computers

MDPI

Article

Run-Time Mitigation of Power Budget Variations and Hardware Faults by Structural Adaptation of FPGA-Based Multi-Modal SoPC

Dimple Sharma [1,*], Lev Kirischian [1] and Valeri Kirischian [2]

[1] Electrical and Computer Engineering Department, Ryerson University, Toronto, ON M5B 2K3, Canada; lkirisch@ee.ryerson.ca

[2] MDA Corporation, Brampton, ON L6S 0B6, Canada; valeri.kirischian@mdacorporation.com

* Correspondence: dsharma@ee.ryerson.ca; Tel.: +1-647-401-1444

Received: 13 July 2018; Accepted: 7 October 2018; Published: 11 October 2018

Abstract: Systems for application domains like robotics, aerospace, defense, autonomous vehicles, etc. are usually developed on System-on-Programmable Chip (SoPC) platforms, capable of supporting several multi-modal computation-intensive tasks on their FPGAs. Since such systems are mostly autonomous and mobile, they have rechargeable power sources and therefore, varying power budgets. They may also develop hardware faults due to radiation, thermal cycling, aging, etc. Systems must be able to sustain the performance requirements of their multi-task multi-modal workload in the presence of variations in available power or occurrence of hardware faults. This paper presents an approach for mitigating power budget variations and hardware faults (transient and permanent) by run-time structural adaptation of the SoPC. The proposed method is based on dynamically allocating, relocating and re-integrating task-specific processing circuits inside the partially reconfigurable FPGA to accommodate the available power budget, satisfy tasks' performances and hardware resource constraints, and/or to restore task functionality affected by hardware faults. The proposed method has been experimentally implemented on the ARM Cortex-A9 processor of Xilinx Zynq XC7Z020 FPGA. Results have shown that structural adaptation can be done in units of milliseconds since the worst-case decision-making process does not exceed the reconfiguration time of a partial bit-stream.

Keywords: run-time structural adaptation; run-time design space exploration; multi-task multi-modal FPGA-based systems

1. Introduction

Modern autonomous embedded systems are expected to be capable of high performance computing and also executing several such high performance tasks on a single platform. They are, therefore, mostly implemented using SoPC platforms due to the advantages they offer [1–4]. While processing of the algorithmically intensive tasks of the supported applications can be carried out on the sequential processors of the SoPC (hard-core processors), the computation-intensive tasks can execute as hardware tasks on FPGAs to provide the requisite high performance. This trend of high performance computing can be observed in several domains, from commercial applications like Global Positioning Systems (GPS), driver assistance, robotic systems etc. to critical military and aerospace applications. However, the increasing complexity and requirements of applications has resulted in the applications having multiple modes of operation, where a mode can be referred to as a set of tasks and their specifications that remain unchanged for a certain period of time. Changes in the functionality, number, priorities and/or performance specifications of the tasks imply a mode change for the application. Consequently, systems supporting multi-task workloads are now also expected to be able to support dynamic changes in their workload.

Considering autonomous and mobile systems deployed for critical applications; they also face dynamic variations in the environmental conditions like available system power budget or hardware resources. Available power budget can vary as the power consumption of tasks in different modes varies. Power budget can also vary depending on the factors that affect the power sources like rechargeable batteries. Some of these factors are: the charging sources like wind or solar energy, operating temperature, number of charge/discharge cycles, etc. Available hardware resources can vary due to occurrence of transient or permanent hardware faults which may arise due to radiations, thermal cycling, vibration/acceleration intensity, hidden manufacturing defects, operation in hazardous environments or aging. Such dynamic environmental changes can impact the performance of the executing workload. Systems must be able to maintain the performance of their critical multi-modal tasks even in presence of dynamically varying environmental constraints. Since, in general, variations in both, workload and environmental conditions are unpredictable, complex autonomous and mobile systems must be able to adapt to the changing constraints at run-time!

Consider a space-borne system, say a satellite in an orbit. It operates on rechargeable batteries, which depend on solar energy. The set of tasks it will execute will depend on events like its position in orbit, intensity of solar storm, external requests (e.g., communication with mission center), etc. In such a case, it cannot be considered that power budget will change very slowly. For example, suppose the system is in a certain mode, performing a certain set of tasks. It encounters an unexpected event or receives a request for system mode change such that it is now required to run a new set of tasks which may have a significantly different power consumption as compared to the previously executing set of tasks. In this case, the corresponding power budget change will occur in very short time (e.g., in range of milliseconds). Moreover, charging of the batteries, which depends on the available solar energy and the type of back-up power system used, could lead to changes in the power budget. The switching period to switch to the back-up power system and oscillations of the power transfer switch could also influence the power budget.

Thus, for autonomous and mobile systems, there are many cases when their power budget changes quite rapidly. There can also be simultaneous requirements for changes in system mode or occurrence of hardware faults. Systems need to be ready to adapt to all such dynamic changes at run-time to sustain the performance of their multi-task workloads and to avoid emergency shut-down.

A potential solution to this challenging requirement is the use of Run-time Structural Adaptation (RT-SA); a method with which a system can dynamically modify its SoPC architecture to mitigate changes in workload, environmental constraints and possible occurrence of faults. In other words, run-time structural adaptation in SoPC means changing the set of components and/or links between the on-chip components during the execution of tasks to optimize system performance to the dynamic environment (e.g., variations in power budget, temperature fluctuations, occurrence of faults, etc.) RT-SA enables: (a) dynamic selection of a suitable system architecture according to the existing set of constraints and (b) reconfiguration of the selected architecture in run-time. For a given hardware task's algorithm and its range of permitted performance specifications, several implementation variants can be obtained for that task, which exhibit different resource usage, operating frequency, performance, and hence power consumption [5]. They are referred as Application Specific Processing (ASP) circuit variants of that task. The ASP circuit variants of all system tasks can be stored in system memory as partial configuration bit-streams (or bit-files), which can reconfigured in the Partially Reconfigurable Regions (PRRs) of the FPGA using Dynamic Partial Reconfiguration (DPR). As conditions change, a suitable task variant which meets these conditions can be dynamically reconfigured to adapt to the situation. For example, in a low power budget scenario, a task variant occupying more hardware resources and operating at a reduced frequency can be reconfigured such that its performance is maintained and Dynamic Power Consumption (DPC) is reduced. Consider a system executing four tasks, each of which has ten ASP circuit variants. This results in a design space of 10^4 possible combinations of SoPC architectures. The system now has the flexibility to select one suitable system configuration, i.e., a combination of ASP circuit variants of the four tasks, depending on factors like

current workload (mode) of the system, available power budget, available hardware resources etc. RT-SA can thus enable systems to dynamically sustain the performance of their tasks within the permitted range in presence of changing workload, environmental conditions, and faults. Its practical application, however, has the following challenges:

1. In multi-task multi-modal systems, when the number of modes, tasks, and their ASP circuit variants increase, a large design space of system configurations is formed. For example, a system with a total of 16 tasks, 16 ASP circuit variants per task, 20 modes, and 5 tasks per mode will have a design space of 16^5 = 1,048,576 system configurations per mode. Since a solution must be found at run-time, within the permitted adaptation time, it may not possible to exhaustively evaluate each configuration at run-time. As a result, there must be a Run-time Design Space Exploration (RT-DSE) method with a small execution-time overhead to select a suitable configuration that satisfies the tasks' performance specifications, DPC and hardware resource constraints.
2. The RT-DSE method will need the DPC of candidate system configurations to decide the most suitable solution. It is practically not feasible to measure and store the DPC of all the possible system configurations in a Look-Up-Table (LUT). In the above example, this would mean measuring the DPC of 20 modes $\times 16^5$ configurations per mode during system design phase and feeding these values in a large LUT. Furthermore, any addition or modification of system modes, tasks, or their variants will imply re-doing the entire offline process all over again! Thus, it is necessary to have a run-time analytical model which can estimate the DPC of system configurations under evaluation during the run-time DSE process itself.
3. Once a solution is provided by the run-time DSE method, the system needs to be dynamically reconfigured with the new chosen ASP circuit variants of the active tasks within the permitted adaptation time. It is to be noted that the permitted adaptation time is application specific. For a commercial video processing application, a loss of a couple frames can be permitted, but for a critical military application, loss of only one frame may be permitted for adaptation. A system must therefore have the infrastructure that allows a quick transformation to the new selected configuration. 'Multi-mode Adaptive Collaborative Reconfigurable self-Organized System' (MACROS) framework has been developed for this purpose [6]. It permits reconfiguration and automatic integration of ASP circuit variants with a very small time overhead in the order of only a couple clock cycles [7,8]. A brief description of MACROS is provided in Section 3.

This paper is our effort in making progress towards practical deployment of RT-SA in autonomous and mobile FPGA-based systems. It has the following novel contributions:

1. It proposes a method for FPGA-based multi-task multi-modal systems for their run-time structural adaptation to an extensive set of possible situations of: (a) changing system modes, (b) changing power budgets, and (c) occurrence of hardware faults. It incorporates an RT-DSE mechanism which finds the most suitable system configuration depending on the existing set of constraints, thus enabling RT-SA.
2. It proposes a method to derive the complete Dynamic Power Consumption Estimation Model (DPCEM) of an FPGA in terms of all its reconfigurable resources; clock frequency, Logic slices, Block RAM (BRAM) slices, and DSP slices. The DPCEM is used by the RT-DSE method to evaluate DPC of potential configurations.

The paper is divided as follows: Section 2 discusses the current research in the field of run-time adaptation and brings out the importance of run-time structural adaptation. Section 3 is a brief description of the MACROS framework's architecture. Section 4 discusses the decision-making functionality of the run-time structural adaptation method. Section 5 presents the experimental setup for the DPCEM derivation method and outlines the derivation procedure using the example of Xilinx Zynq XC7Z020 and Kintex-7 XC7K325T FPGA devices. Section 6 shows how the run-time structural adaptation method uses the DPCEM to evaluate potential solutions during the adaptation process.

Section 7 considers a wide variety of changing constraints and explains how a system can dynamically take decisions and adapt to each scenario using the run-time structural adaptation method. Section 8 analyzes the LUT-storage and time overhead of the method based on its implementation on the ARM Cortex-A9 processor of the Zynq XC7Z020 FPGA. Section 9 concludes the paper.

2. Literature Review

Most systems based on SoPC platforms incorporate a real-time operating system (RTOS) or a management system on the lines of an RTOS to adapt to dynamic workloads, power budget, performance, temperature and/or fault constraints. Research efforts have resulted in a generalized concept for RTOSs and their basic functions, which are time and resource management for optimized multi-tasking. These functions mainly include task scheduling, task mapping and allocation, inter-task communication, task to RTOS communication, task configuration etc. [9,10]. Several RTOSs have been developed based on these concepts. They differ from each other based on the number and complexity of their functions, the mechanisms to carry out the functions, their implementation, all of which depend on the system structure that they are developed upon and the applications they are being used for. Refs [11–15] are some examples that support multi-tasking and workload management for hardware tasks, i.e., tasks executing on FPGAs. R3TOS [16,17], BORPH [18], CAP-OS [19], ReConOS [20] are RTOSs that serve systems with both hardware and software tasks (tasks executing on soft or hardcore processors). Some management systems [20–22] also support tasks that can have software and hardware versions of implementation. Such tasks can be dynamically relocated between software and hardware versions to ensure optimal resource management while maximum tasks are served and their deadlines are met. Several techniques have also been adopted for real-time power consumption and/or temperature management. A commonly used method for dynamic power management is power gating [23,24]; where portions of the configured circuits are turned off when they are not operating. Methods like Dynamic Voltage and Frequency Scaling (DVFS) and DFS are used to control power consumption [25,26] and temperature [27] and also to sustain task performances in presence of temperature variations [28]. Dynamic scheduling techniques [27,29–32] and dynamic mapping (or resource management) techniques [27,33–35] are other methods used to achieve power and/or thermal aware workload management. Dynamic scheduling and mapping techniques are also employed for fault mitigation [15,36,37]. To ensure reliability in mission-critical systems like space-borne systems, the emphasis is on protection against and recovery from transient or permanent faults due to radiation effects. Triple Modular Redundancy (TMR) and scrubbing [38–40] are the most common methods deployed for mitigating transient faults. Built-In Self-Test (BIST) procedures [41] or methods like device reprogramming to avoid damaged regions [42] are used to cater to permanent faults. Refs [7,8] propose a run-time relocation based mechanism with a very small time overhead to mitigate transient and permanent faults in FPGA-based systems deployed with the MACROS framework. The authors of [43] present a method for mitigating permanent faults in FPGA-based heterogeneous systems; several variants of the same task are stored such that they occupy different reconfigurable regions. The appropriate variant is configured when a fault is detected in a reconfigurable region. A method of relocating faulty computation or interconnection tiles to spare tiles to increase fault-tolerance is proposed in [44]. Ref [45] discusses a switching mechanism, where tasks are switched between their hardware/software versions in case of faults. Ref [46] presents a method using DPR to increase reliability in presence of faults in micro-processor based systems.

Although significant progress is seen in the development of run-time adaptive systems, the proposed solutions are not complete to support adaptation in dynamically varying environmental conditions like varying system power budget, temperature or occurrence of hardware faults. This is because the tasks that the systems manage have fixed implementation circuits. It is due to this fixed nature of tasks that systems can only re-schedule them (in time) and re-allocate or re-map them on different available resources (space). With this limited flexibility in the dimensions of time and space, it may be possible to adapt to dynamically varying workloads, but it may not be completely possible

to satisfy dynamically changing environmental constraints. This is because the adaptation techniques are applied to optimize a fixed parameter in presence of a fixed set of environmental constraints. For example, minimize power consumption with fixed task-performance and resource constraints. However, in the case of mobile and autonomous systems operating in non-static environments, the environmental constraints themselves vary and hence even the parameter to be optimized. For example, if power budget drops, system power consumption must be minimized in accordance with the new power budget constraint by trading off task performances. If hardware faults occur, resource utilization must be minimized. If tasks being executed are critical, performance should be maximized. Such a flexibility can only be achieved if a system is able to change its structure, i.e., use different task implementation circuits to accommodate dynamic variations in the set of constraints. Some research efforts have been observed in this direction. Ref [47] proposes selection of a suitable combination of number of processor cores, the clock frequency and the placement of software threads based on the performance and power consumption constraints of the system. Ref [48] discusses the use of software task variants for massively parallel processor arrays. Based on temperature changes, a different configuration of the same task, occupying a different number of processing elements is re-configured. The authors of [49] propose system adaptation using task variants varying from pure software to a mix of hardware and software implementations. The concept of using task variants is also gaining consideration in systems supporting purely hardware tasks. Ref [50] proposes the use of differently shaped variants for hardware tasks, which differ in performance and resource utilization. The focus of the algorithm, however, is only to improve FPGA resource utilization ratio and reduce task rejection ratio. It cannot therefore apply well to systems which have multiple varying constraints. The QoS-aware real-time management system presented in [51] uses tasks with implementation variants on different platforms like processors, FPGA, GPUs etc. Such a system can be useful specifically for heterogeneous systems only. Adaptation using variants of tasks and system configurations is discussed in [52,53] respectively. However, in both cases, the events that trigger the choice of a variant or a configuration are fixed. The system therefore cannot adapt to scenarios outside the pre-defined scope of events.

From a review of the literature, the following points can be noted: (a) Most of the dynamic adaptation methods mainly serve processor-centric systems. Even on FPGA-based SoPC platforms, the research focus is for task management on soft-core processors or tasks which execute as hardware accelerators for processors. (b) Most RTOSs are developed for tasks with deadlines, i.e., which run only for specific periods of time. (c) They are based on task attributes like arrival time, worst-case execution time, period, deadline, etc. The adaptation therefore becomes dependent on the nature of tasks. (d) In most power/thermal/fault-aware systems, the adaptation method caters to only one or two parameters and not all the constraints together. In practical scenarios, critical multi-task multi-modal systems usually run continuously executing stream processing hardware tasks and can face the conditions of varying power budgets, temperature, performance constraints, and hardware faults, all together! Thus, more research is required towards development of systems that can sustain their dynamic workload in varying environmental and fault conditions. Our initial efforts resulted in a method for systems with static workloads to dynamically adapt to depleting power budgets and fault conditions [54]. This paper proposes a run-time structural adaptation method for systems with dynamically varying task-sets and their performance constraints. It allows run-time adaptation to the changing system modes, increasing or decreasing power budgets, and mitigating hardware faults. Adaptation to varying temperature conditions can also be easily incorporated in the proposed method. This method can be integrated with an RTOS to form a decision-making RTOS that supports run-time system adaptation to a wide range of dynamically varying constraints.

3. MACROS Framework

Figure 1 shows the general architecture of the MACROS framework [6]. It is formed by: (a) several identical PRRs called "*slots*" on the FPGA, (b) a Distributed Communication and Control

Infrastructure (DCCI), and (c) a Bit-stream and Configuration Management system (BCM). A task variant's ASP circuit may consist of multiple components, each occupying one slot on the FPGA. Identical slots reduce management of reconfiguration of ASP circuit variants as their components can be reconfigured on any available slot. DCCI, a crossbar switch structure, permits seamless system communication. Inter-communication of task components to find their up-stream and down-stream partners, their self-synchronization and inter-connection also happens over the DCCI [6]. The DCCI thus ensures dynamic and automatic integration of the task components to form the complete ASP circuit of a task. The BCM extracts the required bit-streams from the system memory and configures them on available slots.

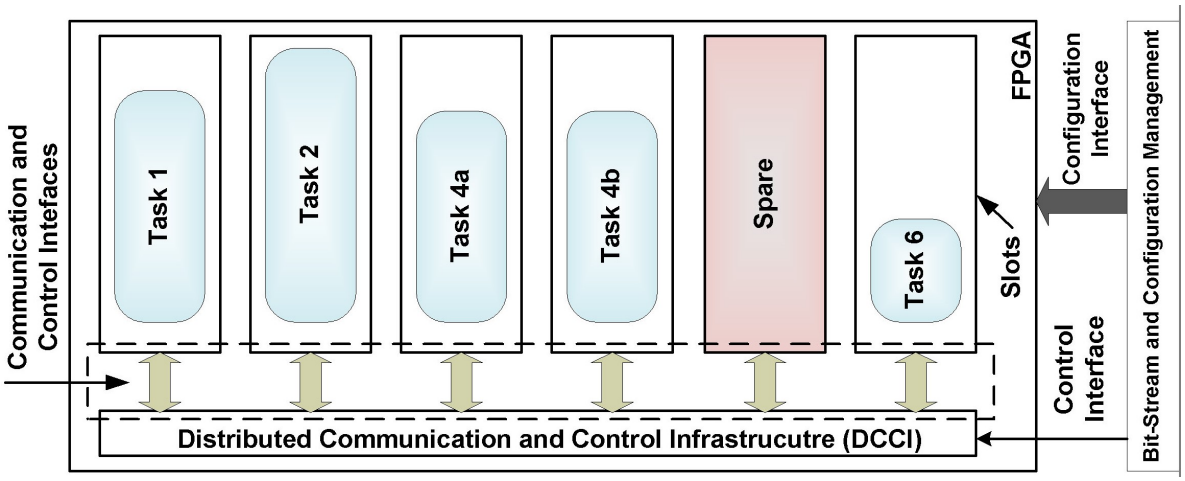

Figure 1. General architecture of MACROS framework.

For a MACROS framework-based system, when a new configuration is selected, the BCM configures the components of the chosen variant of every task in the available FPGA slots without any complex decision-making processes. The DCCI then dynamically interconnects the components that belong to the same task without the need for any additional time-consuming control processes. The new combination of task variants is ready to function with a very small time overhead [7,8], thus making the MACROS framework an essential element of run-time structural adaptation.

4. Method for Run-Time Structural Adaptation to Varying System Modes, Power Budget, and Occurrence of Hardware Faults

The proposed run-time structural adaptation method, termed as 'Explorer', is conceptually based on [54], which was an initial attempt to test a run-time adaptation mechanism on a static set of tasks. In real-life applications, mobile and autonomous systems supporting critical applications do not usually have a fixed set of executing tasks. The set of tasks, number of tasks, their performance specifications, priorities etc. can change based on the occurrence of some events. For example, a system on-board a satellite might need to carry out some tasks when it gets enough sunlight. This means the applications, or the multiple tasks running on the system are multi-modal. Run-time adaptation to environmental conditions like varying power budget or hardware faults etc. is much more complex in the case of multi-modal tasks as compared to when the set of tasks is fixed.

Furthermore, in [54], the only scenario considered with respect to adaptation to power budget was when the power budget is depleting. However, in field deployed systems, power budget can increase as well as decrease. For example, power budget of a space-borne system can increase when it is in the presence of sunlight, which allows charging up it's solar-energy dependent batteries. On the other hand, if the system suddenly needs to run a set of tasks from its sleep mode, if will face a quick drop in its power budget. Adaptation to scenarios when the power budget can vary; i.e., increase or

decrease, is different and more algorithmically intensive as compared to an adaptation mechanism only for depleting power budgets.

Thus, in this paper we are proposing a novel run-time adaptation mechanism for multi-task systems which can have multiple modes, and which can face situations of increasing or decreasing power budgets, and/or hardware faults. The mechanism is a look-up-table based run-time design space exploration mechanism, that finds a close-to-optimum variant for each active task in the given system mode, such that the existing constraints of power budget (increased or decreased), performance specifications of the tasks, available hardware resources, etc., all are met.

4.1. System Description

The Explorer enables run-time structural adaptation for multi-task multi-modal systems. For such systems, let T_j, $j = 0, 1, 2, \ldots, l$ represent the system tasks and M_i, $i = 0, 1, \ldots, n$ represent the system's mode number, where each mode is associated with a set of tasks that execute simultaneously while that mode is active. The number of tasks in a mode m is referred as N_m. Each task has the following attributes:

1. Priority of the task in a mode—P_0 (highest priority), ..., P_k (lowest priority)
2. Range of performance available for this task in a mode, i.e., from h_{spec}, the highest performance level (e.g., 240 frames per second (fps) => 8) to l_{spec}, the lowest performance level (e.g., 60 fps => 2). These values are relative not absolute and thus, can be associated with different performance characteristics.
3. Existence condition, EC, a parameter that determines whether a task in a mode can be eliminated or not. The task can be terminated if its $EC = 0$; not if its $EC = 1$.

The Explorer's behavior is based on specific LUTs, namely, 'Mode-specific LUT' ('MODE-LUT') and 'Variant-specific LUT' ('Variant-LUT'). This approach allows the fastest reaction on unpredicted events which may require minimum possible response time. Consider a multi-task multi-modal system having a total of 6 tasks, T_0 to T_5, and three modes, M_0 to M_2. Table 1 presents an example of MODE-LUT. It stores the tasks, their performance bounds and priorities for each mode. Task priority in the paper's context means the task's criticality, i.e., how important the functioning of a task is, with respect to the other tasks, when there is a need to either reduce some task's performance or eliminate a task in situations like low power budget, limited available hardware resources etc. For example, as seen in Table 1, in mode M_0 ($N_0 = 4$), T_5, the most critical task, has priority P_0, and T_4, the least critical task, has priority P_3. This means adaptation to varying environmental conditions will begin with the least critical task T_4. Since it has the least priority over the other tasks, its performance/functioning will be altered (within its range of specifications) first to adapt to the new set of constraints. MODE-LUT also lists the range of performance specifications, i.e., h_{spec} and l_{spec}, for every task in each mode. For example, in Table 1, T_0 has $h_{spec} = 8$ and $l_{spec} = 2$ in mode M_0, whereas it is a critical task with strict performance constraints, i.e., $h_{spec} = l_{spec} = 8$ in mode M_2. The h_{spec} and l_{spec} values in the Table 1 are a ratio with respect to the minimum performance instead of actual values. For example, if the tasks are video processing tasks, their performance will be in fps. If minimum performance is 30 fps, then in the above case, T_0 will have $h_{spec} = 240$ fps and $l_{spec} = 60$ fps in mode M_0, and $h_{spec} = lspec = 240$ fps in mode M_2.

Each of the six tasks has ten ASP circuit variants for run-time structural adaptation. Characteristics of the ASP circuit variant of all the tasks, i.e., their resource utilization, operating frequency, and performance are stored in the Variant-LUT, as shown in Table 2. In this table, only the task variants used for the discussion in Section 7 are listed. The Variant-LUT, comprising of all the variants of each of the six tasks, is shown in Table A1. A task with a particular performance can be implemented with different combinations of frequency and resource utilization. For example, in Table 2, variants $T_0 - 0$, $T_0 - 1$, and $T_0 - 2$ of task T_0, all have a performance of 8. However, $T_0 - 1$ and $T_0 - 2$ have half and one fourth the operating frequency of $T_0 - 0$, and occupy twice and four times the number of slots

as $T_0 - 0$ respectively. The choice of a variant depends on the existing set of constraints. Consider the example shown in Figure 2. The system frequency, F_{sys}, is initially at 240 MHz and $T_0 - 0$ has been configured. If the system's power budget reduces, F_{sys} can be reduced to 120 MHz to reduce the system's DPC. In order to maintain the performance of T_0, $T_0 - 1$ can be reconfigured instead of $T_0 - 0$. Similarly, if power budget further reduces, F_{sys} can be reduced to 60 MHz and $T_0 - 2$ can be reconfigured.

It must be noted that the set of tasks in a mode together result in the execution of the application supported by system, in that mode. As a result, all the tasks are assumed to execute at the same system frequency, i.e., F_{sys}. Thus, variants of tasks selected for adaptation are such that they all run at the F_{sys} selected at that point of time. For example, in the case of a video processing application, all the tasks will operate at the same F_{sys} to provide the same frame rate at a given resolution.

Table 1. Mode-LUT.

Mode	# 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_5		T_0		T_2		T_4	
		8	8	8	2	8	2	8	2
		1		0		0		0	
M_1	$N_1 = 3$	T_2		T_4		T_1			
		8	8	8	2	8	2		
		1		0		0			
M_2	$N_1 = 3$	T_0		T_1		T_3			
		8	8	8	8	8	2		
		1		1		0			

Table 2. Variant-LUT.

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	86	60
$T_0 - 2$	4	60	8	11,877	172	120
$T_0 - 3$	2	60	4	6062	86	60
$T_0 - 4$	1	60	2	3093	43	30
$T_1 - 0$	1	240	8	2061	22	82
$T_1 - 1$	2	120	8	4040	44	164
$T_1 - 2$	4	60	8	7914	88	328
$T_2 - 0$	1	240	8	5003	27	24
$T_2 - 1$	2	120	8	9806	54	48
$T_2 - 2$	2	60	4	9806	54	48
$T_2 - 3$	1	60	2	5003	27	24
$T_3 - 0$	1	240	8	4009	16	46
$T_3 - 1$	2	120	8	7858	32	92
$T_4 - 0$	1	240	8	5088	39	51
$T_4 - 1$	2	120	8	9972	78	102
$T_4 - 2$	1	60	2	5088	39	51
$T_5 - 0$	1	240	8	2567	33	73
$T_5 - 1$	2	120	8	5031	66	146
$T_5 - 2$	4	60	8	9857	132	292

Since each task has a range of performance specifications, it also has ASP circuit variants with different performance values. A variant with a lower performance will occupy lesser resources than another variant with a higher performance at the same F_{sys}. Thus, having a range of performance specifications enables continuous execution of the task instead of mere elimination if it is unable to fit in the available space at a fixed performance. For example, as seen in Table 1, in mode M_0, T_2 has $h_{spec} = 8$ and $l_{spec} = 2$. As seen in Figure 3, if its current performance is 4 and $F_{sys} = 60$ MHz, from Table 2, variant $T_2 - 2$, occupying 2 slots, needs to be configured. However, if only 1 slot is available in the FPGA, performance of T_2 can be reduced to 2, so that variant $T_2 - 3$, occupying 1 slot, can be configured. This way, T_2 can continue to execute at a reduced performance and its elimination due to lack of space is avoided. The current performance of a task is expressed by a parameter called Current Possible Performance (CPP). Since tasks can be eliminated to adapt to the existing constraints, the active number of tasks in a mode may not always be equal to N_m. A track of the active number of tasks is maintained by a parameter N_a.

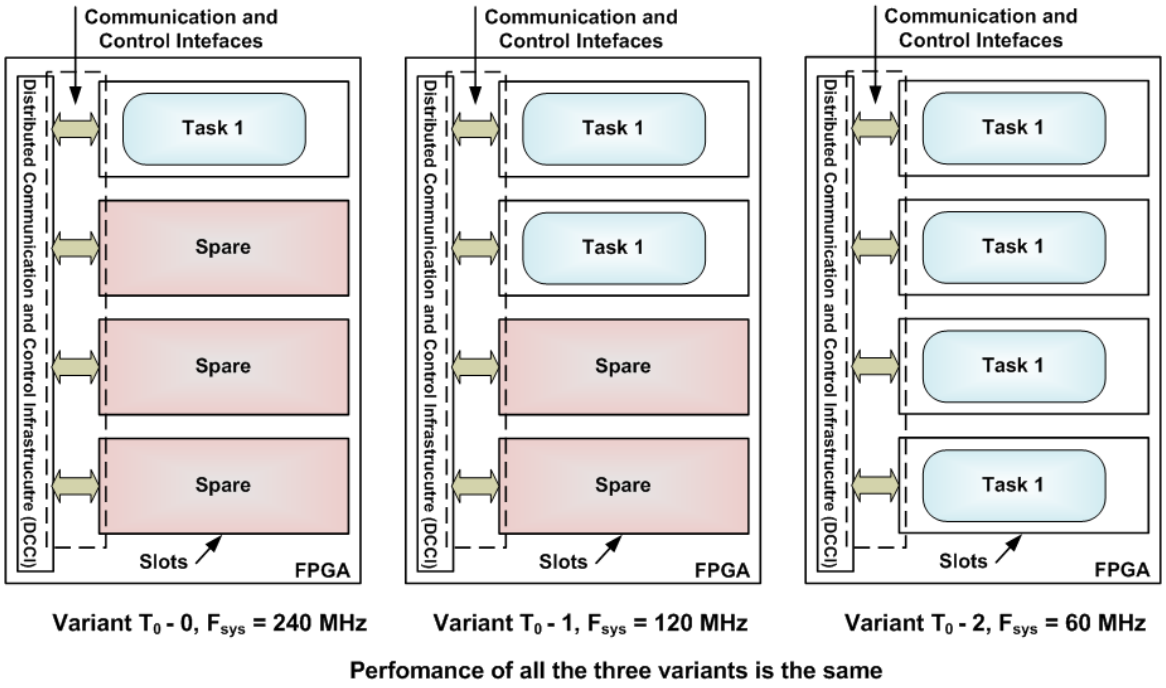

Figure 2. Example of a task's ASP circuit variants that have the same performance.

4.2. Decision-Making Run-Time Structural Adaptation Method

The Explorer monitors the current power budget at regular intervals of time and is also invoked when there is a hardware fault or a need for a system mode change. It takes different decision paths depending on the system mode, power budget, hardware resource and task performance constraints to dynamically select a system configuration which satisfies all the constraints. The system needs spare slots for adaptation; maintaining a task's performance while reducing F_{sys} or relocating a task component due to a hardware fault requires free slots. The Explorer therefore, always selects a configuration at the highest possible F_{sys} that meets the power budget to keep as many slots in reserve as possible.

Mode_Change Flow: The Explorer uses this decision path when the system begins in its default mode or if there is a need to switch to another mode while the system is functioning. As shown in the left side of Figure 4, it extracts the characteristics of the tasks in the required mode, m, from the Mode-LUT and estimates the system's Permitted DPC (PDPC) based on the available power budget. It sets F_{sys} to maximum, $N_a = N_m$, and the current possible performance, CPP, of every task in the mode m to its respective h_{spec} with the goal of finding a configuration where every task will operate at

its h_{spec} at the highest possible F_{sys}. Once the F_{sys} is set, the Explorer scans the Variant-LUT of every active task to find a suitable variant for it. It begins scanning the Variant-LUT of the task with highest priority P_0, and then continues for all the N_m tasks in the mode m. If a variant for a task is not found, it checks if that was because a potential variant could not fit in the available space. In that case, it goes to the *Space_Adjustment Flow* to cater to that issue. If that is not the case, there is an error. This can occur only if there is an error during the design phase of the system. Once variants for all the tasks are found, i.e., a candidate system configuration is selected, the explorer goes to the *Power_Analysis Flow* for checking if the configuration meets the power budget constraint.

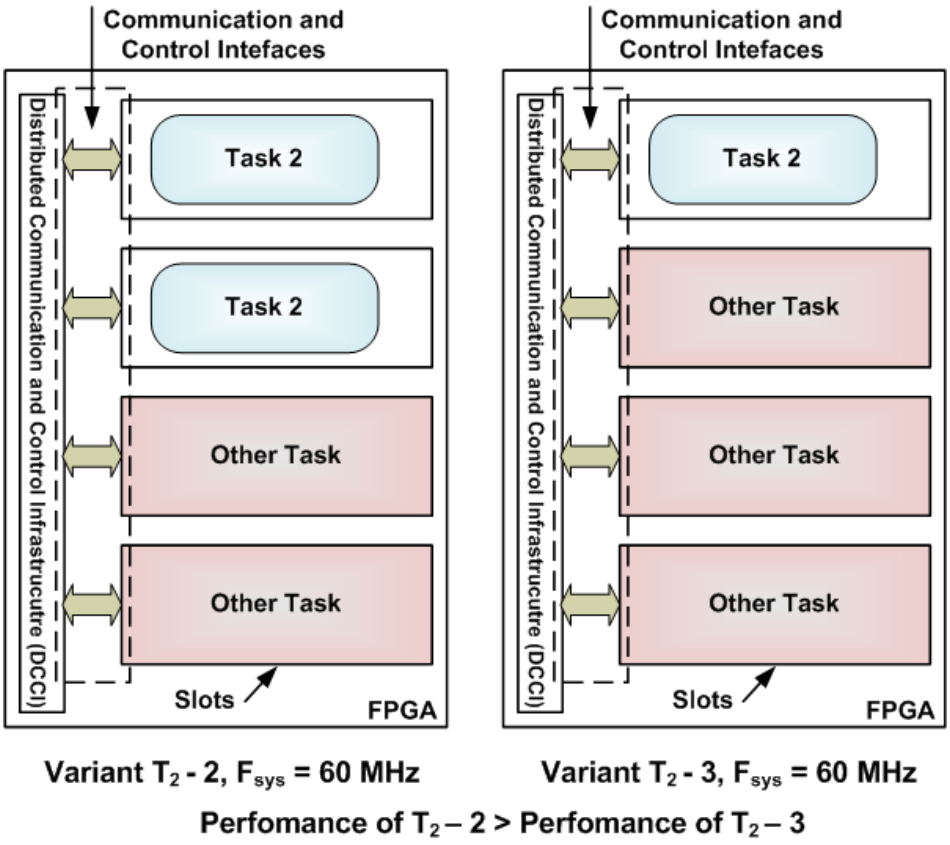

Figure 3. Example of a task's ASP circuit variants that have different performances.

Power_Analysis Flow: This flow is shown in the right side of Figure 4. The Explorer uses the run-time DPCEM discussed in Section 5 to estimate the DPC of the candidate configuration. If it's estimated DPC (*EDPC*) is lower than the PDPC, the set of ASP circuit variants is chosen as the new system configuration and the Explorer waits for the next instance of power budget. Otherwise, it goes to *Reduce_System_DPC Flow* in an attempt to reduce the system's DPC to adapt to the low power budget constraint.

Power_Budget_Check Flow: This flow, as shown in the left side of Figure 5, is used whenever it is time to check the system's power budget. This means, after using the *Mode_Change Flow* either due to system start up or request for changing the system's mode, the Explorer always comes to this flow at regular intervals of time for adaptation to the new monitored power budget. At the time of new power budget check, the Explorer estimates the system's *PDPC*. If the power budget has dropped with respect to the previous one, it checks the DPC of the current configuration (*CDPC*). If it meets the *PDPC* constraint, the Explorer retains the configuration and waits for the next instance of power budget. Otherwise, it goes to *Reduce_System_DPC Flow* in an attempt to reduce the system's DPC to adapt to the low power budget constraint.

When power budget increases as compared to the previous one, the Explorer sets $N_a = N_m$ and the CPP of every task in that mode to its respective h_{spec}. This is because, with the increased budget, it may now become possible to execute all the N_m tasks at their h_{spec}. If F_{sys} is not already maximum, the Explorer increases the F_{sys} by a step and finds suitable variants for all the tasks. If the new configuration has a lower $EDPC$ than the $PDPC$, F_{sys} is increased again and the process repeats. This continues until a configuration is found whose $EDPC$ exceeds the $PDPC$. The explorer now follows *Reduce_System_DPC Flow* and finally settles onto a configuration which has lower $EDPC$ than the $PDPC$. This process is needed to reach the highest possible F_{sys} for the available power budget. If the Explorer accepted a configuration in the first iteration itself, there could still be room for increasing some task's performance or adding in an eliminated task and it would have been missed. During the iterations, if a configuration has a lower $EDPC$ than the $PDPC$ even at maximum F_{sys}, it is accepted as a solution since the tasks are executing in their best possible configuration.

Reduce_System_DPC Flow: This decision flow is shown in right side of Figure 5. Since the system's DPC needs to be reduced due to a reduced power budget condition, the Explorer reduces F_{sys} by one step, finds a new configuration for the active set of tasks at that F_{sys}, and proceeds to the *Power_Analysis Flow*. If F_{sys} is already at its minimum, it goes to *Reduce_System_DPC_Minimum_*F_{sys} *Flow* in an attempt to reduce the system's DPC.

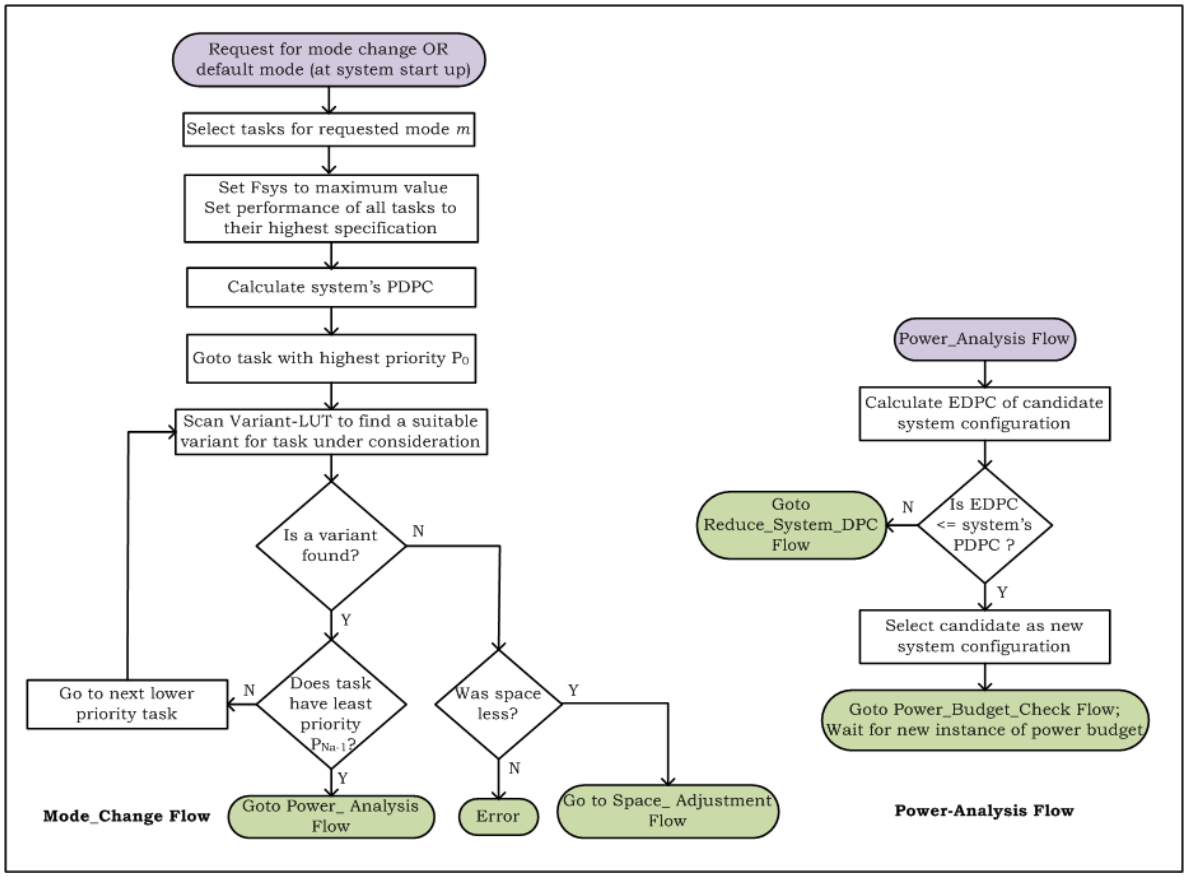

Figure 4. Mode_Change and Power_Analysis Flows of the Explorer.

Space_Adjustment Flow: This flow is shown in Figure 6. When F_{sys} is reduced, the Explorer needs to select ASP circuit variants requiring more number of slots to maintain the performance of every task. Hence, it is possible that even after using the spare slots, all the tasks may not be accommodated. If a task with priority P_s does not fit in the available space, the Explorer tries to reduce it's CPP so that a variant which occupies lesser number of slots can be chosen. If it is already at it's l_{spec}, the Explorer saves the task priority P_s and then tries to reduce the CPP of a higher priority task to make

room for the task under consideration. When successful, it comes back to the task with priority P_s in an attempt to accommodate it in the created space. However, if CPP of any task cannot be reduced even after reaching the task with highest priority P_0, the Explorer eliminates the task with priority P_s if its $EC = 0$ and throws an error if its $EC = 1$. If the task with priority P_s is eliminated, the Explores moves on to the *Power_Analysis Flow*. If the task with priority P_s is accommodated and is the last task, i.e., the least priority task, the Explorer proceeds to the *Power_Analysis Flow*. If not, it moves on with finding variants of the remaining tasks.

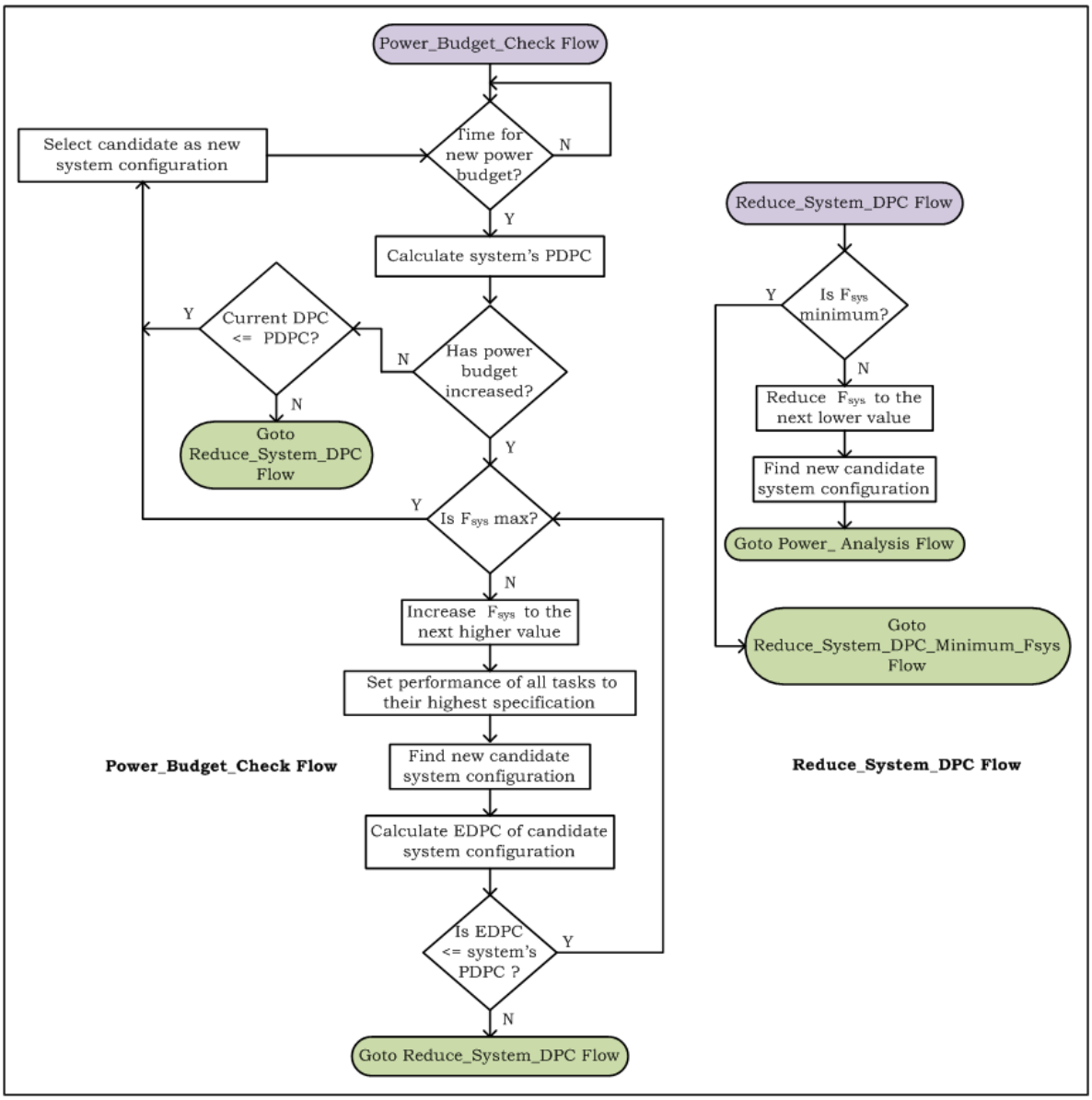

Figure 5. Power_Budget_Check and Reduce_System_DPC Flows of the Explorer.

Reduce_System_DPC_Minimum_F$_{sys}$ Flow: This flow is shown in Figure 7. When F_{sys} is at itś minimum, the Explorer tries to reduce the system DPC by reducing the CPP of a task, starting from the one with the least priority (P_{N_a-1}). If the task is already at its l_{spec}, the Explorer tries to reduce the CPP of a higher priority task. If it is not possible to do so even after reaching the task with highest priority P_0, the Explorer eliminates the least priority task after verifying its EC. It then proceeds to the *Power_Analysis Flow*.

Hardware_Fault Flow: The Explorer chooses the same flow as shown in Figure 7 when a hardware fault occurs in a slot. The fault mitigation method is based on the run-time component relocation method discussed in [7,8]. The basic idea is to relocate the affected task component to a spare slot. This process always takes the same time, which is the slot reconfiguration time. The affected task component thus has a fast recovery. The faulty slot can then be tested for transient or permanent faults, while the affected task component is already up and functioning.

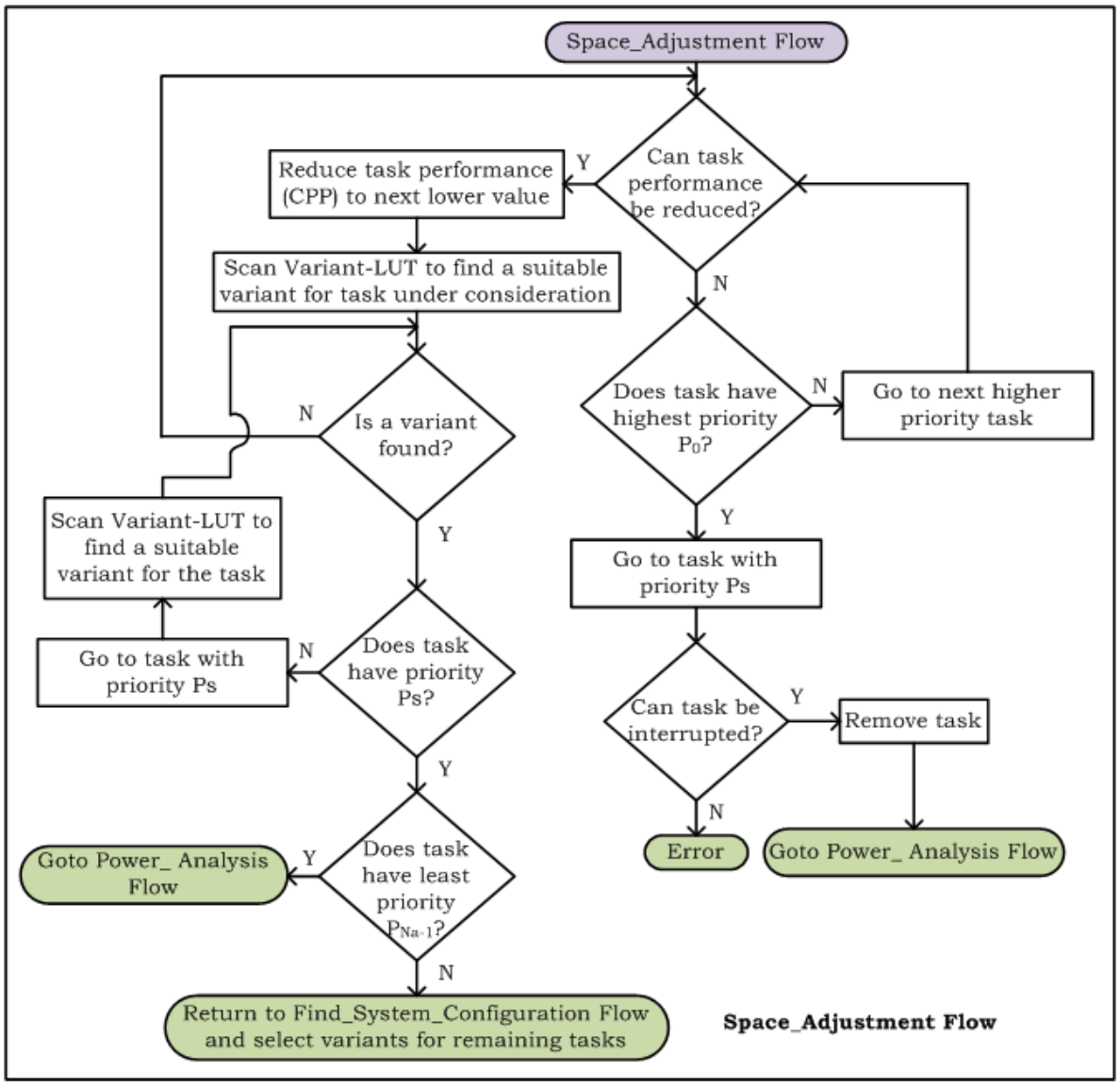

Figure 6. Space_Adjustment Flow of the Explorer.

For the kind of systems discussed here, a spare slot may not always be available for relocation in the case of a fault. In such a case, a spare slot will need to be created to cater to the situation. When a hardware fault is detected in a slot, the Explorer checks the availability of a spare slot. If found, it reconfigures the affected task component to that slot. If there is no spare slot, it tries to create one by reducing the *CPP* of a task, starting from the one with the least priority. If that is not possible even after reaching the task with highest priority P_0, the Explorer eliminates the least priority task after verifying its *EC* and configures the affected task component in the new available spare slot. In the case of a transient fault, this adaptation is temporary; only until the affected slot is restored. However, when the fault is permanent, the available number of slots permanently reduce for the system.

Find_System_Configuration Flow: The Explorer comes to this flow, shown in Figure 8, whenever there is a need to find a candidate configuration for the system, i.e., find a combination of suitable variants of the active set of tasks. The Explorer starts with the task with highest priority P_0. It finds a suitable variant for the task that satisfies the existing set of constraints, using the *Find_Task_Variant Flow*, and moves on to scan the LUT of the next lower priority task. This continues till the task with least priority. The selected variants of all the tasks form the candidate configuration of the system.

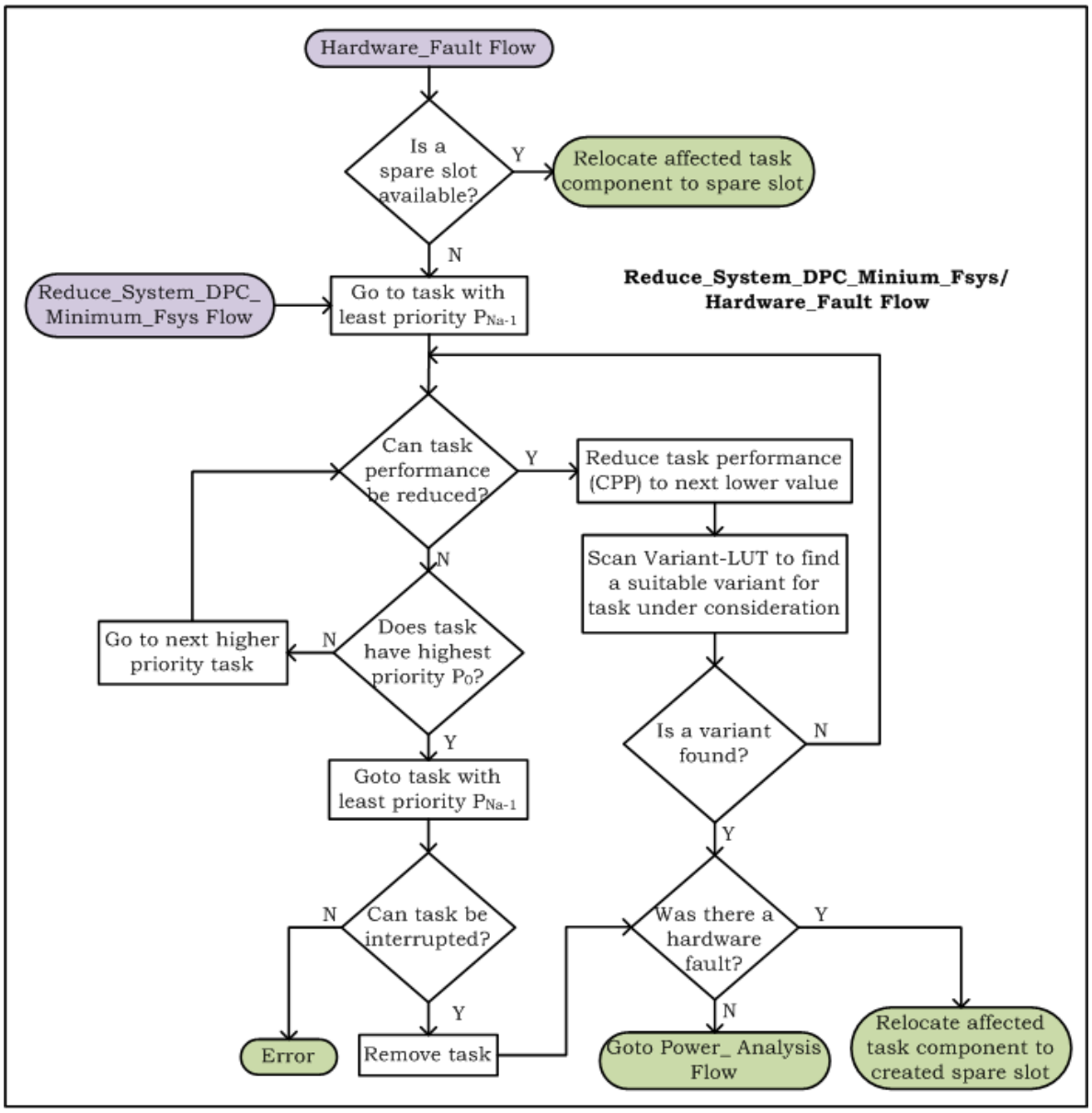

Figure 7. Reduce_System_DPC_Minimum_F_{sys} and Hardware_Fault Flows of the Explorer.

Find_Task_Variant Flow: The flow to select a suitable variant for a task under consideration is shown in the expanded view in Figure 8. The explorer scans the Variant-LUT of the task to find an ASP circuit variant whose frequency and performance are equal to F_{sys} and the CPP of that task, and which fits in the available number of slots on the FPGA. If no variant is selected due to lack of space, it goes to the *Space_Adjustment Flow* in an attempt to accommodate the task. Otherwise, if there is no variant selected, there is a system design error.

4.3. Analyzing Cost of Run-Time Structural Adaptation

Adaptation Time Overhead: From Section 4.2, it can be seen that the Explorer can dynamically maintain the performance of a system's critical tasks by adjusting the performance values of its lower priority tasks, changing system frequency and/or relocating task components such that the system configuration meets the existing power budget, hardware resource constraints, and the performance specifications of all the individual tasks. The cost of such a dynamic flexibility is only a small period of adaptation time where some or all of the executing tasks are affected. This further depends on the availability of spare slots. For example, suppose that a variant of task needs to be reconfigured for adaptation. If spare slots are available, the new variant can be reconfigured to the spare slot(s) while the original task is executing. Once the new variant is reconfigured, the originally running task variant is stalled at an appropriate point in its execution cycle and the new one is integrated by the MACROS framework. The adaptation time in this case is only the time taken by MACORS to integrate the components of the newly configured task. However, if spare slots are not available, the original task variant is stalled, followed by reconfiguration of the new task variant. In this case, the reconfiguration time of the variant adds to the adaptation time. For a dynamic relocation example in [7], the adaptation time is in the order of milliseconds. This time frame can satisfy the permitted adaptation time for many applications; for example, multi-media/video/image processing applications, digital communication etc. The small time overhead of the method makes it suitable for supporting run-time adaptation.

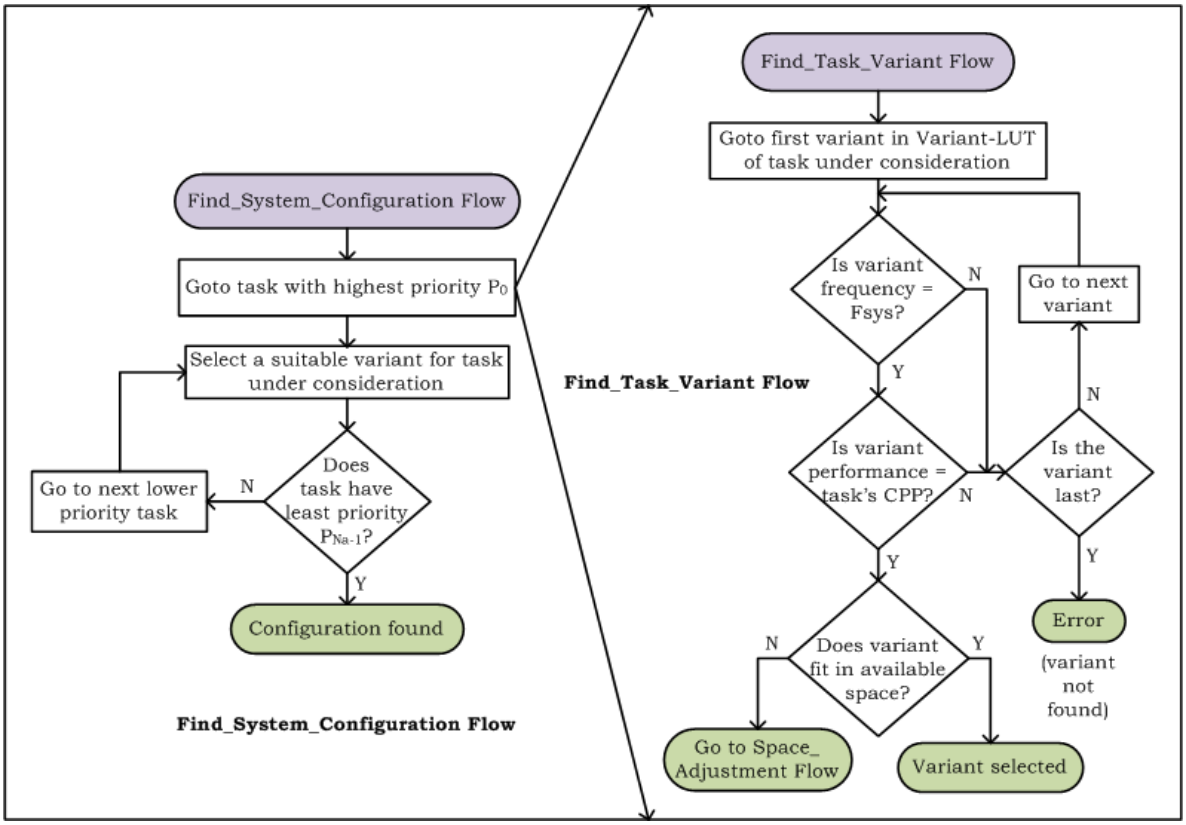

Figure 8. Find_System_Configuration and Find_Task_Variant Flows of the Explorer.

Storage Overhead and Exploration Time: Consider a system having 50 modes and 20 tasks. Each task has 16 ASP circuit variants. For simplicity, let there be 10 tasks in each mode. In this case, if system configurations need to be explored, for each mode, $16^{10} \approx 10^{12}$ configurations will need to be stored and scanned. For a total of 50 modes, the Explorer would need to store $50 \times 16^{10} \approx 5 \times 10^{13}$ configurations. It may not be feasible to store so many configurations in an LUT. Additionally, even if such a large design space is stored in a memory system, exploring it would take a prohibitively large

computation time to be used at run-time. However, as discussed in Section 4.2, the Explorer selects ASP circuit variants for all tasks individually, instead of selecting the entire system configuration. Therefore, only ASP circuit variants of individual tasks need to be stored. For the above example, only $20 \times 16 = 320$ configurations will need to be stored. From these configurations, the Explorer will only explore $10 \times 16 = 160$ configurations for the existing mode. The number of configurations to be stored and explored reduce by a huge factor of $(50 \times 16^{10})/320 = 1.7 \times 10^{11}$ and $(16^{10})/160 = 6.9 \times 10^{9}$ respectively! The method thus not only reduces the memory requirements of the system, but also reduces the exploration time significantly by reducing the number of configurations to be explored; a requisite for run-time adaptation.

Reconfiguration Power Consumption: While reconfiguring a task component on a slot, the task component operating on that slot prior to reconfiguration needs to be stalled. This means, during reconfiguration, DPC is only due to the reconfiguration process since there is no data processing during that time. Furthermore, DPR usually takes hundreds of micro-seconds to tens of milli-seconds depending on the size of the bit-stream and the medium used for reconfiguration. It has been demonstrated that power consumption for this small frame of time is negligible as compared to any task operating on that slot [55]. In fact, since there is no data processing during DPR, the power budget improves for that period of DPR. This is because instead of power being consumed by an executing task, there is only a negligible power consumption of the DPR process. Thus, power consumed by DPR can easily be ignored from the power budget calculations.

Data Loss during Adaptation: For the proposed run-time adaptation mechanism, it is important to consider the data/information lost as a result of the adaptation. As mentioned in Section 2, critical multi-task multi-modal systems usually run continuously executing stream processing hardware tasks. Consider a system executing tasks which process incoming video streams. In such a case, if a new variant for a task needs to be reconfigured, it will be stalled only when a frame being processed is completed and the results are delivered. There is therefore, no loss of data within a frame. Depending on the time taken for reconfiguration, as discussed above, in the 'Adaptation Time Overhead' Section, a couple frames could be lost. Once the task is reconfigured, it begins operating from the new incoming frame. This means, for the kind of systems considered in this paper, adaptation does not require saving the states of tasks, as they process incoming streams, and they are stalled/initiated only at the completion/start of a new set of data (frame in this case).This is efficiently supported by the MACROS framework, which can auto-synchronize the set of tasks executing at that time. Thus, there is no time/storage overhead of the mechanism in terms of saving the tasks' contexts. However, there is some loss of data, the cost of which depends on the nature of the application being supported by the system. For example, losing 2–3 frames in commercial applications does not matter since the loss of a few frames is not even visible to the human eye.

Influence of Adaptation on Task Functioning: As discussed above, the systems considered for this paper are autonomous systems with tasks processing incoming streams of data. Such tasks mostly include BRAM-based buffers to store the incoming streams or the processed outputs. There is usually no interaction with external memory. External memory, in such systems, is mostly used to store the different bit-streams of the task variants and is accessed by the Bit-stream Manager while configuring a selected task variant. When different variants of the ASP circuits of a task are configured for adaptation, they do not affect the functionality of the task. For example, consider an ASP circuit variant of an image processing task which occupies one slot, and operates at 120 MHz to give a performance of 120 fps. The ASP circuit of this variant would include one buffer dealing with the entire image at 120 MHz. Suppose that the system needs to adapt to a low power budget condition and another variant which occupies two slots and operates at 60 MHz needs to be configured. This variant would have two buffers, processing half the image simultaneously at 60 MHz, thus giving a frame rate of 120 fps. Similarly, consider a situation where a variant with a reduced performance needs to be configured to adapt to a limited resource condition. If a variant which occupies one slot and operates at 60 MHz is configured, there would be one buffer processing the full image at 60 MHz, thus giving a

reduced performance of 60 fps. This means adaptation does not affect the task functionality in any way. The variants of ASP circuits of the tasks are developed with the aim of run-time adaptation during their design phase.

Furthermore, if external communication with the DDR memory is required, this is achieved through continuously executing static tasks, which operate at a fixed frequency. Thus, irrespective of the changing operating frequencies of the configured task variants, their communication with the external DDR memory is not affected. It can thus be seen that the Explorer does not have any role to play in terms of the task functioning. It is solely responsible for selecting the appropriate system configuration that meets the changing set of constraints. Run-time adaptation does not affect the functioning of the tasks and their communication with external memory.

5. Method for Derivation of DPC Estimation Model

From Section 4.2, it is seen that during the structural adaptation process, the Explorer uses the DPCEM of an FPGA to estimate the DPC of potential system configurations at run-time. This section therefore discusses the method to derive the DPCEM for an FPGA.

During run-time structural adaptation, it is possible to reconfigure only the PRRs of the FPGA with the ASP circuits of the system tasks. This means run-time structural adaptation can result in a change only in the number and type of reconfigurable resources of the FPGA. Therefore, only the resources in the PRRs and hence only the system tasks contribute to DPC of the FPGA. Power consumption of any FPGA resource other than those in the PRRs become a part of the static FPGA power consumption. DPC of a system configuration and hence the DPCEM can be therefore be expressed as the sum of the DPC of each type of reconfigurable resource used by the ASP circuits. Thus, the DPCEM of an FPGA must have the FPGA's reconfigurable resources as its variable parameters.

To derive the DPCEM, the power consumption behavior of each type of reconfigurable resource must be identified. From experiments performed on different FPGA devices based on the 28 nm CMOS technology (Xilinx 7-series and Zynq 7000 family), it was found that DPC of each type of reconfigurable resource in an ASP circuit has a linear relation with: (a) clock frequency and (b) utilization, i.e., number of slices of the resource deployed. This behavior is in line with DPC trend shown for the Xilinx FPGAs in [56]. Although the focus in [56] is not to observe the DPC behavior of FPGAs, the presented DPC results depict the linear behavior of the Stratix and Virtex-4 family of FPGAs. This can also be validated from the equation that represents the DPC of each resource in a FPGA [57]:

$$DPC_{resource} = C_{switched} \times V^2 \times F \tag{1}$$

where $C_{switched}$ is the switched capacitance, V is the voltage supply, and F is the clock frequency of the resource. From this equation, it is clear that the DPC of any FPGA resource will have a linear relation with the clock frequency.

Another point to be noted is that capacitance of the resource is influenced by its switching activity, i.e., number of transitions in a clock period [57]. The switched capacitance of a resource can therefore be expressed as $C_{switched} = C_{eff} \times S$, where C_{eff} is the effective capacitance of the resource, that depends on the parasitic effects of the interconnection wires and transistors, and S is the switching activity of the resource. Thus, the total DPC of an FPGA resource with a utilization U can be written as:

$$Total\ DPC_{resource} = V^2 \times F \times C_{eff} \times U \times S \tag{2}$$

From Equation (2), the DPC of the FPGA can be expressed as [57]:

$$DPC_{\text{FPGA}} = V^2 \times F \times \sum C_{eff\_i} \times U_i \times S_i \tag{3}$$

where C_{eff_i}, U_i, and S_i are the effective capacitance, utilization, and switching activity of the resource i.

Thus, DPCEM derivation essentially involves identifying the coefficients representing the linear relationship of each type of reconfigurable resource with frequency and its utilization. These identified coefficients will be dependent on the switching activity of the application. The DPCEM can therefore be represented as a simple linear equation. Such a model can have a very small execution time overhead and can hence be suitable for use at run-time.

5.1. Experimental Setup for DPCEM Derivation

As discussed above, DPC of an FPGA strongly depends on its underlying micro-architecture and the switching activity factor of the circuit configured on it. As a result, the DPCEM of an FPGA corresponds to only the application for which it has been derived. If the application and/or the FPGA changes, a new set of DPCEM coefficients will need to be established for that FPGA and application. However, the method of DPCEM derivation stays the same for any FPGA and any application. Thus, to be able to derive a DPCEM, it was necessary to develop an experimental setup, i.e., choose an FPGA platform and develop a test-circuit for which the DPCEM of that FPGA will be derived. Since the MACROS framework requires DPR to enable run-time structural adaptation, the latest Xilinx 7-series FPGA family based on 28 nm CMOS technology was selected. The ZedBoard [58], which houses the Zynq XC7Z020 device (Artix-7 equivalent) [59], and the KC705 evaluation board [60], which houses the Kintex-7 XC7K325T FPGA [61], were selected because they have different methods for DPC measurement. This way, both the methods can be tested for DPCEM derivation and their use in practical systems.

Most modern FPGAs have three types of reconfigurable computing and memory resources, namely, Logic, BRAM and DSP slices [62,63]. A test-circuit was therefore developed such that it includes all the three types of resources and has a structure which permits variation of usage of each type of resource individually to isolate its behavior. It must be noted that in practice, for deriving the actual DPCEM of an FPGA, the application that will run on the FPGA should be implemented and tested instead of using the test-circuit discussed here. This aspect is briefly discussed in the next Section.

The developed test-circuit, as shown in Figure 9a, includes three modules, namely, Memory, Logic and DSP modules. The Memory Module mainly includes 2 buffers, formed using BRAM slices. One buffer acts as the input to the Logic and DSP Modules while the other is being written into, and they switch roles when the buffer is full. The buffers are continuously written into and read from to have some switching activity in the BRAM slices that will result in their DPC. The Logic and DSP modules are mainly formed using Logic and DSP slices respectively. They perform the same set of arithmetic functions, are fed with the same input from the Memory module and produce the same output. Their outputs are switched to form the circuit output to avoid the CAD tool from considering one of the modules redundant and eliminating it from the design.

This circuit, consisting of Memory, Logic and DSP modules forms the base design for the experiments. From this design, several variants with individually varying Logic, BRAM and DSP slice utilization must be generated to identify the DPC behavior of each type of resource. This means, variants with increasing Logic slice utilization must be generated at multiple frequencies to observe the behavior of Logic slices alone. Similarly, another set of variants must be generated with increasing BRAM and DSP slice utilization at multiple frequencies to observe their individual behaviors. To be able to do so, each module is scaled multiple times and the outputs are logically ORed together to produce one single output. For example, design variants with varying Logic utilization can be generated by repeating only the Logic modules n times, where $n = 1$ *to* N; N is the number beyond which, if the module is repeated, the required logic utilization will become higher than the logic slices available in the FPGA. A combination of the scaled modules is termed as a 'Bundle'. The scaled modules are logically ORed to produce a single output so that the design functionality is maintained in every variant. This means although every variant has a different Logic utilization, they all behave as if they have only one Logic module in their design. Any other method that can maintain the design

functionality while varying resource utilization can also be used to generate the variants. Figure 9b shows a design variant where only Logic utilization is increased by scaling Logic Modules in the Logic bundle and the BRAM and DSP Modules remain untouched.

While developing the base design, first, the individual modules, i.e., the Memory, Logic, and DSP modules were designed. The attempt was that each module has a resource utilization between 15–25% so that at-least 4 variants of varying utilization of one reconfigurable resource could be generated by scaling the corresponding module ($N = 4$; $n = 1$ to 4). This way sufficient data would be available for DPCEM derivation. Apart from this criteria, the resource utilization achieved for the base designs was arbitrary; purely dependent on the circuits involved in the design.

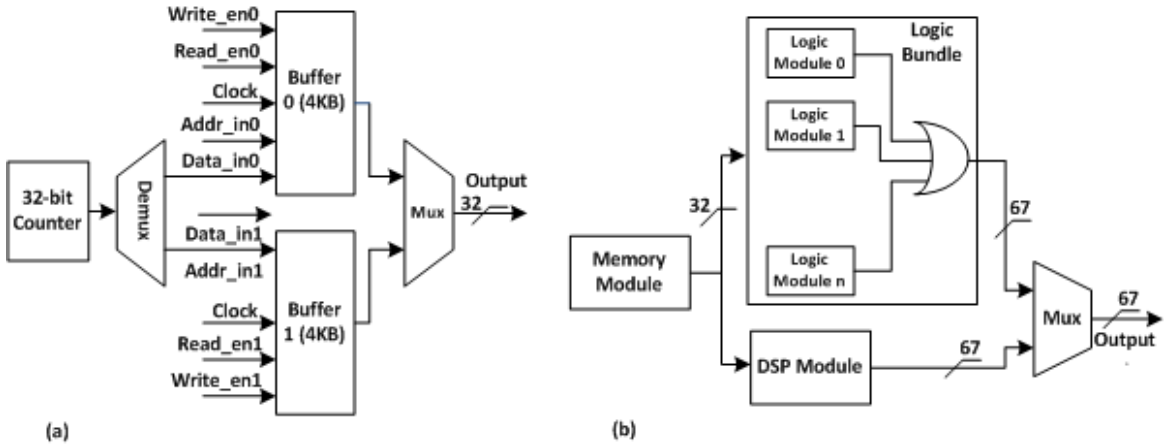

Figure 9. (**a**) Memory Module; (**b**) A design-variant with increased Logic Utilization.

The base design developed for the Zynq XC7Z020 device, which includes one Memory, Logic, and BRAM module, has a resource utilization of 3236, 35 and 22 Logic, DSP and BRAM slices, which is ≈25%, 16% and 16% respectively. These numbers are taken from the resource utilization file generated for the base design by Vivado. With this initial resource utilization, it was possible to generate 4 variants of varying Logic slice utilization and 5 variants of varying BRAM and DSP slice utilization.

Similarly, the base design developed for the Kintex-7 XC7K325T device includes 11219, 140 and 70 Logic, DSP and BRAM slices, which is ≈22%, 17% and 16% respectively. This utilization permitted generation of 5 variants of varying utilization per resource.

5.2. Setup for DPCEM Derivation for Real Applications

As discussed, a test-circuit has been developed in this paper to demonstrate the DPCEM derivation procedure. When deriving the DPCEM for real applications, the same concept that is used for developing the test-circuit must be applied. This means, the main idea is to be able to isolate the reconfigurable resources to identify their individual behaviors. The only assumption here is that the design/circuit for the application is modular. This will allow scaling the number of specific modules in the design to vary specific reconfigurable resources. For this purpose, modules that have a higher concentration of a particular resource must be identified, just like in the case of the test-circuit. For example, video/image processing applications usually have buffers to store the incoming frames. They also have modules that are responsible for data processing; such modules will usually have a higher percentage of logic slice utilization. DSP modules will have more DSP slice utilization, and so on. To generate variants that have varying utilization of a particular resource, only the corresponding module must be scaled multiple times, and not the entire circuit. For example, to generate variants with increasing logic utilization, only the data processing module can be scaled multiple times, with their outputs logically ORed together to produce one single output. This way, there is no change in the functioning of the module and its output interface. Similarly, to generate variants with increasing

BRAM utilization, only the buffer modules can be scaled. The same applies for the DSP modules. Once this is done, the same method as discussed in this paper (Section 5.4) can be used to derive the DPCEM for the desired FPGA device.

All applications may not use all the reconfigurable resources. Some may not need DSP modules, while some may not need BRAM modules. For such applications the model coefficients will need to be derived only for the reconfigurable resources being used. The step in the derivation procedure, corresponding to the resource not being used, can be skipped.

5.3. Power Consumption Measurement Methods

While deriving the DPCEM, DPC values of the FPGA for all the design variants are required in order to gather the initial data for model derivation. Software prediction tools like Xilinx Power Analyzer can be used for this purpose if accurate switching activity factors are available [64–66]. We have used the method of actual DPC measurement to maximize the accuracy of the model derivation process. This Section discusses the method of DPC measurement for the Zynq and Kintex devices housed on the ZedBoard and KC705 evaluation board respectively.

Initially, the static power consumption (SPC) of both the boards is recorded. Total power consumption (TPC) of the boards is then measured for every design variant. SPC is subtracted from the TPC of every design variant to obtain the DPC of the respective FPGA for that design variant. The DPC corresponds to the FPGA and not to the board since the test design uses only the FPGA resources. The power consumed by the other on-board resources is a part of the board's SPC.

Measurement Method on ZedBoard: The ZedBoard has a 10 mΩ sense resister in series with its 12 V power supply [58]. Power consumption of the board is obtained through the voltage recorded across the sense resistor using Agilent Technologies Digital Multimeter, U3401A.

Measurement Method on KC705 Board: For the KC705 board, power supply is distributed in the form of individual rails, which can be monitored with the on-board Texas Instruments power controllers UCD9248PFC [60]. The individual rail voltages, currents, and thus power consumption can be read by I2C-based communication with the Power Management Bus (PMBus) connected to the controllers. Power consumption of the board is the sum of the power consumption of all rails.

5.4. DPCEM Derivation Process

This Section presents the procedure to derive the complete DPCEM of an FPGA in terms of all its reconfigurable resources, i.e., Logic, BRAM and DSP Slices. It is based on [67], which discusses a DPCEM derivation method for applications that use only Logic and BRAM slices. The DPCEM derivation procedure in this paper considers usage of DSP slices along with Logic and BRAM slices. It thus enables DPCEM derivation for an FPGA in terms of all its reconfigurable resources. The method of identifying the coefficients for each resource is much simpler than the method in [67]. Since all the reconfigurable resources are considered, the test designs developed for the derivation are different as compared to the ones used in [67]. Furthermore, in [67], DPCEM results have been presented only for the ZedBoard. However, in this paper, DPCEM results have been presented for the ZedBoard and the KC705 evaluation board, i.e., for the Xilinx Zynq XC7Z020 and Xilinx Kintex 7 devices respectively. This is done to: (a) Validate the fact that the proposed procedure is generic and can be used to derive a DPCEM for any FPGA device or any FPGA-based board; as long as there is a mechanism to measure the power consumption of the FPGA/board, and (b) Analyze the DPC behavior of FPGA devices built on the same micro-architecture.

Zynq XC7Z020 device, housed on the ZedBoard, is used as the experimental platform to demonstrate the derivation procedure. The DPCEM results for the KC705 board are presented next. It is to be noted that even though the Zynq device includes the ARM Cortex A9 core, we are interested in the DPCEM of only the FPGA. Since the test-circuits involve only the resources of the FPGA, the ARM processor has a static power consumption. It is therefore, a part of the SPC of the ZedBoard, which is eliminated from the TPC of the board to obtain the DPC of the FPGA. Thus, the ARM processor does

not influence the derived DPCEM of the FPGA, and thus does not affect Explorer's decision while choosing a system configuration.

Step 1—Isolate Behavior of Logic Slices: In this step, four design variants with varying Logic utilization are generated from base design discussed in Section 5.1. The Logic modules in the Logic Bundle are scaled from $n = 1$ to 4. The Memory and DSP modules stay constant at one module for all the variants. The resource utilization file of the base design for the Zynq device shows that each Memory and DSP module consists of 22 BRAM and 35 DSP slices respectively (also discussed in Section 5.1). These BRAM and DSP slices remain constant for all the variants.

For each of the four variants, five bit-streams are generated whose operating frequencies range from 30 MHz to 150 MHz, in multiples of 30 MHz. DPC of the FPGA is measured for each of the 20 bit-streams using the procedure described in Section 5.3. Since number of Logic slices and frequency are the only variable parameters in the design variants, variation in the measured DPC is also only due to variation in Logic slices and frequency. Therefore, the relation between DPC and number of Logic slices is observed at each of the five frequencies with the help of the graph shown in Figure 10a. Five linear equations are obtained, one for each frequency. The 'Step 1' tab of Table 3 lists the Coefficients and Constants of these equations at each multiple of 30 MHz. The Coefficient at a particular frequency corresponds to the slope of increase in DPC due to increase in number of Logic slices at that frequency. Similarly, the Constant at a particular frequency corresponds to the DPC of the constant resources in the design, including the constant BRAM and DSP slices (22 and 35 respectively in this case), at that frequency. It can be observed from the 'Step 1' tab of Table 3 that the Coefficients and Constants linearly increase with frequency. Their values obtained at 30 MHz scale with the same factor as the frequency. This shows that the Logic slices, BRAM slices, DSP slices and other constant resources in the FPGA have a frequency dependent behavior. The set of five linear equations can, therefore, be encapsulated into a single frequency dependent equation as:

$$DPC_{(Zynq7020)}\,(\text{mW}) = \frac{F_{cc}(\text{MHz})}{30} \times (0.013 \times N_{LS} + 55.17) = \frac{F_{cc}}{F_{min}} \times (C_{LS} \times N_{LS} + 55.17) \quad (4)$$

where, F_{cc} is current operating clock frequency and F_{min} is minimum operating frequency for the test design. C_{LS} is the coefficient relating the DPC to N_{LS}; the number of Logic slices. $\frac{F_{cc}}{F_{min}} \times 55.17$ represents the frequency dependent behavior of the remaining constant resources in the design, including the constant BRAM and DSP slices. This means, in actual circuits, which will have a different BRAM and DSP slice utilization, their influence will be observed in the second term of Equation (4), i.e., $\frac{F_{cc}}{F_{min}} \times 55.17$. The constant 55.17 will be replaced by a different value, which will depend on the BRAM and DSP slice utilization, and also on the switching activity factor of that hardware circuit.

Step 2—Isolate Behavior of BRAM Slices: In this step, five design variants are generated by scaling the Memory modules in the Memory Bundle from 1 to 5. DSP utilization (35 slices) remains constant throughout this step. Frequency is varied from 30 MHz to 150 MHz for each variant, resulting in 25 bit-streams. DPC of Zynq is measured for each bit-stream. It must be noted that increasing BRAM utilization also increases Logic utilization. As a result, DPC of Logic slices must be eliminated to observe the behavior of BRAM slices alone. DPC of Logic slices in every design variant is calculated using the first term of (4), i.e., $\frac{F_{cc}(\text{MHz})}{30} \times 0.013 \times N_{LS}$ and subtracted from the measured DPC values. Variation in DPC is now only due to variation in BRAM slices and frequency. The relation between DPC of BRAM slices is observed at each of the five frequencies with the help of the graph shown in Figure 10b. A set of five linear equations are obtained, whose Coefficients and Constants are listed in the 'Step 2' tab of Table 3. The Coefficient at a particular frequency corresponds to the slope of increase in DPC due to increase in number of BRAM slices at that frequency. Similarly, the Constant at a particular frequency corresponds to the DPC of the constant resources in the design at that frequency. The constant resources in this step include the DSP slices (35 in this case), but exclude the Logic slices as their power consumption has been eliminated from the measured power consumption. It can be observed from the 'Step 2' tab of Table 3 that these Coefficients and Constants also linearly increase

with frequency, like in Step 1. This again shows that the reconfigurable resources and other constant resources in the FPGA have a frequency dependent behavior. The set of equations can hence be combined into the following general equation:

$$DPC_{(Zynq7020)}\,(\text{mW}) = \frac{F_{cc}(\text{MHz})}{30} \times (1.1 \times N_{BS} + 31) = \frac{F_{cc}}{F_{min}} \times (C_{BS} \times N_{BS} + 31) \quad (5)$$

where, C_{BS} is the coefficient relating the DPC to N_{BS}; the number of BRAM slices. Since DPC of Logic slices is eliminated, $\frac{F_{cc}(\text{MHz})}{30} \times 31$ represents the frequency dependent behavior of the remaining constant resources in the design excluding the Logic slices.

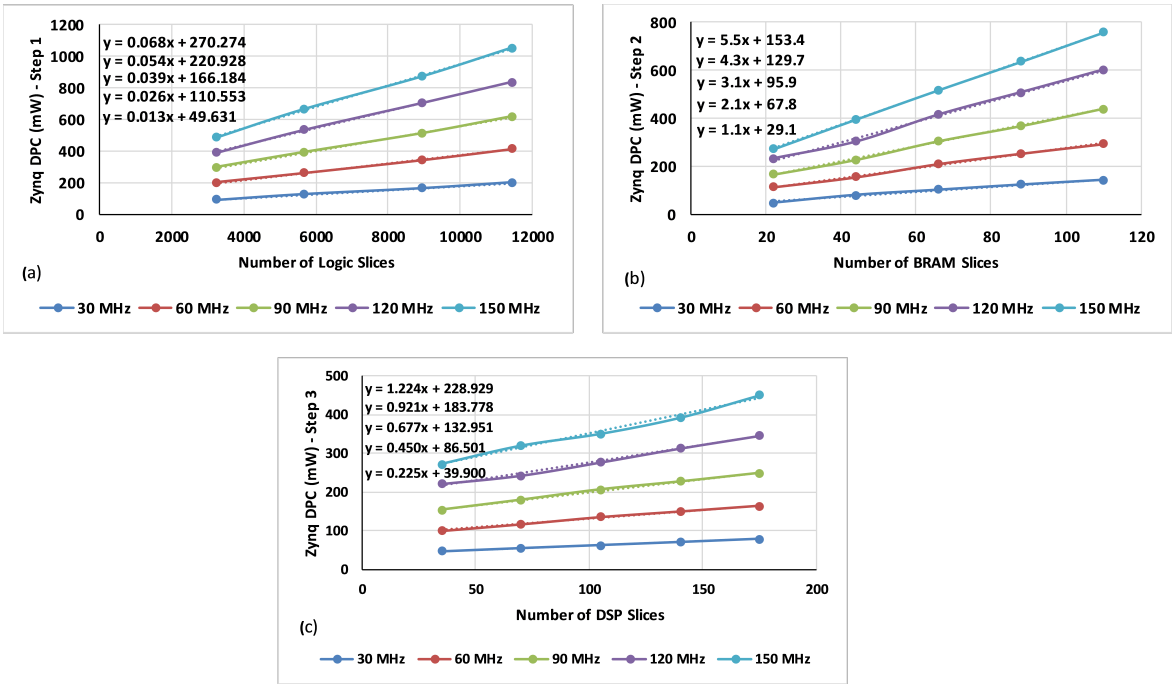

Figure 10. (**a**–**c**) Zynq DPC vs. Number of Logic, BRAM and DSP slices respectively.

Table 3. Coefficients and constants of linear equations obtained during DPCEM derivation.

$F = \frac{F_{cc}\ (\text{MHz})}{30}$	Step 1		Step 2		Step 3	
	Coefficient	**Constant**	**Coefficient**	**Constant**	**Coefficient**	**Constant**
1	0.013	49.631	1.1	29.1	0.225	39.9
2	0.026	110.553	2.1	67.8	0.45	86.501
3	0.039	166.184	3.1	95.9	0.677	132.951
4	0.054	220.928	4.3	129.7	0.921	183.778
5	0.068	270.274	5.5	153.4	1.224	228.929
Approximation	$F \times 0.013$	$F \times 55.17$	$F \times 1.1$	$F \times 31$	$F \times 0.226$	$F \times 47$

Step 3—Isolate Behavior of DSP Slices: In this step, five design variants are generated by scaling the DSP modules in the DSP Bundle from 1 to 5. BRAM utilization (22 slices) remains constant throughout this step. Frequency is varied from 30 MHz to 150 MHz for each variant, resulting in 25 bit-streams. DPC of Zynq is measured for each bit-stream. In this step too, scaling DSP modules also increases Logic slices. This is because the DSP module is made up of DSP slices and some Logic slices used as glue logic. Hence DPC of the Logic slices in every design variant is subtracted from the measured DPC values like in Step 2. The relation between DPC and DSP slices is observed at each of the five frequencies with the help of the graph shown in Figure 10c. A set of five linear equations

are obtained. Their Coefficients and Constants are listed in the 'Step 3' tab of Table 3. The Coefficient at a particular frequency corresponds to the slope of increase in DPC due to increase in number of DSP slices at that frequency. The Constant at a particular frequency corresponds to the DPC of the constant resources in the design at that frequency. The constant resources in this step include the BRAM slices (22 in this case), but exclude the Logic slices as their power consumption has been eliminated from the measured power consumption. Here too, the frequency dependent behavior of the FPGA resources is observed; the Coefficients and Constants from the 'Step 3' tab of Table 3 scale with frequency. Therefore, the set of equations in this step also can be represented as the following general frequency dependent equation:

$$DPC_{(Zynq7020)}\,(\text{mW}) = \frac{F_{cc}(\text{MHz})}{30} \times (0.226 \times N_{DS} + 47) = \frac{F_{cc}}{F_{min}} \times (C_{DS} \times N_{DS} + 47) \tag{6}$$

where, C_{DS} is the coefficient relating the DPC to N_{DS}; the number of DSP slices. Since DPC of Logic slices is eliminated, $\frac{F_{cc}(\text{MHz})}{30} \times 47$ represents the frequency dependent behavior of the remaining constant resources in the design excluding the Logic slices.

Step 4—Complete the DPCEM Equation: From (4)–(6) we have: $C_{LS} = 0.013$, $C_{BS} = 1.1$, and $C_{DS} = 0.226$; the coefficients which relate DPC to the number of Logic, BRAM and DSP slices respectively. Also, from Steps 1 to 3, it has been observed that the DPC of each type of resource linearly depends of frequency. A generic equation can thus be formed from (4)–(6), which represents the DPCEM of an FPGA.

$$DPC_{(FPGA)}\,(\text{mW}) = \frac{F_{cc}}{F_{min}} \times (C_{LS} \times N_{LS} + C_{BS} \times N_{BS} + C_{DS} \times N_{DS} + C_F) \tag{7}$$

where, C_F is a frequency dependent coefficient representing the behavior of the remaining constant resources in the design excluding the Logic, BRAM and DSP slices.

Comparing (4) from Step 1 with (7) from Step 4, we get the following relation:

$$C_{BS} \times N_{BS} + C_{DS} \times N_{DS} + C_F = 55.17 \implies 1.1 \times 22 + 0.226 \times 35 + C_F = 55.17 \implies C_F = 23.06 \tag{8}$$

Similarly, comparing (5) and (6) with (7), the following relations are obtained:

$$C_{DS} \times N_{DS} + C_F = 31 \text{ and } C_{BS} \times N_{BS} + C_F = 47. \tag{9}$$

The identified coefficients C_{LS}, C_{BS}, C_{DS}, and C_F validate the relations in (9). Thus, the complete DPCEM for Zynq XC7Z020 involving the parameters of Logic slices, BRAM slices, DSP slices and frequency can be represented as:

$$DPC_{(Zynq7020)}\,(\text{mW}) = \frac{F_{cc}}{F_{min}} \times (0.013 \times N_{LS} + 1.1 \times N_{BS} + 0.226 \times N_{DS} + 23.06) \tag{10}$$

The maximum difference between the DPC values estimated using (10) and the measured DPC results is 30 mW. Thus, (10) is an accurate DPCEM for the Zynq XC7Z020 FPGA device, for the test design considered. It is clear that the BRAM slices highly influence the DPC of the FPGA as against the Logic slices ($1.1/0.013 \approx 85$ times) and DSP slices ($1.1/0.226 \approx 5$ times). Also, the following can be analyzed from the generic DPCEM Equation (7) obtained for an FPGA:

Due to the presence of the frequency dependent coefficient C_F, it is possible to reduce power consumption by decreasing frequency alone. This means, a task variant V_1 with double the resource utilization and half the operating frequency of another variant V_2 will have lesser DPC as compared to V_2. For example, consider the first two variants, $T_0 - 0$ and $T_0 - 1$ listed in Table 2. $T_0 - 0$ has $F_{sys} = 240$ MHz and uses 1 slot on the FPGA. $T_0 - 1$ has $F_{sys} = 120$ MHz and uses 2 slots on the FPGA to give the same performance as $T_0 - 0$. If we assume that $T_0 - 1$ has double the number of resources

used in $T_0 - 0$, it will still have lower power consumption as compared to $T_0 - 0$ due to the presence of the frequency dependent coefficient C_F in Equation (7). The value of the term $\frac{F_{cc}}{F_{min}} \times C_F$ will be halved when the frequency reduces to 120 MHz. It must be clarified that variant $T_0 - 1$ may not have exactly double the resources as that of $T_0 - 0$, even though it uses double the number of slots as compared to $T_0 - 0$. For example, $T_0 - 0$ could be carrying out some functions using more BRAM slices in a slot. On the other hand, $T_0 - 1$ could be carrying out the same function using more Logic slices in the two occupied slots instead of BRAM slices so that its DPC could further be reduced. To summarize, from the generic DPCEM Equation (7), it can be said that due to the frequency dependent coefficient C_F, the power consumption of a variant with double the resource utilization and half the operating frequency of another variant will be lower instead of being the same. Thus, configuring a variant at a lower frequency will help reduce the system DPC. (Each system mode usually has only a few tasks whose performance need to be sustained at all times. The performance of other tasks may be reduced within the specified performance range, if needed, to accommodate the changing set of constraints. Thus, along with reduced F_{sys}, which helps reduce the system's DPC, the lowered performance of non-critical tasks can further reduce the total DPC of the system.)

Since the DPCEM derivation procedure is generic, the same method has been followed to derive the DPCEM for the Kintex-7 XC7K325T device. For the steps 1, 2 and 3, the Logic, Memory and DSP modules are respectively scaled from 1 to 5. Frequency is varied from 30 to 210 MHz in steps of 30 MHz, resulting in 35 bit-streams per step. Figure 11a–c show the plots representing DPC vs. Logic, BRAM and DSP slices respectively. Based on the set of linear equations obtained, the following coefficient values are derived:

$$C_{LS} = 0.012, C_{BS} = 1, C_{DS} = 0.22 \text{ and } C_F = 29.2 \tag{11}$$

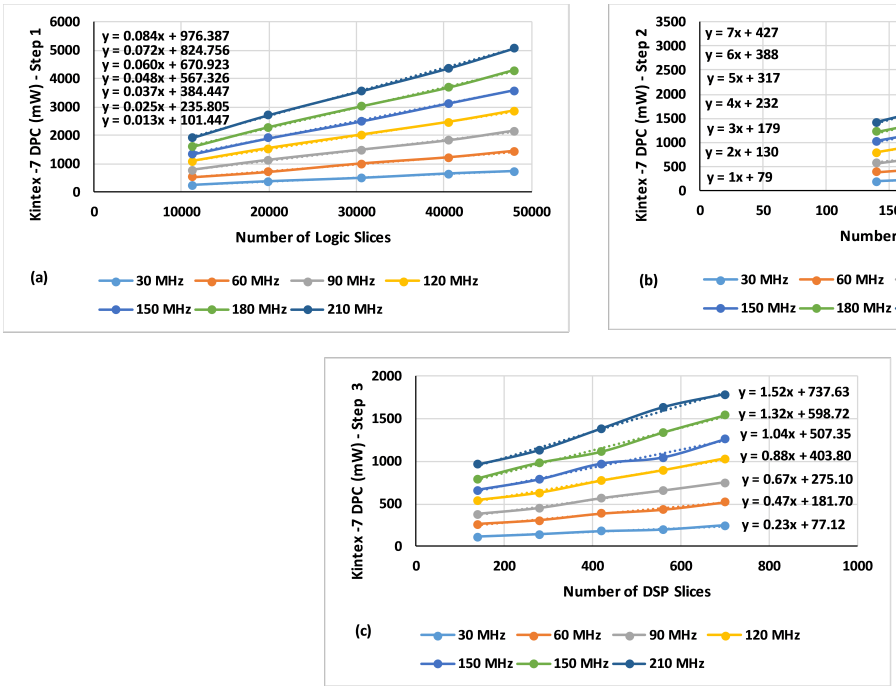

Figure 11. (**a**–**c**) Kintex-7 DPC vs. Number of Logic, BRAM and DSP slices respectively.

The complete DPCEM for Kintex-7 involving the parameters of Logic slices, BRAM slices, DSP slices and frequency can be represented as:

$$DPC_{(Kintex\text{-}7)}\,(\text{mW}) = \frac{F_{cc}}{F_{min}} \times (0.012 \times N_{LS} + 1 \times N_{BS} + 0.22 \times N_{DS} + 29.2) \tag{12}$$

The maximum difference between the DPC values estimated using (12) and the measured DPC results is 100 mW. (12) is an accurate DPCEM for the Kintex-7 FPGA, for the test design considered. It can be observed that the DPCEM for Kintex-7 has the same general form as (7); the one obtained while deriving the DPCEM for Zynq device. It thus validates the generic equation obtained as the DPCEM of an FPGA. Comparing (10) and (12) shows that the coefficients obtained for Zynq and Kintex-7 devices are similar. This is because even though they belong to different FPGA families, they have the same fabrication technology and hence have similar power consumption behavior. The DPCEM results for the Zynq XC7Z020 and the Kintex-7 device also show that the proposed DPCEM derivation procedure is generic and can be applicable to any FPGA device.

6. DPCEM Usage during Run-Time Structural Adaptation

This section shows how the Explorer identifies the $PDPC$ at every power budget check and how it uses the DPCEM to estimate the DPC of a configuration-under-test.

PDPC Calculation: Consider the system to be operating on a rechargeable battery with a capacity of $I\ Ah$ at $V\ volts$. The battery capacity is thus $P_{avail}(\text{Wh}) = V \times I$. If the system needs to function for $H\ hours$, the permitted total power consumption ($PTPC$) is given as: $PTPC(\text{W}) = (P_{avail}/H)$. If the system's SPC is P_{static} W, $PDPC(\text{W}) = PTPC - P_{static}$.

At the time of a new power budget after an interval of $H_{new}\ hours$, the battery capacity reduces by $(I_{config} \times H_{new})$ Wh; where I_{config} is the current being drawn by the present system configuration in Amperes. Thus, $P_{avail}(\text{Wh}) = P_{avail} - (I_{config} \times H_{new})$. The new $PDPC$ of the system, based on the new available power budget can again be found using the above steps.

DPC Estimation of a Configuration-under-Test: Equation (7) is used to estimate the DPC of a candidate configuration. The resource utilization for the configuration can be obtained by accessing the resource utilization of the selected variants of all the tasks in the configuration from the variant-LUT and summing them up. Equation (7) then becomes:

$$EDPC\ (\text{mW}) = \frac{F_{cc}}{F_{min}} \times (C_{LS} \times \sum_{n=0}^{N_c-1} N_{LS} + C_{B_S} \times \sum_{n=0}^{N_c-1} N_{BS} + C_{DS} \times \sum_{n=0}^{N_c-1} N_{DS} + C_F) \tag{13}$$

where N_c is the number of tasks in the candidate configuration.

7. Example of Run-Time Adaptation in Different Scenarios

This section discusses an example of run-time structural adaptation of a system for different mode, power budget and fault conditions. Consider the same example of the multi-task multi-modal system having six tasks and three modes, described in Section 4.1. Tables 1 and 2 represent the 'mode-LUT' and the 'Variant-LUT' respectively. Consider this system to be developed on the Kintex-7 FPGA device which is deployed with the MACROS framework and has 8 slots. The system runs on a rechargeable 12 V battery with a capacity of 48 Wh. F_{sys} can take values of 30, 60, 120 and 240 MHz. Table 4 summarizes the flow of events which are discussed next.

Initial State: The default system mode is M_0. On start-up, the battery capacity (BC) = 100%. Estimated time of next battery recharge is after 9 h. With a $P_{avail} = 48$ Wh and $H = 9$ h, $PTPC = 48/9 = 5.33$ W. Considering a $P_{static} = 1.7$ W, $PDPC = 5.33 - 1.7 = 3.63$ W. The Explorer sets F_{sys} to 240 MHz and selects variant no. 0 for each of the tasks in M_0, i.e., T_5, T_0, T_2 and T_4, to maintain their performance at maximum (Mode_Change Flow). Using Table 2, (11), and (13), $EDPC$ of this combination can be calculated as:

$$EDPC\ (\text{mW}) = \frac{240}{30} \times \{0.012 \times 15751 + 1 \times 142 + 0.24 \times 178 + 26.4\} = 3.195\ \text{W}.$$

Since $EDPC \leq PDPC$, $T_5 - 0, T_0 - 0, T_2 - 0, T_4 - 0$ is selected as the system configuration, as seen in Figure 12a. The system can now function for 9.81 h, ≈48 min more than needed by the budget.

Table 4. Flow of events discussed in Section 7.

Time Elapsed (hours)	System Mode	P_{avail} (%)	Current Lifetime (hours)	Needed Lifetime	*PTPC* (W)	*PDPC* (W)	Candi-Date *EDPC* (W)	New Lifetime
0	M_0	100.00		9.00	5.33	3.63	3.195	9.81
1	M_0	89.80	8.79	9.00	4.79	3.09	3.048	9.08
1	M_0	79.91	8.05	9.00	4.26	2.56	1.440	12.22
3.5	M_2	57.02	8.67	8.00	3.42	1.72	1.251	9.27
0.5	M_2	53.94	8.72	5.00	5.18	3.48	2.039	6.93

Adaptation to a depleted power budget in presence of spare slots: After one hour, BC $\approx$ 90%, which permits the system to function for 8.81 h. However, power budget shows that due to external conditions, battery recharge is possible after 9 h instead of the expected 8 h. *PDPC* reduces to 3.09 W (Power_Budget_Check Flow). Since $CDPC(3.195\text{ W}) \geq PDPC$, the Explorer reduces F_{sys} to 120 MHz (Reduce_System_DPC Flow) and selects variant no. 1 for all the tasks to maintain their performance at h_{spec}. *EDPC* of this configuration, i.e., $T_5 - 1, T_0 - 1, T_2 - 1$ and $T_4 - 1$, as shown in Figure 12b, is 3.048 W. Since $EDPC \leq PDPC$, it is selected as the new system configuration. Thus, due to availability of spare slots, the performance of all the tasks remains unaffected even when system adapts to a reduced power budget. The system can now run for 9.08 h, $\approx$16 min more than it would on the previous configuration and $\approx$5 min more than required by the budget.

Adaptation to a depleted power budget in absence of spare slots: After another hour, BC $\approx$ 80%. The system can function for 8.08 h. The new power budget estimates that since external conditions have not improved, the system will need to run for 9 more hours before a recharge is possible. *PDPC* drops to 2.56 W (Power_Budget_Check Flow). Since $CDPC(3.048\text{ W}) \geq PDPC$, the Explorer reduces F_{sys} to 60 MHz (Reduce_System_DPC Flow) and selects $T_5 - 2$, which occupies 4 slots, to maintain the performance of the critical task T_5 at its h_{spec}. Similarly, it chooses $T_0 - 2$ to maintain the performance of T_0 at $h_{spec} = 8$. Since there is no empty slot left for any other task, it reduces the *CPP* of T_0 to 4 (Space_Adjustment Flow) and selects $T_0 - 3$. Following the same process, it selects variants $T_2 - 3$ and $T_4 - 2$. The *EDPC* of $T_5 - 2, T_0 - 3, T_2 - 3$ and $T_4 - 2$, shown in Figure 12c, is 1.44 W. Since $EDPC \leq PDPC$, it is therefore selected as the new system configuration. In this case, the system can function for 12.22 h, 4.14 h more than the previous configuration and 3.22 h more than that required by the power budget. This happens because performance of lower priority tasks is reduced due to lack of enough spare slots which substantially reduces system DPC.

Mitigating hardware fault in absence of spare slots: While the system is running, there is a hardware fault in a slot where a component of T_5 is executing. Since there are no spare slots available for relocation, the Explorer attempts to reduce the *CPP* of the least priority task, T_4 (Hardware_Fault Flow). T_4 is already at its l_{spec} and so is the higher priority task T_2. It therefore reduces the *CPP* of T_0 to 2 and selects $T_0 - 4$ which occupies only 1 slot. Thus, a free slot is created and the affected task component of T_5 is relocated to that spare slot as shown in Figure 12d. Thus, the system can recover from a hardware fault and still maintain the performance of its critical task at h_{spec} by adjusting the performance and resource utilization of the lower priority tasks.

Adaptation to system mode change: For the next three hours, there are no changes in power budget, resource constraints or mode. So the system continues to have the same configuration (Flow 6) as shown in Figure 12d. After half an hour, there is an interrupt to change the mode to M_2 (Mode_Change Flow). Based on the new estimates, the system needs to be able to run for 8 h with BC $\approx$ 57%, $PDPC = 1.72$ W. Candidate configurations at 240 and 120 MHz do not satisfy the *PDPC* constraint. The Explorer sets F_{sys} to 60 MHz (Reduce_System_DPC Flow) and selects variant no. 2 for T_0 and T_1 to maintain their maximum performance. Since there are no more spare slots to accommodate T_3 and it is not possible to reduce the *CPP* for T_0 or T_1, the Explorer removes task T_3 after checking its $EC = 0$ (Space_Adjustment Flow). The *EDPC* of $T_0 - 2$, $T_1 - 2$ is 1.251 W. It is

selected as the new configuration for M_2, as shown in Figure 12e. The system can run for 9.27 h, 1.27 h more than that required by the power budget.

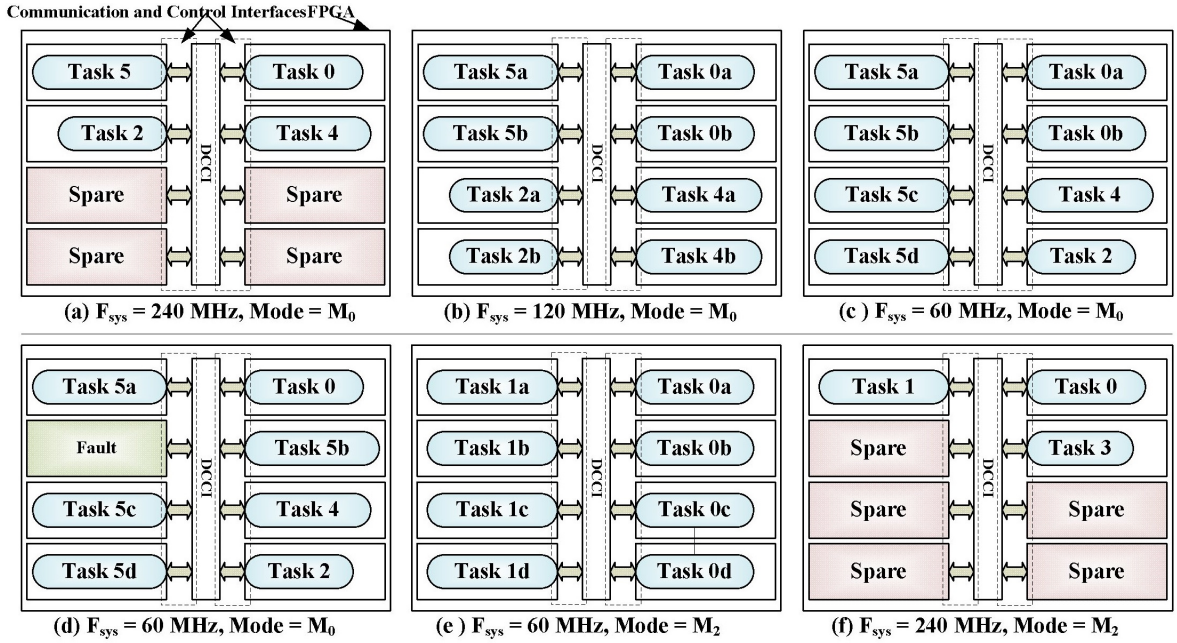

Figure 12. Different run-time structural adaptation scenarios.

Adaptation to increased power budget: After half an hour, BC $\approx$ 54% and the system can run for 8.77 h. External conditions have improved and the battery recharge is now possible after 5 h. The power budget thus goes up to 3.47 W ($Delta = 1$, Power_Budget_Check Flow). The Explorer increases F_{sys} iteratively (Flow 6), until $F_{sys} = 240$ MHz. It selects $T_0 - 0$, $T_1 - 0$ and $T_3 - 0$, as shown in Figure 12f, which has an $EDPC$ of 2.039 W. Since $EDPC \leq PDPC$, F_{sys} is maximum, and all the tasks are at their h_{spec}, there is no need for further iterations. This combination becomes the new system configuration. It lets the system run for 6.93 h, 1.93 h more than required.

These scenarios demonstrate that given any set of conditions, the Explorer can find a suitable system configuration that satisfies the mode, performance, DPC and hardware resource constraints. As a by-product of the run-time structural adaptation, the system life-time can also increase; the extent of which depends on the relation between the power budget and DPC of the selected configuration.

8. Experimental Results

Storage Requirements: The 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 design space of system configurations was used, characteristics of 3×10^6 configurations would need to be stored in the Variant-LUT. The LUT size with the proposed method (8 bytes/variant $\times$ 60 variants = 480 bytes) is only 0.002% of the size when system configurations (8 bytes/variant $\times$ 3 $\times 10^6$ variants $\approx$23 MB) are used. To reduce exploration time further, variants of each task are stored in ascending order of frequency. Thus, every time F_{sys} is reduced, the Explorer only needs to scan variants from the top of the LUT up to F_{sys} and can avoid the remaining ones at higher frequencies.

Execution Time: Execution time of the code has been recorded for different scenarios of M_0 since it has the maximum number of tasks among the other modes in the example.

Case 1—Initial state: When the system begins to function, it has a maximum power budget and a default mode of M_0. In this case, the Explorer will be able to find a configuration at $F_{sys} = 240$ MHz itself. It takes ≈23 µs to select the configuration shown in Figure 12a.

Case 2—Worst-case depleted power budget: For adaptation to a worst-case power budget drop, the system configuration consisting of tasks operating at the highest F_{sys} at their h_{spec} would need to change to the one having only the critical tasks with their $EC = 1$ operating at the lowest F_{sys} at their l_{spec}. So, in M_0, if the initial configuration is the one in Figure 12a, it would change to the one where only T_5 is executing at 30 MHz occupying all the 8 slots. To reach this stage, the Explorer will need to evaluate configurations at 120 and 60 MHz and finally settle at the configuration at 30 MHz. The Explorer takes ≈33.5 µs to reach this conclusion.

Case 3—Worst-case increased power budget: The longest decision-making process in the case of an increased power budget needs to be evaluated for this case. In M_0, suppose that the system configuration consists of only T_5 operating at 30 MHz. An increase in the power budget causes the Explorer to evaluate configurations at 60, 120 and 240 MHz. If $EDPC > PDPC$ for a potential solution at 240 MHz, the Explorer will again reduce F_{sys} to 120 MHz and settle at a configuration at 120 MHz. The Execution time for this case is ≈40 µs. It must be noted in the case of a maximum increase in power budget, the Explorer would settle at 240 MHz itself. It will not return to 120 MHz again for a solution. Hence the execution time would be less than 40 µs.

Case 4—Worst-case mode change: Suppose while the system is functioning in a mode other than M_0, it experiences the worst power budget drop, as described in Case 2. In this situation, there is also an interrupt to change the to mode M_0. In this case, the Explorer will need to evaluate configurations at $240, 120, 60$ MHz and finally settle at the one where only the critical task T_5 is executing at 30 MHz occupying all the 8 slots. Time recorded for this is ≈53 µs. This is also expected because it is a combination of Case 1, where the Explorer finds a configuration at 240 MHz, and Case 3, where the Explorer settles onto the configuration at 30 MHz from the one at 240 MHz.

Case 5—Worst-case hardware fault: Consider the configuration in Figure 12c. Assume for this example that all the tasks are at their l_{specs}. If a hardware fault occurs now, the Explorer will need to check the CPP of every task right from priority P_3 to P_0 and then finally decide that T_4 (P_3) needs to be removed. The time recorded for this is only ≈7 µs.

From all the cases considered, the maximum execution time observed is 53 µs. A partial bit-stream of 395 KB takes a reconfiguration time of 1 ms over the 32-bit ICAP port at a frequency of 100 MHz. The maximum execution time of the decision-making adaptation method is only 5% of the time taken to reconfigure a slot. The method thus proves to be suitable for use at run-time.

9. Conclusions

This paper proposes a method for mobile and autonomous, multi-task multi-modal FPGA-based embedded systems to be able to adapt structurally to unpredictable mode-change events and environmental conditions, and mitigate hardware faults. The decision-making capabilities of the method allow run-time selection of a suitable system configuration for the existing system mode such that critical tasks are sustained at their maximum performance, individual task performances are within the specified range, system's DPC is within the permitted DPC, and the configuration fits in the available resources. The paper also presents a generic procedure to derive a complete DPCEM of an FPGA in terms of all reconfigurable resources; a simple estimation tool used by the adaptation method to evaluate DPC of candidate configurations at run-time. The observed worst-case decision-making time of the method for the example considered in the paper is a very small fraction of the time taken to reconfigure a partial bit-stream on an FPGA slot. The small execution-time overhead validates the method to be suitable at run-time. Future research efforts will be directed towards finding a run-time optimization method to achieve the aforementioned adaption for very large systems where LUT-based search will not be able to satisfy the permitted adaptation time frames.

Author Contributions: Conceptualization, D.S. and L.K. and V.K.; Methodology, D.S. and L.K. and V.K.; Data Curation, D.S.; Software, D.S.; Validation, D.S.; Project Administration, L.K.; Supervision, L.K.; Funding Acquisition, L.K., Resources, L.K.; Writing—Original Draft, D.S.; Writing—Review and Editing, L.K. and V.K.

Funding: This research has been funded by external and internal sources: (a) The instrumentation and CAD software has been funded & provided by Canadian Microsystems Corporation (CMC); (b) Research and implementation of the MACROS framework (Section 3), the SoPC base for the presented research, has been funded by Ontario Centres of Excellence (OCE) and industrial partners; (c) The hardware platforms based on Xilinx Zynq-7020 FPGA and Kintex-7 FPGA have been purchased by internal (lab) funds.

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

Appendix A

Table A1 is the Variant-LUT for the system example discussed in Section 4.1. It contains the characteristics of the ten variants of each of the six tasks T_0 to T_5.

Table A1. Example Variant-LUT for six tasks.

Variant No.	No. of Slots	F_{sys} (MHz)	Perfor-Mance	Logic Slices	BRAM Slices	DSP Slices
$T_0 - 0$	1	240	8	3093	43	30
$T_0 - 1$	2	120	8	6062	86	60
$T_0 - 2$	1	120	4	3093	43	30
$T_0 - 3$	4	60	8	11,877	172	120
$T_0 - 4$	2	60	4	6062	86	60
$T_0 - 5$	1	60	2	3093	43	30
$T_0 - 6$	8	30	8	23,259	344	240
$T_0 - 7$	4	30	4	11,877	172	120
$T_0 - 8$	2	30	2	6062	86	60
$T_0 - 9$	1	30	1	3093	43	30
$T_1 - 0$	1	240	8	2061	22	82
$T_1 - 1$	2	120	8	4040	44	164
$T_1 - 2$	1	120	4	2061	22	82
$T_1 - 3$	4	60	8	7914	88	328
$T_1 - 4$	2	60	4	4040	44	164
$T_1 - 5$	1	60	2	2061	22	82
$T_1 - 6$	8	30	8	15,499	176	656
$T_1 - 7$	4	30	4	7914	88	328
$T_1 - 8$	2	30	2	4040	44	164
$T_1 - 9$	1	30	1	2061	22	82
$T_2 - 0$	1	240	8	5003	27	24
$T_2 - 1$	2	120	8	9806	54	48
$T_2 - 2$	1	120	4	5003	27	24
$T_2 - 3$	4	60	8	19,212	108	96
$T_2 - 4$	2	60	4	9806	54	48
$T_2 - 5$	1	60	2	5003	27	24
$T_2 - 6$	8	30	8	37,623	216	192
$T_2 - 7$	4	30	4	19,212	108	96
$T_2 - 8$	2	30	2	9806	54	48
$T_2 - 9$	1	30	1	5003	27	24
$T_3 - 0$	1	240	8	4009	16	46
$T_3 - 1$	2	120	8	7858	32	92
$T_3 - 2$	1	120	4	4009	16	46
$T_3 - 3$	4	60	8	15,395	64	184
$T_3 - 4$	2	60	4	7858	32	92
$T_3 - 5$	1	60	2	4009	16	46
$T_3 - 6$	8	30	8	30,148	128	368
$T_3 - 7$	4	30	4	15,395	64	184

Table A1. *Cont.*

Variant No.	No. of Slots	F_{sys} (MHz)	Perfor-Mance	Logic Slices	BRAM Slices	DSP Slices
$T_3 - 8$	2	30	2	7858	32	92
$T_3 - 9$	1	30	1	4009	16	46
$T_4 - 0$	1	240	8	5088	39	51
$T_4 - 1$	2	120	8	9972	78	102
$T_4 - 2$	1	120	4	5088	39	51
$T_4 - 3$	4	60	8	19,538	156	204
$T_4 - 4$	2	60	4	9972	78	102
$T_4 - 5$	1	60	2	5088	39	51
$T_4 - 6$	8	30	8	38,262	312	408
$T_4 - 7$	4	30	4	19,538	156	204
$T_4 - 8$	2	30	2	9972	78	102
$T_4 - 9$	1	30	1	5088	39	51
$T_5 - 0$	1	240	8	2567	33	73
$T_5 - 1$	2	120	8	5031	66	146
$T_5 - 2$	1	120	4	2567	33	73
$T_5 - 3$	4	60	8	9857	132	292
$T_5 - 4$	2	60	4	5031	66	146
$T_5 - 5$	1	60	2	2567	33	73
$T_5 - 6$	8	30	8	19,304	264	584
$T_5 - 7$	4	30	4	9857	132	292
$T_5 - 8$	2	30	2	5031	66	146
$T_5 - 9$	1	30	1	2567	33	73

References

1. Kirischian, L. *Reconfigurable Computing Systems Engineering: Virtualization of Computing Architecture*; CRC Press: Boca Raton, FL, USA, 2016.
2. Architecture Brief—What Is an SOC FPGA? Available online: https://www.altera.com/en_US/pdfs/literature/ab/ab1_soc_fpga.pdf (accessed on 1 July 2018).
3. Xilinx Explains Thinking Behind Zynq. Available online: https://www.electronicsweekly.com/news/products/fpga-news/xilinx-explains-thinking-behind-zynq-2011-11/ (accessed on 1 July 2018).
4. MCUs or SoC FPGAs? Which Is the Best Solution for Your Application? Available online: https://www.digikey.ca/en/articles/techzone/2015/nov/mcus-or-soc-fpgas-which-is-the-best-solution-for-your-application (accessed on 1 July 2018).
5. Dumitriu, V.; Kirischian, L.; Kirischian, V. 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, QC, Canada, 15–18 June 2015; pp. 1–8.
6. Dumitriu, V.; Kirischian, L. SoPC Self-Integration Mechanism for Seamless Architecture Adaptation to Stream Workload Variations. *IEEE Trans. Very Large Scale Integr. Syst.* **2016**, *24*, 799–802. [CrossRef]
7. Dumitriu, V.; Kirischian, L.; Kirischian, V. Run-Time Recovery Mechanism for Transient and Permanent Hardware Faults Based on Distributed, Self-Organized Dynamic Partially Reconfigurable Systems. *IEEE Trans. Comput.* **2016**, *65*, 2835–2847. [CrossRef]
8. Dumitriu, V.; Kirischian, L.; Kirischian, V. 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, 14–17 July 2014; pp. 47–54. [CrossRef]
9. Wigley, G.B.; Kearney, D.A. Research Issues in Operating Systems for Reconfigurable Computing. In Proceedings of the International Conference on Engineering of Reconfigurable System and Algorithms(ERSA), Las Vegas, NV, USA, 24–27 June 2002; pp. 10–16.
10. Eckert, M.; Meyer, D.; Haase, J.; Klauer, B. Operating System Concepts for Reconfigurable Computing. *Int. J. Reconfig. Comput.* **2016**, *2016*, 2478907. [CrossRef]

11. Santambrogio, M.D.; Rana, V.; Sciuto, D. Operating system support for online partial dynamic reconfiguration management. In Proceedings of the 2008 International Conference on Field Programmable Logic and Applications, Heidelberg, Germany, 8–10 September 2008; pp. 455–458.
12. Jozwik, K.; Tomiyama, H.; Edahiro, M.; Honda, S.; Takada, H. 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, 30 November–2 December 2011; pp. 416–421. [CrossRef]
13. Steiger, C.; Walder, H.; Platzner, M. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Trans. Comput.* **2004**, *53*, 1393–1407. [CrossRef]
14. Clemente, J.A.; Beretta, I.; Rana, V.; Atienza, D.; Sciuto, D. A Mapping-Scheduling Algorithm for Hardware Acceleration on Reconfigurable Platforms. *ACM Trans. Reconfig. Technol. Syst.* **2014**, *7*, 9:1–9:27. [CrossRef]
15. Iturbe, X.; Benkrid, K.; Erdogan, A.T.; Arslan, T.; Azkarate, M.; Martinez, I.; Perez, A. R3TOS: A reliable reconfigurable real-time operating system. In Proceedings of the 2010 NASA/ESA Conference on Adaptive Hardware and Systems, Anaheim, CA, USA, 15–18 June 2010; pp. 99–104.
16. Iturbe, X.; Benkrid, K.; Hong, C.; Ebrahim, A.; Torrego, R.; Martinez, I.; Arslan, T.; Perez, J. R3TOS: A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient, and Dependable Computing on FPGAs. *IEEE Trans. Comput.* **2013**, *62*, 1542–1556. [CrossRef]
17. Iturbe, X.; Benkrid, K.; Hong, C.; Ebrahim, A.; Torrego, R.; Arslan, T. Microkernel Architecture and Hardware Abstraction Layer of a Reliable Reconfigurable Real-Time Operating System (R3TOS). *ACM Trans. Reconfig. Technol. Syst.* **2015**, *8*, 5:1–5:35. [CrossRef]
18. So, H.K.H.; Brodersen, R. A Unified Hardware/Software Runtime Environment for FPGA-based Reconfigurable Computers Using BORPH. *ACM Trans. Embed. Comput. Syst.* **2008**, *7*, 14:1–14:28. [CrossRef]
19. Göhringer, D.; Hübner, M.; Zeutebouo, E.N.; Becker, J. 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, 19–23 April 2010; pp. 1–8.
20. Agne, A.; Happe, M.; Keller, A.; Lübbers, E.; Plattner, B.; Platzner, M.; Plessl, C. ReconOS: An Operating System Approach for Reconfigurable Computing. *IEEE Micro* **2014**, *34*, 60–71. [CrossRef]
21. Pellizzoni, R.; Caccamo, M. Real-Time Management of Hardware and Software Tasks for FPGA-based Embedded Systems. *IEEE Trans. Comput.* **2007**, *56*, 1666–1680. [CrossRef]
22. Hsiung, P.A.; Huang, C.H.; Shen, J.S.; Chiang, C.C. Scheduling and Placement of Hardware/Software Real-Time Relocatable Tasks in Dynamically Partially Reconfigurable Systems. *ACM Trans. Reconfig. Technol. Syst.* **2010**, *4*, 9:1–9:32. [CrossRef]
23. Tabkhi, H.; Schirner, G. Application-Guided Power Gating Reducing Register File Static Power. *IEEE Trans. Very Large Scale Integr. Syst.* **2014**, *22*, 2513–2526. [CrossRef]
24. Hosseinabady, M.; Nunez-Yanez, J.L. 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, 2–4 September 2014; pp. 1–6.
25. You, D.; Chung, K.S. Quality of Service-Aware Dynamic Voltage and Frequency Scaling for Embedded GPUs. *IEEE Comput. Arch. Lett.* **2015**, *14*, 66–69. [CrossRef]
26. Khan, M.U.K.; Shafique, M.; Henkel, J. Power-Efficient Workload Balancing for Video Applications. *IEEE Trans. Very Large Scale Integr. Syst.* **2016**, *24*, 2089–2102. [CrossRef]
27. Kornaros, G.; Pnevmatikatos, D. Dynamic Power and Thermal Management of NoC-Based Heterogeneous MPSoCs. *ACM Trans. Reconfig. Technol. Syst.* **2014**, *7*, 1:1–1:26. [CrossRef]
28. Carlo, S.D.; Gambardella, G.; Prinetto, P.; Rolfo, D.; Trotta, P. SATTA: A Self-Adaptive Temperature-Based TDF Awareness Methodology for Dynamically Reconfigurable FPGAs. *ACM Trans. Reconfig. Technol. Syst.* **2015**, *8*, 1:1–1:22. [CrossRef]
29. Lu, Y.H.; Benini, L.; Micheli, G.D. Low-power task scheduling for multiple devices. In Proceedings of the Eighth International Workshop on Hardware/Software Codesign, CODES 2000 (IEEE Cat. No.00TH8518), San Diego, CA, USA, 5 May 2000; pp. 39–43.
30. Yang, P.; Marchal, P.; Wong, C.; Himpe, S.; Catthoor, F.; David, P.; Vounckx, J.; Lauwereins, R. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In Proceedings of the 2002 15th International Symposium on System Synthesis, Kyoto, Japan, 2–4 October 2002; pp. 112–119.

31. Qiu, M.; Chen, Z.; Yang, L.T.; Qin, X.; Wang, B. 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, 25–27 June 2012; pp. 1466–1472.
32. Ganeshpure, K.; Kundu, S. Performance-driven Dynamic Thermal Management of MPSoC Based on Task Rescheduling. *ACM Trans. Des. Autom. Electron. Syst.* **2014**, *19*, 11:1–11:33. [CrossRef]
33. Ost, L.; Mandelli, M.; Almeida, G.M.; Moller, L.; Indrusiak, L.S.; Sassatelli, G.; Benoit, P.; Glesner, M.; Robert, M.; Moraes, F. Power-aware Dynamic Mapping Heuristics for NoC-based MPSoCs Using a Unified Model-based Approach. *ACM Trans. Embed. Comput. Syst.* **2013**, *12*, 75:1–75:22. [CrossRef]
34. Rodríguez, A.; Valverde, J.; Castañares, C.; Portilla, J.; de la Torre, E.; Riesgo, T. 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, 29 June–1 July 2015; pp. 1–8.
35. Lin, K.W.; Chen, Y.S. Online Thermal-aware Task Placement in Three-dimensional Field-programmable Gate Arrays. In Proceedings of the 2015 RACS Conference on Research in Adaptive and Convergent Systems, Prague, Czech Republic, 9–12 October 2015; ACM: New York, NY, USA, 2015; pp. 412–417.
36. Iturbe, X.; Benkrid, K.; Hong, C.; Ebrahim, A.; Arslan, T.; Martinez, I. Runtime Scheduling, Allocation, and Execution of Real-Time Hardware Tasks onto Xilinx FPGAs Subject to Fault Occurrence. *Int. J. Reconfig. Comput.* **2013**, *2013*. [CrossRef]
37. Biedermann, A.; Huss, S.A.; Israr, A. Safe Dynamic Reshaping of Reconfigurable MPSoC Embedded Systems for Self-Healing and Self-Adaption Purposes. *ACM Trans. Reconfig. Technol. Syst.* **2015**, *8*, 26:1–26:22. [CrossRef]
38. Xilinx. XAPP1088: Correcting Single Event Upsets in Virtex-4 FPGA Configuration Memory, v1.0. 2009. Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.169.426&rep=rep1&type=pdf (accessed on 1 July 2018).
39. Bolchini, C.; Miele, A.; Sandionigi, C. A Novel Design Methodology for Implementing Reliability-Aware Systems on SRAM-Based FPGAs. *IEEE Trans. Comput.* **2011**, *60*, 1744–1758. [CrossRef]
40. Salvador, R.; Otero, A.; Mora, J.; de la Torre, E.; Sekanina, L.; Riesgo, T. 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, 30 November–2 December 2011; pp. 164–169. [CrossRef]
41. Abramovici, M.; Breuer, M.A.; Friedman, A.D. Index. In *Digital Systems Testing and Testable Design*; Computer Science Press: New York, NY, USA, 1990; pp. 647–652.
42. Zhang, H.; Bauer, L.; Kochte, M.A.; Schneider, E.; Braun, C.; Imhof, M.E.; Wunderlich, H.J.; Henkel, J. Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures. In Proceedings of the 2013 IEEE International Test Conference (ITC), Anaheim, CA, USA, 6–13 September 2013; pp. 1–10. [CrossRef]
43. Vallero, A.; Carelli, A.; Carlo, S.D. 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, 9–12 April 2018; pp. 1–6. [CrossRef]
44. Carlo, S.D.; Gambardella, G.; Prinetto, P.; Rolfo, D.; Trotta, P.; Vallero, A. 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), Girona, Spain, 7–9 July 2014; pp. 87–92. [CrossRef]
45. Carlo, S.D.; Prinetto, P.; Scionti, A. A FPGA-Based Reconfigurable Software Architecture for Highly Dependable Systems. In Proceedings of the 2009 Asian Test Symposium, Taichung, Taiwan, 23–26 November 2009; pp. 125–130. [CrossRef]
46. Carlo, S.D.; Miele, A.; Prinetto, P.; Trapanese, A. Microprocessor fault-tolerance via on-the-fly partial reconfiguration. In Proceedings of the 2010 15th IEEE European Test Symposium, Praha, Czech, 24–28 May 2010; pp. 201–206. [CrossRef]
47. De Sensi, D.; Torquati, M.; Danelutto, M. A Reconfiguration Algorithm for Power-Aware Parallel Applications. *ACM Trans. Archit. Code Optim.* **2016**, *13*, 43:1–43:25. [CrossRef]

48. Sousa, E.; Hannig, F.; Teich, J.; Chen, Q.; Schlichtmann, U. Runtime Adaptation of Application Execution Under Thermal and Power Constraints in Massively Parallel Processor Arrays. In Proceedings of the SCOPES '15 18th International Workshop on Software and Compilers for Embedded Systems, St. Goar, Germany, 1–3 June 2015; ACM: New York, NY, USA, 2015; pp. 121–124. [CrossRef]
49. Loukil, K.; Amor, N.B.; Abid, M. 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, 23–26 March 2009; pp. 1–9. [CrossRef]
50. Wassi, G.; Benkhelifa, M.E.A.; Lawday, G.; Verdier, F.; Garcia, S. 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, 26–28 May 2014; pp. 1–7. [CrossRef]
51. Ullmann, M.; Jin, W.; Becker, J. Hardware Enhanced Function Allocation Management in Reconfigurable Systems. In Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Denver, CO, USA, 4–8 April 2005; p. 156a. [CrossRef]
52. Gueye, S.M.K.; Rutten, E.; Diguet, J.P. 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, 24–27 July 2017; pp. 48–55. [CrossRef]
53. Vipin, K.; Fahmy, S.A. 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, QC, Canada, 15–18 June 2015; pp. 1–8. [CrossRef]
54. Sharma, D.; Kirischian, L.; Kirischian, V. 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, 24–27 July 2017; pp. 32–39. [CrossRef]
55. Rihani, M.A.; Nouvel, F.; Prévotet, J.C.; Mroue, M.; Lorandel, J.; Mohanna, Y. Dynamic and partial reconfiguration power consumption runtime measurements analysis for ZYNQ SoC devices. In Proceedings of the 2016 International Symposium on Wireless Communication Systems (ISWCS), Poznan, Poland, 20–23 September 2016; pp. 592–596. [CrossRef]
56. Xilinx. Power vs. Performance: The 90 nm Inflection Point, v1.2. 2006. Available online: https://www.xilinx.com/support/documentation/white_papers/wp223.pdf (accessed on 1 July 2018).
57. Shang, L.; Kaviani, A.S.; Bathala, K. Dynamic Power Consumption in Virtex™-II FPGA Family. In Proceedings of the FPGA '02 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 24–26 February 2002; ACM: New York, NY, USA, 2002; pp. 157–164. [CrossRef]
58. Xilinx. ZedBoard Hardware Users Guide, v2.2. 2014. Available online: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf (accessed on 1 July 2018).
59. Xilinx. Zynq-7000 All Programmable SoC Overview, v1.10. 2016. Available online: https://cdn.hackaday.io/files/19354828041536/ds190-Zynq-7000-Overview.pdf (accessed on 1 July 2018).
60. Xilinx. KC705 Evaluation Board for the Kintex-7 FPGA, v1.7. 2016. Available online: https://www.xilinx.com/support/documentation/boards_and_kits/kc705/ug810_KC705_Eval_Bd.pdf (accessed on 1 July 2018).
61. Xilinx. 7 Series FPGAs Data Sheet: Overview, v2.5. 2017. Available online: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (accessed on 1 July 2018)
62. Xilinx. Vivado Design Suite User Guide—Partial Reconfiguration, v206.1. 2016. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug909-vivado-partial-reconfiguration.pdf (accessed on 1 July 2018).
63. Intel. Intel Quartus Prime Pro Edition Handbook Volume 1. 2017. Available online: https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/qts-qpp-handbook.pdf (accessed on 1 July 2018).
64. Meintanis, D.; Papaefstathiou, I. Power Consumption Estimations vs Measurements for FPGA-Based Security Cores. In Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 3–5 December 2008; pp. 433–437. [CrossRef]

65. Becker, J.; Huebner, M.; Ullmann, M. Power estimation and power measurement of Xilinx Virtex FPGAs: Trade-offs and limitations. In Proceedings of the 16th Symposium on Integrated Circuits and Systems Design, SBCCI 2003, Sao Paulo, Brazil, 8–11 September 2003; pp. 283–288. [CrossRef]
66. Oliver, J.P.; Acle, J.P.; Boemo, E. Power estimations vs. power measurements in Spartan-6 devices. In Proceedings of the 2014 IX Southern Conference on Programmable Logic (SPL), Buenos Aires, Argentina, 5–7 November 2014; pp. 1–5. [CrossRef]
67. Sharma, D.; Dimitriu, V.; Kirischian, L. Architecture Reconfiguration as a Mechanism for Sustainable Performance of Embedded Systems in case of Variations in Available Power. In *Applied Reconfigurable Computing(ARC 2017)*; Springer: Cham, Germany, 2017. [CrossRef]