Technology Mapping with All Spin Logic

Boyu Zhang, and Azadeh Davoodi
University of Wisconsin - Madison
1415 Engineering Dr. Madison WI USA 53706
{bzhang93,adavoodi}@wisc.edu

***Abstract*—This work is the first to propose a technology mapping algorithm for All Spin Logic (ASL) device. The ASL is the most actively-pursed one among spintronic devices which themselves fall under emerging post-CMOS nano-technologies. We identify the shortcomings of directly applying classical technology mapping with ASL devices, and propose techniques to extend it to handle these shortcomings. Our results show that our ASL-aware technology mapping algorithm can achieve on-average 9.15% and up to 27.27% improvement in delay (when optimizing delay) with slight improvement in area, compared to the solution generated by classical technology mapping. In a broader sense, our results show the need for developing circuit-level CAD tools that are aware of and optimized for emerging nano-technologies in order to better assess their promise as we move to the post-CMOS era.**

I. Introduction

With the slowing of Moore's Law over the past decade, it is no longer viable to build computing systems by using doubling the number of CMOS transistors as core building blocks of Integrated Circuits. A number of emerging post-CMOS nano-technologies have introduced new devices as candidates to replace the building block of future computing systems. However, the logic models in many of these devices are different from that of traditional CMOS devices. For example, a number of post-CMOS nano-technologies have an elementary device abstraction that logically function as a majority voter. Examples includes silicon nano-wires, graphene, resistive random-access memory, spintronic devices, quantum-dot cellular automata, nano-magnets, DNA logic, etc [3]. To assess the promise of these devices, it is essential to develop new CAD tools for synthesizing systems in terms of these emerging devices.

Recently there have been a number of efforts to develop logic abstractions and synthesis tools for majority-based logic such as [2], [3]. Specifically Majority Inverter Graph (MIG) is proposed as a new data structure with a novel Boolean algebra to more effectively represent and handle majority-based logic [2], [3]. It is shown to achieve great improvements in delay, area, and power compared to other forms of representation such as AIG and BDD. While MIG has shown to be a great model for representing majority-based post-CMOS nano-technologies, the techniques used to optimize an MIG can vary depending on the specific nano-technology. For example, recently, the work [7] proposes MIG optimization for logic synthesis when the underlying nano-technology is Resistive RAM (RRAM).

In this work, we propose CAD techniques for technology mapping when the underlying nano-technology is spintronic devices. Specifically we focus on the most actively-pursued class of spintronic device, namely the All Spin Logic (ASL). We first review a summary of ASL device structure, logic gate construction, as well as cascade of ASL logic gates. Specifically, we focus on how "cost" should be computed when ASL logic gates are connected to each other.

For two cases, when cost is area and delay, we discuss how cascaded ASL gates are fundamentally *interlocked* to one another. We show ignoring this factor when applying the basic standard technology mapping algorithm can result in significant over-estimation of delay and area, and an adjustment in cost computation during standard technology mapping to account for this over-estimation. Furthermore, we discuss how ASL technology opens new opportunities during the matching phase of technology mapping. To illustrate this point, we discuss how standard pattern matching can miss some ASL-specific opportunities. Another distinguishing feature of ASL technology is the fact that the MIG patterns representing the ASL logic gates often contain constant binary values corresponding to fixed magnets. This can complicate the pattern generation process within technology mapping, if these constant values are also allowed to be flipped. To take advantage of ASL-specific opportunities and address the challenges, we insert a new step during the pattern matching phase of technology mapping in which we make local transformations to the MIG representing the "subject graph" during pattern matching.

II. Background and Preliminaries

A. Basics of All-Spin Logic (ASL) Device and Logic Gates

One family of spintronic devices is based on Spin Transfer Torque switching. In these devices, data is represented by two states of (high and low) electrical resistance, which can be controlled by manipulating the relative magnetization orientation of two adjacent ferromagnet layers using the torque generated by spin-polarized current [4], [8]. Among these devices, the one that has actively being pursued is All Spin Logic (ASL) devices [5]. Fig. 1(a) is a conceptual layout of an ASL-based inverter. It is composed of an input and an output nanomagnet where the orientations of the magnets represent binary information. It is also made of a channel to transfer spin information from input to output, an isolation layer to separate devices from each other, an interface between the nanomagnet and channel to inject spin-polarized electrons [5], and connections to supply and properly-placed ground nodes [9]. The supply voltage (V_{supply}) and ground are used to provide input current (I_{supply}). When positive V_{supply} is applied to the input magnet, electrons flow from ground to V_{supply}. The electrons with the same spin direction as the input magnet pass through the input magnet and reach V_{supply}, while the electrons with opposite spin polarity get blocked and accumulate in the channel under the input magnet. These accumulated spin-polarized electrons induce nonequilibrium magnetization, enforcing spin diffusion along the channel in the form of spin current (I_{spin}).

Besides single-input ASL gates such as inverter and buffer shown in Fig. 1(a) and Fig. 1(b), the model can be extended to realize multi-input Boolean functions. In the ASL-based implementation of multi-input functions, multiple input magnets connect to a single output magnet, as shown in Fig. 1(c) and Fig. 1(d). The magnetization orientation of the output magnet is determined by the *superposition* of I_{spin} currents generated of the input magnets. In other words, the output value of a multi-input gate corresponds to the minority (or majority if V_{supply} is negative) of its inputs. In this sense, multi-input ASL logic gates inherently implement majority or minority operations.

978-3-9815370-8-6/17/$31.00

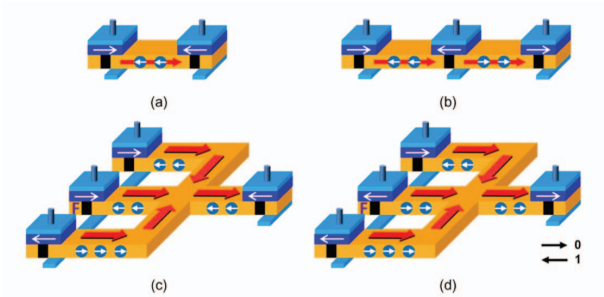


Fig. 1: Examples of Boolean functions implemented with ASL device [5]: (a) inverter (b) buffer (c) NAND2 (d) NOR2.

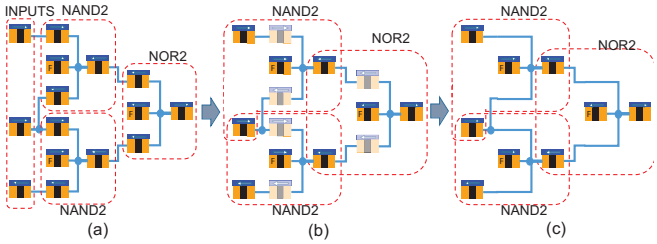


Fig. 2: Two implementations of cascaded ASL gates (a) without magnet sharing and (c) with magnet sharing which can reduce cost. The magnets that can be removed are highlighted in (b).

In Fig. 1(c) and Fig. 1(d), the magnets with fixed magnetization orientation (indicated by 'F') are used as input magnets. The arrow direction corresponds to 0 or 1 as drawn in the legend. For example, the structure in Fig. 1(c) realizes a NAND2 gate when the middle input magnet is fixed corresponding to a constant 0. Similarly, Fig. 1(d) shows realization of a NOR2 function. Table I shows a library of different ASL logic gates. For each gate, estimates of delay and area are provided in terms of magnet count. (For area, all the magnets are counted as given in [5] while for delay we count the magnets on the longest path.)

When it comes to building circuits with ASL gates, the natural way of thinking is to connect individual ASL gates together as it is done in CMOS technology. This approach will result in a physical layout like Fig. 2(a). However a better approach for cascading is shown in [5] in which each output magnet of a gate becomes the input magnet of its fanout gate. With this approach, one magnet can be removed between each gate and it's fanout(s). This results in saving in delay and area while logic equivalence is maintained. Essentially, in this approach, ASL gates are not connected to but rather interlocked with each other. In the previous example, the magnets that can be removed are highlighted in Fig. 2(b) and the final result is shown in Fig. 2(c). Compared to the first approach, area and delay are reduced.

B. Basics of Majority Inverter Graph (MIG) and Its Boolean Algebra

In this work we assume the ASL-based logic gates and circuits are initially expressed as a Majority Inverter Graph (MIG) which is a recently-proposed data structure for representing logic networks [2]. An MIG is a directed graph in which each node represents the majority function and has in-degree of 3. Each directed edge can have a regular or complemented attribute. Table I shows the MIG representations of a few gates. Note that the MIG of an inverter is represented as a complemented edge (edge with black circle on it). The work [2] proposes a novel Boolean algebra for MIGs . The rule that interests us and is used in our work is *Inverter Propagation*:

$$M'(x, y, z) = M(x', y', z') \quad (1)$$

where M is the majority operator and $'$ is the complement operator.

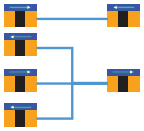
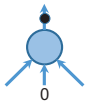
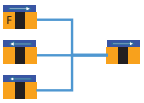
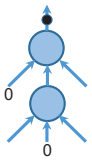
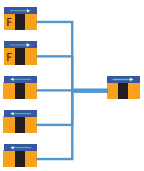
Gate	Area	Delay	MIG pattern	Conceptual Layout
INVERTER	2	2		
MIN (minority)	4	2		
NAND2	4	2		
NAND3	6	2		
NOR2	4	2	—	—
NOR3	6	2	—	—
MAJ (majority)	5	3	—	—
AND2	5	3	—	—
AND3	7	3	—	—
OR2	5	3	—	—
OR3	7	3	—	—
Notes:	 stands for 0;  stands for 1.			

Table I: Area, delay, MIG pattern, and layout of the gates in our used ASL library. Some entries are not shown due to lack of space.

III. Problem Statement

We are given a design's netlist expressed as an MIG which we refer to as a subject graph. We are also given a library of ASL gates. For each gate in the library a set of pattern graphs are created which is a group of equivalent MIGs representing the gate's logic function[1]. Let a "cover" be a collection of patterns such that every node of the subject graph is contained in one (*or more*) pattern, and each input required by a pattern graph is an output of some other pattern.

ASL-Map problem: *Given a subject graph as an MIG and an ASL-based gate library, the ASL-Map aims to find a minimum-cost cover of the subject graph by choosing from the pattern graphs in the library.* In this work, we assume cost is delay. Specifically, we minimize the latest *arrival time* of the signal at the primary outputs.

We briefly review the classical technology mapping to accommodate our discussions and refer the reader to [6] for more details. The classical algorithm is made of two phases of pattern matching and covering. In the matching phase, the nodes in the subject graph are traversed in topological order from primary inputs to outputs. At each node i, pattern matching is performed against all the pattern graphs in the ASL gate library. For each pattern $p \in P$ (where P is the set of patterns matching at i), we calculate an arrival time by finding the maximum of the sum of the delay of the pattern and the arrival time stored at each of its fanin nodes. The fanins are the nodes which feed as inputs of the matched pattern. Among all the matched patterns, the one with the minimum arrival time is selected as the best one for that node. The above process is mathematically expressed as $AT_i = \min_{\forall p \in P} \max_{\forall j \in fanin_p}(d_p + AT_j)$ where for pattern p, d_p is delay of the pattern and AT_j is the arrival time stored at its fanin j. This arrival time and the corresponding pattern are stored as the solution at node i so they may be looked up in subsequent stages of the algorithm. The algorithm then moves to the next node. The procedure is repeated until all nodes are processed.

[1]Table I shows only one MIG per gate. We explain our process of generating pattern graphs for each gate later in Section IV.

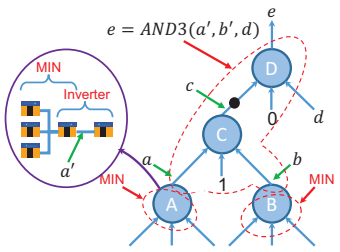


Fig. 3: ASL technology opens new, non-standard, opportunities for matching gates on the subject graph.

Next the algorithm enters the covering phase, in which a mapped circuit is generated by backward traversing the subject graph from primary outputs to inputs. For a visited node, its stored pattern from the matching phase will be selected to be part of the final cover, if it does not fall in an already-selected pattern. For directed acyclic subject graphs (DAGs), additional techniques must be used.

IV. Our Techniques

A. Key Observations

Our techniques are addressing two shortcomings of the classical technology mapping algorithm in solving the ASL-Map problem.

First, during pattern matching at each node, computation of the arrival time (AT) for a matched ASL pattern is always over-estimated in the classical algorithm. This can result in selecting the wrong solution at each node and yield to overall sub-optimality.

Second, ASL technology opens up new opportunities for matching gates on the subject graph which cannot be achieved using standard pattern matching. As an example, consider the subject graph shown as an MIG in Fig. 3. The large red circle represents the AND3 operation ($e = AND3(a', b', d)$). This circle *partially* covers nodes A and B because signals a' and b' are taken from inside the majority nodes.

Since the signals a' and b' are not available as edges in the MIG, none of the AND3 pattern graphs matches at node D. However the ASL technology allows providing a'. This is because as shown in the figure, the majority node A is implemented as a minority gate cascaded with an inverter. Therefore a' is available at the output of the minority gate. So we only need to implement a minority gate to obtain a' and can then can cascade it with the AND3 gate in this example. Similarly signal b' can also be extracted. With this ASL-specific assumption, a new "match" can be found at node D which does not follow the classical pattern matching process.

Besides the above two shortcomings, a distinguishing feature of ASL-based technology mapping is that most of ASL gates (except for simple gates such as INV, MAJ, MIN) include one or more fixed magnets. Therefore their pattern graphs include one or more constants as shown in the MIGs for some gates in Table I.

During the process of generating logically-equivalent pattern graphs for each gate, starting from an initial pattern, it is also possible to generate variations of it by allowing the values of the constants to be flipped. However we observed that in practice generating all attainable variations using this rule can result in a prohibitively-large number of patterns. Therefore, in our framework, we found it is best to limit the pattern generation process to those patterns which are structurally unique, and do not allow changing the constants in the patterns. Instead, we discuss how local transformation to the subject graph is used during the matching phase which allows the possibility of changing the constants in the subject graph.

B. Details of Our Techniques

We present two aspects of our techniques in detail. We only discuss the modifications made to the classical algorithm given in Section III.

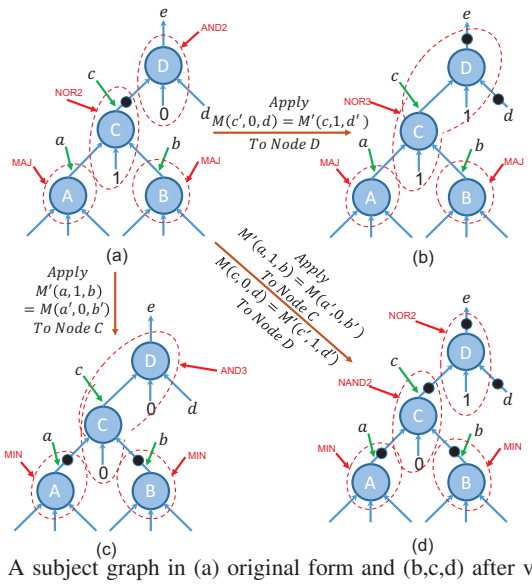


Fig. 4: A subject graph in (a) original form and (b,c,d) after various local transformations using the *Inverter Propagation* rule.

(1) Accurate Computation of Cost During Pattern Matching: For a matched pattern p at an arbitrary node n with an arbitrary fanin f (where f is input pin of the matched pattern), magnet sharing should be considered in the corresponding ASL gates. Specifically, we consider magnet sharing between the ASL gate corresponding to pattern p and the ASL gate corresponding to fanin f, for each fanin of p. Therefore for a pattern with k fanins, there will be k shared magnets. During the computation of the AT metric, a small adjustment is made to account for this sharing. Specifically, there will be 1 unit of overestimation in delay between pattern p and the pattern stored at f so our adjustment subtracts 1 unit from the AT.

(2) Applying Local Transformations to the Subject Graph: During the matching phase of technology mapping, and while visiting a node, we first apply a number of local transformations to the subject graph itself. These transformations result in slightly different subject graphs. These subject graphs are all logically equivalent and structurally very similar. Only the complement attribute on some edges and some constant values may be flipped. We then apply standard pattern matching on each subject graph at that visited node.

Next, we discuss how our subject graphs are generated. When considering node n for pattern matching, we use the inverter propagation rule given by Eq. 1 to alter the subject graph at that node. We also apply the same rule to each (non-primary input) fanin node of n, as well as to the pair consisting of the node and each of its fanins.

Consider Fig. 4 as an example. Assume the *current* node that is being considered is D. The original subject graph is shown in Fig. 4(a). Fig. 4(b) shows how the inverter propagation rule is applied to node D. Fig. 4(c) shows how the rule is applied to fanin node C. Finally Fig. 4(d) shows when the rule is applied to both nodes C and D. The other input of D is primary input node so no action is needed. Overall three new subject graphs are created in this example.

Next standard pattern matching is applied at node D for these (overall) four subject graphs. New matching opportunities now exist as a result of these local alterations. These new matching opportunities are shown by red circles in the figure. For example, the matched pattern shown in Fig. 4(c) cannot be found on the (original) MIG in Fig. 4(a). This example in fact addresses the shortcoming illustrated in Fig. 3 (and for the same original MIG). Also notice in Figs. 4(b,c,d) some constants are flipped compared to the subject graph in Fig. 4(a).

V. Simulation Results

A. Simulation Configuration

We implemented our ASL technology-specific mapping procedure in C++ and ran experiments on a 2.3 GHz Intel Core i7 processor with 16 GB memory. In our experiments, all subject graphs are MIGs which are obtained from [1]. These benchmarks have already been optimized extensively at the technology-independent phase within the synthesis flow. Table I shows the library of ASL gates used in this work. We note this library is similar to the one given in [5] and also comparable in size to the one used in [2]. We tested three approaches in our experiments. The details of each approach are listed as below:

- **SM0:** This is the classical technology mapping algorithm when applied on ASL-Map problem as explained in Section III.
- **SM1:** This is same as SM0 except that cost is calculated accurately to account for magnet sharing in cascaded ASL gates.
- **SM2:** This is our presented framework. Compared to SM1 (which includes accurate cost computation), SM2 *also* applies various local transformations to the subject graph.

As for handling the DAG issues in the covering phase, the exact same techniques were used in SM0, SM1, and SM2. We skip the explanation of these techniques due to lack of space.

B. Analysis of Results

Table II shows the results of our experiments. For each benchmark, we report the number of nodes in its subject graph (MIG), the absolute delay and area values (in terms of magnet count) under SM0, the percentage improvement compared to SM0 under SM1 and SM2. To measure area and delay, the mapping solution generated by each approach is evaluated with the accurate cost computation, i.e., by accounting for magnet sharing in cascaded ASL gates.

Recall delay is our primary cost function during technology mapping. As can be seen, it has been improved in both SM1 and SM2. Specifically, compared to SM0, SM1 improves the delay by on-average 2.04% over 31 benchmarks, and the highest improvement reached by SM1 is 9.38%, which is in benchmark `usb_funct`. Not surprisingly, this result illustrates that just accurate computation of delay can result in improvement.

The improvements achieved by SM2 are more noticeable. Delay has been improved in SM2 by on-average 9.15% and at most 27.27% which is observed in benchmark `sasc`. This shows applying local transformations to subject graph in SM2 can achieve significant improvement in delay.

Note that the delay improvements achieved by SM1 and SM2 are consistent across all benchmarks; except one design under SM1, all benchmarks show non-negative improvements under both SM1 and SM2. Moreover, the improvements in SM2 are consistently same or better compared to SM1, across all benchmarks.

Even though our primary optimization goal is delay, we have also reported the area for each benchmark in the table. (Area is evaluated the same way for the solution generated by each approach by accounting for magnet sharing.) Interestingly, in some cases, we observed small area improvements as well. That is, for all the benchmarks under SM1 (except two that remain unchanged), the average improvement is 1.35% and maximum is 3.32%. These improvements can been observed for most benchmarks under SM2 as well, with 3.9% as the average and up to 12.69% in one benchmark. Note that, under SM2, area was slightly degraded in 3 benchmarks. *We like to note that runtime is not an issue for our framework. For SM2, the longest runtime is under 22s for a single benchmark. The runtimes of SM1 and SM0 are substantially smaller than SM2.*

		DELAY			AREA		
Benchmark	#nodes	SM0	SM1(%)	SM2(%)	SM0	SM1(%)	SM2(%)
DSP	40212	61	0.00	3.28	97707	1.09	1.90
MAC32	9326	63	-1.59	1.59	22481	0.67	2.94
MUL32	9096	60	3.33	5.00	21489	2.57	5.79
ac97_ctrl	10745	13	0.00	15.38	26100	0.33	-2.75
aes_core	20947	30	0.00	6.67	47866	1.11	5.19
comp	18493	130	7.69	15.38	43813	2.71	6.34
des_area	4186	38	5.26	10.53	10298	2.71	6.57
des_perf	67194	26	0.00	7.69	170515	2.58	4.21
diffeq1	17725	329	3.34	4.86	42410	0.87	2.06
div16	4374	174	1.72	11.49	10622	1.92	12.69
ethernet	57959	28	0.00	10.71	141500	0.22	-0.42
hamming	2071	93	2.15	3.23	5317	1.30	3.23
i2c	971	15	6.67	13.33	2560	3.32	3.98
log2	31326	327	0.92	8.26	73088	0.57	4.43
max	4210	50	0.00	18.00	10610	2.31	8.75
mem_ctrl	7143	32	3.13	6.25	17745	0.47	0.94
mult64	25772	167	0.60	4.19	60158	2.50	2.75
pci_bridge32	18603	31	3.23	9.68	49046	1.69	2.38
pci_spoci_ctrl	932	19	0.00	0.00	2347	0.21	8.05
revx	7516	235	2.13	8.51	18264	2.04	5.14
sasc	621	11	0.00	27.27	1539	1.17	-0.71
simple_spi	837	14	0.00	7.14	2068	0.00	1.55
spi	3337	32	3.13	12.50	8168	0.94	5.52
sqrt32	2156	269	1.49	9.67	5161	0.87	8.51
square	17887	61	1.64	6.56	42976	0.60	3.11
ss_pcm	397	10	0.00	0.00	1019	0.00	0.00
systemcaes	9547	39	5.13	7.69	22621	2.20	1.42
systemcdes	2453	34	0.00	14.71	5977	1.24	4.73
tv80	7397	51	3.92	15.69	17858	1.61	3.91
usb_funct	12995	32	9.38	9.38	32776	1.41	4.12
usb_phy	372	11	0.00	9.09	1054	0.66	4.46
average			2.04	9.15		1.35	3.90
max			9.38	27.27		3.32	12.69

Table II: Comparison of simulation results in three approaches.

VI. Conclusions and Future Work

This work identified specific challenges and opportunities of technology mapping with All Spin Logic (ASL) device. We proposed new techniques for taking advantages of these opportunities including making local transformations to the subject graph during the pattern matching process. Our results show that our ASL-aware technology mapping algorithm can achieve substantial improvement in delay with slight improvement in area, compared to the solution generated by classical technology mapping, hence the importance of CAD tools that are aware of the specific features of emerging nano-technologies.

References

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

[2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization. In *Design Automation Conference*, pages 1–6, 2014.

[3] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Majority-based synthesis for nanotechnologies. In *Asia and South Pacific Design Automation Conference*, pages 499–502, 2016.

[4] Michel Julliere. Tunneling between ferromagnetic films. *Physics letters A*, 54(3):225–226, 1975.

[5] Jongyeon Kim, Ayan Paul, Paul A Crowell, Steven J Koester, Sachin S Sapatnekar, Jian-Ping Wang, and Chris H Kim. Spin-based computing: device concepts, current status, and a case study on a high-performance microprocessor. *Proceedings of the IEEE*, 103(1):106–130, 2015.

[6] Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[7] Saeideh Shirinzadeh, Mathias Soeken, Pierre-Emmanuel Gaillardon, and Rolf Drechsler. Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 948–953, 2016.

[8] John C Slonczewski. Current-driven excitation of magnetic multilayers. *Journal of Magnetism and Magnetic Materials*, 159(1):L1–L7, 1996.

[9] Srikant Srinivasan, Angik Sarkar, Behtash Behin-Aein, and Supriyo Datta. All-spin logic device with inbuilt nonreciprocity. *IEEE Transactions on Magnetics*, 47(10):4026–4032, 2011.