Inverter Propagation and Fan-out Constraints for Beyond-CMOS Majority-based Technologies

Eleonora Testa*, Odysseas Zografos†‡, Mathias Soeken*, Adrien Vaysset‡,
Mauricio Manfrini‡, Rudy Lauwereins†‡ and Giovanni De Micheli*
*Integrated Systems Laboratory, EPFL, Switzerland
†KU Leuven, Belgium
‡IMEC, Belgium

***Abstract*—Traditional logic synthesis faces challenges of meeting the requirements demanded by the many emerging nanotechnologies that are based on logic models different from standard CMOS. Several emerging nanodevices including *Quantum-dot Cellular Automata* (QCA) and *Spin Torque Majority Gates* (STMG) are based on majority logic. In addition, technology constraints require to restrict the number of fan-outs or impose difficulties in realizing inversions. In this paper, we use a majority-based logic synthesis approach to synthesize inversion-free networks with restricted fan-out. We propose one algorithm that propagates all inversions to the primary inputs and another algorithm that limits the number of fan-outs of each majority gate. These algorithms show significant impact on QCA- and STMG-based circuits. Experimental results demonstrate that the average area-delay-energy product can be improved by 3.1× in QCA-based circuits and from 2.9× to 8.1× for STMG-based circuits.**

I. Introduction

The downscaling of CMOS technologies dictated by Moore's Law [1] will reach its limit in the next decade [2]. This has given rise to a search for alternative devices that extend the semiconductor industry roadmap beyond the CMOS technology [3].

A wide variety of beyond-CMOS devices has been studied in the last decade. They include (i) charged-based components such as *Carbon Nanotubes* (CNT, [4]) and *Quantum-dot Cellular Automata* (QCA, [5]) and (ii) non-charge-based solutions such as *Spin Torque Majority Gate* (STMG, [6]) and *NanoMagnet Logic* (NML, [7]). One main difference compared to CMOS technology is that a large amount of novel alternative devices have different logic abstraction with respect to standard transistors. In particular, the device model for QCA, STMG, and NML technologies is the majority voter. The variety of beyond-CMOS devices further leads to a broad range of various technological constraints. Two main drawbacks apply to several devices. First, since all devices are targeted towards ultra-low energy operation, the inherent amplification or the driving capabilities of these devices are low [8]. This leads to the need of constraining the fan-out characteristics of the implemented circuits. Second, several beyond-CMOS technologies do not offer efficient implementations of *inverters* (INVs) [9], [10]. Therefore, it is required to minimize their application [11] or even to eliminate them from implemented circuits. Logic synthesis needs to consider all these characteristics.

In this work, we synthesize and optimize circuits by making use of a majority-based data-structure, called *Majority Inverter Graphs* (MIG, [12]). In particular, we introduce two algorithmic techniques to rewrite MIGs. The first algorithm eliminates inverter components, moving them to *primary inputs*. The second algorithm constrains the maximum fan-out of each node to n, in our case 3. These two algorithms produce networks that can be adapted for majority-based beyond-CMOS technologies, such as QCA and STMG. As a matter of fact, both QCA and STMG have fan-out limitations, since their primitive logic structure relies on a cross-like shape. This means that the primitive fan-out gate for these two technologies has a fan-out of 3 (n = 3). QCA-based circuits can realize inversion, even if not in an efficient way [9]. To implement the inversion in STMG-based circuits, we propose two hybrid solutions combining STMG with CMOS or NML inverters. We demonstrate that QCA-based circuits benefit from inversion-free circuits with constrained fan-out. The *Area-Delay-Energy Product* (ADEP) is on average 3.1× lower compared to the not optimized case. Further, our algorithms allow the realization of circuits based only on STMG. The inversion-free STMG networks produce on average 8.1× smaller ADEP compared to STMG/CMOS hybrids and 2.9× smaller ADEP compared to STMG/NML hybrids.

II. Background on Technologies

In this section, background on QCA and STMG is introduced. Both nanotechnologies have fan-out limitations and they benefit from inversion-free networks.

A. Quantum-Dot Cellular Automata

QCA technology was first proposed in 1993 [5]. This technology is based on the interaction of QCA cells and it can be useful to design circuits with high switching speed, high device density, and low power consumption. Each cell consists of four quantum dots and two free electrons. The electrons are able to tunnel between the dots, which are coupled by tunnel barriers. The electrons are forced by Coulomb repulsion in opposite corners of the cell producing two energetically equivalent polarizations, i.e., $P = 1$ and $P = -1$. The two polarizations are used to represent logic `1` and `0` respectively. However for these polarization states to be energetically stable, the operating temperature is limited to ∼1 K [9].

2159-3477/17 $31.00 © 2017 IEEE
DOI 10.1109/ISVLSI.2017.37

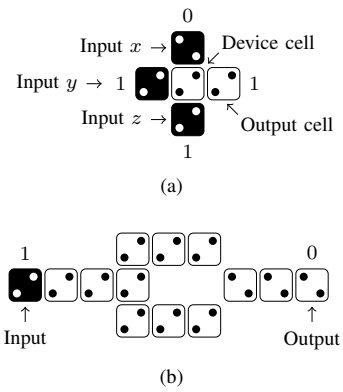

Fig. 1. (a) QCA layout for majority, (b) inverter

TABLE I
TECHNOLOGY PARAMETERS FOR QCA INVERTER AND MAJORITY GATES.

	Area (μm^2)	Delay (ns)	Energy (fJ)
INV	4.0×10^{-3}	1.4×10^{-2}	$9.8{\cdot}10^{-6}$
MAJ	1.2×10^{-3}	4×10^{-3}	$2.9{\cdot}10^{-6}$

It has been demonstrated that the fundamental logic element of QCA is the three-inputs majority gate [13], and QCA based circuits can be build using only majority and inverters [14]. Figs. 1a and 1b show the layout of a QCA majority gate and inverter, respectively. Five QCA cells are needed to build one majority gate. The polarization of the central logic cell, called device cell, is the majority of the three inputs, while the output cell follows the polarization of the device cell. Thirteen quantum cells are used for the inverter (Fig. 1b). At the first branch point, the input wire goes one cell farther the beginning of the offset wires. In this case, aligning effects prevail and the two wires have the same polarization as the input. At the second join, antialigning effects control the polarization of the next cell, causing an inversion in the signal.

Table I introduces the primitive area, delay, and energy constants used for the QCA technology, extracted from [13], [15].

As shown in Fig. 1 and in Table I, the QCA inverter implementation is significantly less efficient compared to the majority gate. For this reason, propagating inverters to the circuit inputs will be beneficial for QCA implementations.

B. Spin Torque Majority Gate

STMG is a three-input majority gate driven by *Spin Transfer Torque* (STT) [16], [17] and has been proposed by Nikonov et al. [6]. It consists of a cross-shaped free layer shared between four *Magnetic Tunnel Junctions* (MTJ) (see Fig. 2). The bit information (`0` or `1`) in the device is represented by the magnetization orientation (up or down) in the free layer. Three MTJs write the input states via STT in a current perpendicular to plane configuration. The forth MTJ reads the output state via tunnel magnetoresistance. The magnetic domains are mainly driven by domain wall automotion, the transport of a magnetic domain wall under the influence of demagnetization and magnetic anisotropy [18]. The operating range has been extensively studied by micromagnetic simulations and numerical modeling in [19] and [20], respectively. The

TABLE II
TECHNOLOGY PARAMETERS FOR STMG HYBRID FLAVORS INVERTER AND MAJORITY GATES

	Area (μm^2)	Delay (ns)	Energy (fJ)
MTJ write/read	0	4	70
MAJ	3.6×10^{-3}	1.5	0
INV-CMOS	6.0×10^{-2}	2.6×10^{-2}	4.0×10^{-1}
INV-NML	2.3×10^{-2}	10	0

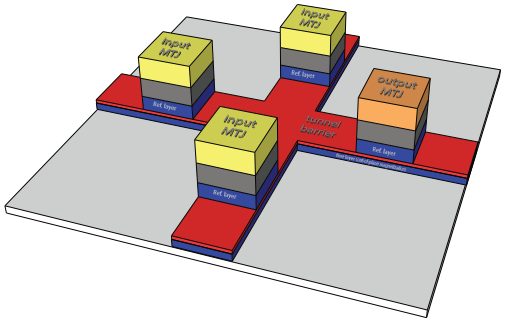

Fig. 2. Schematic of STMG [6]. The three MTJ inputs are visible in yellow, while the output is the orange block.

STMG concept carries the potential of smaller area, low power, nonvolatility, reconfigurability, and radiation hardness [6]. A first inverter concept was presented in [10] where it was assumed that the functionality of an inverter is achieved through a ferromagnetic wire that connects two STMG devices and is fabricated as a slanted layer in the magnetic material stack. However, this concept cannot be realized with state-of-the-art magnetic material integration technology. We can envision two possible flavors of STMG devices that can implement netlists:

1) STMG/CMOS hybrid, where each inversion is implemented by CMOS inverters. This assumes that for each inversion an MTJ is read and the next one is written with the inverted result.
2) STMG/NML hybrid, where each inversion is implemented by an out-of-plane nanomagnet, as in the NML concept. This assumes that there is no conversion to the electric domain and the inversion is implemented in the magnetic domain.

We benchmark and compare the results of our algorithms to these two aforementioned STMG hybrid flavors. For this comparison we use the primitive area, delay, and energy constants shown in Table II. The MTJ parameters are extracted from [21], the CMOS inverter parameters are extracted from a CMOS 7 nm node [22], and the NML inverter is assumed to be a 150 nm×150 nm nanomagnet based on [23].

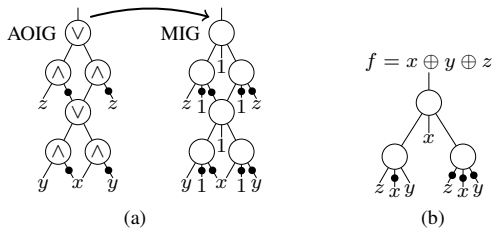

Fig. 3. Example (a) of AOIG (left) and the derived MIG (right) for $f = x \oplus y \oplus z$. From now on, complements will be denoted with bubbles on the edges. (b) is the optimized MIG for f.

III. Majority-based Logic Manipulation

In this section, we describe the data structure used for Boolean functions representation and optimization, i.e., MIGs [12], [24]. A MIG is defined as a homogeneous logic network consisting of 3-input majority nodes and regular and complemented edges. MIGs take advantage of the expressiveness of the majority operator to efficiently represent Boolean functions. Indeed, traditional AND and OR can be obtained from the majority operator. In the case of 3-input majorities, AND is realized by fixing one input to `0` while OR is realized by fixing one input to `1`. As a consequence, *AND/OR/INV Graphs* (AOIGs) are a special case of MIGs and MIGs can be derived from AOIGs. Fig. 3(a) shows an example of AOIG (left) and the MIG (right) obtained from it. MAJ operators with a constant input take the place of AND/OR operators.

Even more advantageous MIG representations can be obtained using nodes with non-constant inputs [12]. A dedicated Boolean algebra was first presented in [24] to manipulate MIG representations. The corresponding axiomatic system consists of five primitive transformation rules and it has been proven that, by using a sequence of transformations from this axiomatic system, it is possible to traverse the entire MIG representation space [12]. In other words, given any two equivalent MIG representations, it is possible to transform one into the other by just using these rules. One MIG can then be transformed into its optimized representation by using these transformations. An optimized MIG for function f is presented in Fig. 3b. Both depth (number of levels) and size (number of nodes) are optimized. All details about MIG and its axiomatic system can be found in [12].

IV. MIG Rewriting Algorithms

This section presents two MIG-rewriting algorithms. The first algorithm, called *Inversion Propagation Algorithm*, propagates all inverters to the inputs in order to obtain an inversion-free MIG. The second algorithm, called *Fan-Out Restriction Algorithm*, limits the maximum fan-out of each node. Both methods aim at not changing the depth of the resulting graph.

A. Inversion Propagation Algorithm

The *Inversion Propagation Algorithm* rewrites the MIG to obtain a network where all inversions are placed on primary inputs. This is achieved by propagating inverters using the transformation rule $\overline{\langle xyz \rangle} = \langle \bar{x}\bar{y}\bar{z} \rangle$, which is one of the axioms presented in [12] and called $\Omega.I$. The idea is to recursively apply $\Omega.I$ to move complemented edges from the outputs to the inputs. Previous works have been presented on inversion minimization [11]. Here, the aim is to obtain a network with all inversions on inputs, then the total number of inversions may not be decreased.

```
   Input: MIG node v
   Output: MIG node v with inversions on inputs
1  p ← polarity(v);
2  if p = 1 ∧ v is not PI then
3  |  apply Ω.I on v;
4  |  p ← 0;
5  end
6  if v is PI then
7  |  return v;
8  end
9  if v' ← is_cached(v, p) then
10 |  return v';
11 end
12 foreach child c of v do
13 |  inv_free(c);
14 end
15 cache(v, p);
16 return v;
```

Algorithm 1: Inversion Propagation Algorithm 'inv_free'

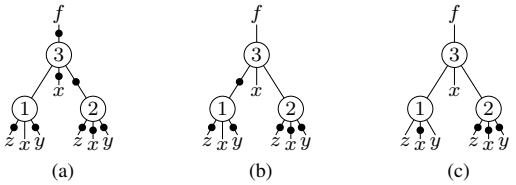

Fig. 4. Example for Alg. 1. (a) is the original MIG for function $f = x \oplus y \oplus z$; (b) represents the same graph where $\Omega.I$ has been applied on node 3; (c) is the inversions free graph.

The algorithm is based on a dynamic programming approach, and it consists of a recursive function called inv_free. The inversion propagation algorithm is shown in Alg. 1. The algorithm starts by applying the function to each output of the network. If the node v is not complemented, the function inv_free is applied recursively to the children (lines 12–13 in Alg. 1). If v is complemented, $\Omega.I$ is applied to the node before applying inv_free to the children (line 2 in Alg. 1). In this second case, the polarities of the children are changed and the algorithm is applied taking into account the new polarities. The function is applied for each output. To avoid solving the same subgraphs more than once, all the computed solutions are cached.

An example is given in Fig. 4. Fig. 4(a) represents the original MIG for function $f = x \oplus y \oplus z$. Since output f is complemented, the rule $\Omega.I$ is applied on node 3. Fig. 4(b) shows the MIG with changed polarities for node 3. At this point, function inv_free can be applied on each child of node 3. Since node 1 is complemented, $\Omega.I$ is applied. Since children of node 1 are all primary inputs, this subgraph is cached. The same procedure applies for the second and third child of node 3. The resulting MIG is shown in Fig. 4(c). All the inversions are on primary inputs.

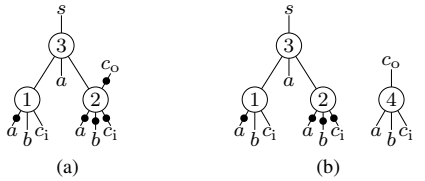

Fig. 5. Example for Alg. 1 leading to a size increase. (a) is the original MIG; (b) represents the same graph after Alg. 1 has been applied.

The proposed algorithm does not change the depth of the graph, but it may result in an increase in the MIG's size. This happens if a node has fan-out with two different polarities. The example in Fig. 5 shows the MIG of a full adder and it explains the size increase. In Fig. 5(a), node 2 has fan-out of 2. The edge going to node 3 is not complemented, while the one going to output c_o has a complementation. To be able to move the negation on output c_o to the inputs, and without changing the polarity of the other outcoming edge, two copies of the same node are necessary. The final result is shown in Fig. 5(b). The depth remains constant, but the size is increased.

B. Fan-Out Restriction Algorithm

The *Fan-out Restriction Algorithm* rewrites the MIG such that every node has a fan-out less or equal to n. The main idea is to create copies of nodes with large fan-out.

The algorithm is similar to the *Inversion Propagation Algorithm*. It is based on a dynamic programming approach and it exploits a recursive function called fo_restr. The recursive function is shown in Alg. 2. The algorithm starts by applying the function to each output. A counter called fo_counter cached the number of times node v is used (line 1 of Alg. 2). For each node v, four different cases are possible.

1) v is a primary input. In this case, the function returns the primary input since the fan-out limit is not applied on inputs of the network.
2) The node is already cached and it has been used less than n times. Since the node has fan-out $< n$, the function returns the cached node. The fo_counter is updated (line 6 in Alg. 2).
3) The node is already cached but it has been used n times (fan-out = n). The same node cannot be returned since the maximum fan-out is reached. For this case, a new node needs to be created. The function fo_restr is applied to the children. The cached value is updated and the counter is reset (lines 12–16).
4) The node is not cached. The recursive function fo_restr is applied to the children and the node is created and cached (lines 12–16).

It follows from this algorithm that if there is at least 1 node with fan-out $> n$, the size of the graph is increased; the depth however remains unchanged as we aim at keeping the same circuit speed. Fig. 6 shows an example with $n = 1$. The algorithm starts from node 4 and recursively reaches the node 1. When child 1 is accessed from node 2, the node itself is returned. Node 3 has node 1 as child as well, but it has already been used $n = 1$ times. In this case, a copy of the node 1 is created. Fig. 6 shows the final result. All nodes have fan-out ≤ 1; the size is increased, while the depth remains constant.

```
   Input: MIG node v
   Output: MIG node v with fan-out ≤ n
1  f ← fo_counter(v);
2  if v is PI then
3  |  return v;
4  end
5  if v' ← is_cached(v, p) ∧ f < n then
6  |  fo_counter(v)++;
7  |  return v';
8  end
9  if v' ← is_cached(v, p) ∧ f = n then
10 |  remove (v');
11 end
12 foreach child c of v do
13 |  fo_restr(c);
14 end
15 fo_counter(v) ← 1;
16 cache(v);
17 return v;
```

Algorithm 2: Fan-out Restriction Algorithm 'fo_restr'

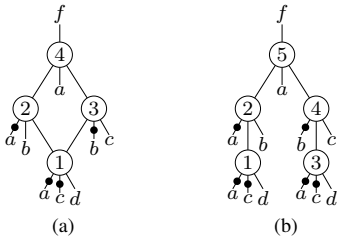

Fig. 6. Example for the Fan-Out Restriction Algorithm. (a) is the original MIG; (b) represents the same graph after fan-out restriction.

It is important to highlight that this algorithm can lead to an exponential size increase. New copies of nodes lead to a fan-out increase for their children. This can result in new copies also for nodes that were not reaching the fan-out limit in the original MIG. A possible solution could be to use a method similar to the one proposed in [25]. The method consists of introducing one more level with *buffer nodes* reproducing the original node. In this way the maximum fan-out of the node is 9 without introducing any new copy, hence keeping children fan-out unchanged. This alternative method could limit the size increase, but it results in a depth increase for the MIG. We applied this alternative method on the non-critical paths of some benchmarks. This mixed method leads to small improvements as compared to Alg 2. Furthermore, for some circuits, the area increase is larger. For these reasons, we present here only the results obtained with Alg. 2.

V. Results

This section describes results obtained with the algorithms proposed in Section IV. We also test our methods for two different technologies: QCA and STMG.

A. Inversion-Free and Fan-Out Restriction

In order to obtain inversions free circuits with restricted fan-out, we developed two C++ programs to implement Alg. 1 and Alg. 2. We evaluate our methods on circuits from EPFL benchmarks.[1] First we applied Alg. 1, then, on the same network, Alg. 2 using $n = 3$. Results are listed in Table III. The depth of the circuit remains constant, while the final size is 1.9× larger on average. The 'Depth + Inv' is the path with maximum number of majority nodes and inversions. The '#INV' is the number of MIG nodes with at least one complemented fan-out. The '#INV' in Table III refers to inversions on primary input, since all circuits are inversion-free. The number of inversions is lower as compared to the original graph; only two circuits (*dec* and *router*) have the same number of inversions after the algorithms are applied.

B. QCA and STMG Results

In this section, we show how QCA-based circuits benefit from inversion-free networks with limited fan-out. We also demonstrate the realization of STMG-based circuits which do not include hybrid inversions.

We evaluate area, delay, and energy of QCA-based circuits using the specification from Table I. Table IV shows the experimental results. The optimized MIG is the one without inversions and with fan-out limited to 3. Even if the optimized MIG has larger size (see Table III), this leads to a smaller QCA area, since QCA inverters are much larger compared to a majority gate. The inversion-free and fan-out restricted netlists yield on average 3.1× smaller ADEP. Only one benchmark (*dec*) has a slightly worse ADEP; all others circuits benefit from inversion-free and fan-out restriction.

We evaluate area, delay, and energy for the STMG-based circuits. The inversion-free MIGs obtained with our algorithms allow realization of STMG-based circuits. We compare our results both with STMG/CMOS hybrid and STMG/NML hybrid circuits. In both cases CMOS and NML are used in order to obtain inversions. Table V shows the STMG experimental results. The STMG-based circuits produce, on average, both lower area and delay. On average, a 8.1× smaller ADEP compared to STMG/CMOS hybrids and 2.9× smaller ADEP compared to STMG/NML hybrids is obtained using only STMG. In this case, all circuits benefit from inversion-free and fan-out restriction.

VI. Conclusion

We described majority-based synthesis methods that consider two main technological constraints for emerging technologies. We implemented an algorithm that can synthesize inversion-free networks in which all inversion are on primary inputs. Further, we developed a method to restrict the number of fan-outs for each majority node of the circuit. We applied the two algorithms on the same networks.

At the logic level, a size increase is obtained, but a decrease in the number of inversions is achieved. Experiments showed that both QCA and STMG circuits benefit from this technology aware synthesis. The ADEP is decreased of 3.1× in the QCA case. 8.1× smaller ADEP compared to STMG/CMOS hybrids and 2.9× smaller ADEP compared to STMG/NML hybrids are obtained using STMG-based circuits.

Acknowledgment: This research was supported by H2020-ERC-2014-ADG 669354 CyberCare, the Swiss National Science Foundation (200021-169084 MAJesty), and the I&C Fellowship from EPFL.

[1] http://lsi.epfl.ch/benchmarks

References

[1] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.
[2] V. V. Zhirnov, R. K. Cavin, J. A. Hutchby, and G. I. Bourianoff. Limits to binary logic switch scaling - a gedanken model. *Proc. of the IEEE*, 91(11):1934–1939, 2003.
[3] J. A. Hutchby, G. I. Bourianoff, V. V. Zhirnov, and J. E. Brewer. Extending the road beyond CMOS. *IEEE Circuits and Devices Magazine*, 18(2):28–41, 2002.
[4] Y.-M. Lin, J. Appenzeller, J. Knoch, and P. Avouris. High-performance carbon nanotube field-effect transistor with tunable polarities. *TNANO*, 4(5):481–489, 2005.
[5] C. S. Lent, D. P. Tougaw, W. Porod, and G. H. Bernstein. Quantum cellular automata. *TNANO*, 4(1):49–57, 1993.
[6] D. E. Nikonov, G. I. Bourianoff, and T. Ghani. Proposal of a spin torque majority gate logic. *IEEE Electron Device Letters*, 32(8):1128–1130, 2011.
[7] G. Csaba, A. Imre, G. H. Bernstein, W. Porod, and V. Metlushko. Nanocomputing by field-coupled nanomagnets. *TNANO*, 99(4):209, 2002.
[8] D. E. Nikonov and I. A. Young. Overview of beyond-CMOS devices and a uniform methodology for their benchmarking. *Proc. of the IEEE*, 101(12):2498–2533, 2013.
[9] C. S Lent and P. D. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85(4):541–557, 1997.
[10] D. E. Nikonov, G. I. Bourianoff, and T. Ghani. Nanomagnetic circuits with spin torque majority gates. In *IEEE-NANO*, pages 1384–1388, 2011.
[11] E. Testa, M. Soeken, O. Zografos, L. Amaru, P. Raghavan, R. Lauwereins, P.-E. Gaillardon, and G. De Micheli. Inversion optimization in majority-inverter graphs. In *NANOARCH*, pages 15–20, 2016.
[12] L. Amarú, P.-E. Gaillardon, and G. De Micheli. Majority-inverter graph: a new paradigm for logic optimization. *TCAD*, 35(5):806–819, 2016.
[13] D. P. Tougaw and C. S. Lent. Logical devices implemented using quantum cellular automata. *AlP*, 75(3):1818–25, 1993.
[14] R. Zhang, P. Gupta, and N. K. Jha. Synthesis of majority and minority networks and its applications to QCA, TPL and SET based nanotechnologies. *VLSID*, pages 229–234, 2005.
[15] I. Hanninen and J. Takala. Pipelined array multiplier based on quantum-dot cellular automata. In *ECCTD*, pages 938–941, 2007.
[16] L. Berger. Emission of spin waves by a magnetic multilayer traversed by a current. *Physical Review B*, 54(13):9353–8, 1996.
[17] J. C. Slonczewski. Current-driven excitation of magnetic multilayers. *Journal of Magnetism and Magnetic Materials*, 159(1-2):L1–L7, 1996.
[18] D. E. Nikonov, S. Manipatruni, and I. A. Young. Automotion of domain walls for spintronic interconnects. *Journal of Applied Physics*, 115(21):213902, 2014.
[19] A. Vaysset, M. Manfrini, D. E. Nikonov, S. Manipatruni, I. A. Young, G. Pourtois, I. P. Radu, and A. Thean. Toward error-free scaled spin torque majority gates. *AIP Advances*, 6(6):065304, 2016.
[20] A. Vaysset, M. Manfrini, D. E. Nikonov, S. Manipatruni, I. A. Young, I. P. Radu, and A. Thean. Operating conditions and stability of spin torque majority gates: Analytical understanding and numerical evidence. *Journal of Applied Physics*, 121(4):043902, 2017.
[21] K. Ikegami, H. Noguchi, C. Kamata, M. Amano, K. Abe, , and *et al.* Low power and high density stt-mram for embedded cache memory using advanced perpendicular mtj integrations and asymmetric compensation techniques. In *IEDM*, pages 28–1, 2014.
[22] P. Raghavan, M. G. Bardon, D. Jang, P. Schuddinck, D. Yakimets, and *et al.* Holisitic device exploration for 7nm node. In *CICC*, pages 1–5, 2015.
[23] S. Breitkreutz, J. Kiermaier, I. Eichwald, C. Hildbrand, G. Csaba, and *et al.* Experimental demonstration of a 1-bit full adder in perpendicular nanomagnetic logic. *IEEE Trans. on Magnetics*, 49(7):4464–4467, 2013.

TABLE III
INVERSION PROPAGATION AND FAN-OUT RESTRICTION ALGORITHMS ON EPFL BENCHMARKS

Benchmark				Original MIG			Optimized MIG				
Name	IN	OUT	Depth	Depth + Inv	Size	# INV	Depth + Inv	Size	× Larger	# INV	× Smaller
adder	256	129	12	21	2978	1449	13	6963	2.3	256	5.7
arbiter	256	129	14	27	2179	1042	15	2892	1.3	128	8.1
bar	135	128	14	28	3054	2999	15	3054	1.0	7	428.4
cavlc	10	11	11	21	745	370	12	955	1.3	10	37.0
ctrl	7	26	5	10	127	72	6	138	1.1	7	10.3
dec	8	256	4	5	420	8	5	528	1.3	8	1.0
i2c	147	142	9	18	1108	762	10	1354	1.2	93	8.2
int2float	11	7	9	15	246	121	10	279	1.1	11	11.0
log2	32	32	181	343	37582	22481	182	109839	2.9	32	702.5
max	512	130	27	53	7202	3147	28	21250	3.0	512	6.1
mem	1204	1231	18	36	10362	8519	19	14659	1.4	801	10.6
mult	128	128	111	205	41885	25594	112	113215	2.7	128	200.0
priority	128	8	10	18	498	295	11	621	1.2	122	2.4
router	60	30	9	13	113	62	10	185	1.6	60	1.0
sin	24	25	91	165	7890	3823	92	25557	3.2	24	159.3
sqrt	128	64	690	1249	52344	28734	691	174018	3.3	126	228.0
square	64	128	36	70	19200	17158	37	39234	2.0	64	268.1
voter	1001	1	60	114	14078	12064	61	31980	2.3	1001	12.1
Average									1.9		116.7

TABLE IV
QCA EXPERIMENTS

	Original				Optimized MIG			
Name	Area (μm^2)	Delay (ns)	Energy (fJ)	ADEP (a.u.)	Area (μm^2)	Delay (ns)	Energy (fJ)	ADEP (a.u.)
adder	9.4	1.7×10^{-1}	2.3×10^{-2}	3.7×10^{-2}	9.4	6.2×10^{-2}	2.3×10^{-2}	1.3×10^{-2}
arbiter	6.8	2.4×10^{-1}	1.7×10^{-2}	2.7×10^{-2}	4.0	7×10^{-2}	9.8×10^{-3}	2.7×10^{-3}
bar	1.6×10^{1}	2.5×10^{-1}	3.8×10^{-2}	1.5×10^{-1}	3.7	7×10^{-2}	9.0×10^{-3}	2.3×10^{-3}
cavlc	2.4	1.8×10^{-1}	5.8×10^{-3}	2.5×10^{-3}	1.2	5.8×10^{-2}	2.9×10^{-3}	2×10^{-4}
ctrl	5×10^{-1}	9×10^{-2}	1.1×10^{-3}	4.3×10^{-5}	1.9×10^{-1}	3.4×10^{-2}	4.7×10^{-4}	3.1×10^{-6}
dec	5×10^{-1}	3×10^{-2}	1.3×10^{-3}	2.1×10^{-5}	6.7×10^{-1}	3×10^{-2}	1.6×10^{-3}	3.3×10^{-5}
i2c	4.4	1.6×10^{-1}	1.1×10^{-2}	7.6×10^{-3}	2.0	5×10^{-2}	4.9×10^{-3}	4.9×10^{-4}
int2float	8×10^{-1}	1.2×10^{-1}	1.9×10^{-3}	1.8×10^{-4}	3.8×10^{-1}	5×10^{-2}	9.3×10^{-4}	1.8×10^{-5}
log2	1.4×10^{2}	3.0	3.3×10^{-1}	1.3×10^{2}	1.3×10^{2}	7.4×10^{-1}	3.2×10^{-1}	3.1×10^{1}
max	2.1×10^{1}	4.7×10^{-1}	5.2×10^{-2}	5.2×10^{-1}	2.8×10^{1}	1.2×10^{-1}	6.7×10^{-2}	2.3×10^{-1}
mem	4.7×10^{1}	3.2×10^{-1}	1.1×10^{-1}	1.7	2.1×10^{1}	8.6×10^{-2}	5.1×10^{-2}	9.1×10^{-2}
mult	1.5×10^{2}	1.8	3.7×10^{-1}	1.0×10^{2}	1.4×10^{2}	4.6×10^{-1}	3.3×10^{-1}	2.1×10^{1}
priority	1.8	1.5×10^{-1}	4.4×10^{-3}	1.2×10^{-3}	1.2	5.4×10^{-2}	3.0×10^{-3}	2.0×10^{-4}
router	4×10^{-1}	9.2×10^{-2}	9.4×10^{-4}	3.3×10^{-5}	4.6×10^{-1}	5×10^{-2}	1.1×10^{-3}	2.6×10^{-5}
sin	2.5×10^{1}	1.4	6.1×10^{-2}	2.1	3.1×10^{1}	3.8×10^{-1}	7.5×10^{-2}	8.8×10^{-1}
sqrt	1.8×10^{2}	1.1×10^{1}	4.4×10^{-1}	8.2×10^{2}	2.1×10^{2}	2.8	5.1×10^{-1}	3.0×10^{2}
square	9.2×10^{1}	6.2×10^{-1}	2.2×10^{-1}	1.3×10^{1}	4.7×10^{1}	1.6×10^{-1}	1.2×10^{-1}	8.7×10^{-1}
voter	6.5×10^{1}	10.0×10^{-1}	1.6×10^{-1}	1.0×10^{1}	4.2×10^{1}	2.5×10^{-1}	1.0×10^{-1}	1.1
Averages	4.2×10^{1}	1.1	1.0×10^{-1}	6.0×10^{1}	3.7×10^{1}	3.1×10^{-1}	9.1×10^{-2}	2.0×10^{1}

TABLE V
STMG EXPERIMENTS

	Original MIG — STMG/CMOS				Original MIG — STMG/NML				Optimized MIG			
Name	Area (μm^2)	Delay (ns)	Energy (nJ)	ADE (a.u.)	Area (μm^2)	Delay (ns)	Energy (nJ)	ADE (a.u.)	Area (μm^2)	Delay (ns)	Energy (nJ)	ADE (a.u.)
adder	8.7×10^{1}	9.4×10^{1}	2.5×10^{-1}	2.1×10^{3}	4.3×10^{1}	1.1×10^{2}	1.5×10^{-1}	7.2×10^{2}	2.5×10^{1}	1.8×10^{1}	4.8×10^{-1}	2.2×10^{2}
arbiter	6.3×10^{1}	1.3×10^{2}	2.7×10^{-1}	2.2×10^{3}	3.1×10^{1}	1.6×10^{2}	2.0×10^{-1}	9.5×10^{2}	1.0×10^{1}	2.1×10^{1}	2.7×10^{-1}	5.8×10^{1}
bar	1.8×10^{2}	1.4×10^{2}	6.0×10^{-1}	1.5×10^{4}	7.8×10^{1}	1.7×10^{2}	3.9×10^{-1}	5.1×10^{3}	1.1×10^{1}	2.1×10^{1}	3.9×10^{-1}	9.0×10^{1}
cavlc	2.2×10^{1}	1.0×10^{2}	1.1×10^{-1}	2.4×10^{2}	1.1×10^{1}	1.2×10^{2}	8.3×10^{-2}	1.1×10^{2}	3.4	1.7×10^{1}	1.2×10^{-1}	6.6
ctrl	4.3	5.2×10^{1}	2.2×10^{-2}	4.8	2.1	6.2×10^{1}	1.7×10^{-2}	2.1	5.0×10^{-1}	7.5	1.9×10^{-2}	7.1×10^{-2}
dec	1.5	1.8×10^{1}	5.0×10^{-2}	1.4	1.7	2.0×10^{1}	5.0×10^{-2}	1.7	1.9	6.0	6.5×10^{-2}	7.4×10^{-1}
i2c	4.6×10^{1}	9.0×10^{1}	1.9×10^{-1}	7.8×10^{2}	2.1×10^{1}	1.1×10^{2}	1.4×10^{-1}	3.1×10^{2}	5.6	1.4×10^{1}	1.7×10^{-1}	1.3×10^{1}
int2float	7.3	6.6×10^{1}	3.9×10^{-2}	1.9×10^{1}	3.6	7.8×10^{1}	3.1×10^{-2}	8.6	1.0	1.4×10^{1}	3.6×10^{-2}	5.0×10^{-1}
log2	1.3×10^{3}	1.6×10^{3}	4.2	8.9×10^{6}	6.4×10^{2}	1.9×10^{3}	2.6	3.2×10^{6}	4.0×10^{2}	2.7×10^{2}	7.9	8.5×10^{5}
max	1.9×10^{2}	2.5×10^{2}	7.8×10^{-1}	3.7×10^{4}	9.7×10^{1}	3.0×10^{2}	5.6×10^{-1}	1.6×10^{4}	7.7×10^{1}	4.1×10^{1}	1.6	5.0×10^{3}
mem	5.1×10^{2}	1.8×10^{2}	1.8	1.6×10^{5}	2.3×10^{2}	2.1×10^{2}	1.2	5.8×10^{4}	5.3×10^{1}	2.7×10^{1}	1.7	2.4×10^{3}
mult	1.5×10^{3}	9.2×10^{2}	4.9	6.9×10^{6}	7.3×10^{2}	1.1×10^{3}	3.1	2.5×10^{6}	4.1×10^{2}	1.7×10^{2}	8.2	5.6×10^{5}
priority	1.8×10^{1}	8.3×10^{1}	7.5×10^{-2}	1.1×10^{2}	8.4	9.9×10^{1}	5.4×10^{-2}	4.5×10^{1}	7.3	1.5×10^{1}	7.2×10^{-2}	7.9
router	3.7	5.0×10^{1}	1.9×10^{-2}	3.5	1.8	5.8×10^{1}	1.4×10^{-2}	1.5	3.6	1.4×10^{1}	2.4×10^{-2}	1.2
sin	2.3×10^{2}	7.3×10^{2}	7.3×10^{-1}	1.2×10^{5}	1.1×10^{2}	8.8×10^{2}	4.6×10^{-1}	4.6×10^{4}	9.2×10^{1}	1.4×10^{2}	1.6	2.0×10^{4}
sqrt	1.7×10^{3}	5.5×10^{3}	5.3	5.0×10^{7}	8.3×10^{2}	6.6×10^{3}	3.2	1.8×10^{7}	6.3×10^{2}	1.0×10^{3}	1.1×10^{1}	7.0×10^{6}
square	1.0×10^{3}	3.3×10^{2}	2.8	9.4×10^{5}	4.6×10^{2}	4.0×10^{2}	1.5	2.8×10^{5}	1.4×10^{2}	5.4×10^{1}	3.2	2.5×10^{4}
voter	7.2×10^{2}	5.3×10^{2}	2.0	7.5×10^{5}	3.2×10^{2}	6.3×10^{2}	1.1	2.3×10^{5}	1.2×10^{2}	9.0×10^{1}	2.5	2.6×10^{4}
Averages	4.3×10^{2}	6.0×10^{2}	1.3	3.8×10^{6}	2.0×10^{2}	7.2×10^{2}	8.3×10^{-1}	1.3×10^{6}	1.1×10^{2}	1.1×10^{2}	2.2	4.7×10^{5}

[24] L. Amarú, P.-E. Gaillardon, and G. De Micheli. Majority-inverter graph: a novel data-structure and algorithms for efficient logic optimization. *DAC*, pages 194:1–194:6, 2014.

[25] O. Zografos, A. De Meester, E. Testa, M. Soeken, P.-E. Gaillardon, and *et al.* Wave pipelining for majority-based beyond-CMOS technologies. *DATE*, 2017.