Integrated Optimization of Partitioning, Scheduling, and Floorplanning for Partially Dynamically Reconfigurable Systems

Song Chen, Member, IEEE, Jinglei Huang, Xiaodong Xu, Bo Ding, and Qi Xu

Abstract—Confronted with the challenge of high performance for applications and the restriction of hardware resources for field-programmable gate arrays (FPGAs), partial dynamic reconfiguration technology is anticipated to accelerate the reconfiguration process and alleviate the device shortage. In this paper, we propose an integrated optimization framework for task partitioning, scheduling, and floorplanning on partially dynamically reconfigurable FPGAs. The partition, schedule, and floorplan of the tasks are represented by the partitioned sequence triple (P-ST) \((PS, QS, RS)\), where \((PS, QS)\) is a hybrid nested sequence pair for representing the spatial and temporal partitions, as well as the floorplan, and \(RS\) is the partitioned dynamic configuration order of the tasks. The floorplanning and scheduling of task modules can be computed from the P-ST in \(O(n^3)\) time. To integrate the exploration of the scheduling and floorplanning design space, we use a simulated annealing-based search engine and elaborate a perturbation method, where a randomly chosen task module is removed from the partition sequence triple and then reinserted into a proper position selected from all the \(O(n^3)\) possible combinations of partition, schedule, and floorplan. We also prove a sufficient and necessary condition for the feasibility of the partitioning of tasks and scheduling of task configurations, and derive conditions for the feasibility of the insertion points in a P-ST. The experimental results demonstrate the efficiency and effectiveness of the proposed framework.

Index Terms—Field-programmable gate arrays (FPGAs), floorplanning, partially dynamically reconfigurable (PDR), partitioned sequence triple (P-ST), partitioning, scheduling.

I. INTRODUCTION

In recent decades, reconfigurable hardware, and field programmable gate arrays (FPGAs) in particular, have received much attention because of their ability to be reconfigured to any custom desired computing architecture rapidly [1]. We can construct an entire hardware system on an FPGA chip or include an FPGA on a system-on-chip to provide hardware programmability. Traditionally, FPGAs are exploited using compile-time (static) reconfiguration, and the configuration remains the same throughout the running time of an application. To change the configuration, we have to stop the computation, reconfigure the chip by means of power-on resetting, and then start a new application. With the evolution of FPGA technology, dynamic reconfiguration (DR) has been developed, which provides more flexibility to reconfigure the FPGA by changing its predetermined functions at run-time. Through DR, one large application can be partitioned into smaller tasks; then the tasks can be sequentially configured at run-time. In this process, the entire chip must be reconfigured for each task; thus, significant reconfiguration overhead is incurred for loading the configuration each time [2].

To reduce the reconfiguration overhead and improve performance, several techniques are employed in modern FPGA architectures, such as partially DR (PDR), module reuse, and configuration prefetching, where PDR is a technique that reconfigures part of the FPGA at run-time while retaining normal operation of the remaining areas of the FPGA [3]. By applying the PDR technique, different tasks can be executed and configured in parallel, and a portion of the configuration latency can be hidden by careful scheduling of the configurations and executions of tasks. Hereafter, the FPGA, with the characteristic of PDR, is regarded as a PDR-FPGA.

To implement a large application composed of task modules on a PDR-FPGA, we must consider two problems: when the task modules should be configured and executed and where the task modules should be placed. The former is a scheduling problem, and the latter is a floorplanning problem. Unfortunately, both of them are NP-hard [4], [5]. In addition, to enable PDR, the reconfigurable resources on the FPGA are partitioned into several reconfigurable regions, which will be dynamically reconfigured to realize different tasks over time. Therefore, the number of partitioned reconfigurable regions and their sizes should be considered in this process.

A. Related Work

Many studies have focused on partitioning, scheduling, and floorplanning for PDR. Cordone et al. [6] proposed an
integer linear programming (ILP)-based method and a heuristic method for partitioning and scheduling task graphs (TGs) on PDR-FPGAs, where configuration prefetching and module reuse are considered to minimize the reconfiguration overhead. Purgato et al. [7] proposed a fast task scheduling heuristic to schedule the tasks in either the hardware or the software with minimization of the overall execution time on partially reconfigurable systems. However, the proposed method only focuses on generating reconfigurable regions to satisfy the resource requirements, which will easily cause the final result to fail to produce a valid floorplan. Jiang and Wang [8] proposed a network flow-based multilayer task partitioning algorithm to minimize the total communication costs (CCs) across temporal partitions. However, in this paper, the partitioning is simplified without considering the partial reconfiguration, and it is difficult to effectively estimate the CCs without the floorplan information. All the aforementioned works mainly focus on partitioning/scheduling of the tasks without consideration of the floorplan, which will often cause the schedule to fail to be floorplanned effectively, as they do not consider the resource constraints on the FPGA chips.

Deiana et al. [9] proposed a mixed-ILP (MILP)-based scheduler for mapping and scheduling applications on partially reconfigurable FPGAs, and if the schedule cannot be successfully floorplanned, the scheduler is re-executed until a feasible floorplan is identified. However, the time-consuming MILP-based method is impractical for large applications. In addition, scheduling and floorplanning are solved separately, which can cause large CCs in the spatial domain. Vasilik [10] proposed a temporal floorplanning method for solving the scheduling and floorplanning of dynamically reconfigurable systems. Yuh et al. [11], [12] modeled the tasks as 3-D boxes and proposed simulated annealing-based 3-D floorplanners to solve the floorplanning and scheduling problems of the tasks. However, the task modules are assumed to be reconfigured at any time and in any region, which may not match practical reconfigurable architectures. For example, in the Virtex 7 series FPGA chips from Xilinx [13], the reconfiguration partitions [dynamically reconfigurable regions (DRRs)] cannot be overlapped. Given scheduled TGs, many works have focused on the floorplanning of partially reconfigurable designs [14]–[19].

The design of reconfigurable systems with PDR generally involves partitioning, scheduling, and floorplanning of the tasks, which are interdependent considering CCs and system performance. Therefore, these three problems have to be solved in an integrated optimization framework to effectively explore the design space. However, the aforementioned works either solve the three problems sequentially, where, at most, a simple iterative refinement between scheduling and floorplanning is included, or solve only two of the three problems in an integrated framework.

B. Contributions

In this paper, we propose an integrated optimization framework for task partitioning, scheduling, and floorplanning on PDR-FPGAs. This paper expands our previous work [20]. Numerous theoretical analyses are provided for the feasibility of the P-STs (defined below). The main contributions of this paper are outlined as follows.

1) The term P-ST (PS, QS, RS) is proposed to represent the partitions, schedule, and floorplan of n task modules, where PS, QS, and RS are the sequences of n task modules. (PS, QS) is regarded as a hybrid nested sequence pair (HNSP) representing the floorplan with spatial and temporal partition, and RS is the partitioned dynamic configuration order of the tasks. The floorplan can be computed from the HNSP in $O(n \log n)$ time, and the schedule of tasks can be computed in $O(n^2)$ time by solving a single-source longest-path problem on a reconfiguration constraint graph (RCG), which is constructed based on P-ST and the task precedence graph.

2) We elaborate a perturbation method to integrate the exploration of the schedule and floorplan design space into simulated annealing-based searching. In the perturbation, a randomly chosen task module is removed from a P-ST and is then reinserted into the partitioned sequence triple (P-ST) at a proper position selected from all $O(n^3)$ possible insertion points, which are efficiently evaluated in $O(n^3)$ time based on an insertion point enumeration procedure.

3) We prove a sufficient and necessary condition for the feasibility of the partitioning of tasks and scheduling of task configurations, which is not included in [20], and derive conditions for the feasibility of the insertion points in a P-ST.

The experimental results demonstrate the efficiency and effectiveness of the proposed optimization framework.

The remainder of this paper is organized as follows. Section II describes the target hardware architecture and the problem definition. Section III discusses the representation of a sequence triple. Section IV shows the optimization framework to explore the design space of partitioning, scheduling, and floorplanning of task modules. Experimental results and conclusions are shown and discussed in Sections V and VI, respectively.

II. PROBLEM DESCRIPTION

A. Dynamically Reconfigurable Architecture

The dynamically reconfigurable system typically includes a host processor, an FPGA chip, an external memory, and the communication infrastructure among them. The host processor and communication infrastructure could be on-chip or off-chip. Presynthesized task modules are stored in off-chip external memory in the form of bitstreams. According to the scheduled sequence and floorplanned locations, the host processor deploys task modules on the FPGAs.

Modern FPGAs have evolved into complex heterogeneous and hierarchical devices. However, the basic logic cell still comprises configurable logic blocks (CLBs) [21]. In the target architecture, the CLB is the smallest reconfigurable element. Configuration bitstreams are transferred into FPGAs using one configuration port, which is an external Joint Test Action Group protocol or an internal configuration access port (ICAP).
TABLE I

| Notation | Description |
|----------|-------------|
| $m_i$    | 1 ≤ $i$ ≤ $n$ and $m_i$ is a task module. |
| $w_i, h_i$ | number of CLB rows and CLB columns required by $m_i$. |
| $c_i, t_i$ | configuration span (time) and execution span (time) of $m_i$. |
| $bc_i, bt_i$ | start configuration time / start execution time of $m_i$. |
| $drr_i$ | 1 ≤ $i$ ≤ $N$ and $drr_i$ is a dynamically reconfigurable region. |
| $drr(m_i)$ | the DRR where $m_i$ is located. |
| $tl_i$ | the $j$-th time layer in $drr_i$. |
| $tl(m_i)$ | the time layer where $m_i$ is located. |
| $c_{tl_i}$ | configuration span of time layer $tl_i$. |
| $bc_{tl_i}$ | start configuration time of time layer $tl_i$. |
| $CO[tl_i]$ | configuration order of time layer $tl_i$. |
| $li_{s_i}$ | start of lifetime of a time layer $tl_i$. |
| $li_{e_i}$ | end of lifetime of a time layer $tl_i$. |
| $LT[tl_i]$ | lifetime of a time layer $tl_i$. |

On the other hand, PDR is subject to the technology limitation, which is that the configuration process of a task module must not disrupt the execution of other task modules [14]. Thus, generally, DRRs, where the task modules are dynamically reconfigured in a manner similar to that of a context (time layer) switching mode, are used for implementing partial reconfiguration. On an FPGA chip, we can have multiple DRRs and one DRR can be dynamically reconfigured while the others continue to execute.

A DRR is a rectangular region on FPGAs because irregular-shaped reconfiguration regions (such as T or L shapes) can introduce routing restriction issues [13]. A task can be implemented as a rectangular hardware module on the FPGA. The module area represents the occupied CLBs (the number of rows and columns on the FPGA).

### B. Problem Definition

The design is composed of presynthesized tasks whose resource usage and internal routing are predetermined. Let $M = \{m_i\}_{1 \leq i \leq n}$ be a set of $n$ tasks. A task $m_i$, has a physical attribute vector, $(w_i, h_i, c_i, t_i)$. The meanings are shown in Table I. $c_i$ is proportional to the area and is estimated by $c_{clb} \times w_i \times h_i$, where $c_{clb}$ is the configuration time of a single CLB.

The data dependencies among these tasks are given as a task dependence graph, $TG = (V_{TG}, E_{TG})$, where $V_{TG} = M$ and $E_{TG} = \{(m_i, m_j)\}_{1 \leq i, j \leq n, i \neq j}$ and $m_i$ must end before $m_j$ starts. $TG' = (V_{TG}, E_{TG}')$ denotes the transitive closure of $TG$.

The partitioning, scheduling, and floorplanning of PDR are formulated as follows.

In the spatial domain, the $n$ tasks are partitioned into DRRs. Let $N$ be the number of DRRs.

**Definition 1:** The DRRs are denoted as $DRR = \{drr_i\}_{1 \leq i \leq N}$, $drr_i \subseteq M$, $\bigcup_{i=1}^{N} drr_i = M$, where $\forall i \neq j$, $drr_i \cap drr_j = \emptyset$. If $m_i \in drr_i$, we denote the DRR of $m_i$ as $drr(m_i) = drr_i$.

In the temporal domain, the $n$ tasks are partitioned into different time layers to reuse the resources of DRRs. A time layer is configured as a whole. Thus, in the same DRR, a time layer can only be configured after the completion of all the tasks in the previous time layer. Let $l_i$ be the number of time layers in $drr_i$.

**Definition 2:** The time layers are denoted as $TL = \{tl_i\}_{1 \leq i \leq N}$, $1 \leq j \leq l_i$, $tl_i \subseteq drr_i$, $\bigcup_{i=1}^{N} tl_i = drr_i$, where $\forall j_i \neq j_j$, $tl_{j_i} \cap tl_{j_j} = \emptyset$. If $m_i \in tl_i$, we denote the time layer of $m_i$ as $tl(m_i)$ and the total number of time layers as $|TL| = \sum_{i=1}^{N} l_i$.

For convenience, we define CO[$tl_i$] to be the configuration order of time layer $tl_i$, $1 \leq CO[tl_i] \leq |TL|$ and stipulate that CO[$tl_i$] < CO[$tl_{j_i+1}$], $1 \leq j < l_i$.

The configuration span (time) of the time layers in a DRR is proportional to the area of the DRR, we use $c_{drr}$ or $c_{tl_i}$ ($= c_{drr_i}$) to denote the configuration span of a time layer $tl_i$. To reduce the time complexity in the proposed integrated optimization framework, $c_{tl_i}$ is also under-estimated by summing the configuration time of task modules

$$c_{tl_i} = \sum_{m_i \in tl_i} c_{pt_i}$$

For the scheduling, we consider the following constraints.

1) The precedence constraints between tasks cannot be violated, that is, $\forall (m_i, m_j) \in E_{TG}$, $bt_i + t_i \leq bt_j$.
2) A task must be configured before execution, that is, $\forall 1 \leq i \leq n$, $bc_{tl(m_i)} + c_{tl(m_i)} \leq bt_i$.
3) Considering the technical limitation of only one configuration port, the configuration span of time layers must be nonoverlapped.
4) In the same DRR, a time layer can only be configured after the execution of all the tasks in the previous time layer because they share the same hardware resources.

The constraints for the floorplanning process are as follows.

1) Each DRR occupies a rectangular region, and all the rectangular regions of the DRRs should be placed without overlapping each other and should be within the FPGA chip area, which is defined by the chip width and chip height (fixed-outline constraint).
2) The task modules in the same time layer must be nonoverlapped and placed within their corresponding DRR.

Under the above constraints, we solve the partitioning problem to determine DRR and TL, the scheduling problem to determine the start configuration time and start execution time of the tasks (time layers), and the floorplanning problem to determine the floorplan of DRRs and the floorplan of tasks inside the DRRs.

We define schedule length to be the time from the beginning of the configuration process to the end of the executions of all tasks. The objective is to find a reasonable floorplan of tasks on a PRD-FPGA while minimizing the schedule length of designs as well as the CCs among tasks.

### III. Partitioned Sequence Triple

#### A. Representation

In this paper, a P-ST is proposed to represent the partitioning, scheduling, and floorplanning of tasks for PDR designs.
Definition 3: The P-ST is a 3-tuple of task sequences, (PS, QS, RS), where (PS, QS) forms an HNSP to represent the spatial partition (DRR), the temporal partition (time layer) and the floorplan of the task modules, and RS defines the configuration order of the time layers.

In a P-ST, task partitioning is constrained as follows.
1) The task modules in the same time layer will consecutively appear in PS, QS, and RS.
2) The task modules in the same DRR will consecutively appear in both PS and QS.

The structure of P-ST is illustrated as follows:

\[
\begin{align*}
\text{PS} &= (\ldots (m_p \ldots m_q)_{i1} \ldots (m_p \ldots m_{1})_{i2} \ldots ) \\
\text{QS} &= (\ldots (m_q \ldots m_p)_{j1} \ldots (m_q \ldots m_{1})_{j2} \ldots ) \\
\text{RS} &= (\ldots (m_p \ldots m_q)_{i1} \ldots (m_p \ldots m_{1})_{i2} \ldots )
\end{align*}
\]

In a P-ST, \( (\cdot )_i \) denotes the sequence of tasks in the time layer \( tl^i \) and \( [\cdot ]_i \) is the sequence of tasks in the DRR drr\(_i\).

An HNSP(PS, QS) imposes the position relationship between each pair of task modules as follows.

Definition 4: If \( tl(m_i) = tl(m_j) \) or \( drr(m_i) \neq drr(m_j) \), then \( (\ldots m_i \ldots m_j \ldots ) \rightarrow m_i \) is left to \( m_j \); \( (\ldots m_i \ldots m_j \ldots ) \rightarrow m_i \) is below \( m_j \).

Notice that the relationship between the task modules from different time layers in the same DRR is not defined, as there are no nonoverlapping constraints involved. Without loss of generality, we require the task modules in the same time layer to occur consecutively in PS and QS for clarity in representing the partitions of time layers and the floorplan of time layers.

The configuration order of a time layer can be represented by a configuration sequence RS, which is defined as follows.

Definition 5: Given an RS sequence, \((\ldots m_1 \ldots m_j \ldots )\), the configuration constraints are defined as follows.

1) If \( tl(m_i) = tl(m_j) \), then \( m_i \) and \( m_j \) are configured simultaneously, along with the corresponding time layer.
2) If \( tl(m_i) \neq tl(m_j) \), then \( tl(m_i) \) is configured before \( tl(m_j) \), and the configuration order relationship is \( \text{CO}[tl(m_i)] < \text{CO}[tl(m_j)] \).

In the RS, the ordering of task modules within a time layer makes no sense because the time layer is configured as a whole.

For example, a task graph TG with ten task modules is shown in Fig. 1 and a P-ST in this example is given as follows:

\[
\begin{align*}
\text{PS} &= ([1 2]_{i1} (9 10)_{i2} (8 7)_{i3} [6 4]_{i4} [3 5]_{i5}) \\
\text{QS} &= ([8 7]_{i1} [2 1]_{i2} (9 10)_{i3} [3 5]_{i4} [6 4]_{i5}) \\
\text{RS} &= ([1 2]_{i1} (3 5)_{i2} (6 4)_{i3} (7 8)_{i4} (9 10)_{i5})
\end{align*}
\]

(2)

For simplifying the notations, we use \( i \) to represent the task module \( m_i \) in the examples of P-ST. From the P-ST, we can obtain the corresponding configuration order and floorplan on the FPGA as shown in Fig. 2.

According to Definition 5 and the given configuration sequence \( RS([1 2]_{i1} (3 5)_{i2} (6 4)_{i3} (7 8)_{i4} (9 10)_{i5}) \), Fig. 2(a) shows the configuration order of the time layers. First, the time layer \( tl^1 \) from drr\(_2\) is configured and, second, the time layer \( tl^1 \) from drr\(_3\) can be configured during the executions of \( m_1 \) and \( m_2 \). The computation of the beginning configuration times of time layers will be discussed in Section III-C.

Considering the relationship between each pair of task modules defined in Definition 4, Fig. 2(b) shows the corresponding floorplan of task modules, where \( m_2 \) is below \( m_1 \) because they are in the same time layer \( tl(m_2) = tl(m_1) = tl^2 \), and \( m_7 \) is below \( m_6 \) because they are in different DRRs (drr\(_7\) = drr\(_1\) and drr\(_7\) = drr\(_2\)).
Knowing the dimensions of task modules, we can compute the floorplan from the HNSP in \(O(n \log \log n)\) time by solving the longest weighted common subsequence of \(PS\) and \(QS\) [22] hierarchically. We can compute the floorplan of task modules within every DRR to obtain the occupied resource arrays of DRRs, and then compute the floorplan of DRRs to determine the total resource usage by regarding each DRR as a whole. The computation of the schedule will be discussed in the following sections.

B. Feasibility of Partition and Configuration Order

Owing to the dependencies between tasks, not all the P-STs are feasible. In this section, we prove a sufficient and necessary condition for the feasibility of partitions and configuration order of task modules.

1) Lifetime of Time Layers: The task modules in a time layer can be executed only after the configuration of the time layer and will be destroyed while configuring the next time layer (if more time layers exist) in the same DRR. Consequently, we have the following definition.

**Definition 6:** Given the spatial partition DRR, the temporal partition TL, and the configuration order of the time layers, we define the **lifetime** of a time layer \(tl_i\), \(LT[tl_i] = \{lt_{-s_i}, lt_{-e_i}\}\), as follows. \(vt_{j} \in drr\)

\[
lt_{-s_i} = CO[tl_i], \quad \text{and} \quad lt_{-e_i} = \begin{cases} CO[tl_{i+1}], & 1 \leq j < li \\ \infty, & j = li. \end{cases}
\]

Note that the lifetime of a time layer is also the lifetime of the task modules in the time layer.

To discuss the feasibility of a configuration order, we define the dependencies between time layers based on the dependency graph of tasks given DRR and TL. A dependence graph \(LTG(VLTG, ELTG)\) is constructed as follows.

\[
VLTG = TL; \quad ELTG = \{(t_{i1}, t_{i2}) | \text{there exist } mk_i \text{ and } mk_j, \text{respectively, from } t_{i1} \text{ and } t_{i2} \text{ such that } (mk_i, mk_j) \in E_{TG}\}
\]

Note that \(E_{TG}\) is the edge set of the transitive closure of TG.

2) Dependencies Between Time Layers: Given a configuration order, the dependencies between time layers fall into two groups: 1) forward dependencies and 2) backward dependencies.

**Definition 7:** A dependence \((t_{i1}, t_{i2}) \in E_{LTG}\) is **forward** if \(CO[t_{i1}] < CO[t_{i2}]\), which indicates that the output of a task module in a time layer, \(t_{i1}\), is the input to a task module from a future time layer, \(t_{i2}\).

Forward dependencies are always feasible because even if the lifetime of a time layer ends, the computed data can be stored and used in the future.

**Definition 8:** A dependence \((t_{i1}, t_{i2}) \in E_{LTG}\) is **backward** if \(CO[t_{i1}] > CO[t_{i2}]\), which indicates that the output of a task module in a time layer \(t_{i1}\) is the input to a task module from an earlier configured time layer, \(t_{i2}\).

However, backward dependencies are infeasible if there is no overlapping between the lifetimes of the dependent time layers, \(t_{i1}\) and \(t_{i2}\), that is, \(lt_{-e_i} < lt_{-s_i}\). In this situation, \(t_{i2}\) is destroyed (replaced by a new time layer) before the time layer \(t_{i1}\) is configured, so the input to a task module is generated after the task module has been destroyed.

Fig. 3 shows examples of lifetimes of time layers and the dependencies between time layers. The spatial partition DRR and the temporal partition TL of the tasks are shown in Fig. 2, and the dependencies between tasks are shown in Fig. 1. The configuration order of the time layers is as follows: \(RS((1 2)^{3} (6)^{4} (3)^{1} (4)^{2} (5)^{3} (7 8)^{1} (9 10)^{2})\) (also shown as the x-axis in Fig. 3).

The time layers \(tl_1\) and \(tl_3\) have backward dependence because \(mk_5\) needs the data from \(mk_6\), as shown in Fig. 1, and their lifetimes \(LT[tl_1] = (2, 4)\) and \(LT[tl_3] = (5, \infty)\) are nonoverlapped. That is, \(mk_6\) in \(tl_1\) is destroyed \((tl_2\) in the same DRR \(drr_4\) has occupied the hardware resource) before the execution of \(mk_5\) in \(tl_2\). Consequently, \(mk_5\) will never receive the data from \(mk_6\), so the configuration order of task modules shown in Fig. 3 is infeasible.

3) Condition of Feasibility: We thus argue that the given spatial partition, temporal partition, and configuration order is feasible if a schedule of executions and configurations of task modules can be computed without consideration of resource constraints. We have the following theorem.

**Theorem 1:** The given spatial partition, temporal partition, and configuration order is feasible if and only if there are no backward dependencies between time layers that have no lifetime overlap.

**Proof:** Given a partition, a configuration order, and the task dependency graph, we can construct an RCG for scheduling the configurations of the time layers and the executions of the task modules, i.e., the computation of \(bt_i\), \(bc_i\), and \(bc_{tl}(m)\) defined in Section II-B.

\(RCG(V, E)\) is constructed by adding to the graph \((TG = (V, E))\) the vertex set \(V_{LTG}\) and three edge sets representing the scheduling constraints. \(V_{RCG} = V_{LTG} \cup V_{LTG}\), where \(V_{LTG}\) represents time layers and is defined in Section III-B. \(E_{RCG} = E_{TG} \cup E_{cr} \cup E_{ce} \cup E_{ec}\) and \(E_{cr}, E_{ce}, \text{ and } E_{ec}\) are defined as follows.

1) The set of edges represents the configuration order. \(E_{cr} = \{(t_{j1}, t_{j2})| CO[t_{j1}] < CO[t_{j2}])\}\.

2) The set of edges indicates that a task \(mk\) must be executed only after the configuration of the time layer, where \(mk\) is located. \(E_{ce} = \{(t_{j1}, mk)| t_{j1} \in V_{LTG}, mk \in V_{TG} \text{ and } t(mk) = t_{j1}\}\).
There must be a cycle in RCG. Notice that overlap.

A schedule can be computed only if the RCG is acyclic because a cycle produces a conflict in the constraints.

\[ E_{ec} = \{(mk_j, tl'_j) | \forall tl'_j \in V_{TG}, mk_j \in V_{TG}, \text{and tl}_j(mk) \text{ is the time layer before tl}'_j \text{ in drr} \}. \]

A cycle can be computed only if the RCG is acyclic because a cycle produces a conflict in the constraints.

If there are backward dependencies between the time layers that have no lifetime overlap, there will be a cycle in the RCG and hence the given partition and configuration order is infeasible.

A pair of time layers, \( tl'_j \) and \( tl''_j \), with \( CO[tl'_j] > CO[tl''_j] \), have a backward dependence if there are two task modules, \( mk_j \) and \( mk_{j-1} \), respectively, from \( tl'_j \) and \( tl''_j \) and there is a direct or indirect data dependence between them \((mk_j, mk_{j-1}) \in E_{TG'}\).

\[ V_{TG} \rightarrow V_{TG} \rightarrow \text{tg}(\{mk_j, mk_{j-1}\} \in E_{TG'} \}]\]

Fig. 4(a) shows an illustration of this, where a dashed arrow represents an edge or a path and solid arrows represent edges. While there is no overlap between the lifetime of \( tl'_j \) and \( tl''_j \), the hardware resources occupied by the time layer \( tl''_j \) must be reconfigured to be the next time layer in the same DRR, \( tl''_{j+1} \) before the configuration of \( tl'_j \), and there must be an edge from \( mk_j \) to \( tl''_{j+1} \) [shown in a bold dashed arrow in Fig. 4(a)] because a time layer can only be configured after the execution of all tasks in the earlier time layers in the same DRR.

Accordingly, a cycle is formed, which indicates the conflict of constraints.

\[ \text{only if:} \] Here, we show that if the given partition and configuration order is infeasible, there must be backward dependencies between the time layers that have no lifetime overlap.

If the given partition and configuration order is infeasible, there must be a cycle in RCG. Notice that \( V_{TG} = V_{TG} \cup V_{LTG} \). The subgraph induced by \( V_{LTG} \), which includes the edge set \( E_{cr} \) representing the configuration order of the time layers, is acyclic. The subgraph induced by \( V_{TG} \) (exactly TG), which includes the edge set \( E_{TG} \) representing the dependences between tasks, is also acyclic. Moreover, all the edges in \( E_{cr} \) are from \( V_{LTG} \) to \( V_{TG} \), which represents that a task must be configured before it is executed, and all the edges in \( E_{cr} \) are from \( V_{TG} \) to \( V_{LTG} \), which represents that a time layer can only be configured after the execution of all the tasks in the earlier time layers. Consequently, the cycle must include four parts: 1) a path (one or more edges) from \( E_{cr} \); 2) a path (one or more edges) from \( E_{TG} \); and 3) an edge from \( E_{cr} \); and 4) an edge from \( E_{TG} \).

Without loss of generality, we assume that the cycle includes a path from \( tl'_i \) to \( tl''_i \) and a path from \( mk_i \) to \( mk_j \), respectively, constructed by the edges from \( E_{cr} \) and \( E_{TG} \). The cycle must also include two edges: \((mk_i, tl'_i) \) and \((tl''_i, mk_j) \). Fig. 4(b) shows an illustration of this, according to the definition of \( E_{TG} \), \( mk_i \) is in \( tl''_i \) because we have the edge \((tl''_i, mk_j) \). On the other hand, the edge \((mk_j, tl''_i) \) indicates that \( tl''_i \) is configured after \( mk_j \) is executed, which means that \( mk_j \) must be located in the previous time layer of \( tl''_i \) in drr. \( tl''_{i-1} \). We can see that \( tl''_{i-1} \) and \( tl''_i \) have a backward data dependence and their lifetimes are nonoverlapping, as the region occupied by \( tl''_{i-1} \) has been reconfigured to be \( tl''_i \) before \( tl''_{i+1} \) is configured.

Note that if RS represents a topological ordering of TG, the partition and configuration order will always be feasible because there are no backward dependencies involved.

**Corollary 1:** Given a partition, a configuration order, and the task dependency graph, the RCG is acyclic if there is always lifetime overlap between time layers that have backward dependencies.

Fig. 5(a) shows the RCG of the feasible P-ST in (2), where \( RS = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11) \). If RS is changed to the configuration order in Fig. 3, \( (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11) \), the corresponding RCG is shown in Fig. 5(b), where a cycle \( tl''_1 \rightarrow tl''_2 \rightarrow tl''_3 \rightarrow tl''_4 \rightarrow tl''_5 \) is formed and no feasible schedule can be found.

**C. Computation of the Schedule**

The schedule can be computed by finding the longest paths on the RCG with edges weighted as follows:

\[ \forall tl'_j \in V_{LTG}, wt(tl'_j) = c_{tl'_j}, \forall mk_j \in V_{TG}, wt(mk_j) = t_j. \]
Let $s$ be the vertex corresponding to the time layer that is configured first (having zero in-degree in $\text{RCG}$), and $lp(v_i)$ denote the vertex-weighted longest-path from $s$ to a vertex $v_i$. The schedule (be $b_{ij}$ and $b_{ik}$) can be determined by computing $lp(t_{ij})$ and $lp(m_k)$, respectively.

The schedule length $T$ of the PDR system is the maximum of the paths, and can be calculated as follows: $T = \max_{1 \leq s < n}(lp(m_k) + t_k)$.

Given a feasible P-ST, we can construct RCG and compute the schedule in $O(n^2)$ time if the RCG is acyclic, where $n$ is the number of task modules.

IV. OPTIMIZATION FRAMEWORK

Definition 9: An insertion point in the P-ST ($PS, QS, RS$) is defined as a four-tuple, $(p, q, r, tl_i)$, where $p$, $q$, and $r$ are the positions immediately after the $p$th task module in $PS$, the $q$th task module in $QS$, and the $r$th task module in $RS$, respectively, and $tl_i$ is the $j$th time layer in $drr_s$, $p = 0$ (or $q = 0$ or $r = 0$) indicates the position before the first task module of the sequence.

In Section IV-B2, we will discuss the feasibility and types of insertion points in detail.

A. Overall Design Flow

In this paper, we modify the perturbation method, insertion-after-remove (IAR) in [23], to explore the design space of the schedule and floorplan in a simulated annealing-based search. With the IAR operation, we can perturb the partitioning, scheduling, and floorplanning of task modules simultaneously. The detailed steps are as follows.

1) Select and remove a task module $m_k$ randomly and then compute the floorplan and schedule of task modules without the removed task module $m_k$.

2) Select a fixed number of feasible candidate insertion points, $SCIP = \{(p, q, r, tl_i)\}$, for $m_k$ by rough evaluations of all the feasible insertion points.

3) Choose the best insertion point from $SCIP$ for the removed task module $m_k$ by accurate evaluations.

In step b, the feasible insertion points are evaluated by the linear combination of resource costs, schedule length, and CCs. In this step, the resource costs are calculated accurately. To reduce the time complexity, the CC is calculated roughly without updating the floorplan and schedule of task modules, and the schedule length is roughly evaluated by under-estimating the configuration spans of time layers using (1). In step c, all the insertion points in $SCIP$ will be evaluated accurately based on the entire floorplan considering the CCs, and the best one will be chosen as the candidate insertion point. The feasibility of insertion points will be discussed in Section IV-B.

In the experiments, we set the size of $|SCIP|$ at 15. The objective function $Cost$ is defined as the linear combination of the area cost (AC), which depends on the dimensions of all occupied resources ($Col \times Row$), the schedule length ($T$), and the CC

$$Cost = \alpha \times AC + \beta \times T + \gamma \times CC.$$  

The detailed steps are as follows.

B. Feasible Insertion Points in P-ST

Generally, given a P-ST of $n - 1$ task modules, there are a total of $O(n^2)$ insertion points for inserting a task module. However, when considering Theorem 1 and the definition of P-ST, some insertion points are infeasible. Here, we discuss the feasibility of insertion points in P-STs.

1) Lifetime Overlap Constraint: First, inserting $m_k$ could introduces new backward dependencies between time layers. To ensure the lifetime overlap between the backward-dependent time layers, we have the following corollary from Theorem 1.

Corollary 2: The lifetime of a time layer, $tl_i$, where $m_k$ is inserted, must satisfy the following condition:

$$\begin{align*}
&\{lt_{-ls}^j \leq \min_{v_i} \{ v_{i1} \geq m_k \} \land (m_k, m_k) \in E_{\text{TG}} \} \land \{lt_{ee}^j \} \land \{lt_{+ls}^j \} \\
&\{lt_{+ls}^j \} = \{lt_{-ls}^j \} \cup \{lt_{ee}^j \} \\
&\{lt_{ee}^j \} = \{lt_{-ls}^j \} + \{lt_{+ls}^j \} + 1
\end{align*}$$

The minimum lifetime constraints ensure a lifetime overlap between any two time layers that have a backward dependence. This constraint cannot be violated after $m_k$ is inserted back into the P-ST. In Section IV-B3, an example is provided.

2) Feasibility of Insertion Points: Let $ps[i], qs[i], and rs[i], 1 \leq i < n$ represent the $i$th task in $PS, QS$, and $RS$, respectively, with $m_k$ removed. For each possible insertion point $(p, q, r, tl_i)$, there exist three possible types of optional partitions to insert $m_k$ depending on the time layer $tl_i$.

Type-1: Create a new time layer in a new DRR, $t_{\text{new}}$, for $m_k$. In this case, $(p, q, r)$ must be located within the boundary of task sequences corresponding to different DRRs in P-ST, i.e.,

$$drr(ps[p]) \neq drr(ps[p + 1]), drr(qs[q]) \neq drr(qs[q + 1])$$

and $tl(r) \neq tl(r + 1)$.

(8)

Without loss of generality, we assume that $ps[0] = ps[n] = -1$, and that $drr(-1)$ and $tl(-1)$ correspond to a virtual DRR and to virtual time layers, respectively. $qs[0], qs[n], rs[0], and rs[n]$ are dealt with similarly.
This type of insertion point will not change the lifetimes of any other time layers according to Definition 3. Consequently, if the constraint (6) in Corollary 2 is satisfied, then \((p, q, r, t_{\text{new}}')\) is feasible. Note that the new generated time layer \(t_{\text{new}}'\) is configured between \(t_l(rs[p])\) and \(t_l(rs[r+1])\).

Type-2: Create a new time layer, \(t_{\text{new}}'\), in an existing DRR, \(d_{rr}\), for \(m_k\). In this case, the insertion point \((p, q, r, t_{\text{new}}')\) must be located within the boundary of task sequences corresponding to different time layers, i.e., there is a combination \((p', q', r') \in \{p, p+1\} \times \{q, q+1\} \times \{r, r+1\}\) such that 
\[
drr(ps[p']) = drr(qs[q']) = drr(rs[r']) \neq drr(-1),
\]
and 
\[
t_l(ps[p]) \neq t_l(ps[p+1]), t_l(qs[q]) \neq t_l(qs[q+1])\] and \(t_l(rs[r]) \neq t_l(rs[r+1])\).

This type of insertion point will change the lifetime of the time layer that is immediately before \(t_{\text{new}}'\) in \(d_{rr}\). An insertion point \((p, q, r, t_{\text{new}}')\) is feasible if the constraints in both Corollary 2 and Corollary 3 are satisfied. Note that the new generated time layer \(t_{\text{new}}'\) is configured between \(t_l(rs[r])\) and \(t_l(rs[r+1])\).

Type-3: Insert \(m_k\) into an existing time layer, \(t_l'\). In this case, an insertion point \((p, q, r, t_l')\) must satisfy the condition that there is a combination \((p', q', r') \in \{p, p+1\} \times \{q, q+1\} \times \{r, r+1\}\), such that 
\[
t_l(ps[p']) = t_l(qs[q']) = t_l(rs[r']) \neq t_l(-1)\]
This type of insertion point will not change the lifetime of any other time layers. \((p, q, r, t_l')\) is feasible if the constraint (6) in Corollary 2 is satisfied.

3) Example: Given the task dependencies shown in Fig. 1, we have the following P-STS with \(m_8\) removed:
\[
\begin{align*}
(1, 2, 1, 9, 10, 5, 4, 2, 7, 1) & \quad (6, 4, 3, 1, 5, 2) \\
(7, 5, 3, 2, 8, 1) & \quad (3, 5, 2) \\
(1, 2, 3, 6, 4, 2, 7, 1) & \quad (6, 4, 3, 1, 5, 2) \\
(1, 2, 3, 6, 4, 2, 7, 1) & \quad (6, 4, 3, 1, 5, 2) \\
\end{align*}
\]
(9)
Fig. 6 (black edges) shows the lifetime of time layers with the task module \(m_8\) removed.

According to Corollary 2, for the lifetime, \((l_t.s', l_t.e')\), of a time layer \(t_l'\), where the removed task module \(m_8\) will be inserted, we have the following basic constraints.

1) Because \((m_8, m_4)\) and \((m_8, m_1)\) are in \(E_{TG}\), and \(m_4\) is in \(t_l'\), then \(l_t.s'\) must satisfy 
\[
l_t.s' \leq \min(l_t.e, l_t.e') \]
As shown in Fig. 6, LT\([t_l'] = (5, \infty)\) and LT\([t_l' = (7, \infty)\), thus, \(l_t.s' \leq \min(5, \infty) = \infty\), which means that there is no constraint on the beginning of the lifetime.

2) Because \((m_5, m_8)\) and \((m_7, m_8)\) are in \(E_{TG}\), and \(m_5\) is in \(t_l' \) and \(m_7\) is in \(t_l' \), then \(l_t.e'\) must satisfy 
\[
l_t.e' \geq \max(l_t.s', l_t.s') \]
As shown in Fig. 6, LT\([t_l' = (3, \infty)\) and LT\([t_l' = (6, \infty)\), thus, \(l_t.e' \geq \max(3, 6) = 6\).

Consequently, the task module \(m_8\) must be inserted into a time layer whose lifetime ends after 6.

Here, we assume that \(m_8\) is inserted back into an insertion point, (6th, 8th, 5th, \(t_l'\)), of the P-STS(PS, QS, RS) shown in (9).
of the lifetime is \( t_{l_{e}}^{2} = \infty \geq 6 \), thus, the insertion point (6th, 8th, 5th, \( tl_k \)) is feasible.

4) Discussion of the Reachability of the Solution Space:

**Theorem 2:** Any two feasible solutions of \( n \) task modules, represented by P-STs, are reachable to each other through at most \( 2n \) feasible solutions generated by iteratively removing and reinserting a task module.

**Proof:** As discussed in Section III-B3, if the RS is a topological order of the task dependence graph, the P-ST will be reachable. From any feasible solution P-ST': \((P', Q', R, S')\), we are able to reach another feasible solution P-ST: \((P, Q, R, S)\), where RS is a topological order, by iteratively removing and reinserting some task module. If we select the task modules for removing and inserting back in the order of RS', no backward dependencies between time layers will be introduced because all the time-layer dependencies introduced by the moved task module are forward ones (per Definition 7). Consequently, all the intermediate solutions generated from P-ST, assumed to be P-ST1, P-ST2, ..., P-STk (\( k \leq n \)), will always be reachable according to Theorem 1, and from one solution P-ST' \((P', Q', R, S')\) we can reach any other solution P-ST \((P, Q, R, S)\), where RS is a topological order of TG in at most \( k \) (\( k \leq n \)) steps: P-ST', P-ST1, P-ST2, ..., P-STk = P-ST.

On the other hand, we can reach a generic feasible solution P-ST'' \((P'', Q'', R, S''\)) with no constraints on \( S'' \) from any solution P-ST \((P, Q, R, S)\), where RS is a topological order through at most \( n \) feasible solutions, which can be obtained by inverting the sequence of removing and reinserting operations from P-ST to P-ST'.

Therefore, starting from one generic solution P-ST we can reach another generic solution P-ST'' (through a solution P-ST in which RS is a topological order of TG) in at most \( 2n \) steps.

C. Evaluation of Insertion Points

1) Computation of \( T \): To reduce the complexity, we use the area sum of the task modules to underestimate the configuration span of the time layers instead of accurately computing the configuration span of the DRRs. In this section, we discuss a method (used in step b of the IAR perturbation shown in Section IV) to evaluate, in amortized constant time, the schedule length while inserting a task module into an insertion point.

After a task module \( m_k \) is removed from the P-ST, RCG is updated by removing some edges related to time layers according to the following two situations.

1) If \( m_k \) is the only task module in \( tl(m_k) \), we remove the vertex \( tl(m_k) \) along with its incoming and outgoing edges and the edges between \( m_k \) and any vertices that represent time layers.

2) If \( m_k \) is not the only task module in time layer \( tl(m_k) \), the weight of vertex \( tl(m_k) \) will be subtracted by the configuration time span of \( m_k \) (\( c_k \)) and the edges between \( m_k \) and any vertices that represent time layers are removed.

Let RCG\(^0\) be the updated RCG. To simplify the description, we add to RCG\(^0\) a source vertex \( v_s \) with outgoing edges to all the task modules that have zero in-degree and a sink vertex \( v_t \) with incoming edges from all the task modules that have zero out-degree. Both \( v_s \) and \( v_t \) have zero weight. Let rRCG\(^0\) be the graph obtained by reversing all the edges of RCG\(^0\).

We precompute the longest paths from \( v_s \) to each vertex \( v_i \in RCG\(^0\) \( \in RCG\(^0\) \), denoted as \( lp_0(v_i) \), and the longest paths from \( v_t \) to each vertex, \( lp'_0(v_i) \), based on rRCG\(^0\) in \( O(n^2) \) time using the longest-path algorithm on directed acyclic graphs.

To evaluate the schedule length \( T \) for inserting \( m_k \) into a feasible insertion point \((p, q, r, tl')\) in P-ST, a new RCG RCG\(^new\) is generated. Let \( T_0 \) be the longest path from \( v_s \) to \( v_t \) in RCG\(^0\). \( T \) can be roughly and incrementally evaluated from the longest paths in RCG\(^0\) and RCG\(^0\) by considering only the paths passing through the vertex \( tl' \) or \( m_k \) because all the changed edges are related to either \( tl' \) or \( m_k \).

\[
T = \max \left( T_0, \sum_{i} \sum_{j} \left( l_{p_0}(\text{tl'}_i) + l_{p'_0}(\text{tl'}_j) + c_{ij} l_{p_0}(m_k) + l_{p'_0}(m_k) + t_k \right) \right)
\]

where \( l_{p_0}(\text{tl'}_i), l_{p'_0}(\text{tl'}_j), l_{p_0}(m_k), \) and \( l_{p'_0}(m_k) \) are incrementally computed based on \( lp_0(v_i) \) and \( lp'_0(v_i) \), for the three types of partitions discussed in Section IV-B.

**Type-1:** Both a new DRR \( d r_t \) and a new time layer \( tl' \) are created for \( m_k \), and the RCG\(^new\) can be constructed by adding three edges (red dotted lines) in RCG\(^0\), as shown in Fig. 7(a).

**Type-2:** A new time layer \( tl' \) is created in an existing DRR \( d r_t \), for \( m_k \), and the RCG\(^new\) can be constructed by adding some edges (red dotted lines) in RCG\(^0\), where there are three situations, respectively, shown in Fig. 7(b)–(d). In the situation shown in Fig. 7(d), there are at least three time layers \( tl'_{-1}, tl'_{0}, \) and \( tl'_{+1} \) in DRR \( d r_t \).

**Type-3:** \( m_k \) is inserted into an existing time layer \( tl' \) in the DRR \( d r_t \). There are two situations for updating RCG\(^new\), respectively, shown in Fig. 7(e) and (f).

In the RCG\(^new\), the computations of \( l_{p_0}(\text{tl'}_j) \) and \( l_{p_0}(m_k) \) are summarized as follows:

\[
l_{p_0}(\text{tl'}_j) = \max \left\{ l_{p_0}(t_{l_{e}}^{0}) + c_{ij}, \right. \quad \left. l_{p_0}(t_{l_{e}}^{0}) + c_{ij} \max_{m_i, e} \left[ l_{p_0}(m_i) + t_i \right] \right\}
\]

\[
l_{p_0}(m_k) = \max \left\{ l_{p_0}(m_k), \right. \quad \left. l_{p_0}(m_k) + c_k \max_{e} \left[ l_{p_0}(m_i) + t_i \right] \right\}
\]

In the second part of (11), the computation can be performed in amortized constant time because the total number of \( m_k \) is at most \( n - 1 \).

The longest paths \( l_{p'_0}(\text{tl'}_j) \) and \( l_{p'_0}(m_k) \) in the reverse graph of RCG\(^new\), rRCG\(^new\), can be calculated in constant time as follows:

\[
l_{p'_0}(m_k) = \max \left\{ l_{p'_0}(m_k), \right. \quad \left. l_{p'_0}(m_k) + c_{i_{l_{e}}^{0} j_{l_{e}}^{0}} \max_{e} \left[ l_{p'_0}(m_i) + t_i \right] \right\}
\]

\[
l_{p'_0}(\text{tl'}_j) = \max \left\{ l_{p'_0}(t_{l_{e}}^{0}) + c_{ij}, \right. \quad \left. l_{p'_0}(t_{l_{e}}^{0}) + c_{ij} \max_{m_i, e} \left[ l_{p'_0}(m_i) + t_i \right] \right\}
\]
Consequently, the evaluation of $T$ in (10) for all the possible $O(n^3)$ insertion points can be completed in $O(n^3)$ time.

2) Computation of Communication Costs $CC$: For an edge $(m_i, m_j) \in E_{TG}$, the CC between $m_i$ and $m_j$, $CC(m_i, m_j)$, can be evaluated as follows:

$$CC_{m_i, m_j} = w_{i,j} \cdot \left\{ a_d \cdot (|x_i - x_j| + |y_i - y_j|) + \beta_t \cdot (bt_j - et_i) \right\}$$

(15)

where $w_{i,j}$ is the communication requirement between $m_i$ and $m_j$, and $(x_i, y_i)$ and $(x_j, y_j)$ are, respectively, the coordinates of $m_i$ and $m_j$ on the FPGA chip. $et_i$ and $bt_j$ are, respectively, the ending execution time of $m_i$ and the starting execution time of $m_j$.

If the two task modules span multiple time layers, we project the two task modules onto one time layer and calculate the Manhattan distance. In the experiments, the parameters $a_d$ and $\beta_t$ are set based on the temporal and spatial dimensions.

1) If $m_i$ and $m_j$ are partitioned into the same time layer, we set $a_d$ and $\beta_t$ to 1 and 0, respectively.
2) If $m_i$ and $m_j$ are partitioned into different time layers in a DRR, then $a_d$ and $\beta_t$ will be set to 1 and 1.5, respectively.
3) If $m_i$ and $m_j$ are partitioned into different DRRs, $a_d$ and $\beta_t$ will be set to 3 and 1.5, respectively.

The communication cost $CC_{m_k}$ can be calculated as follows:

$$CC_{m_k} = \sum_{(m_i, m_j) \in E_{TG}} CC(m_i, m_k) + \sum_{(m_k, m_i) \in E_{TG}} CC(m_k, m_i).$$

(16)

Thus, the total CC can be evaluated as follows: $CC = \sum_{m_k \in V_{TG}} CC_{m_k}$.  

3) Computation of Area Costs: Let $Row_0$ and $Col_0$, respectively, be the number of rows and the number of columns of the CLB array available in the FPGA chip. The AC is calculated using a method similar to that in [23]

$$AC = E_{Row} + E_{Col} \cdot \lambda + C_1 \cdot \max(E_{Row}, E_{Col} \cdot \lambda)$$

(17)

where $E_{Col} = \max(Col - Col_0, 0)$ and $E_{Row} = \max(Row - Row_0, 0)$ are, respectively, the excessive columns and rows required by the current solution, $\lambda = Row_0/Col_0$, and $C_1$ is a user-defined constant.

The Col and Row are evaluated in constant time by a method similar to that in [23] and [24]. Considering a feasible insertion point $(p, q, r, tl_i)$, we can first calculate the row and column within the DRR using a method similar to [24] in amortized constant time because the time layers in the DRR are represented by a multilayer sequence pair. After the dimensions of the DRR are obtained, we can use a similar method to that in [23] to calculate the total rows and total columns required by inserting the removed task module into the insertion point $(p, q, r, tl_i)$ in amortized constant time.

D. Complexity Analysis

In this section, we analyze the complexity of an IAR perturbation on a P-ST. For an insertion point $(p, q, r, tl_i)$, if conditions for both type-1 and type-3 in Section IV-B2 are satisfied, there will be at most five possible time layer candidates: one type-1 insertion point, where $tl_i$ is a new created time layer in a newly created DRR; two type-2 insertion points, where $tl_j$ is a newly created time layer in an existing DRR; two type-3 insertion points, where $tl_k$ is an existing time layer. Consequently, there are $O(n^3)$ possible insertion points for a removed task module.

The feasibility of an insertion point can be judged in amortized constant time, as both the minimum lifetime in Corollary 3 and the lifetime constraint in Corollary 2 can be computed previously in $O(n^2)$ time.

For each insertion point, the cost function shown in (5) can be calculated in $O(n)$, where the AC and the schedule length $T$
can be evaluated in amortized constant time, and the complexity of computing CCs is \(O(n)\). Consequently, the evaluation of \(O(n^3)\) insertion points can be performed in \(O(n^4)\), where \(n\) is the number of task modules.

V. EXPERIMENTS AND RESULTS

A. Experimental Setup

The proposed method has been implemented in C-language on a Linux 64-bit workstation (Intel 2.0 GHz, 62 GB RAM). The input consists of a set of tasks with dependencies given in a TG, the resource requirements, configuration time \(c_{ti}\), and execution time \(e_{ti}\) of each task module \(m_i\). The benchmarks are constructed by combining the TGs generated by TGs for free (TGFF) [25] and the standard floorplanning benchmark, GSRC suites [26]. The dimensions (width and height) of task modules are from GSRC benchmarks and the width and the height of a task module, respectively, define the number of CLB columns and the number of CLB rows on an FPGA chip. The task dependencies are generated by TGFF. The execution times of task modules and the communication requirements between task modules are randomly generated. Note that we are considering only the allocation of CLB resources. We also generate two benchmarks from a popular convolutional neural network model, AlexNet [27]. One is AN_Part1, which includes two convolutional layers and one pooling layer of the model. The other is AN_Part2, which includes three convolutional layers and two pooling layers of the model. The task module in convolutional layers performs a convolution operation over a feature map with a specified convolutional kernel and the task module in pooling layers performs a pooling operation over all the output feature maps of a convolutional layer. The task modules are stipulated to consume the least hardware resources, and the execution times are estimated based on a frequency of 200 MHz.

Table II lists the benchmark parameters used in our experiments. Each randomly generated benchmark has three different implementations, which have the same number of task modules but different task dependencies. \#V and \#E are the number of vertexes and edges in TG, respectively. The columns VWR and EWR are the range of random values for execution time of tasks and communications between tasks, respectively. The column CPT shows the longest paths of the TGs and the vertexes are weighted by the execution times.

We take one of the widely used Xilinx Virtex 7 series FPGA chips, XC7VX485T, as the target chip. There are about 37,950 CLBs and the ratio of rows to columns is 3:1. Therefore, the CLB array in XC7VX485T has approximately 350 rows and 117 columns. Configuring all resources of XC7VX485T requires 50.7 ms through the interface ICAP with the maximum bandwidth of 3.2 Gb/s [13]. Thus, we consider the time overhead of reconfiguring one CLB to be 0.0013 ms, and the configuration time span of a task module is proportional to the module area because the configuration time is proportional to the synthesized bitstream of a design.

In the experiments, the sum of the three coefficients in \(5\) is set to one and the cost values are normalized. To avoid violating the resource constraints in the final solutions, the coefficient of AC, \(\alpha\), is the dominant factor and is set to around 0.8 to ensure almost 100% success rate. The coefficients of schedule length \(T\) and CCs, \(\beta\) and \(\gamma\), are, respectively, set to 0.15 and 0.05. We can make a tradeoff between the schedule length and the CC by changing \(\beta\) and \(\gamma\) because the AC computed by \(17\) will be zero if the resource constraint is satisfied. The initial solution is generated randomly and each task module is partitioned into an individual time layer in a common DRR and the configuration order is a topological ordering of the TG. The starting temperature, the ending temperature, and the cooling ratio of the simulated annealing are, respectively, set to 2000, 0.01, and 0.98. The iteration number in each temperature is set to 50 for the benchmarks with less than 50 tasks. For other benchmarks, the iteration number is increased slightly along with the increasing number of task modules.

Table III shows the experimental results. The proposed integrated optimization framework is called Int_PSF. We execute the proposed method ten times independently for each benchmark, and list the average results. The columns in Table III are organized as follows. \(T\) is the schedule length of each design, which corresponds to the longest paths in the RCGs. RunT is the run-time of the optimization framework. \#succ is the success rate of floorplanning. \(N\) is the number of DRRs. CC indicates the CCs calculated based on the temporal and spatial dimensions.

As a baseline situation, we solve the simplified scheduling problem, where the hardware resources are considered unlimited and every task module occupies an individual DRR, using an ILP formulation similar to that in [9]. The obtained \(T\) indicates the schedule length in the case when the configuration times are maximally hidden. In the experiment, Gurobi [28] is used as the ILP solver. The column ILP shows the results, where the ‘w’ means the solutions are incumbent within 1 h. The results demonstrate that Int_PSF can effectively hide
the reconfiguration time overhead in the dependency dominated TGs (the designs having long CPT) under the resource constraints.

To explore the effectiveness of the proposed integrated optimization framework, we perform a two-phases approach (TP_PSF) for partitioning, scheduling, and floorplanning of task modules. In the first phase, we evaluate the hardware resources for a time layer using the sum of the task module area instead of calculating a floorplan of the task modules. Consequently, the order of task modules within a time layer (in PS and QS) makes no sense, but the partitioning of task modules and scheduling of the time layers are solved in an integrated optimization framework. In the second phase, a simulated annealing-based search is used for placing the task modules and the DRRs. The initial floorplan of DRRs and the initial floorplan of time layers are generated randomly. In the simulated annealing, the IAR perturbation is adopted for a DRR (as a whole) or a task module, and the task modules are removed and inserted only within a time layer to keep the partitioning unchanged. As shown in Table III, Int_PSF achieves a success rate of 100%, whereas the two-phase method TP_PSF achieves only a success rate of 72.8% in the case when the schedule length and CC are almost the same.

In benchmark t50-2, the obtained by the proposed method is obviously higher than the baseline situation because t50-2 has high parallelism, whereas FPGA hardware resources constrain the parallel executions of the tasks. Table IV shows the detailed experimental results on the relationships between FPGA resources and performances for applications with different degrees of parallelism.

According to our experimental results, the CC can be reduced by 33.18% on average by considering the CCs in the optimization framework. The detailed data are listed in the supplementary material (Table I).

To evaluate the impacts of FPGA resources on the schedule length of designs, we perform the experiments for all test benchmarks under different FPGA resource constraints, which are set as 3/4x, 1.0x, 3/2x, and 2.0x of the targeted FPGA architecture (37 950 CLBs). We execute Int_PSF10 times independently for each benchmark, and show the average results in Table IV. The column resource represents the amount of resources for the target FPGA architecture. As shown in Table IV, for all the benchmark circuits, with increasing FPGA resources, the trends of schedule length T and CC decrease to be gentle. On the one hand, DRRs can be executed in parallel and configured independently, thus the configuration latency can be effectively hidden in the executions of tasks. With increased FPGA resources, the number of DRRs N is increasing overall, which will maximize the parallel execution of tasks and increase the possibility of hiding the configuration of tasks. On the other hand, the schedule length should be greater than that in the baseline situation shown in Table II. For the benchmark t200-1, which involves a long CPT and fewer data dependencies, the schedule length T remains the same with increasing FPGA resources and is close to the length of the corresponding CPT because the configuration latency is effectively hidden. Furthermore, Int_PSF achieves 100% success rate for the different FPGA resource constraints, which demonstrates the effectiveness of the method.

As discussed in Section IV-C, to reduce the complexity, we use the area sum of the task modules to underestimate the configuration span of the time layers instead of accurately computing the configuration span of the DRRs, which should be computed using the DRR area. According to our experimental results, for the same solution, if the DRR area is used instead of summing the module area, the schedule length is increased by 5%. In the optimization framework, if we use the DRR area to accurately estimate the configuration span of time layers, the obtained schedule length is increased only by negligible 1%. The possible reason is that the configurations of the time layers are well hidden in the execution of task modules. A detailed analysis is included in the supplementary material (Fig. 1).

C. Vertically Aligning DRRs to Reconfigurable Frames

As was demonstrated in [13], when applying the Reset After Reconfiguration methodology, a DRR must vertically align to reconfigurable frames (aligning vertically to clock regions) for 7-series and Zynq-7000 AP SoC devices. When the height of the DRRs is vertically aligned to the reconfigurable frames, the height of the reconfiguration frame (50 rows of CLBs) is adopted as the measurement unit of DRR height. Table V shows the experimental results of with/without

| Benchmark | imp | I/LP | TP_PSF | Int_PSF |
|-----------|-----|------|--------|---------|
| t50       | 1   | 299.45° | 40%   | 462.7  |
|           | 2   | 468.82 | 20%   | 688.4  |
|           | 3   | 604.90 | 80%   | 616.4  |
| t100      | 1   | 706.22 | 60%   | 712.1  |
|           | 2   | 1266.29| 100%  | 1266.3 |
|           | 3   | 381.13 | 64.9% | 611.1  |
| t200      | 1   | 5579.37| 100%  | 5575.11 |
|           | 2   | 436.92 | 100%  | 436.92 |
|           | 3   | 1438.24| 70%   | 1501.1 |
| t300      | 1   | 1120.05° | 100%  | 1110.3 |
|           | 2   | 1579.49°| 100%  | 1575.6 |
|           | 3   | 2815.2 | 100%  | 2815.2 |
| AN_Part1  | 1   | 555.87° | 100%  | 552.1  |
| AN_Part2  | 1   | 3600.0 | 100%  | 1214.25 |

Cmp. | - | 72.8% | 1 | 1 | 1 | - | 1 | - | - | - | - | - | - | - |

The reconfiguration time overhead in the dependency dominated TGs (the designs having long CPT) under the resource constraints.

In benchmark t50-2, the obtained by the proposed method is obviously higher than the baseline situation because t50-2 has high parallelism, whereas FPGA hardware resources constrain the parallel executions of the tasks. Table IV shows the detailed experimental results on the relationships between FPGA resources and performances for applications with different degrees of parallelism.

According to our experimental results, the CC can be reduced by 33.18% on average by considering the CCs in the optimization framework. The detailed data are listed in the supplementary material (Table I).

To evaluate the impacts of FPGA resources on the schedule length of designs, we perform the experiments for all test benchmarks under different FPGA resource constraints, which are set as 3/4x, 1.0x, 3/2x, and 2.0x of the targeted FPGA architecture (37 950 CLBs). We execute Int_PSF10 times independently for each benchmark, and show the average results in Table IV. The column resource represents the amount of resources for the target FPGA architecture. As shown in

Table IV, for all the benchmark circuits, with increasing FPGA resources, the trends of schedule length T and CC decrease to be gentle. On the one hand, DRRs can be executed in parallel and configured independently, thus the configuration latency can be effectively hidden in the executions of tasks. With increased FPGA resources, the number of DRRs N is increasing overall, which will maximize the parallel execution of tasks and increase the possibility of hiding the configuration of tasks. On the other hand, the schedule length should be greater than that in the baseline situation shown in Table II. For the benchmark t200-1, which involves a long CPT and fewer data dependencies, the schedule length T remains the same with increasing FPGA resources and is close to the length of the corresponding CPT because the configuration latency is effectively hidden. Furthermore, Int_PSF achieves 100% success rate for the different FPGA resource constraints, which demonstrates the effectiveness of the method.

As discussed in Section IV-C, to reduce the complexity, we use the area sum of the task modules to underestimate the configuration span of the time layers instead of accurately computing the configuration span of the DRRs, which should be computed using the DRR area. According to our experimental results, for the same solution, if the DRR area is used instead of summing the module area, the schedule length is increased by 5%. In the optimization framework, if we use the DRR area to accurately estimate the configuration span of time layers, the obtained schedule length is increased only by negligible 1%. The possible reason is that the configurations of the time layers are well hidden in the execution of task modules. A detailed analysis is included in the supplementary material (Fig. 1).

C. Vertically Aligning DRRs to Reconfigurable Frames

As was demonstrated in [13], when applying the Reset After Reconfiguration methodology, a DRR must vertically align to reconfigurable frames (aligning vertically to clock regions) for 7-series and Zynq-7000 AP SoC devices. When the height of the DRRs is vertically aligned to the reconfigurable frames, the height of the reconfiguration frame (50 rows of CLBs) is adopted as the measurement unit of DRR height. Table V shows the experimental results of with/without
TABLE IV
IMPACTS OF FPGA RESOURCES ON Int_PSF

| imp. | Resource | #succ | N | T (ms) | Int_PSF |
|------|----------|-------|---|--------|---------|
| 1    | 3/4      | 100%  | 5.6 | 887.45 | 707821 |
| 1    | 100%     |       | 9.3 | 715.22 | 644379 |
| 2    | 1/10     | 14.1  | 120.36 | 644379 |
| 1    | 100%     |       | 9.4 | 1266.29 | 644379 |
| 2    | 1/10     | 14.1  | 120.36 | 644379 |

TABLE V
ALIGNING DRRS TO RECONFIGURABLE FRAMES

| Bench | Int_PSF with aligning | Int_PSF |
|-------|-----------------------|---------|
|       | #succ     | T (ms) | N | CC | #succ | T (ms) | N | CC |
| t10   | 100%    | 273.6  | 3.0 | 65125 | 100% | 269.8 | 3.9 | 54078 |
|       | 200%    | 370.8  | 2.5 | 82022 | 100% | 369.6 | 2.5 | 79027 |
|       | 300%    | 232.5  | 3.0 | 38566 | 100% | 201.4 | 3.7 | 57750 |
| t30   | 100%    | 147.09 | 2.7 | 7516390 | 100% | 1459.8 | 3.0 | 6806495 |
|       | 200%    | 799.8  | 2.8 | 582164 | 100% | 793.1 | 3.0 | 599203 |
|       | 300%    | 738.7  | 3.0 | 602132 | 100% | 734.2 | 3.0 | 615524 |
| t50   | 100%    | 473.5  | 4.0 | 257776 | 100% | 456.9 | 3.3 | 278647 |
|       | 200%    | 691.2  | 4.6 | 3826900 | 100% | 705.7 | 3.9 | 364433 |
|       | 300%    | 611.9  | 4.7 | 625038 | 100% | 618.9 | 3.8 | 589759 |

TABLE VI
COMPARISON BETWEEN THE ILP+Flooprlan [9] AND THE PROPOSED Int_PSF

| Bench | ILP+Flooprlan [9] | Int_PSF |
|-------|-------------------|---------|
|       | #succ | T (ms) | N | Run (s) | #succ | T (ms) | N | Run (s) |
| t10   | 100% | 269.8 | 4.0 | 0.84 | 100% | 269.8 | 4.0 | 1.36 |
|       | 90%  | 369.7 | 4.0 | 0.54 | 100% | 369.7 | 3.0 | 1.64 |
|       | 300% | 176.2 | 4.0 | 2.24 | 100% | 202.6 | 4.0 | 1.47 |
| t30   | 100% | 459.8 | 6.0 | 7.1  | 100% | 1460.1 | 4.6 | 6.82 |
|       | 90%  | 792.8 | 6.0 | 7.9  | 100% | 794.1 | 5.1 | 6.55 |
|       | 300% | 734.2 | 5.0 | 17.8 | 100% | 739.1 | 4.7 | 6.47 |
| t50   | 100% | NF    | NF  | >1000 | 100% | 487.6 | 6.7 | 16.08 |
|       | 90%  | NF    | NF  | >1000 | 100% | 733.3 | 6.6 | 17.17 |
|       | 300% | 605.8 | 8.0 | 1085 | 100% | 623.4 | 6.8 | 17.61 |

D. Comparison With Previous Work

Cordone et al. [6] proposed a partitioning method to extract cores (isomorphic and nonoverlapping subgraphs) from the TGs for module reuse and an ILP-based method and a heuristic method for scheduling TGs on PDR-FPGAs. The core extraction method provides preprocessing of the TGs and can be combined with other scheduling methods to consider module reuse. However, it is difficult to extend the scheduling method for processing DRR partitions, while the task partitioning algorithm in Jiang and Wang [8] can be used only within a DRR. Deiana et al. [9] proposed an MILP-based scheduler for mapping and scheduling applications on partially reconfigurable FPGAs with consideration of DRRs, where only one task module is involved in each time layer, followed by floorplanning the DRRs. Consequently, in this paper, we make a comparison with the methodology in [9]. We adapt the ILP method in [9] to the problem in this paper by skipping the module reuse, and in the proposed optimization framework, we add a constraint (one task constraint) so that each time layer includes only one task module. As for the floorplanning of DRRs, there are no detailed descriptions in [9], so we use the method in [23], which performs very well in the fixed-outline-constrained floorplanning for FPGAs and spends only several seconds for the floorplanning of 100 task modules.

Table VI shows the experimental results. The results are the average of ten independent runs. In the ILP+Flooprlan method, we solve the ILP model once and run the floorplanning algorithm ten times. Because the ILP-based method is time-consuming, we use the small test cases, t10, t30, and t50 (the largest test cases in [9] includes 50 tasks), for the comparison. “NF” represents that the ILP solver fails to find any feasible solution in a reasonable time. The results show that the proposed optimization framework achieves much higher success rates with comparable schedule lengths.

VI. CONCLUSION

In this paper, we proposed an integrated optimization framework for partitioning, scheduling, and floorplanning PDR-FPGAs, where the P-ST (PS, QS, RS) was proposed to represent the partitions, schedule, and floorplan of the task modules, and a sufficient and necessary condition is given for the feasibility of P-ST considering the scheduling problem. An elaborated method was proposed to generate new solutions by simultaneously perturbing the partition, schedule, and floorplan. Based on the proposed optimization framework, we integrated the exploration of spatial and temporal design space to search the optimal solutions of partitioning, scheduling, and floorplanning. Experimental results demonstrated the effectiveness of the proposed framework. In future work, we will further consider the reuse of task modules, variable dimensions for task modules, and integration of the allocation of RAM and DSP resources.

VII. ACKNOWLEDGMENT

The authors would like to thank the Information Science Laboratory Center of USTC for the hardware and software services.
