Non-Volatile Memories in FPGAs: Exploiting Logic Similarity to Accelerate Reconfiguration and Increase Programming Cycles

Yuan Xue[1] Patrick Cronin[1] Chengmo Yang[1] Jingtong Hu[2]

[1]Department of Electrical and Computer Eng.
University of Delaware
Newark, DE 19716 USA
{xueyuan, ptrick, chengmo}@udel.edu

[2]School of Electrical and Computer Eng.
Oklahoma State University
Stillwater, OK 74078 USA
jthu@okstate.edu

***Abstract*—Non-volatile memory (NVM) technologies have been known for their advantages of large capacity, low energy consumption, high error-resistance, and near-zero power-on delay. It is expected that they will replace traditional SRAM as FPGA reconfigurable blocks. While NVMs promise FPGAs with more reconfigurable resources, lower power consumption, and higher resilience to power interruptions, they also impose two new design challenges: the slow write performance of NVMs may degrade FPGA reconfiguration speed, while their limited write endurance constrains FPGA programming cycles. To overcome these challenges, we propose a similarity driven approach to reduce reconfiguration cost in NVM-based FPGAs. When synthesizing a new design, its similarity to the design currently on the FPGA is characterized by taking both LUT contents and CLB-level topology into consideration. The reconfiguration cost minimization problem is formulated as a bipartite graph matching problem and solved optimally. Experiments on standard circuit benchmarks show that the proposed algorithms eliminate more than 57.4% of NVM writes during the reconfiguration process, thus effectively improving performance and endurance of NVM-based FPGAs.**

***Keywords*—non-volatile memory, FPGA, reconfiguration, logic similarity**

I. Introduction

Non-volatile Memory (NVM) devices such as Phase Change Memory (PCM) and Spin Transfer Torque Magnetic RAM (STT-MRAM) have been known for their advantages of large capacity, low energy consumption, high error-resistance, and near-zero power-on delay. They are promising candidates for replacing traditional dynamic and static RAMs as off-chip and on-chip memories, and even as reconfigurable blocks in Field Programmable Gate Arrays (FPGAs).

In current FPGA design, all the reconfigurable components are implemented in SRAM. Configuration information is stored in off-chip Flash memory, and loaded to on-chip SRAM when the FPGA is powered on. While the fast write/read speed of SRAM accelerates FPGA reconfiguration and performance, it unfortunately also leads to four major drawbacks. First, the large standby current of SRAM makes the FPGA suffer from high leakage power consumption which dominates total power dissipation [1]. Second, the low scalability of SRAM limits the amount of on-chip reconfigurable resources in the FPGA, which in turn constrains the size and complexity of FPGA-implementable designs. Third, SRAM is vulnerable to radiation and particles, leading to not only transient data errors but also permanent logic malfunctions when errors occur in configuration data [2]. Finally, since SRAM is volatile, upon a power outage, all the on-chip configuration information gets lost and needs to be reloaded from off-chip storage when the power is back on. This long recovery time is unacceptable in critical application domains such as military.

978–1-4673–9140–5/15/$31.00

A promising approach to overcome the limitations of SRAM is to use NVMs as FPGA on-chip storage, such as Flash-based [3], PCM-based [1], [4], and MRAM-based FPGAs [5]. These NVM-based FPGAs are expected to contain more logic and storage cells on a single FPGA chip, which consumes much less power and is more resilient to power interruptions than SRAM-based FPGAs. However, the characteristics of NVMs pose two new challenges to FPGA design. First, the slow write performance of NVMs makes reconfiguration time non-trivial. Second, the limited write endurance of NVMs constrains FPGA programming cycles. Frequently reconfiguring the FPGA may wear some building blocks heavily and cause stuck-at faults [6]. These obstacles must be overcome in order to qualify NVM-based FPGAs for industrial needs.

One effective technique to mitigate the adverse impact of NVM writes is *redundant write elimination* [7]. Before programming a block, the original content is read out and compared with the new content, and the identical part is excluded from programming. This technique was not adopted in SRAM-based FPGAs since SRAM read and write operations have similar speed and overhead. However, in NVMs, write operations are usually much slower and more costly than read operations [8]. Therefore, adopting a *read-before-write* strategy is effective to reduce reconfiguration cost, with the amount of reduction proportional to the number of NVM writes eliminated. In light of this observation, we propose to exploit the flexibilities inherent in placing lookup tables (LUTs), so as to maximize the logic similarity between two designs and minimize the cost for reconfiguring one to the other. The main contributions of this paper include:

- Two types of similarity based metrics to evaluate LUT content similarity and CLB-level topology similarity.
- Formulation of NVM write minimization as a *weighted bipartite graph matching* problem, and generation of a *min-reconfig-cost* (MRC) placement with a polynomial-time optimal algorithm.
- Incorporation of the obtained MRC placement into the Verilog-to-Routing (VTR) CAD tool [9] to enable the co-optimization of reconfiguration cost and traditional area, routing, and delay goals.

The rest of this paper is organized as follows. Section II reviews background knowledge of NVM-based FPGAs and related works. Section III outlines the technical motivation and challenges. Section IV describes the proposed similarity model and the reconfiguration optimization algorithm. Section V presents the experimental results while Section VI concludes this work.

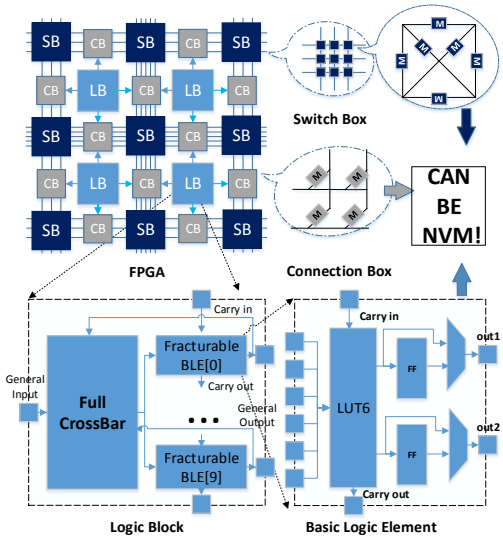

Fig. 1. Representative NVM-based FPGA Architecture

II. Technical Background

A. NVM-based FPGAs

One representative FPGA architecture is presented in Fig. 1, which consists of three types of configurable blocks: configurable logic blocks (CLB/LB), switch boxes (SB), and connection blocks (CB). Each CLB includes 10 Basic Logic Elements (BLEs), connected with a full crossbar that implements arbitrary connections across BLEs. In this architecture, the SBs, CBs, BLEs, crossbars, flip-flops and multiplexors can be implemented with NVM cells.

Various types of NVM-based FPGAs architectures have been developed or proposed recently. Flash memory is one of the most mature NVM technologies. Commercial Flash-based FPGAs have been developed by Actel [10] and Lattice [11]. However, as Flash suffers from a quite short endurance (10^5 cycles), the programming cycles of Flash-based FPGAs is quite limited. Another shortcoming is its requirement for a time-consuming erase operation before programming a block, thus slowing down the reconfiguration process in Flash-based FPGAs. In comparison, STT-MRAM uses magnetic storage elements. An FPGA using STT-MRAM as basic memory cells was proposed in [12]. In [13], STT-MRAM was used to implement LUTs. STT-MRAM delivers better scalability than SRAM, but suffers from slower performance of read and write operations, which respectively constrain the shortest clock period and reconfiguration speed in STT-MRAM based FPGAs. The third type of NVM-based FPGA uses PCM, a resistive memory regarded as one of the most promising NVMs. Researchers have proposed several PCM-based FPGA architectures: the work in [1] proposed several storage organization and management schemes for replacing SRAM with PCM, the work in [4] proposed a 3D architecture for PCM-based FPGA, while the work in [14] demonstrated the feasibility of using PCM in SBs and LUTs. Other types of NVM-based FPGAs include MRAM-based [5] and Resistance-change memories (RRAM) based [15].

B. Reconfiguration Optimizations

The conventional goal of reconfiguration optimization is to reuse parts of existing blocks on the FPGA to reduce the configuration size of a new design. Existing techniques can be grouped in two categories. One type of techniques exploit *partial reconfiguration* (PR) offered in many commercial FPGA products to support hardware reuse across different tasks [16]. Several techniques [17], [18], [19] have been proposed to perform design partitioning and/or scheduling based on coarse-grained functional information.

Another type of technique aims at maximizing similarity between two designs to reduce the reconfiguration cost. Design similarity can be checked either at the register-transfer level [20] or the gate level [21], [22]. At the higher level, the goal is to reuse entire logic components such as ALUs [20]. In comparison, gate-level approaches try to find identical LUTs for reuse. In [22], a logic remapper is proposed to detect topological similarity between two designs. In [21], similarity between two nodes is defined as the weighted sum of the position similarities between them and their adjacent nodes. Since the target platform is an SRAM-based FPGA, these works abstract LUTs as nodes without taking the similarity between LUT contents into consideration. In other words, two LUTs are considered "similar" only if they are 100% identical. Another difference is that [21] extracts design topology as an undirected graph, while the proposed work extracts topology as a directed graph to differentiate incoming and outgoing interconnects.

To summarize, most of of previous reconfiguration optimizations target SRAM-based FPGAs. They model reconfiguration overhead as the number of LUTs and/or SBs to be programmed when implementing a new design. In contrast, the proposed technique targets NVM-based FPGAs, and models reconfiguration overhead as the number of NVM cells to be programmed, which is at a much finer granularity than existing approaches.

III. Technical Motivation

While NVMs outperform SRAM in many aspects such as high density, near-zero power-on delay, and superior energy efficiency [1], their long write latency makes the reconfiguration time in NVM-based FPGA non-trivial. Taking Phase Change Memory (PCM) as an example, ITRS reports [6] indicate that its density is 35 times larger than SRAM, and its programming cycles is 10^4 times larger than Flash. However, the write speed of PCM is 500 times slower than SRAM. This will become the bottleneck in reconfiguration when PCM replaces SRAM as FPGA on-chip storage. What is more, in PCM, write is more than 8 times slower than read. This feature motivates the adoption of a *read-before-write* strategy: by reading out the original content of a reconfigurable component before programming it, the identical part in the new content can be eliminated from programming. Since read is much cheaper and faster than write, this strategy is effective in reducing reconfiguration overhead, eliminating unnecessary NVM writes, and increasing FPGA programming cycles.

It is not only the technology characteristics that dictate the need for reducing reconfiguration overhead. From the application's perspective, frequent reconfiguration is commonly required in a number of scenarios. First, limited by the size of on-chip storage, sometimes the FPGA is not large enough to hold multiple hardware accelerators simultaneously. It is therefore necessary to quickly switch between different hardware accelerators so that the application can quickly switch from one task to another. Second, many safety-critical applications use FPGAs as backup hardware resources. If an active unit fails, it is necessary to reconfigure the FPGA to conduct the functionality of the failed unit. Reconfiguration latency therefore directly determines how quick the recovery process can be. Another possible scenario is incremental design and implementation. Designers usually rely on FPGA implementations to test and refine their product. By

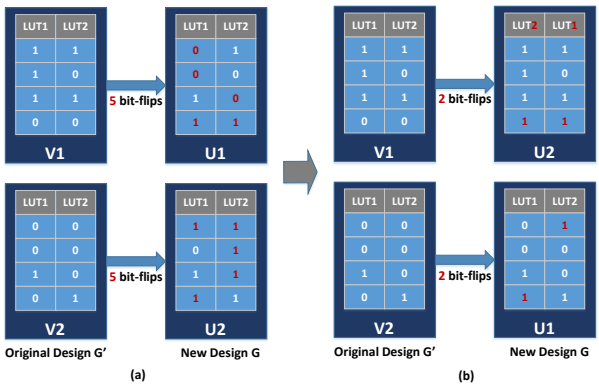

Fig. 2. Content similarity. (a) shows the original cost (10 bit-flips) for configuring $U1$ and $U2$. (b) shows by switching $U1$ and $U2$ and $LUT1$ and $LUT2$ in $U2$, the cost is reduced to 4 bit-flips.

exploiting the high similarity between two consecutive implementations, the reconfiguration overhead could be largely saved.

To summarize, both the technology characteristics and the application needs motivate the proposal of a framework to speed up reconfiguration and reduce the number of writes in NVM-based FPGAs. We achieve this goal by characterizing the logic similarity between two designs at the bit level, and minimizing the bit-flips when reconfiguring one to implement the other. More specifically, our previous work [23] considered the reconfiguration cost in NVM-based FPGAs and exploited different placement approaches to reduce such cost. However, that work only considered LUT contents and could only reduce CLB reconfiguration cost. On the other hand, as Fig. 1 shows, the majority portion of reconfigurable resources in the FPGA are SBs. Therefore, in this work we propose several similarity metrics that take not only LUT contents but also CLB-level topology into consideration, with the goal of reducing the overall reconfiguration cost of both CLBs and SBs.

IV. Proposed Reconfiguration Optimizations

To reduce the total number of NVM cells to be programmed during FPGA (re)configuration, the proposed framework identifies a *min-reconfig-cost* (MRC) mapping between two designs (or two tasks in one design), one to be configured and one currently on the FPGA. This section describes the proposed approaches for modeling similarities between two designs, identifying the MRC mapping, and integrating the MRC mapping into the VTR synthesis flow.

A. Design Similarity Characterization

Assuming that G is the design to be configured and G' is the design already on the FPGA, the goal of the proposed work is to minimize the cost for configuring G by increasing its similarity with G'. This goal will be achieved by exploiting both the flexibility in determining CLB positions and the flexibility in changing LUT positions within a CLB. Fig. 2 shows an example illustrating these two types of flexibilities. There are four CLBs, with $V1$ and $V2$ belonging to the old design G' and $U1$ and $U2$ to the new design G. Each CLB has two LUTs. Fig. 2(a) shows that by directly mapping $U1$ to $V1$ and $U2$ to $V2$, the reconfiguration cost is 10 bit-flips. In comparison, the two types of flexibilities imply that both $V1$ and $V2$ can be reconfigured to either $U1$ or $U2$, and the LUT positions in $U1$ and $U2$ can be permuted as well. Fig. 2(b) shows that by mapping $U1$ to $V2$ and $U2$ to $V1$ and switching LUT1 and LUT2 in $U2$, the reconfiguration cost is reduced to 4 bit-flips.

Algorithm 1 Content Similarity Computation

Input: All LUTs of each CLB in G and each BLK in G'
Output: Matrix C
1: **for all** $CLB(i)$ in G **do**
2: **for all** $BLK(j)$ in G' **do**
3: **for** LUT $x = 1$ to l in $CLB(i)$ **do**
4: **for** LUT $y = 1$ to l in $BLK(j)$ **do**
5: LUT-level similarity $W[x][y] \Leftarrow p/2^k$;
6: **end for**
7: **end for**
8: Best matching score $Sum_{max} \Leftarrow$ **KM** $(W[l][l])$;
9: CLB-level similarity $C_{i,j} \Leftarrow Sum_{max}/l$;
10: **end for**
11: **end for**

Exploiting the flexibilities outlined above, the proposed approach characterizes similarity between two designs combining two factors: *CLB content similarity* and *CLB-level topology similarity*. More specifically, the approach takes two sets of inputs: *post-placement* netlist of design G', as well as *pre-placement* netlist of design G. Using $CLB(i)$ to denote a CLB in G and $BLK(j)$ a CLB in G', three matrices will be computed:

- Matrix C: $C_{i,j}$ measuring *content similarity* between $CLB(i)$ and $BLK(j)$;
- Matrix T: $T_{i,j}$ measuring *topological similarity* between $CLB(i)$ and $BLK(j)$.
- Matrix S: $S_{i,j}$ measuring *similarity index*, a weighted sum of $C_{i,j}$ and $T_{i,j}$.

Each matrix has m rows and n columns, with m and n respectively denoting the number of CLBs in G and G'. Approaches for computing these three matrices are presented below.

1) CLB Content Similarity: Content similarity represents the cost for reconfiguring $CLB(i)$ to $BLK(j)$. The higher the similarity, the less the reconfiguration cost.

Given two LUTs, computing their content similarity is quite straightforward. For two k-input LUTs, if p bits are identical[1], their similarity can be measured by $p/2^k$. As a concrete example, in Fig. 2(a), the content of $LUT1$ in $V1$ is 1110[2] and the content of $LUT1$ in $U1$ is 0011. Since these two LUTs only have one identical bit, their similarity is $1/2^2 = 0.25$.

Computation of CLB-level content similarity depends not only on LUT contents but also on the positions of LUTs within a CLB. For a CLB with l LUTs, there are a total number of $l!$ permutations of LUT positions, and $C_{i,j}$ should represent the highest similarity score among these cases when mapping $CLB(i)$ to $BLK(j)$. Instead of enumerating all these possibilities, the proposed approach first computes the similarity between each LUT in $CLB(i)$ and each LUT in $BLK(j)$. Algorithm 1 shows the details. An $l \times l$ array W is used to record similarity scores between all the LUTs in $CLB(i)$ and $BLK(j)$. The similarity score between LUT x in $CLB(i)$ and LUT y in $BLK(j)$ is computed at line 5. Subsequently, a bipartite graph is constructed between the two sets of LUTs, with the LUT similarity scores represented as weights of edges. The best matching between the LUTs can be obtained with the classical *Kuhn-Munkres* (KM) algorithm [24], which computes the highest matching score Sum_{max} based on Matrix W at line 8. Then this score is normalized (i.e., divided by l) to obtain the CLB-level similarity score between $CLB(i)$ and $BLK(j)$ at line 9.

[1]Sometimes the inputs of a LUT are not fully used. Therefore certain entries in a LUT are *don't cares*, whose configuration cost will be counted as 0.

[2]1110 represents a 2-input LUT of outputs 1, 1, 1, and 0 (a NAND gate).

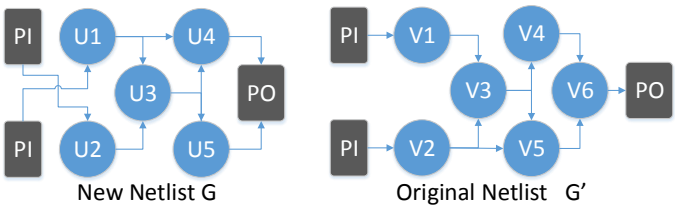

Fig. 3. CLB-level topologies.

Since the complexity of the KM algorithm is $O(l^3)$, the overall complexity of Algorithm 1 is $O(l^3mn)$.

As a concrete example, consider the computation of CLB content similarity between $U1$ and $V2$ in Fig. 2. Algorithm 1 first computes LUT content similarity $W[2][2] = \{0.75, 0.75, 0.25, 0.75\}$. Then it calls the KM algorithm, which identifies the best matching {LUT1→LUT1, LUT2→LUT2} and the corresponding score Sum_{max}=1.5. Finally, the CLB content similarity score $C_{1,2}$=1.5/2=0.75.

2) Topological Similarity: Topology similarity represents the position relation between two CLBs. As CLBs are connected, such similarity depends also on how similar their neighbors are. In light of this observation, we develop an iterative algorithm to compute topological similarity, with the k^{th} iteration depending on the result obtained in the $(k-1)^{th}$ iteration. Specifically, at each iteration k, the algorithm uses Equation (1) to compute the similarity score $T_{i,j}$. The first part of the equation accounts for self-influence, computed during the previous iteration $k-1$. The second part accounts for neighbor-influence, including both the *in* sets (nodes coming to $CLB(i)$ and $BLK(j)$) and *out* sets (output nodes of $CLB(i)$ and $BLK(j)$). The two parts are summed up with a weight α that determines their relative importance. A higher value of α implies self-influence is more important than neighbor-influence, and vice versa. In our computation, α is set to 0.75.

$$T^k_{i,j} = \alpha T^{k-1}_{i,j} + (1-\alpha)\ \frac{1}{|in(i)| + |out(i)|}$$
$$(max(\sum_{u\in in(i), v\in in(j)} T^{k-1}_{u,v}) + max(\sum_{u\in out(i), v\in out(j)} T^{k-1}_{u,v})) \quad (1)$$

In Equation (1), the *in* and *out* sets characterize the local topology of $CLB(i)$ and $BLK(j)$. To compute the weight of the *in* set, the best mapping from $in(i)$ to $in(j)$ is determined using the KM algorithm [24]. If $in(i)$ is larger than $in(j)$, the best mapping from $in(j)$ to $in(i)$ is determined instead. Weight of the *out* set can be computed in the same way. Then these two weights are added and then divided by the total number of neighbors of $CLB(i)$, represented as $|in(i)| + |out(i)|$. As a concrete example, Fig. 3 shows two designs with their CLBs, primary I/Os, and CLB-level topologies. The new design G contains five CLBs, $U1$ through $U5$, to be mapped the CLBs in the old design G', $V1$ through $V6$. Consider the computation of $T_{3,4}$, the topological similarity between *U3* and *V4*. Fig. 3 shows that $in(U3) = \{U1, U2\}$, $out(U3) = \{U4, U5\}$, $in(V4) = \{V3\}$, and $out(V4) = \{V6\}$. The neighboring-influence score is the sum of the highest mapping scores from $in(V4)$ to $in(U3)$ and from $in(V4)$ to $out(U3)$.

To initiate the computation of the T matrix, we compute $T^0_{i,j}$ considering the difference between $CLB(i)$ and $BLK(j)$ in their numbers of incoming and outgoing neighbors. This is shown in Equation (2). Clearly, if $CLB(i)$ and $BLK(j)$ has the same number of incoming neighbors, the first fraction in Equation (2) is 1. When both $in(i)$ and $in(j)$ are empty (no neighbor), the fraction is also set to 1. On the other hand, if only one of $in(i)$ and $in(j)$ is empty, the first fraction is set to 0. For $U3$ and $V4$ in Fig. 3, we have $|in(U3)|$=2, $|in(V4)|$=1, $|out(U3)|$=2, and $|out(V4)|$=1. Therefore the initial topological similarity score between $U3$ and $V4$ is $T^0_{3,4} = (\frac{1}{2} + \frac{1}{2})/2$=0.5.

$$T^0_{i,j} = \left[\frac{\min(|in(i)|, |in(j)|)}{\max(|in(i)|, |in(j)|)} + \frac{\min(|out(i)|, |out(j)|)}{\max(|out(i)|, |out(j)|)}\right]/2 \quad (2)$$

Algorithm 2 Topological Similarity Computation

Inputs: Netlists of G and G', Termination threshold V_{end}, Max loop iterations N_{max}
Output: Matrix T

```
1:  Initialize T^0_{i,j} with Equation (2)
2:  for k = 1 to N_max do
3:      δ ⇐ 0
4:      for all CLB(i) in G do
5:          for all BLK(j) in G' do
6:              if |out(i)| + |in(i)| > 0 then
7:                  Compute T^k_{i,j} with Equation (1)
8:                  δ ⇐ δ + |T^k_{i,j} − T^{k−1}_{i,j}|
9:              else
10:                 T^k_{i,j} ⇐ T^{k−1}_{i,j}
11:             end if
12:         end for
13:     end for
14:     if δ < V_end then
15:         break
16:     end if
17: end for
```

Algorithm 2 shows the proposed topological similarity computing algorithm. Equations (2) and (1) are used to initialize $T^0_{i,j}$ at line 1 and compute $T^k_{i,j}$ at line 7, respectively. A variable δ is used to keep track of the sum of the differences between two consecutive iterations. The algorithm terminates when reaching an upper-bound of iterations N_{max}, or when δ is less than a given threshold V_{end}.

3) Similarity Index: Once the two matrices C and T are computed, they can be combined to form the similarity index matrix S with a weight β:

$$S = \beta C + (1-\beta)T \quad (3)$$

The value of β should be determined based on the actual reconfiguration overhead of CLBs and switch boxes. For the FPGA platform used in our experiments, the ratio of actual configuration bits needed for CLBs and SBs is around 4:1. Therefore a higher weight is given to content similarity and β is set to 0.8.

B. Optimal Placement Identification

Once the similarity index matrix S is obtained, the algorithm starts to search for the *min-reconfig-cost* (MRC) placement of the new design. One key observation is that this problem can be converted to a *weighted bipartite matching* problem. CLBs of the two designs can be represented as the two sets of nodes in the bipartite graph, while $S_{i,j}$ is the weight of the edge from node i to node j. The graph includes $m+n$ nodes and $m \times n$ edges, with m and n respectively representing the number of CLBs in the new design G and the number of BLKs in old design.

Given the bipartite graph model, the problem of maximizing similarity between the two designs is equivalent to the problem of finding a one-to-one mapping that maximizes the sum of the weights of the matched edges. This problem can be solved

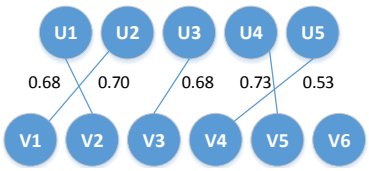

Fig. 4. Similarity based optimal bipartite graph matching

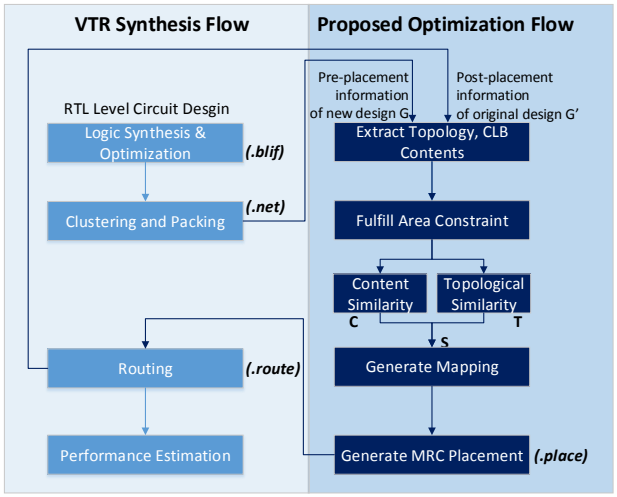

Fig. 5. Proposed reconfiguration optimization design flow

optimally in $O(m^2n)$ time with the classical *Kuhn-Munkres* (KM) algorithm [24]. As a concrete example, the bipartite graph model and the best mapping between the two designs in Fig. 3 is shown in Fig. 4. Given this mapping and the coordinates of the CLBs in the old design G', positions of each CLB in the new design G can be determined.

C. Implementation in VTR

The MRC placement identified in the last subsection should be integrated into the VTR CAD tool [9], so that the NVM reconfiguration overhead can be considered together with traditional design constraints such as area, timing, and routability. The augmented VTR synthesis flow is shown in Fig. 5.

1) Fulfilling Area Constraint: When determining the best positions of the CLBs in the new design G, the first consideration is the set of blocks a CLB could be mapped to. The proposed algorithm will not map G to the old design G' directly. Instead, it will map G to blocks within its given area. Specifically, before placement, VTR specifies an area constraint for G as well as the positions of boundary boxes. A concrete example is given in Fig. 6, which shows the boundaries of both designs. As can be seen, the old design G' is larger than the new design G, and only six out of the 10 CLBs of G' are included in the area specified for G (shown in gray). Moreover, within that area, a set of blocks are not used in G'. The proposed algorithm will map a CLB in G to either one of the six CLBs of G' (V1 through V6) or one blank block.

2) Replacing Original VTR Placement Procedure: Under the given area constraint, the proposed similarity maximization algorithm is applied. As Fig. 5 shows, the *content similarity* matrix C, the *topological similarity* matrix T, and the *similarity index* matrix S are generated with the algorithms presented before. Then, a bipartite graph is constructed based on S, and the similarity maximization problem is solved with the Kuhn-Munkres algorithm [24]. Based on the obtained G to G' mapping

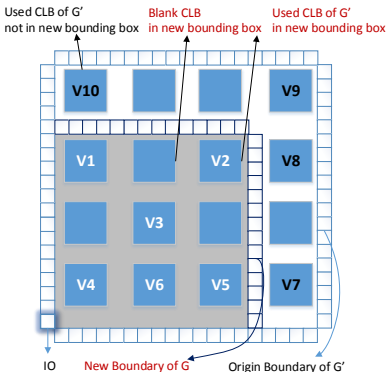

Fig. 6. Areas of new design G and old design G' (larger than G). The proposed algorithm maps G to its original area (shown in gray) including 6 CLBs of G' (V1 through V6) and 3 blank blocks.

TABLE I. BENCHMARK CIRCUITS

No	Benchmark	CLB#	LUT#
1	tseng	105	1046
2	ex5p	107	1064
3	diffeq	150	1494
4	alu4	153	1522
5	seq	175	1750
6	s298	194	1930
7	frisc	356	3539
8	spla	369	3690
9	ex1010	460	4598

and the post-placement information of G' extracted by VTR, the MRC placement of the new design G is generated. Finally, this placement is sent to VTR [9], which performs routing and applies its traditional timing and congestion optimizations.

V. EXPERIMENTAL EVALUATION

A. Experimental Setup

The experimental FPGA architecture is *k06n10* (based on Altera Stratix IV) with a 45nm fabrication library and an area-delay model offered by the VTR 7.0 CAD tool [9]. Each CLB consists of 10 BLEs, while each BLE contains one 4-input LUT and two flip-flops, as shown in Fig. 1. The proposed techniques are implemented in C++ and incorporated into the VTR synthesis flow, as shown in Fig. 5. Experiments are performed on standard MCNC benchmark circuits [25] to make sure that the obtained results are representative and convincing. Table I shows the 9 benchmarks used in the experiments, sorted in the ascending order of their sizes.

Eight evaluation cases have been performed following a "$Design_{new} \rightarrow Design_{old}$" pattern. All the mapping pairs are shown in Table II. In each case we compare the proposed MRC placement with the original placement result generated by VTR. The results are shown in three metrics: *reconfiguration cost*, *critical path delay*, and *total wire length*. Note that area data are not reported since the proposed algorithm fulfills the given area constraint and imposes no overhead.

B. Results

The first set of results in Table II shows the reconfiguration costs, collected by counting the number of bit-flips of all CLBs and SBs used to implement $Design_{new}$ in the FPGA. The fewer the bit-flips, the better the performance and the lower the overhead. Results in Table II confirm that the proposed reconfiguration optimization is very effective, as it reduces 57.4% bit-flips on average.

TABLE II. RECONFIGURATION COST IN BIT-FLIPS AND DESIGN PERFORMANCE

No	Benchmark Mapping	Reconfiguration Cost			Critical Path (ns)			Wire Length		
		Baseline	Proposed	Reduction	Baseline	Proposed	Overhead	Baseline	Proposed	Overhead
1	tseng→ex5p	17260	8188	52.6%	6.24	7.28	16.6%	8646	10640	23.1%
2	ex5p→diffeq	19352	9652	50.1%	5.11	5.18	1.4%	13255	13294	0.3%
3	diffeq→alu4	23763	10102	57.5%	6.43	7.32	14.0%	13119	16176	23.3%
4	alu4→seq	24588	9233	62.4%	4.61	4.76	3.2%	13243	15314	15.6%
5	seq→s298	28379	10502	63.0%	4.88	4.97	1.9%	20310	24209	19.2%
6	s298→frisc	34032	11782	65.4%	8.83	8.84	0.1%	12327	15637	26.9%
7	frisc→spla	62763	30891	50.8%	8.08	8.98	11.1%	40732	54071	32.7%
8	spla→ex1010	71143	30343	57.3%	6.15	7.06	14.8%	49892	64434	29.1%
Average				**57.4%**			**7.9%**			**21.3%**

Besides reconfiguration cost, routing cost and critical path delay are also important design metrics. These data are collected at the post-routing stage for the "Baseline" and "Proposed" schemes. As the baseline VTR placement is a timing-driven scheme that optimizes timing and wire length, it is expected for the proposed scheme to have slightly longer critical paths and wire lengths. As Table II shows, the proposed scheme slightly degrades the critical path by 7.9% and increases the wire length by 21.3% on average. Such overhead is insignificant compared to the 57.4% reduction in reconfiguration cost.

Overall, the results show that the proposed scheme is able to significantly reduce the cost for reconfiguring a design on NVM-based FPGA while at the same time fulfilling area constraints with acceptable delay overhead.

VI. CONCLUSIONS

This paper has proposed a similarity-based reconfiguration optimization framework for NVM-based FPGAs, aiming to mitigate the adverse impact on reconfiguration caused by NVM writes. Several similarity metrics have been proposed to characterize LUB content similarity and CLB-level topology similarity between two designs, and an algorithm has been developed to generate a placement that minimizes reconfiguration cost. The proposed optimizations have been incorporated into the VTR synthesis flow. Experimental results show that our technique is able to reduce the number of bit-flips by 57.4% when configuring a new design on NVM-based FPGA, and introduce no area overhead and acceptable performance overhead in wire length and critical path delay.

REFERENCES

[1] K. Huang, Y. Ha, R. Zhao, A. Kumar, and Y. Lian, "A low active leakage and high reliability phase change memory (PCM) based non-volatile FPGA storage element," *IEEE Transactions on Circuits and Systems*, vol. 61, Aug. 2014.

[2] H. Asadi and M. Tahoori, "Analytical techniques for soft error rate modeling and mitigation of FPGA-based designs," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 15, Dec. 2007.

[3] K. J. Han, N. Chan, S. Kim, B. Leung, V. Hecht, and B. Cronquist, "A novel flash-based FPGA technology with deep trench isolation," in *Non-Volatile Semiconductor Memory Workshop (NVSMW)*, 2007, pp. 32–33.

[4] Y. Chen, J. Zhao, and Y. Xie, "3D-NonFAR: Three-dimensional non-volatile FPGA architecture using phase change memory," in *International Symposium on Low-Power Electronics and Design (ISLPED)*, 2010, pp. 55–60.

[5] W. Zhao, E. Belhaire, V. Javerliac, C. Chappert, and B. Dieny, "Evaluation of a non-volatile FPGA based on MRAM technology," in *International Conference on Integrated Circuit Design and Technology (ICICDT)*, 2006, pp. 1–4.

[6] *International Technology Roadmap for Semiconductors*, 2013.

[7] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mosse, "Increasing PCM main memory lifetime," in *Design, Automation and Test in Europe (DATE)*, 2010, pp. 914–919.

[8] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *International Symposium on Computer Architecture (ISCA)*, 2009, pp. 24–33.

[9] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, Jun. 2014.

[10] *Proasic3 flash family fpgas handbook*, Actel, 2011.

[11] *Lattice xp2 family handbook*, Lattice Semiconductor, 2010.

[12] W. Zhao, E. Belhaire, C. Chappert, and P. Mazoyer, "Spin transfer torque (STT)-MRAM–based runtime reconfiguration FPGA circuit," *ACM Transactions on Embedded Computing Systems*, vol. 9, Oct. 2009.

[13] S. Paul, S. Mukhopadhyay, and S. Bhunia, "A circuit and architecture codesign approach for a hybrid CMOS STTRAM nonvolatile FPGA," *IEEE Transactions on Nanotechnology*, vol. 10, Feb. 2011.

[14] D. Choi, K. Choi, and J. D. Villasenor, "New non-volatile memory structures for FPGA architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, Jul. 2008.

[15] P. Gaillardon, D. Sacchetto, G. Beneventi, M. Ben Jamaa, L. Perniola, F. Clermidy, I. O'Connor, and G. De Micheli, "Design and architectural assessment of 3-D resistive memory technologies in FPGAs," *IEEE Transactions on Nanotechnology*, vol. 12, Jan. 2013.

[16] *Partial Reconfiguration User Guide*, Xilinx, March 2011.

[17] H. Tan and R. DeMara, "A physical resource management approach to minimizing FPGA partial reconfiguration overhead," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2006, pp. 1–5.

[18] R. He, Y. Ma, K. Zhao, and J. Bian, "ISBA: an independent set-based algorithm for automated partial reconfiguration module generation," in *International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 500–507.

[19] S. Hansen, D. Koch, and J. Torresen, "High speed partial run-time reconfiguration using enhanced ICAP hard macro," in *International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 2011, pp. 174–180.

[20] Z. Yang and G. Choi, "Reconfigurable multi-functioning logic structures: a case study of MMX/floating-point unit design," in *Computer Society Workshop On VLSI (ISVLSI)*, 1999, pp. 76–81.

[21] X. Shi, D. Zeng, Y. Hu, G. Lin, and O. Zaiane, "Enhancement of incremental design for FPGAs using circuit similarity," in *International Symposium on Quality Electronic Design (ISQED)*, 2011, pp. 1–8.

[22] M. Rullmann and R. Merker, "A reconfiguration aware circuit mapper for FPGAs," in *International Symposium on Parallel and Distributed Processing (IPDPS)*, 2007, pp. 1–8.

[23] Y. Xue, P. Cronin, C. Yang, and J. Hu, "Fine-tuning CLB placement to speed up reconfigurations in NVM-based FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[24] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," in *Naval Research Logistics Quarterly (NRLQ)*, 1955, pp. 29–47.

[25] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *International Symposium on Circuits and Systems (ISCAS)*, 1989, pp. 1929–1934.