Sums-of-Products (ESOPs) expressions with exact and fast heuristic methods. ESOP expressions are, however, very limiting compared to VG networks, as they comprise only AND-XOR terms. Decomposition methods have also been explored, using graph structures and concepts such as x-dominators [22] to find structural XOR relationships. However, an XOR decomposition can only be extracted if found on the graph structure; an XOR decomposition with expression sharing always exists in our approach. [23] addresses some shortcomings of previous decomposition methods, by finding linear relationships between sub-functions of form $f = g_1h_1 \oplus g_2h_2$, thereby reducing the area of XOR-based logic functions. While this performs an XOR decomposition, it does not create common sub-expression sharing by design. The technique described in [24] applies a heuristic method of sharing sub-functions of positive-polarity Reed-Muller expansions for Toffoli gate synthesis. However, as with other Toffoli synthesis methods [25], expression sharing of this type is incompatible with our approach because expression outputs cannot be shared across the opto-electro barrier. We therefore present a technique for finding XOR decompositions for VG networks, while simultaneously performing common sub-expression sharing.

4. MULTI-OUTPUT EXPRESSION SHARING

We represent functions using ROBDDs [18]; this enables a more compact representation of functions, while allowing efficient XOR-based manipulations. Rather than using minterms to manipulate functions, as in the motivating example, we use cubes derived from the BDDs. The number of literals in the function (our metric) is the sum of all literals from the cubes of a BDD. It should be noted that while the literal count of the BDDs is used as a metric during synthesis, the BDD is used soley as a function-manipulation data-structure, not as a technology-mapping/implementation structure.

Two functions can be decomposed into a structure depicted in Fig.5(c). This can be extended to a multi-level decomposition by repeating the process hierarchically. The function in Algo.1 implements this procedure as a top-level function decomposition from a multi-output design, returning a decomposition tree of sub-functions representing the optimized design.

Algorithm 1 Function Optimization

```
function OPTIMIZEDESIGN(D:design)
    MAPTOTREE(D → T);
    F_0 := FUNCTIONSFROM(D); U_p := ∅;                    ▷ U_p = Used pairs
    while (F := (x ∈ F_0, y ∈ F_0) : x ≠ y, (x,y) ∉ U_p)) ≠ ∅ do
        (f_0, f_1) := BESTPAIR(F); U_p := U_p ∪ (f_0, f_1);
        if (B := XORDECOMP(f_0, f_1)) ≠ FALSE then
            MAPTOTREE(B[P], B[Q_0], B[Q_1] → T);
            REMOVE(f_0, f_1)FROM(F_0);
            F_0 := F_0 ∪ B[P];
        end if
    end while
    return T;
end function
```

The algorithm selects most "compatible" functions f_0 and f_1 using BESTPAIR(), where compatibility of functions is ranked such that the number of shared variables is maximized, and the number of function-exclusive variables is minimized – increasing the probability of producing a useful decomposition. Using f_0 and f_1, the algorithm attempts to find a P, Q_0 and Q_1 decomposition that can replace f_0 and f_1 as a branch in the tree. When a decomposition improves the literal count, the result is mapped into the decomposition tree, the stems of the decomposition (Q_0 and Q_1) are removed from the function pool (F_0), and the root P is added to F_0 for further decomposition. The result of this procedure, applied to all segments of the BCD-to-7-segment display can be seen in Fig.8. Segment outputs S_0, S_1, S_2, S_3 and S_6 are able to benefit from multi-level sharing, where outputs S_4 and S_5 are only able to share functionality with each other and are implemented separately.

The actual XOR decomposition is performed by Algo.2, taking two functions f_0 and f_1 as inputs and producing an improved decomposition, or FALSE if no decomposition could be found. $\theta()$ counts the number of literals in a given set of BDDs. Variable N_0 is a chosen maximum number of passes; we use $N_0 = 130$.

The XOR decomposition technique works by applying cubes to the decomposed functions such that net literal count is reduced. SEED() returns all cubes from $\{Q_0, Q_1, Q_0+Q_1, Q_0 \oplus Q_1\}$, to provide an initial pool of cubes to optimize with. In each iteration, a cube is selected from C and XORed with P, Q_0, and Q_1 to attempt to reduce the net literal count. Cubes from the resulting decomposition are then added to the cube pool C, further increasing the available cubes that can be used. These cubes are repeatedly used until no improvement is found. The technique then tests the result against the best found result, storing it if there is improvement. A new starting point is then chosen to repeat the process. This continues for a chosen number of passes N_0.

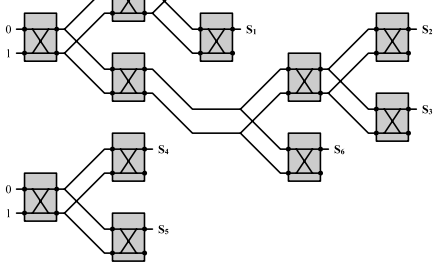


Figure 8: BCD-to-7-Segment complete decomposition.

An important part of our approach is the ability to hill-climb out of local minima. This comes in two forms: the first occurs at lines 16 and 18, and is important for allowing the decomposition to apply cubes to the decomposition even when they cause no literal-count improvement. To prevent deadlocks, this is allowed only for E_0 number of times ($E_0 = 10$ in our implementation). This gives the technique more flexibility in finding a better decomposition. The second method allows a restart at a point based on the best decomposition and a cube which caused the largest effect (line 22) on the decomposition. The process then repeats and continues until there are no more passes left.

Algorithm 2 XOR Decomposition

```
function XORDECOMP(f_0, f_1)
    P := 0; Q_0 := f_0; Q_1 := f_1; L_0 := θ({P, Q_0, Q_1});
    best := [P, Q_0, Q_1, L_0];                          ▷ Current best results
    N := N_0;                                  ▷ N = Passes left; N_0 = total passes
    V[] := ∅;                                  ▷ V[e] maps e → SetofCubes;
    U_v := ∅                                           ▷ U_v = used V cubes
    while (N > 0) do
        C := SEED({P, Q_0, Q_1});                              ▷ C = cubes;
        U_c := ∅;                                          ▷ U_c = used cubes
        L_1 := θ({P, Q_0, Q_1});                  ▷ Starting # literals for pass
        L := L_1; E := E_0;
        repeat
            m := REMOVECUBEFROM(C); U_c := U_c ∪ m;
            p := P ⊕ m; q_0 := Q_0 ⊕ m; q_1 := Q_1 ⊕ m;
            v := θ({p, q_0, q_1}) − θ({P, Q_0, Q_1});
            if (v < 0) or ((v = 0) and (E > 0)) then
                P := p; Q_0 := q_0; Q_1 := q_1;                ▷ Accept the change
                E := if (v = 0) then E − 1 else E_0;
                L := L + v;
            end if
            C := C ∪ (CUBESOF({p, q_0, q_1}) \ U_c);
            e := |θ(p) − θ(P)| + |θ(q_0) − θ(Q_0)| + |θ(q_1) − θ(Q_1)|;
            if m ∉ CUBESOF(V) then
                V[e] := V[e] ∪ m;                  ▷ Map cube's effect e to cube
            end if
            if (C = ∅) and (L ≠ L_1) then
                C := U_c; U_c := ∅; L_1 := L;                  ▷ Retry until no change
            end if
        until (C = ∅);
        if L_1 < best[L_0] then
            best := [P, Q_0, Q_1, L_0];
            N := N + 1;                ▷ Reward improvement with extra passes
        else
            N := N − 1;
        end if
        m_0 := (c ∈ V[e] : largest(e), c ∉ U_v);
        P := best[P] ⊕ m_0;   Q_0 := best[Q_0] ⊕ m_0;   Q_1 := best[Q_1] ⊕ m_0;
    end while
    if best[L_0] < L_0 then return best else return FALSE;
end function
```

After a complete decomposition is performed for a design, the sub-functions of the decomposition tree are implemented as optimized factored-forms and mapped to VGs. The final decomposed multi-output design is implemented as a tree of XOR-decomposed functions, in the same type of structure as seen in Fig.8. We evaluate this technique's efficacy on a number of logic designs in the next section.

5. EXPERIMENTAL RESULTS

The crossbar logic synthesis technique described in Sect.4 is applied to a number of logic designs from the ACM/SIGDA (i.e.

Design	In	Out	L_{orig}	L_{decomp}	ΔL	# funcs
5xp1	7	10	294	160	**-134**	15
alu2	10	6	25645	899	**-24746**	9
alu4	14	8	6227	4906	**-1321**	13
apex4	9	18	15967	4154	**-11813**	32
b1	3	4	16	9	**-7**	3
b12	15	9	1847	146	**-1701**	13
bcd7seg	4	7	132	35	**-97**	11
bw	5	28	955	314	**-641**	43
c8	28	18	200	406	+206	27
cc	21	20	147	136	**-11**	27
clip	9	5	888	736	**-152**	9
cm162a	14	5	85	125	+40	9
cm163a	16	5	43	65	+22	8
cmb	16	4	76	48	**-28**	4
cps	24	109	7156	5332	**-1824**	152
cu	14	11	91	71	**-20**	11
decod	5	16	80	65	**-15**	16
duke2	22	29	2174	2220	+46	43
ex1010	10	10	86694	5433	**-81261**	19
ex5p	8	63	60960	902	**-60058**	79
f51m	8	8	317	109	**-208**	11
i1	25	16	82	88	+6	17
inc	7	9	744	176	**-568**	14
lal	26	19	184	196	+12	25

Design	In	Out	L_{orig}	L_{decomp}	ΔL	# funcs
ldd	9	19	427	141	**-286**	25
misex1	8	7	122	93	**-29**	12
misex2	25	18	188	175	**-13**	24
misex3	14	14	17971	13232	**-4739**	25
misex3c	14	14	5006	6892	+1886	27
pcle	19	9	87	131	+44	14
pcler8	27	17	199	420	+221	22
pdc	16	40	208008	41269	**-166739**	79
pm1	16	13	67	72	+5	16
rd53	5	3	144	74	**-70**	5
rd73	7	3	840	249	**-591**	5
rd84	8	4	3288	465	**-2823**	6
sao2	10	4	532	250	**-282**	7
sct	19	15	141	265	+124	22
spla	16	46	141815	3372	**-138443**	69
sqrt8	8	4	155	120	**-35**	6
sqrt8ml	8	4	1382	44	**-1338**	7
squar5	5	8	387	70	**-317**	11
table3	14	14	7021	4446	**-2575**	25
tcon	17	16	48	48	0	24
ttt2	24	21	337	544	+207	31
x2	10	7	87	132	+45	9
z4ml	7	4	62	114	+52	6

Table 1: Benchmark Results (L = # literals)

MCNC) logic synthesis benchmark suites [26]. We also include the BCD-to-7-segment design. Two designs (cm138a and cm42a) saw no change via our technique and are not included in the table.

The results of the technique's application is seen in Tbl.1. The original literal count L_{orig} represents the number of literals counted for implementing all outputs separately as VGs. The decomposed literal count L_{decomp} represents the number of literals of the decomposed network after applying our technique. Also included is the number of sub-functions (**#funcs**) implemented as VGs in the decomposed implementations.

Overall, most designs enjoy reduced literal counts when the decomposition is applied (negative ΔL), in some cases orders of magnitude differences. An increase or no change in literal counts for some designs can be attributed to discrepancies between the literal counts of the BDD-functions used in the technique's internal metrics, and the actual implementations of those functions as VGs.

Limitations: Our synthesis procedure does not allow for electro-optical interfaces (receivers and transmitters) except at the start and endpoints of the circuit. However, in larger systems, these functional blocks comprise parts of the overall design that must be interconnected. A more extensive synthesis procedure is needed to partition larger circuits at electro-optical transceiver boundaries, with individual blocks implemented via synthesis techniques such as presented in this paper.

6. CONCLUSION AND FUTURE WORK

This paper describes design and synthesis methods for implementing digital logic using integrated optical devices that function as crossbar devices. We have shown a design methodology for constructing arbitrary logic functions using VGs, and present an XOR-based methodology for expression sharing for multi-output designs. The efficacy of our synthesis techniques is shown on a number of logic designs, often with large improvements.

Future Work: This synthesis procedure is limited to implementations that do not incorporate electro-optical transceivers. As part of a more extensive synthesis procedure, we are exploring partitioning techniques to enable larger designs to be implemented as a series of interconnected sub-functions designed using techniques such as presented in this paper. However, the physical design of the optical network is an integral part of such partitioning. Parameters such as signal degradation from splitters, routing congestion, and delay balance will ultimately decide how circuits are partitioned for separate implementation. Therefore, we are exploring ways to integrate this technique with automated layout and routing using the same "building block" concept used for synthesis.

7. REFERENCES

[1] W. Haensch et al., "Silicon CMOS devices beyond scaling", *IBM J. Res. Dev.*, vol. 50, pp. 339–361, 2006.

[2] R. Beausoleil et al., "A Nanophotonic Interconnect for High-Performance Many-Core Computation", *Symposium on High-Performance Interconnects*, pp. 182–189, 2008.

[3] L. Liao et al., "High speed silicon Mach-Zehnder modulator", *Optics Express*, vol. 13, pp. 3129–3135, 2005.

[4] W. Green et al., "Ultra-compact, low RF power, 10 Gb/s silicon Mach-Zehnder modulator", *Optics Express*, vol. 15, pp. 17106–17113, 2007.

[5] H. Rong, R. Jones, A. Liu, O. Cohen, D. Hak, A. Fang, and M. Paniccia, "A continuous-wave Raman silicon laser", *Nature*, vol. 433, pp. 725–728, 2005.

[6] P. Ganapati, "Germanium Laser Breakthrough Brings Optical Computing Closer", *Wired Magazine*, Feb 2010.

[7] K. Okamoto, *Fundamentals of Optical Waveguides*, Academic Press, 2000.

[8] S. J. Koester et al., "Ge-on-SOI-Detector/Si-CMOS-Amplifier Receivers for High-Performance Optical-Communication Applications", *J. Lightwave Technol.*, vol. 25, pp. 46–57, 2007.

[9] S. Emelett and R. Soref, "Analysis of dual-microring-resonator cross-connect switches and modulators", *Optics Express*, vol. 13, pp. 7840–7853, 2005.

[10] S. J. Emelett and R. Soref, "Design and Simulation of Silicon Microring Optical Routing Switches", *Journal of Lightwave Technology*, vol. 23, pp. 1800, 2005.

[11] A. J. Poustie and K. J. Blow, "Demonstration of an all-optical Fredkin gate", *Optics Communications*, vol. 174, pp. 317 – 320, 2000.

[12] J. Shamir, H. J. Caulfield, W. Micelli, and R. J. Seymour, "Optical computing and the Fredkin gates", *Appl. Opt.*, vol. 25, pp. 1604–1607, 1986.

[13] J. Hardy and J. Shamir, "Optics Inspired Logic Architecture", *Optics Express*, vol. 15, pp. 150–165, 2007.

[14] H. J. Caulfield, Vikram C. S., and Zavalin A., "Optical logic redux", *Optik*, vol. 117, pp. 199–209, 2006.

[15] H. J. Caulfield et al., "Generalized optical logic elements GOLEs", *Optics Communications*, vol. 271, pp. 365–376, mar 2007.

[16] D. Ding, Y. Zhang, H. Huang, R. T. Chen, and D. Z. Pan, "O-Router: an optical routing framework for low power on-chip silicon nano-photonic integration", *in Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pp. 264–269, New York, NY, USA, 2009. ACM.

[17] L. Pavesi and D. Lockwood, *Silicon Photonics (Topics in Applied Physics)*, Springer, 2004.

[18] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, August 1986.

[19] C. Condrat, P. Kalla, and S. Blair, "Exploring Design and Synthesis for Optical Digital Logic", *International Workshop on Logic Synthesis*, 2010.

[20] A. Mishchenko and M. Perkowski, "Fast Heuristic Minimization of Exclusive-Sums-of-Products", *in Proc. RM'2001 Workshop*, 2001.

[21] T. Sasao, "An Exact Minimization of AND-EXOR Expressions Using BDDs", *in Proc. IFIP 10.5 Workshop Applications of the Reed-Muller Expansion in Circuit Design*, Sept 1993.

[22] C. Yang, M. Ciesielski, and V. Singhal, "BDS: A BDD-Based Logic Optimization System", *in Proc. of DAC 2000*, pp. 92–97, 2000.

[23] T. S. Czajkowski and S. D. Brown, "Functionally linear decomposition and synthesis of logic circuits for FPGAs", *in Proceedings of the 45th annual Design Automation Conference*, DAC '08, pp. 18–23, New York, NY, USA, 2008. ACM.

[24] P. Gupta, A. Agrawal, and N. K. Jha, "An Algorithm for Synthesis of Reversible Logic Circuits.", *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, pp. 2317–2330, 2006.

[25] R. Wille and R. Drechsler, "BDD-based Synthesis of Reversible Logic for Large Functions", *in Design Automation Conf.*, 2009.

[26] ACM/SIGDA, "ACM/SIGDA benchmarks", `http://www.cbl.ncsu.edu/benchmarks/Benchmarks-upto-1996.html`.