# A Multi-stage Leakage Aware Resource Management Technique for Reconfigurable Architectures

Nam Khanh Pham
Department of Electrical and Computer Engineering
National University of Singapore
a0095804@nus.edu.sg

Amit Kumar Singh
Department of Electrical and Computer Engineering
National University of Singapore
amit.singh@nus.edu.sg

Akash Kumar
Department of Electrical and Computer Engineering
National University of Singapore
akash@nus.edu.sg

## ABSTRACT

Shrinking size of transistors has enabled us to integrate more and more logic elements into FPGA chips leading to higher computing power. However, it also brings serious concern to the leakage power dissipation of the FPGA devices. One of the major reasons for leakage power dissipation in FPGA is the utilization of prefetching technique to minimize the reconfiguration overhead (delay) in Partially Reconfigurable (PR) FPGAs. This technique creates delays between the reconfiguration and execution parts of a task, which may lead up to 44% leakage power of FPGA since the SRAM-cells containing reconfiguration information cannot be powered down. In this work, a resource management approach containing *scheduling*, *placement* and *post-placement* stages has been proposed to address the aforementioned issue. In scheduling stage, a leakage-aware cost function is derived to cope with the leakage power. The placement stage uses a cost function that allows designers to decide a trade-off between performance and leakage-saving. The post-placement stage employs a heuristic approach and shows further improvements. Experiments show that our approach can achieve large leakage savings for both synthetic and real life applications with acceptable extended deadline. Furthermore, different variants of the proposed approach can reduce leakage power by 40-65% when compared to a performance-driven approach and by 15-43% when compared to state-of-the-art works.

## Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*

## General Terms

Design

## Keywords

Resource management; FPGA; leakage aware

## 1. INTRODUCTION

Field-programmable gate arrays (FPGAs) are promising candidates for digital circuit implementation because of their growing density and speed, short design cycle, and steadily decreasing cost. Furthermore, most of the FPGA devices nowadays can be partially reconfigured at run time, i.e., a configuration can be loaded into part of the device while the rest of the system continues operating. This feature obviously provides greater flexibility and more powerful computing ability. However, these advantages come with additional problems related to reconfiguration time and power dissipation.

A drawback of FPGA due to its hardware redundancy is its inefficiency in term of power consumption when compared to ASIC components [12] [16]. In practice, an FPGA circuit implementation may use only a fraction of the hardware resource but the power is dissipated in both the used and the unused components. The total power consumption includes static (leakage) and dynamic power [17], and their contribution into the total power consumption heavily depends on the circuit technology. Beyond 65 nm technology, leakage power becomes an increasingly dominant component of total power dissipation [24]. This has motivated us to focus our work on reducing the leakage power dissipation.

Configuration prefetching [9] is a widely adopted technique for reducing the reconfiguration delay in Partially Reconfigurable (PR) FPGA. In prefetching, a task is loaded into the FPGA as soon as possible and this may result in overlap between the configuration part of the waiting task (to be executed) with the execution part of operating tasks, facilitating for reduced reconfiguration overhead (time). However, even after the task is loaded (prefetched), it may not execute and wait until few other tasks complete due to involved dependencies. Such waiting introduces delays between the configuration and execution part of the same task. During the delay interval, the SRAM-cells of the FPGA (containing bits of the waiting task to be executed) cannot be powered down to avoid the loss of configuration data from the cells. Therefore, the cells dissipate a significant amount of power.

**Motivational Example:** Fig. 1 presents an example to demonstrate aforementioned issues. In this example, the task graph on the left-hand side is scheduled on an FPGA platform with prefetching technique. During the interval between R3 and E3, the logic blocks of columns 1 and 2 can be powered down to remove leakage wastes. However, since the SRAM-cells of these columns cannot be powered down as the configuration data will be lost, they consume a considerable amount of power. As SRAM cells leakage contributes $\approx 38\%$ to FPGA leakage [22] (up to $44\%$ for Spartan-3 family [21]), reducing FPGA SRAM leakage is of paramount importance.

*In order to reduce leakage, a scheduling approach needs to be developed aiming at allocating reconfiguration and execution parts as close as possible while keeping task dependencies, timing and architecture constraints into account.* Several works have been proposed to solve this problem [25], [10]. However, these works attempt to address the leakage problem in a single phase of the resource management process (details in later sections). As a result, the leakage power cannot be significantly reduced. It has also been


*GLSVLSI'14,* May 21–23, 2014, Houston, Texas, USA.

http://dx.doi.org/10.1145/2591513.2591526.

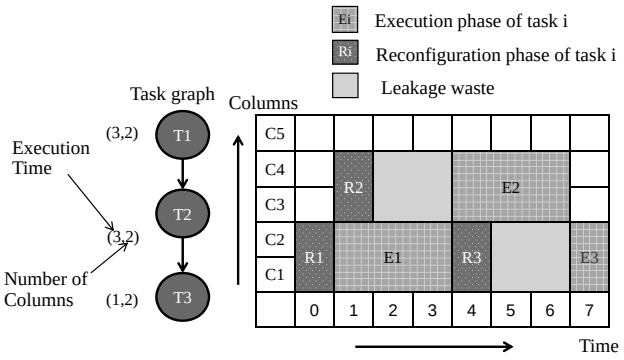


**Figure 1: Example of Leakage Waste caused by Prefetching Technique**

observed that there exists a trade-off between leakage savings and performance [25]. However, the trade-off analysis by employing the existing approaches is not efficient. A high degradation in performance is noticed in order to achieve small amount of leakage savings. To tackle the problem in a comprehensive perspective towards achieving high leakage reductions, we propose a multi-stage resource management approach consisting of three stages. Our main contributions to each stage are as follows:

- **Scheduling:** A list-scheduling algorithm has been developed with a specific priority function that is customized for addressing the leakage power reduction.
- **Placement:** A cost function has been derived for the placement stage to further reduce the leakage power. This function provides designers a flexibility to manage the trade-off between performance and leakage savings.
- **Post-placement:** A post-placement heuristic has been proposed to improve the scheduling results (leakage savings) from previous stages.

In our multi-stage approach, the application to be executed is processed iteratively in the scheduling and placement stage, where in each iteration the scheduling stage defines next application task to be placed in the placement stage by using a dynamic priority scheme (details in a later section). The placement stage identifies the FPGA column to map the task. Once all the application tasks are mapped, the post-placement stage tries to reduce gaps between configuration and execution parts for all the application tasks in order to improve results obtained from earlier stages. *To the best of our knowledge, this is the first work that considers leakage optimization in multiple stages while considering application deadline.*

**Paper Organization:** Section 2 presents state-of-the-art related to leakage power reduction. Section 3 provides the targeted FPGA architecture, application model and problem definition. The main contributions are presented in Section 4. In Section 5, experimental results are reported and Section 6 provides the conclusion.

## 2. RELATED WORK

There are various techniques reported in literature to reduce the leakage power of FPGAs. At *architecture level*, Calhoun *et al.* [4] introduce a fine-grained leakage control scheme using sleep transistors [15]. Fei Li *et al.* [13] proposed the programmable supply voltage ($V_{dd}$) in FPGAs. Elements on critical path are provided high supply voltage to ensure high performance, while components on noncritical path are supplied with low voltage and unused part of the device is switched-off. The leakage power of FPGAs significantly depends upon the threshold voltage ($V_T$) and an approach using high $V_T$ transistors is proposed in [6]. A profound survey of leakage reduction techniques for SRAMs has been provided in [5].

At *system level*, works focusing on leakage power problems are fewer than those of the architecture level [14]. Bharadwaj *et al.* [3] propose a design methodology that groups temporal locality design into cluster. The authors also develop a power state controller for effectively switching the states of the cluster. Addressing the leakage problem from system level as well, Zapater *et al.* has proposed an empirical model for leakage components and used it to design a energy-efficient control mechanism for servers in data centers [26].

Task graph scheduling for FPGA is an extensively studied topic [1, 2, 18, 20]. In [18], an efficient technique to schedule real-life applications on FPGA is proposed, but partial reconfiguration and resource constraint has not been considered. Most of the scheduling methods for FPGA focus on specific problems related to reconfiguration overhead and defragmentation. Ahmadinia *et al.* [1] combined scheduling and placement method for 2D FPGA architecture using cluster-based method to improve the performance by 20% and task rejection by 16.2%. Christoph *at el.* [20] integrated an online placement into a scheduling algorithm using small tasks first and earliest deadline first techniques. However, they do not take into account prefetching technique and resource constraint due to single reconfiguration controller pertaining to PR FPGA. The first work that considered both prefetching technique and resource constraint was introduced by Banerjee *at el.* [2]. The scheduling and placement models are included with the partitioning stage to form a complete HW-SW co-design approach for PR systems. The linear placement model in this work is later adopted by Yuh *et al.* [25] and Hsieh *et al.* [10] to address the leakage power issues.

Yuh *et al.* [25] first introduced the idea of using scheduling approach to mitigate the leakage issue. The authors utilized the scheduling and placement results from [2] and on top of that they developed a post-placement heuristic to reduce the delays between execution and reconfiguration parts. They also proposed an exact ILP solution to perform the post-placement in order to verify the effectiveness of the heuristic. Since their work tackles the leakage optimization after the tasks are already allocated onto the FPGA, the existing placement results may not allow their approach to significantly eliminate the leakage power. To achieve maximal leakage saving, our work addresses the leakage problem in all phases of the resource management process: *scheduling stage*, *placement stage* and *post-placement stage*.

With the same model and target, Hsieh *et al.* [10] introduced another approach to reduce the leakage waste. Their method consists of 3 phases: *binding*, *priority dispatching* and *split-aware placing*. First, the reconfiguration and execution parts of all tasks are combined together in the binding phase so that the leakage power is minimal. Then, each task is assigned a priority value based on the position of the task in the task graph. Finally, while placing the tasks into FPGA architecture, the split-aware placer checks for the deadline. If the deadline is violated, the placer splits the reconfiguration and the execution phase of the task. While the work in [10] tried to solve the leakage problem in the placement phase only, we propose a more complete solution having multiple stages. Furthermore, the scheduling algorithms in [10] used static priority, which is computed before the actual scheduling process takes place. The static priority is computed based on the characteristic of the task graph and remains unchanged during the scheduling process. In contrast, our algorithm dynamically recalculates the priorities of all available tasks every time a task is allocated onto the FPGA. Therefore, our algorithm updates the current available resource of the FPGA, leading to a better scheduling decision.

Table 1 summarizes the distinction of our work in comparison to the closely related works reported in the literature. As can be seen, existing works perform leakage aware optimization in scheduling, placement, or post-placement stages, whereas our approach performs optimization in all the stages. Further, unlike most of the approaches that consider static priorities of tasks, our approach considers dynamic priorities.

**Table 1: Comparison of various approaches**

| Features | Ref. [2] | Ref. [25] | Ref. [10] | Our work |
|---|---|---|---|---|
| Scheduling | Performance driven | No | Performance driven | Leakage aware |
| Placement | Performance driven | No | Leakage aware | Leakage aware |
| Post placement | No | Leakage aware | No | Leakage aware |
| Priority of tasks | Dynamic | No | Static | Dynamic |

# 3. SYSTEM MODEL AND PROBLEM DEFINITION

The **targeted architecture** used in this work is 1 dimensional (1D) FPGA, where the configurable logic blocks (CLBs) are arranged in fixed vertical columns, and a task occupies an integral number of columns. Moreover, the device supports dynamic partial reconfiguration: a part of the platform can be configured while other parts operate without interruption. The basic configuration unit is a column. A task can be deployed on an adjacent set of columns, and the reconfiguration time of the task is proportional to the number of columns. Such an architecture is similar to Xilinx FPGA Virtex family [23]. The device can be configured by a bitstream through configuration ports like JTAG or ICAP. However, both configuration ports are managed by only one configuration controller. Therefore, two different tasks cannot be reconfigured at the same time. Such architectural constraint plays a critical role in the process of scheduling and placement. Another key element realizing the benefits of scheduling algorithm on FPGA are sleep transistors. It is assumed that unused CLBs can be totally powered off by the sleep transistors integrated in the device. Based on this assumption, each column can be independently controlled by a sleep transistor [25].

**Task model:** We consider only hardware tasks, i.e., a task can be synthesized and implemented on the FPGA platform. In comparison to software tasks, hardware tasks have some additional parameters related to the required hardware area and configuration time. Directed acyclic graph (DAG) is used to represent the task set of an application. An example of the task graph model is presented in Fig. 1. In the DAG, each node $u$ represents a task, while an edge $e(u; v)$ indicates the dependency between tasks $u$ and $v$.

A task has two components: reconfiguration and execution. Reconfiguration part is scheduled under the architectural constraint (only one reconfiguration controller) while scheduling of execution part depends on the data dependencies, where a linear task placement model as that of [2] has been adopted. In the scheduling process, the communication overhead between tasks is ignored due to two reasons: *1)* tasks communicate with each other through a shared memory with the same latency and cost; and *2)* this latency is negligible in comparison to runtime reconfiguration overhead (time) and execution time.

**Scheduling Problem**

The problem targeted in this paper considers following set of input, constraints and objective.

- **Input:** The application task graph and FPGA architecture (number of columns, 1 reconfiguration controller and 1D architecture).
- **Constraints:** Task graph dependency for execution parts, reconfiguration controller constraint for reconfiguration part and sequential relation between the reconfiguration and execution parts of the same task.
- **Objective:** Minimize leakage power dissipation because of the delays between the reconfiguration and execution parts, minimize schedule length.

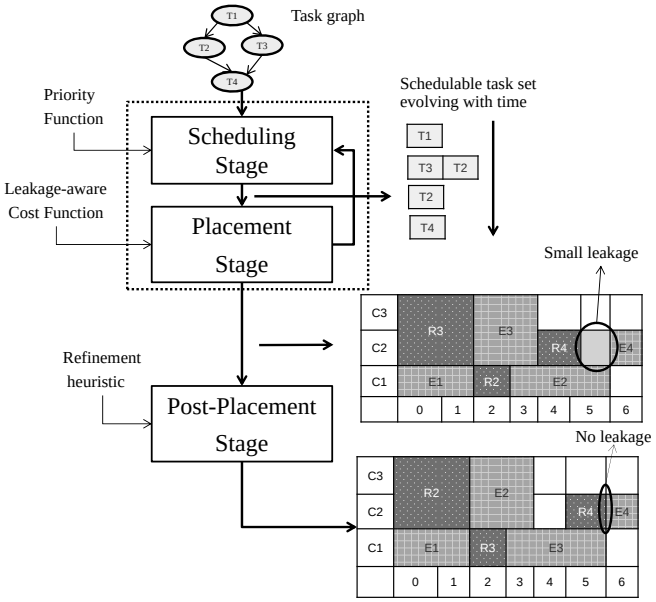


**Figure 2: Multi-stage Scheduling Scheme**

# 4. PROPOSED MULTI-STAGE RESOURCE MANAGEMENT APPROACH

An overview of the proposed resource management approach is provided in Fig.2. The approach has 3 stages: *Scheduling, Placement* and *Post-placement*. At first, the application task graph is processed iteratively in the first two stages (Scheduling and Placement). In each iteration, the Scheduler will define the next task coming to the Placer by a dynamic priority scheme, which means that the priorities of all the schedulable tasks are changed after each iteration. The Placer then decides the column where the task should be mapped and update the current status of the platform for the Scheduler. After all the tasks in task graph are allocated into the platform, the refinement heuristic in Post-Placement Stage will further improve the result from previous stages.

## 4.1 Scheduling Stage

**Algorithm 1** presents our algorithm for the scheduling phase. At each step, all schedulable tasks whose parents have been scheduled are stored in a set of ready task $-$ $S$. Then, the scheduler calculates the dynamic priorities of all tasks in set $S$ according to a priority function defined by Equation 1. Thereafter, it chooses the task with highest priority to pass to the placer. As mentioned in Section 2, we use a dynamic priority function so that the scheduling process can adapt with the current status of the FPGA. Since the priority function has a strong impact on the schedule quality, it is carefully designed to address both leakage saving and performance requirement. The function includes different components that reflect the affection of constraints (FPGA architecture and task graph dependency) as well as optimization targets (leakage saving and schedule length) on scheduling decision. Our priority function is described as follows:

$$F = \alpha BT + \sigma C - \beta EET - \gamma ERT - \mu LK \quad (1)$$

$$LK = C * (EET - (RT + ERT)) \quad (2)$$

where,

$BT$ : bottom level of the task that represents the length of the longest path in task graph starting from this task;
$EET$: earliest execution time of the task;
$ERT$: earliest reconfiguration time of the task;
$C$ : number of columns required by the task;
$RT$ : the reconfiguration time of the task;

**Algorithm 1** Leakage Aware Task Scheduling Algorithm

**Input:** Task graph G=(U,V)
**Output:** Schedule with minimal $LK$
1: Put source tasks $\{t_i \in U : \textbf{pred}(t_i) = \emptyset\}$ into set **S**
// **S** − Set of schedulable tasks
2: **while S** $\neq \emptyset$ **do**
3: Calculate priorities of unscheduled tasks in **S** (by Equation 1)
4: Choose the task $t$ with maximum priority
5: Choose the best column $C$ for task $t$ (by Algorithm 2)
6: Schedule task $t$ starting from column $C$
7: **if** child tasks of $t$ are not already added to **S then**
8: Add new available tasks to **S**
9: **end if**
10: Remove task $t$ from **S**
11: **end while**

$LK$ : leakage waste caused by scheduling the task. The leakage waste is the product of the used columns and the delay between reconfiguration and execution parts.

$EET, ERT$ and $LK$ are dynamic factors and are computed in scheduling process based on the current status of the partial schedule. Since these variables are fundamentals for scheduling problem, the details of their calculation can be found in basic textbook about task scheduling, such as [19]. $\alpha, \beta, \gamma, \sigma, \mu$ are coefficients related to each factor and used to determine the intensity of their impact on the cost function. The signs of elements in the function are given based on their impact on the schedule: tasks requiring larger columns should be placed earlier to increase the space for other tasks; tasks with higher bottom level (close to leaf tasks) should be scheduled first because they strongly affect the schedule length. Additionally, tasks with minimal $EET$, $ERT$ and $LK$ should be chosen for the desired optimization objective. As shown in Fig.2 the output of the scheduling stage is a set of schedulable tasks with the task of the highest priority in the front of the set. This highest priority task is then transferred to Placement Stage to be allocated onto the FPGA. Since we are using a dynamic priority scheme, both the schedulable task set and the priorities of tasks in the set are changed every time a task is placed in FPGA.

## 4.2 Placement stage

After getting the task with highest priority, the placer applies the steps in **Algorithm 2** to allocate the task into physical column(s) of FPGA. When a task comes to this stage, the algorithm scans all the columns to find available positions for the task and for each available position, the cost function is computed. Then, the task is placed into the position with minimal cost value. Here, also the cost function is also designed to optimize for both performance and leakage waste, which is presented as follows:

$$G = \frac{a}{10} * LK + (1 - \frac{a}{10}) * EST \quad (3)$$

where, $LK$ and $EST$ represent leakage power and earliest start time for a placement; $a$ is the leakage-schedule length trade-off coefficients, which can be used to provide a balance between the two optimization goals. Therefore, the cost function not only facilitates to reduce the leakage dissipation but also provides designer the ability to manage the trade-off between performance (schedule length) and leakage saving. The trade-off values can be achieved by adjusting the value of $a$ in Equation 3. By increasing the value of $a$, designer can save more leakage power with a longer schedule length.

Fig. 2 demonstrates the placement results from the first 2 stages of our approach. It is expected to have small leakage power as a result of above optimization techniques as shown in the figures.

**Algorithm 2** Leakage Aware Placement Algorithm

**Input:** Task $t$, set of columns **P**
**Output:** column $C$- with minimal $LK$
1: **for** each column $c_i \in$ **P do**
2: Schedule task $t$ starting from column $c_i$
3: Calculate cost of placing $t$ on $c_i$ (by Equation 3)
4: **end for**
5: Choose the column $C$ with minimal cost function

**Algorithm 3** Leakage Aware Post-placement Algorithm

**Input:** Task graph G=(U,V), Tasks' placement after placement stage
**Output:** Optimized placement of tasks
1: **for** each leaf task $t_i \in$ **U do**
2: Schedule configuration and execution of task $t_i$ by considering architectural constraint
3: **while** $parents\ of\ t_i \neq \emptyset$ **do**
4: Find reconfiguration costs for parent tasks of $t_i$ by Equation 4
5: Sort reconfigurations in descending order based on cost
6: Schedule reconfigurations considering architectural constraints
7: Select parents one by one from maximum to minimum cost as $t_i$
8: **end while**
9: Move executions close to reconfigurations if dependencies do not violate
10: **end for**

## 4.3 Post-placement Heuristic

Our post-placement heuristic is presented in Algorithm 3. The heuristic takes task graph & tasks' placement as input and provides optimized placement of tasks so that leakage power due to delays between reconfigurations and executions is further minimized. The heuristic first schedules leaf tasks to maintain the same finish time towards meeting the timing deadline. For each leaf task, it's parent tasks are evaluated for their reconfiguration costs and scheduled by taking architectural constraints into account. The cost is computed as follows

$$C = lw * NC - sw * SP \quad (4)$$

where, $NC$ and $SP$ are the number of occupied columns and range of reconfiguration space, respectively. The $lw$ and $sw$ are the weights to be given to $NC$ and $SP$ respectively, which determine the leakage power dissipation.

After all the tasks are scheduled, the executions are tried to place close to the respective reconfigurations if dependencies are not violated. This helps us to achieve placement that contains reconfigurations and executions close to each other as shown in Fig. 2, leading to reduced leakage power.

# 5. EXPERIMENTAL RESULTS

A series of experiments are conducted to demonstrate the performance of our resource management approach. Three versions of our scheduling and placement approach with different value of constant $a$ in Equation 3 ($a$=1, $a$=2, $a$=10) are compared with following existing approaches: performance-driven algorithm (PDA) proposed in [2], Enhanced Leakage Aware Algorithm (ELAA) employed in [10], the ILP and Iterative Refinement (ITE) heuristic approach proposed in [25]. The PDA does not consider the leakage waste in the scheduling process, and has been used as the baseline approach for comparisons. ELAA demonstrates high performance when dealing with the leakage problem [10]. One important target in this work is to examine the trade-off between leakage saving and the schedule length, so no deadline (in terms of schedule length) is set for the trade-off analysis. The results from our post-placement approach are compared to that of [25].

Our algorithm is implemented in Java language and experiments are performed on an Intel Core i7 2.26GHz CPU with 4 GB RAM.

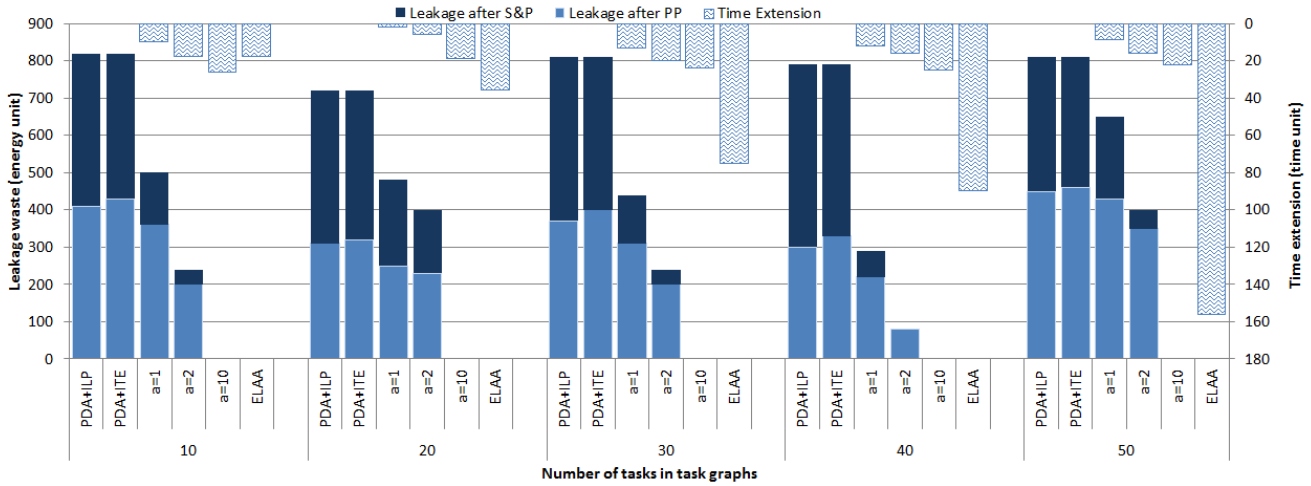


**Figure 3: Leakage and Schedule Length when employing Different Approaches**

The experiments are performed with real-life task graphs and synthetic task sets generated by the TGFF tool [8]. For the synthetic case, five task sets are considered. Each task set contains 10 task graphs with different level of parallelism; and each task in the task graph requires 10 to 50 columns and has the execution time from 1 to 9 time units. The FPGA platform is considered to have a fixed number of columns as 100. For real-life task graphs, JPEG encoder [2], MP3 decoder [11] and MPEG4 decoder [7] are considered with their specifications provided in respective references in order to demonstrate the applicability of our approach for real-life scenarios.

The criteria of the comparison are schedule length, leakage waste, and the runtime of the algorithms. The schedule length is measured in time unit, while the leakage waste is measured in energy unit, which is the power dissipation of one column during 1 time unit. The leakage waste of a particular task is computed by Eqn. 2. The leakage waste of the task graph after scheduling is the sum of leakage waste of all its tasks. For leakage waste of a task set, leakage values of all the contained task graphs are added. Further, as sleep transistors are used to stitch-off the unused SRAM cells for each column, the leakage waste for a task before its configuration and after the execution is considered as zero.

## 5.1 Leakage Waste and Schedule Length

Fig. 3 presents the leakage waste and schedule length (in terms of time extension over baseline approach PDA) of all the approaches over the five task sets. The whole bars present the leakage waste obtained after Scheduling and Placement (S&P) stage, while the lower parts of the bars describe the leakage waste after applying Post-Placement (PP) methods. Therefore, for existing approaches, the whole bars describe the leakage waste of PDA methods, and the lower part of each bar is the leakage after post-placement refinement (PDA+ITE or PDA+ILP). The time extension is the extended deadline required for leakage reduction. It is computed by subtracting the schedule length of each approach to the schedule length of the baseline (PDA) and these values are presented by columns with reversed direction (up to down). The horizontal axis declares notations for different approaches. For example, the first two notations PDA+ILP and PDA+ITE denote two approaches used in [25], where PDA is used in Scheduling and Placement (S&P) phase and either ILP or ITE is used in Post Placement phase.

It can be seen from Fig. 3 that all versions of our approach achieve better leakage saving when compared with the two approaches in [25]. Furthermore, when the number of tasks is large (greater than 10), our approach with $a = 10$ can reach the optimal leakage saving (leakage waste = 0) with smaller extension in time when compared to ELAA. On an average, our approach adopted with the parameter $a = 1$ and $a = 2$ shows leakage power savings of 40% and 65% respectively when compared to PDA. Furthermore, when compared with existing approach PDA+ITE, our approach achieves 15% and 43% more leakage savings with parameter $a = 1$ and $a = 2$, respectively. The reason behind superior results by our approach over other approaches is that we consider leakage optimization first in scheduling and placement stages and then in post-placement stage as well. The optimization in scheduling and placement stages results in minimize delays between configurations and executions, and the post-placement stage try to further minimize the left delays in order to reduce the leakage dissipation. However, other approaches tackle the leakage optimization in only one stage (e.g., in placement stage in ELAA [10] and in post-placement stage in [25]).

## 5.2 Post-placement Leakage Waste and Algorithm Runtime

In this experiment, we examine the leakage saving and runtime of 3 post-placement methods ILP, ITE in [25], and our proposed heuristic. The methods are executed with the same inputs, which are the placement results from PDA. The deadline of all the task graphs are set to the schedule length of our approach when achieving optimal value of leakage saving (i.e., $a = 10$).

Table 2 shows leakage waste and algorithm runtime for various post-placement methods. As can be seen from Table 2, in many cases, all the post-placement methods are unable to totally eliminate the leakage dissipation over the PDA placement. However, for the same deadline, our multi-stage approach can achieve the optimal solution (leakage waste = 0) as described earlier. This signifies the advantages of our comprehensive strategy that addresses the leakage problem throughout the resource management process. Although our scheduling and placement stages achieve high leakage savings, they still can leave spaces between reconfiguration and execution parts of many tasks. Our post-placement stage tries to reallocate reconfigurations and executions so that the spaces between them are minimized in order to achieve further leakage savings. Table 2 shows that our post-placement heuristic can produce better leakage results than ITE. Additionally, our heuristic obtains the results in a smaller runtime.

## 5.3 Case-study: Real-life Applications

We applied different scheduling approaches on real-life applications: JPEG encoder [2], MP3 decoder [11] and MPEG4 decoder [7] as mentioned earlier. Table 3 shows leakage waste and schedule length for real-life applications. The notations used in

**Table 2: Leakage waste and algorithm runtime of post-placement methods**

| Algorithms | Number of tasks in task graphs | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | | 20 | | 30 | | 40 | | 50 | |
| | Leakage | Runtime (s) | Leakage | Runtime (s) | Leakage | Runtime (s) | Leakage | Runtime (s) | Leakage | Runtime (s) |
| PDA+ILP | 0 | 2.278 | 40 | 12.451 | 60 | 25.812 | 0 | 50.24 | 60 | 199.24 |
| PDA+ITE | 20 | 2.46E-04 | 80 | 4.32E-04 | 180 | 8.17E-04 | 80 | 1.14E-03 | 80 | 3.69E-03 |
| PDA + Our heuristic | 20 | 2.15E-04 | 80 | 4.36E-04 | 100 | 3.66E-04 | 80 | 4.32E-04 | 80 | 5.02E-04 |

this experiment are the same as those in previous experiments. The ELAA and our approach with $a = 10$ always achieve the optimal value of leakage waste (zero) with some extension in schedule length. Therefore, leakage in these cases does not need any improvement by Post-placement methods and not applicable (NA) has been mentioned for the same. As can be seen from the table, for MPEG and JPEG, our approach with $a = 1$ can obtain the same results as that of approach *PDA+ITE*. However, when it comes to MP3 decoder, the advantage of our comprehensive strategy becomes obvious. Due to low quality solution in the first two phases, the ITE approach cannot remove all the leakage from initial placement of previous phases. In contrast, all stages of our approach still work well to get maximum leakage saving.

**Table 3: Leakage waste and schedule length for real-life applications**

| | | PDA+ITE | a=1 | a=10 | ELAA |
|---|---|---|---|---|---|
| MPEG | Schedule length | 44 | 44 | 53 | 57 |
| | Leakage S&P | 140 | 80 | 0 | 0 |
| | Leakage PP | 0 | 0 | NA | NA |
| JPEG | Schedule length | 22 | 23 | 24 | 29 |
| | Leakage S&P | 60 | 20 | 0 | 0 |
| | Leakage PP | 20 | 20 | NA | NA |
| MP3 decoder | Schedule length | 50 | 57 | 61 | 63 |
| | Leakage S&P | 270 | 30 | 0 | 0 |
| | Leakage PP | 270 | 30 | NA | NA |

# 6. CONCLUSION

We present a multi-stage resource management approach to tackle the leakage power problem in Partially Reconfigurable FPGAs. Our multi-stage approach employs leakage-aware priority function in scheduling stage, leakage-performance trade-off function in placement stage and a heuristic in post-placement stage. A series of experiments are performed to highlight the advantages of the proposed approach over existing works. The results demonstrate that the proposed approach dominates the existing approaches when the application task graph contains higher number of tasks. Additionally, experiments show that our approach can always achieve the optimal value as a comprehensive strategy is adopted, whereas other single-stage methods may not achieve the optimal value. Furthermore, our approach also provides the flexibility to the designers to achieve trade-off values between leakage saving and performance. In the future, we plan to examine the dependencies between task graph parameters and coefficient $a$ to enhance the schedule quality. Extending the problem for two-dimensional FPGAs is also a promising direction.

# 7. ACKNOWLEDGMENT

This work is supported by Singapore Ministry of Education Academic Research Fund Tier 1, grant number R-263-000-B02-112.

# 8. REFERENCES

[1] A. Ahmadinia, C. Bobda, and J. Teich. A dynamic scheduling and placement algorithm for reconfigurable hardware. *Organic and Pervasive Computing–ARCS 2004*, pages 443–465, 2004.

[2] S. Banerjee, E. Bozorgzadeh, and N. Dutt. Physically-aware hw-sw partitioning for reconfigurable architectures with partial dynamic reconfiguration. In *DAC*, pages 335–340, 2005.

[3] R. Bharadwaj et al. Exploiting temporal idleness to reduce leakage power in programmable architectures. In *ASP-DAC*, pages 651–656, 2005.

[4] B. Calhoun, F. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in mtcmos. In *ISLPED*, pages 104–109, 2003.

[5] A. Calimera et al. Design techniques and architectures for low-leakage srams. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, pages 1992–2007, 2012.

[6] L. Ciccarelli, A. Lodi, and R. Canegallo. Low leakage circuit design for fpgas. In *CICC*, pages 715–718, 2004.

[7] J. Cong and K. Gururaj. Energy efficient multiprocessor task scheduling under input-dependent variation. In *DATE*, pages 411–416, 2009.

[8] R. Dick, D. Rhodes, and W. Wolf. Tgff: task graphs for free. In *CODES*, pages 97–101, 1998.

[9] S. Hauck. Configuration prefetch for single context reconfigurable coprocessors. In *FPGA*, pages 65–74, 1998.

[10] J. Hsieh et al. An enhanced leakage-aware scheduler for dynamically reconfigurable fpgas. In *ASP-DAC*, pages 661–667, 2011.

[11] P. Kumar and L. Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *ASP-DAC*, pages 123–128. IEEE Press, 2011.

[12] I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *TCAD*, 26:203–215, 2007.

[13] F. Li, Y. Lin, and L. He. Field programmability of supply voltages for fpga power reduction. *TCAD*, 26:752–764, 2007.

[14] A. Raghunathan, N. K. Jha, and S. Dey. *High-Level Power Analysis and Optimization*. 1998.

[15] A. Sathanur et al. Row-based power-gating: a novel sleep transistor insertion methodology for leakage power optimization in nanometer cmos circuits. *VLSI*, 19:469–482, 2011.

[16] M. Shafique, L. Bauer, and J. Henkel. Remis: Run-time energy minimization scheme in a reconfigurable processor with dynamic power-gated instruction set. In *ICCAD*, pages 55–62, 2009.

[17] A. K. Singh et al. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In *DAC*, pages 115:1–115:7, 2013.

[18] A. K. Singh et al. Mapping real-life applications on run-time reconfigurable noc-based mpsoc on fpga. In *FPT*, pages 365–368, 2010.

[19] O. Sinnen. *Task scheduling for parallel systems*, volume 60. 2007.

[20] C. Steiger, H. Walder, and M. Platzner. Heuristics for online scheduling real-time tasks to partially reconfigurable devices. *FPGA*, pages 575–584, 2003.

[21] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger. A 90nm low-power fpga for battery-powered applications. In *FPGA*, pages 3–11, 2006.

[22] T. Tuan and B. Lai. Leakage power analysis of a 90nm fpga. In *CICC*, pages 57–60, 2003.

[23] Xilinx. Partial reconfiguration user guide. Technical report.

[24] S. Yang et al. Accurate stacking effect macro-modeling of leakage power in sub-100 nm circuits. In *VLSI Design, 2005. 18th International Conference on*, pages 165–170, 2005.

[25] P. Yuh et al. Leakage-aware task scheduling for partially dynamically reconfigurable fpgas. *TODAES*, 14:52, 2009.

[26] M. Zapater et al. Leakage and temperature aware server control for improving energy efficiency in data centers. In *DATE*, pages 266–269, 2013.