Adaptive Selection and Clustering of Partial Reconfiguration Modules for Modern FPGA Design Flow

KANG ZHAO, Beijing University of Posts and Telecommunications, China
YUCHUN MA and RUINING HE, Tsinghua University, China
JIXING ZHANG and NING XU, Wuhan University, China
JINIAN BIAN, Tsinghua University, China

Dynamic Partially Reconfiguration (DPR) on FPGA has attracted significant research interest in recent years since it provides benefits such as reduced area and flexible functionality. However, due to the lack of supporting synthesis tools in the current DPR design flow, leveraging benefits from DPR requires specific design expertise with laborious manual design effort. Considering the complicated concurrency relations among various functions, it is challenging to select appropriate **Partial Reconfiguration Modules (PR Modules)** and cluster them into proper groups with a proper reconfiguration schedule so that the hardware modules can be swapped in and out correctly during the run time. Furthermore, the design of PR Modules also impacts reconfiguration latency and resource utilization greatly. In this paper, we propose a Maximum-Weight Independent Set model to formulate the PR Module selection and clustering problem so that the original manual exploration can be solved efficiently and automatically. We also propose a step-wise adjustment configuration prefetching strategy incorporated in our model to generate optimized reconfiguration schedules. Our proposed approach not only supports various design constraints but also can consider multiple objectives such as area and reconfiguration delay. Experimental results show that our approach can optimize resource utilization and reduce reconfiguration delay with good scalability. Especially, the implementation of the real design case shows that our approach can be embedded in Xilinx's DPR design flow successfully.

CCS Concepts: • **Hardware → Electronic design automation**; **Methodologies for EDA**; **Software tools for EDA**;

Additional Key Words and Phrases: Dynamic partially reconfiguration, partial reconfiguration module, independent set-based model, prefetching

ACM Reference format:
Kang Zhao, Yuchun Ma, Ruining He, Jixin Zhang, Ning Xu, and Jinian Bian. 2023. Adaptive Selection and Clustering of Partial Reconfiguration Modules for Modern FPGA Design Flow. *ACM Trans. Reconfig. Technol. Syst.* 16, 2, Article 27 (March 2023), 24 pages.
https://doi.org/10.1145/3567427

This work was supported in part by National Key R&D Program of China (2022YFB2901100).
Authors' addresses: K. Zhao (corresponding author), Beijing University of Posts and Telecommunications, Beijing, China, 100083; email: zhaokang@bupt.edu.cn; Y. Ma, R. He, and J. Bian, Tsinghua University, Beijing, China, 100084; emails: myc@tsinghua.edu.cn, ruining.he@mails.tsinghua.edu.cn, bianjn@tsinghua.edu.cn; J. Zhang and N. Xu, Wuhan University of Technology, Wuhan, China, 430070; emails: jixing.zhang@whut.edu.cn, xuning@whut.edu.cn.

1936-7406/2023/03-ART27 $15.00
https://doi.org/10.1145/3567427

1 INTRODUCTION

In recent years, advances in FPGA technology have enabled reconfigurable computing to become viable and be used in end-products. This empowers application developers to design and use their own hardware accelerators to significantly increase the speed of algorithms by using reconfigurable hardware. In real-time systems, tasks must meet their constraints and deadlines. To guarantee that no deadlines are missed, the **worst-case execution time (WCET)** of tasks needs to be determined statically. Moving functionality from software into hardware can reduce the WCET. Furthermore, timing-analysis of algorithms in hardware may be easier to perform than the WCET analysis of software solutions [1].

However, one of the issues in using FPGAs is that the size of the hardware accelerators is limited by the available resources, especially considering that the FPGA cost is very sensitive to its size. The **dynamic partial reconfiguration (DPR)** feature offered by modern FPGAs [2] can be used to overcome this limitation, by enabling run-time reconfiguration of selected regions on the FPGA. This allows a more efficient utilization of FPGA resources since the hardware accelerators that are required only for limited amounts of time can be replaced when the functionality implemented in these regions is no longer required.

This promising feature of an FPGA is the ability to reuse the same hardware for different tasks at different phases of application execution. Moreover, the tasks can be swapped on the fly while another untouched part of the hardware continues to operate. This DPR technique is supported by modern FPGA architectures like Xilinx's Virtex series with DPR design flow EAPR [3]. Though the DPR technique has attracted significant interest from both the industry and academic regions, current design tools still impose several limitations. The designer must manually define how to partition the designs into the static part and **Partial Reconfiguration Modules (PR Modules)**, requiring detailed knowledge of the FPGA architecture. The designer also must decide how to cluster the PR Modules into **Partially Reconfigurable Regions (PR Regions)** so that each group of PR Modules can be loaded or unloaded within their corresponding PR Region dynamically. Given a PR Module combination scheme, the designer also needs to plan the reconfiguration schedule of different PR Module clusters during execution. A straightforward way to accomplish this is to swap in an outside PR Module whenever it needs to be activated. But the intervals between PR Modules sharing a PR Region allow the prefetching of the configuration bit-stream to reconfigure PR Regions in advance. The reconfiguration penalty imposed by DPR can be saved if a proper prefetching strategy is employed to make full use of these intervals.

The lack of automated PR Module design tools impedes the widespread usage of DPR technology. The major challenges include: (1) To guarantee the feasibility of the DPR designs, the process of PR Module selection and clustering requires a laborious analysis that involves the time-based mutual exclusion relationship among hardware modules. What's more, the design of PR Modules heavily influences the trade-off between the cost of reconfiguration latency and resource utilization. (2) To provide better solutions with optimized reconfiguration latency, the designer needs to come up with an efficient configuration prefetching strategy. Moreover, when exploring candidate PR Module combination schemes, the prefetching strategy also needs to be incorporated into the PR Module selection and clustering model to evaluate each candidate scheme.

Without any supporting synthesis tool to aid the analysis, a system with a large number of candidate PR Modules could be a big challenge. It is very difficult for designers to make full use of DPR advantages and the poor design ability may lead to long development cycles or even design failure. This paper proposes an adaptive approach for PR Modules selection and clustering with configuration prefetching integrated. Instead of enumerating all the possible combinations, we analyze the non-concurrency relationships among candidate PR Modules and transfer the original problem into a standard **Maximum-Weight Independent Set Problem (MWISP)** with design

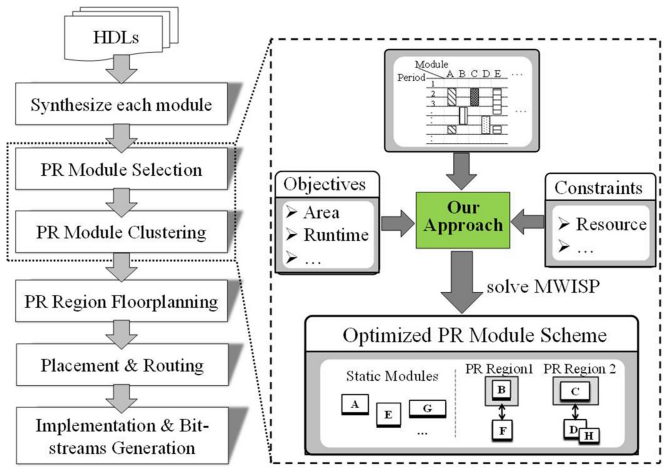

Fig. 1. Diagram of Xilinx's EAPR design flow and our approach (highlight part).

constraints. The key challenge is how to obtain a good selection and clustering results to satisfy delay and resource constraints. Our key contributions of this paper include:

- **Formal representation:** Selection and clustering of PR Modules are logically two sequential steps and it is not trivial to find an appropriate formal model to represent it. In this paper, according to the analysis of the time-based non-concurrency relationship among candidate PR Modules, we propose the model of **Partial-Vertex Coloring Problem (PVCP)** to formally represent the selection and clustering of PR Modules. PVCP can represent the two steps in a fairly concise way.
- **Independent set-based optimization algorithm:** To find the best PR Module combinations represented by the partial vertex coloring, we transfer this Partial-Vertex Coloring Problem (PVCP) to a **Maximum-Weight Independent Set Problem (MWISP)** with constraints. Therefore, multiple optimization objectives such as resource minimization and reconfiguration delay minimization can be accomplished by solving an MWISP using well-studied algorithms.
- **Step-wise adjustment strategy for configuration prefetching:** During the optimization process, we also incorporate a configuration prefetching strategy to further improve the performance of the DPR implementation of the application. In this paper, we propose a step-wise adjustment strategy to improve the total reconfiguration delay imposed by employing DPR on a normal application.
- **Support for modern DPR design flow:** Our approach can be embedded into modern DPR design flow smoothly since it can replace the manual PR Module selection and clustering task in practical design flow (as shown in Figure 1). By supporting traditional module-based high-level synthesis tools such as Mentor Graphics Catapult® C Synthesis, our approach successfully fills the gap between high-level synthesis and current DPR design flow, so that even the application specified with a high-level programming language such as C/SystemC can be synthesized into partially reconfigurable hardware implementation.

The remainder of this paper is structured as follows. Section 2 presents some related works. Section 3 reports the problem description. In Section 4, we discuss the formal representation of PR Module selection and clustering problem as a Partial-Vertex Coloring Problem (PVCP).

In Section 5, we propose an independent set-based algorithm to solve the PVCP problem. In Section 6, we discuss how to optimize common design objectives with configuration prefetching integrated. The experimental results on benchmarks are given in Section 7. We conclude our current work in Section 8.

2 PREVIOUS WORKS

Some studies discuss DPR design flow such as [4] and [25]. In [4], Chris Conger et al. discussed the DPR design framework in detail for special-purpose DPR systems and multipurpose DPR systems, respectively. However, no solution to the selection and partitioning of PR Modules is provided in their paper. In [25], a DPR design flow that tried to automate Xilinx's DPR design flow was proposed. However, they also didn't put forward how to select and cluster PR Modules effectively. Due to the lack of supporting tools, most of the current DPR designs are limited to only simple designs when non-concurrency constraints among alternative PR Modules can be derived directly from the application background [12, 27].

In [5] and [6], floorplanning methods for partial reconfiguration were described. But they still require the designer to provide the PR Module partitioning schemes manually. The authors of [27] proposed to efficiently allocate PR Regions for several given groups of PR Modules. But they mainly target a specific type of application with several different execution modes and PR Modules must be selected and partitioned into groups beforehand.

In recent years, studies that focus on **Hardware-Software (HW-SW)** partitioning and scheduling for architectures with partial dynamic reconfiguration ability have been proposed [14–16]. In [14], the authors proposed a placement-aware HW-SW partitioning heuristic based on the well-known **Kernighan–Lin/Fiduccia–Matheyses (KLFM)** paradigm. Their proposed heuristic partitions, schedules, and performs the linear placement of tasks simultaneously on the target device. However, they assumed that each task would require several whole columns on the target device. This assumption is no longer practical with the development of modern DPR, which supports more robust 2-D PR Regions nowadays. Similar to [14], the work [15] and [16] only target to homogeneous FPGA, which is composed of a matrix of $n \times m$ **Configurable Logic Blocks (CLBs)**, and the smallest reconfigurable element is a column that is one CLB wide. Modern FPGA devices are composed of heterogeneous resources such as CLBs, DSPs, and **Block RAMs (BRAMs)**. This makes many previous works out of date. Configuration prefetching is usually considered when building their models by these studies [14–16]. But to the best of our knowledge, there is no such a configuration strategy that can be used by DPR designers when considering PR Module selection and clustering.

In [17], S. Ghiasi et al. proposed a reconfiguration sequence management problem that uses scheduled operations as input. They supposed that all tasks are of equal area and focused on exploiting similarities between a given set of scheduled tasks. Their assumption is impractical because each task requires a particular group of active hardware modules in real designs. And the reconfiguration overhead and resource requirements of operations are also various, and they are key factors to design a highly efficient DPR system.

Related studies such as online placement [7–9] attracted research interest in the past few years. But the increasing heterogeneous blocks and uneven routing in modern devices make these approaches very difficult to implement in reality and keep up with the times [18]. As a consequence, we feel that it is more meaningful and effective to address the challenges of DPR in the vendor-supported tool flows by developing algorithms to aid designers who are not architecture experts.

Furthermore, we also make a survey on the recent five years. In [28], A. Lifa et al. proposed a complete IP-based framework with comprehensive application programming interface. This framework could accelerate the application by dynamically scheduling hardware prefetches. Lifa

also gave a survey of FPGA configuration prefetching. Additionally, in [30] S. Brennsteiner et al. improved and extended the Xilinx DPR flow based on fixed-point multiplication and division circuits. So, both [28] and [30] tried to bring improvements through hardware framework optimization. This is different from our focus in this paper. Except for hardware optimization, [31–33] focused on the splitting and partitioning method on DPR and proposed different strategies. For example, [32] proposed a partitioning method based on petri net, [31] proposed a QoS-aware approach that can bring efficient joint optimization across all factors in FPGA-based DPR system, and [33] focused on hardware and software partitioning to ensure predictable delays when managing hardware accelerators by DPR. We could see that although the works in [31–33] have a relationship with FPGA DPR, their focus is not on PR module partitioning. Clustering on PR module is also not their focus. Similar to [5] and [6], J. Wang et al. also focused on the dynamic partial reconfiguration floorplanning and proposed a mixed-integer linear programming method [29]. In conclusion, we did not find related works in the recent five years that have a tight relationship with this paper. They did not focus on the selection and clustering method for PR modules. However, we could refer to the strategy in [29] to improve our approach in the future, especially how to describe DSP/RAM layout positions during multiple constraints.

In reality, DPR imposes many physical constraints that need to be incorporated explicitly and existing related works often don't consider this special challenge. In this paper, we try to propose an effective algorithm to deal with PR Module selection and clustering problems in the vendor-supported DPR design flow. We not only support various real-life design constraints but also try to incorporate configuration prefetching and consider multiple objectives such as resource utilization and reconfiguration delay.

3 PROBLEM DESCRIPTION

3.1 Preliminary

According to the state-of-the-art DPR design flow EAPR [3], a PR Region is a designated rectangular region on the FPGA where a group of PR Modules can be loaded and unloaded dynamically. Each PR Region can accommodate at most one PR Module at any time. Whenever a PR Module is swapped in, there will be a reconfiguration latency which is determined by the size of the PR Region.

When implementing an application on a partially dynamically reconfigurable FPGA, the designer must select and cluster PR Modules manually. Though some hardware modules such as the master module can be identified directly as static modules, analyzing the undetermined modules as candidate PR Modules still require extensive labor from the designer, who is expected to know the details of the target FPGA architecture. In this paper, we build a model to provide efficient PR Module schemes for designers.

3.2 Problem Formulation

For a module-based application, the schedule of hardware modules during the application execution is represented with a Gantt chart derived from the outputs of high-level synthesis tools or provided by designers (as shown in Figure 2). In Figure 2, the dependence is shown through the scheduling order. Module B must run after A and C finish; B should also run after module E in the period 1-2 finishes. Module D must run after module B in the period 3 finishes. Module A in the period 6 and module E in the period 6 must run after module D in the period 4-5 finishes.

Given:

(1) A candidate PR Module set $C = \{M_1, M_2, \ldots, M_n\}$ with a Gantt chart $\mathcal{G}$ describing their schedule. The application execution life cycle can be divided into a series of periods and each module is active during certain periods. As shown in Figure 2, each column depicts

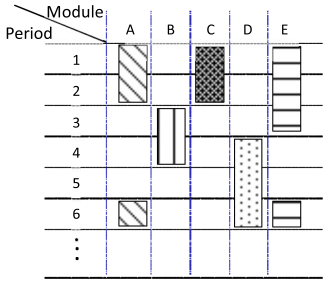

Fig. 2. Gantt chart describing the schedule of candidate PR Modules during the application execution.

the active periods of a specific hardware module. For example, module A is active during periods 1 to 2 and period 6. For demonstration, we will take the application from period the 1 to the period 6 described in Figure 2 as an example throughout this paper.

(2) *Resource Requirement Vector* [11] of each candidate PR Module. Each module requires a specific group of resources whose number can be depicted by a Resource Requirement Vector. Resource Requirement Vector is a 3-tuple vector $Req_m = (m_{clb}, m_{ram}, m_{dsp})$ that represents the number of CLBs, BRAMs, and DSPs required by a module m. This information can be acquired by employing Xilinx's ISE to synthesize each module generated by a high-level synthesis tool.

The PR Module selection and clustering problem is to select a subset $\mathcal{S} = \{M_{i1}, M_{i2}, \ldots M_{im}\}$ of C as PR Modules and partition $\mathcal{S}$ into appropriate groups S_1, S_2, ..., S_p such that $\cup_{k=1}^{p} S_k = \mathcal{S}$ and $\forall k1 \neq k2 \in \{1, 2, \ldots, p\}$, $S_{k1} \cap S_{k2} = \emptyset$. Note that im and p are unknown variables to be determined.

This process needs to optimize the target design objective under given constraints such as:

- **Non-concurrency constraint:** According to the modern DPR design flow, a PR Region can accommodate at most one PR Module at the same time. Therefore, for each group of PR Modules sharing a PR Region, there must be at most one active module in this group in each period of the application execution. For example, as shown by Figure 2, since module B and module C are not concurrently scheduled, it will satisfy this non-concurrency constraint if module B and module C are selected as PR Modules and share a PR Region on the FPGA fabric.
- **Resource requirement:** Each static module requires a specific group of resources on the target FPGA fabric. While each cluster of PR Modules requires a specific group of resources whose number is determined by the superset of the *Resource Requirement Vectors* of all the PR Modules in this cluster. The example in Figure 2 is only used to show the non-concurrency. However, it has no motivation to also show resource requirement.
- **Design constraints:** Optional constraints such as the limit of available resources on the target FPGA and the upper bound of the number of PR Regions can be restricted by the designer. The designer can also constrain the area discrepancy of PR Modules sharing a PR Region to reduce fragments caused by various resource requirements of the PR Modules. For applications where a strict limit of reconfiguration latency must be met, an upper limit of the total reconfiguration delay can also be set as a constraint for the design space exploration.

We only consider non-concurrency and resource, not considering the power/energy. Generally, the idea of the proposed algorithm has no big change. The only concern is the complexity when

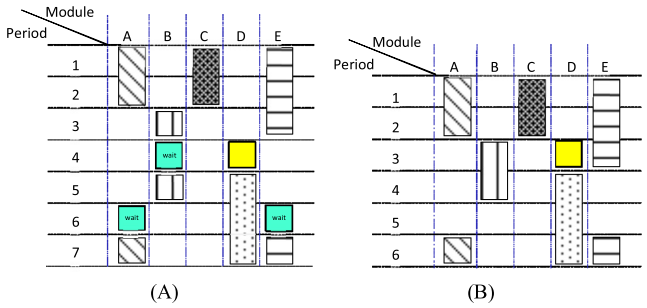

Fig. 3. Comparison between employing configuration prefetching and not for the application shown in Figure 2: (A) Without prefetching: the application needs an extra period to finish its execution; (B) With prefetching: reconfiguration delay can be hidden.

exploring the design space. The reason why power is not considered is due to the experimental environment. For the real-life applications, we obtain the estimated resource values through **High-level synthesis (HLS)**. However, Xilinx HLS has no power estimation. This is an important reason why power is not a constraint in our approach.

3.3 Optimization Objective

There may be several common situations that the designer needs to address in real life. One case is that the designer needs to generate PR Module combinations with the minimized total reconfiguration latency and enable the application to fit into a given FPGA chip.

Another case is that for applications that have no strict time constraint, the designer tries to minimize the resource utilization of the implementation with little concern for the reconfiguration latency.

Therefore, the optimization approach of PR Module generation needs the ability to explore the trade-off between the resource utilization and the reconfiguration delay.

3.4 Configuration Prefetching

In this paper, when evaluating each PR Module scheme, we also incorporate a configuration prefetching strategy to improve the reconfiguration latency imposed by dynamic partial reconfiguration. For the application shown in Figure 2, suppose that we select *C*, and *D* as PR Modules and cluster them to share a PR Region PRR_1 on the FPGA. Assume the refreshing of PRR_1 takes one period. If we don't employ any configuration prefetching strategy, firstly in period 4 we need to swap PR Module *D* in PRR_1, and then in period 5 PR Module *D* can begin to execute. As shown in Figure 3(A), the execution of modules depending on PR Module *D* is delayed by one period compared to the original schedule shown in Figure 2. However, in Figure 2 module *B* and *D* in period 4 run in parallel. This means that they have no control dependence and then cannot share the same resource. But they might be data dependent. If two modules run in parallel, there may be data communication between them. Therefore, to guarantee the same behavior, module *D* which might be data dependent will also have one-period delay correspondingly. Same scenario happens on both module *A* and *E*. Therefore extra waiting periods are introduced. On the contrary, if we employ configuration prefetching, the swap-in of *D* can be performed beforehand during the period 3, as shown in Figure 3(B). In this way, the reconfiguration of PRR_1 imposes no latency on the entire execution. Apparently, there is usually more than one PR Region in a DPR system, and not all reconfiguration delays can be hidden perfectly as in this example. A good prefetching strategy can hide as much reconfiguration latency as possible during the application execution life cycle.

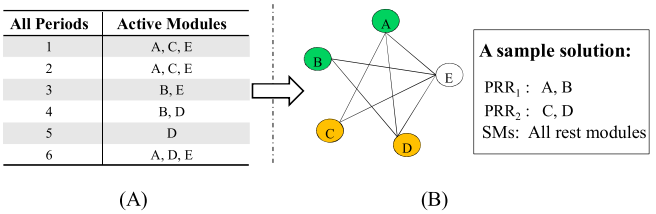

Fig. 4. Build Exclusion Graph according to the Gantt chart: (A) Execution Table derived from the Gantt chart; (B) Exclusion Graph.

In this paper, we propose a novel flexible formal model for this PR Module selection and clustering problem. For different situations when different optimization objectives are required, our approach succeeds in transferring the original problem into a **Maximum-Weight Independent Set Problem (MWISP)** with design constraints well considered. When evaluating and comparing possible PR Module schemes, we consider the total reconfiguration latency of each scheme by incorporating a configuration prefetching strategy to reduce its imposed delay over the entire application execution life cycle.

4 EXCLUSION GRAPH-BASED FORMAL REPRESENTATION

Considering the non-concurrency relationships among hardware modules, we build a graph (i.e. Exclusion Graph) to formally describe the PR Module selection and clustering problem. The exclusion graph is used to describe whether any two hardware modules meet the non-concurrency constraint discussed in Section 3. First, we define several concepts and discuss what we can derive based on these concepts. At the end of the discussion, we will find that the PR Module selection and clustering problem can be formally described as a special Partial-Vertex Coloring Problem on the *Exclusion Graph*.

Definition 1 (Exclusive). Module A and module B are exclusive if they are both active in one or more periods.

Based on Definition 1, we can say that module A and module B meet the non-currency constraint if and only if they are not exclusive. Now we construct a graph (called Exclusion Graph) to describe the exclusive relationship between each pair of hardware modules. To accomplish this, for each period, we identify all active modules and we get the execution table as shown in Figure 4(A). Then, we build a graph with five isolated vertices representing the five modules, respectively. Next, for each period we complete a pair-wise connection between vertices that represent active modules in this period, and finally we obtain Figure 4(B).

Similarly, we can say that a set of modules satisfies the non-concurrency constraint if and only if any two modules in this set are not exclusive. This means that there is no connection between any two modules in the set. Therefore, we can conclude that a solution to the PR Module generation problem satisfies the non-concurrency constraint if and only if it consists of several intersection-free sets on the Exclusion Graph such that for every two vertices in each set, there is no edge connecting them.

In fact, to represent multiple independent sets on the Exclusion Graph, the PR Module generation problem can be formally described as a **Partial-Vertex Coloring Problem (PVCP)**:

PVCP: Consider an undirected graph $G = (V; E)$, (i.e. Exclusion Graph in this paper) where V is the set of vertices and E is the set of edges. The Partial-Vertex Coloring Problem (PVCP) requires selecting certain sets $(S_1, S_2, S_3, \ldots, S_k)$ of vertices and assigning each set a unique color in such a way that

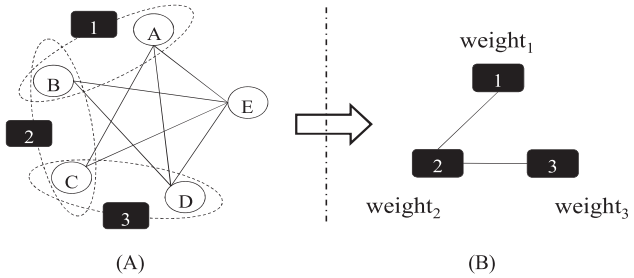

Fig. 5. Build the Intersection Graph: (A) Identify alternative independent sets on the Exclusion Graph; (B) Intersection Graph.

(1) No intersection: $\forall i,\ j \in \{1, 2, \ldots, k\},\ i \neq j,\ S_i,\ S_j \subseteq V,\ S_i \cap S_j = \emptyset$;
(2) No connection: $\forall i \in \{1, 2, \ldots, k\},\ \forall v_a,\ v_b\ \in S_i,\ v_a \neq v_b$, v_a and v_b are not adjacent.

Note that k corresponds to the number of PR Regions and it is unconstrained by default.

So far, we have represented the PR Module selection and clustering problem as a Partial-Vertex Coloring Problem (PVCP). Figure 4(B) also presents a sample of possible solutions. As we can see from this figure, $\{A, B\}$ and $\{C, D\}$ are two intersection-free sets assigned with two colors, respectively. Green vertices $\{A, B\}$ can be designated as PR Modules and share a PR Region PRR_1, while orange vertices $\{C, D\}$ can be designated as PR Modules and share another PR Region PRR_2. All the rest of vertices are designated as static modules and mapped to a static region on the FPGA fabric. Apparently, Theorem 1 can be established.

Theorem 1. There is a one-to-one correspondence between the solutions to PR Module selection and clustering problem and the solutions to this Partial-Vertex Coloring Problem.

5 INDEPENDENT SET-BASED ALGORITHM FOR PVCP

In the previous section, we have formulated PR Module selection and clustering problem as a special Partial-Vertex Coloring Problem (PVCP) on the Exclusion Graph. In this section, we will discuss how we build a model to solve this special problem.

5.1 Alternative Independent Set Identification

In the graph theory, an independent set (or stable set) is a set of vertices in a graph, no two of which are adjacent [26]. According to the discussion in Section 4, due to the no connection requirement of PVCP, each valid solution must consist of a group of independent sets. Therefore, we can first identify all alternative independent sets on the Exclusion Graph, then analyze the relationships among them and choose an optimal group of independent sets under design constraints.

In this step, given the Exclusion Graph (shown by Figure 5(A)), we identify all the alternative independent sets that constitute a set $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_t\}$. Figure 5(A) shows the situation that all independent sets are identified on the Exclusion Graph. Certain ineligible independent sets can be discarded according to the given design constraints. Constraints like limiting the upper bound of the area discrepancy of PR Modules sharing a PR Region can be handled in this step. If the resource requirements of the PR Modules sharing a PR Region vary considerably, there would be many unused fragments when the smaller PR Module is active. Restricted by such constraints, ineligible independent sets that don't meet these constraints are discarded and only alternative independent sets are preserved, which not only favors the later-on optimization by limiting the solution space, but also controls the possible PR Module design to be uniform in terms of resource utilization.

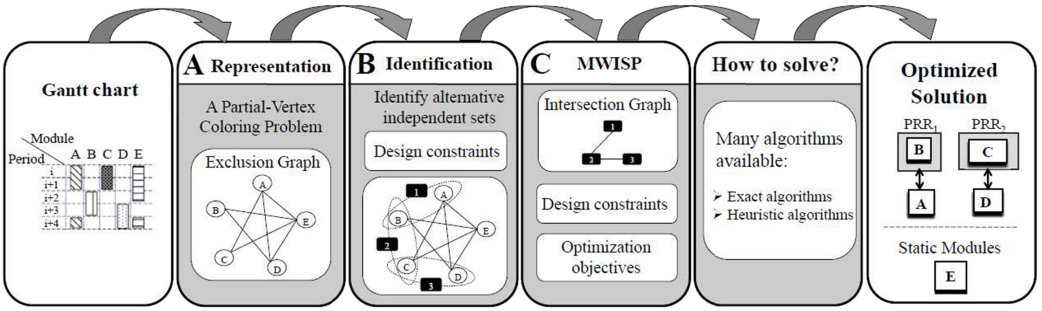

Fig. 6. Workflow of our approach for PR Module selection and clustering.

5.2 Build Intersection Graph

In this step, given the set of alternative independent sets $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_t\}$ from the previous step, we cannot directly get the feasible coloring solution for PVCP since there may be some intersection between different independent sets. For example, as shown in Figure 5(A), the intersection of independent set 1 (i.e. $\{A, B\}$) and independent set 2 (i.e. $\{B, C\}$) is $\{B\}$. Therefore, the two independent sets cannot be assigned with two different colors, which means it is unfeasible to include both of them in the solution to PVCP. According to the no-intersection requirement of PVCP in Section 4, we need to find an intersection-free subset of the alternative independent sets as $\mathcal{A}' = \{\mathcal{A}_{j1}, \mathcal{A}_{j2}, \ldots, \mathcal{A}_{jh}\} \subseteq \mathcal{A}$ with each element in $\mathcal{A}'$ to be assigned a unique color and $\forall u \neq v \in \{j1, j2, \ldots, jh\}$, $\mathcal{A}_u \cap \mathcal{A}_v = \emptyset$ because we cannot assign two different colors to a single vertex.

To describe the intersection relations between each pair of independent sets in $\mathcal{A}$, we construct an Intersection Graph ($G' = (V'; E')$) whose vertices represent the alternative independent sets and edges represent Intersection relations (see Figure 5(B)). If there is intersected part between two independent sets, their corresponding two vertices will be connected by an edge.

Apparently, every solution to the Partial-Vertex Coloring Problem corresponds to a set of vertices SL on the Intersection Graph such that $SL \subseteq V'$ and no two vertices of SL are adjacent. This means SL is an independent set of the Intersection Graph and we can solve the Partial-Vertex Coloring Problem (PVCP) by finding an independent set of the Intersection Graph with objectives optimized under design constraints. By assigning each vertex with the optimization objective evaluation as a weight, we can solve the Partial-Vertex Coloring Problem (PVCP) by solving a Maximum-Weight Independent Set Problem (MWISP) on the Intersection Graph. Design constraints such as limited resources or the upper limit of reconfiguration delay can be used to prune during the searching process to obtain some speed-up.

5.3 Overall Workflow

In this subsection, we describe the PR Module selection and clustering flow based on our independent set-based algorithm. As shown in Figure 6, firstly we build an undirected graph (i.e. Exclusion Graph) to describe whether any two given modules satisfy the non-concurrency constraint. And then the PR Module selection and clustering problem is formally represented as a Partial-Vertex Coloring Problem (PVCP) with certain design constraints. To solve the PVCP, we identify all alternative independent sets on the Exclusion Graph and use an Intersection Graph to analyze the intersection between each pair of these independent sets. With the constraints and optimization objectives considered, the PR Module selection and clustering problem is finally formulated into a Maximum-Weight Independent Set Problem (MWISP) with certain PR Module design constraints.

When defining the weight of each vertex on the Intersection Graph, configuration prefetching can be incorporated to improve the reconfiguration penalty imposed by DPR. We will discuss it in the next section.

Note that there are usually a few vertices (several dozen) on the Intersection Graph due to the few number of candidate PR Modules in real life. Exact algorithms that target optimal solutions can be employed. These algorithms have been well studied by many previous works such as [19–21]. When the problem size is large in extreme cases (for example, thousands of vertices on the Intersection Graph), there are two strategies which can be adopted by the designer. One strategy is that more static modules could be extracted manually so that fewer candidate modules are needed to be formulated by our approach. The other is to employ some heuristic algorithms, which have also been well studied [22–24], to find approximate optimal solutions within a reasonable time.

6 DESIGN OPTIMIZATION WITH CONFIGURATION PREFETCHING INCORPORATED

In this section, first we introduce the configuration prefetching strategy we employ when evaluating every possible PR Module scheme. Then, we will discuss how to accomplish common design optimization based on our proposed approach. Resource optimization and reconfiguration delay optimization are the two most important design objectives that often need to be addressed in real life. We will discuss them in detail respectively.

6.1 Prefetching Strategy: Step-Wise Adjustment

In a DPR system, each swapping of a PR Region can impose a delay (**Reconfiguration Delay, RD**) on the total application runtime during system execution. The general method is to add up the switching delay simply before each module to be swapped in. Taking Figures 2 and 3(A) as an example, another delay is added in period 4, which is bad news for total RD since it also brings other waiting delays on module *A, B* and *E*. However, since there is an interval between module *C* and *D*, we could add up the delay into the interval periods instead to reduce total RD, as shown by Figure 3(B). It requires a step-wise strategy to analyze each data dependency and how many intervals between modules, and then determine which period can be used for prefetching. Instead of simply adding up all switching delays of all PR Regions during the application execution, we adopt a stepwise adjustment strategy for configuration prefetching based on the schedule analysis to lighten the total RD imposed by DPR. And then we use the optimized total RD to evaluate a given PR Module scheme.

Based on the Gantt chart, we can analyze the task schedule to discover intervals between the PR Modules, so that some of the modules can be swapped in advance to lighten the reconfiguration overhead. As shown in Figure 7, suppose that there are two groups of PR Modules {*A*, *B*} and {*C*, *D*}, in which *A* and *B* share PR Region PRR_1 on the FPGA with the reconfiguration delay of one period. While *C* and *D* share PR Region PRR_2 with the reconfiguration delay of two periods. As shown in Figure 7(A), assume PR Module *A* is in PRR_1 and *C* is in PRR_2 before period 1. To analyze the reconfigure process, we define switching points on the Gantt chart by a point set $\mathcal{PS} = \{p_1, p_2, \ldots, p_n\}$. To figure out the possible intervals for switching, we can calculate the earliest switching time (est_i) and required switching time (rst_i) for switching point p_i. Each point is located at its earliest switching time on the Gantt chart.

As shown in Figure 7(A), there are three switching points p_1, p_2, p_3 corresponding to the three reconfiguration operations. For the reconfiguration between module *A* and *B* in PRR_1, the switching point p_1 need to be located at the end of period 2 and there is no interval between module *A* and module *B*. Therefore, both the earliest switching time and the required switching time are at the end of period 2 ($est_1 = rst_1 = 2$). But for the switching point p_2 in PRR_2, there are some gaps

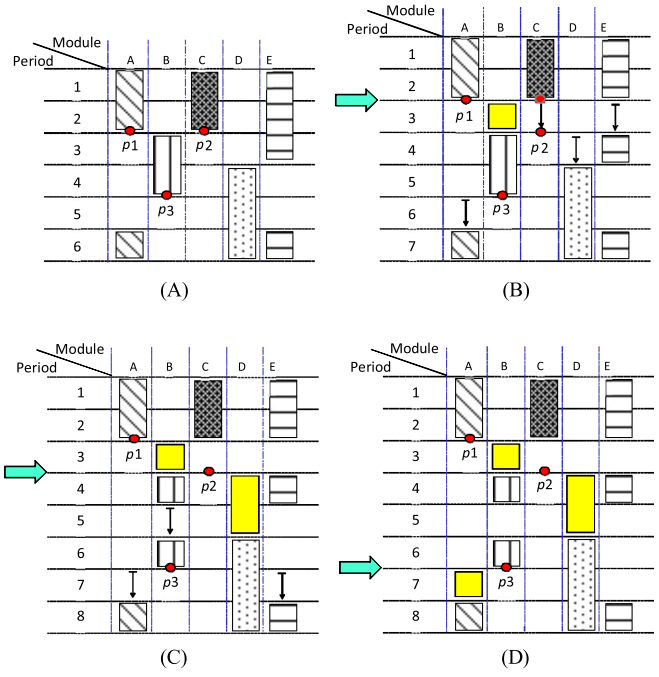

Fig. 7. Diagram of the step-wise adjustment prefetching strategy. Assume there are two PR Regions shared by {*A, B*} and {*C, D*}, respectively. (A) Calculate all switching points locating at the earliest switching time of all the PR Regions; (B) Choose the switching point p1 with the least margin ($Min\_mar = 0$, $rd = 1$) and postpone the whole schedule from period 3 down and all the other switching points by 1(1-0) period, $p2$ is also postponed because there is only one configuration port; (C) Process point $p2$ ($Min\_mar = 1$, $rd = 2$); (D) Process point $p3$ ($Min\_mar = 1$, $rd = 1$). The step-wise adjustment is finished.

between module C and module D as shown in Figure 7(A). The earliest switching time is the end of period 2 ($est_2 = 2$) and the required switching time is the end of period 3 ($rst_2 = 3$). We define the gap between the earliest switching time and the required switching time as the margin (mar_i) for switching point p_i denoted by $mar_i = rst_i - est_i$. Therefore, the margin values in Figure 7(A) are $mar_1 = 0$, $mar_2 = 1$, $mar_3 = 1$.

After marking these switching points with est and rst, we begin to handle them one by one following the reconfiguration schedule. First, we rank $\mathcal{PS}$ into an ordered set $\mathcal{PS}'$, in the ascending order of the earliest switching time (est_i). Note that there may be more than one points with the same earliest switching time in $\mathcal{PS}'$. For example, in Figure 7(A) p_1 and p_2 have the same earliest switching time. Suppose the point/points with the smallest est is/are $CS = \{p_{i1}, p_{i2}, \ldots, p_{im}\}$ that $est_{i1} = est_{i2} = \cdots = est_{im} = Min\_est$. And then we choose the point p_j with the smallest margin value and handle it first. Assume the smallest margin value is Min_mar ($\geq$0) and the reconfiguration delay corresponding to p_j is rd. Since part of the reconfiguration delay (Min_mar) can be hidden by the prefetching process, the reconfiguration cost is reduced to ($rd - Min\_mar$) instead of rd.

To make sure the application can be executed correctly, the functions which have the data dependency relations with the current swap-in modules should be postponed for ($rd - Min\_mar$) periods to wait for the completion of the reconfiguration process. If the data dependency is not given, then the entire schedule after Min_est should be postponed. In this paper, we demonstrate the worst situation by assuming all the functions afterward should be postponed. As shown in

ALGORITHM 1: Step-Wise Adjustment For Configuration Prefetching

```
Calculate all the earliest switching points on the Gant chart;          // i.e., get set PS
Calculate est_i, rst_i, mar_i for each switching point;
Sort (PS, PS');                                  // in ascending order of est;
RdS := Ø;
While PS' != Ø
    Collect all points/point with the smallest period value Min_est;    // get CS
    Select p_j with the smallest margin value Min_mar (≤0);             //with the RD rd
    Postpone the schedule from Min_est down by (rd−Min_mar);
    Postpone all points in PS'/p_j by (rd−Min_mar);
    PS' := PS'/p_j;
    RdS := RdS ∪ p_j;
End While;
```

Figure 7(B), the reconfiguration process on switching point p_1 will generate the delay for all the other functions after period 2. Switching points in $\mathcal{PS}'/CS$ are adjusted by increasing est_i and rst_i by $(rd - Min\_mar)$. And the switching points with the same earliest switching time Min_est should also be postponed since there is usually only one configuration port in the DPR system. For example, in Figure 7(B) p_2 is also postponed by 1 period since reconfiguration operations of 2 PR Regions cannot be executed at the same time. Therefore, all the point/points in $\mathcal{PS}'/p_j$ should be adjusted by adding $(rd - Min\_mar)$ to their est_i and rst_i.

After dealing with p_j, we remove p_j from $\mathcal{PS}'$ to the ready point set $Rd\mathcal{S}$ which is used to hold the switching points that are already adjusted. With the updated Gantt chart, then we begin to handle the point/points with the smallest est_i in the new $\mathcal{PS}'$. We repeat the above handling procedures until all points in $\mathcal{PS}'$ are added to $Rd\mathcal{S}$. Figure 7 described the whole stepwise adjustment process for the sample application. As we can see from the example shown by Figure 7, incorporated with our configuration prefetching strategy, the margins for points p_2 and p_3 are used to prefetech the PR Modules in advance. Therefore, the original application can be finished in eight periods instead of 10(= 6 + 1 + 2 + 1) periods, with two periods hidden by configuration prefetching schedule.

After the algorithm finished, the earliest switching time (est_i) of each point in $Rd\mathcal{S}$ indicates the start point for the swap-in of the corresponding PR Module. Algorithm 1 concludes our step-wise adjustment procedures. Apparently, the time complexity of this algorithm is $O(n^*\log n + n^*(n - 1)/2) = O(n^2)$.. n is the number of the switching points which are usually only a handful.

6.2 Resource Optimization based on Our Approach

In practical work, the designer often needs to economize the available hardware resources on the FPGA chip. Our proposed approach can help minimize the resource consumption of the DPR implementation of the given application by providing optimal PR Module schemes that save the maximum resources.

In this paper, resource consumption is estimated according to the utilized area on the target chip. Area can be calculated in terms of the number of CLB's size. For example, as shown in Figure 8, the area of a BRAM is 3 CLBs, and the area of PRR_1 is 36 CLBs. A hardware module's area can be calculated by adding up the areas of all its required resources. Since a PR Region is shared by a set of PR Modules, its resources must be a superset of all the kinds of resources required by the PR Modules. Therefore, its area can be estimated according to the maximum demand for each type of resource from the accommodated PR Modules. To simplify the problem, this paper does not consider the complex scenario when DSP/RAM has special position requirements. For

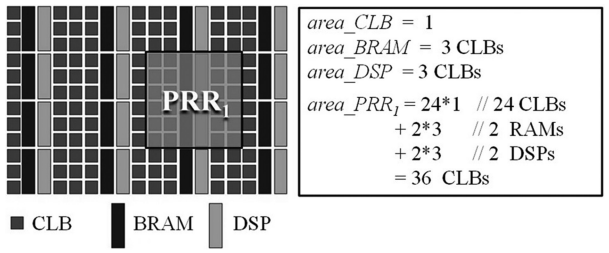

Fig. 8. A schematic diagram of area estimation.

example in Figure 8, if the RAM can only be placed in the 7th column, it will occupy more CLBs (at least 18 CLBs). However, the chip position for RAM/DSP is not considered as a constraint, so our approach does not cover such complex scenario. We will refer to the floorplanning technique and improve our method next.

$$area\_PRR_i = \sum_{k=1}^{num\_types} (area_k \times max\_demand_k) \tag{1}$$

$area_k$ is the area of resource of type k, and max_demand_k is the maximum resource demand for type k from all the PR Modules sharing PRR_i. Since a set of PR Modules share a PR Region on the FPGA fabric, the area saved by a PR Region PRR_i can be calculated by Equation (2).

$$saved\_area\_PRR_i = \left(\sum_{PRM_k \in PRR_i} area\_PRM_k \right) - area\_PRR_i \tag{2}$$

Apparently, for a given feasible PR Module scheme, the saved area can be calculated by

$$saved\_area = \sum_{PRR_i \in \text{Partition}} saved\_area\_PRR_i \tag{3}$$

where $saved\_area\_PRR_i$ is calculated according to Equation (2). Based on our proposed approach, we know that each vertex on the Intersection Graph represents an alternative group of PR Modules that will share a PR Region if this vertex is included in the final PR Module scheme. Therefore, there is a potential gain (i.e. the saved area) for each vertex on the Intersection Graph. The gain can be calculated according to Equation (1)(2) and attached to the vertex as a weight.

The resource optimization turns out to be a standard Maximum-Weighted Independent Set Problem (MWISP). In other words, we need to find an independent set of the Intersection Graph and maximize the sum of the weights of the vertices in the selected independent set.

6.3 Performance Optimization based on Our Approach

Because of the swapping of PR Modules during system execution, reconfiguration overhead imposes a delay (RD) on the total application runtime. Therefore, the overall reconfiguration latency should be optimized with the trade-off between performance and resource utilization. Since all tasks have been scheduled and execution time of each task is already fixed, the application runtime is minimized if and only if RD is minimized. In other words, RD optimization equals runtime optimization in this problem, and therefore we put them together to discuss.

According to Algorithm 1, we can find that switching points are handled one by one and the margin of each point remains the same before and after each adjustment. Therefore, for a given PR Module scheme, the total reconfiguration delay design during the entire application life cycle

(*TRD_DPR*) can be calculated by Equation (4).

$$TRD\_DPR = \sum_{j \in \mathcal{PS}} (RD\_PRR_j - mar_j) \tag{4}$$

RD_PRR_j is the reconfiguration delay of each swapping of the PR Region that corresponds to switching point j. Therefore, *TRD_DPR* can be calculated by adding up the total reconfiguration delay (*TRD_PRR*) of each PR Region incorporating configuration prefetching, respectively.

$$TRD\_DPR = \sum_{j} TRD\_PRR_j \tag{5}$$

In this paper, TRD_PRR_j is calculated by our step-wise adjustment configuration prefetching algorithm, which needs us to calculate RD_PRR_j. Xilinx design flow estimates reconfiguration latency according to the maximum bandwidth of configuration port and the number of configuration frames to be reconfigured [3]. Therefore, we can estimate RD_PRR_j according to the reconfiguration speed and the amount of configuration frames to reconfigure PRR_j.

$$RD\_PRR_j = cf_j \times W_f / Rs \tag{6}$$

cf_j is the number of configuration frames occupied by PR Region PRR_j. W_f is the frame lengths measured in bits. *Rs* is the maximum bandwidth of the configuration port. This calculation is reasonable because modern DPR technology requires a complete refresh of a PR Region whenever a PR Module is being swapped in it. In Equation (6), cf_j is calculated by adding up the number of configuration frames of each resource type in PR Region PRR_j.

$$cf_j = \sum_{k=1}^{num\_types} \left(\frac{area_k \times max\_demand_k}{area\_RF} \times u_k \right) \tag{7}$$

area_RF means the area of a reconfigurable frame. Reconfigurable frame is the smallest physical region that can be reconfigured and *area_RF* rests with the device series. u_k represents the number of configuration frames corresponding to a reconfigurable frame of resource type k, and it is also constant depending on specific FPGA chip.

Based on our proposed approach, each feasible PR Module scheme is an independent set of the Intersection Graph. Similar to the discussion about resource optimization, each vertex on the Intersection Graph can be attached with the total reconfiguration delay of its represented candidate PR Region, configuration prefetching incorporated, as a weight. This weight is calculated according to Equation (6), (7) and Algorithm 1. Since available resources are limited, the final implementation of the design must be small enough to fit into the target FPGA fabric. According to Equation (1), for a given PR Module scheme, the area occupied by all the PR Regions can be estimated by

$$area_{PRR} = \mu \times \sum_{\mathrm{j}} area\_PRR_{\mathrm{j}} \tag{8}$$

where μ is a coefficient which enlarge area estimation by a certain ratio to make up for the underestimation due to the uneven distribution of available resources on the FPGA fabric. As all the rest hardware modules are designated as static modules, each of them will consume an independent area on the FPGA fabric. According to Equation (8), we can estimate the area required by the PR Module scheme by

$$area\_total = \mu \times \sum_{\mathrm{j}} area\_SM_{\mathrm{j}} + area_{PRR} \tag{9}$$

Table 1. Selected Seven Groups of Benchmarks Manually Designed

Benchmark Groups			Origin Area
ID	*#Candidate PR Modules*	*#Periods*	
TG1	16	400	12956
TG2	20	500	15940
TG3	24	600	18984
TG4	28	700	22103
TG5	32	800	25570
TG6	36	900	28796
TG7	40	1000	32341

In Equations (8) and (9), μ is set according to the resource distribution on the specific FPGA chip. Finally, we conclude that a PR Module scheme satisfies the resource constraints if

$$area\_total \leq area\_FPGA \tag{10}$$

where *area_FPGA* means available area on the given FPGA fabric.

In conclusion, our approach-based performance optimization turns out to be a Minimum-Weight Independent Set Problem on the Intersection Graph under resource constraints. In other words, we need to find an independent set of the Intersection Graph, minimizing the sum of the weights of the vertices in the selected independent set while guaranteeing that the DPR implementation can fit into the given FPGA chip. To solve it, we can transform it into an equivalent Maximum-Weighted Independent Set Problem (MWISP) by making the weight attached to each vertex to be its additive inverse. Then we can solve the MWISP by adding a pruning strategy based on the area constraint to the traditional MWISP algorithm framework. With our approach, it is also possible to optimize the resource utilization and reconfiguration costs at the same time by combining weights from different optimization objectives for each vertex on Intersection Graph.

7 EXPERIMENTAL RESULTS

In this section, based on an exact MWISP algorithm [19] to find the optimal solution, we present experiments to demonstrate the efficiency of our approach. The goals of our experimental study are three-fold: (1) to demonstrate that our approach can find efficient solutions using exact algorithms while spending little runtime, (2) to demonstrate our approach's ability to effectively explore the PR Module design space, and (3) to give a real-life example in which our approach is helpful.

7.1 Experimental Setup

We take the XC5VLX330T fabric from the Xilinx Virtex-5 series as our target FPGA chip. Using SelectMAP mode or ICAP, the maximum bandwidth of the configuration port is 3.2 Gbps. Since no standard DPR benchmark is available, we have manually designed seven groups of benchmarks with big problem scales according to the execution mode of common applications to evaluate our proposed approach. Each group consists of 10 different benchmarks that vary in the degree of parallelism and continuity of active periods to represent various types of applications. The Resource Requirement Vector of each module is generated randomly while guaranteeing that the number of required CLB is between 200 and 700 and the number of required BRAM and DSP is 5% ∼ 10% of the number of required CLB. So during the seven groups of benchmarks, there are also cases with both RAM and DSP. Table 1 demonstrates the seven groups of benchmarks (TG1, TG2, . . ., TG7). Avg. Origin Area means the average original area of the benchmarks before optimization. These benchmarks have covered different complexity of cases. And from the generation method for

Table 2. Efficiency of the Two Optimization Algorithms

ID	#V	Resource Optimization					*RD* Optimization					
		Without Prefetching		*With Prefetching*		*Avg. Area*	*Without Prefetching*			*With Prefetching*		
		Avg. RD (ms)	*Avg. RT (s)*	*Avg. RD (ms)*	*Avg. RT (s)*		*Avg. Area*	*Avg. RD (ms)*	*Avg. RT (s)*	*Avg. Area*	*Avg. RD (ms)*	*Avg. RT (s)*
TG1	47	6.1597	0.0002	5.9320	0.0001	9500	11168	2.4668	0.0002	11192	2.3571	0.0002
TG2	91	7.6041	0.0013	7.2831	0.0011	11875	14059	2.3457	0.0005	14054	2.2078	0.0005
TG3	124	9.3956	0.0282	8.9379	0.0287	13832	16675	2.9725	0.0024	16673	2.7323	0.0026
TG4	175	10.8635	0.1601	10.3291	0.1645	16433	19418	3.3216	0.0131	19428	3.0445	0.0124
TG5	168	11.6432	1.3412	10.9832	1.3660	19344	22450	4.1119	0.0666	22457	3.7533	0.0727
TG6	248	14.1199	27.910	13.3396	27.751	21257	25224	4.5165	0.6248	25221	4.0565	0.7248
TG7	424	14.8985	109.97	14.0074	112.73	24022	28359	4.5812	4.5167	28349	4.0854	6.3085
ratio		*1*	*1*	*0.9502*	*1.0189*	*0.741*	*0.8760*	*0.3302*	*0.2378*	*0.8762*	*0.3040*	*0.2417*

RAM/DSP usage, it should guarantee the general scenarios. However, it still has some limitations. For example, it cannot guarantee to cover complex scenario when RAM/DSP has a very special position requirement. Since these seven groups of benchmarks are not real-life applications, it's too hard to design to cover such complex scenario. During the following experiment, we will not consider such scenarios. We take 0.0005 *ms* as one execution period. This is an augmented value only for easy comparison. From Xilinx UG702 [3], a general period timing constraint used for register-to-register paths is set as 10 ns. For easy comparison, we use a 50X value to indicate an obvious comparison of RD improvements. To avoid unused fragments in PR Regions as we discussed in Section 5.1, we limit that within a group of PR Modules sharing the same PR Region, the area of the smallest PR Module must be bigger than 1/3 of that of the largest one.

To show the exploration ability of our approach, we have implemented two algorithms to find efficient PR Module schemes with different optimization objectives. The first algorithm minimizes the area of the design (Resource Optimization); the second algorithm minimizes the RD of the design under a given resource constraint (RD Optimization). The two algorithms are implemented according to what we discussed in Section 6. To demonstrate the efficiency of our step-wise adjustment prefetching strategy, for each of the two algorithms, we implement two versions (with/without prefetching) to compare the performance of their final PR Module schemes. Specifically, for Resource Optimization, the prefetching-incorporated version adopts the step-wise adjustment algorithm to improve the final PR Scheme found by the prefetching-free version. While for RD Optimization, the prefetching-incorporated version also employs the step-wise adjustment algorithm to recalculate the whole schedule of the final PR Module scheme generated on its own, apart from what we discussed in Section 4.

For the RD Optimization algorithm, we also generate the available resource vector. The number of available resources on XC5VLX-330T is set to 85% ∼ 90% of that required by the original design. In this experiment, we also constrain that the total number of PR Regions is no more than seven. This constraint is probably because too many PR Regions often bring high reconfiguration overhead and may exceed the available storage capacity of external memory according to the DPR design flow [3].

7.2 Efficiency of Our Approach

All the algorithms are written in C++ and compiled under Red Hat OS on Intel(R) Xeon(R) CPU E5620 @2.40 GHz Server with 12 GB (8G free) of RAM. Table 2 demonstrates the experimental results of the seven groups of benchmarks (TG1, TG2, . . ., TG7). In Table 2, #*V* means the average

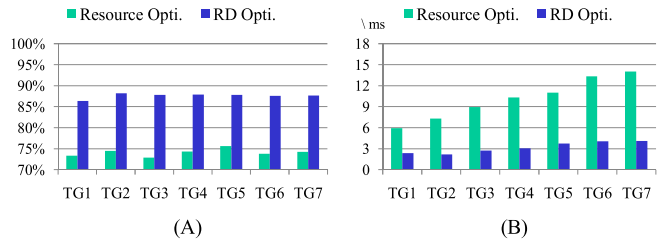

Fig. 9. Optimization effects of different approaches: (A) Resource utilization. (B) Reconfiguration delay.

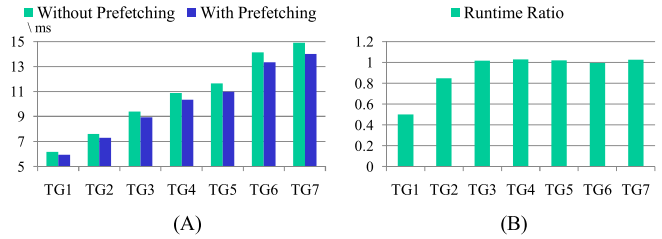

Fig. 10. Different versions of Resource Optimization: (A) RD comparison; (B) Runtime comparison (With/Without Prefetching).

number of vertices on the Intersection Graph. *Avg. Area* shows the average value of the optimized area; *Avg. RD* represents the average total reconfiguration delay after optimization; *Avg. RT* means average algorithm runtime. Note that there may be no solution to some benchmarks for the RD Optimization algorithm due to the shortage of available resources for the given application. We don't take them into account in Table 2 when collecting statistics.

(1) Comparison between the two algorithms: As we can see from Table 2, the Resource Optimization approach provides a PR Module scheme with the least resource utilization (on average 74.11% of that of the original design), but the RD is relatively high. RD Optimization approach needs to fit the application into an FPGA fabric with a 10% ~ 15% shortage of resources while minimizing RD at the same time. The limited available resources provide an additional pruning condition when solving the MWISP, which makes RD Optimization take less time than the Resource Optimization.

Figure 9(A) and (B) compare the optimization effects of the stepwise adjustment prefetching incorporated version of the two algorithms. Resource Optimization provides the PR Module schemes with the least possible area consumption, as shown by Figure 9(A). As the benchmarks increase in complexity, the reconfiguration delay achieved by the two algorithms tends to grow at different rates. Therefore, the ability to effectively explore the trade-off between resource utilization and reconfiguration delay for PR Module generation is more critical.

(2) Comparison between the two versions of each algorithm: As we can see from Table 2, for Resource Optimization, the prefetching-incorporated version takes very little extra time (1.89%) than the prefetching-free version, while providing around 5% RD optimization. Figure 10 presents a comparison of RD and runtime between the two versions. Table 2 also shows that for RD Optimization, the prefetching-incorporated version provides more than 7.93% RD optimization on average than the prefetching-free version. Sometimes prefetching-incorporated version even takes less time than the prefetching-free version to finish. This is because most of the runtime of the whole algorithm is spent on solving

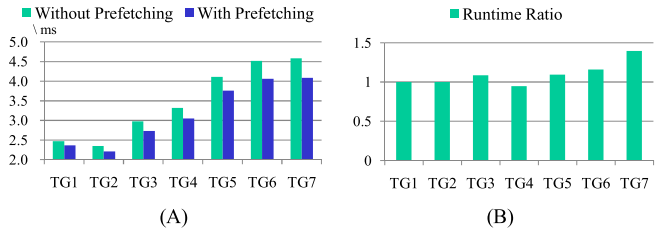

Fig. 11. Different versions of RD Optimization: (A) RD comparison; (B) Runtime comparison (With/Without Prefetching).

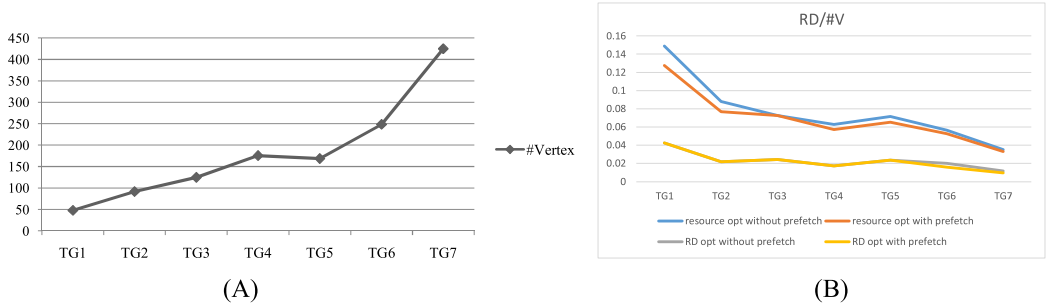

Fig. 12. (A) Scalability of our approach; (B) Approximate linear relationship between RD and scale.

the final MWISP. After incorporating our prefetching strategy, the weight of each vertex on the Intersection Graph is different from that in the prefetching-free version. Therefore, we can't be sure which one of the two versions of RD Optimization spends more time. Figure 11 presents the comparison of RD and runtime between the two versions.

(3) Scalability: In the experiment, we find that building the Intersection Graph is pretty efficient in that we can build the corresponding MWISP in less than 2 *ms* for all the benchmarks. Most of the runtime of our algorithms is spent on the part of solving the MWISP. Therefore, the runtime of our approach relies on the scale of the Intersection Graph greatly. In Table 2, column #*V* presents the scale of the MWISP, namely the number of vertices in the Intersection Graph. Figure 12(A) presents the trend of #*V* with the increase in the complexity of the benchmarks.

From Figure 12(A) we can see that the problem scale of our approach tends to increase linearly as the benchmarks increase in complexity. To show the relationship between RD and scale value, Figure 12(B) presents the trend for four sets of results. Its Y-axis means RD/#V. The four curves in Figure 12(B) show that there is an approximately linear relationship. However, the slope of Resource Optimization is obviously bigger than RD Optimization, which denotes that our approach could lighten the impact of the scale. Furthermore, the method with prefetching can obtain a better effect than the one without prefetching. As shown in Table 2, even for the largest cases with 40 candidate PR Modules and 425 vertices in the Intersection Graph, our proposed approaches can find efficient PR Module schemes within just a few minutes. Therefore, our approach with exact algorithms to solve MWISP can efficiently deal with current modern applications.

7.3 Case Study on Real-life Applications

Nowadays, advanced algorithms applied in medical, wireless, and consumer applications are more complicated than ever before. **High-level synthesis (HLS)** tools such as Mentor Graphics

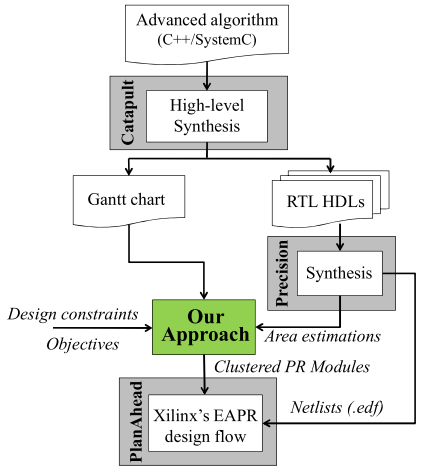

Fig. 13. Our approach bridges the gap between HLS and DPR.

Catapult accelerate design implementation by enabling C, C++, or SystemC specifications to be directly synthesized into production quality RTL, which provides designers with a faster and more robust way of delivering high-quality designs. Our proposed approach can analyze the output of HLS tools and provide efficient dynamic partially reconfiguration schemes for the synthesized design. Figure 13 presents the complete toolchain which can be used by advanced algorithms to be implemented with the DPR feature.

This subsection demonstrates the design process by implementing several real-life applications written in C++ code from the CHStone benchmark suite [13]. First of all, we synthesized these applications with Catapult targeting XC5VLX50T-FF1136 FPGA chip from Xilinx Virtex-5 series. We identify several functions in the algorithm to be hardware modules by setting their hierarchy attributes to be "Block", and they are all chosen as candidate PR Modules, the others are chosen as Static Modules.

After the hierarchical synthesis, Catapult generates synthesizable RTL HDL files as well as a Gantt chart from which the schedule of hardware modules can be derived. Then we synthesize the HDL files using Precision Synthesis RTL 2011a.61 and get the Resource Requirement Vector of each module. The available resources on XC5VLX50T and the resources required by the hardware modules of each application are described in the Area Report file. And we can estimate the RD of each PR region by Equation (6) proposed above.

Based on our proposed approach, efficient DPR implementation schemes can be generated automatically for designers. Particularly, there are complex design constraints and the non-concurrency relationships among candidate PR Modules. In the application of GSM, there are 88 vertices on the Intersection Graph, therefore the solution space is pretty large which is impossible to explore all by hand.

In this experiment, to compare the efficiency based on real-life applications, we have implemented Resource Optimization and RD Optimization to find efficient PR Module schemes with different optimization objectives for real-life applications. As shown in Table 3, Resource Optimization provides the PR Module schemes with the least possible area consumption while the RD cost of each PR region is no more than the RD limits which could be set by designers. RD Optimization tries to find the DPR scheme with the least reconfiguration delay within the resource limits which could be set by designers. We set the least area saving scales (PR CLBs/Total PR CLBs) and RD limits of each real-life application. For example, in the GSM case, we set its area saving scale

Table 3. Efficiency of the Two Optimization Algorithms on Real-life Applications

ID	Total CLBs	Resource Optimization					RD Optimization					
		Total CLBs with PR	Without Prefetching		With Prefetching		Without Prefetching			With Prefetching		
			RD(ms)	Avg RT (ms)	RD(ms)	Avg RT (ms)	Total CLBs with PR	RD(ms)	Avg RT (ms)	Total CLBs with PR	RD(ms)	Avg RT (ms)
gsm	7306	6465	8.6095	12.94	8.1908	13.07	6890	1.2893	13.03	6890	0.8369	13.16
adpcm	3479	3373	15.7932	6.33	6.4944	6.41	3383	7.3800	7.15	3373	6.4944	7.23
dfadd	529	456	7.4007	3.07	7.4007	3.09	466	4.7527	3.19	466	1.0727	3.21
dfmul	1175	979	9.1807	1.56	7.1807	1.56	1037	7.1438	2.08	1022	7.1438	2.13
dfdiv	2117	1925	15.7489	2.94	13.5084	2.95	1979	13.5084	3.17	1925	13.5084	3.20
dfsin	3231	2375	23.3533	5.61	23.3533	5.66	2375	23.3533	5.36	2636	20.1755	5.41

Table 4. The Comparison between our Approach and Two Heuristic Algorithms

ID	Our Approach			RD Greedy			Area Greedy		
	Total CLBs with PR	Optimized RD (ms)	Algor. Runtime (ms)	Total CLBs with PR	Optimized RD (ms)	Algor. Runtime (ms)	Total CLBs with PR	Optimized RD (ms)	Algor. Runtime (ms)
gsm	6890	0.8369	12.22	6890	5.3847	7.39	6547	8.1908	7.55
adpcm	3373	6.4944	7.23	3385	7.3800	3.60	3385	14.7600	3.78
dfadd	466	1.0727	1.64	466	1.0727	2.19	456	7.4007	1.92
dfmul	1022	7.1438	1.78	979	7.1807	1.68	1022	7.1438	1.75
dfdiv	1925	13.5084	3.20	1941	13.5084	1.90	1941	27.0167	1.71
dfsin	2636	20.1755	5.41	2636	22.7422	2.82	2375	23.3533	2.44
Avg. Ratio	1.02	0.51	1.58	1.02	0.63	1.05	1	1	1

to be 95% and set its RD limits to 10 *ms*. From Table 3, we could see that sometimes the method with prefetching has similar results to the one without prefetching. Let us talk about RD and RT, respectively. Because RD improvement due to prefetching only happens when two modules have sufficient intervals between them, if there are not enough intervals, there will be no chance to place the prefetching delays. Additionally, different optimization targets could also produce different effects. That's why sometimes the method with prefetching has no obvious effect. For RT, sometimes prefetching-incorporated version even takes less time than the prefetching-free version to finish. This is because most of the runtime of the whole algorithm is spent on solving the final MWISP. After incorporating our prefetching strategy, the weight of each vertex on the Intersection Graph is different from that in the prefetching-free version.

Though there is no available automatic approach for PR Module scheme generation to be compared with, we implement two heuristic algorithms to demonstrate the competence of our approach. The first one (RD Greedy) gives precedence to the vertex with the minimum RD among the vertices. Then the sorted vertices on the Intersection Graph are iteratively selected if the chosen vertex is not connected to any currently selected ones. Once the area saved by increasing PR Module groups enables the DPR implementation to fit into the XC5VLX50T chip, the vertex selection stops immediately. While the second heuristic algorithm (Area Greedy) gives precedence to the vertex with the maximum saved area (see Equation (3)) among the vertices that are not adjacent to any currently selected vertices and have the RD cost no more than its RD limits. All the three algorithms are incorporated with our step-wise adjustment prefetching strategy, as shown in Table 4. If we compare the RD only, our approach is best. It could obtain 2X-10X speedup than Area Greedy, and 6X maximum speedup than RD Greedy. This meets our expectation, because

Table 5. Required Resources of GSM

GSM	CLB Slices	DSPs	BRAMs
50 Mhz	7305	32	0
100 Mhz	7306	32	0
200 Mhz	7255	48	1
500 Mhz	7623	0	0
#Available resources on XC5VLX50T	7200	48	60

Table 6. The Comparison between our Approach and Two Heuristic Algorithms

GSM	Our Approach (RD<10ms, CLB<7200)			RD Greedy (RD<10ms, CLB<7200)			Area Greedy (RD<10ms, CLB<7200)		
	Total CLBs with PR	*Optimized RD (ms)*	*Algor. Runtime (ms)*	*Total CLBs with PR*	*Optimized RD (ms)*	*Algor. Runtime (ms)*	*Total CLBs with PR*	*Optimized RD (ms)*	*Algor. Runtime (ms)*
50 Mhz	7096	0.1410	12.28	6823	4.9143	6.70	6530	8.4073	7.86
100 Mhz	7122	0.1107	12.51	6890	5.3847	7.04	6547	8.1908	8.14
200 Mhz	7186	0.1063	10.13	7043	0.8620	8.48	6506	8.0103	7.71
500 Mhz	7143	0.9520	9.38	7143	4.9254	6.70	6718	7.5739	7.45
Avg. Ratio	-	0.04		-	0.50		-	1	

delay has higher priority than resource area. However, if we compare the total CLBs our approach is not always better. For example, Area Greedy can get fewer CLBs for these cases: *gsm, dfadd*, and *dfsin*. Obviously, it is estimated since the area is its priority. Even though our approach obtains more CLBs, we could achieve a good trade-off result. For example, when handling *adpcm* and *dfdiv*, our approach can get both good RD and less resource. This proves that our approach is more practical and can be used for the partial reconfiguration.

To evaluate the exploration of our approach under different performance constraints, we conduct a case study GSM with four execution frequencies (i.e. 50 MHz, 100 MHz, 200 MHz, and 500 MHz), respectively. In other words, we synthesized GSM with four different execution frequencies. All required resources of each version are shown in Table 5. As we can see from Table 5, Precision generates different optimized netlists which may vary greatly in the number of DSP and BRAM under different frequency configurations. Generally speaking, as the frequency increases, more on-chip resources are required by hardware modules. On the one hand, this increases the reconfiguration delay of the entire application. On the other hand, this requires better exploration ability to find feasible solutions that can fit on the chip. As we can see from Table 5, the available resources on XC5VLX50T are not sufficient to implement any of the four versions of GSM if DPR technology is not adopted. The experimental results of the comparison between our approach and the two algorithms are shown in Table 6. We can see that the heuristic methods often lead to the local optimum due to their limitation that the selected vertex will impose the no-intersection restriction on the later-on vertex selection process. Consequently, the greedy methods cannot handle the complex non-concurrency relations efficiently. Unlike the two heuristic strategies, our approach can address this challenge by providing an efficient way to optimally and automatically solve the PR Module generation problem so that the hardware resources could be utilized with less reconfiguration overhead.

Taking the PR Modules design proposed by our approach, we uniform the ports of each group of PR Modules and re-synthesize them separately with Precision Synthesis RTL 2011a.61. The generated netlist files can be fed to Xilinx's PlanAhead directly to build the DPR implementation

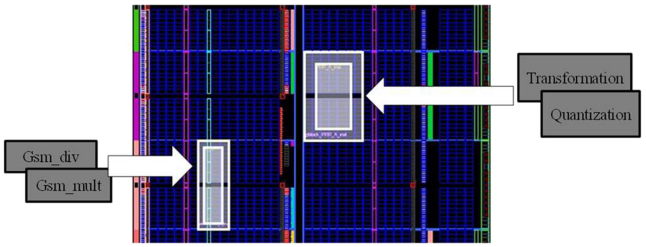

Fig. 14. Partial layout of GSM with 2 PR Regions in PlanAhead.

of the applications successfully. Taking GSM as an example shown in Figure 14, PlanAhead reports that each of the four DPR implementations consumes fewer resources than available resources on the XC5VLX50T chip. This means GSM can fit into XC5VLX50T by using DPR technology, saving area, power, and cost. The successful design of GSM from an advanced algorithm to DPR implementation demonstrates that our proposed approach can bridge the gap between high-level synthesis and the current DPR design flow successfully.

8 CONCLUSION

This paper proposed a novel approach for PR Module selection and clustering problems, which is essential in the modern DPR design flow. Firstly, this problem is formally represented as a Partial-Vertex Coloring Problem (PVCP) with constraints and then an independent set-based model is proposed to solve this problem. We formulate this PVCP as a Maximum-Weight Independent Set Problem (MWISP) with multiple objectives optimized and design constraints well considered. Additionally, we also proposed a step-wise adjustment strategy for configuration prefetching to improve the reconfiguration delay imposed by DPR. We demonstrate the usefulness of our approach in modern FPGA design flow by showing the design process of the applications in the real world. As high-level synthesis tools are becoming popular and playing an increasingly important role in hardware design nowadays, our approach that can fill the gap between high-level synthesis of algorithm description and DPR implementations is very promising.

REFERENCES

[1] L. Pezzarossa, A. T. Kristensen, M. Schoeberl, and J. Spars. 2018. Using dynamic partial reconfiguration of FPGAs in real-time systems. *Microprocessors and Microsystems* 61 (2018), 198–206.
[2] D. Koch et al. 2009. Hardware decompression techniques for FPGA-based embedded systems. *ACM Trans. Reconfigurable Technol. Syst.* (2009).
[3] Xilinx. 2013. Partial Reconfiguration User Guide (UG702), (2013).
[4] C. Conger, A. Gordon-Ross, and A. D. George. 2008. Design framework for partial run-time FPGA reconfiguration. In *Proc. ERSA* (2008), 122–128.
[5] P. Banerjee, M. Sangtani, and S. Sur-Kolay. 2011. Floorplanning for partially reconfigurable FPGAs. *Presented at IEEE Trans. on CAD of Integrated Circuits and Systems* (2011), 8–17.
[6] L. Singhal and E. Bozorgzadeh. 2006. Multi-layer floorplanning on a sequence of reconfigurable designs. *FPL*, 2006.
[7] K. Bazargan, R. Kastner, et al. 2000. Fast template placement for reconfigurable computing systems. *Presented at IEEE Design & Test of Computers* (2000), 68–83.
[8] A. Ahmadinia and J. Teich. 2003. Speeding up online placement for Xilinx FPGAs by reducing configuration overhead. In *Proc. VLSI- SOC* (2003), 118–122.
[9] H. Walder, C. Steiger, et al. 2003. Fast online task placement on FPGAs: Free space partitioning and 2D-hashing. In *Proc. IPDPS*, 2003.
[10] A. Ahmadinia, C. Bobda, et al. 2007. Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices. *IEEE Trans. Comput.* 56, 5 (2007), 673–680.
[11] L. Cheng and M. D. F. Wong. 2006. Floorplan design for multimillion gate FPGAs. *IEEE Trans. on CAD of Integrated Circuits and Systems* 25, 12 (2006), 2795–2805.

[12] R. Tessier, S. Swaminathan, et al. 2005. A reconfigurable, power-efficient adaptive Viterbi decoder. *Presented at IEEE Trans. VLSI Syst.* (2005), 484–488.
[13] Y. Hara, H. Tomiyama, et al. 2009. Proposal and quantitative analysis of the CHStone benchmark program suite for practical c-based high-level synthesis. *Journal of Information Processing* 17 (2009), 242–254.
[14] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt. 2006. Integrating physical constraints in HW-SW partitioning for architectures with partial dynamic reconfiguration. *Presented at IEEE Trans.* VLSI Syst. (2006), 1189–1202.
[15] F. Redaelli, M. D. Santambrogio, and D. Sciuto. 2008. Task scheduling with configuration prefetching and anti-fragmentation techniques on dynamically reconfigurable systems. In *Proc. DATE* (2008), 519–522.
[16] M. D. Santambrogio, M. Redaelli, and M. Maggioni. 2009. Task graph scheduling for reconfigurable architectures driven by reconfigurations hiding and resources reuse. In *Proc. ACM Great Lakes Symposium on VLSI* (2009), 21–26.
[17] S. Ghiasi and M. Sarrafzadeh. 2003. Optimal reconfiguration sequence management. *Proceedings of the 2003 Conference on Asia South Pacific Design Automation* (2003), 359–365.
[18] D. Koch, C. Beckhoff, and J. Torrison. 2010. Fine-grained partial runtime reconfiguration on Virtex-5 FPGAs. In *Proc. FCCM*, 2010.
[19] P. M. Pardalos and J. Xue. 1994. The maximum clique problem. *J. Global Optim.* 4, 3 (1994), 301–328.
[20] D. Warrier, W. E. Wilhelm, J. S. Warren, and I. V. Hicks. 2005. A branch-and-price approach for the maximum weight independent set problem. *Presented at Networks* (2005), 198–209.
[21] P. M. Pardalos and N. Desai. 1991. An algorithm for finding a maximum weighted independent set in an arbitrary graph. *Int. J. Comput. Math.* 38, (1991) 163–175.
[22] M. Pelillo. 2009. Heuristics for maximum clique and independent set. *Encyclopedia of Optimization* (2009), 1508–1520.
[23] I. R. Bomze, M. Pelillo, and V. Stix. 2000. Approximating the maximum weight clique using replicator dynamics. *IEEE Trans. Neural Networks* 11, 6 (2000), 1228–1241.
[24] S. Busygin, S. Butenko, and P. M. Pardalos. 2002. A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. *Presented at J. Comb. Optim.* (2002), 287–297.
[25] S. Yousuf and A. Gordon-Ross. 2010. DAPR: Design automation for partially reconfigurable FPGAs. In *Proc. ERSA* (2010), 97–103.
[26] http://en.wikipedia.org.
[27] K. Vipin and S. Fahmy. 2011. Efficient region allocation for adaptive partial reconfiguration. In *Proc. FPT* 2011.
[28] A. Lifa, P. Eles, and Z. Peng. 2016. A reconfigurable framework for performance enhancement with dynamic FPGA configuration prefetching. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 1 (2016), 100–113.
[29] J. Wang, Y. Kang, W. Wu, G. Xing, and L. Tu. 2021. DUPRFloor: Dynamic modeling and floorplanning for partially reconfigurable FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 8 (2021), 1613–1625.
[30] S. Brennsteiner, T. Arslan, and J. Thompson. 2019. Evaluation of partially constant, fine-grained, dynamic partial reconfigurable functions in FPGAs. *2019 International Conference on Field-Programmable Technology (ICFPT'19)*, 347–350.
[31] S. Satyendra Sahoo, T. D. A. Nguyen, B. Veeravalli, and A. Kumar. 2018. QoS-aware cross-layer reliability-integrated FPGA-based dynamic partially reconfigurable system partitioning. *2018 International Conference on Field-Programmable Technology (FPT'18)*, 233–236.
[32] Remigiusz Wiśniewski. 2018. Dynamic partial reconfiguration of concurrent control systems specified by petri nets and implemented in Xilinx FPGA devices. *IEEE Access* 6 (2018), 32376–32391.
[33] A. Biondi and G. Buttazzo. 2017. Timing-aware FPGA partitioning for real-time applications under dynamic partial reconfiguration. *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS'17)*, 172–179.

Received 19 May 2022; revised 20 August 2022; accepted 26 September 2022