A Multiple-Valued Logic Synthesis Tool for Optical Computing Elements

Kaitlin N. Smith, Mitchell A. Thornton
Darwin Deason Institute for Cyber Security
Department of Electrical Engineering
Southern Methodist University
Dallas, TX, USA
knsmith@smu.edu, mitch@smu.edu

***Abstract*—Optical computing elements offer benefits over traditional CMOS-based electronic logic gates such as increased performance and reduced power. Using polarization to encode the information to be processed allows for the possibility of non-binary switching theory to be applied that further offers the benefit of reducing the number of required elements in an optical computing circuit. A methodology for synthesizing non-binary optical computing circuits is described and experimental results are provided that justify the approach.**

Keywords—Optical Computing, Multiple-Valued Logic, Synthesis

I. Introduction

Multiple-valued logic, or MVL, has the potential to revolutionize modern computing. Rather than switching between two states as in binary, a higher resolution of outcomes can be achieved with the addition of logic levels. MVL expressions are capable of modeling more realistic systems that need more state representation than just a "true" or "false." Additionally, higher radix systems have the advantage of faster and more efficient computations since fewer gates are required and interconnecting busses hold more data [1]. Many other advantages of higher-radix computation circuitry are given in [3].

Currently, metal-oxide-semiconductor field effect transistors, MOSFETs, are used in most integrated circuits, ICs, to support Boolean logic. These transistors are well understood and easy to manufacture, making them the preferred choice when computing only requires either an asserted or non-asserted state to be present. What happens, however, when MVL is introduced? Traditional MOSFETs only support two logic states and would need complex modifications to support a radix higher than 2. Because of the difficulty of implementing MVL with current IC technology, a possible solution could be exploring alternative computing methods. Of these alternative methods, the use of photonic devices for MVL seems to be the best choice.

Photonic devices such as semiconductor lasers are commonly found in data reading sensors and communication networks. Lasers have widespread use in digital communications because they provide many advantages such as small size, low power consumption, and the ability to be fabricated on a mass scale [2]. Additionally, data travels much faster since photons travel at a quicker rate than electrons. For this reason, optical computers have the potential to compute at a speed 107 times faster than the fastest conventional electric computer [7]. These advantages could also make optical computing devices a suitable platform for MVL. Unlike the electrons in a transistor, photons have the ability to travel in distinct orientations, or polarizations. Since the polarization of light can be detected, measured, and manipulated, it is possible that multi-state systems could be represented with semiconductor photonic devices.

In this paper, the development of a MVL synthesis tool will be discussed. The final product is a program capable of reading a logic table with a radix of three or above and outputting a netlist with a Verilog-like syntax that represents the logic expression of the table data. Eventually, this netlist will map to photonic elements capable of producing a gate representation that mirrors the original logic table.

II. Background

Representing MVL in gate form requires logic blocks that build on the concepts of traditional Boolean operators. To represent a multi-state system, combinations of MIN, MAX, and literal selection gates may be used [3]. The visual rendering of these MVL gates can be seen in Fig. 1.

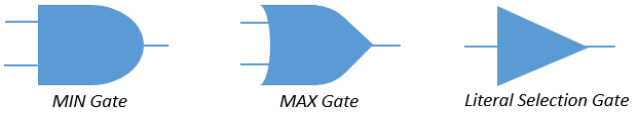


Fig. 1. MIN, MAX, and literal selection gate shapes

A MIN gate, taking resemblance to an AND gate, finds the minimum logic value input to the gate and sets that as output. Conversely, a MAX gate, closely resembling an OR gate, outputs the maximum logic value that is input into the gate's terminals. Lastly, the literal selection gate, with the appearance of a buffer, acts as a value detector. The input to this gate will always be a single variable and the output will be the highest available logic level for a system's radix whenever the desired state is detected. The truth tables describing the three MVL gates utilized in this paper have been included in Fig. 2.

MIN Operation

•	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

MAX Operation

+	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

Literal Select Operation

x	$J_0(x)$	$J_1(x)$	$J_2(x)$
0	2	0	0
1	0	2	0
2	0	0	2

Fig. 2. MIN, MAX, and literal selection truth tables as defined for radix 3

Creating the gate representation of MVL logic tables was done using the concepts of Shannon Expansion. In Boolean logic, Shannon Expansion is an identity used to express a function, F, as the simplified form $F = x'F_{x'} + xF_x$. The terms $F_{x'}$ and F_x in the expression have the x variable equal to 0 and 1, respectively, allowing the x argument to be factored out completely. This process can continue for the remaining variables in the expression so that the original function, F, can be reduced to be in terms of a single argument. The result is a simplified sum of products binary equation.

Shannon Expansion has become a fundamental tool in Boolean algebra, but its application is not limited to a radix of 2. By adding terms to the original identity, Shannon Expansion can be used to express MVL functions of any radix. As an example, the logic function, F, would take the form $F = x^{\{0\}}F_x^{\,0} + x^{\{1\}}F_x^{\,1} + x^{\{2\}}F_x^{\,2}$ if radix 3, logic was used. The terms $F_x^{\,0}$, $F_x^{\,1}$, and $F_x^{\,2}$ in the expression have the x variable equal to 0, 1, and 2, respectively. Just as with Boolean logic, an MVL table can be reduced to an expression that can be represented using logic gates. A two variable, radix 3 system can be seen in Table I.

TABLE I. RADIX 3 SYSTEM

x	*y*	*F*
0	0	0
0	1	1
0	2	2
1	0	0
1	1	1
1	2	2
2	x	2

Application of the Shannon Expansion results in the following equation that represents the table data:

$F = x^{\{0\}}F_x^{\,0} + x^{\{1\}}F_x^{\,1} + x^{\{2\}}F_x^{\,2}$

$F = x^{\{0\}}(y^{\{0\}} \bullet 0 + y^{\{1\}} \bullet 1 + y^{\{2\}} \bullet 2) + x^{\{1\}}(y^{\{0\}} \bullet 0 + y^{\{1\}} \bullet 1 + y^{\{2\}} \bullet 2) + x^{\{2\}}(y^{\{0\}} \bullet 2 + y^{\{1\}} \bullet 2 + y^{\{2\}} \bullet 2)$

$F = x^{\{0\}}(y^{\{1\}} \bullet 1 + y^{\{2\}} \bullet 2) + x^{\{1\}}(y^{\{1\}} \bullet 1 + y^{\{2\}} \bullet 2) + x^{\{2\}}(2)$

$\boldsymbol{F = x^{\{0,1\}}(y^{\{1\}} \bullet 1 + y^{\{2\}} \bullet 2) + x^{\{2\}}(2)}$

With a MVL expression for a system, a gate depiction can also be defined. In order to accomplish a circuit from a logic expression, all multiplication operations become MIN gates whereas all addition operations become MAX gates. The above logic expression can be seen as a circuit in Fig. 3.

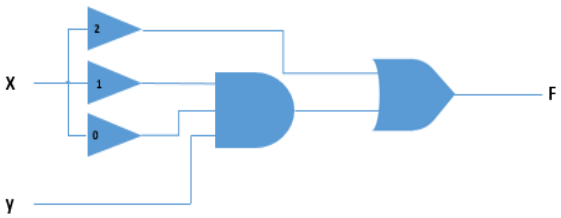


Fig. 3. Gate representation of Table I

III. The Integration of Photonics in MVL

The benefits that accompany photonic devices in computing are only present whenever all of the components from circuit start to finish are optical. In optical systems, if a signal constantly converts between electric to photonic, high levels of energy are lost and complicated interconnects are necessary [4]. These requirements, unfortunately, increase the power consumption and spatial requirements of a circuit, decreasing its overall efficiency.

In order to make photonic devices a reasonable platform for computing, especially for MVL, it is essential for functionally complete set to be developed. Here, we use the functionally complete set {MIN, MAX, and Literal Select}. Different optical MVL gates can be easily added to our library should they be desirable. The only constraint is that the library must contain a functionally complete subset of gates.

Ternary computing allows for one extra logic level as compared to binary. A possibility for an optically based radix 3 computer was presented in the article "Ternary Optical Computer Architecture." In this paper, the possibility of representing 3 logic states, 0, 1, and 2, is discussed using the absence of light, vertically polarized light, and horizontally polarized light, respectively, from a lamp house [5]. If an angle smaller than 90 degrees was used between polarizations and the absence of light represented logic level 0, photonic devices could extend their application to n-valued logics. A possible equation to calculate the angle of separation between polarizations representing different logic levels (with the absence of light included as a logic level) could be $\theta = 180°/(\text{radix-1})$.

Any set of MVL operators could be used during gate synthesis as long as they are functionally complete. The MVL synthesis tool in this paper maps to the functionally complete set of MIN, MAX, and literal selection gates that is popular for MVL computing. Ideally, methods to implement these three gates optically so that the logic operators are dynamic in radix will be developed with further research. In theory, a literal selection gate could be created simply by filtering incoming light for a particular polarization and then rotating the detected light to match the polarization angle of the highest state. Such a task might be accomplished using a filter and a dove prism combined with a half-wave plate since dove prisms invert transmitted images and can rotate images at twice the rotation frequency of the prism [8].

IV. Approach

The MVL synthesis tool was written in *C++*. The purpose of the program is to produce a simplified MVL gate representation by inputting radix, number of variables, and the complete logic table. Shannon Expansion as well as other factoring techniques were implemented in order to accomplish the final netlist. The overall flow diagram for gate synthesis can be seen in Fig. 4.

The netlist synthesis tool begins by reading information about the MVL system from a text file indicated on the command line. The radix and number of variables are indicated by the first two digits of the first line of text, respectively. The MVL logic table begins on the second line of text in the input file. The last column of the table indicates the system output, F, while the proceeding columns hold the

logic levels of the variables. After the logic table has been read, optimizations for the netlist begin with column sort.

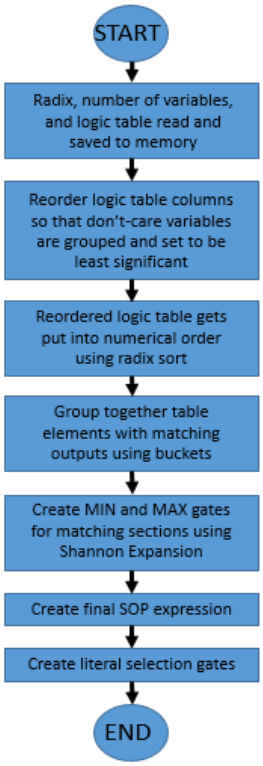
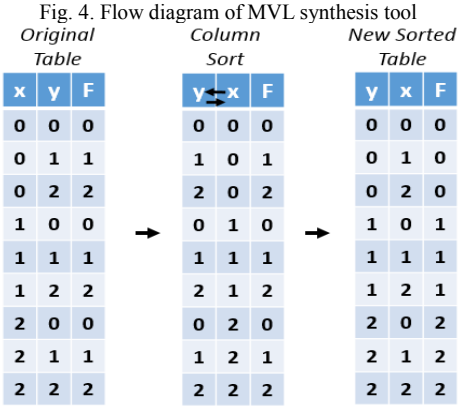


Fig. 4. Flow diagram of MVL synthesis tool



x	y	F
0	0	0
0	1	1
0	2	2
1	0	0
1	1	1
1	2	2
2	0	0
2	1	1
2	2	2

y	x	F
0	0	0
1	0	1
2	0	2
0	1	0
1	1	1
2	1	2
0	2	0
1	2	1
2	2	2

y	x	F
0	0	0
0	1	0
0	2	0
1	0	1
1	1	1
1	2	1
2	0	2
2	1	2
2	2	2

Fig. 5. Illustration of MVL table column sorting with x as a don't-care

The order of the logic table columns is important for the simplification of the final logic expression. Sorting of the MVL table begins by reordering the columns so that don't-care variables, or variables that have minimum to no impact on the output, are set as least significant in the table while variables that are more influential on the output of the system are set as most significant. This is accomplished by comparing each column of the logic table to the output and setting a mismatch score for the variables. The variables that differ most from the output are moved to a less significant column on the table while variables that closely match the output are made more significant. After the columns are reordered, radix sort allows the new MVL table to be put back into numerical order. Fig. 5 helps to explain the importance of reordering of columns during table sorting. As can be seen in Fig. 5, sorting the MVL table columns helps to group matching output terms. This grouping of *F* helps reduce the final logic expression whenever Shannon Expansion is implemented in the code.

After sorting is complete and a new MVL table is achieved, the creation of the netlist officially begins. First, Shannon Expansion applied on the table and MIN and MAX gates appear with the syntax of min(output, input1, input2….) and max(output, input1, input2…..), respectively. Either wires or constants can be inputs to both MIN and MAX gates while the outputs will always be wires. Next, a MAX gate that creates the sum of products expression is generated. The input to the final MAX gate will be wires while the output will be F. Finally, the literal selection gates needed for the circuit are printed in the form of lit#(output, input) where the # represents the detected state. Only variables will be input to these gates while the output will always be wires.

As each gate for the netlist is created, it prints to the screen. The result of each run is also saved into a text file titled netlist_year_month_day_hour_min_sec.txt. The runtime for the netlist creator program is also located in this test file.

V. Experimental Results

During testing, it was found that the MVL netlist synthesis tool was capable of creating gate representations of systems with any number of variables and with a radix greater than two. Tests were completed on a portable laptop running Windows with an i3-4010U CPU at 1.7 GHz and 4 GB of RAM. Run times listed in this section with the output netlists are specific for this machine.

Because it is the lowest radix in MVL, ternary logic was often used during initial testing. Table II describes a two-variable, ternary system that was used as a program input.

TABLE II. SAMPLE SYSTEM USED FOR TESTING

Variable 0	*Variable 1*	*F*
0	0	0
0	1	0
0	2	0
1	0	1
1	1	1
1	2	1
2	0	0
2	1	0
2	2	2

Whenever the MVL synthesis tool was executed on the data above, a netlist containing 2 MIN gates, 1 MAX gate, and 3 literal selection gates was generated:

```
Generated Netlist:
min(w0, v0_set2, v1_set2)
min(w1, 1, v0_set1)
max(F, w0, w1)
lit1(v0_set1, Variable0)
lit2(v0_set2, Variable0)
lit2(v1_set2, Variable1)
```

This netlist realized in gate form appears as the MVL circuit seen in Fig. 6.

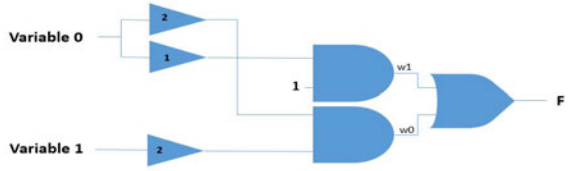


Fig. 6. Gate Representation of Table II

To verify the functionality of the program, binary circuits with known truth tables were converted into 4-valued logic for testing. Table III describes a 2-bit by 2-bit binary adder with no carry input. The radix 2 and radix 4 representation of the system has been included.

TABLE III. 2-bit by 2-bit binary adder

Radix 2		***Radix 4***	
Input (v0,v1,v2,v3)	*Output (F1,F2,F3)*	*Input (v0,v1)*	*Output (F0,F1)*
0000	000	00	00
0001	001	01	01
0010	010	02	02
0011	011	03	03
0100	001	10	01
0101	010	11	02
0110	011	12	03
0111	100	13	10
1000	010	20	02
1001	011	21	03
1010	100	22	10
1011	101	23	11
1100	011	30	03
1101	100	31	10
1110	101	32	11
1111	110	33	12

With the radix 4 representation of the 2-bit by 2-bit adder known, it can be saved into a text file and input to the netlist synthesis tool. Since the radix 4 output has two digits, the netlist synthesis needs to be run twice to create a complete circuit. The first run of the table data when mapped to the F0 column generated a netlist including 3 MIN gates, 3 MAX gates and 6 literal selection gates. Next, the 2-bit by 2-bit binary adder data was mapped to F1 to create a second netlist. This netlist contained 12 MIN gates, 1 MAX gate, and 8 literal selection gates.

To see the 2-bit by 2-bit adder system as a whole, the two netlists corresponding to the outputs F0 and F1 needed to be combined:

```
min(w0, 1, v0_set1, v1_set3)
max(w1, v1_set2, v1_set3)
min(w2, 1, v0_set2, w1)
max(w3, v1_set1, v1_set2, v1_set3)
min(w4, 1, v0_set3, w3)
max(F0, w0, w2, w4)
min(w0a, 1, v0_set0, v1_set1)
min(w1a, 2, v0_set0, v1_set2)
min(w2a, v0_set0, v1_set3)
min(w3a, 1, v0_set1, v1_set0)
min(w4a, 2, v0_set1, v1_set1)
min(w5a, v0_set1, v1_set2)
min(w6a, 2, v0_set2, v1_set0)
min(w7a, v0_set2, v1_set1)
min(w8a, 1, v0_set2, v1_set3)
min(w9a, v0_set3, v1_set0)
min(w10a, 1, v0_set3, v1_set2)
min(w11a, 2, v0_set3, v1_set3)
max(F1, w0a, w1a, w2a, w3a, w4a, w5a, w6a, w7a, w8a, w9a, w10a, w11a)
lit0(v0_set0, Variable0)
lit0(v1_set0, Variable1)
lit1(v0_set1, Variable0)
lit1(v1_set1, Variable1)
lit2(v0_set2, Variable0)
lit2(v1_set2, Variable1)
lit3(v0_set3, Variable0)
lit3(v1_set3, Variable1)
```

The gate representation of the system as a whole features 15 MIN gates and 4 MAX gates. Eight literal selection gates are needed since repetitive literal selection gates are unnecessary and eliminated. A total of 0.655 sec was needed to create the complete netlist for the 2-bit by 2-bit adder.

Table IV includes resulting test data when the MVL netlist generator was used to recreate benchmark circuits.

TABLE IV. BENCHMARK CIRCUIT DATA

Circuit Name	***Radix***	***Size (gates)***	***Inputs***	***Outputs***	***Time (sec)***	***Cost***
Table II system	3	6	2	1	0.189	4.5
2-bit by 2-bit binary adder	4	27	2	2	0.655	34.5
4 to 1 mux	4	38	3	1	0.796	55
XOR5	4	27	3	1	0.656	33

To determine total function cost, each literal selection gate was given the value 0.5 while each MIN and MAX gate was given the value of 1. Each additional gate input above 2 increases the total value of the block by 0.5. Cost is a unitless rate and is used as a rough indicator of how much hardware the optical circuit will require.

VI. Conclusion and Future Results

The MVL synthesis tool described in this paper creates netlists that represent the information found in MVL logic tables. The tool's purpose to be mapped to photonic devices was explained and its functionality was demonstrated through example logic tables and corresponding gate representations. Through the testing process, generated netlists were found to be accurate, but opportunities for further optimization and improvement exist. For example, a calculation that automatically determines the total cost of an MVL circuit could be added to the program so that the user is more aware of the energy requirements of a system. Additionally, further checks could be placed in the program code to insure that the generated netlist appears in its most simplified state. Finally, work will continue to develop a set of verified photonic gates that this tool will map to.

References

[1] V.T Gaikwad, P.R. Deshmukh, "Multi-Valued Logic Applications in the Design of Switching Circuits," *Int. Journal of Advanced Research in Computer Science and Software Engineering,* vol. 2, no. 5, pp. 446-449. May 2015.

[2] A. Yariz, P.Yeh, "Semiconductor Lasers - Theory and Application," in *Photonics*, 6th ed. New York: The Oxford University Press Inc., 2007, ch. 15, pp. 673-713.

[3] D. M. Miller, M. A. Thornton, "MVL Concepts and Algebra," in *Multiple Valued Logic Concepts and Representations.* Morgan & Claypool Publishers, 2007.

[4] P. Singh, D. K. Tripathi, S. Jaiswal, H. K. Dixit, "All-Optical Logic Gates: Designs, Classification, and Comparison," *Advances in Optical Technologies*, vol. 2014, Article ID 275083, 13 pages, 2014.

[5] J. Yi, H. Huacan, L. Yangtian. "Ternary Optical Computer Architecture," *Physica Scripta,* vol. T118, pp. 98-101. 2005

[6] G. D. Jenkins. " All- Optical Logic Gates." 2007

[7] A. Gupta, A. Rai, S. Kumari. "A Review on the Reality and Promises of Optical Computing." *International Conference on Computer Science and Information Technology*. 2013, pp. 106-110

[8] M. J. Padgett, J. P. Lesso. "Dove prisms and polarized light," *Journal of Modern Optics,* vol. 46, no. 2, pp. 175-179. 1999