Hindawi
International Journal of Reconfigurable Computing
Volume 2021, Article ID 5512938, 29 pages
https://doi.org/10.1155/2021/5512938

Research Article

A Decision-Making Method Providing Sustainability to FPGA-Based SoCs by Run-Time Structural Adaptation to Mode of Operation, Power Budget, and Die Temperature Variations

Dimple Sharma and Lev Kirischian

Electrical and Computer Engineering Department, Ryerson University, 245 Church Street, Toronto, ON M5B 2K3, Canada

Correspondence should be addressed to Dimple Sharma; dimple.sharma@ryerson.ca

Received 2 March 2021; Revised 19 July 2021; Accepted 13 August 2021; Published 1 September 2021

Academic Editor: Gokhan Memik

One of the growing areas of application of embedded systems in robotics, aerospace, military, etc. is autonomous mobile systems. Usually, such embedded systems have multitask multimodal workloads. These systems must sustain the required performance of their dynamic workloads in presence of varying power budget due to rechargeable power sources, varying die temperature due to varying workloads and/or external temperature, and varying hardware resources due to occurrence of hardware faults. This paper proposes a run-time decision-making method, called Decision Space Explorer, for FPGA-based Systems-on-Chip (SoCs) to support changing workload requirements while simultaneously mitigating unpredictable variations in power budget, die temperature, and hardware resource constraints. It is based on the concept of Run-Time Structural Adaptation (RTSA); whenever there is a change in a system's set of constraints, Explorer selects a suitable hardware processing circuit for each active task at an appropriate operating frequency such that all the constraints are satisfied. Explorer has been experimentally deployed on the ARM Cortex-A9 core of Xilinx Zynq XC7Z020 SoC. Its worst-case decision-making time for different scenarios ranges from tens to hundreds of microseconds. Explorer is thus suitable for enabling RTSA in systems where specifications of multiple objectives must be maintained simultaneously, making them self-sustainable.

1. Introduction

As the famous proverb goes, "necessity is the mother of invention," considering the aspect of "necessity," over the past few decades, human necessity has significantly increased in every walk of life. There is a necessity to delegate much of the human workload to autonomous and mobile systems, from routine chores to industrial robotics, to aerospace, to defense, and to many other areas where presence of humans is not efficient, or is unsafe or even dangerous for humans. The requirement is not just delegation of workload, it is delegation with the assurance of system reliability and workload security, and an endeavor to carry out functions and reach places beyond human limits. A few examples of the said "necessity" are need for self-driven cars and driver assistance systems, robotic systems for flexible manufacturing, civil and military drones, different types of satellites and autonomous planetary mission spacecraft, unmanned submarine systems, etc. A common aspect in all these fields of applications is high complexity. All the applications require continuous processing of several high-performance data-stream workloads such as video-streams, communication data-streams, LiDAR or RADAR data-streams, and acoustical and audio-streams, which usually have strict performance constraints. The requirement does not end at being multitask applications. Since most of these applications require systems to be autonomous and mobile, they need to cater to different situations and scenarios. This means that the systems may need to change the number of active tasks, their functions, priorities, and/or performance specifications due to any scheduled/sudden external/internal event or interrupt. Thus, we are looking at the need to have autonomous and mobile systems that can support complex, high-performance, multitask multimodal applications!

Considering the latter part of the saying, "necessity is the mother of invention," there has been a great technological advancement in the recent past and is still progressing to support the growing needs and complexity of applications. Programmable Logic Devices (PLDs), especially Field Programmable Gate Arrays (FPGAs), have progressed significantly in terms of the number and types of resources, and the capabilities being integrated in them. A recent advancement is System-on-Programmable-Chip (SoPC) devices, where instruction-based processors and reconfigurable logic (FPGA) are integrated together on a single die. As the name SoPC suggests, entire systems can be built on a single chip; algorithmically intensive tasks can be mapped on the sequential processors and time critical/high-speed tasks can be mapped on the reconfigurable logic. The most important feature that makes these devices capable of supporting today's applications and even future applications is Dynamic Partial Reconfiguration (DPR). DPR allows configuring and reconfiguring dynamically, portions of the reconfigurable region of the SoPC/FPGA die, called Partially Reconfigurable Regions (PRRs). With this feature, tasks can be programmed and reprogrammed while other tasks are functioning. Thus, SoPC devices are suitable platforms to develop complex multitask multimodal applications.

Although SoPC devices are the choice to support today's applications, they face a set of challenges when used to develop autonomous and mobile systems operating in different environmental conditions. If the requirements of multitask multimodal applications are static, i.e., the tasks, their modes, and events for switching modes are well defined, the complexity of system development boils down to management, i.e., mapping and scheduling of tasks in a given static set of power budget, hardware resource, and temperature conditions. A lot of research efforts have been observed in this area. Several Real-Time Operating Systems (RTOSs), mapping and scheduling algorithms, optimization algorithms to minimize power/die temperature and maximize performance, etc. have been developed, and the functionality of which depends on the implementation platform and application being supported. This is discussed in detail in Section 2. However, when autonomous and mobile systems are considered, they face a different set of challenges where research has not yet dug deep. The requirements of their multitask multimodal workloads are dynamic, i.e., events for switching modes are not always predictable, and so are the other conditions such as available hardware resources, power budget, and external/on-chip temperature. For example, an Advanced Driver Assistance System (ADAS) may need to dynamically carry out tasks related to collision avoidance at a high performance if there is a sudden possibility of collision with an object or pedestrian; otherwise, it may carry out at a reduced performance or may not execute them at all. This means the system's mode, i.e., number of tasks and their performance specifications can change dynamically depending on several unpredictable events. In terms of hardware resources, there is a static limit on the total amount of resources available on the FPGA/SoPC device chosen depending on the area, mass, or weight specifications of the system. The resources available for a task to function at a certain time can also dynamically vary depending on resources occupied by other executing tasks at that time and/or the occurrence or restoration of hardware faults caused due to radiation, thermal cycling effects, aging, etc. Similarly, although the system has a fixed maximum power budget, the available budget at a certain time can vary dynamically depending on factors such as the set of tasks functioning at that time, reduction of fuel or available solar power, and fault of power generator or solar panel(s) or other sources of power. There can be changes in the external and on-chip temperatures resulting in dynamic variations in the difference between the two temperatures. This can result in thermal stress, which may cause hardware faults and affect system reliability. Thus, the challenge faced by autonomous and mobile systems is that they must be capable of sustaining the performance requirements of their dynamic multitask multimodal workload while simultaneously adapting to the dynamic changes in the power budget, die temperature, and hardware resource conditions. All of this must be catered to in run-time, while the system is running, and every time there is a change in the said requirements. When a system can sustain itself to these multiobjective requirements, it can continue functioning. A failure to meet the requirements could cause system shutdown, which could affect other dependent systems too.

The above aspects and associated problems are also common for all animals which can be considered as self-regulating natural mobile and autonomous systems. The similarity of self-regulating processes in the animals and in the machines was first pointed in the classical Norbert Wiener's book "Cybernetics: or Control and Communication in the animal and in the Machine" [1]. The evolution of wild life has found the mechanism for mitigation of environmental changes and stress for animals, which is their ability for run-time and overgeneration adaptation. The same concept can be used for autonomous self-controlled machines working in the real environment. There are three types of adaptation: behavioral, parametric, and structural. Behavioral and parametric adaptations can be implemented by changing the set of active tasks and/or their modes of operation. However, the behavioral and parametric adaptations would be sufficient for systems only in cases when performance, power consumption, thermal conditions, etc. are either not constrained or have fixed limits. When the set of constraints themselves is dynamic, these types of adaptations are not always sufficient. Changes in the set of tasks and/or their modes of operation can affect performance and response time, increase power consumption, and even reduce reliability of the system. Also, external factors such as radiation, overheating, or overcooling can cause faults in parts of the system or the entire system. In the case of systems on long missions, internal factors such as aging of electronic circuits and degradation of elements in power supplies also can cause system faults. Thus, in the case of self-dependent autonomous mobile systems, their own dynamic workload and the external and internal factors together cause system constraints to be dynamic. The only mechanism that can mitigate all these factors is structural

adaptation of the system. Structural adaptation of a system is based on changes in the system's functional elements and interconnections between them. It requires reconfiguration of the functional elements and their associated interconnects. This process of structural reconfiguration needs to be carried out dynamically, whenever system constraints change, and in a time dictated by the mitigation time limit. If the structural adaptation time exceeds this limit, the system or its part(s) can permanently fail. Therefore, Run-Time Structural Adaptation (RTSA) is required. RTSA consists of two time periods: (a) period of decision-making on what should be the system's new configuration and (b) reconfiguration time itself. Reconfiguration time mostly depends on the component base (e.g., type of FPGA device and method of bitstream loading). Thus, the focus of this research is on the decision-making process for RTSA and the time overhead associated with this process. It is obvious that structural adaptation follows the system's mode of operation (e.g., set of required tasks and their current mode) and the current set of constraints. In other words, RTSA follows the behavioral and parametric adaptation.

Thus, RTSA is the solution to the above-discussed problem [2–4]. With RTSA, systems can change their task structures to satisfy the changing set of constraints. In [5], it has been shown that, for a given task algorithm, several implementation variants can be obtained. This means each task can be implemented in the form of different dedicated hardware circuits, which exhibit different resource utilization, operating frequency, performance, and power consumption characteristics. Implementation variants of a task can be referred to as Application-Specific Processing circuit (ASP circuit) variants of that task. A task with a specific performance can have ASP circuit variants with different combinations of frequency and resource utilization, and hence different power consumption, to provide the same performance. If the application specifications permit, a task can have a range of allowed performance specifications, instead of a fixed performance constraint. In such a case, the task also has ASP circuit variants with different combinations of frequency and resource utilization, and hence different power consumption, to provide different performance outcomes within the allowed limits. All the ASP circuit variants of each task are stored in the form of partial configuration bit-files in system memory. Thus, with the availability of different ASP circuit variants, a suitable variant for each active system task can be configured based on the existing set of constraints, so that all the system requirements are satisfied. This set of suitable variants of active tasks is called a system configuration. For example, assuming the availability of spare hardware resources, in a low power budget scenario, ASP circuit variants for tasks can be reconfigured such that they occupy more hardware resources and operate at a reduced frequency to reduce power consumption. Alternatively, if the available number of resources reduces due to occurrence of a hardware fault, tasks can operate at a higher frequency while occupying a smaller area such that the faulty region can be avoided. In either case, the performance specifications of tasks are maintained. Thus, with this flexibility of form, a system can efficiently adapt to all its dynamically varying constraints.

Although RTSA is the solution to the problem at hand, the issue here is as follows: field-deployed mission critical multitask multimodal systems are complex systems with multiple tasks and their modes, where each task is implemented as an ASP circuit. To enable such systems with RTSA, each task has several ASP circuit variants. Since the number of tasks and their modes of operation can be quite large, from tens to hundreds, the decision space to be explored for RTSA can be very large. For example, a system with a total of 64 tasks, 64 ASP circuit variants per task, 25 modes, and 20 tasks per mode will have a decision space of $\prod_{n=0}^{19} 64 = 64^{20}$ system configurations per mode. The total number of system configurations on the decision space for all possible modes will be 25 modes $\times\ 64^{20}$ system configurations per mode $= 25 \times 2^{120}$ system configurations! This decision space is too large for it to be stored as a set of configuration bit-files in the associated memory subsystem. Also, carrying out RTSA for such systems means that, if there is any dynamic change in the system's set of constraints, one suitable system configuration will need to be selected in run-time and within a small permitted adaptation time which could be in units of seconds or even less, from this tremendously large decision space of system configurations such that all the constraints at that time are satisfied. The system will also need to repeat this procedure every time there is a change in its set of constraints, which is mostly unpredictable as discussed above. Certainly, it is not possible to exhaustively evaluate each configuration at run-time to find a solution that satisfies a multiobjective constraint set. Therefore, a run-time multiobjective Decision Space Exploration method must be used which can solve the issue of memory requirement for storing a tremendously large decision space and the issue of identifying a suitable system configuration for adaptation from the decision space within a time permissible at run-time.

In most cases, multiobjective decision space exploration is considered in high-level synthesis of digital system architectures. Thus, it is represented in the form of Design Space Exploration (DSE), a design-time process applied when a system structure needs to be selected from multiple solutions in the system-design phase [6, 7]. Since these methods are used to find this optimum architecture which remains fixed for the entire lifetime of the system, they need to be accurate for this one-time process and do not require finding the optimum architecture in run-time. They are, therefore, iterative and involve detailed evaluation of a large number of candidate solutions [6, 8–15]. As a result, they have large exploration times. Although this is acceptable for a design-time process, methods with large exploration times cannot be adopted for use in real-time scenarios. Thus, there is a need to devise a method, which can drastically reduce the set of variants on the decision space thus reducing the exploration time and making it acceptable for run-time adaptation. Such a method should be able to select the best system configuration that satisfies the multiple constraints whenever there is a change in the set of requirements.

The initial approaches for creating the run-time decision-making method for RTSA have been presented in [2, 3].

These methods allow run-time adaptation to system workload variations and variations in power budget considering variations in hardware resource constraints due to hardware faults. They are, however, limited by not considering one external factor. They do not consider simultaneous mitigation of off-chip and on-chip thermal factors and thermal cycling along with other system constraints. Thermal cycling is one of the common factors causing hardware faults in flip-chip technology-based FPGAs nowadays. Also, the SoPC die has faster thermal dynamics as compared to the off-chip thermal dynamics, i.e., FPGA package connected to the power dissipation units (e.g., heat sink and board layers) [4]. This can result in thermal instability of the SoPCs, thus affecting system reliability. Additionally, functionality of the task circuits can also be affected if the die temperature exceeds certain specifications. Thus, it is necessary to maintain the die temperature in the desired range while simultaneously adapting to changes in workload, power budget, and hardware resource conditions. This aspect of simultaneous thermal regulation is not considered in the methods presented in [2, 3]. Also, these methods assume that all the tasks function on a common system frequency. However, in practical applications, tasks can run at different individual frequencies. The ability to operate on different frequencies is also a benefit for RTSA. It becomes possible to change the frequency of a required individual task alone for RTSA instead of changing the common frequency for all the tasks and finding ASP circuit variants for each task at that frequency.

This paper proposes a run-time decision-making method for SoPCs deployed on the partially reconfigurable FPGA devices using flip-chip technology. This method is called Decision Space Explorer (referred as Explorer in the rest of the paper) to carry out RTSA to mitigate dynamic changes in any multitask multimodal workload, power budget, FPGA die temperature variations and thermal cycling, and available hardware resources. The novelty of the proposed method is that it incorporates simultaneous mitigation of several interdependable factors (e.g., thermal conditions and power budget) while allowing individual tasks of the workload to run on their independent frequencies. Whenever there is a change in the system's aforementioned set of constraints, Explorer, the run-time decision space exploration method, explores, evaluates, and selects ASP circuit variants for each task at their individual operating frequencies such that the resultant system configuration fits in the available hardware resources, the critical tasks of the system's required mode operate at their required performance, the other noncritical tasks operate within the permitted performance specifications, the system's power consumption is lower than the power budget, and its die temperature is maintained within the permitted range! Explorer achieves this by selecting an ASP circuit variant for every task individually instead of selecting a system configuration as a whole. Therefore, characteristics of only ASP circuit variants of individual tasks need to be stored and not of entire system configurations. This approach translates the decision space of product of the decision space of individual tasks, into a linear decision space of sum of the decision space of individual tasks. For the example system considered with 64 tasks and 64 ASP circuit variants each, the total decision space will be $\sum_{n=0}^{63} 64 = 64 \times 64 = 4096$ only! The decision space is reduced by a factor of 25×2^{108}! This tremendous reduction in the decision space drastically reduces the system's memory requirements to store the configuration bitstreams; only 4096 configuration bit-files need to be stored. Furthermore, since tasks are allowed to operate at their respective frequencies, ASP circuit variants of only some tasks of a mode need to be explored for RTSA. This further reduces the exploration time within the reduced decision space, making the proposed method a proper fit for run-time multiobjective structural adaptation.

Along with the proposed decision-making approach, there is another aspect of the method that can be considered as novel. This is the mechanism of evaluating potential configurations in run-time. It uses mathematical models that can predict the power consumption and die temperature for the set of ASP circuits, i.e., system configuration under test. The model coefficients are dynamic in nature; they are self-calibrated by the system whenever there are changes in (a) the SoC platform (e.g., FPGA type and package and PCB layers to which FPGA is connected), (b) the application being supported (e.g., ASP circuit specifics such as utilization of CLB-slices, Block RAM, and DSP-modules), (c) environmental changes (e.g., external temperature in system compartment), or (d) aging (e.g., FPGA die, BGA connections due to oxidizing, and vibration/acceleration). In other words, multiobjective RTSA is based on self-adaptation of evaluation models to multiple factors which influence the performance of the SoPC deployed on a particular FPGA platform. The proposed approach is based on direct measurements of power consumption and on-chip temperature using the temperature sensor and hard-core Analog-to-Digital Converter(s) (ADC) embedded in the FPGA device. Thus, the model coefficients are always accurate and up-to-date for the system. Due to this, the system configuration chosen by Explorer for RTSA can be considered as an accurate and suitable solution. For example, if a field-deployed autonomous system experiences cold weather conditions instead of a warm climate due to season change, it changes the behavior of the SoC's power consumption and die temperature. This means the power consumption of a particular system configuration and the resultant die temperature will be different than it was in a warmer environmental condition. The system self-calibrates its model coefficients such that they correspond to the current environmental scenario. When Explorer uses these updated coefficients to predict the power consumption and die temperature of candidate configurations during evaluation for RTSA, the selected solution is accurate. If the model coefficients remained static and were not updated, Explorer's solution would be according to the static set of coefficients. Thus, the solution's accuracy would become dependent on the difference between the static coefficients used and the actual coefficients which should have been used. Thus, use of dynamic models to predict power consumption and die temperature helps Explorer to always provide accurate solutions for RTSA.

Also, since the proposed method deals with mitigation of both, die temperature and power consumption, which are interrelated [16], there can be situations where both can be satisfied, or when the requirements are contradicting each other. Complex processing is required to decide whether both can be satisfied or one of the two needs to be prioritized during RTSA, depending on different scenarios. Explorer includes this complexity in its decision-making process. Experimental implementation of Explorer on the ARM Cortex-A9 core of the Xilinx Zynq ZX7Z020 device shows that the execution time is in the order of microseconds. Thus, in the knowledge of the authors, the proposed decision-making method is the only method that can enable multiobjective decision space exploration in the range of microseconds. Therefore, it allows structural adaptation to dynamic changes in multiple interrelated constraints, in run-time, for FPGA-based multitask multimodal systems, thus making them self-sustainable!

The paper has the following structure: Section 2 discusses the present-day power/thermal/fault management methods and design space exploration methods used for embedded systems. It analyzes them from the perspective of multiobjective RTSA. Section 3 provides an overview on the static infrastructure required to efficiently carry out RTSA in systems, which has been detailed in [17–19]. Section 4 presents an overview on the run-time power consumption and die temperature estimation models for FPGA-based devices, which can be derived using methods presented in [3, 4, 16, 20]. These models are used by Explorer to evaluate potential system configurations while exploring solutions for RTSA. Section 5 describes the system characteristics/parameters that are required for Explorer to function properly. Section 6 presents details of the different decision paths that Explorer follows under different cases in order to find a suitable system configuration that satisfies the set of system constraints at that time. Section 7 demonstrates how Explorer performs its run-time decision-making to select the appropriate solution for RTSA, using different example scenarios. Section 8 presents an analysis of the worst-case execution time of Explorer for the example considered in Section 7 and validates its suitability for RTSA. Section 9 concludes the paper.

2. Literature Review

Since the focus of this paper is on multiobjective run-time structural adaptation, the related research publications have been observed from two perspectives:

(a) Existing methods for power/thermal/fault management in embedded systems: this part of the study observes and analyzes why the currently used techniques cannot directly be applied to achieve run-time adaptation to multiple objectives, namely, dynamic workload, power budget, die temperature, and hardware resource constraints, in multitask multimodal systems.

(b) Existing design space exploration methods for embedded systems: the present-day research efforts in the area of decision-making methods for structural adaptation are mostly associated with DSE for SoC architecture and optimization. As discussed in the Introduction section, RTSA is the solution for FPGA-based multitask multimodal systems to sustain themselves against multiple changing constraints. This part of the literature study observes and analyzes why currently used methods are difficult to adopt to achieve RTSA for the said class of systems.

A look into the present-day research efforts shows that there are several methods used for run-time adaptation to different parameters. Methods such as power gating [21, 22], Dynamic Voltage and Frequency Scaling (DVFS) and DFS [23–26], dynamic scheduling techniques [25, 27–30] and dynamic mapping techniques [25, 31–33], and dynamic task migration are used for power and/or thermal aware workload management. Methods such as Triple Modular Redundancy (TMR) and scrubbing [34–36], Built-in Self-Test (BIST) procedures [37], device reprogramming to avoid damaged regions [38], dynamic scheduling and mapping [39–41], run-time relocation based methods [42, 43], and variant-based methods [44–47] are commonly used methods for fault management. Many systems deployed on FPGAs/SoPCs make use of RTOS or similar management systems to adapt to different dynamic system parameters. The basic functions of these RTOSs have been outlined in the literature; they carry out task scheduling, task mapping and allocation, intertask communication, task to RTOS communication, task configuration, etc. [48, 49]. R3TOS [39, 50, 51], BORPH [52], CAP-OS [53], ReConOS [54], Operating System for Reconfigurable Systems (OS4RS) [55], and references [39, 54, 56–61] are some examples of RTOSs or management methods that cater to different types of systems, those which support hardware or software tasks alone and those which include both. They carry out the same functions outlined in the literature; their extent and complexity depends on the system structure and the type of workload being supported.

The following can be found from the different adaptation methods observed above:

(a) Most RTOSs and management methods are directed towards instruction-based processor-centric systems; for SoCs, these methods focus on the hard- or soft-core processors associated with the SoCs. These methods therefore depend on the nature of the tasks which have their specific deadlines, worst-case execution time, slack times, etc. Hence, they are well suited for algorithmically intensive tasks performed by sequential processors. Considering autonomous and mobile systems, which are the focus of this paper, they process multiple multimodal data-stream processing tasks, which are implemented as dedicated hardware circuits. Since these tasks are continuously processing incoming data streams, their characteristics are not similar to the algorithmically intensive tasks. Therefore, it is difficult to use the above observed methods to manage the type of tasks considered here.

(b) The observed methods cannot enable run-time adaptation to all the parameters: workload performance, power, thermal, and resource constraints. They carry out workload management to satisfy only one or two parameters. This is because the effort is to optimize a single parameter keeping all the other constraints fixed. For example, the system's power consumption is minimized under fixed task-performance and resource constraints. Adaptation to multiple dynamic parameters requires varying the parameter to be prioritized based on need. This means that, if the system's power budget is low, its power consumption must be reduced in accordance with the new power budget by adjusting the performance of the tasks accordingly. This reduced power consumption needs not be the system's minimum power consumption. It must be just below the new constraint such that the other system parameters are also satisfied. Alternatively, if executing a set of tasks is imperative, task performance is prioritized. In this case, performance of noncritical tasks is adjusted such that performance of the critical tasks remains at the required value and the overall power consumption of the system and die temperature are within the set limits at that time. Since the observed methods are mainly single parameter optimization methods, they cannot achieve a balanced adaptation to multiple dynamic parameters.here.

(c) The observed run-time adaptation methods manage tasks with fixed implementation circuits. In other words, only a procedural way of adaptation is available by rescheduling the tasks (flexibility of time) and reallocating/remapping them on different available processing, memory, and communication resources (flexibility of space). With this, it may not always be possible to satisfy dynamically varying environmental constraints. For example, the minimum power consumption achieved by rescheduling the tasks may satisfy a fixed power budget constraint. However, when the power budget constraint itself is dynamic, the obtained minimum power consumption may be higher than a new reduced power budget of the system. To be able to satisfy multiple dynamic system and environmental parameters, there must a higher level of flexibility. To achieve this, tasks must have different implementation variants, i.e., ASP circuit variants (flexibility of form). In turn, this dictates the need for RTSA, as discussed in the Introduction section. Since the existing methods lack this flexibility of form, they cannot be directly applied for run-time multiobjective adaptation.

The idea of using different implementation versions of tasks for run-time adaptation is gaining consideration, and some research work can be seen in this direction. The use of task implementation variants is observed for systems supporting software tasks [62–64], hardware tasks [65–67], and those supporting both, software and hardware tasks [68]. However, these methods target adaptation to individual parameters and not multiobjective adaptation process. Also, for most methods, the number of possible scenarios to adapt to which the different variants are used is not many and is predefined or predictable. Therefore, these methods do not need to consider development of associated decision-making mechanisms due to the simplified process of variant selection. Thus, although these methods use the concept of variants, they cannot be applied for run-time adaptation to multiple dynamic unpredictable constraints.

It is understood from the above observation of the literature and the Introduction section that RTSA seems to be the potential solution for FPGA-based systems to be self-sustainable against (a) multitask and multimodal workloads with unpredictable combination of tasks activated for parallel execution, (b) unpredictable variations of external to SoC and environmental factors, and (c) unpredictable variation of hardware resource constraints caused by transient or permanent hardware faults. To practically deploy RTSA, a run-time decision-making method is required which can select a suitable configuration at run-time that satisfies the system's set of constraints whenever there is change in any of the constraints. The currently adopted design space exploration methods are observed from this perspective. Heuristic methods, evolutionary algorithms, or their combinations are mostly adopted for design space exploration and multiobjective optimization in the domain of embedded system design. In the case of embedded systems, heuristic methods have mostly been applied at system level for design of processor systems, memory subsystems, SoC/MPSoC platforms, etc. Pareto Simulated Annealing (PSA) [8], sensitivity-based local search multiobjective design space exploration [69], Discrete Particle Swarm Optimization (DPSO) [9], and Ant Colony Optimization [10, 70–72] are some examples of such heuristic methods. Evolutionary algorithms (EAs) are commonly used in component-level and system-level design for embedded systems. They make use of EAs such as Greedy Evolutionary Multiobjective Optimization (GEMO) [73], Strength Pareto Evolutionary Algorithm-2 (SPEA2) [11, 12], Genetic Algorithm (GA) [74–77], and Nondominated Sorting Genetic Algorithm II (NSGA-II) [13–15]. Use of methods involving Design of Experiments (DoE) and Resource Surface Modeling (RSM) is also observed for design space exploration [78, 79].

From the observed design space exploration methods for embedded systems, it can be analyzed that they are mostly design-time processes and are carried out while choosing/designing a suitable system architecture based on design specifications. Since the heuristic methods and evolutionary algorithms are used to find an optimum architectural solution that will last for the entire lifetime of the system, they ensure that the process is extremely accurate. They, therefore, involve a large number of iterations and detailed evaluation of multiple configurations in every iteration for every system objective, which may take seconds to many hours per configuration depending on the application. Thus, they have very large exploration times ranging from hours to days. DoE- and RSM-based methods also involve iterative learning of models while generating optimal solutions. As a

result, use of actual synthesis or detailed simulations is inevitable for these methods, making them unsuitable to be adopted for run-time decision space exploration.

Thus, from a review of the literature, it can be seen that in general, there is a lack of methods which can enable RTSA in autonomous and mobile systems. For RTSA, a noniterative method is required, which can explore and evaluate a very large decision space against multiple objectives to select a suitable solution within the mode-switching time available with the system, in the order of units of seconds or even less, such that it closely satisfies all the system constraints. It must be able to do so every time the system requires to adapt due to the changing set of constraints, i.e., changing workload, power, thermal, or hardware resource constraints. Thus, there is a need to formulate a decision-making method which achieves this aforementioned goal so that systems can carry out RTSA to unpredictable changes in multiple internal and external factors, whenever the need arises, and thus can be self-sustainable.

This paper proposes a novel run-time decision-making method, called Decision Space Explorer, which meets the said requirements. With the approach used by the method, the huge decision space of system configurations to be explored is tremendously reduced. Whenever there is a change in the system's set of constraints, Explorer scans through a small number of candidate configurations from the already reduced decision space and selects an appropriate configuration that satisfies the system's mode, power budget, die temperature, and available hardware resource conditions, all within a time frame permissible for RTSA. Explorer uses mathematical models to predict the power consumption and die temperature for the candidate configurations in run-time. The models are dynamic in nature; the coefficients get self-updated whenever required; in situations such as system updates, changes in environmental conditions, and aging. This allows Explorer to select an accurate solution for RTSA at any given time on any partially reconfigurable FPGA-based hardware platform.

3. MACROS Framework

A system capable of RTSA must have an underlying architecture that supports the RTSA process. It must allow a system to dynamically change its configuration within a time span allowed by the application running on the system, usually within units of seconds or even less. Such a framework, called the "Multimode Adaptive Collaborative Reconfigurable self-Organized System" (MACROS) framework, has been developed, and details of which can be found in [17–19]. This section provides a brief overview on the MACROS framework.

The basic architecture of MACROS framework is shown in Figure 1. It consists of three main parts: (a) several identical PRRs called slots on the FPGA, (b) a Distributed Communication and Control Infrastructure (DCCI), and (c) a Bitstream and Configuration Management system (BCM). Partial bitstreams that can be configured in the PRRs/slots of the FPGA are referred as Collaborative Macro-Functional Units (CMFUs). A CMFU consists of two parts: ASP circuit/component of a task, and a control circuit called Co-Op unit. The portion of a task's ASP circuit that occupies a PRR is called an ASP component. The Co-Op unit is responsible for control and communication on behalf of the ASP circuit/component it is associated with. The CMFUs can be static or dynamic. Static CMFUs are always needed by the SoPC because these CMFUs provide application-specific interface to the sensors, actuators, external memory modules, etc. In contrast to static CMFU(s), dynamic CMFUs correspond to the tasks that form the current system mode. The DCCI, in essence a crossbar, implements the on-chip communication and control capabilities. The communication ports of the DCCI, which connect to one slot each, have control circuits called Local Connection Control Units (LCCUs). These LCCUs communicate with the Co-Op units of CMFUs to establish the needed communication links and synchronization. Finally, the BCM is responsible for all configuration activities in the system; it contains interfaces to bitstream storage memories and one of the FPGA configuration interfaces. With the help of this architecture, components of an ASP circuit can self-integrate to form a functional task and the different tasks can self-integrate to form a functional system mode. The BCM deploys the required CMFUs in the system through DPR. Individual CMFUs, with the help of their Co-Op units and the corresponding LCCUs in the DCCI, manage their own connectivity and synchronization with the CMFUs of the same or other tasks. The CMFUs can decide for themselves when they need to be connected to/disconnected from the system and when their activity needs to be initiated or terminated. All this information is hard-coded in the Co-Op units of the CMFUs at design time.

Thus, the process of RTSA becomes seamless when the static MACROS framework is deployed on the FPGA, and the ASP circuit variants of all the tasks are appropriately packed into CMFUs/partial bitstreams at design time. At the time of run-time adaption, the system only needs to select the appropriate variants of active system tasks, and the rest is taken care of by the MACROS framework.

4. Dynamic Run-Time Power Consumption and Die Temperature Estimation Models

To carry out multiobjective RTSA, Explorer needs to evaluate potential system configurations to choose the appropriate one that closely satisfies all the constraints. This means that Explorer needs to know how much power a system configuration would consume and how it will affect the temperature when it is configured on the SoPC die. One possible solution is to store the power consumption and die temperature for all the possible system configurations in a Look-Up-Table (LUT) during system-design phase. Using the same example of the system having a total of 64 tasks, 64 variants per task, and 25 modes and 20 tasks per mode, this would mean measuring the power consumption and die temperature of 25 modes $\times$ 64^{20} configurations per mode during system-design phase and feeding these values in a very large LUT. If there are changes in the functionality of some tasks, addition of some tasks, changes in the hardware platform, or in environmental conditions, the values of power consumption and die

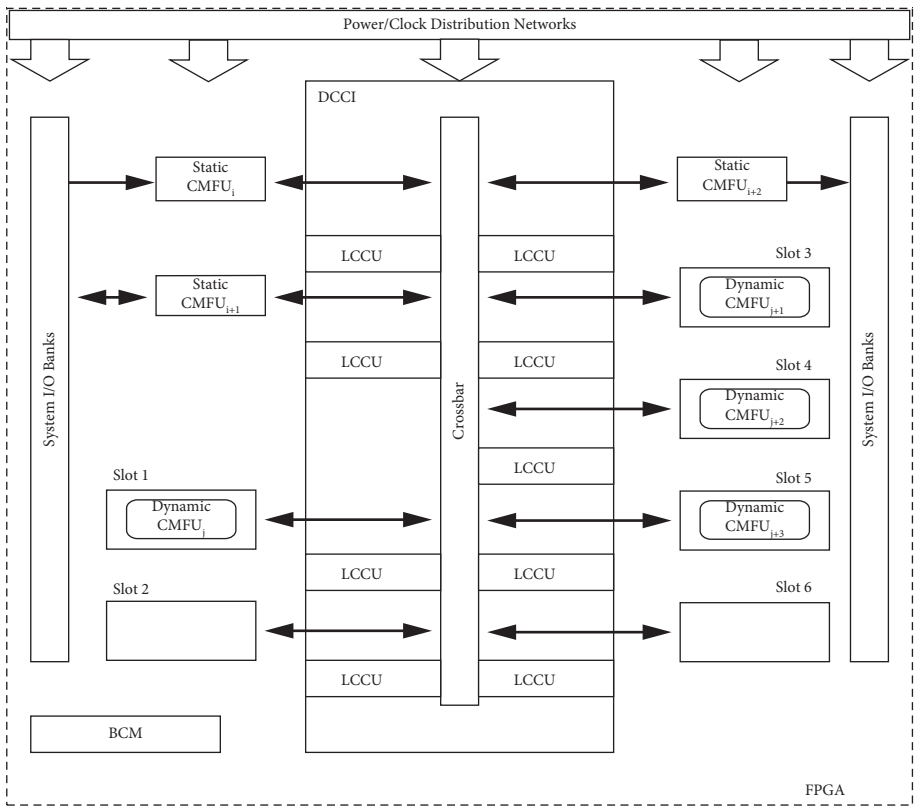

Figure 1: Architecture of MACROS framework.

temperature stored in the LUT will not hold true. This means that the LUT will need to be updated; that is, the power consumption and die temperature for all the system configurations will need to be measured again and stored in the LUT. It is impractical to store a LUT of such a large size and also repeatedly update the LUT every time there is a change in some internal or external factor. Thus, measuring and storing the power consumption and die temperature of all the possible system configurations in LUTs is not a feasible solution. It is necessary to have dynamic analytical models which can estimate the power consumption and die temperature of system configurations under evaluation at run-time. The models need to be simple enough to allow prediction in a run-time scenario and accurate enough to serve the purpose of adaptation. The model coefficients are dynamic; the system updates/calibrates these coefficients whenever the need arises. This way, all the possible unpredictable changes are taken care of. Thus, dynamic run-time models help accurate evaluation of candidate configurations under consideration so that an accurate solution can be chosen for RTSA.

It has been demonstrated in [4, 16] that while modeling the TPC and DT of a FPGA for a configuration, it is the Saturated TPC (STPC) and Saturated DT (SDT) that need to be modeled. The method to derive the SDT Estimation Model (SDTEM) and the STPC Estimation Model (STPCEM) for a FPGA/SoC device is presented in [16]. The STPCEM estimates the STPC of the FPGA/SoC under consideration with the help of the Dynamic Power Consumption Estimation Model (DPCEM). The SDTEM in turn estimates the SDT in terms of the STPC of the system. The method to derive the DPCEM is presented in [3, 20], which results in a linear equation that estimates the DPC of the FPGA/SoC in terms of the operating frequency of the tasks and the reconfigurable resources of the FPGA, namely, Logic, BRAM, and DSP slices [80, 81] used by the tasks. The following equations summarize how the DPC, STPC, and SDT of a FPGA can be estimated at run-time for a system configuration of multiple tasks running at different frequencies with the models derived using the methods presented in [3, 16, 20].

Equation (13) in [3] represents the DPCEM of a FPGA that estimates the DPC for a system configuration with N_c tasks such that all the tasks operate at the same frequency at a point of time and have the same model coefficients. It is as follows:

$$\mathrm{DPC}_{(\mathrm{FPGA})}\,(\mathrm{mW}) = \frac{F_{\mathrm{cc}}}{F_{\mathrm{min}}} \times \left(C_{\mathrm{LS}} \times \sum_{n=0}^{N_c-1} N_{\mathrm{LS}} + C_{\mathrm{BS}} \times \sum_{n=0}^{N_c-1} N_{\mathrm{BS}} + C_{\mathrm{DS}} \times \sum_{n=0}^{N_c-1} N_{\mathrm{DS}} + C_F \right), \tag{1}$$

where F_{cc} is the operating frequency, F_{min} is the minimum operating frequency of the tasks, and C_{LS}, C_{BS}, and C_{DS} are the coefficients relating DPC to the number of Logic slices N_{LS}, BRAM slices N_{BS}, and DSP slices N_{DS}, respectively. C_F is the frequency-dependent constant that represents the DPC of the total IOBs used by the tasks.

In equation (1), since all the tasks are operating at the same frequency and have the same model coefficients, the resource utilization of the system configuration is the sum of the resource utilization of the ASP circuit variants of the individual tasks. However, in general, in multitask applications, individual tasks of a configuration can operate at different frequencies and they may or may not have the same model coefficients. Due to this, resource utilization of individual tasks cannot be summed up together. Therefore, the DPC of the system configuration with N_c tasks operating at different frequencies is obtained as

$$\mathrm{DPC}_{(\mathrm{FPGA})}(\mathrm{mW}) = \mathrm{DPC}_{(\mathrm{Task}_1)} + \mathrm{DPC}_{(\mathrm{Task}_2)} + \cdots + \mathrm{DPC}_{(\mathrm{Task}_{N_c})}, \tag{2}$$

where the DPC of each task can be obtained using equation (7) of [3] as

$$\mathrm{DPC}_{(\mathrm{Task}_n)}(\mathrm{mW}) = \frac{F_{ccn}}{F_{\min}} \times \left(C_{LSn} \times N_{LSn} + C_{BSn} \times N_{BSn} + C_{DSn} \times N_{DSn} + C_{Fn}\right), \tag{3}$$

where F_{ccn} is the operating frequency of Task_n and C_{LSn}, C_{BSn}, and C_{DSn} are the coefficients relating DPC of the task to the number of Logic slices N_{LSn}, BRAM slices N_{BSn}, and DSP slices N_{DSn}, respectively used by Task_n. C_{Fn} is the frequency-dependent constant that represents the DPC of the IOBs used by Task_n.

Thus, equations (2) and (3) together represent the DPCEM of a FPGA for a system configuration with multiple tasks operating at different frequencies. Once DPC is estimated using this DPCEM, it can be incorporated in the STPCEM equation; that is, equations (4) and (5) of [16] combined together, to estimate the STPC of the system configuration, shown as follows:

$$\mathrm{STPC}_{(\mathrm{FPGA})}(\mathrm{W}) = \mathrm{SPC}(\mathrm{W}) + \mathrm{SPP}(\mathrm{W}) + \mathrm{DPC}(\mathrm{W}) + R(\mathrm{W}), \tag{4}$$

where SPC is the system's static power consumption, SPP is the power consumption of the hard-core processor on the SoPC if any, and R is the rise in TPC to reach STPC. SPC, SPP, and R are one-time offline measurements, and the DPC is estimated using the DPCEM; that is, equations (2) and (3) combined together.

Once STPC is estimated using this STPCEM, it can be incorporated in the SDTEM equation, i.e., equation (7) of [16] to estimate the SDT of the system configuration as follows:

$$\mathrm{SDT}(^{\circ}\mathrm{C}) = M \times \mathrm{STPC}(\mathrm{W}) + C, \tag{5}$$

where M is the slope which relates the SDT and STPC, and C is a constant. STPC can be estimated from equation (4).

It can be seen that the DPCEM, STPCEM, and SDTEM are linear equations. Explorer can, therefore, estimate the STPC and SDT of the FPGA/SoC device for candidate system configurations with a very small execution time, which is apt for run-time evaluation of possible solutions and selection of an appropriate one that closely satisfies all the system constraints.

5. System Description for Explorer Functioning

With the MACROS framework and the run-time STPC and SDT estimation models in place, this section describes the system characteristics/parameters that are required for proper functioning of Explorer. Let the system have a total of N tasks, denoted as T_j, where $j = 0$ to $N-1$. Let M be the number of modes of operation, denoted as M_m, where $m = 0$ to $M-1$. Let N_m be the number of tasks in a system mode M_m. Each task in a mode M_m has a certain priority, P_k, where $k = 0$ to $N_m - 1$. The task with priority P_0 has the highest priority, and the one with priority $P_{(N_m-1)}$ has the least priority. Characteristics of tasks in each mode are stored in a "Mode-LUT." Each system task has several ASP circuit variants for RTSA. The number of ASP circuit variants of a task T_j is referred as V_j. Characteristics of the ASP circuit variant of all the tasks, i.e., their resource utilization, operating frequency, and performance, are stored in a "Variant-LUT." Consider an example of a multitask multimodal system having a total of 6 tasks ($N = 6$), T_0 to T_5, and three modes ($M = 3$), M_0 to M_2. The number of tasks in each mode is represented as N_0 to N_2. The Mode-LUT for this example is shown in Table 1. The table shows the following:

(i) The number of tasks in each mode

(ii) The set of tasks that form the mode along with their priorities

(iii) Range of permitted performance specification of each task in each mode, i.e., maximum performance h_{spec} to minimum permitted performance l_{spec}

(iv) The existence condition EC for every task, which determines whether a task in a mode can be eliminated during system operation or not

For example, mode M_2 of the system has $N_2 = 3$ tasks. The mode involves tasks T_0, T_1, and T_3 in the same order of priority. This means $P_0 = T_0$, $P_1 = T_1$, and $P_2 = T_3$. The h_{spec} and l_{spec} values provided in the Mode-LUT are relative with

Table 1: An example of Mode-LUT for Explorerv3.

Mode	No. of tasks (N_m)	$P = 0$		$P = 1$		$P = 2$		$P = 3$	
		h_{spec}	l_{spec}	h_{spec}	l_{spec}	h_{spec}	l_{spec}	h_{spec}	l_{spec}
		EC		EC		EC		EC	
M_0	$N_0 = 4$	T_2 8 1	8	T_5 8 1	8	T_0 8 0	8	T_4 8 0	4
M_1	$N_1 = 4$	T_3 8 1	8	T_2 8 0	2	T_5 8 0	2	T_1 8 0	2
M_2	$N_2 = 3$	T_0 8 1	8	T_1 8 1	8	T_3 8 0	2		

respect to the minimum values. For example, performance of T_0 is measured in frames per second (fps). If the maximum and minimum frame rates are 240 fps and 30 fps, they can be referred in a relative scale of 8 to 1. Thus, in mode M_2, as shown in Table 1, T_0 has $h_{spec} = l_{spec} = 8$, which means it is a critical task that always needs to operate at 240 fps. Similarly, if performance of T_3 is measured in Mbps and if the maximum and minimum permitted data rate is 16 and 2 Mbps, respectively, they can also be referred on a relative scale of 8 to 1. In mode M_2, T_3 has $h_{spec} = 8$ and $l_{spec} = 2$, which means it can have an output data rate in the range of 16 Mbps and 4 Mbps. From the values of EC of the tasks, T_0 and T_1 cannot be eliminated during system operation, while T_3 can be eliminated.

For the considered example, each task T_0 to T_5 has 10 ASP circuit variants, i.e., V_0 to $V_5 = 10$. Table 2 represents the Variant-LUT for the tasks. It stores the performance, operating frequency, number of FPGA slots, Logic slices, BRAM slices, and DSP slices occupied by each variant of each task.

Some other system parameters required for Explorer operation are as follows:

(i) The current performance of a task is expressed by the parameter called Current Possible Performance (CPP). Since a task's performance can be modified between h_{spec} and l_{spec} during RTSA, CPP refers to the performance of a task at which Explorer is attempting to find an ASP circuit variant for that task during RTSA.

(ii) Since the ASP circuit variants can operate at different frequencies, the current operating frequency of a task T_j is referred as F_j.

(iii) Since tasks can be eliminated to adapt to the existing constraints, the active number of tasks in a mode M_m may not always be equal to N_m. A track of the active number of tasks is maintained by a parameter N_a, where $N_a \leq N_m$.

(iv) While the performance constraints (h_{spec} to l_{spec}) for the system tasks can be obtained from the Mode-LUT, the system also stores the other constraints in a set called Constraint Set. The constraints include the current required mode M_m, the power budget in terms of permitted total power consumption (PTPC), permitted die temperature range (PDTR) ($T_{low} - T_{high}$), and available hardware resources in terms of number of slots N_s. An example of the Constraint Set is shown in Table 3. This set of constraints is dynamic; that is, it is updated in run-time whenever there is a change in some or all the constraints. Change in system mode may be required either due to events such as position of the system (e.g., orbital position of a satellite requiring execution of a set of tasks) or unpredictable events such as approach to/collision with an object (e.g., pedestrian appearing in front of a self-driven car), etc. PTPC depends on the power consumption of the current set of executing tasks and the available power (e.g., solar/wind energy for rechargeable batteries). PTPC can drop/rise due to factors such as more/less power consumed by the tasks in the current system mode, a possible fault/fault restoration in the power generator, absence/abundance of solar or wind energy to recharge the power sources, etc. PDTR depends on factors such as PTPC [16], current die temperature, off-chip, i.e., on-board temperature, and external environmental temperature. PDTR can rise or fall depending on PTPC and the difference between the on-chip and off-chip temperatures. Usually, since power consumption and die temperature are related, a rise/fall in PTPC does result in a rise/fall in the PDTR, and the amount of which depends on the other factors which influence the PDTR. N_s can dynamically change due to occurrence or restoration of hardware faults.

(v) The system stores the DPCEM, STPCEM, and SDTEM coefficients for the current application and FPGA/SoPC device on which the system is developed. These coefficients are used by Explorer to evaluate the power consumption and die temperature of candidate system configurations during the RTSA process.

6. Explorer Functioning

On system start-up, Explorer functions to find the system configuration that sets the system in its default mode and which satisfies the default values in the Constraint Set. It is then invoked whenever there is a change in the set of

Table 2: An example of Variant-LUT for Explorer.

Variant no.	No. of slots	F_{sys} (MHz)	Performance	Logic slices	BRAM slices	DSP slices
$T_0 - 0$	1	240	8	3093	43	30
$T_0 - 1$	2	120	8	6062	79	54
$T_0 - 2$	1	120	4	3093	43	30
$T_0 - 3$	3	60	8	11877	142	101
$T_0 - 4$	2	60	4	6062	79	54
$T_0 - 5$	1	60	2	3093	43	30
$T_0 - 6$	5	30	8	23259	270	193
$T_0 - 7$	3	30	4	11877	142	101
$T_0 - 8$	2	30	2	6062	79	54
$T_0 - 9$	1	30	1	3093	43	30
$T_1 - 0$	1	240	8	2061	22	82
$T_1 - 1$	2	120	8	4040	36	158
$T_1 - 2$	1	120	4	2061	22	82
$T_1 - 3$	3	60	8	7914	67	304
$T_1 - 4$	2	60	4	4040	36	158
$T_1 - 5$	1	60	2	2061	22	82
$T_1 - 6$	5	30	8	15499	121	589
$T_1 - 7$	3	30	4	7914	67	304
$T_1 - 8$	2	30	2	4040	36	158
$T_1 - 9$	1	30	1	2061	22	82
$T_2 - 0$	1	240	8	5003	27	24
$T_2 - 1$	2	120	8	9806	44	37
$T_2 - 2$	1	120	4	5003	27	24
$T_2 - 3$	3	60	8	19212	79	66
$T_2 - 4$	2	60	4	9806	44	37
$T_2 - 5$	1	60	2	5003	27	24
$T_2 - 6$	5	30	8	37623	142	122
$T_2 - 7$	3	30	4	19212	79	66
$T_2 - 8$	2	30	2	9806	44	37
$T_2 - 9$	1	30	1	5003	27	24
$T_3 - 0$	1	240	8	4009	17	47
$T_3 - 1$	2	120	8	7858	27	86
$T_3 - 2$	1	120	4	4009	17	47
$T_3 - 3$	3	60	8	15395	43	159
$T_3 - 4$	2	60	4	7858	27	86
$T_3 - 5$	1	60	2	4009	17	47
$T_3 - 6$	5	30	8	30148	76	298
$T_3 - 7$	3	30	4	15395	43	159
$T_3 - 8$	2	30	2	7858	27	86
$T_3 - 9$	1	30	1	4009	17	47
$T_4 - 0$	1	240	8	5088	39	51
$T_4 - 1$	2	120	8	9972	68	82
$T_4 - 2$	1	120	4	5088	39	51
$T_4 - 3$	3	60	8	19338	122	148
$T_4 - 4$	2	60	4	9972	68	82
$T_4 - 5$	1	60	2	5088	39	51
$T_4 - 6$	5	30	8	38162	228	274
$T_4 - 7$	3	30	4	19338	122	148
$T_4 - 8$	2	30	2	9972	68	82
$T_4 - 9$	1	30	1	5088	39	51
$T_5 - 0$	1	240	8	2567	33	73
$T_5 - 1$	2	120	8	5011	53	131
$T_5 - 2$	1	120	4	2567	33	73
$T_5 - 3$	3	60	8	9857	92	255
$T_5 - 4$	2	60	4	5011	53	131
$T_5 - 5$	1	60	2	2567	33	73
$T_5 - 6$	5	30	8	19104	164	497
$T_5 - 7$	3	30	4	9857	92	255
$T_5 - 8$	2	30	2	5011	53	131
$T_5 - 9$	1	30	1	2567	33	73

Table 3: An example of Constraint Set for Explorer.

Mode M_i	PTPC (W)	PDTR (°C)	N_s
M_0	6	68–70	8

constraints. Based on the changes observed in the Constraint Set, Explorer follows different decision flows which are discussed below.

6.1. Constraint_Monitor Flow. Explorer uses this flow, shown in Figure 2, when it is invoked due to dynamic changes in the Constraint Set, i.e., changes in the system mode, power budget, die temperature, and/or hardware resources. Based on the change observed in the Constraint Set, Explorer follows one of the flows discussed in this section. It is to be noted that Explorer can be invoked only when it has completed one flow cycle; that is, it has come out of the flow that it was currently working on. Therefore, if there are multiple changes in the Constraint Set, the order of priority is as follows: change of mode followed by change in number of available slots, followed by power budget change and die temperature constraint change. If a change in mode is required, the active set of tasks must be modified according to the new required mode. Therefore, Explorer must cater to this change first. If there is a hardware fault or it is restored, it affects the available number of slots. This can reduce the hardware resources available for the executing tasks or it can make space for tasks to function at a better performance. Therefore, change in hardware resource constraint needs to be considered before power budget and die temperature constraints. It has been discussed in the Temperature_Analysis Flow that for the system's survival, meeting the PTPC constraint has a higher preference over the PDTR constraint. Therefore, change in the PTPC is catered to before change in PDTR. Based on this discussion, the following cases are possible:

(i) If there is a request for change in system mode, Explorer goes to the Mode_Change Flow to find a suitable system configuration for the set of tasks that form the mode. If there are simultaneous changes in resource, power budget, and/or die temperature constraints, the system configuration for the new mode is selected based on the new constraints.

(ii) If there is a change in the available hardware resources due to occurrence of a hardware fault, Explorer goes to the Hardware_Fault Flow to adapt the system to the reduced hardware resources. Alternatively, if a hardware fault is restored, Explorer goes to the Hardware_Fault_Recovery Flow to adapt to the increased hardware resources. If there are simultaneous changes in power budget and/or die temperature constraints, they are catered to, in the said flows, after the hardware resource constraint is satisfied.

(iii) If the PTPC has dropped, Explorer checks if the STPC of the current configuration (CSTPC) meets the new PTPC constraint. If it satisfies the new PTPC specification, Explorer goes to the Temperature_Analysis Flow to see if the PDTR requirement is met. However, if CSTPC > PTPC, it means the current system configuration fails the new power budget restriction. Explorer goes to the Reduce_System_DPC Flow to find a candidate configuration with lower Estimated STPC (ESTPC). Once such a configuration is found and evaluated for the PTPC constraint using the Power_Analysis Flow, Explorer goes to the Temperature_Analysis Flow to see if the PDTR requirement is met.

(iv) If power budget has increased, Explorer needs to find a configuration such that CSTPC < ESTPC < PTPC. This means that the ESTPC of the new candidate configuration must be higher than the TPC of the current configuration, but lower than the new PTPC constraint. To achieve this, Explorer jumps to the Increased_Power_Budget Flow. Once such a configuration is found and evaluated for the PTPC constraint using the Power_Analysis Flow, Explorer goes to the Temperature_Analysis Flow to see if the PDTR requirement is met.

(v) If there is no change in PTPC, but there is a change in the PDTR, Explorer goes to the Temperature_Analysis Flow to see where the die temperature of the current configuration stands with respect to the new PDTR and accordingly decide the next steps to be taken.

6.2. Mode_Change Flow. Explorer comes to this flow, shown in Figure 3, during the system start-up in default mode or when it invoked to change the mode of operation while the system is running. Explorer extracts the PTPC, PDTR, and N_s from the Constraint Set. Explorer then extracts the tasks that form the mode M_m, their priorities, and performance specifications from the Mode-LUT. It sets $N_a = N_m$, the CPP of each task in the mode M_m to its h_{spec} and frequency of every task T_j to its maximum operating frequency $F_{j\_\max}$. It then jumps to the Find_System_Configuration Flow to select ASP circuit variants for all the tasks in the mode M_m such that the resultant configuration establishes system parameters that are closest to the Constraint Set.

6.3. Find_System_Configuration Flow. This decision flow, shown in Figure 4, helps Explorer to select a suitable system configuration, i.e., ASP circuit variants for the tasks in the active set, which are appropriate according to the existing set of constraints. The search begins from the task with highest priority P_0. Explorer jumps to the Find_Task_Variant Flow to find a suitable ASP circuit variant for that task. Once a variant is selected and Explorer is back to this flow, it moves to the next task, i.e., task with priority P_1, and repeats the process. This continues up to the last active task, i.e., task with priority P_{N_a-1}. Once the ASP circuit variants of all the tasks are selected, the set forms the candidate system configuration to be evaluated against the power budget and temperature constraints. Note that selection of a candidate

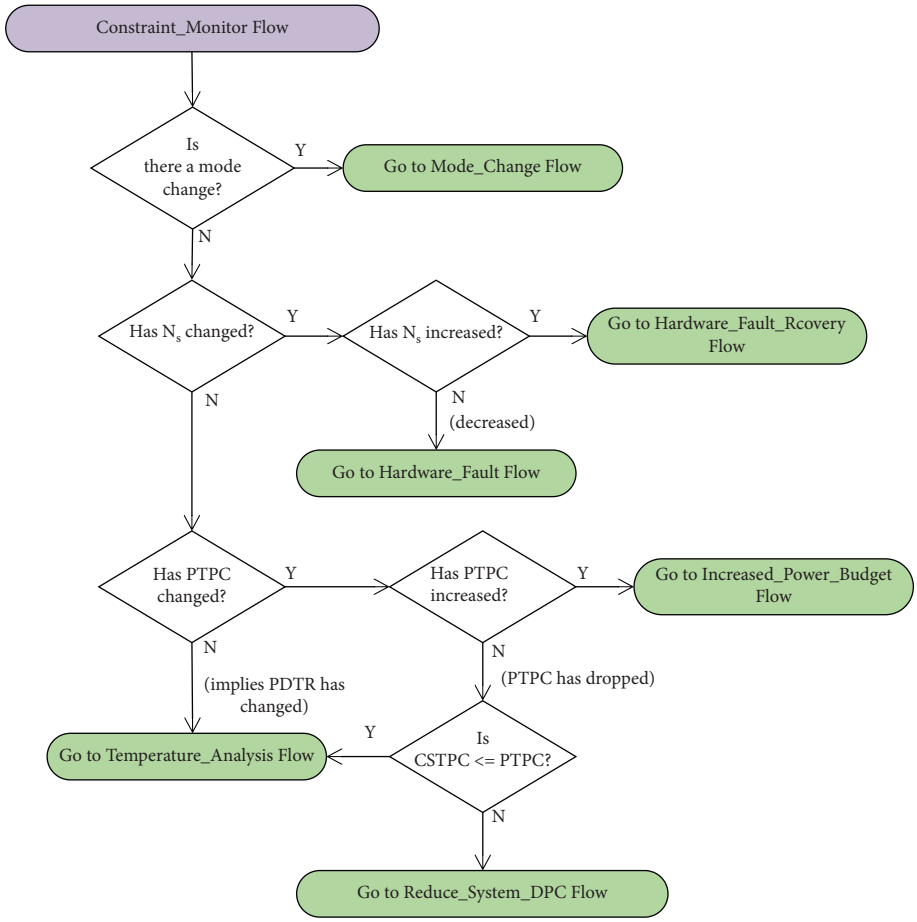

Figure 2: Constraint_Monitor flow of Explorer.

implies that it meets the hardware resource constraint. Explorer therefore goes to the Power_Analysis Flow for the required verification.

6.4. Find_Task_Variant Flow. Explorer uses this flow, shown in Figure 5, to select a suitable ASP circuit variant for a task T_j under consideration from its Variant-LUT. Starting from the top of the LUT, Explorer checks if the frequency of the ASP circuit variant is equal to the required frequency F_j, if performance of the ASP circuit variant is equal to CPP of the task, and if it fits in the available number of slots on the FPGA. If no variant is selected due to lack of available number of spare slots, Explorer goes to the Space_Adjustment Flow to create space for the task. If there is no variant selected due to any other reason, there is an error. This can happen only if there is some error while defining the system and task specifications during design time. Once an ASP circuit variant that matches all the conditions is found, Explorer returns to the flow that had invoked the Find_Task_Variant Flow.

6.5. Power_Analysis Flow. This flow, shown in Figure 6, is used to evaluate whether a candidate system configuration meets the PTPC constraint. Explorer extracts the resource utilization of the system configuration from the Variant-LUT, and the STPCEM coefficients to estimate the STPC of this configuration. If ESTPC ≤ PTPC, the configuration meets the power budget constraint. It then goes to the Temperature_Analysis Flow and checks whether the candidate configuration satisfies the PDTR. On the other hand, if ESTPC > PTPC, the candidate configuration does not satisfy the PTPC constraint. Explorer therefore goes to the Reduce_System_DPC Flow to begin the search for another system configuration which has a lower ESTPC to satisfy the PTPC.

6.6. Temperature_Analysis Flow. This flow, shown in Figure 7, is used to evaluate whether a system configuration meets the PDTR constraint. Explorer comes to this flow either after evaluating a candidate system configuration for the power budget constraint using the Power_Analysis Flow or it approaches this flow directly if there is a change in only the PDTR in the Constraint Set. In the former case, Explorer uses the SDTEM coefficients to estimate the die temperature (ESDT) that the configuration would result in, while in the latter case, Explorer evaluates the SDT of the current configuration (CSDT) against the PDTR constraint. If $T_{\text{low}} \leq \text{ESDT/CSDT} \leq T_{\text{high}}$, the configuration satisfies the PDTR constraint and is, therefore, selected as the new system configuration. Explorer then waits for the next

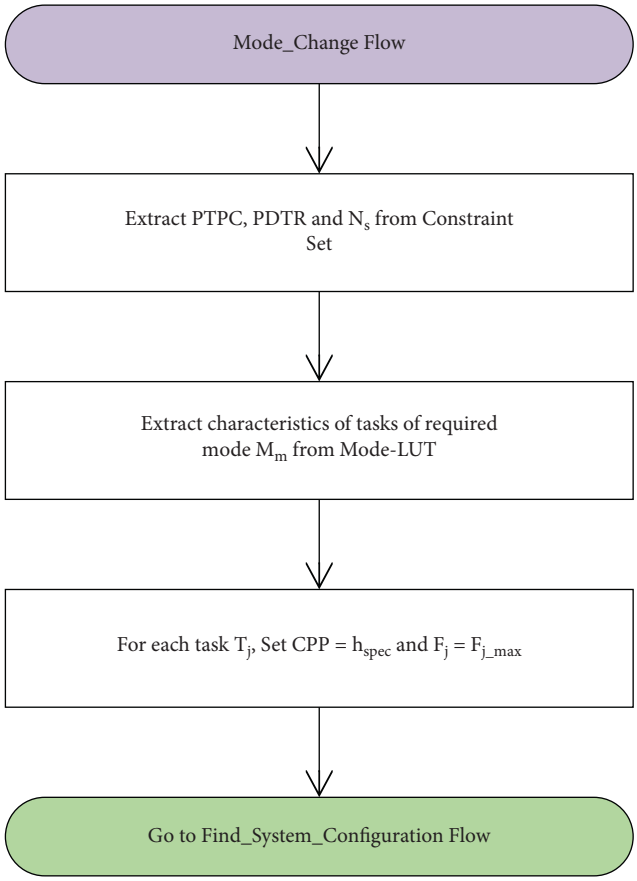

Figure 3: Mode_Change flow of Explorer.

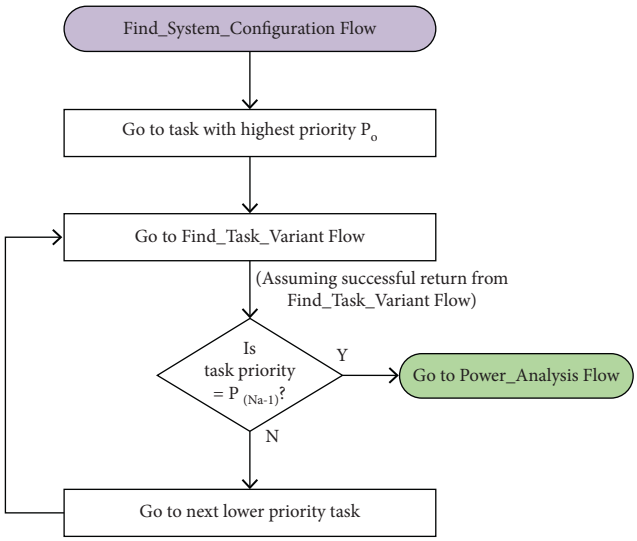

Figure 4: Find_System_Configuration flow of Explorer.

instance it is invoked. If ESDT/CSDT $<T_{\text{Low}}$, the system will still have to accept the configuration as the new system configuration. This is because, to increase the die temperature, a candidate configuration which has a higher ESTPC will need to be found. Such a configuration will fail the power budget constraint. If the system works on a configuration that does not meet the PTPC constraint, it may not

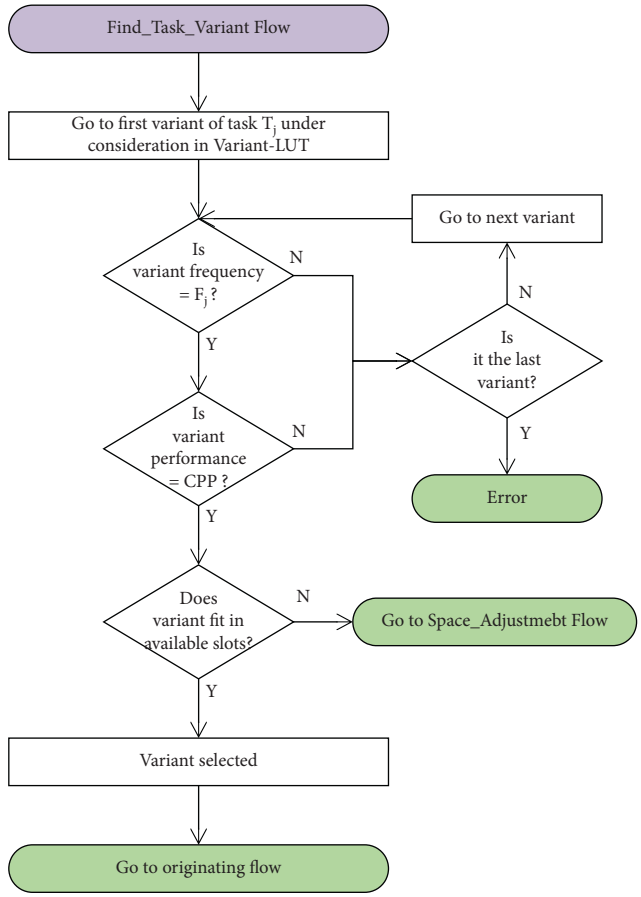

Figure 5: Find_Task_Variant flow of Explorer.

be able to survive for the required amount of time and may die down before the system can be recharged. To avoid this, power budget constraint gets a priority over the die temperature constraint. Therefore, in case of contradicting constraints, the system sticks to the configuration that meets the PTPC requirement. If ESDT/CSDT $>T_{\text{High}}$, the candidate configuration fails the PDTR constraint. Explorer goes to the Reduce_System_DPC Flow to begin the search for another system configuration which has a lower ESTPC and hence lower ESDT to satisfy the PDTR. Finding a configuration which has lower ESTPC than the current candidate does not affect the system, as that configuration will also satisfy the power budget constraint. In fact, it will increase the time the system can survive without being recharged. The only cost could be degradation in the performance of some less critical tasks if enough slots are not available to maintain their performance.

6.7. Space_Adjustment Flow. When the frequency F_j of a task T_j is reduced to reduce the system's power consumption, the corresponding ASP circuit variant will occupy more slots if the task performance needs to be maintained. In such a scenario, it is possible that enough spare slots are not available to accommodate the ASP circuit variant. Explorer uses this flow, shown in Figure 8, to cater to the issue. It tries to reduce the performance, i.e., CPP, of task T_j by a step

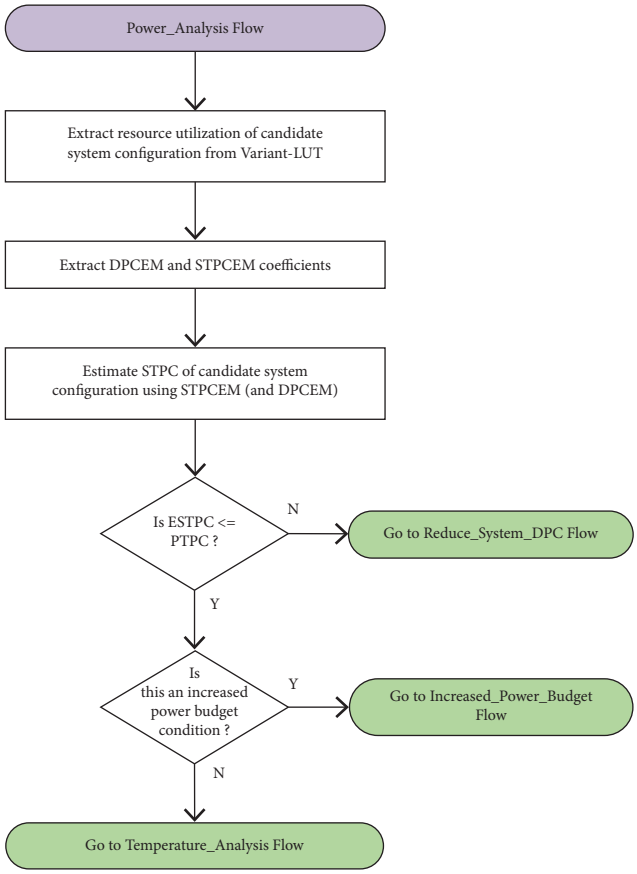

Figure 6: Power_Analysis flow of Explorer.

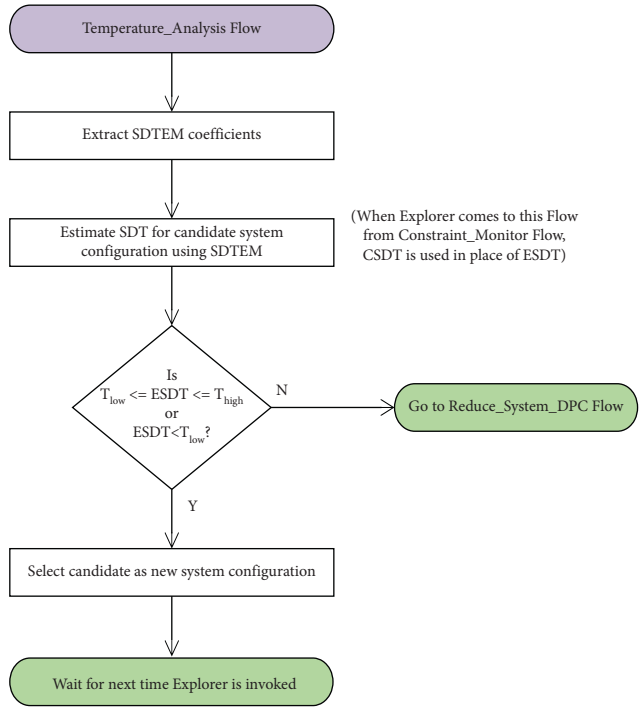

Figure 7: Temperature_Analysis flow of Explorer.

(within the specified performance range), so that another ASP circuit variant that occupies lesser slots could be configured in the available space. It goes to the Find_Variant Flow to select such an ASP circuit variant. However, if the CPP of task T_j is already at l_{spec}, Explorer saves the task priority P_s and goes to the task with a higher priority. It reduces the CPP of this task by a step to find an ASP circuit variant that can take up lesser number of slots so that space is created for the task T_j with priority P_s in consideration. If such an ASP circuit is found, Explorer goes back to the task with priority P_s and tries to find an ASP circuit variant that can fit in the new created space using the Find_Variant Flow. If an ASP circuit variant for task T_j is found, Explorer goes back to the original flow that had invoked this flow. While working with tasks of higher priority than P_s to create free slots, if all the tasks right up to P_0 are operating at l_{spec}, as a result of which CPP of none of the tasks can be lowered further, the task with least priority, $P_{(N_a-1)}$, among the active set of tasks needs to be eliminated. Note that a task with priority P_s may or may not be the one with least priority $P_{(N_a-1)}$. If the EC of the least priority task = 0, the task is removed. If EC = 1, Explorer throws an error. This again indicates erroneous specifications during system-design time. It is to be noted that when a task is removed, there can be a big drop in power consumption of the resultant system configuration. This means, although Explorer came to this flow while in the process of reducing the system's STPC, the big drop in STPC due to the task removal may result in ESTPC of the configuration being way lower than the PTPC constraint. This could, therefore, reverse the situation; that is, the PTPC constraint could now act as an increased power budget constraint. To cater to this possibility, Explorer goes to the Increased_Power_Budget Flow when a task is removed from a configuration.

6.8. Reduce_System_DPC Flow. This flow, shown in Figure 9, is used to find a system configuration that has a lower ESTPC as compared to the current one so that the system's power consumption is reduced and the PTPC and/or PDTR constraints are met. Explorer begins with the least priority task. It reduces the frequency F_j of the task by a step and goes to the Find_Variant Flow to look for an ASP circuit variant at that frequency and at the same performance or a reduced one if enough spare slots are not available. If such a variant is found, a new candidate system configuration is obtained. Explorer goes to the Power_Analysis Flow to evaluate the ESTPC of this configuration with respect to PTPC. If a variant is not found, Explorer follows the different paths in the Find_Variant Flow depending on the situation. For example, if a variant is not selected due to lack of available spare slots, Explorer goes to the Space_Adjustment Flow, as discussed in the Find_Variant Flow description. While trying to reduce frequency F_j of the task, if F_j is at its minimum value, Explorer tries to reduce the performance of the task. It reduces CPP by a step and goes to the Find_Variant Flow just as discussed above. If the task is already at its l_{spec}, it goes to a higher priority task to reduce its frequency or CPP by a step like in the case of the least

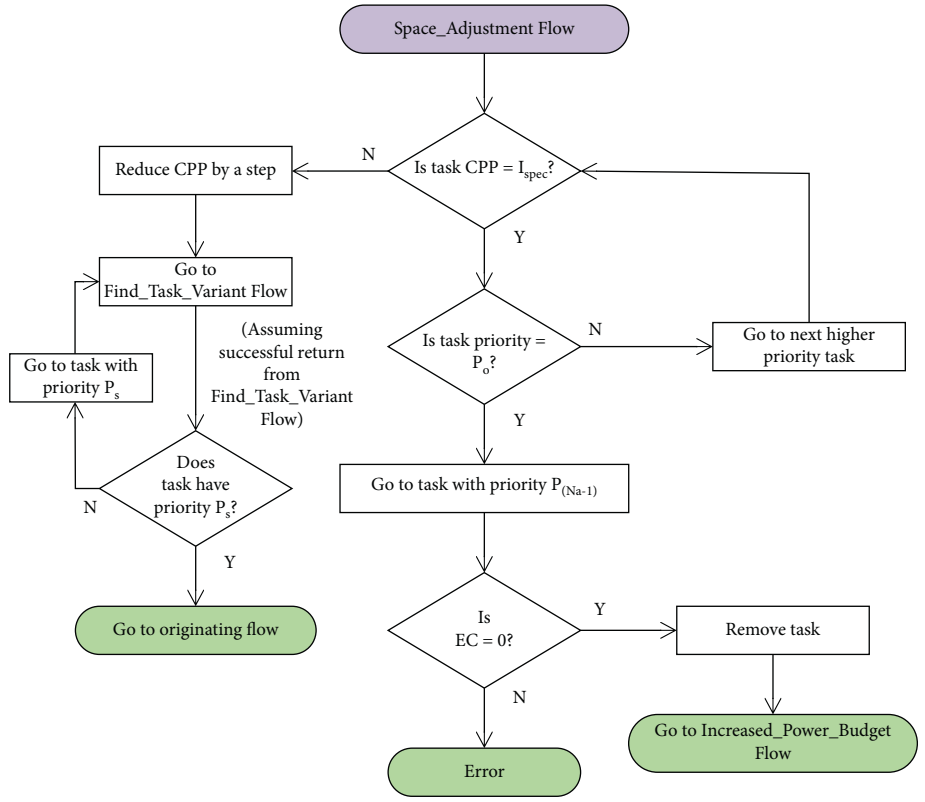

Figure 8: Space_Adjustment flow of Explorer.

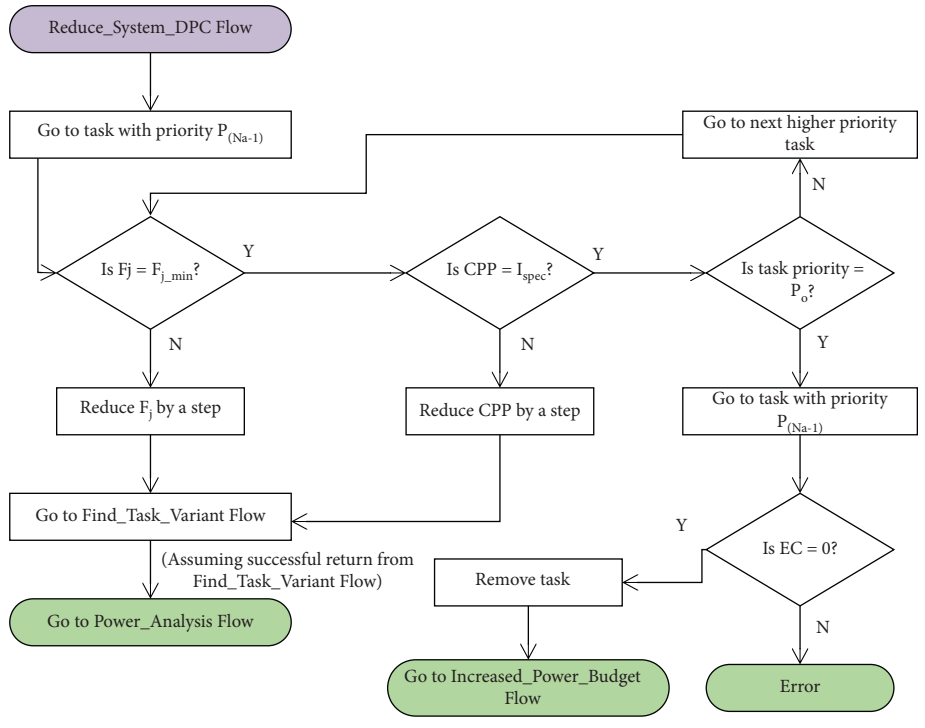

Figure 9: Reduce_System_DPC flow of Explorer.

priority task. If the frequencies of all the higher priority tasks are at the minimum and/or the CPP of these tasks are at their respective l_{spec}, Explorer checks the EC condition for the least priority active task and removes it if EC = 0. It then goes to the Increased_Power_Budget Flow due to same reason as discussed in the Space_Adjustment Flow.

6.9. Increased_Power_Budget Flow. When PTPC condition improves over the existing one, there is a scope to increase the system's STPC. This means that there is a scope to increase the performance of some or all the tasks which have been operating at a lower performance due to previous low power budget constraint. Operation at a higher performance and frequency also frees up some slots which further allows adding the tasks of the mode which have been eliminated due to the previously low PTPC requirement. When Explorer is in the Reduce_System_DPC Flow, it keeps a track of the task whose frequency or performance is last reduced during adaptation, so that if the power budget further reduces, the run-time adaptation can begin from the same task to save exploration and adaptation time. Let the priority of this task be P_k. Now, when the power budget has increased, Explorer begins adaptation using this flow, shown in Figure 10, from the same task. It increases the operating frequency of this task by a step and sets its CPP to h_{spec}. It then goes to the Find_Variant Flow to select an ASP circuit variant of the task with priority P_k that meets the new frequency and performance requirements. Once a variant is found, and if the ESTPC of the resultant configuration is still lower than the PTPC, the operating frequency F_j is increased again by a step and the process repeats. If the frequency reaches maximum and the ESTPC of the candidate configuration is still lower than PTPC, the frequency of the next lower priority task is increased by a step and its CPP is set to h_{spec}. This continues till the frequency of the last active task reaches maximum with its CPP set to h_{spec}. If the ESTPC is still lower than the PTPC, Explorer adds an eliminated task of the next lower priority, if any, and repeats the above process. The above procedure stops when a candidate configuration is found which has ESTPC > PTPC. Explorer then goes to the Reduce_System_DPC Flow and finally settles onto a configuration which has ESTPC < PTPC. This process is needed so that the performance of as many tasks as possible can be increased and as many eliminated tasks as possible can be added back. If the power budget has increased significantly, it could be possible to have a system configuration with all the tasks of a mode running at their maximum frequencies and performances. If Explorer accepted a configuration in the first round itself, there could still be room for increasing some task's performance or adding in an eliminated task and the opportunity would have been missed. Once a system configuration that satisfies the PTPC condition is selected, it is evaluated against the PDTR requirements using the Temperature_Analysis Flow.

6.10. Hardware_Fault Flow. The method of run-time ASP component relocation presented in [42, 43] is used by Explorer as the fault mitigation method. The method proposes that if there is a hardware fault in a slot, the affected ASP component must be relocated to a spare slot. This way, recovery time of the affected component is only its relocation time. The faulty slot can then be diagnosed simultaneously while the recovered ASP component is functioning. Explorer implements this method using the Hardware_Fault Flow, shown in Figure 11. If a spare slot is available, the relocation can be immediate. However, if there is no available spare slot, Explorer needs to adapt the system configuration to create a spare slot. To do so, it begins by trying to reduce the CPP of a task, starting from the one with the least priority among the active set of tasks. If the CPP of any task can be reduced, Explorer finds a suitable variant for that task using the Find_Variant Flow such that it occupies lesser number of slots, thus creating a spare slot(s). The affected ASP component is then relocated to a created spare slot. Once the affected ASP component is restored, Explorer goes to the Power_Analysis Flow to verify whether the resultant configuration meets the PTPC constraint. While trying to reduce CPP of a task, if all the tasks are operating at their l_{spec} and reducing CPP is not possible, Explorer eliminates the last active task after verifying its EC to create a spare slot(s). Once a spare slot is available, the affected ASP component is relocated to the spare slot. As discussed in the Space_-Adjustment Flow, if a task is removed for fault mitigation, there can be a big drop in the power consumption of the resultant system configuration, resulting in a big gap between its STPC and PTPC. Therefore, there is a possibility for tasks operating at degraded performances to increase their performances. To achieve this, Explorer goes to the Increased_Power_Budget Flow after relocating and recovering the affected ASP component.

6.11. Hardware_Fault_Recovery Flow. Once a slot with a transient fault is recovered, it is back in the system as a spare slot. Explorer is therefore invoked to put the slot to use, if needed, using this flow shown in Figure 12. If a spare slot was available when the hardware fault occurred, there is no change in the system configuration as the affected task component is simply relocated to the spare slot. In this case, no change needs to be made when the faulty slot is recovered. However, if Explorer carried out RTSA due to lack of a spare slot when the fault occurred, the original system configuration that existed prior to the fault must be restored. Explorer therefore extracts the original system configuration from the memory and selects it as the potential new system configuration which can rectify any degradation or elimination of a task which occurred due to the fault. Explorer then goes to the Power_Analysis Flow to verify if the configuration meets the power consumption constraint.

Thus, the above scenarios demonstrate that a MACROS-based system deployed with Explorer can adapt in run-time to changing mode, power budget, die temperature, and/or hardware resource constraints by dynamically choosing suitable ASP circuit variants of tasks at suitable operating frequencies such that they together fit in the available number of slots on the FPGA die, critical tasks of the desired system mode operate at their maximum performance, noncritical tasks operate within a permitted performance range, the system's power consumption is within the available power budget, and the die temperature is within the permitted range. As a by-product of the run-time structural adaptation, the system's lifetime can also increase; the extent of which depends on the relation between the system's PTPC and STPC of the selected configuration. This versatility of

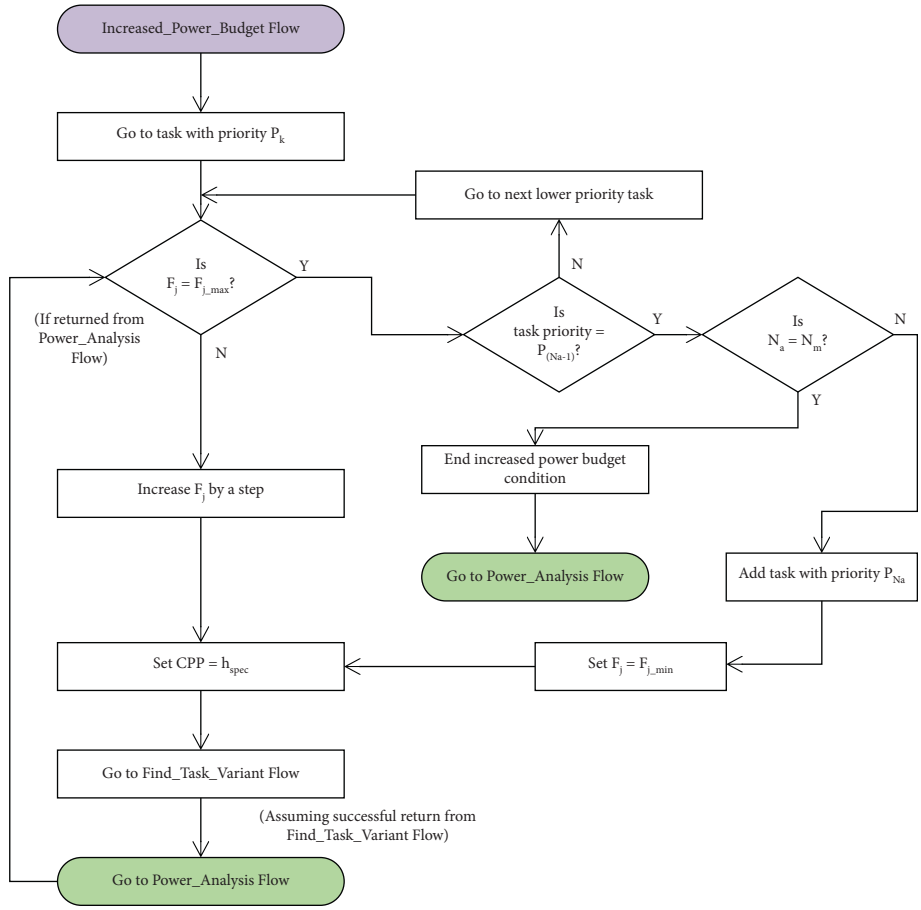

Figure 10: Increased_Power_Budget flow of Explorer.

Explorer allows autonomous and mobile systems to be completely sustainable!

7. Demonstration of RTSA Using Explorer

This section discusses an example that shows how Explorer enables a system to structurally adapt to the varying workload, power budget, die temperature, and hardware resource conditions in run-time. It demonstrates how, whenever there is a change in the set of constraints, Explorer dynamically selects a suitable system configuration that closely satisfies the constraints at that time. Consider a system developed on the Zynq XC7Z020 device. The system has 6 tasks, T_0 to T_5, and three modes of operation, M_0 to M_2. Each task has 10 ASP circuit variants for RTSA. The tasks can operate at 30, 60, 120, and 240 MHz. Tables 1 and 2 presented in Section 5 represent the Mode-LUT and Variant-LUT, respectively, for the example system considered here. The Zynq device is configured with the MACROS framework and is divided into 8 PRRs/slots to support the system's dynamic workload and RTSA. The system runs on a rechargeable 12 V battery with a capacity of 48 Wh.

The DPC required to estimate the STPC for Zynq is obtained using the DPCEM; that is, equations (2) and (3) combined together. The coefficients for the DPCEM of the Zynq SoC are assumed to be the same for all tasks, and they are as follows [3]: $C_{LS} = 0.013$, $C_{BS} = 1.1$, $C_{DS} = 0.226$, and $C_F = 23.046$ for a total of 69 IOBs on Zynq. With this information, the DPCEM equation for the Zynq SoC is obtained as follows:

$$\mathrm{DPC}_{(\mathrm{Zynq})}(\mathrm{mW}) = \mathrm{DPC}_{(\mathrm{Task}_1)} + \mathrm{DPC}_{(\mathrm{Task}_2)} + \cdots + \mathrm{DPC}_{(\mathrm{Task}_{N_m})}, \tag{6}$$

where N_m is the number of tasks in a mode M_m of the system considered in this example. The DPC of each task can be obtained as

$$\mathrm{DPC}_{(\mathrm{Task}_n)}(\mathrm{mW}) = \frac{F_{ccn}}{F_{\min}} \times \left(0.013 \times N_{LSn} + 1.1 \times N_{BSn} + 0.226 \times N_{DSn} + C_{Fn}\right). \tag{7}$$

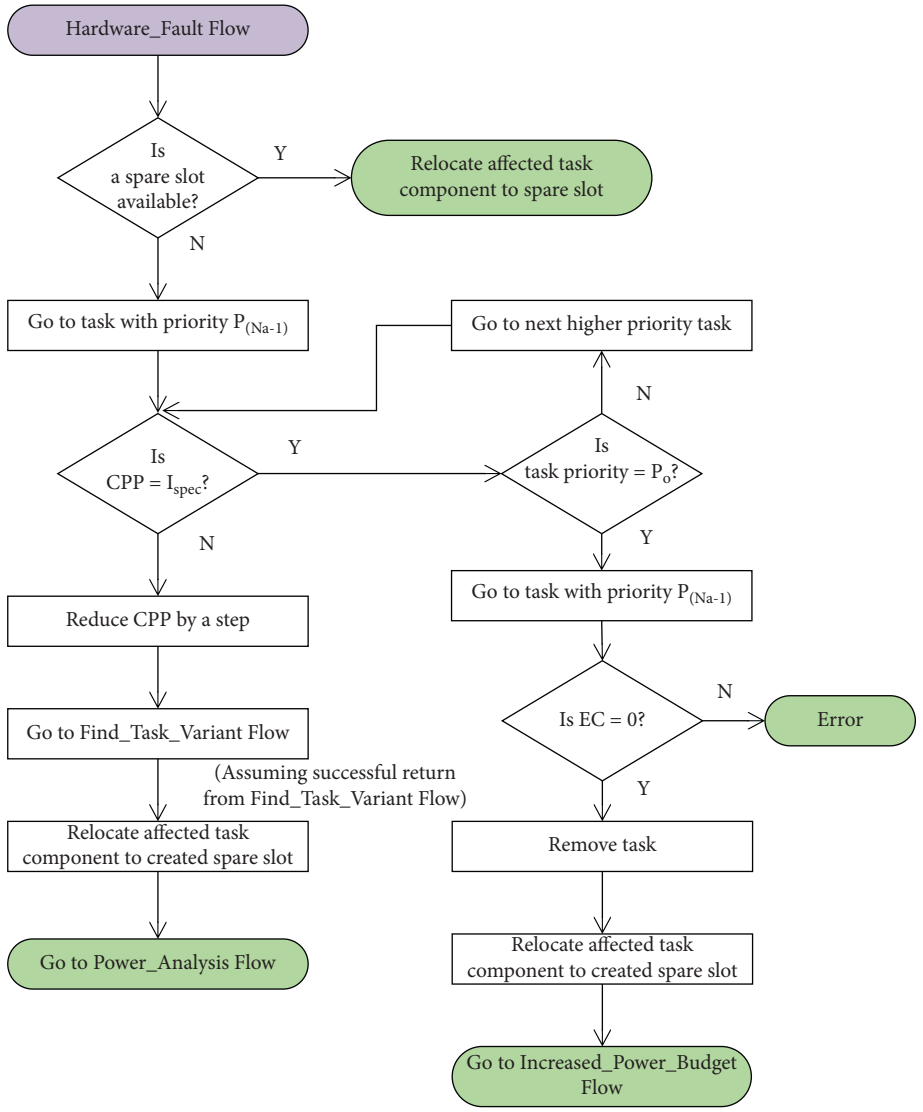

Figure 11: Hardware_Fault flow of Explorer.

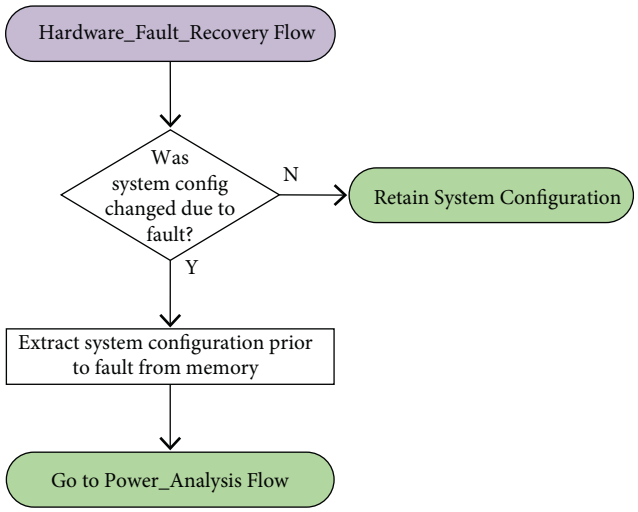

Figure 12: Hardware_Fault_Recovery flow of Explorer.

Note that, for the example considered here, the coefficient C_F is used once for all the IOBs used by the tasks together. This means

$$C_F = \frac{F_{cc}}{F_{min}} \times \left(\sum_{n=0}^{N_m - 1} C_{Fn} \right) = \frac{F_{cc}}{F_{min}} \times 23.046. \quad (8)$$

The frequency F_{cc} for the IOBs is the operating frequency of the least priority task among the active tasks.

For the example in this section, SPC of Zynq is 2.340 W, and R, the rise in TPC to reach STPC, is 0.06 W [16]. All the tasks utilize only the FPGA resources, and the ARM Cortex-A9 processor of the Zynq SoC is therefore not used. Hence, SPP = 0. With these data, the STPCEM equation, equation (4) can be rewritten for Zynq as follows:

$$\begin{aligned} \mathrm{STPC}_{(\mathrm{Zynq})}(\mathrm{W}) &= 2.34(\mathrm{W}) + \mathrm{DPC}(\mathrm{W}) + 0.06(\mathrm{W}) \\ &= \mathrm{DPC}(\mathrm{W}) + 2.4. \end{aligned} \quad (9)$$

Equation (5) is the SDTEM equation for a FPGA. For the Zynq device, $M = 11.85$ and $C = 4.88$ [16]. With these values, the SDTEM for Zynq is obtained as

$$\mathrm{SDT}_{(\mathrm{Zynq})}(^{\circ}\mathrm{C}) = 11.85 \times \mathrm{STPC}(\mathrm{W}) + 4.88. \quad (10)$$

Explorer uses the set of equations to estimate the STPC and SDT for candidate configurations while carrying out RTSA for the system considered here. Table 4 summarizes the flow of events that occur, which are discussed next.

7.1. Case 1: Initial State. At system start-up, the battery capacity is 100%, i.e., 48 Wh. The system needs to be able to function for at least 8 hours without refueling. This means that PTPC = 6 W. The system's PDTR is 68–70°C. The Constraint Set for this case is shown in Table 5(a). The default mode is M_0, formed by tasks T_2, T_5, T_0, and T_4 in the same order of priority, as shown in the Mode-LUT in Table 1. Explorer goes to the Mode_Change Flow, extracts the mode and task characteristics, and goes to the Find_System_Configuration Flow. It selects variant number 0 for all the tasks using the Task_Variant Flow; each variant operates at 240 MHz, occupies 1 slot, and provides a performance of 240 fps. It then goes to the Power_Analysis Flow to estimate the STPC for the candidate configuration. Using Table 2 and the above-mentioned model equations, ESTPC of this combination is obtained as

$$\mathrm{ESTPC}(\mathrm{mW}) = \frac{240}{30} \times \{0.013 \times 15751 + 1.1 \times 142 + 0.226 \times 178 + 0.334 \times 69\} + 2400 = 5793.9\,\mathrm{mW} = 5.79\,\mathrm{W}. \quad (11)$$

Since ESTPC ≤ PTPC, the candidate configuration satisfies the power budget. Explorer, therefore, goes to the Temperature_Analysis Flow to estimate its SDT. Using the SDTEM equation, ESDT for this configuration of $T_2 - 0$, $T_5 - 0$, $T_0 - 0$, and $T_4 - 0$ is obtained as

$$\mathrm{ESDT}^{\circ}\mathrm{C} = 11.85 \times 5.79 + 4.89 = 73.5^{\circ}\mathrm{C}. \quad (12)$$

The ESDT of 73.5°C does not satisfy the PDTR. Explorer, therefore, goes to the Reduce_System_DPC Flow to reduce system power consumption further. It starts with T_4, the least priority task, and keeps reducing the frequency of T_4 to reduce the task's DPC. It goes through 4 candidate system configurations (CSCs) as shown in Table 5(b) and finally settles at configuration $T_2 - 0$, $T_5 - 0$, $T_0 - 0$, and $T_4 - 6$. Task T_4 now operates at 30 MHz and occupies 5 slots to provide the same performance of 240 fps. This configuration has ESTPC = 5.48 W, and ESDT = 69.79°C. This candidate satisfies all the constraints and is therefore selected as the system configuration. It occupies all the 8 slots, as shown in Figure 13(a), and maintains the performance of all the tasks. With this configuration, the system can function without recharging for 8.76 hours.

7.2. Case 2: Hardware Fault. After 15 minutes, there is a hardware fault in the slot where a component of critical task T_2 is operating, reducing N_s to 7. The Constraint Set changes to Table 6(a). Explorer is invoked, and it goes to the Hardware_Fault Flow. Since there is no spare slot available, one needs to be created to relocate the affected task component. Explorer reduces the CPP of T_4 to 120 fps and selects variant $T_4 - 7$ as listed in Table 6(b), which operates at 30 MHz and occupies 3 slots to provide a performance of 120 fps. This creates 2 spare slots, and the affected task component is relocated to a spare slot, as shown in Figure 13(b). The resultant configuration has ESTPC = 5.09 W and ESDT = 65.17°C. It satisfies all the constraints except for the PDTR. In this case, the priority of Explorer is to avoid the faulty slot and keep all the tasks functioning without failing the PTPC requirement. The selected configuration is the best possible solution that can satisfy these requirements. Therefore, PDTR is sacrificed until the time the faulty slot is recovered, since it has the least priority among all the parameters in this case.

7.3. Case 3: Hardware Fault Restoration. Within the next 15 minutes, the fault is identified as a transient fault and is rectified (e.g., by scrubbing technique). The affected slot is back in the system as a spare slot, changing N_s to 8 again. The Constraint Set changes to Table 7(a). Explorer is invoked again, and it goes to the Hardware_Fault_Recovery Flow to bring back the configuration that existed prior to the fault from system memory. Thus, CSC2, as shown in Table 7(b) is selected and reconfigured so that T_4 can operate at its maximum performance as seen in Figure 14(a). Since the original system configuration is restored, which has

Table 4: Flow of events discussed in Section 7.

Case no.	Time elapsed (hours)	System mode	Battery capacity (%)	Required lifetime (hours)	PTPC (W)	PDTR (°C)	ESTPC (W)	ESDT (W)	Achieved lifetime (hours)
1	0	M_0	100.00	8	6	68–70	5.48	69.79	8.76
2	0.25	M_0	97.15	7.75	6	68–70	5.09	65.17	9.17
3	0.25	M_0	94.5	7.5	6	68–70	5.48	69.79	8.28
4	0.5	M_0	88.79	7	6	64–66	5.09	65.17	8.38
5	0.5	M_0	83.49	8.25	4.86	61–63	4.83	62.12	8.30
6	0.25	M_1	81	8	4.86	61–63	4.81	61.89	8.08
7	0.25	M_1	78.47	6.75	5.6	64–66	5.15	65.87	7.32
8	1	M_1	67.75	5.75	5.6	66–68	5.28	67.49	6.16

Table 5: Constraint Set and CSCs evaluated for case 1.

(a) Constraint Set				
Mode	PTPC (W)	PDTR (°C)		N_s
M_0	6	68–70		8
(b) Sequence of CSCs evaluated				
	CSC1	CSC2	CSC3	CSC4
Case 1	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 0$	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 1$	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 3$	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 6$
ESTPC (W)	5.794	5.629	5.529	5.477
ESDT (°C)	73.55	71.6	70.41	69.79
No. of slots	4	5	6	8

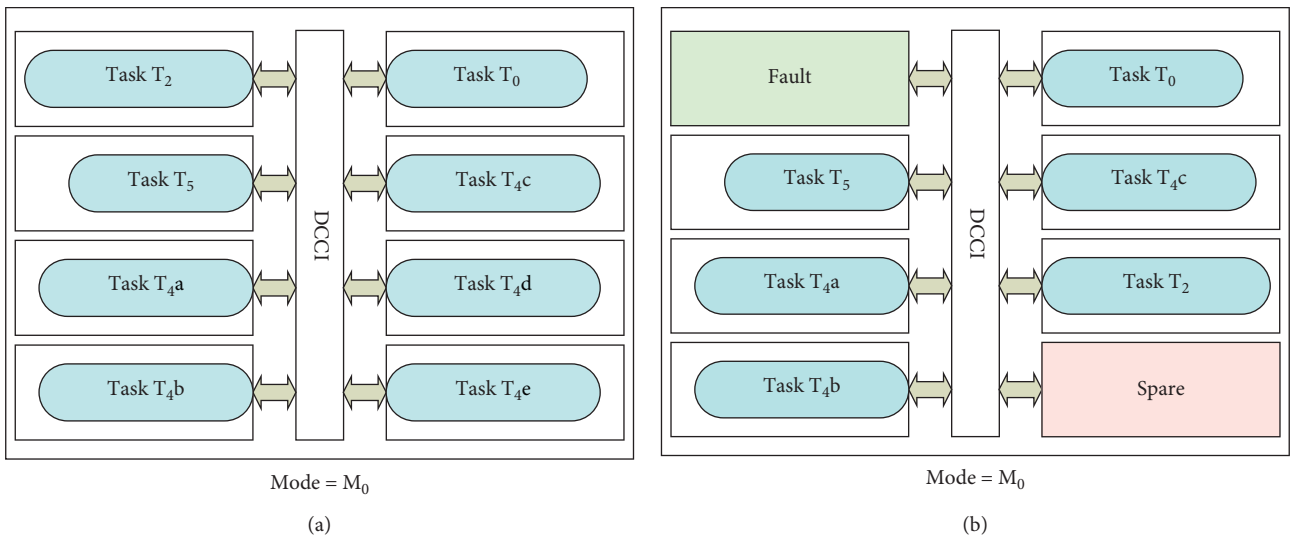

Figure 13: RTSA using Explorer for cases 1 and 2. (a) Selected system configuration for case 1. (b) Selected system configuration for case 2.

Table 6: Constraint Set and CSCs evaluated for case 2.

(a) Constraint Set			
Mode	PTPC (W)	PDTR (°C)	N_s
M_0	6	68–70	7
(b) Sequence of CSCs evaluated			
	CSC1	CSC2	
Case 2	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 6$	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 7$	
ESTPC (W)	5.48	5.09	
ESDT (°C)	69.79	65.17	
No. of slots	8	6	

Table 7: Constraint Set and CSCs evaluated for case 3.

	(a) Constraint Set		
Mode	PTPC (W)	PDTR (°C)	N_s
M_0	6	68–70	8
	(b) Sequence of CSCs evaluated		
	CSC1		CSC2
Case 3	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 7$		$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 6$
ESTPC (W)	5.09		5.48
ESDT (°C)	65.17		69.79
No. of slots	6		8

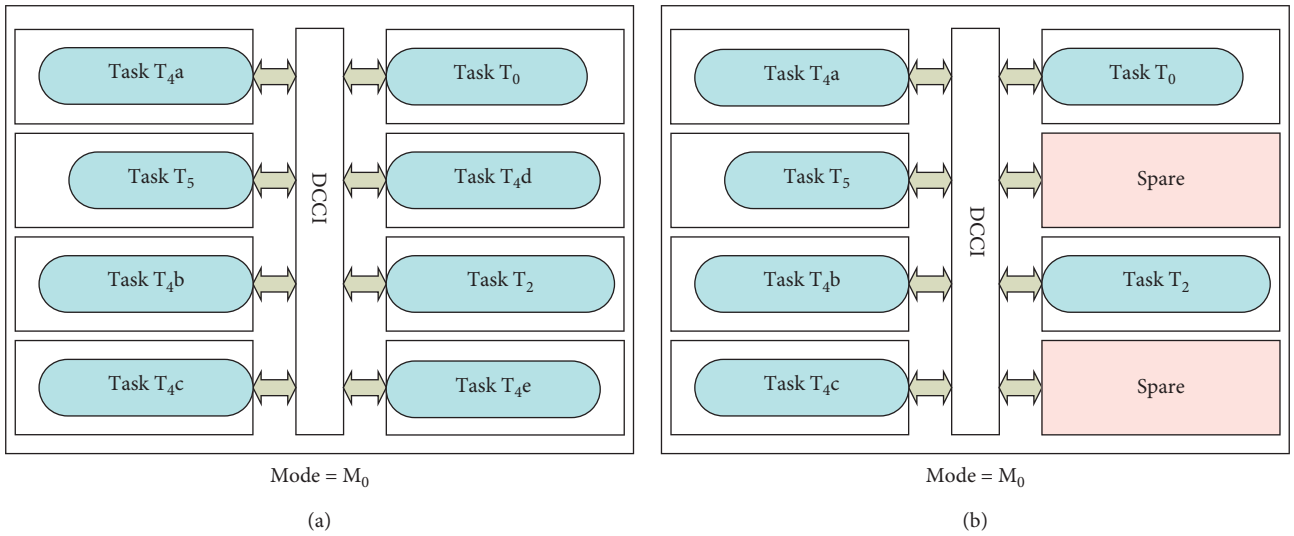

Figure 14: RTSA using Explorer for cases 3 and 4. (a) Selected system configuration for case 3. (b) Selected system configuration for case 4.

ESTPC = 5.48 W and ESDT = 69.79°C, all the constraints of the Constraint Set are satisfied.

7.4. Case 4: PDTR Drop. After half an hour, from Table 4, the PDTR drops to 64–66 (°C) and the PTPC continues to remain at 6 W. The Constraint Set changes to Table 8(a). Since the current configuration does not satisfy the PDTR, Explorer goes to Reduce_System_DPC Flow to find a configuration with lower ESTPC and hence lower ESDT. Since T_4 is already at its minimum frequency, Explorer reduces the performance of T_4. It selects $T_4 - 7$ which operates at 30 MHz and occupies 3 slots to provide a performance of 120 fps. The resultant configuration CSC2 as listed in Table 8(b) has ESTPC = 5.09 W and ESDT = 65.17°C. This candidate satisfies both the PTPC and PDTR constraints and is therefore selected as the new system configuration as shown in Figure 14(b).

7.5. Case 5: PTPC and PDTR Drop. After another half hour, as shown in Table 4, there is a drop in the power budget and die temperature constraint. If the power budget had not changed, it would have meant that expected system lifetime is 6.5 hours. However, due to external conditions, the system is now supposed to continue operating without recharging for 8.25 hours, which brings down the PTPC to 4.86 W. The PDTR also drops to 61–63°C. Thus, the Constraint Set changes to Table 9(a). The current system configuration has STPC = 5.09 W, which is greater than the new PTPC and will be able to sustain the system for only 7.88 hours. Explorer therefore goes to Reduce_System_DPC Flow to adapt to the situation. Since T_4 is at its l_{spec} and operating at its minimum frequency in the current configuration, Explorer reduces the frequency of T_0 to 120 MHz and selects variant $T_0 - 1$. The resultant configuration still does not satisfy the PTPC constraint. Reducing the frequency to 60 MHz also does not satisfy the PTPC. When the frequency is further reduced to 30 MHz, Explorer needs to select $T_0 - 7$, which occupies 5 slots to maintain the performance at 240 fps. Since enough spare slots to accommodate this variant are not available, Explorer goes to the Space_Adjustment Flow. T_0 has $h_{spec} = l_{spec} = 240$ fps, and so the performance of T_0 cannot be reduced further. Explorer therefore tries to reduce performance of higher priority tasks T_5 and T_2. However, they also have $h_{spec} = l_{spec} = 240$ fps, and it is not possible to reduce their performance. Thus, the least priority task T_4 needs to be removed. Since a task is removed, Explorer goes to the Increased_Power_Budget Flow to check the possibility

Table 8: Constraint Set and CSCs evaluated for case 4.

(a) Constraint Set			
Mode	PTPC (W)	PDTR (°C)	N_s
M_0	6	64–66	8
(b) Sequence of CSCs evaluated			
	CSC1	CSC2	
Case 4	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 6$	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 7$	
ESTPC (W)	5.48	5.09	
ESDT (°C)	69.79	65.17	
No. of slots	8	6	

Table 9: Constraint Set and CSCs evaluated for case 5.

(a) Constraint Set					
Mode	PTPC (W)	PDTR (°C)		N_s	
M_0	4.86	61–63		8	
(b) Sequence of CSCs evaluated					
	CSC1	CSC2	CSC3	CSC4	CSC5
Case 5	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$ $T_4 - 7$	$T_2 - 0$ $T_5 - 0$ $T_0 - 1$ $T_4 - 7$	$T_2 - 0$ $T_5 - 0$ $T_0 - 3$ $T_4 - 7$	$T_2 - 0$ $T_5 - 0$ $T_0 - 1$	$T_2 - 0$ $T_5 - 0$ $T_0 - 0$
ESTPC (W)	5.09	5.04	5	4.69	4.83
ESDT (°C)	65.17	64.67	64.14	60.52	62.12
No. of slots	6	7	8	4	3

of a drastic drop in the ESTPC. Since the Explorer's current frequency for T_0 was 60 MHz, Explorer now increases its frequency to 120 MHz and sets its CPP to 240 fps by selecting $T_0 - 1$. The resultant configuration has ESTPC = 4.7 W, lower than the PTPC of 4.86 W. Explorer therefore again increases the frequency of T_0; it selects $T_0 - 0$, which operates at 240 MHz and provides a performance of 240 fps. The resultant configuration has ESTPC = 4.83 W, still lower than PTPC. The next step would be to add the removed task; however, it was removed once during the flow is hence not added back. The exploration ends at $T_0 - 0$. All the candidates evaluated in this case are listed in Table 9(b). The final configuration of $T_2 - 0$, $T_5 - 0$, and $T_0 - 0$ has ESTPC = 4.83 W and ESDT = 62.12°C. It satisfies both, the PTPC and the PDTR, and is therefore selected and configured as the new system configuration as shown in Figure 15.

7.6. Case 6: Mode Change. After 15 minutes, there is a request for a mode change to M_1, formed by tasks T_3, T_2, T_5, and T_1. The PTPC continues to be 4.86 W, and the PDTR also remains 61–63°C. The Constraint Set for this case is shown in Table 10(a). Explorer goes to the Mode_Change Flow to find a suitable configuration that satisfies the task performance and system constraints. It goes though the candidates as shown in Table 10(b). The final configuration is $T_3 - 0, T_2 - 0, T_5 - 0$, and $T_1 - 7$. Tasks T_3, T_2, and T_5 operate a maximum performance of 240 fps, and T_1 operates at a reduced performance of 120 fps. This configuration has ESTPC = 4.81 W and ESDT = 61.89°C. It satisfies both, the PTPC and PDTR, and is therefore selected and configured as the new system configuration as shown in Figure 16.

7.7. Case 7: PTPC and PDTR Rise. After 15 minutes, the situations improve and the system lifetime before next recharge reduces to 6.75 hours. Thus, the power budget improves to 5.58 W. The PDTR also rises to 64–66°C. The Constraint Set changes to Table 11(a). Explorer goes to Increased_Power_Budget Flow. It goes on increasing the frequency of T_1, as shown in Table 11(b), until it reaches variant $T_1 - 0$, which operates at 240 MHz to give a performance of 240 fps. The ESTPC of the resultant configuration is 5.28 W. Since it is not possible to increase the DPC further, this configuration is chosen as the candidate that satisfies the PTPC. However, since its ESDT = 67.46 (°C), it fails in the Temperature_Analysis Flow. Explorer, therefore, goes to the Reduce_System_DPC Flow and selects $T_1 - 1$, which operates at 120 MHz to give a performance of 240 fps. The resulting candidate $T_3 - 0, T_2 - 0, T_5 - 0$, and $T_1 - 1$ has ESTPC = 5.15 W and ESDT = 65.87°C. It satisfies all the constraints and is therefore selected and configured as the system configuration as shown in Figure 17(a).

7.8. Case 8: Increased PDTR. After 1 hour, the PDTR increases to 66–68°C. The Constraint Set changes to Table 12(a). Explorer goes to Increased_Power_Budget Flow and increases the frequency of T_1 to 240 MHz by selecting $T_1 - 0$ as listed in Table 12(b). The final configuration, as shown in Figure 17(a), has ESTPC = 5.28 W, and ESDT = 67.5°C. The resulting candidate $T_3 - 0, T_2 - 0, T_5 - 0$, and $T_1 - 0$ satisfies all the constraints and is therefore selected as the system configuration.

Thus, the above scenarios demonstrate that a MACROS-based system deployed with Explorer can adapt in run-time to changing system mode, power budget, die temperature, and hardware resource constraints by dynamically choosing a suitable configuration such that the critical tasks of the desired system mode operate at their maximum performance, noncritical tasks operate within a permitted performance range, the system's power budget (increased or decreased) is satisfied, its die temperature is within or close to the specified range, and the total area occupied is less than or equal to the available number of slots.

8. Analysis

8.1. Storage Requirements. Explorer has been implemented as a bare-metal C code on the ARM Cortex-A9 processor of the Zynq XC7Z020 device, operating at 666 MHz. The implementation covers the example discussed in Section 7. Since each of the six tasks in the example has ten ASP circuit variants, the Variant-LUT stores the operating frequency, performance, number of slots, Logic slices, BRAM slices, and DSP slices used, for only $6 \times 10 = 60$ variants, irrespective of the modes. If a decision space of system configurations was used, characteristics of 3 modes

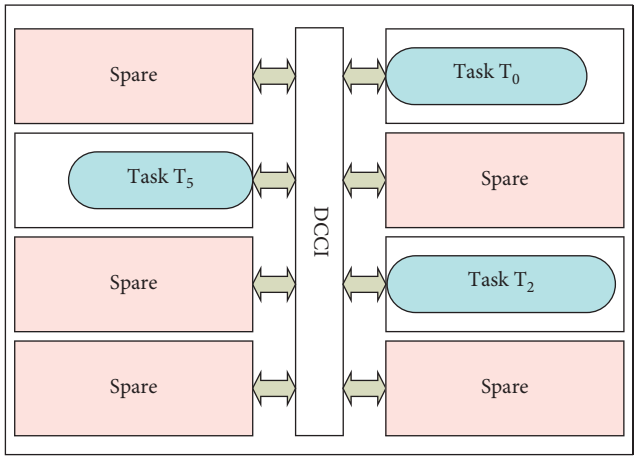

Figure 15: Selected system configuration for RTSA using Explorer for case 5.

Table 10: Constraint Set and CSCs evaluated for case 6.

	(a) Constraint Set				
Mode	PTPC (W)	PDTR (°C)			N_s
M_1	4.86	61–63			8
	(b) Sequence of CSCs evaluated				
	CSC1	CSC2	CSC3	CSC4	CSC5
Case 6	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 0$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 1$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 3$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 6$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 7$
ESTPC (W)	5.28	5.15	5.08	5.03	4.81
ESDT (°C)	69.49	65.87	65.07	64.53	61.89
No. of slots	4	5	6	8	6

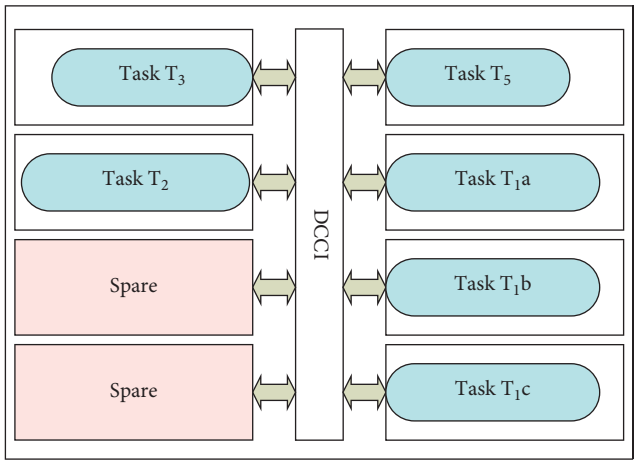

Figure 16: Selected system configuration for RTSA using Explorer for case 6.

$\times 10^6$ configurations per mode would need to be stored in the Variant-LUT. The LUT size with the proposed method is only 0.002% of the size when a decision space of system configurations is used.

Table 11: Constraint Set and CSCs evaluated for case 7.

	(a) Constraint Set				
Mode	PTPC (W)	PDTR (°C)			N_s
M_1	5.58	64–66			8
	(b) Sequence of CSCs evaluated				
	CSC1	CSC2	CSC3	CSC4	CSC5
Case 7	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 7$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 3$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 1$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 0$	$T_3 - 0$ $T_2 - 0$ $T_5 - 0$ $T_1 - 1$
ESTPC (W)	4.81	5.08	5.15	5.28	5.15
ESDT (°C)	61.89	65.07	65.87	67.49	65.87
No. of slots	6	6	5	4	5

8.2. Execution Time. Execution time of Explorer running on the ARM Cortex-A9 core has been recorded for different worst-case scenarios of the example discussed in Section 7 to observe the time taken in making the worst-case algorithmically taxing decisions. It is to be noted that Explorer is implemented on the ARM Cortex-A9 processor of the Zynq XC7Z020 device since the example system in Section 7 is considered to be developed on the Zynq XC7Z020 device. Any other hard/soft-core processor can be chosen to implement Explorer depending on the FPGA/SoC device that a system is developed on. The execution time of Explorer will be according to the processor chosen. However, it can be said that using any equivalent or advanced processor core will produce similar or better results.

8.3. Case 1: Worst-Case PTPC Drop or PDTR Drop. In the case of either a power budget and/or die temperature drop, Explorer resorts to the same solution, reducing the system's power consumption. In the worst-case scenario, RTSA would require changing the system's configuration from the one consisting of tasks operating at their highest frequency and highest performance h_{spec} to the one having only the critical tasks with their EC = 1 operating at their lowest frequency and lowest performance l_{spec}. Consider mode M_1 as an example. The initial configuration is $T_3 - 0$, $T_2 - 0$, $T_5 - 0$, and $T_1 - 0$, each ASP circuit variant occupying one slot, operating at 240 MHz at 240 fps. A heavy dip in the PTPC and/or PDTR condition results in a configuration comprising of only $T_3 - 6$ occupying 5 slots and providing a performance of 240 fps at 30 MHz. To reach this stage, Explorer needs to try reducing the frequency and CPP of each task starting from the lowest priority, then decide to remove each task one after the other, and then reach the last possible system configuration of $T_3 - 6$ alone, which has the lowest power consumption. Explorer takes only 102.59 μs to reach this conclusion.

8.4. Case 2: Worst-Case PTPC Rise and/or PDTR Rise. The longest decision-making process in the case of an increased power budget and/or required die temperature needs to be evaluated here. It must be noted that for worst-case decision-making in this case, there should be a big jump in both, PTPC and PDTR. If there is a drastic increase in only one parameter, the process of exploration will stop

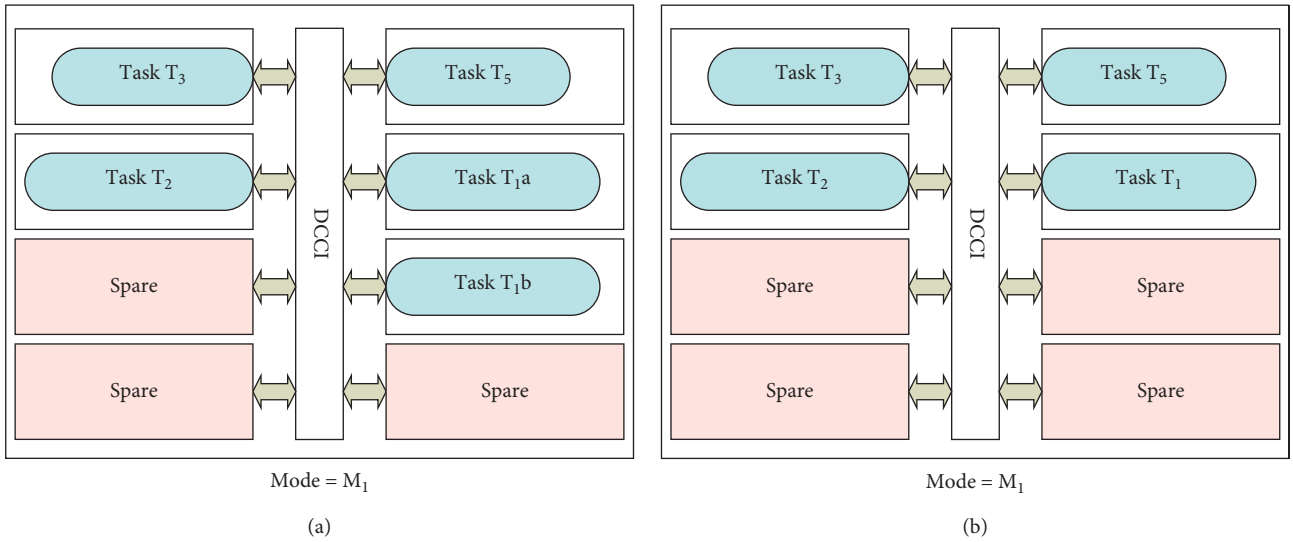

Figure 17: RTSA using Explorer for cases 7 and 8. (a) Selected system configuration for case 7. (b) Selected system configuration for case 8.

Table 12: Constraint Set and CSCs evaluated for case 8.

(a) Constraint Set			
Mode	PTPC (W)	PDTR (°C)	N_s
M_1	5.58	66–68	8
(b) Sequence of CSCs evaluated			
	CSC1	CSC2	
Case 8	$T_3 - 0$	$T_3 - 0$	
	$T_2 - 0$	$T_2 - 0$	
	$T_5 - 0$	$T_5 - 0$	
	$T_1 - 1$	$T_1 - 0$	
ESTPC (W)	5.15	5.28	
ESDT (°C)	65.87	67.49	
No. of slots	5	4	

sooner than the worst-possible exploration. For example, if there is a big rise in PTPC, but not in the PDTR, when Explorer tries to find a candidate configuration that has an increased ESPTC, it may fail the PDTR. This will force Explorer to select a configuration with lower ESTPC, thus not allowing it to explore all possible candidates and reach the one that closely satisfies the increased PTPC. Similarly, since PTPC has a higher priority over PDTR, if there is a big rise in PDTR alone, candidates closer to satisfying the PDTR will not be explored since their ESTPC will not satisfy the PTPC. Therefore, for noting the worst-case execution time, a big rise in both, PTPC and PDTR, is considered here.

Consider mode M_1; the current system configuration consists of only $T_3 - 6$, occupying 5 slots and providing a performance of 240 fps at 30 MHz. A big rise in the PTPC and PDTR results in a configuration $T_3 - 0$, $T_2 - 0$, $T_5 - 0$, and $T_1 - 1$; tasks T_3, T_2, and T_5 providing a performance of 240 fps at 240 MHz and T_1 providing a performance of 240 fps at 120 MHz. To reach this decision, Explorer first evaluates T_3 at increasing frequency and maximum performance. It then adds the task T_2 and evaluates it at increasing frequency and maximum performance. This repeats for tasks T_5 and T_1. Explorer finally meets the condition ESTPC > PTPC for configuration $T_3 - 0, T_2 - 0, T_5 - 0$, and $T_1 - 0$. It then again reduces the frequency of T_1 to 120 MHz and settles at $T_1 - 1$ to satisfy both PTPC and PDTR. The Execution time for this case is $\approx 83.46\,\mu$s. It must be noted in the case of a maximum increase in power budget, Explorer settles at $T_1 - 0$ itself. It does not return to $T_1 - 1$ again to decrease the system's power consumption. Hence, in this case, the execution time is less than $83.46\,\mu$s. Maximum increase in PTPC is therefore not considered as the worst-case increased power budget condition from the point of view of run-time decision-making.

8.5. Case 3: Worst-Case Mode Change. Suppose while the system is functioning in the mode M_0, it experiences the worst power budget drop. In this situation, there is also an interrupt to change the mode M_1. In this case, Explorer carries out RTSA by finding a system configuration for M_1 that satisfies the worst PTPC condition. This is a superset of Case 1; Explorer first finds the configuration $T_3 - 0$, $T_2 - 0$, $T_5 - 0$, and $T_1 - 0$, goes on reducing power consumption, and finally settles at $T_3 - 6$ alone. Time recorded for this case is $129.84\,\mu$s. This can be expected since the execution time recorded for a mode change to M_1 with the selected system configuration as $T_3 - 0, T_2 - 0, T_5 - 0$, and $T_1 - 0$ is $25.97\,\mu$s and the time recorded for a worst-case power budget drop is $102.59\,\mu$s as mentioned in Case 1.

8.6. Case 4: Worst-Case Hardware Fault. Assume that in mode M_1, $h_{\text{specs}} = l_{\text{specs}}$ for all the tasks. The current system configuration is $T_3 - 0$, $T_2 - 0$, $T_5 - 0$, and $T_1 - 6$; all the slots are occupied to maintain the performance of all tasks at their h_{spec}. If there is hardware fault in such a case, Explorer tries to reduce the CPP of each task starting from T_1 up to T_3 and finally decides to remove $T_1 - 6$ since performance of no task can be reduced to create a spare slot for relocating the task component in the faulty slot. The time recorded for this is $21.39\,\mu$s.

From all the cases considered, the maximum execution time observed is 129.84 μs. Comparing this decision-making time with the reconfiguration time of one slot: if the PCAP configuration port providing 145 MB/sec [82] is used, reconfiguring a partial bitstream of size say 492.8 KB, which is one-eighth the size of a full bitstream for Zynq XCZ7020 SoC, in a PRR, is 3.3 ms. It is possible to see that the decision-making time is less than 4% of one slot reconfiguration period; Explorer can make a decision for RTSA even before a slot is reconfigured! The observed execution time is obtained when four tasks per mode with 10 ASP circuit variants each are considered; that is, a maximum of 40 ASP circuit variants can be explored. Considering a larger system with 25 tasks per mode and 64 ASP circuit variants each, a maximum of 1600 ASP circuit variants will be explored. Extrapolating the observed worst-case execution time to this large system results in 5.2 ms. Comparing this time with an application requirement; consider an application involving video processing at 120 fps. It requires 8.33 ms to process a frame. Explorer's decision-making time is less than 0.65 times the time taken to process a frame. This means there will be a loss of only one frame to find a solution even for such a large system. A loss of one frame can be tolerated by most applications. The method thus proves to be suitable to carry out a system's structural adaptation at run-time for many applications even with quite strict performance constraints.

9. Conclusion

This paper presents Decision Space Explorer, a run-time decision-making method to enable multiobjective RTSA in FPGA/SoC-based autonomous and mobile systems supporting multitask multimode applications, thus making them self-sustainable. Whenever there is a change in a system's mode, power budget, die temperature, and/or hardware resource constraints, Explorer evaluates potential system configurations and selects the appropriate ASP circuit variants for the tasks at their respective appropriate operating frequencies such that they satisfy all the constraints. This means that the selected configuration fits in the available hardware resources, the critical tasks of the system's required mode continue to operate at their required performance, the other noncritical tasks operate within the permitted performance specifications, the system's STPC is lower than the power budget, and its SDT is maintained within the permitted range! To find such a system configuration, the proposed method explores the decision space of ASP circuit variants of individual tasks instead of a decision space of entire system configurations, thus drastically reducing the number of configurations to be explored and analyzed, and hence the exploration time. Also, it evaluates potential system configurations using dynamic run-time mathematical power consumption and die temperature estimation models, which have a very small execution time. Implementation of Explorer on the ARM Cortex-A9 core of Zynq XZ7Z020 device shows that the total execution time to select the final configuration in the worst-case scenario for an example system, i.e., where longest possible algorithmic processing would be required, is $\approx 130\,\mu$s. Extrapolating it to larger systems still results in an execution time in the order of ms. This demonstrates the suitability of Explorer to be used by systems in run-time for self-survival against multiple dynamic system constraints.

Data Availability

The data used to support the findings of this study are available in the manuscript.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] N. Wiener, *Cybernetics: Or the Control and Communication in the Animal and the Machine*, The MIT Press, Cambridge, MA, USA, 1965.

[2] D. Sharma, L. Kirischian, and V. Kirischian, "Run-time adaptation method for mitigation of hardware faults and power budget variations in space-borne FPGA-based systems," in *Proceedings of the 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Pasadena, CA, USA, 2017.

[3] D. Sharma, L. Kirischian, and V. Kirischian, "Run-time mitigation of power budget variations and hardware faults by structural adaptation of FPGA-based multi-modal SoPC," *Computers*, vol. 7, p. 52, 2018.

[4] D. Sharma, V. Kirischian, and L. Kirischian, "On-chip thermal balancing using dynamic structural adaptation of FPGA-based multi-task SoPCs for space-borne applications," in *Proceedings of the 2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Colchester, UK, 2019.

[5] V. Dumitriu, L. Kirischian, and V. Kirischian, "Mitigation of variations in environmental conditions by SoPC architecture adaptation," in *Proceedings of the 2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Montreal, Canada, 2015.

[6] A. D. Pimentel, "Exploring exploration: a tutorial introduction to embedded systems design space exploration," *IEEE Design & Test*, vol. 34, no. 1, pp. 77–90, 2017.

[7] M. Belwal and T. S. B. Sudarshan, "A survey on design space exploration for heterogeneous multi-core," in *Proceedings of the 2014 International Conference on Embedded Systems (ICES)*, Coimbatore, India, 2014.

[8] C. Piotr and A. Jaszkiewicz, "Pareto simulated annealing," in *Proceedings of the 12th International Conference on Multiple Criteria Decision Making*, pp. 297–307, Hagen, Germany, 1997.

[9] G. Palermo, C. Silvano, and V. Zaccaria, "Discrete particle swarm optimization for multi-objective design space exploration," in *Proceedings of the 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, Parma, Italy, 2008.

[10] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 911–924, 2010.

[11] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the 10th International Symposium on Hardware/Software Codesign. CODES 2002*, Estes Park, CO, USA, 2002.

[12] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for Pareto-optimal configurations in parameterized system-on-a-chip," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10, no. 4, pp. 416–422, 2002.

[13] C. Pilato, D. Loiacono, F. Ferrandi, P. L. Lanzi, and D. Sciuto, "High-level synthesis with multi-objective genetic algorithm: a comparative encoding analysis," in *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, Hong Kong, China, 2008.

[14] F. Ferrandi, P. L. Lanzi, D. Loiacono, C. Pilato, and D. Sciuto, "A multi-objective genetic algorithm for design space exploration in high-level synthesis," in *Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI*, Montpellier, France, 2008.

[15] P. van Stralen and A. Pimentel, "Fitness prediction techniques for scenario-based design space exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1240–1253, 2013.

[16] D. Sharma and L. Kirischian, "A method for run-time prediction of on-chip thermal conditions in dynamically reconfigurable SoPCs," *International Journal of Reconfigurable Computing*, vol. 2021, Article ID 8818788, 20 pages, 2021.

[17] V. Dumitriu and L. Kirischian, "SoC self-integration mechanism for dynamic reconfigurable systems based on collaborative macro-function units," in *Proceedings of the 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 2013.

[18] V. Dumitriu and L. Kirischian, "SoPC self-integration mechanism for seamless architecture adaptation to stream workload variations," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 2, pp. 799–802, 2016.

[19] V. Dumitriu, *A framework and method for the run-time on-chip synthesis of multi-mode self-organized reconfigurable stream processors*, Ph.D. thesis, Ryerson University, Toronto, Canada, 2015.

[20] D. Sharma, V. Dimitriu, and L. Kirischian, "Architecture reconfiguration as a mechanism for sustainable performance of embedded systems in case of variations in available power," *Applied Reconfigurable Computing (ARC 2017)*, Springer, Berlin, Germany, 2017.

[21] H. Tabkhi and G. Schirner, "Application-guided power gating reducing register file static power," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 22, no. 12, pp. 2513–2526, 2014.

[22] M. Hosseinabady and J. L. Nunez-Yanez, "Run-time power gating in hybrid ARM-FPGA devices," in *Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Munich, Germany, 2014.

[23] D. You and K.-S. Chung, "Quality of service-aware dynamic voltage and frequency scaling for embedded GPUs," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 66–69, 2015.

[24] M. U. K. Khan, M. Shafique, and J. Henkel, "Power-efficient workload balancing for video applications," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 6, pp. 2089–2102, 2016.

[25] G. Kornaros and D. Pnevmatikatos, "Dynamic power and thermal management of NoC-based heterogeneous MPSoCs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 1, pp. 1–26, 2014.

[26] S. D. Carlo, G. Gambardella, P. Prinetto, D. Rolfo, and P. Trotta, "Satta," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 8, no. 1, pp. 1–22, 2015.

[27] Y. H. Lu, L. Benini, and G. De Micheli, "Low-power task scheduling for multiple devices," in *Proceedings of the 8th International Workshop on Hardware/Software Codesign*, San Diego, CA, USA, 2000.

[28] P. Yang, P. Marchal, C. Wong et al., "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *Proceedings of the 15th International Symposium on System Synthesis, 2002*, Kyoto, Japan, 2002.

[29] M. Qiu, Z. Chen, L. T. Yang, X. Qin, and B. Wang, "Towards power-efficient smartphones by energy-aware dynamic task scheduling," in *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, Liverpool, UK, 2012.

[30] K. Ganeshpure and S. Kundu, "Performance-driven dynamic thermal management of MPSoC based on task rescheduling," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 2, pp. 1–33, 2014.

[31] L. Ost, M. Mandelli, G. M. Almeida et al., "Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 3, pp. 1–22, 2013.

[32] A. Rodríguez, J. Valverde, C. Castañares, J. Portilla, E. de la Torre, and T. Riesgo, "Execution modeling in self-aware FPGA-based architectures for efficient resource management," in *Proceedings of the 2015 10th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, Bremen, Germany, 2015.

[33] K.-W. Lin and Y.-S. Chen, "Online thermal-aware task placement in three-dimensional field-programmable gate arrays," in *Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems*, Prague, Czech Republic, 2015.

[34] XILINX, *XAPP1088: Correcting Single Event Upsets in Virtex-4 FPGA Configuration Memory*, XILINX, San Jose, CA, USA, 2009.

[35] C. Bolchini, A. Miele, and C. Sandionigi, "A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 60, no. 12, pp. 1744–1758, 2011.

[36] R. Salvador, A. Otero, J. Mora, E. d. l. Torre, L. Sekanina, and T. Riesgo, "Fault tolerance analysis and self-healing strategy of autonomous, evolvable hardware systems," in *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, 2011.

[37] M. Abramovici, M. A. Breuer, and A. D. Friedman, "Index," in *Digital Systems Testing and Testable Design*, pp. 647–652, Wiley-IEEE Press, Hoboken, NJ, USA, 2009.

[38] H. Zhang, L. Bauer, M. A. Kochte et al., "Module diversification: fault tolerance and aging mitigation for runtime reconfigurable architectures," in *Proceedings of the 2013 IEEE International Test Conference (ITC)*, Anaheim, CA, USA, 2013.

[39] X. Iturbe, K. Benkrid, A. T. Erdogan et al., "R3TOS: a reliable reconfigurable real-time operating system," in *Proceedings of the 2010 NASA/ESA Conference on Adaptive Hardware and Systems*, Anaheim, CA, USA, 2010.

[40] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, T. Arslan, and I. Martinez, "Runtime scheduling, allocation, and execution of real-time hardware tasks onto Xilinx FPGAs subject to fault occurrence," *International Journal of Reconfigurable Computing*, vol. 2013, Article ID 905057, 32 pages, 2013.

[41] A. Biedermann, S. A. Huss, and A. Israr, "Safe dynamic reshaping of reconfigurable MPSoC embedded systems for

self-healing and self-adaption purposes," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 8, no. 4, pp. 1–22, 2015.

[42] V. Dumitriu, L. Kirischian, and V. Kirischian, "Run-time recovery mechanism for transient and permanent hardware faults based on distributed, self-organized dynamic partially reconfigurable systems," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2835–2847, 2016.

[43] V. Dumitriu, L. Kirischian, and V. Kirischian, "Decentralized run-time recovery mechanism for transient and permanent hardware faults for space-borne FPGA-based computing systems," in *Proceedings of the 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Leicester, UK, 2014.

[44] A. Vallero, A. Carelli, and S. Di Carlo, "Trading-off reliability and performance in FPGA-based reconfigurable heterogeneous systems," in *Proceedings of the 2018 13th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, Taormina, Italy, 2018.

[45] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta, and A. Vallero, "A novel methodology to increase fault tolerance in autonomous FPGA-based systems," in *Proceedings of the 2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, Platja d'Aro, Spain, 2014.

[46] S. D. Carlo, P. Prinetto, and A. Scionti, "A FPGA-based reconfigurable software architecture for highly dependable systems," in *Proceedings of the 2009 Asian Test Symposium*, Taichung, Taiwan, 2009.

[47] S. Di Carlo, A. Miele, P. Prinetto, and A. Trapanese, "Microprocessor fault-tolerance via on-the-fly partial reconfiguration," in *Proceedings of the 2010 15th IEEE European Test Symposium*, Prague, Czech Republic, 2010.

[48] G. B. Wigley and D. A. Kearney, "Research issues in operating systems for reconfigurable computing," in *Proceedings of the 2002 International Conference on Engineering of Reconfigurable System and Algorithms*, pp. 10–16, Las Vegas, NV, USA, 2002.

[49] M. Eckert, D. Meyer, J. Haase, and B. Klauer, "Operating system concepts for reconfigurable computing: review and survey," *International Journal of Reconfigurable Computing*, vol. 2016, Article ID 2478907, 11 pages, 2016.

[50] X. Iturbe, K. Benkrid, C. Hong et al., "R3TOS: a novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on FPGAs," *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1542–1556, 2013.

[51] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, and T. Arslan, "Microkernel architecture and hardware abstraction layer of a reliable reconfigurable real-time operating system (R3TOS)," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 8, pp. 1–35, 2015.

[52] H. K.-H. So and R. Brodersen, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 2, pp. 1–28, 2008.

[53] D. Göhringer, M. Hübner, E. N. Zeutebouo, and J. Becker, "CAP-OS: operating system for runtime scheduling, task mapping and resource management on reconfigurable multiprocessor architectures," in *Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Atlanta, GA, USA, 2010.

[54] A. Agne, M. Happe, A. Keller et al., "ReconOS: an operating system approach for reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 60–71, 2014.

[55] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Designing an operating system for a heterogeneous reconfigurable SoC," in *Proceedings of the 2003 International Parallel and Distributed Processing Symposium*, Nice, France, 2003.

[56] M. D. Santambrogio, V. Rana, and D. Sciuto, "Operating system support for online partial dynamic reconfiguration management," in *Proceedings of the 2008 International Conference on Field Programmable Logic and Applications*, Heidelberg, Germany, 2008.

[57] K. Jozwik, H. Tomiyama, M. Edahiro, S. Honda, and H. Takada, "Rainbow: an OS extension for hardware multitasking on dynamically partially reconfigurable FPGAs," in *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, 2011.

[58] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[59] J. A. Clemente, I. Beretta, V. Rana, D. Atienza, and D. Sciuto, "A mapping-scheduling algorithm for hardware acceleration on reconfigurable platforms," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, pp. 1–27, 2014.

[60] R. Pellizzoni and M. Caccamo, "Real-time management of hardware and software tasks for FPGA-based embedded systems," *IEEE Transactions on Computers*, vol. 56, no. 12, pp. 1666–1680, 2007.

[61] P.-A. Hsiung, C.-H. Huang, J.-S. Shen, and C.-C. Chiang, "Scheduling and placement of hardware/software real-time relocatable tasks in dynamically partially reconfigurable systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, pp. 1–32, 2010.

[62] D. De Sensi, M. Torquati, and M. Danelutto, "A reconfiguration algorithm for power-aware parallel applications," *ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 4, pp. 1–25, 2016.

[63] É. Sousa, F. Hannig, J. Teich, Q. Chen, and U. Schlichtmann, "Runtime adaptation of application execution under thermal and power constraints in massively parallel processor arrays," in *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*, Sankt Goar, Germany, 2015.

[64] M. Ullmann, W. Jin, and J. Becker, "Hardware enhanced function allocation management in reconfigurable systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, CO, USA, 2005.

[65] G. Wassi, M. E. A. Benkhelifa, G. Lawday, F. Verdier, and S. Garcia, "Multi-shape tasks scheduling for online multitasking on FPGAs," in *Proceedings of the 2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, Montpellier, France, 2014.

[66] S. M.-K. Gueye, E. Rutten, and J.-P. Diguet, "Autonomic management of missions and reconfigurations in FPGA-based embedded system," in *Proceedings of the 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Pasadena, CA, USA, 2017.

[67] K. Vipin and S. A. Fahmy, "Mapping adaptive hardware systems with partial reconfiguration using CoPR for Zynq," in *Proceedings of the 2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Montreal, Canada, 2015.

[68] K. Loukil, N. Ben Amor, and M. Abid, "Self adaptive reconfigurable system based on middleware cross layer adaptation model," in *Proceedings of the 2009 6th International*

Multi-Conference on Systems, Signals and Devices, Djerba, Tunisia, 2009.

[69] W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria, "A sensitivity-based design space exploration methodology for embedded systems," *Design Automation for Embedded Systems*, vol. 7, no. 1-2, pp. 7–33, 2002.

[70] V. Kathail, S. Aditya, R. Schreiber, B. Ramakrishna Rau, D. C. Cronquist, and M. Sivaraman, "PICO: automatically designing custom computers," *Computer*, vol. 35, no. 9, pp. 39–47, 2002.

[71] G. Snider, "Spacewalker: automated design space exploration for embedded computer systems," Technical report HPL-2001-220, Hewlett-Packard Laboratories, Palo Alto, CA, USA, 2001.

[72] B. C. Schafer and K. Wakabayashi, "Divide and conquer high-level synthesis design space exploration," *ACM Transactions on Design Automation of Electronic Systems*, vol. 17, no. 3, pp. 1–19, 2012.

[73] G. Mariani, V. Sima, G. Palermo, V. Zaccaria, C. Silvano, and K. Bertels, "Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures," in *Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2012.

[74] S. M. Logesh, D. S. H. Ram, and M. C. Bhuvaneswari, "Multi-objective optimization of power, area and delay during high-level synthesis of DFG's—a genetic algorithm approach," in *Proceedings of the 2011 3rd International Conference on Electronics Computer Technology*, Kanyakumari, India, 2011.

[75] M. Holzer, B. Knerr, and M. Rupp, "Design space exploration with evolutionary multi-objective optimisation," in *Proceedings of the 2007 International Symposium on Industrial Embedded Systems*, Costa da Caparica, Portugal, 2007.

[76] P. V. Huong and N. N. Binh, "An approach to design embedded systems by multi-objective optimization," in *Proceedings of the 2012 International Conference on Advanced Technologies for Communications*, Hanoi, Vietnam, 2012.

[77] J. Madsen, T. Stidsen, P. Kjærulff, and S. Mahadevan, "Multi-objective design space exploration of embedded system platforms," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, vol. 225, pp. 185–194, Springer, Berlin, Germany, 2006.

[78] G. Palermo, C. Silvano, and V. Zaccaria, "ReSPIR: a response surface-based Pareto iterative refinement for application-specific design space exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 12, pp. 1816–1829, 2009.

[79] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano, "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *Proceedings of the 47th Design Automation Conference*, Anaheim, CA, USA, 2010.

[80] Xilinx Inc, *Vivado Design Suite User Guide Partial Reconfiguration*, Xilinx, San Jose, CA, USA, 2018.

[81] Intel, *Intel Quartus Prime Pro Edition User Guide Partial Reconfiguration*, Intel, Santa Clara, CA, USA, 2021.

[82] Xilinx Inc, *Zynq-7000 SoC Technical Reference Manual*, Xilinx, San Jose, CA, USA, 2021.