electronics

Article

AMROFloor: An Efficient Aging Mitigation and Resource Optimization Floorplanner for Virtual Coarse-Grained Runtime Reconfigurable FPGAs

Zeyu Li, Zhao Huang *, Quan Wang and Junjie Wang

School of Computer Science and Technology, Xidian University, Xi'an 710071, China; zeyuli@stu.xidian.edu.cn (Z.L.); qwang@xidian.edu.cn (Q.W.); junjiewang@stu.xidian.edu.cn (J.W.)
* Correspondence: z_huang@xidian.edu.cn; Tel.: +86-187-9261-0378

Abstract: With the rapid reduction of CMOS process size, the FPGAs with high-silicon accumulation technology are becoming more sensitive to aging effects. This reduces the reliability and service life of the device. The offline aging-aware layout planning based on balance stress is an effective solution. However, the existing methods need to take a long time to solve the floorplanner, and the corresponding layout solutions occupy many on-chip resources. To this end, we proposed an efficient Aging Mitigation and Resource Optimization Floorplanner (AMROFloor) for FPGAs. First, the layout solution is implemented on the Virtual Coarse-Grained Runtime Reconfigurable Architecture, which contributes to avoiding rule constraints for placement and routing. Second, the Maximize Reconfigurable Regions Algorithm (MRRA) is proposed to quickly determine the RRs' number and size to save the solving time and ensure an effective solution. Furthermore, the Resource Combination Algorithm (RCA) is proposed to optimize the on-chip resources, reducing the on-Chip Resource Utilization (CRU) while achieving the same aging relief effect. Experiments were simulated and implemented on Xilinx FPGA. The results demonstrate that the AMROFloor method designed in this paper can extend the Mean Time to Failure (MTTF) by 13.8% and optimize the resource overhead by 19.2% on average compared to the existing aging-aware layout solutions.

Keywords: FPGA; aging-aware layout; MTTF; resource optimization; genetic algorithm

Citation: Li, Z.; Huang, Z.; Wang, Q.; Wang, J. AMROFloor: An Efficient Aging Mitigation and Resource Optimization Floorplanner for Virtual Coarse-Grained Runtime Reconfigurable FPGAs. *Electronics* **2022**, *11*, 273. https://doi.org/10.3390/electronics11020273

Academic Editors: Deok-Hwan Kim and Mehdi Pirahandeh

Received: 10 November 2021
Accepted: 12 January 2022
Published: 15 January 2022

1. Introduction

Field Programmable Gate Array (FPGA) is a highly integrated semi-custom electrical device with the characteristics of high parallelism, low power consumption, and fast calculation speed. This device has been the core component of computing systems in many fields [1–3]. However, with the dramatic reduction of CMOS process size, FPGAs face increasingly severe reliability issues related to aging. Aging not only causes a decrease in Mean Time to Failure (MTTF) but also triggers an increase in failure rate [4], which can seriously affect FPGAs in highly reliable applications such as aviation, aerospace, and nuclear energy. Therefore, the impact of aging effects must be considered during the application phase.

Many efforts [5–7] have been devoted to FPGA aging mitigation to deal with these problems effectively. Among them, the aging-aware layout technologies are the current mainstream solution [7]. This approach aims to change the layout planning of logical resources and tasks, thereby reducing maximum stress and increasing Mean Time to Failure (MTTF). Most of the early research on aging-aware layout designs focus on using homogeneous RRs [5,6]. However, the number of RR determining the effect of aging mitigation is limited by the requirement of task resources and the size of the FPGA. The above issues inspire us to use heterogeneous RRs for more flexible layout tasks [7]. RRs' reasonable number and size are crucial to adopting heterogeneous RRs for the aging mitigation layout. However, the existing studies have adopted Design Space Explore (DSE)

or methods based on empirical attempts to determine the number and size of RRs [8], which need a long time for solving, and it cannot ensure getting an optimal layout solution. In addition, the heterogeneous RRs-based layout solutions are usually restricted by the rules of placement and routing in the underlying. Therefore, the layout of heterogeneous RR in the overlay layer has become a more efficient choice [9–11].

Even though the existing aging-aware layouts can mitigate the FPGA aging to a certain extent, these benefits come with occupying almost all the resources on-chip. However, none considered using fewer resources to achieve the same effect of aging mitigation [5–8]. The available research [8] illustrates that the Mean Time to Failure (MTTF) of the FPGA is determined by the RR with Maximum Stress (*RR_MS*). Hence, there is a theory that if some RRs are merged (some resources free), the maximum MTTF of the FPGA will not be changed as long as the accumulated stress in the merged RRs does not exceed the current *RR_MS*. For this, we investigate the On-Chip Resource Utilization (CRU) of three floorplanning solutions with different resources through a set of experiments to verify this idea. Figure 1 presents the experimental results under three different on-chip resource. From Figure 1, the resources occupied by the three layout solutions are not the same while obtaining the same MTTF. The experiments argue the possibility of optimizing resources by changing the layout solution while ensuring the same effects of aging mitigation.

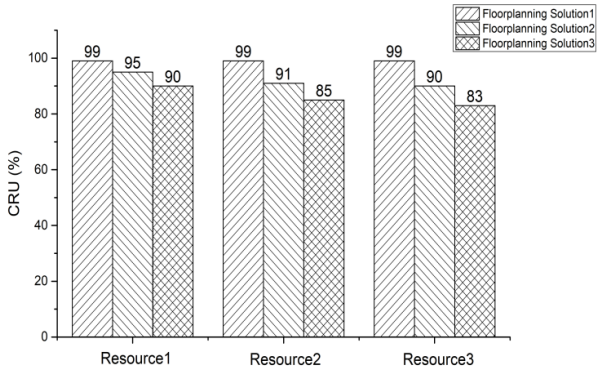

Figure 1. CRU corresponding to three floorplanning solutions with different resource.

These issues above open the possibility of achieving FPGA aging mitigation and resource optimization based on heterogeneous RRs. To avoid the restrictions of placement and routing rules on layout heterogeneous RRs, our research aims to introduce aging mitigation in heterogeneous RRs based on FPGA overlays. Moreover, we should quickly determine the appropriate number and size of RRs to reduce the solving time and optimize the issue of the existing layout planning occupying too many on-chip resources. Therefore, we proposed a floorplanner method for aging mitigation and resource optimization for virtual coarse-grained runtime reconfigurable FPGAs—AMROFloor. In this regard, we make the following contributions:

1. It is the first time to achieve aging mitigation and resource optimization for Virtual Coarse-Grained Runtime Reconfigurable Architecture (VCGRRA);
2. A Maximize Reconfigurable Regions Algorithm (MRRA) is proposed to quickly determine the number and size of RRs that are most conducive to aging mitigation, which improves the convergence rate of the algorithm and ensures a better layout solution;
3. A Resource Combination Algorithm (RCA) is proposed to further optimize the resources of the layout planning that has achieved aging mitigation;
4. Experimental results show that the AMROFloor method can extend the Mean Time to Failure (MTTF) of FPGAs by 13.8% and optimize the resource overhead by 19.2% on average.

We provide a brief overview of aging mitigation techniques in FPGA and VCGRRA in Section 2. In Section 3, we provide a detailed description of the study background and re-

lated technologies. The aging mitigation floorplannner is detailed in Section 4. In Section 5, we describe the content of the resource optimization algorithm. The experiment setup and results for evaluating the proposed design methodology are described in Section 6. Finally, we conclude the paper in Section 7 with directions and scope for future work.

2. Related Work

FPGA devices tend to degrade with time and stress. The continuous scaling of transistor size exacerbates the influence of different aging mechanisms, such as Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI), Electromigration (EM), and Time-Dependent Dielectric Breakdown (TDDB), which accelerate the degradation of circuits [12].

Various approaches have been proposed for mitigating the aging effects in FPGAs. Dynamic Voltage Scaling (DVS) technology is a common method for the anti-aging of many CMOS devices, including FPGA [13], but it is easy to cause an excessive drop in operating frequency. With the development of DPR techniques [14], the reconfigurable fabrics can be divided into multiple RRs to be used in a time-division multiplexing manner, which opens up the possibility of using an aging-aware layout planning to mitigate the aging of FPGA. In fact, the aging-aware layout is achieved by changing the accumulated stress of the logic resources used to reduce the maximum stress and increase the MTTF [5,6,8,15–17].

Khaleghi et al. [15], concerning transistor performance degradation on FPGA wiring paths, proposed a run-time wiring approach to prevent transistor aging by distributing stress uniformly over interconnect resources. In the work of Zhang et al. [6,16], methods are proposed to reduce the maximum stress on fine-grained FPGA configurable logic blocks (CLBs) by periodically swapping between different CLB configurations. Furthermore, this team proposed an aging-aware placement method for gas pedals across layers in FPGA-based run-time reconfigurable architectures to reduce the degree of hardware wear within and between RRs [5,6]. Although it can dynamically balance on-chip stresses, this run-time placement approach requires real-time monitoring of aging information and calculation of stresses, which has significant resource and time overhead. In the work of Ghaderi et al. [17], an aging-aware FPGA layout planning was proposed to reduce the stress time at the block level or system level by using a delay-based degradation estimation model. Sahoo et al. [8] proposed a remapping method that maximizes the system MTTF on heterogeneous, dynamic, partially reconfigurable, fine-grained FPGAs using an aging-aware scheduler. It is the first time we attempt using heterogeneous RRs layouts to mitigate aging, which effectively improve the MTTF. However, the heterogeneous RRs-based layout solutions are usually restricted by the rules of placement and routing in the underlying structure. In addition, the solution process of DSE is time consuming, since the solution space is quite ample, and the layout solution usually occupies almost all on-chip resources.

Regarding aging mitigation on VCGRRAs, most researchers consider placing tasks on different fabrics to alleviate the accumulation of stress. Srinvasan et al. [18] suggested periodically remapping the design to the less-used region by using two different configurations and switching between them to mitigate HCI degradation. Gu et al. [19] provided a rotation-based mapping strategy to balance the pressure on multi-context CGRRAs. Similarly, Afzali-Kusha et al. [20] proposed a method to reduce the CGRRA temperature by generating increments using different configurations of different PEs. Based on this, Hu et al. [21] further considered the performance degradation from these stress–time re-balancing strategies and propose an aging-aware layout scheme that takes into account critical path delays. However, almost of the aforementioned approaches do not employ aging-aware layout solutions. Moreover, these methods only target tasks with multiple contexts and execute only a single task at the same time without considering the impact on virtual layer stresses when multiple tasks are placed in parallel.

The aging-aware layout planning based on balance stress effectively achieves aging mitigation. However, the layout solution based on heterogeneous RRs at the bottom probably cannot be realized due to violating placement and routing rules. Moreover, the

existing methods need to take a long time to solve the floorplanner, and the corresponding layout solution occupies almost all on-chip resources. To solve these limitations, we propose an aging mitigation and resource optimization method for Virtual Coarse-Grained Runtime Reconfigurable FPGAs. On the one hand, we aim to determine a reasonable number and size of RRs quickly to save the computing time and ensure an effective solution; on the other hand, we optimize on-chip resources for efficient resource utilization.

3. Preliminary

Before describing the details of our method in Sections 4 and 5, we describe the architecture of VCRGGA in detail and clarify representations about the RR and task and how we evaluate the MTTF of RR and task.

3.1. VCGRRA

VCGRRA is an FPGA overlay that is conceptually located between the user application and the physical FPGA. User applications are not implemented directly on the physical FPGA but in the intermediate architecture. This shields the difference in resource requirements for different tasks and improves the portability and compatibility of tasks. Therefore, VCGRRA can be used to define quickly and dynamically change custom applications without (re)compilation, shortening the entire design cycle [11,22–24].

The calculation part of VCGRRA is composed of PE, which is defined at a higher abstract level, corresponding to the resources of the physical layer such as LUT, DSP, and BRAM. PE is a coarse-grained calculation execution unit [25]. To meet the concurrent execution of multiple tasks simultaneously, PE resources can be integrated into RRs of different sizes to be compatible with different tasks. The VCGRRA structure and task mapping process proposed in this paper are shown in Figure 2. In VCGRRA, layout planning can be used to balance the working stress in each RR for achieving excellent aging relief and area efficiency.

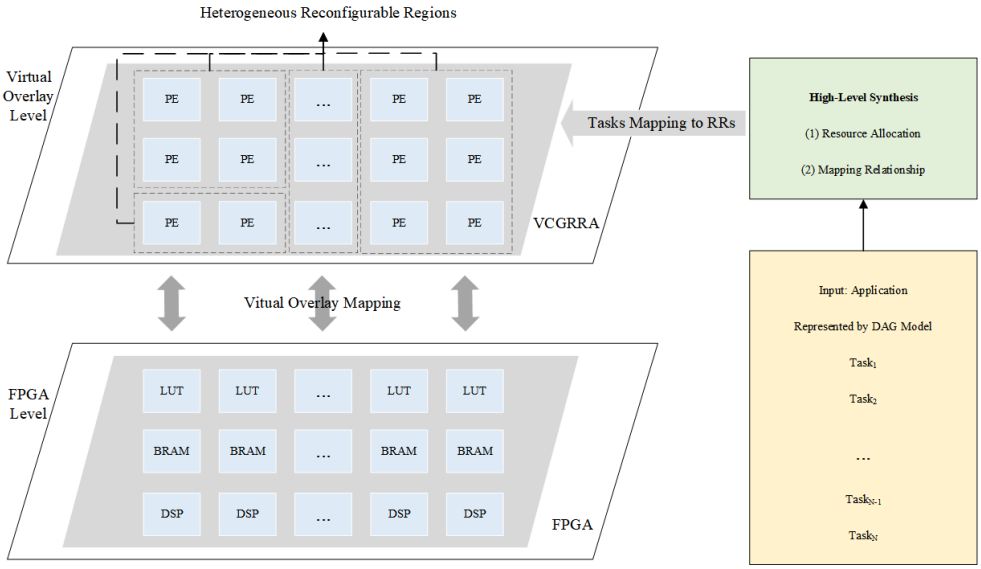

Figure 2. Overview of an FPGA with VCGRRA overlay and multi-tasks mapping flow.

CAD supports state-of-the-art VCGRA flows from High-Level Synthesis (HLS) to placement and routing. By using widely used programming languages (e.g., ANSI C or C++) in HLS, higher design productivity and shorter design times can be achieved. To build VCGRRA, the PE and interconnect routing are first designed. Then, the VCGRRA grid is constructed. In this paper, the VCGRRA is made using the tool described in [26]. The tool automatically creates the top-level VHDL description of the VCGRRA based on the description of the hardware architecture. The grid structure is described by the number of PEs in each layer of the architecture and the input and output bandwidth of the elements.

In this paper, we use heterogeneous RRs to deploy tasks to increase the flexibility of the layout for better realizing on-chip aging mitigation. RR is the basic unit of the deployment task and the smallest granularity that characterizes the stress. Its parameters are shown in Table 1. Among them, RR_{num} is the unique number of RR, *RR_R* is the number of resources contained in RR, *RR_MTTF* is the estimated MTTF of RR, *RR_Stress* is the stress accumulated on RR, and *RR_Exec* represents the sequence of tasks executed on the RR. The series depends on the final task scheduling strategy.

Table 1. RR paremeters.

Parameter	Description
RR_{num}	Number of RRs
RR_R	Resources included in the RR
RR_MTTF	Expected MTTF of RR
RR_Stress	Accumulated stress on RR
RR_Exec	Queue of tasks executed on RR

3.2. DAG Task Model

In this paper, the tasks are modeled using Directed Acyclic Graph (DAG) in the mathematical form G_{app} (T_{app}Tapp, E_{app}, P_{app}), where T_{app}Tapp denotes the set of tasks, E_{app} denotes the set of dependencies between these tasks, and P_{app} denotes the total time for all tasks to be executed and completed. The parameters of the task are described as shown in Table 2.

Table 2. Task parameters.

Parameter	Description
$Task_{num}$	Number of tasks
Task_R	Resource used for task implementation
Task_MTTF	Expected MTTF of task
Task_Stress	Expected stress of task
Task_S	Start time of task
Task_E	Execution time of task
Task_D	Deadline of task

Among them, the number of each task is unique, and it is represented by $Task_{num}$. *Task_R* is the resource used for task implementation. *Task_MTTF* is the expected MTTF of the task, and *Task_Stress* is the expected stress of the task. *Task_S*, *Task_E*, and *Task_D* represent the start time, execution time, and execution time of the task, respectively. Deadline, *Task_E*, and *Task_D* are known, and *Task_S* is determined by the constraint relationship of task scheduling.

3.3. Task_Stress/RR_MTTF Evaluation

The methods used to evaluate *Task_Stress* and *RR_MTTF* in this section are the same as those used in [8]. *RR_MTTF* is determined jointly by the *Task_Stress* and the execution time of all the tasks running on it. The *Task_E* is given by the user, while the *Task_Stress* needs to be calculated. Note that the relationship of *Task_Stress* and *task_MTTF* is inversely proportional. Hence, we employ the EM model to calculate the value of *Task_MTTF* and then infer the *Task_Stress*. EM is the leading mechanism of device failure in long-term operation compared with the other aging effects. Nowadays, the normal lifetime of most of the current FPGAs is more than ten years, so the EM model is more suitable for this paper, and its equation is shown in (1):

$$\eta(T_i) = \frac{A_0(J - J_{crit})^{-n} e^{\frac{Ea}{kT_i}}}{\Gamma(1 + \frac{1}{\beta})} \tag{1}$$

where A_o denotes the linewidth constant, which is determined by the properties of the metal interconnects within the FPGA, and β, the shape parameter, can be used to represent the hardware fault profile. In this paper, we do not consider the effect of process variations in the model. Hence, the shape parameter β remains constant. k denotes the Boltzmann constant, $J - J_{crit}$ denotes the offset of the current density with and without electromigration, T denotes the Kelvin temperature, and E_a denotes the activation energy. The above parameters can be obtained by FPGA design software and power analysis tools, and the flow of calculating *Task_Stress* is shown in Figure 3.

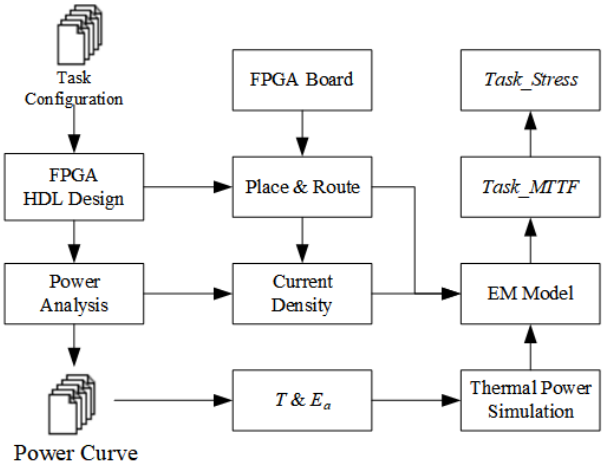

Figure 3. Evaluation flow of the *Task_Stress*.

In this paper, considering that the RR resources are time-divisionally multiplexed for different tasks, and therefore, the aging caused is time-dependent, the reliability is characterized by time segmentation, as shown in (2) [27]:

$$\eta_{eff} = \frac{1}{\sum(\frac{\Delta t_i}{\eta_i})}; t = \sum \Delta t_i; \eta_i = \frac{MTTF_i}{\Gamma(1+\frac{1}{\beta})} \tag{2}$$

where η_{eff} denotes the actual aging of the RR caused by the task in the time segmentation case. η_i and $MTTF_i$ are positively correlated and denote the summation of the segmentation time, i.e., the total time that a particular RR is occupied during one application execution, and β is the shape parameter. Based on this formula, the average failure time of a single reconfigurable region can be obtained by combining the task model and RR model described in Sections 3.1 and 3.2, as follows:

$$RR\_MTTF_r = \frac{1}{\sum_{i=1}^{M} ExecT_i \times Task\_Stress_i} \tag{3}$$

where RR_MTTF_r denotes the MTTF of the r-th RR, $Task\_Stress_i$ denotes the *Task_Stress* value of the i-th task, $ExecT_i$ is the execution time of the i-th task, and M denotes the total number of executed tasks on the r-th RR.

4. Aging Mitigation Floorplanner Based on GA

The aging mitigation floorplanner is essentially the objective optimization problem. We propose a Genetic Algorithm (GA)-based optimization method to determine the Task to RR mapping for maximizing the MTTF of FPGA. In this section, GA is employed to model the problem, and the required optimization objective function and related constraints are defined. In addition, MRRA was proposed to determine the number and size of RRs quickly.

4.1. Problem Description

Aging-aware layout planning generates a task to RR mapping to balance the stress on the used logic resources. Therefore, solving the mapping relation essentially belongs to the objective optimization problem. The well-known GA is a powerful optimization method for solving such problems. To get an effective layout solution, it is necessary to model the optimization problem reasonably. In this paper, GA is selected as a heuristic method to solve the aging mitigation layout solution, which is mainly based on three reasons: (1) Genetic Algorithm has strong applicability and fewer application restrictions. It can well model the problem of aging mitigation layout; (2) It is easy to generate new layout schemes and evaluate by using the coding method of genetic algorithm; (3) It is scalable and easy to combine with other algorithms. In this paper, the number and size of RRs obtained by MRRA are the input of GA, and the output of GA is the input of the RCA algorithm.

In GA, a population is a collection of individuals, and individuals are entities with chromosome characteristics. Population and individuals are shown in Figure 4. Each chromosome corresponds to an individual, namely a set of solution results. Chromosomes are composed of genes, and genotypes are encoded by phenotypes that can be regarded as independent variables.

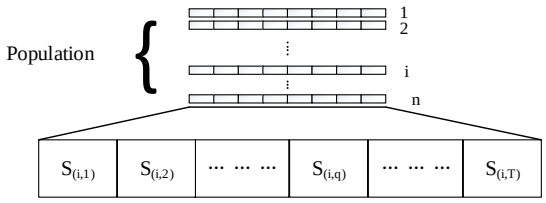

Figure 4. Population and chromosomes individuals.

It can be seen that the population contains n individuals corresponding to n chromosomes. Taking the chromosome corresponding to the i-th individual as an example, $s(i,q)$ represents the q-th gene on the i-th chromosome. After decoding, the gene will correspond to the q-th independent variable value in the phenotype. The independent variable is shown in Figure 5:

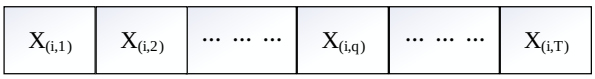

Figure 5. Independent variable of phenotype.

Among them, $X(i,q)$ is derived from the decoding of gene $s(i,q)$. In this paper, the independent variables in the phenotype are defined as follows: $X(i,q)$ represents the mapping relationship between the q-th task and a RR on FPGA, and its value corresponds to a RR_{num}. Therefore, a chromosome can correspond to a layout solution. The evolution of the genetic algorithm is to use selection, crossover, mutation, and other operations to finally get the individual with the strongest adaptability, that is, the layout solution that can achieve the optimal aging mitigation effect.

4.2. Objective and Constraints

We use MTTF to represent the maximum lifetime of the whole FPGA. Note that there are dependencies between the tasks deployed on the FPGA. If one RR turns to breakdown, it will lead to the entire offline layout solution failing. Hence, the FPGA device is regarded as a whole in this paper, and the MTTF of the device is equivalent to the minimum RR_MTTF (RR_MTTF_{min}) across all RRs. This is also similar to the viewpoint of Sahoo et al. and Hu et al. [8,21]. That is:

$$MTTF = RR\_MTTF_{min}. \quad (4)$$

At this point, the optimization objective is to maximize the RR_MTTF_{min}:

$$Objective : Max\{RR\_MTTF_{min}\}. \tag{5}$$

The purpose of constraints is to ensure the rationality and practicability of the solution results. This paper mainly considers resource constraints, timing constraints, and the dependency between tasks. In terms of resource constraints, it is necessary to ensure that the hardware resources contained in the RR can meet the resource requirements of the tasks arranged on the RR. For every RR r across all RRs and every *Task_R* m, that is:

$$RR_r \geq Max\{Task\_R_1, Task\_R_2, \ldots, Task\_R_m\}. \tag{6}$$

Among them, when m tasks are mapped to an RR for execution, the *RR_R* of this RR must be greater than or equal to the maximum resources required in tasks mapped to this RR. Moreover, it is necessary to ensure that the overall resource size of the reconfigurable regions corresponding to the layout solution cannot exceed the total resources used on the chip. For every *RR_R* e, that is:

$$\sum_{e=1}^{n} RR\_R_e \leq R. \tag{7}$$

Regarding timing constraints, task deadlines and task dependencies are mainly considered to ensure that all tasks can be executed correctly. Almost all tasks require the latest completion time, namely the *deadline* attribute. Assuming that the task set to be scheduled T, for every task i, the characterization of the *deadline* constraint is shown in (8):

$$\forall i \in T, Task\_S_i + Task\_E_i \leq Task\_D_i. \tag{8}$$

Tasks often have execution order requirements. For example, task A must be executed after task B is completed, that is, B is the front task of A. Assuming that the task set to be scheduled T and $task_i$ is the predecessor of $task_j$, which can be expressed as $i \rightarrow j$, the representation of dependency between tasks is shown in (9):

$$\forall i \in T, j \in T, i -> j, Task\_S_j > Task\_S_i + Task\_E_i. \tag{9}$$

4.3. Number and Size of RRs

The appropriate number and size of RRs used in the layout solution is the key to achieving aging mitigation. However, there are many possibilities for the number and size of RRs that can be chosen in a heterogeneous layout solution. Existing methods use genetic algorithms or MILP modeling combined with DSE for solving such problems. However, the solution procedure can be very time consuming, because the solution space can be very large. In addition, the number and size of RRs obtained by the current methods are not necessarily good results. The reason is that the results depend on the empirical setting or the scope of the search of the solution model itself. Assuming that one RR placement can be provided for all tasks, the aging mitigation effect will be optimal, because the stresses are almost equally distributed. Inspired by this idea, constructing as many RRs as possible is the key to achieving the optimal aging-aware layout. For this, we proposed Maximize Reconfigurable Regions Algorithm (MRRA) for determining the reasonable number and size of RRs.

We show the pseudo-code of MRRA (Algorithm 1) and briefly explain it. In line 2, quantify the *Task_R* and the total resource of the chip in terms of the number of PEs. Then, initialize the number of RRs equal to the number of tasks and the resources of each *RR_R* similar to *Task_R* (line 4, 6). Next, determine the number of RRs and the number of resources for each RR (lines 8–22). First, determine whether the sum of the current *RR_R* is less than the total amount of on-chip resources. If it is less, the number and size of the initialized RR are output (lines 8–11). Otherwise, sort *RR_R* by descending order (line 12). Combine the two RRs with the smallest *RR_R* in turn, and constantly judge the relationship

between the sum of the current*RR_R* and the total amount of resources on the chip. As long as the sum of the current *RR_R* is less than the total amount of on-chip resources, the number and size of the current RR will be output (lines 13–21).

Algorithm 1: MRRA

```
   Input: Task_R; NT: Number of Tasks;
   Output: RR_R; NRR: Number of RRs;
   NPE: Number of PEs; R: All resources on-chip
(1) /* Quantify the resources in terms of the number of PEs */
(2) Quantification Task_R → NPE; R → NPE;
(3) /* Initialize the number of RRs to the number of tasks */
(4) Initialization NRR → NT;
(5) /* Initialize the resources of each RR_R to Task_R */
(6) Initialization RR_R → Task_R;
(7) /* Determine the number of RRs and the amount of resources for each RR */
(8) If R ≥ sum (RR_R) then
(9)       Output RR_R; NRR;
(10)      breaks;
(11) else
(12)      sorting RR_R by descending order;
(13)      while (NRR > 1)
(14)           combine the two RRs with the smallest RR_R in the sort into the larger one;
(15)           NRR = NRR -1;
(16)           If R ≥ sum (RR_R) then
(17)                Output RR_R; NRR;
(18)                breaks;
(19)           end
(20)      end
(21)      Output RR_R = R; NRR = 1;
(22) end
```

5. Resource Combination Algorithm

The current offline aging-aware layout solution has the problem of resource waste. The fundamental reason is that the goal of aging mitigation is to balance the on-chip stresses. Therefore, maximizing the usage of resources to generate a larger number of RRs to place the task is the key to balancing the stresses. However, the MTTF of the whole chip is determined by the minimum *RR_MTTF*. When the *RR_MTTF* cannot be reduced, the aging mitigation effect is optimal at this point. Therefore, if tasks in some RRs are combined, there is a possibility to optimize resources as long as the accumulated stress does not exceed the maximum *RR_Stress*. Inspired by this, we proposed a resource combination algorithm (RCA) to try to combine RR resources to reduce on-chip resource usage and achieve the best on-chip resource utilization while guaranteeing the maximum *RR_Stress* is constant.

We show the pseudo-code of the RCA (Algorithm 2) and briefly explain it. In lines 1–3, Initialize 3 lists—ϕ, γ, and δ, to store the number and resource of RRs, the queue of tasks executed on RRs, and the accumulated stress on RRs, respectively. In the first step (line 5), try to merge any two RRs of all RRs into the larger one and combine the tasks in them. In the second step (line 7), calculate the *RR_Stress* of each RR and store the merged RR in the list δ in descending order of *RR_Stress*. In the third step (lines 9–22), compare the stress differences of all RRs, and check if constraints are met after merging. First, compare the values of the merged *RR_Stress* and the *RR_MS* of the original layout in turn (line 11). If the current *RR_Stress* is less than *RR_MS*, continue to check the constraints (line 12). If the constraints are met, the combination of these two RRs is effective. Then, update the list ϕ and γ and repeat steps 1–3. If the current *RR_Stress* is greater than *RR_MS* or the constraint conditions are not met, compare the next one (lines 17 and 20). In the fourth step, check if the list ϕ and γ have been updated. If the number of RRs is the same as NRR, it proves that resources are not optimized and output I_FS (line 25). Otherwise, prove that the resources are optimized and output RO_FS (line 27).

Algorithm 2: RCA

Input: Initial Floorplanning Solution (I_FS);
Output: Resource Optimization Floorplanning Solution (RO_FS);
RR_MS: the RR with max stress in the IFS; *NRR*: Number of Initial RRs;
(1) **Initialization** list ϕ to store the number and resource of RRs;
(2) **Initialization** list γ to store the *RR_Exec* of each RR;
(3) **Initialization** list δ to sore *RR_Stress* of each RR;
(4) /* **Step1:** Merger any two RRs into the larger one */
(5) $Max(RR_i, RR_j) \rightarrow RR_{ij}$; $RR_i\_Exec + RR_j\_Exec \rightarrow RR_{ij}\_Exec$; $(i, j \in \phi, \gamma)$
(6) /* **Step2:** Calculate the *RR_Stress* after the merger and sort them by descending */
(7) $Calculate(RR_{ij}\_Stress)$; $Sort(RR_{ij}\_Stress) \rightarrow \delta$;
(8) /* **Step3:** Compare stress differences and check if constraints are met after merging */
(9) r = 0;
(10) **while** (r < δ.length)
(11) **If** *RR_MS* - $RR_{ij}\_Stress[r] > 0$ **then**
(12) check constrains (return **Boolean**);
(13) **If True then**
(14) update ϕ, γ;
(15) jump to the **step1**;
(16) **else**
(17) r = r + 1;
(18) **end**
(19) **else**
(20) r = r + 1;
(21) **end**
(22) **end**
(23) /* **Step4:** Determine if ϕ and γ have been updated */
(24) **If** ϕ.size == *NRR* then
(25) **Output** I_FS;
(26) **else**
(27) **Output** RO_FS;
(28) **end**

6. Experiments and Results

6.1. Experiment Setup

In this paper, the AMROFloor layout planning solution is built by the Python language. The specific experimental environment configuration is described as follows: CPU is Intel(R) Xeon(R) Silver 4116 @2.1GHz, memory is 32GB DDR4. This solution applies to most FPGAs configured with VCGRRA, and it has been simulated and implemented on Xilinx XCKU115-FLVB2104-2-E.

A total of 60 task benchmarks were selected from different open sources, such as CH-Stone, OpenCores, XILINX, etc. [28–30], and all these tasks make up the whole experiment database. Table 3 describes examples of eight tasks, which partly presents the differences of *Task_Stress* and the resource requirements (PE) of tasks. Moreover, these eight tasks are used in Section 6.4. Please note that the *Task_Stress* is evaluated by the existing method in Section 3.3, and our contributions do not include the evaluation of *Task_Stress*. Experiments for evaluation involved using task graphs with a varying number of tasks. These task graphs were generated using the Task Graphs for Free (TGFF) tool [31] to give task timing constraints and dependency constraints.

Table 3. Examples of descriptions of several tasks used in the experiments.

Task	Number	PE	Task_Stress
T1	0	79	3
T2	1	192	1.8
T3	2	534	5
T4	3	1011	1.1
T5	4	87	1.2
T6	5	3089	5
T7	6	1526	0.3
T8	7	787	2.6
...	...	...	...

In this paper, we select the relevant layout solutions in [7,21] for comparison. These include random layout (RL), homogeneous minimize makespan (Hom_MS) layout, heterogeneous shortest makespan (Het_MS) layout, homogeneous aging mitigation (Hoe_AM) layout, and heterogeneous aging mitigation (Het_AM) layout solutions. Table 4 presents the partition mode and purpose of these layout methods. It can be seen that AMROFloor is the only one that considers resource optimization.

- Hom_MS: The layout solution with homogeneous RRs aims at minimizing the makespan of the tasks;
- Het_MS: The layout solution with heterogeneous RRs aims at minimizing the makespan of the tasks;
- Hom_AM: The layout solution with homogeneous RRs aims at aging mitigation;
- Het_AM: The layout solution with heterogeneous RRs aims at aging mitigation;
- RL: Random layout solution.

Table 4. Compared methods in the experiments.

Method	Heterogeneous	Homogeneous	Aging Mitigation	Resource Optimization
Hom_MS		✓		
Het_MS	✓			
Hom_AM		✓	✓	
Het_AM	✓		✓	
RL		✓		
AMROFloor	✓		✓	✓

In the experiment, we design three resource fabric sizes, namely small, medium, and large, which correspond to the number of PEs as 10,000, 15,000, and 20,000, respectively. According to the difference of task resource requirements (small, medium, and large), 50 tasks are selected from the experiment database by three times the three task sets: 1, 2, and 3, which correspond to the small, medium, and large variation in task resource requirements, respectively. The parameter configuration of the GA is as follows: set the crossover probability as 0.5, the variation probability as 0.7, the initial population as 50, and the maximum generations as 20,000.

6.2. Evaluation Metrics

MTTF: Mean Time to Failure of the FPGA, common reliability indicators, is used to measure the effect of aging mitigation.

CRU: On-Chip Resource Utilization, the total resources contained in all RRs as a percentage of the total resources on the chip, is used to evaluate the resource usage on FPGA.

$$CRU = \frac{\sum_{i=1}^{n} RR\_R_i}{R} \times 100\% \tag{10}$$

where n denotes the total number of RRs and R denotes the total resources of the chip.

6.3. Results and Analysis

6.3.1. MTTF

We quantify the aging mitigation performance of our proposed methodology in terms of the MTTF. Figure 6 shows the detailed results for three task sets in order from top to bottom. Each subfigure presents the MTTF results for our method versus the five widely used methods under three different resource scenarios.

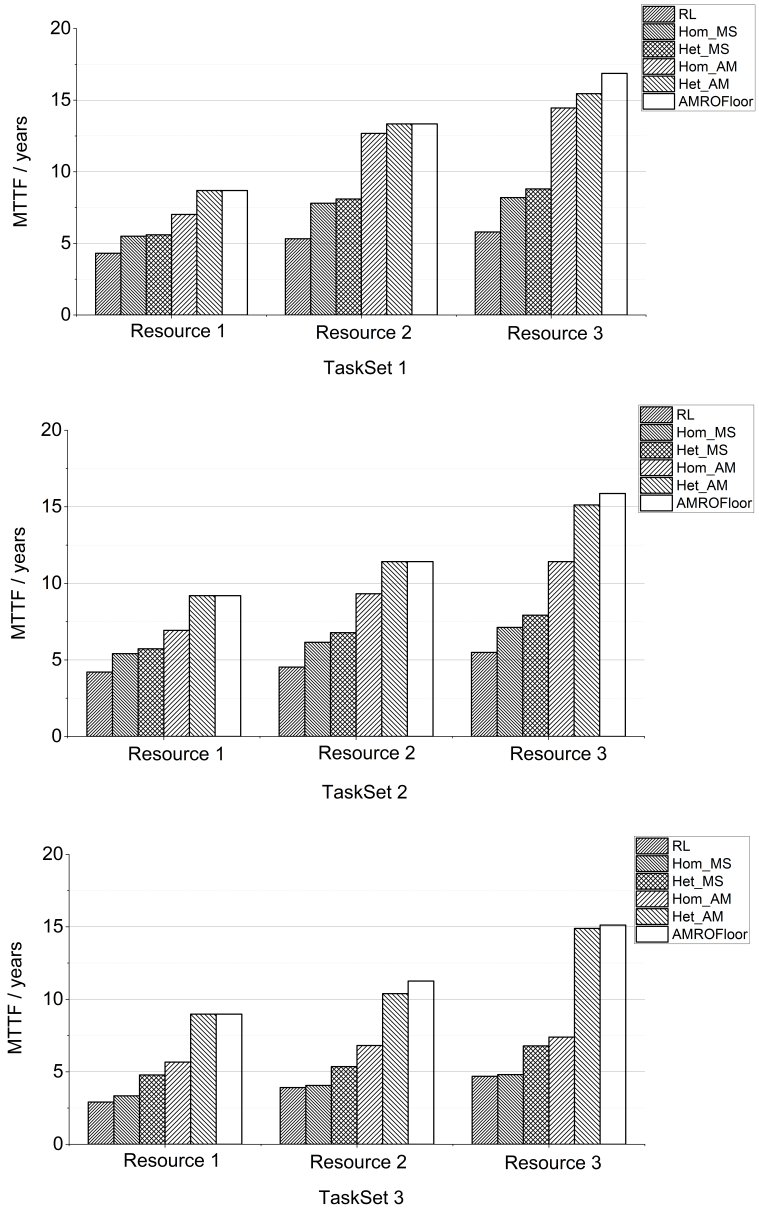

Figure 6. MTTF of different methods in three task sets under three resources.

It can be seen that the MTTF of the aging-aware layout solutions is significantly better than the MS and RL with the increase of total resources. Then, more on-chip resources and more RRs can be allocated to perform tasks to achieve a more balanced stress distribution. Meanwhile, with the increase of total resources, the MTTF of RL and MS increase slightly. Overall, the MTTF of MS is better than RL because the former has advantages in reducing the total task execution time. Reducing the whole task execution time will also make the number of tasks executed by each RR on the chip relatively balanced, which is conducive to increasing aging mitigation indicators. Comparing the results of homogeneous and heterogeneous aging-aware layout solutions, in addition to the situation

of the sufficient resources, the optimization effect of the heterogeneous layout in aging mitigation is significantly better than that of the homogeneous layout. Moreover, the larger the difference of task resource requirements in the task set further highlights the advantages of heterogeneous layouts. AMROFloor is based on an aging-aware layout based on the heterogeneous RRs, further using MRRA to obtain the appropriate number and size of RRs, which gives it a better layout solution. Hence, AMROFloor can averagely extend the MTTF by 61.6% than other layout solutions. In general, it illustrates that the AMROFloor can achieve a better aging mitigation effect of FPGA.

6.3.2. CRU

We evaluate the effect of resource optimization in terms of the CRU. Table 5 shows the experimental results of different task sets under three resource scenarios. AMROFloor is contrasted with the Hom_AM and Het_AM. The premise of CRU comparison is that the MTTF obtained by all methods is the same.

Table 5. CRU results.

Scenes/Methods		Hom_AM (%)	Het_AM (%)	AMROFloor (%)
TaskSet 1	Resource 1	99	99	99
	Resource 2	95	89	81
	Resource 3	90	77	69
TaskSet 2	Resource 1	99	99	99
	Resource 2	94	82	70
	Resource 3	85	61	50
TaskSet 3	Resource 1	99	98	94
	Resource 2	90	70	53
	Resource 3	83	56	41
Avg.		93	81	73

The experimental results show that the homogeneous layout solution is much more resource-intensive than the heterogeneous layout solution. The reason is that the size of RRs in the homogeneous layout is determined by the task with the highest resource requirement, reducing resource usage flexibility. In addition, the goal of aging mitigation makes it possible to generate as many RRs as possible, thus taking up a large amount of on-chip resources. In particular, as the difference in resource requirements of tasks in a task set becomes larger, a heterogeneous layout solution has a more obvious advantage. Both AMROFloor and Het_AM are heterogeneous layouts, but AMROFloor further optimizes resources based on implementing an aging-aware layout. When the total resources are small, the CRU of AMROFloor optimization is slightly better than that of Het_AM. As the complete resources increase, the CRU of AMROFloor optimization is significantly better than that of Het_AM. When the total resources are sufficient, the CRU of AMROFloor is reduced by up to 36.6% compared with that of Het_AM. In general, It can be seen that the AMROFloor is more beneficial to save on-chip resources.

6.3.3. Solution Efficiency

The MRRA algorithm proposed in this paper can quickly determine the number and size of RRs. The most significant advantage of this method is to accelerate the solving of layout solutions in addition to improving the MTTF. To verify the optimization effect, our method is compared with the Standard Genetic Algorithm (SGA) and MILP combined with DSE (MILP_DSE) used in [7]. The solving time is the main metric for comparison. The specific results are shown in Table 6, which is also the result of comparing different task sets in three resource scenarios.

Table 6. Solution efficiency.

Scenes/Methods		SGA (s)	MILP_DSE (s)	AMROFloor (s)
TaskSet 1	Resource 1	124.3	1153.2	46.3
	Resource 2	175.4	1899.8	68.6
	Resource 3	354.2	2386.1	116.3
TaskSet 2	Resource 1	133.9	1156.7	47.1
	Resource 2	197.7	1801.3	68.2
	Resource 3	312.1	2385.7	106.4
TaskSet 3	Resource 1	116.8	1158.2	56.2
	Resource 2	203.5	1800.4	77.9
	Resource 3	308.2	2382.1	119.3
Avg.		214.0	1791.5	78.5

It can be seen from Table 6 that the solving time of MILP_DSE has an average increase of more than 8× over the SGA and 22× over the AMROFloor. The reason is that the solution space of DSE is enormous, and many attempts are required to find a better solution. By comparing the results of SGA and AMROFloor, it can be seen that the solving time is independent of the task group condition and is only related to the total resource condition. Under the state of the same total resources, the solving time of the two algorithms changes very little when the task group condition is changed. Overall, AMROFloor still has a more significant advantage over SGA. The reason is that the MRRA algorithm reduces the spatial extent of the solution, which allows AMROFloor to converge as soon as possible. It indicates that predetermining the reasonable number and size of RRs does have an optimal effect on the convergence speed of the algorithm.

6.4. Case Analysis

Eight benchmark tasks are used in the case analysis. The PEs required by the tasks and *Task_Stress* are shown in Table 3. TGFF is used to generate a DAG diagram to represent task timing constraints and dependencies (Figure 7). All tasks are connected by a 64-bit bus.

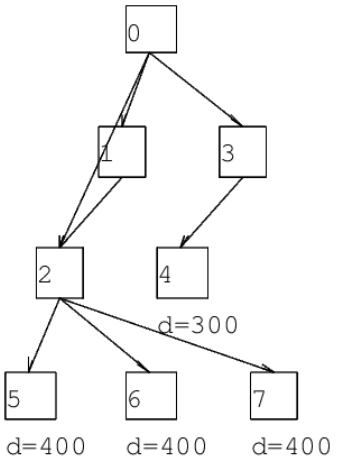

Figure 7. Task DAG diagram.

Taking the above tasks and DAG diagram as the data source, the AMROFloor l is compared with the Random layout in [21]. XCKU115-FLVB2104-2-E chip is used as the target chip. The chip includes 663360 LUT, 2160 BRAM, 5520 DSP, etc., resources.

In this case, 5% of the total on-chip resources are selected to place the task. Figure 8 shows the comparison of the results of the two layout solutions. The Random layout uses 4.95% of on-chip resources, while the AMROFloor uses 4.16% of on-chip resources;

In terms of on-chip stress, the stress distribution of AMROFloor is more balanced. The maximum single RR stress of the Random layout is 7.4, and the maximum single RR stress of AMROFloor is 5. In summary, the CRU of AMROFloor is reduced by 15.96%, and the MTTF is increased by 32.5%.

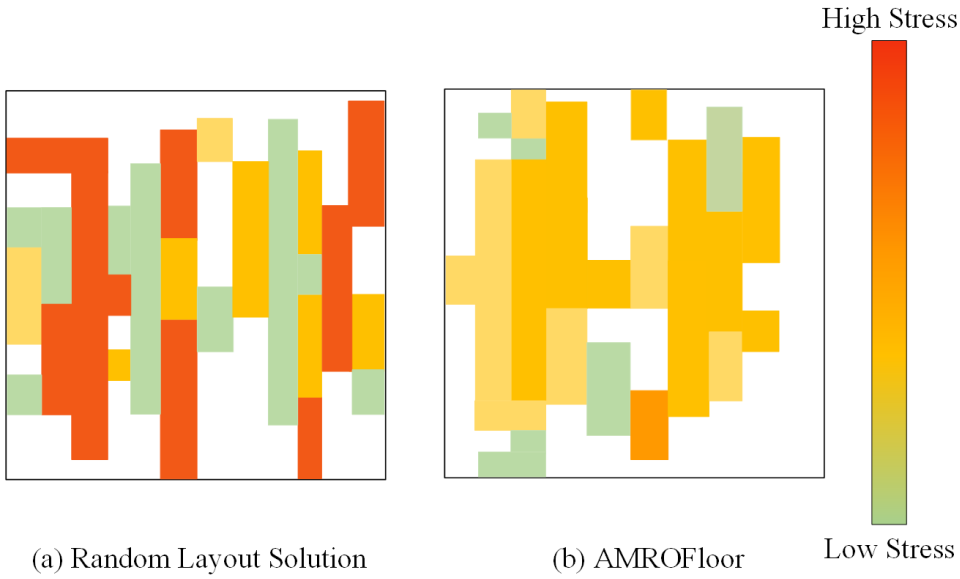

Figure 8. The comparison of AMROFloor and Random layout solution.

7. Conclusions

In this work, we proposed a layout planning method for aging mitigation and resource optimization for virtual coarse-grained runtime reconfigurable FPGAs—AMROFloor. In the VCGRRA overlay layer, PEs are integrated as heterogeneous RRs to use FPGA resources more effectively to improve system life. The aging-aware layout planning is modeled based on GA, and the MRRA algorithm is proposed to quickly determine the size and number of RRs to accelerate the solution efficiency of the layout planning. At the same time, the RCA algorithm is proposed to optimize further the layout planning resources that have achieved aging mitigation. Compared with the existing aging-aware layout solutions, Hom_AM and Het_AM, the experimental results show that AMROFloor can averagely extend the MTTF by 13.8% while reducing resource usage by 19.2%. In terms of solution efficiency, AMROFloor has an average increase of more than 22× over the MILP_DSE method.

This method also has certain limitations, and it is tough to obtain an effective aging mitigation layout plan under resource-constrained conditions. Hence, AMROFloor is only expected to work for most (not all) applications/devices. Moreover, the current method does not consider the impact of failures on layout planning. In future work, we will explore the possibility of using idle resource-optimized by RCA as fault-tolerant redundant resources to obtain appropriate trade-offs between aging mitigation and tolerance against failures.

Author Contributions: Conceptualization, Q.W. and Z.H.; methodology, Z.H. and Z.L.; software, Z.L. and J.W.; validation, Z.H. and Z.L.; writing—original draft preparation, Z.L.; writing—review and editing, Z.H. and J.W.; supervision, Q.W.; project administration, Q.W.; funding acquisition, Q.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Natural Science Foundation of China (Nos. 61972302 and 61962019) and the Group Intelligence Behavior Analysis-based Cultural Material Identification and Digital Product Development & Reuse (2019YFB1406402), and is partly supported by the Shannxi Key Technology R&D Program (Nos. 2021ZDLGY07-01 and 2021ZDLGY07-04), and is partly supported by the Key Laboratory of Smart Human–Computer Interaction and Wearable Technology of Shaanxi Province.

Acknowledgments: The authors would like to thank the editors and reviewers for their efforts and suggestions to improve our manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shokry, B.; Mahmoud, D.G.; Amer, H.H.; Shatta, M.; Alkady, G.I.; Daoud, R.M.; Adly, I.; Shaker, M.N.; Refaat, T. Work-in-Progress: Triple Event Upset Tolerant Area-Efficient FPGA-Based System for Space Applications and Nuclear Plants. In Proceedings of the 16th IEEE International Conference on Factory Communication Systems (WFCS), Porto, Portugal, 27–29 April 2020; pp. 1–4.
2. Giordano, R.; Perrella, S.; Barbieri, D.; Izzo, V. A Radiation-Tolerant, Multigigabit Serial Link Based on FPGAs. *IEEE Trans. Nucl. Sci.* **2020**, *67*, 1852–1860. [CrossRef]
3. Fang, Z.; Yang, C.; Jin, H.; Lou, L.; Tang, K.; Tang, X.; Guo, T.; Wang, W.; Zheng, Y. A Digital-Enhanced Chip-Scale Photoacoustic Sensor System for Blood Core Temperature Monitoring and In Vivo Imaging. *IEEE Trans. Biomed. Circuits Syst.* **2020**, *13*, 1405–1416. [CrossRef] [PubMed]
4. Zhen, W.; Jianhui, J.; Naijin, C.; Guangming, L.; Ying, Z. Effects of three factors under BTI on the soft error rate of integrated circuits. *J. Comput. Res. Dev.* **2018**, *55*, 1108–1116.
5. Zhang, H.; Kochte, M.A.; Schneider, E.; Bauer, L.; Wunderlich, H.J.; Henkel, J. STRAP: Stress-aware placement for aging mitigation in runtime reconfigurable architectures. In Proceedings of the IEEE/ACM International Conference on ComputerAided Design (ICCAD), Austin, TX, USA, 2–6 November 2015; pp. 38–45.
6. Zhang, H.; Bauer, L.; Kochte, M.A.; Schneider, E.; Wunderlich, H.J.; Henkel, J. Aging resilience and fault tolerance in runtime reconfigurable architectures. *IEEE Trans. Comput.* **2017**, *66*, 957–970. [CrossRef]
7. Sahoo, S.S.; Nguyen, T.D.; Veeravalli, B.; Kumar, A. Lifetime-aware design methodology for dynamic partially reconfigurable systems. In Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, Korea, 22–25 January 2018; pp. 393–398.
8. Sahoo, S.S.; Nguyen, T.D.; Veeravalli, B.; Kumar, A. Multi-objective design space exploration for system partitioning of FPGA-based Dynamic Partially Reconfigurable Systems. *Integration* **2019**, *67*, 95–107. [CrossRef]
9. Kourfali, A.; Stroobandt, D. In-circuit fault tolerance for FPGAs using dynamic reconfiguration and virtual overlays. *Microelectron. Reliab.* **2019**, *102*, 113438–113452. [CrossRef]
10. Fricke, F.; Werner, A.; Shahin, K.; Huebner, M. CGRA tool flow for fast run-time reconfiguration. In Proceedings of the ARC 2018: Applied Reconfigurable Computing, Architectures, Tools, and Applications, Santorini, Greece, 2–4 May 2018; Springer International Publishing: Cham, Switzerland, 2018; pp. 661–672.
11. Coole, J.; Stitt, G. Intermediate fabrics: virtual architectures for circuit portability and fast placement and routing. In Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), New York, NY, USA, 24–29 October 2010; pp. 13–22.
12. Stott, E.A.; Wong, J.S.; Sedcole, P.; Cheung, P.Y. Degradation in FPGAs: measurement and modelling. In Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, New York, NY, USA, 21–23 February 2010; pp. 229–238.
13. Ahmed, I.; Zhao, S.; Meijers, J.; Trescases, O.; Betz, V. Automatic bram testing for robust dynamic voltage scaling for fpgas. In Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 27–31 August 2018; pp. 681–687.
14. Xilinx Inc. Vivado Design SuiteUserGuide: Partial Reconfiguration. Available online: https://china.xilinx.com/support/documentation/sw_manuals/xilinx2019/ug909vivadopartialreconfiguration.pdf. (accessed on 20 May 2021).
15. Khaleghi, B.; Omidi, B.; Amrouch, H.; Henkel, J.; Asadi, H. Estimating and Mitigating Aging Effects in Routing Network of FPGAs. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 651–664. [CrossRef]
16. Zhang, H.; Bauer, L.; Kochte, M.A.; Schneider, E.; Braun, C.; Imhof, M.E.; Wunderlich, H.J.; Henkel, J. Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures. In Proceedings of the IEEE International Test Conference (ITC), Anaheim, CA, USA, 6–13 September 2013; pp. 1–10.
17. Ghaderi, Z.; Bozorgzadeh, E. Aging-aware high-level physical planning for reconfigurable systems. In Proceedings of the 21st IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), Macao, China, 25–28 January 2016; pp.631–636.
18. Srinivasan, S.; Krishnan, R.; Mangalagiri, P.; Xie, Y.; Narayanan, V.; Irwin, M.J.; Sarpatwari, K. Toward increasing FPGA lifetime. *IEEE Trans. Dependable Secur. Comput.* **2008**, *5*, 115–127. [CrossRef]
19. Gu, J.; Yin, S.; Wei, S. Stress-aware loops mapping on cgras with considering nbti aging effect. In Proceedings of the 54th ACM/IEEE Design Automation Conference (DAC), Austin, TX, USA, 18–22 June 2017; pp. 1–6.
20. Afzali-Kusha, H.; Akbari, O.; Kamal, M.; Pedram, M. Energy and reliability improvement of voltage-based, clustered, coarse-grain reconfigurable architectures by employing quality-aware mapping. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2018**, *8*, 480–493. [CrossRef]
21. Hu, B.; Shihab, M.; Makris, Y.; Schaefer, B.C.; Sechen, C. An efficient MILP-based aging-aware floorplanner for multicontext coarse-grained runtime reconfigurable FPGAs. In Proceedings of the 2002 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 1526–1531.

22. Kourfali, A.; Kulkarni, A.; Stroobandt, D. SICTA: A superimposed in-circuit fault tolerant architecture for SRAM-based FPGAs. In Proceedings of IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS), Thessaloniki, Greece, 3–5 July 2017; pp. 5–8.
23. Koch, D.; Beckhoff, C.; Lemieux, G.G.F. An efficient FPGA overlay for portable custom instruction set extensions. In Proceedings of the 23rd International Conference on Field programmable Logic and Applications (FPL), Porto, Portugal, 2–4 September 2013; pp. 1–8.
24. Quraishi, M.H.; Tavakoli, E.B.; Ren, F. A Survey of System Architectures and Techniques for FPGA Virtualization. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *9*, 2216–2230. [CrossRef]
25. Grant, D.; Wang, C.; Lemieux, G.G. A CAD framework for Malibu: an FPGA with time multiplexed coarse-grained elements. In Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, New York, NY, USA, 27 February–1 March 2011; pp. 123–132.
26. Heyse, K.; Davidson, T.; Vansteenkiste, E.; Bruneel, K.; Stroobandt, D. Efficient implementation of Virtual Coarse Grained Reconfigurable Arrays on FPGAs. In Proceedings of the 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal, 2–4 September 2013; pp. 1–8.
27. Xiang, Y.; Chantem, T.; Dick, R.P.; Hu, X.S.; Shang, L. System-level reliability modeling for MPSoCs. In Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Scottsdale, AZ, USA, 24–29 October 2010; pp. 297–306.
28. Hara, Y.; Tomiyama, H.; Honda, S.; Takada, H. Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis. *J. Inf. Process.* **2009**, *17*, 242–254. [CrossRef]
29. OpenCores. Available online: https://www.opencores.org (accessed on 8 June 2021).
30. Xilinx. Intellectual Property. Available online: https://www.xilinx.com/products/intellectualproperty.html (accessed on 23 May 2021).
31. Dick, R.P.; Rhodes, D.L.; Wolf, W. TGFF: task graphs for free. In Proceedings of the Sixth International Workshop on Hardware/Software Codesign, Seattle, WA, USA, 18 March 1998; pp. 97–101.