RM_3 based logic synthesis

(Special session paper)

Mathias Soeken[1] Pierre-Emmanuel Gaillardon[2] Giovanni De Micheli[1]
[1]Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
[2]University of Utah, Salt Lake City, UT, USA

***Abstract*—In-memory computing devices, such as resistive RAMs, natively implement material implication or a variant of the majority-of-three operation called RM_3. This operation generalizes material implication and has been used as target operation in several logic synthesis algorithms for in-memory computing applications. In this work, we investigate a homogeneous logic network data structure that uses RM_3 as only logic operation. Such a data structure makes an ideal fit for the use in design automation algorithms for in-memory computing. We show how to derive RM_3 networks from well-known logic synthesis data structures and a technique how to obtain such networks using technology mapping.**

***Index Terms*—logic synthesis, in-memory computing, majority logic**

I. Introduction

Logic synthesis is an essential tool in the design of emerging technologies. It is inevitable for the task of finding efficient realizations once the basic logic primitives of the technology are known. But also before, in the phase of developing a new technology, logic synthesis can guide how primitives are modeled in order to obtain the best cost trade-offs [1].

We are investigating logic synthesis for memristive circuits. Memristive circuits have opened up the promising research of in-memory computing. As one example, resistive RAMs (RRAMs) allow for both storing data and performing primitive Boolean operations. These Boolean operations of course depend on the RRAM's inherent functionality. It has been shown that RRAMs can natively perform material implication $x \to y$ (see, e.g., [2], [3], [4]). Since material implication is universal, one can realize all Boolean functions using a sequence of in-memory computations and several works for automatic synthesis of Boolean functions into such sequences have been proposed (see, e.g., [5], [6], [7], [8], [9]).

More recently, it has further been shown that the logic capabilities of RRAM devices go beyond material implication with an expressive implementation of the majority-like operation $\langle x\bar{y}z\rangle = x\bar{y} \lor xz \lor \bar{y}z = (x \lor \bar{y})(x \lor z)(\bar{y} \lor z)$, which is true, whenever at least two of the values $\{x, \bar{y}, z\}$ are true. [10].[1] The operation has been coined RM_3, and it is in fact a generalization of material implication, since $\langle y\bar{x}1\rangle = \bar{x} \lor y = x \to y$. Of course, RM_3 is therefore also universal. Several approaches have been presented to obtain a sequence of in-memory computations based on RM3 for any combinational Boolean function (see, e.g., [10], [11], [12], [13]).

[1] $\langle xyz\rangle$ means majority-of-three and is true, if and only if at least two operands are true.

Current logic synthesis approaches for in-memory computing, both based on material implication and RM_3, make use of existing general purpose logic data representations such as binary decision diagrams [14], [15], And-inverter graphs [16], implication networks [5], [7], [8], and majority-inverter graphs [11], [12].

In this paper, we investigate a dedicated homogeneous network structure, called RM_3 logic network, which has the RM_3 operation as its only logic operation. The paper is explicitly focusing on logic synthesis based on these primitives and is motivated by several technological findings in the recent past. For more details on the technology, the reader is referred to [2], [3], [4], [10]. We show how RM_3 networks can be derived from other logic representations and some rewriting operations that allow optimizing RM_3 networks for both size and logic depth. Since RM_3 networks are a natural representation of sequences for in-memory computations, the size and depth provide for a meaningful cost metric without the need of a technology mapping step.

II. Preliminaries

A Boolean *logic network* is a simple digraph whose vertices are constants, primary inputs, primary outputs, and gates and whose arcs connect gates to inputs, outputs, and other gates. Each gate in the logic network realizes a Boolean function. Also some networks allow an arc to be complemented instead of having a gate that realizes an inverter. If each gate can realize an arbitrary Boolean function with up to k inputs, the network is called k-feasible network or k-LUT network. LUT means lookup-table and LUT mapping (see, e.g., [17], [18], [19], [20]) refers to a family of algorithms that obtain k-feasible networks. If each gate realizes the same Boolean function, the network is referred to as homogeneous logic network. A larger flexibility can be achieved by using complemented arcs and constant inputs. Frequently used homogeneous logic representations are And-inverter graphs (AIGs, [21], [22]) or Majority-inverter graphs (MIGs, [23]). These are logic networks with complemented arcs in which all gates realize the AND or majority-of-three function, respectively. In this paper, the proposed RM_3 networks are homogeneous logic networks with constant inputs but without complemented arcs in which each node realizes the operation $\langle x\bar{y}z\rangle$.

III. Synthesis from Logic Networks

In this section, we discuss how one can obtain RM_3 networks from well-known network data structures used in logic synthesis.

A. Synthesis from Implication Networks

Implication networks are logic networks which have the material implication as single logic operation. They have been extensively used in the design of implication-based memristor circuits (see, e.g., [5], [6], [7], [8], [9]). With the help of a constant 0 value, material implication is universal, as

$$\overline{x \land y} = x \to (y \to 0). \tag{1}$$

We can similarly realize NOT, AND, OR, and NOR [5]:

$$\begin{aligned} \bar{x} &= x \to 0 & x \land y &= (x \to (y \to 0)) \to 0 \\ x \lor y &= (x \to 0) \to y & \overline{x \lor y} &= ((x \to 0) \to y) \to 0 \end{aligned} \tag{2}$$

Synthesis from implication networks is straightforward since they only contain a single operation $x \to y$, which is trivially contained in RM_3 logic, since

$$x \to y = \bar{x} \lor y = \langle y\bar{x}1\rangle. \tag{3}$$

However, RM_3 can use a third non-constant input and is therefore functionally more powerful compared to material implication. Indeed, it can be used to realize AND using a single operation instead of three:

$$x \land y = \langle x\bar{1}y\rangle \tag{4}$$

All operations in (2) can be expressed using at most two RM_3 operations, which is detailed in the next section.

We have not performed any experiment for the translation of implication networks into RM_3 networks, as the resulting networks will have both the same size and logic depth.

B. Synthesis from And-inverter Graphs

And-inverter graphs (AIGs, [21], [22]) are one of the most commonly used homogeneous network data structures in today's industrial and academic logic synthesis tools. AIGs contain implication networks since $x \to y = \overline{x \land \bar{y}}$. When translating an AIG into an RM_3 network, one can encounter eight different configurations depending on whether the input edges and output edge are regular or inverted. For six of these configurations, a single RM_3 operations suffices, while two configurations require two RM_3 operations:

$$\begin{aligned} x \land y &= \langle x\bar{1}y\rangle & \bar{x} \land y &= \langle 0\bar{x}y\rangle \\ x \land \bar{y} &= \langle 0\bar{y}x\rangle & \bar{x} \land \bar{y} &= \langle 0\overline{\langle x\bar{0}y\rangle}1\rangle \\ \overline{x \land y} &= \langle 0\overline{\langle x\bar{1}y\rangle}1\rangle & \overline{\bar{x} \land y} &= \langle 1\bar{y}x\rangle \\ \overline{x \land \bar{y}} &= \langle 1\bar{x}y\rangle & \overline{\bar{x} \land \bar{y}} &= \langle x\bar{0}y\rangle \end{aligned} \tag{5}$$

We have performed an experiment to investigate the increase of size and logic depth in the RM_3 networks compared to AIGs. Benchmarks for this experiment, and also for all other experiments in this paper, are the non-optimized AIGs of the arithmetic EPFL benchmarks.[2] Table I lists the results. The first column shows the name of the benchmark. The second and third column give the size and logic depth of the AIG, while the last two columns give the size and logic depth of the RM_3 network. As can be seen, the size may increase up to 28% in the case of *Max* and the logic depth may increase up to 36% in the case of *Square-root*.

TABLE I
OBTAINING RM_3 NETWORKS FROM AND-INVERTER GRAPHS

Benchmark	AIG		RM_3	
	size	depth	size	depth
Adder	1020	255	1275	256
Barrel shifter	3336	12	3470	15
Divisor	57247	4372	68040	4635
Hypotenuse	214335	24801	243697	25120
Log2	32060	444	39387	531
Max	2865	287	3668	328
Multiplier	27062	274	33070	293
Sine	5416	225	6335	250
Square-root	24618	5058	31310	6890
Square	18484	250	22318	253

C. Synthesis from Majority-inverter Graphs

Majority logic has been studied since the 1960s [24], [25], [26], and has recently obtained much attention in the logic synthesis community. In the last few years, it has been used in scalable logic synthesis flows to find optimized networks for Boolean functions. Majority logic owes its interest to many emerging nanotechnologies that implement majority as their primitive building block [1]. Recently, Majority-inverter graphs (MIGs, [23]) have been proposed which exploit the algebraic properties of majority logic. MIGs use majority-of-three operations and inverters as the only logic primitives [1], [23]. MIGs use complemented edges to represent inverters. MIGs contain AIGs, since $x \land y = \langle 0xy\rangle$.

Translating a MIG into an RM_3 network is straightforward if we make use of the inverter propagation axiom $\langle\bar{x}\bar{y}\bar{z}\rangle = \overline{\langle xyz\rangle}$ that allows us to have at most one inverted input to each majority operation without increasing the number of total operations. The following configurations consider the case in which all inputs are non-constant:

$$\begin{aligned} \langle\bar{x}yz\rangle &= \langle y\bar{x}z\rangle & \langle x\bar{y}z\rangle &= \langle x\bar{y}z\rangle \\ \langle xy\bar{z}\rangle &= \langle x\bar{z}y\rangle & \langle xyz\rangle &= \langle x\overline{\langle 0\bar{y}1\rangle}z\rangle \\ \overline{\langle\bar{x}yz\rangle} &= \langle x\bar{y}\langle 0\bar{z}1\rangle\rangle & \overline{\langle x\bar{y}z\rangle} &= \langle y\bar{x}\langle 0\bar{z}1\rangle\rangle \\ \overline{\langle xy\bar{z}\rangle} &= \langle\langle 0\bar{x}1\rangle\bar{y}z\rangle & \overline{\langle xyz\rangle} &= \langle\langle 0\bar{x}1\rangle\bar{y}\langle 0\bar{z}1\rangle\rangle \end{aligned} \tag{6}$$

For three out of the eight configurations, a single RM_3 operation suffices, four require two RM_3 operations and one configuration cannot be done with less than three RM_3 operations. If one operand is constant, the rules as in (5) apply.

Note that we can express $\langle xyz\rangle$ also as

$$\langle xyz\rangle = \langle x\overline{\langle x\bar{y}z\rangle}z\rangle \tag{7}$$

which follows from (6) by applying the relevance rule which has been proposed in [27]. Finally, to realize the XOR of 3 variables, we need at least three RM_3 operations:

$$x \oplus y \oplus z = \langle\langle y\bar{x}z\rangle\bar{y}\langle x\bar{z}y\rangle\rangle \tag{8}$$

[2] lsi.epfl.ch/benchmarks

TABLE II
OBTAINING RM_3 NETWORKS FROM MAJORITY-INVERTER GRAPHS

Benchmark	MIG		RM_3	
	size	depth	size	depth
Adder	386	129	513	129
Barrel shifter	3110	14	3440	14
Divisor	57272	4401	67720	4565
Hypotenuse	153311	9320	184254	9489
Log2	25040	230	30201	274
Max	2491	290	2783	311
Multiplier	19844	143	25499	158
Sine	4496	167	5231	192
Square-root	21066	5989	23377	6080
Square	13671	156	17458	176

One obtains XOR of two by, e.g., setting y to 0; a realization with less than three RM_3 operations is not possible.

We performed a similar experiment as for the And-inverter graphs to compare MIGs to RM_3 networks after translating them using (6). Starting points are MIGs obtained using LUT-based resynthesis [28]. Table II depicts the results. The size may increase up to 33% in the case of *Adder* and the logic depth may increase up to 19% in the case of *Log2*.

D. Post-optmization Rewriting Rules

One striking advantage of using MIGs for logic synthesis is that they define a complete axiomatic system based in five rules [27]. In special cases these rules also hold for the RM_3 operation.

$$\langle x\bar{z}x\rangle = x, \langle x\bar{x}z\rangle = z \tag{9}$$

$$\langle x\bar{y}z\rangle = \langle x\bar{y}x\rangle \tag{10}$$

$$\langle x\bar{u}\langle y\bar{u}z\rangle\rangle = \langle z\bar{u}\langle y\bar{u}x\rangle\rangle \tag{11}$$

$$\langle x\bar{u}\langle y\bar{v}z\rangle\rangle = \langle\langle x\bar{u}y\rangle\bar{v}\langle x\bar{u}z\rangle\rangle \tag{12}$$

The set of Boolean values with the RM_3 operation are not a median algebra since RM_3 commutes only on its uncomplemented arguments (see (10)). Also, note that the inverter propagation axiom does not hold for RM_3.

However, the rules can be used to optimize an RM_3 network, e.g., after it has been obtained from an AIG or an MIG. The majority rule (9) can be used to reduce both size and logic depth. Associativity (11) can be used to reduce logic depth, and distributivity (12) for either reducing size or logic depth, depending on which direction it is applied.

IV. SYNTHESIS USING TECHNOLOGY MAPPING

One can use technology mapping to obtain an RM_3 network from a given technology-independent logic network, e.g., an And-inverter graph. For this purpose, one needs a cell library that contains only the functions that can be realized using a single RM_3 operation. All these functions are listed in Table III. There are three 3-input functions depending on which of the operands is negated. Then, there are six 2-input functions, which are the ones from (5) that require a single RM_3 operation. Also, both 1-input functions, identity and inversion, can be realized using a single RM_3 operation and are therefore part of the cell library. A similar approach has been presented in [29] for the synthesis of implication networks.

TABLE III
ALL FUNCTION PRIMITIVES THAT APPEAR IN RM_3 NETWORKS

Function	Truth table	Number of inputs
$\langle x\bar{y}z\rangle$, $\langle z\bar{y}x\rangle$	1011 0010	3
$\langle y\bar{x}z\rangle$, $\langle z\bar{x}y\rangle$	1101 0100	3
$\langle x\bar{z}y\rangle$, $\langle y\bar{z}x\rangle$	1000 1110	3
$\langle 0\bar{x}y\rangle$, $\langle y\bar{x}0\rangle$	0100	2
$\langle 0\bar{y}x\rangle$, $\langle x\bar{y}0\rangle$	0010	2
$\langle 1\bar{x}y\rangle$, $\langle y\bar{x}1\rangle$	1101	2
$\langle 1\bar{y}x\rangle$, $\langle x\bar{y}1\rangle$	1011	2
$\langle x\bar{0}y\rangle$, $\langle y\bar{0}x\rangle$	1110	2
$\langle x\bar{1}y\rangle$, $\langle y\bar{1}x\rangle$	1000	2
$\langle 0\bar{x}1\rangle$, $\langle 1\bar{x}0\rangle$	01	1
$\langle 0\bar{0}x\rangle$, $\langle x\bar{0}0\rangle$, $\langle 1\bar{1}x\rangle$, $\langle x\bar{1}1\rangle$	10	1

TABLE IV
OBTAINING RM_3 NETWORKS FROM TECHNOLOGY MAPPING

Benchmark	AIG		RM_3				
	size	depth	1 input	2 inputs	3 inputs	size	depth
Adder	1020	255	127	766	127	1020	129
Barrel shifter	3336	12	6	3336	0	3342	12
Divisor	57247	4372	219	57235	6	57460	4369
Hypotenuse	214335	24801	4852	188709	13812	207373	9115
Log2	32060	444	297	28804	1222	30323	280
Max	2865	287	70	2365	138	2573	212
Multiplier	27062	274	402	23641	1176	25219	158
Sine	5416	225	131	5068	171	5370	184
Square-root	24618	5058	229	24622	1	24852	5058
Square	18484	250	405	16914	788	18107	132

We tested this approach for the EPFL arithmetic benchmarks. Starting point are the non-optimized AIGs (as in Section III-B). We use ABC [30] to read the AIGs and map them using the command '*map*'. Table IV lists the results of the experiment. The first three columns list the name of the benchmark, its initial AIG size and logic depth. The next four columns list the size after mapping, where column 'k *inputs*' refers to the number of gates that realize a k-input function (cf. Table III). It can be seen that technology mapping is able to recover a lot of RM_3 operations with three non-constant operands for some benchmarks. In these cases (e.g., *Hypotenuse*, *Log2*, and *Multiplier*), significant reductions in depth and considerable reductions in size can be obtained. Clearly, this approach works far better than translating AIGs directly into RM_3 networks, as done in Section III-B.

V. CONCLUSIONS

In-memory computing is based on devices that natively implement material implication or RM_3, and RM_3 generalizes material implication. It is a natural consequence to have a dedicated logic network structure tailored for the use of in-memory computing design automation algorithms. We have shown how to obtain RM_3 networks from well-known logic network data structures. A direct translation is disadvantageous, since some elementary logic operations require several RM_3 operations.

Post-synthesis optimization techniques are required—and need further investigation—to obtain better quality results. We have shown that a simple method based on technology mapping is already capable of deriving good initial results in which many RM_3 operations with non-constant operands are recovered. This work can be seen as the starting point to several logic synthesis algorithms for in-memory computing applications. The RM_3 logic representation has been implemented on top of the logic synthesis framework CirKit[3] and is available at *github.com/msoeken/cirkit-addon-plim*.

ACKNOWLEDGMENTS

The authors wish to thank Winston Haaswijk for his support in the experimental evaluation. This research was supported by H2020-ERC-2014-ADG 669354 CyberCare and the Swiss National Science Foundation (200021-169084 MAJesty and and 200021-146600).

REFERENCES

[1] L. G. Amarù, P.-E. Gaillardon, S. Mitra, and G. De Micheli, "New logic synthesis as nanotechnology enabler," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2168–2195, 2015.

[2] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[3] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[4] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Trans. VLSI Syst.*, vol. 22, no. 10, pp. 2054–2066, 2014.

[5] A. Raghuvanshi and M. A. Perkowski, "Logic synthesis and a generalized notation for memristor-realized material implication gates," in *Int'l Conf. on Computer-Aided Design*, 2014, pp. 470–477.

[6] E. Lehtonen, J. H. Poikonen, and M. Laiho, "Two memristors suffice to compute all Boolean functions," *Electronics Letters*, vol. 46, no. 3, pp. 239–240, Feb 2010.

[7] ——, "Implication logic synthesis methods for memristors," in *Int'l Symp. on Circuits and Systems*, 2012, pp. 2441–2444.

[8] J. H. Poikonen, E. Lehtonen, and M. Laiho, "On synthesis of boolean expressions for memristive devices using sequential implication logic," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 1129–1134, 2012.

[9] Y. V. Pershin and M. Di Ventra, "Neuromorphic, digital, and quantum computation with memory circuit elements," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 2071–2080, 2012.

[10] P.-E. Gaillardon, L. G. Amarù, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (PLiM) computer," in *Design, Automation and Test in Europe*, 2016, pp. 427–432.

[11] M. Soeken, S. Shirinzadeh, P.-E. Gaillardon, L. G. Amarù, R. Drechsler, and G. De Micheli, "An MIG-based compiler for programmable logic-in-memory architectures," in *Design Automation Conference*, 2016, pp. 117:1–117:6.

[12] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs," in *Design, Automation and Test in Europe*, 2016, pp. 948–953.

[13] D. Bhattacharjee and A. Chattopadhyay, "Delay-optimal technology mapping for in-memory computing using ReRAM devices," in *Int'l Conf. on Computer-Aided Design*, 2016, p. 119.

[14] S. Chakraborti, P. V. Chowdhary, K. Datta, and I. Sengupta, "BDD based synthesis of Boolean functions using memristors," in *Int'l Design and Test Symp.*, 2014, pp. 136–141.

[15] S. Shirinzadeh, M. Soeken, and R. Drechsler, "Multi-objective BDD optimization for RRAM based circuit design," in *Int'l Symp. on Design and Diagnostics of Electronic Circuits & Systems*, 2016, pp. 46–51.

[16] J. Bürger, C. Teuscher, and M. A. Perkowski, "Digital logic synthesis for memristors," in *Reed-Muller Workshop*, 2013.

[17] J. Cong and Y. Ding, "FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[18] D. Chen and J. Cong, "DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs," in *Int'l Conf. on Computer-Aided Design*, 2004, pp. 752–759.

[19] A. Mishchenko, S. Cho, S. Chatterjee, and R. K. Brayton, "Combinational and sequential mapping with priority cuts," in *Int'l Conf. on Computer-Aided Design*, 2007, pp. 354–361.

[20] S. Ray, A. Mishchenko, N. Eén, R. K. Brayton, S. Jang, and C. Chen, "Mapping into LUT structures," in *Design, Automation and Test in Europe*, 2012, pp. 1579–1584.

[21] L. Hellerman, "A catalog of three-variable Or-invert and And-invert logical circuits," *IEEE Trans. Electronic Computers*, vol. 12, no. 3, pp. 198–223, 1963.

[22] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, 2002.

[23] L. G. Amarù, P. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2016.

[24] S. B. A. Jr., "Synthesis of combinational logic using three-input majority gates," in *Symp. on Switching Circuit Theory and Logical Design*, 1962, pp. 149–157.

[25] R. Lindaman, "A theorem for deriving majority-logic networks within an augmented Boolean algebra," *IRE Trans. Electronic Computers*, vol. 9, no. 3, pp. 338–342, 1960.

[26] H. S. Miller and R. O. Winder, "Majority-logic synthesis by geometric methods," *IRE Trans. Electronic Computers*, vol. 11, no. 1, pp. 89–90, 1962.

[27] L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Design Automation Conference*, 2014, pp. 194:1–194:6.

[28] W. Haaswijk, M. Soeken, L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic optimization," in *Asia and South Pacific Design Automation Conference*, 2017.

[29] F. Lalchhandama, B. G. Sapui, and K. Datta, "An improved approach for the synthesis of Boolean functions using memristor based IMPLY and INVERSE-IMPLY gates," in *Annual Symp. on VLSI*, 2016, pp. 319–324.

[30] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Computer Aided Verification*, 2010, pp. 24–40.

[3] *github.com/msoeken/cirkit*