Routing Path Reuse Maximization for Efficient NV-FPGA Reconfiguration *

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-based FPGAs (NV-FPGAs) are expected to replace traditional SRAM-based FPGAs to achieve higher scalability and lower power consumption. Yet the slow write performance of NVMs not only challenges FPGA reconfiguration speed and overhead but also constrains the programming cycles of FPGAs. To efficiently configure switch boxes, the majority component of an FPGA, this paper proposes a routing path reuse technique. Technical contributions include a mathematical reconfiguration cost model of routing resources, a reuse-aware routing algorithm, as well as the incorporation of the proposed algorithm into the standard VTR CAD tool. Experiments on standard MCNC benchmarks show that the proposed scheme is able to achieve as much as 40% path reuse rate and reduce as much as 34.0% configuration cost for routing resources.

I. Introduction

Field Programmable Gate Arrays (FPGAs) are widely used in many embedded systems to flexibly implement different functions according to user needs. The programmable soft logic on the FPGA delivers great systematic adaptivity which is critical to many future computing trends, such as neuromorphic computing and machine learning [1]. Unfortunately, current SRAM-based FPGAs cannot meet the increasing needs of future applications. First, SRAM-based FPGAs suffer from low scalability and cannot satisfy the dramatically increased design size requirements. Second, SRAM-based FPGAs have high leakage power [2], thus limiting their applicability to long-lasting or computation-intensive applications. Third, SRAM-based FPGAs are volatile and hence require time-consuming processes to reload the entire design to SRAM under intermittent power supplies. Another drawback is the susceptibility of SRAM to soft errors, which limits the ability of FPGAs to serve as dependable candidates in critical applications such as military and aerospace [3].

In past several years, emerging non-volatile memories (NVMs) have brought promising opportunities to improve FPGA design. NVMs have much higher scalability, lower power consumption, non-volatility, and much better error-resistance compared to SRAMs. The possibility of implementing FPGA building blocks with various NVMs, including Phase Change Memory (PCM), Resistive RAM (RRAM) and Magnetic RAM (MRAM), have already been demonstrated [4, 5, 6]. However, current FPGA design tools and flows target SRAM-based FPGAs without considering any NVM characteristic. Specifically, almost all NVMs suffer from slow write speed and short write endurance. Taking PCM as an example, a write operation takes up to $100ns$ [7], which in turn slows down FPGA (re)configuration speed and becomes the bottleneck of the entire (re)configuration process. Meanwhile, the limited endurance of NVM cells may lead to early malfunction and short device lifetime. For instance, Flash-based FPGAs only offer 500 programming cycles [8]. These limitations become extremely crucial in multi-application scenarios, wherein frequent reconfigurations between different tasks are expected on the FPGA. As a result, the need for reducing reconfiguration overhead appears be to significant.

In FPGAs, memory cells are mainly used for holding on-chip configuration information (i.e., bitstream) and implementing configurable blocks, including both lookup tables (LUTs) and switch boxes (SBs). SBs are the majority component of the FPGA as they occupy 90% of area [9], 80% of delay [10], and 85% of power consumption [11]. Accordingly, during NV-FPGA (re)configuration, SB programming would be one of the most time-consuming processes. To speed up NV-FPGA (re)configuration, we propose a path reuse algorithm during the routing stage, to reduce the cost in configuring SBs. To the best of our knowledge, this is the first paper focusing on routing optimizations in NV-FPGAs. The technical contributions include:

- A mathematical reconfiguration cost model of switch boxes;
- A path reuse maximization technique and the corresponding reuse-aware routing algorithm;
- An enhanced VTR [12] CAD flow that incorporates the proposed cost model and the routing algorithm.

The rest of this paper is organized as follows. Section II describes the background knowledge of NV-FPGAs and related works on FPGA routing optimization. Section III introduces the proposed SB reconfiguration cost model. Section IV presents the path reuse maximization approach, the reuse-aware routing algorithm and its implementation. Section V presents the experimental results collected for standard MCNC benchmarks. Finally, Section VI concludes this paper.

II. Preliminary

A. NVM-based FPGA

Figure 1 shows a representative FPGA architecture composed of three types of configurable blocks: configurable logic blocks (CLBs), connection blocks (CBs) and switch

*This work is supported by NSF grant #1527464, #1527506 and #1464429.

978-1-4673-9569-4/16/$31.00 ©2016 IEEE

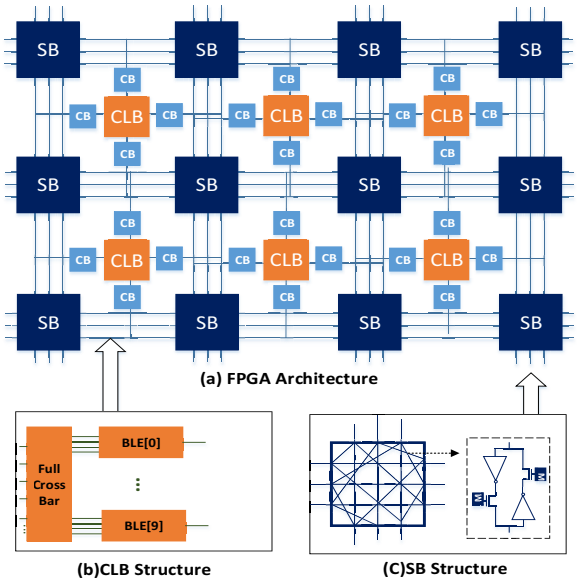

Fig. 1. A representative FPGA architecture

boxes (SBs). CLBs are used to implement logic functions. Each CLB contains a cluster of basic logic elements (BLEs), each of which is composed of one lookup table (LUT) and two flip-flops. On the input side of a CLB, a full crossbar is used to arbitrarily connect input pins to BLEs, while on the output side, one BLE output is directly connected to a CLB output pin. CBs are responsible for connecting CLBs to routing tracks locally, while SBs are used to switch between tracks and connect CLBs globally. Each SB contains multiple switches, implemented as single directional switches in commercial FPGAs [13].

Recently, a number of NVM-based FPGA architectures, such as PCM-based FPGA [14], RRAM-based FPGA [6], and 3D-stacking NV-FPGA structure [5] have been proposed. These works prove that NV-FPGAs are feasible and promising. Some industrial attempts for commercializing non-volatile FPGAs have also been made. For example, Altera [8] has developed Flash-based FPGAs.

B. Related Work on Routing Optimization

Regarding routing resource reuse and optimization, previous works tried to reduce either that routing cost or the routing time. In [15], a timing-driven routing algorithm with time-multiplexed interconnects was proposed. In [16], a *constrained delaunay triangulation* (CDT) based layout extraction was proposed to preserve routing behavior of the reference layout. And a prototyping methodology was proposed to reuse routing resources. The goal of that work was to generate a layout meeting design constraints. Consequently, [16] aims to reuse between different placements of the same layout template, while the proposed work aims to reuse between different designs under the same CAD flow. In [17], a mask-cost-aware routing problem for *engineering change order* (ECO) was discussed, by taking into account old routes for possible reuse. ECO routing at either pre-mask or post-mask stage was also discussed in [18], which tried to replace or remove redundant vias in order to increase routability of the design. These ECO routing works are at the post-silicon tape-out stage, and all the path considerations are based on the physical metal layer. In comparison, the proposed work is at the more abstract logic level and can be applied to both highly-similar and dissimilar designs.

The most closely related work is [19] which proposed a router to reduce track width by reusing routes during reconfiguration. This work is built upon a partial reconfiguration framework and relies on bitstream partition, which divides the bitstream into dynamic and static parts. To minimizing overall routing cost, this work requires the bitstreams of all the designs to be available *a priori*. In comparison, the proposed work only requires information of two designs, the one on the FPGA and the one to be reconfigured, and can be used in synthesis flow without any additional partial reconfiguration support. Furthermore, since it aims to reduce routing cost at the bit level, the achievement of more fine-grained reuse is expected.

Our previous work has proposed an algorithm [20] to fine-tune the placement procedure of VTR to reduce the reconfiguration of CLBs. In [21], an approach that jointly considers CLB content similarity and CLB-level topology similarity has been proposed to reduce NV-FPGA reconfiguration cost. Both of [20] and [21] exploit the flexibility in CLB placement. In comparsion, this work focuses on developing solutions at the routing stage to reduce SB reconfiguration cost.

III. Bit-level SB Reconfiguration Cost Model

As mentioned before, switch boxes (SBs) are the major routing resources on the FPGA which occupy 90% of area [9], 80% of delay [10], and 85% of power consumption [11]. To reduce the reconfiguration cost of SBs, the proposed work takes advantage of a *read-before-write* strategy [22] and constructs a bit-level cost model for SBs.

A. Read-before-write Strategy

One unique property of NVMs is their asymmetric read and write costs: write operations are usually longer and more energy-consuming by an order of magnitude than read operations [7, 23]. Such asymmetry motivates the adoption of a *read-before-write* strategy [22] to eliminate redundant writes and hence reduce programming latency, energy and prolong device endurance. Before writing a block, its existing content is first read out and then compared to the new content. Only the modified bits will be overwritten. This read-before-write strategy is adopted as the baseline programming strategy in the proposed scheme. Note that this strategy is not applied to traditional SRAM-based FPGAs because their read and write speed are symmetric. Therefore, eliminating redundant writes delivers no benefit for them.

To apply the read-before-write strategy to an NV-FPGA, the proposed scheme splits reads and writes into two separate stages instead of performing them in a row. Before synthesizing the new design, the contents of all the NVM cells in all the configurable resources are read out first. Then the proposed framework takes such information as inputs, performs placement and routing, and identifies the cells to reconfigure. After that, the identified cells are reconfigured to implement the new design.

B. Reconfiguration Cost Model

The proposed reconfiguration cost model is based on the Wilton Switch pattern [24], while the switches are single directional pass transistors, as shown in Figure 1(c). Each line represents a pair of NVM cells, each of which controls signal propagation from one side. The two switches can never be "on" at the same time. Clearly, the configuration of every switch can be represented by one NVM bit.

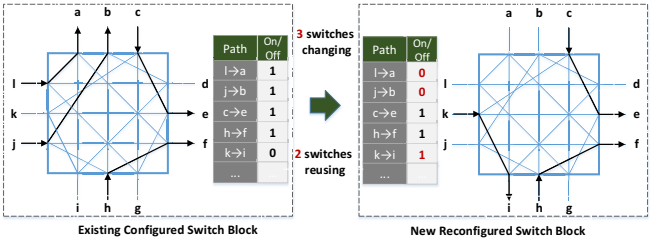

Fig. 2. SB reconfiguration cost. Black lines are "on" switches. Left SB has 5 paths, while right SB has 3. They share 2 paths in common.

As a result, if there are C channel tracks on each side of an SB and each track can connect to F other tracks, the configuration cost of one SB is $4 \times C \times F$ bits.

Given two SBs, if their configurations differ in q bits, the cost of reconfiguring one to implement the other is q. More importantly, this reconfiguration cost is directly related to the number of reusable path switches. Mathematically speaking, if two SBs i and j respectively have n_i and n_j "on" switches and they share p "on" switches in common, then the reconfiguration cost RR_{SB} can be computed using the following equation:

$$RR_{SB} = n_i + n_j - 2p \tag{1}$$

The example in Figure 2 illustrates the SB reconfiguration cost. Here, $C = 3$, $F = 3$, and therefore each SB includes $4 \times 3 \times 3 = 36$ switches. The left SB and the right SB respectively have 4 and 3 "on" switches, and they share 2 "on" switches ($c \rightarrow e$, $h \rightarrow f$) in common. As a result, it requires $4 + 3 - 2 \times 2 = 3$ 3 bit-flips (denoted in red in Figure 2) to reconfigure the left SB to implement the right SB.

The overall reconfiguration cost of all SBs, denoted as $Cost_{\text{reconfig}}$, can be computed by summing up the reconfiguration cost of each SB. Here, N_{SB} denotes the total number of SBs used by the new design.

$$Cost_{\text{reconfig}} = \sum_{k=1}^{N_{SB}} RR_{SB_k} \tag{2}$$

IV. SB Reuse-aware Routing

This section discusses the path reuse maximization problem and presents the proposed reuse-aware routing algorithm with its implementation. The proposed work focuses on path-level reconfiguration optimization because of two reasons. First, as illustrated in Equation (1), increasing path reuse directly translates to the reduction in bit-level reconfiguration cost. Second, by exploiting the flexibility inherent in placement and routing, both the position of starting pin and the direction of a path can be adjusted to reduce the reconfiguration cost. In comparison, bitstream information, although it directly reflects reconfiguration cost, does not provide any insight regarding where the bottleneck of reconfiguration cost is and what kind of flexibility can be exploited to reduce the cost.

A. Reusable Path Recognition

In this paper, a *path* is defined as a single source-to-sink connection. A *net* is the set of paths that share the same source node. As an example, the grid in Figure 3 represents the FPGA, while the circles are CLBs and there is an SB at every cross point in the grid. Initially there are two nets

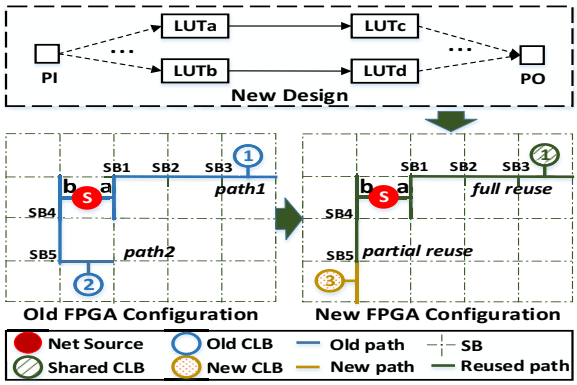

Fig. 3. Paths and reuse types. Net source CLB "S" contains BLE_a and BLE_b. By mapping LUT_b to BLE_a and LUT_a to BLE_b, "path 1"=$\{CLB_s(a), SB_1, SB_2, SB_3, CLB_1\}$ is "fully reusable", and "path 2"=$\{CLB_s(b), SB_4, SB_5, CLB_3\}$ is "partially reusable".

on the FPGA: one starts from BLE_a in CLB "S" (in red) and ends at CLB_1, while the other starts from BLE_b in "S" and ends at CLB_2.

Formally, an existing path on the FPGA can be characterized as P=$\{CLB_i(u), SB_r....SB_d, CLB_j\}$. Its starting point $CLB_i(u)$ is the (output) pin u in the source CLB_i, while its ending point is sink CLB_j. Note that the exact input pin in the sink CLB is ignored because as Figure 1(b) shows, within each CLB, a full crossbar is used to arbitrarily connect input pins to BLEs. $SB_r....SB_d$ represents the set of SBs that path P goes through, with SB_r and SB_d respectively denoting the first SB and the last SB.

The proposed framework considers two types of path reuse. First, an existing path P on the FPGA is *fully reusable* if there exists a path P' in the new design who shares the same starting pin and ending CLB as P. While this type of reuse can be easily recognized, the reuse definition is also strict. For two dissimilar designs, the number of exactly matching pairs is quite limited.

To exploit more path reuse potential, the proposed framework additionally considers *partial reuse*. Specifically, an existing path P is considered *partially reusable* if there exists a path P' in the new design who shares the same starting pin $CLB_i(u)$ and last SB SB_d as P. The reason for checking only the last SB for partial reuse is twofold. First, it delivers maximum benefit since all the SBs on the path except for the last one can be reused. Second, searching for matches of the last SB only involves one-stage back track, thus minimizing path-matching overhead.

As a concrete example, in Figure 3, "path1" is *fully reusable* since the new configuration contains one path that shares the same starting point a of CLB "S" and ending point CLB_1 with "path1". In comparison, "path2" is *partially reusable* since it shares the same starting point b of "S" and last SB SB_5 with another path in the new net, but "path2" ends at CLB_3 and the new path ends at CLB_4.

B. Path Reuse Maximization

As reusable paths require the share of the same starting and ending CLBs, path reusability is determined by CLB positions which are in turn determined by placement. Furthermore, as each BLE is directly connected to a CLB output pin, the BLE positions within a CLB will directly determine the starting point of a path.

During FPGA synthesis, LUTs are grouped into logic blocks which are mapped to CLBs at the placement stage.

Within each CLB, LUTs are usually mapped to BLEs arbitrarily. This implies that the mappings between LUTs and BLEs can be manipulated to increase the number of reusable paths. Taking Figure 3 as an example, assume that LUT_a and LUT_b in the new design are placed in CLB_s, while LUT_c and LUT_d are placed in CLB_2 and CLB_1, respectively. If LUT_a is mapped to BLE_b, the $LUT_a \rightarrow LUT_c$ connection starts at pin b and "path1" is fully reusable. Meanwhile, if LUT_b is mapped to BLE_a, the $LUT_b \rightarrow LUT_d$ connection starts at pin a and "path2" is partially reusable. On the other hand, if we map LUT_a to BLE_a and LUT_b to BLE_b, no path can be reused.

Given this observation, we propose a post-placement/pre-routing optimization which increases path reuse by manipulating the LUT-to-BLE mapping in each CLB. Mathematically, if a CLB contains n BLEs, there are $n!$ possibilities to map n LUTs to n BLEs. The best mapping is the one that maximizes the overall benefit delivered by all the reusable paths. To efficiently explore the design space of $n!$ possibilities, we develop a *bipartite graph mapping* approach.

The proposed bipartite graph contains $2n$ nodes and n^2 edges. The two set of nodes respectively represent the LUTs and BLEs, while the weight of edge RS_{ij} $(1 \leq i, j \leq n)$ represents SB reuse benefit, measured by the number of reusable switches when mapping LUT_i to BLE_j. Since this graph needs to be constructed at the pre-routing stage but the exact number of reusable switches is not available until the post-routing stage, it is necessary to estimate the amount of reuse benefit. The proposed approach uses the wire-length of the reused path as an estimation. Figure 3 shows a concrete example. The wires reused in the new configuration are shown in green. By counting the number of grids, we can get that $RS_{ab} = 2$ (i.e., the reuse benefit of "path1") and $RS_{ba} = 4$ (i.e., the reuse benefit of "path2"). Once the weight of each edge RS_{ij} is obtained, the best LUT-to-BLE mapping is the one that maximizes $\sum_{i=1}^{n} RS_{ij}$. This mapping can be identified optimally in polynomial time using the classical Kuhn–Munkres (KM) algorithm [25].

C. Reuse-aware Routing

The proposed reuse-aware routing algorithm is based on VTR *pathfinder* routing algorithm, which aims at generating a feasible routing plan with good timing results. Given the netlist of a design, the router performs global routing iteratively and terminates upon reaching either a feasible routing plan or the upper bound number of iterations. In each iteration, the router first ranks all the nets based on their numbers of sinks, and then routes them one by one based on the ranking. When routing a net, its routing structure in the last iteration is ripped up first. Then the *pathfinder* algorithm [12] is used to search for a source-to-sink path and add it to the routing tree. This process continues until all the sinks of the net are found and routed. After that, the congestion and cost of the routing resources consumed by the net are updated. Note that the cost is a function of timing, and the time-driven router always prioritizes the use of resources of the lowest costs.

Same as the original VTR router, the proposed router is also timing-driven and adopts an iterative procedure. However, to maximize path reuse, the proposed algorithm makes three major changes, as described below. Pseudo code of the algorithm is shown in Algorithm 1, while variable definitions are summarized in Table I.

TABLE I
VARIABLES USED IN ROUTING ALGORITHM 1

Variables	Descriptions
$Candidate\_set$	Routing resource candidates for Pathfinder
ϵ	Threshold for reusable path relaxation
NUM_{reuse_path}	Total number of reusable paths
NUM_{net}	Total number of nets in the new design
$Nets[n]$	Net n in the new design
$Order[i]$	Rank of net i
$path[n][j]$	Path j in net n
$rrpath[i]$	Reusable path i
rr_node	Routing resource, either free or dedicated to a net
$sink\_num[n]$	Number of unrouted sinks in net n
$Trace[n]$	Routing result of net n
$T_{rrpath[i]}$	Timing delay of $rrpath[i]$

Algorithm 1 Path Reuse Aware Routing Algorithm

Input: Netlist, Placement, Reusable paths, Relax threshold ϵ
Output: Routing results

```
1: Mark all routing resources rr_node.type = general;
2: ********1st STAGE: FIX REUSABLE PATH********
3: for i = 0 to NUM_reuse_path increment by 1 do
4:     for n= 0 to NUM_net increment by 1 do
5:         if rrpath[i] ⊆ Nets[n] then
6:             Mark its corresponding rr_node.type = n; //dedicated
7:             Mark rrpath[i] ⊆ Trace[n];
8:         end if
9:     end for
10: end for
11: Rank all nets based on their numbers of unrouted sinks;
12: for loop = 0 to Iteration_limit increment by 1 do
13:     *****2nd STAGE: ROUTE UNFIXED PATH*****
14:     for i= 0 to NUM_net increment by 1 do
15:         n = Order[i]; //Select the net with maximum sink_num
16:         for j = 0 to sink_num[n] increment by 1 do
17:             if path[n][j] ≠ reusable path then
18:                 if rr_node.type == (general | n) then
19:                     Add rr_node to Candidate_set;
20:                 end if
21:                 Pathfinder(Trace[n],path[n][j],Candidate_set);
22:             end if
23:         end for
24:     end for
25:     Perform timing analysis and update path criticality values;
26:     *****3rd STAGE: RELAX REUSABLE PATH*****
27:     for i = 0 to NUM_reuse_path increment by 1 do
28:         if T_rrpath[i] > (1 − ε) × T_critical_path then
29:             Mark its corresponding rr_node.type = general;
30:             Remove rrpath[i] from the reuse list;
31:         end if
32:     end for
33:     if no congestion exist then
34:         Exit loop;
35:     end if
36: end for
```

The first major change is the partition of routing resources. Based on the bipartite graph mapping, routing resources are divided into two categories: *dedicated* and *general*. SBs on those reusable paths are *dedicated* to those matching paths in the new design, while the remaining SBs are *general* resources that can be used to implement any path. In Algorithm 1, all the routing resources, denoted as *rr_node*, are marked as *general* initially (line 1). The resources on reusable path i are marked as *dedicated* to net n (line 6). When routing a net containing a reusable path, the router prioritizes the resources dedicated to it (line 18), thus preserving the reusable paths identified through bipartite graph mapping to the maximum extention.

Second, the routing order of paths belonging to the same net is also revised. The proposed algorithm first fixes reusable paths in phase 1 (line 7). Then, in phase 2, the algorithm ranks nets based on their numbers of unrouted sinks (line 11). Within each net routing subroutine, it routes the remaining paths using the *pathfinder*

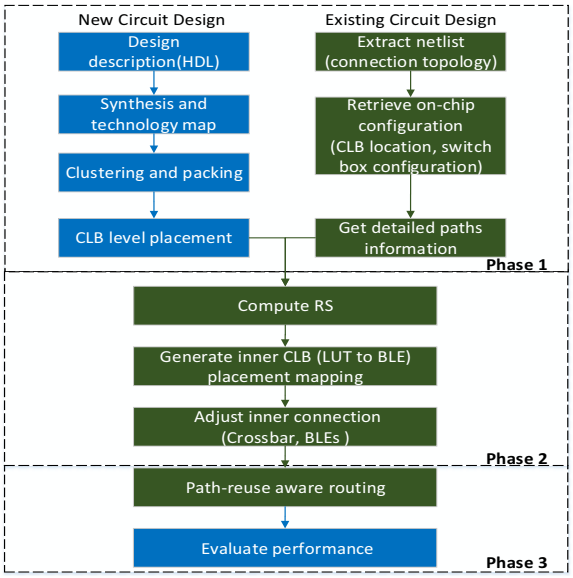

Fig. 4. Proposed CAD Synthesis Flow. The blue blocks are original steps on the VTR flow, while the green blocks are augmented steps.

algorithm, which selects resources from *Candidate_set* to route $path[n][j]$ and then adds the path to $Trace[n]$, the routing result of net n (line 21).

Finally, the proposed algorithm balances design timing and path reuse by selectively rerouting long reusable paths. Specifically, in phase 3, the algorithm checks if the delay of a reusable path is close to that of the critical path (i.e., $T_{rrpath[i]} > (1-\epsilon) \times T_{critical\_path}$). Those paths are considered potentially critical and will be removed from the reusable list (line 30), allowing them to be re-routed in the next iteration using other SBs to potentially shorten its delay. Meanwhile, its previous routing resources will be marked as *general* instead of *dedicated* (line 29). The ϵ is a user defined value. A higher threshold imposes stricter timing constraints, resulting in more reusable paths being relaxed. In our experiments, ϵ is set to 1%.

D. Enhanced Synthesis Flow

The proposed path reuse maximization technique is incorporated into an enhanced VTR synthesis flow shown in Figure 4. The blue blocks are steps in the original VTR synthesis procedure, while the green blocks are steps augmented or revised to maximize path reuse.

Overall, the flow shown in Figure 4 can be divided into three phases. In Phase 1, the new design is synthesized down to the CLB-level placement stage and its CLB-level netlist is obtained. Meanwhile, the existing configuration on the FPGA is read out to retrieve the connection topology, switch box configuration and detailed path information of the old design. In Phase 2, the path reuse matrix *RS* is computed based on the design configurations extracted in Phase 1. Then the bipartite graph matching procedure is invoked to identify the best LUT-to-BLE mapping. Based on mapping results, CLB pin positions and inner CLB structures are adjusted to complete fine-grained intra-CLB placement. Finally, in Phase 3, the reusable path information is delivered to the router, which performs reuse-aware routing following the procedure shown in Algorithm 1. At the end, the area, delay, and power consumption of the new design will be estimated.

TABLE II
BENCHMARK CIRCUITS

No	Benchmark	CLB#	LUT#	Net#	Track Width
1	bigkey	170	1699	829	38
2	s298	194	1930	683	30
3	frisc	356	3539	1859	56
4	ellipti	361	3602	1950	48
5	spla	369	3690	1866	56
6	pdc	458	4575	2292	66
7	ex1010	460	4598	2668	62
8	s38584	635	6177	3697	44
9	s38417	636	6042	3613	42
10	clma	837	8365	4981	66

TABLE III
PATH REUSE RESULTS

Group	Total Path#	DIR Path Reuse			Proposed Path Reuse		
		Full	Part	%	Full	Part	%
TP1	1597	101	244	21.6%	187	450	39.9%
TP2	2543	33	35	2.7%	114	388	19.7%
TP3	5969	40	133	2.9%	126	744	14.6%
TP4	5057	77	162	4.7%	191	814	19.9%
TP5	6641	59	157	3.3%	128	551	10.2%
TP6	8564	129	373	5.9%	239	796	12.1%
TP7	8854	76	169	2.8%	159	1066	13.8%
TP8	7330	131	335	6.4%	347	1779	29.0%
TP9	8291	51	276	3.9%	186	1293	17.8%
Ave				6.0%			19.7%

V. EXPERIMENTAL EVALUATION

This section evaluates the proposed path reuse scheme in term of its effectiveness in reducing reconfiguration costs and design performance.

A. Experimental Setup

The proposed work is based on VTR7.0 [12] CAD tool. The experimental FPGA architecture is Altera Stratix IV whose delay model and cell library are provided in the VTR toolkit. Each CLB contains 10 BLEs. The evaluation set consists of the 10 largest MCNC benchmarks. Table II reports some information of these benchmarks, sorted in ascending order of the design size. In our experiments, 9 test pairs (TP) are studied. TP_i ($1 \leq i \leq 9$) considers the i^{th} benchmark as the new design and the $(i+1)^{th}$ benchmark as the existing design on the FPGA.

The following three different schemes are compared in our experiments: "Baseline" is original VTR placement and routing. No path reuse is considered. "DIR" employs the proposed reuse-aware routing algorithm but does not maximize path reuse with bipartite graph matching. Instead, LUT_i is directly mapped to BLE_i. The reuse relax threshold ϵ is set to 1%. "Proposed" performs not only reuse-aware routing but also bipartite graph matching. Again, ϵ is set to 1%. To ensure fairness in comparison, the read-before-write strategy described in Section A is adopted consistently in all three schemes.

B. Experimental Results

Table III presents path reuse results for "DIR" and "Proposed". Although "DIR" does not optimize path reuse, the reuse-aware routing algorithm is still able to reuse 6.0% paths on average, either fully or partially. In comparison, "Proposed" achieves much more path reuses, at an average rate of 19.7% and up to 39.9% (for TP 1). This confirms the effectiveness of the proposed bipartite graph matching technique in exploiting LUT-to-BLE mapping flexibilities to maximize path reuse possibilities. Finally, a comparison between "Full" and "Partial" columns confirm that the latter consistently makes a greater contribution to the total reuse rate.

Table IV presents the overall reconfiguration costs of all the switch boxes, computed using Equation (2). Compared

TABLE IV
SWITCH BOX RECONFIGURATION COST

Group	Baseline Cost	DIR Cost		Proposed Cost	
		Value	Reduction	Value	Reduction
TP1	4655	3190	31.5%	3074	34.0%
TP2	8336	7844	5.9%	6753	19.0%
TP3	17147	16231	5.3%	13749	19.8%
TP4	17641	16708	5.3%	12747	27.7%
TP5	21589	19812	8.2%	16029	25.8%
TP6	22947	20923	8.8%	18032	21.4%
TP7	18453	17457	5.4%	14778	19.9%
TP8	20428	18685	8.5%	13878	32.1%
TP9	28688	26022	9.3%	22623	21.1%
Ave			9.8%		24.5%

TABLE V
CRITICAL PATH DELAY

Group	Baseline CP (ns)	DIR CP (normalized)	Proposed CP (normalized)
TP1	2.58	0.962	1.000
TP2	8.66	1.020	1.008
TP3	10.55	1.015	1.000
TP4	7.68	1.052	1.008
TP5	5.93	1.036	1.081
TP6	6.40	1.030	1.090
TP7	5.67	1.022	1.030
TP8	5.35	0.987	1.012
TP9	6.17	1.076	1.088
Ave		1.020	1.035

to "Baseline" that does not consider any path reuse, "DIR" slightly reduces SB reconfiguration cost by 9.8% on average, while "Proposed" is able to reduce the cost by 24.5%. Again, the maximum reduction of 34.0% is achieved for TP1. More importantly, for each TP, the reduction rate in Table III is strongly correlated with the reuse rate in Table IV, but the two rates are not exactly the same. The difference is due to the variation in path length. As reused paths are of different length and involve different numbers of SBs, their contributions to the reduction in bit-level reconfiguration cost also vary.

Since the proposed scheme adjusts both CLB routing and LUT positions within each CLB, it will impact critical path delay, as reported in Table V. "Baseline" is a purely timing-driven routing algorithm. "DIR" adopts the proposed reuse-aware routing algorithm to fix certain paths during routing. As a result, it slightly increases critical path delay by 2.0% on average. "Proposed" not only performs reuse-aware routing but also adjusts LUT positions within each CLB, and hence increases critical path delay by 3.5%. Note that the amount of delay increased is not proportional to the number of reused paths shown in Table III, but instead determined by the reuse relax threshold ϵ, which is set to 1% in these tests. By adjusting this threshold, the designer can flexibly balance path reuse rates and design performance. Overall, considering the significant benefits in reusing paths and reducing reconfiguration costs, the slight degradation in performance is acceptable.

VI. CONCLUSION

In this paper, we have proposed an approach to reuse routing paths in order to reduce the reconfiguration cost of switch boxes, which are the major component of an NV-FPGA. This in turn mitigates both the endurance limitation introduced by NVMs and the reconfiguration bottleneck due to slow and costly NVM writes. An SB reconfiguration cost model is first established and then used to guide path reuse. Mechanisms to recognize reusable paths and furthermore to maximize the benefits are introduced. Finally, the VTR CAD flow is enhanced to incorporate the proposed reuse-aware routing algorithm. Experimental results confirm that the proposed scheme is able to deliver as much as 40% path reuse rate and 34% reduction in SB reconfiguration cost, thus confirming its potential in promoting the popularity and practicality of NV-FPGAs.

REFERENCES

[1] X. Zhu and Y. Chen, "Improved FPGA implementation of probabilistic neural network for neural decoding," in *International Conference on Apperceiving Computing and Intelligence Analysis (ICACIA)*, Dec. 2010, pp. 198-202.

[2] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 26, Feb. 2006.

[3] *Understanding soft and firm errors in semiconductor devices*, Actel, Dec. 2002.

[4] W. Zhao, E. Belhaire, C. Chappert, B. Dieny, and G. Prenat, "TAS-MRAM-based low-power high-speed runtime reconfiguration (RTR) FPGA," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, Jun. 2009.

[5] Y. Chen, J. Zhao, and Y. Xie, "3D-NonFAR: Three-dimensional non-volatile FPGA architecture using phase change memory," in *International Symposium on Nanoscale Architectures (NANOARCH)*, Aug. 2010, pp. 55-60.

[6] J. Cong and B. Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," in *International Symposium on Low-Power Electronics and Design (ISLPED)*, Aug. 2011, pp. 1-8.

[7] *International technology roadmap for semiconductors*, 2013.

[8] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation (FTEDA)*, vol. 2, Feb. 2008.

[9] V. George, "Low energy field-programmable gate array," *Ph.D. dissertation, University of Toronto*, 2000.

[10] A. DeHon, "Reconfigurable architectures for general-purpose computing," *Ph.D. dissertation, MIT*, 1996.

[11] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 12, Apr. 2004.

[12] J. Luu et al, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, Jun. 2014.

[13] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *International Conference on Field-Programmable Technology (FPT)*, Dec. 2004, pp. 41-48.

[14] 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 (TCAS)*, vol. 61, Sep. 2014.

[15] H. Liu, X. Chen, and Y. Ha, "An architecture and timing-driven routing algorithm for area-efficient FPGAs with time-multiplexed interconnects," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2008, pp. 615-618.

[16] C.-Y. Chin, P.-C. Pan, H.-M. Chen, T.-C. Chen, and J.-C. Lin, "Efficient analog layout prototyping by layout reuse with routing preservation," in *International Conference on Computer-Aided Design (ICCAD)*, Nov. 2013, pp. 40-47.

[17] H.-A. Chien et al, "Mask-cost-aware ECO routing," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Mar. 2014, pp. 1-4.

[18] H.-A. Chien and T.-C. Wang, "Redundant-via-aware ECO routing," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2014, pp. 418-423.

[19] B. Al Farisi, K. Bruneel, and D. Stroobandt, "Staticroute: A novel router for the dynamic partial reconfiguration of FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2013, pp. 1-7.

[20] 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)*, Sep. 2015.

[21] Y. Xue, P. Cronin, C. Yang, and J. Hu, "Non-volatile memories in FPGAs: Exploiting logic similarity to accelerate reconfiguration and increase programming cycles," in *International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct. 2015.

[22] A. Ferreira et al, "Increasing PCM main memory lifetime," in *Design, Automation and Test in Europe (DATE)*, Mar. 2010, pp. 914-919.

[23] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, "Flash memory cells-an overview," *Proceedings of the IEEE*, vol. 85, Aug. 1997.

[24] S. Wilton, "Architectures and algorithms for field programmable gate arrays with embedded memory," *Ph.D. dissertation, UC Berkeley*, 1997.

[25] H. Kuhn, "The hungarian method for the assignment problem," in *Naval Research Logistics Quarterly (NRLQ)*, Mar. 1955, pp. 83-97.