Hindawi
International Journal of Reconfigurable Computing
Volume 2021, Article ID 8818788, 20 pages
https://doi.org/10.1155/2021/8818788

Research Article

A Method for Run-Time Prediction of On-Chip Thermal Conditions in Dynamically Reconfigurable SOPCs

Dimple Sharma and Lev Kirischian

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

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

Received 8 September 2020; Revised 7 December 2020; Accepted 18 December 2020; Published 7 January 2021

Academic Editor: John Kalomiros

Copyright © 2021 Dimple Sharma and Lev Kirischian. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Autonomous mobile systems nowadays deploy FPGA-based System on Programmable Chips (SoPCs) for supporting their dynamic multitask multimodal workloads. For such field-deployed systems, activation times, execution periods of tasks, and variations in environmental conditions are usually difficult to predict. These dynamic variations result in a new challenge of dynamic thermal cycling stress on the SoPC die, which can result in transient and even permanent hardware faults in the computing system. This paper proposes the approach of run-time structural adaptation (RTSA) to mitigate dynamic thermal cycling stress on the SoPC dies. RTSA assumes the tasks to have multiple implementation variants, called Application Specific Processing (ASP) circuit variants, which vary in hardware resources, operating frequency, and power consumption. Dynamically reconfiguring appropriate ASP circuit variants of tasks allow systems to maintain their die temperature in the desired range while taking into account variations in power budget and modes of operation. This means the essence of RTSA is a decision-making mechanism which can select at run-time, a suitable system configuration (set of ASP circuit variants of active tasks), whenever needed, to meet the die temperature constraints. To do so, run-time die temperature prediction for potential system configurations using an estimation model is required. This paper presents a generic method to derive an analytical model for any SoPC that can estimate the die temperature in real time and thus support the decision-making mechanism. To develop this method, the thermal behavior of SoPC die under different task scenarios is studied and relation of die temperature to frequency, resource utilization, and power consumption is analyzed. An RTSA-enabled experimental platform is set up on Xilinx Zynq XC7Z020 SoPC for this purpose. Experimental results also demonstrate that the proposed method can be used to derive a model in run-time, thus enabling systems to self-derive and dynamically update the model in run-time.

1. Introduction

Modern generation autonomous mobile systems such as mobile robotic systems, car driver assistance systems and autonomous cars, civil and military drones, satellites and planetary mission spacecraft, and unmanned submarine systems demand high-performance processing of multitask data-stream workloads. The workloads usually include multiple real-time video-streams, communication data-streams, LiDAR or radar data-streams, and acoustical and audio-streams. Such systems running complex applications have critical performance specifications. For example, a certain number of tasks must satisfy a performance range, not exceeding certain power consumption/dissipation limits and keeping on-chip and system thermal conditions in a determined range. Additionally, the number, performance, criticality, and functionality of the tasks can change in run-time based on the occurrence of external and/or internal events. This means systems are required to have multiple modes of operation to survive in different scenarios. Thus, we are looking at autonomous mobile systems supporting multimodal multitask stream processing workloads. On the other hand, considering the FPGA-based SoPC computing platforms on which such systems are deployed, there are always limitations of available resources such as (a) limited amount of platform hardware resources; (b) limited power generation, accumulation, and dissipation resources; (c) limited thermal range at on-chip, board, and system levels;

and (d) limited reliability of system components. It is necessary to mention that the above limitations are not only static but can dynamically change due to predicted or sudden events caused by different factors. For example, hardware resources are statically limited by restrictions of the available area, mass, or weight of the system. However, the dynamic workload may limit available resources for upcoming tasks. Hardware faults caused by radiation, thermal cycling effects, aging, or other factors can also limit certain amount of resources for some time (transient fault) or permanently. The same can be considered for power resources. There is an upper limit for available power that can be generated. However, over a period of time, reduction of fuel or available solar power, etc., can reduce the power budget dynamically. As well, the fault of the power generator or solar panels or other sources of power can suddenly reduce the power budget for the system. Therefore, autonomous mobile systems need to be able to sustain their dynamic workloads in the presence of dynamic changes in power budget, thermal conditions, and hardware resource conditions. Thus, for such systems, self-sustainability implies run-time adaptation to (a) dynamic multitask multimodal workload, (b) variations in power budget, (c) variations in thermal conditions, (d) external conditions affecting system performance, and (e) possible transient and permanent hardware faults. When a system itself can keep the above multiobjective requirements in the permitted range, it can continue its mission. Otherwise, the system may stop functioning permanently and that can cause domino effect for associated systems.

A system can adapt in three possible ways: (a) parametric, (b) behavioral, and (c) structural. All these forms of adaptation are widely used by different types of organisms from plants to almost all animals. Parametric adaptation is the simplest form of adaptation for systems. As the name suggests, it alters system parameters for adaptation, for example, power regulation, temperature control at on-chip and board levels, and change in operating frequency. Changing system parameters can help in run-time adaptation when changes in workload and environmental constraints are within the limits and resolution of the system parameters that can be changed. However, beyond those limits, other forms of adaptation must come to the rescue. Behavioral level of adaptation is expected to be provided by variation in workload. This can be achieved by varying the active set of tasks or changing the modes of the active tasks. There is a lot of work done in this area of workload management. However, with these existing solutions, it may not always be possible to satisfy the changing constraints on temperature, performance, power consumptions, and/or available resources along with dynamic changes in the workload [1]. This type of adaptivity is therefore not considered in this work. Run-time structural adaptation (RTSA) means that a system can change its architecture at run-time to adapt to the current: (a) multitask workload specifications, (b) power budget, (c) temperature range, and (d) hardware resource limitations. Since the system can adapt by changing its architecture, RTSA is the most flexible form of adaptation a system can use to sustain itself against multiple dynamic internal and external factors. Therefore, the presented research is focused on RTSA. The paper targets complex autonomous mobile systems processing multimodal multitask workloads where all or most of the tasks are computationally intensive by nature. The research is therefore applicable to run-time reconfigurable FPGA devices or FPGA-based SoPCs; they are the most suitable platforms to develop such systems.

For the discussed class of systems having computationally intensive workloads, each task is implemented in the form of a dedicated hardware circuit called ASP circuit. This circuit is based on the task algorithm and data structure and is designed using certain hardware description language like VHDL or Verilog. The ASP circuits of all the tasks are stored in the form of configuration bit-files (or bitstreams) for the targeted FPGA/SoPC device. Each task capable of RTSA must have several implementation variants, i.e., ASP circuit variants [2]. The ASP circuit variants of each task vary in resource utilization, operating frequency, performance, and power consumption. They are stored in the form of partial configuration bitstreams for the target FPGA in the system memory and can be configured/reconfigured on the partially reconfigurable regions (PRRs) of the FPGA as and when required [3]. Depending on the required system mode, a suitable system configuration, i.e., a set of suitable ASP circuit variants corresponding to the active set of tasks can be configured such that all the mode conditions are satisfied. The mode of operation assumes (a) required set of active tasks and their performance range, (b) available power budget, (c) die temperature limits, and (d) currently available hardware resources. For example, in a low power budget condition, ASP circuit variants of tasks which can operate at a lower frequency and occupy more resources as compared to the system configuration in high power budget condition can be configured [1]. This way, the performance of the critical tasks is not affected, but SoPC power consumption is reduced. Similarly, when hardware resources are scarce, ASP circuit variants that operate at a higher frequency and utilize lesser number of PRRs can be configured. Although this will increase the power consumption of the SoPC, it will allow more tasks in active set to run simultaneously in the limited resource condition.

Field-deployed mission-critical systems are complex systems with a very large number of tasks and their modes. To enable such systems with RTSA, their tasks will have a multitude of ASP circuit variants. With a huge number of tasks and their ASP circuit variants, a tremendously large design space of system configurations is formed. Therefore, the key aspect for making such systems capable of RTSA is a run-time decision-making mechanism that can find in run-time, the most efficient combination of ASP circuit variants of all active tasks that can fit in the available hardware resources, can provide acceptable power consumption, maintain the die temperature in the required range, and simultaneously satisfy the performance constraints of the tasks. The concept and initial experimental verification of the aforementioned multiobjective decision-making mechanism was done on the Xilinx Zynq 7000 family of FPGA devices [1, 4]. The mechanism presented in [1, 4] enables run-time adaptation to

system workload/mode, power budget, and hardware resource constraints. However, an important aspect yet to be considered is simultaneous mitigation of on-chip thermal cycling due to changes in external temperature or the current set of executing tasks. This means run-time adaptation to thermal changes in order to maintain the die temperature in the desired range needs to be incorporated for achieving self-sustainability to all the discussed parameters. It is necessary to mention that thermal cycling associated with (a) dynamic variations of the workload and (b) variations of external-to-FPGA device temperature has become one of the most significant reasons for transient and even permanent hardware faults in the flip-chip technology-based FPGAs. In addition to that, the thermal inertia of the SoPC is very different from the thermal inertia of the FPGA package connected to power dissipation units (e.g., heat sink and board layers) [5]. Thus, SoPCs incorporating dynamic multistream processing ASP circuits bring in the new problem of thermal stability of the SoPCs, absence of which may cause significant issues with system reliability. Therefore, the main aspect of the presented research is to provide the supporting tools to develop the discussed run-time decision-making mechanism that can enable RTSA in systems to dynamically maintain the die temperature in the desired range and thus provide temperature stability (the decision-making mechanism is not in the scope of this paper).

It can be foreseen that, for the run-time decision-making process, an analytical model for predicting die temperature will be needed. This is because the run-time decision-making mechanism will need to evaluate the die temperature of different candidate system configurations for selecting the appropriate ASP circuit variants for the active set of tasks that can maintain the die temperature in the desired range. For systems with a large design spaces, it is practically not feasible to measure and store the die temperature for all the system configurations in a lookup table (LUT). 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. This means die temperature for $20x16^5$ system configurations will need to be measured and stored in a large LUT during the system design phase. Although this is not practical, even if this much information is stored in a LUT, the search time would be very large to make it useful at run-time. Furthermore, any addition or modification of system modes, tasks, or their variants will imply redoing the entire offline process all over again! Thus, it is necessary to have an analytical model which can estimate the die temperature of system configurations under evaluation at run-time so that the decision-making system can select the most appropriate configuration for RTSA. An analytical model is required since we are looking at a model that can give results in run-time, i.e., within units of seconds or even less. Only a mathematical equation can satisfy this condition. It saves time, memory, and hardware resources and meets the run-time requirements. It is thus necessary to create an efficient methodology that can derive an analytical die temperature estimation model for a FPGA platform and the set of tasks running on it. Certainly, different FPGAs supporting different applications will result in different thermal model coefficients. Therefore, the methodology must be able to derive a model for any FPGA device and application pair.

This paper presents a methodology to derive a simplified run-time mathematical model that can estimate the die temperature for any given system configuration to be deployed in any partially reconfigurable FPGA device or FPGA-based SoC. Along with the feature of being a universal method for any FPGA device, the proposed method is also simple enough for systems to self-derive the die temperature estimation model for themselves in run-time. For example, consider a system developed on a Xilinx Zynq XC7Z020 SoC connected with a Xilinx Kintex-7 FPGA device. The proposed generic method to derive the die temperature estimation model can be programmed in the ARM-Cortex A9 core of the Zynq device. Using this code, the system can self-derive a die temperature estimation model for the programmable logic (PL) region of the Zynq device and the Kintex-7 device for the application being supported. If the FPGA device in use changes, or the application changes, the system can use the same programmed generic method once again to derive the new model parameters for the new FPGA device and/or application. Thus, offline manual derivation of model coefficients is not required.

Development of a die temperature estimation model requires a detailed investigation of on-chip thermal behavior of the dynamically reconfigurable SoPCs in different workload scenarios. The complexity of this behavior depends on the resource utilization and operating frequencies of the different ASP circuits running on the die and the dynamic variations of external-to-FPGA environmental temperature. This paper, therefore, conducts a detailed study of the die temperature behavior in different multitask scenarios. It analyzes the relationship of the die temperature to factors such as resource utilization, frequency, and power consumption in detail. It closely observes the behavior of heat distribution in the FPGA die and the pattern of temperature rise and fall in different multitask cases implemented on Xilinx partially reconfigurable FPGA platform (Xilinx Zynq 7000 family).

Thus, the novel contributions of the paper are as follows:

(a) A detailed analysis of the thermal behavior of SoPC die when different ASP circuits of tasks are operating at different frequencies in different PRRs of the die is presented. Relation of die temperature to factors such as frequency, resource utilization, and thus power consumption is observed.

(b) A methodology is presented to derive a mathematical model that can predict the die temperature for any system configuration in run-time. The method is generic and can be used to derive a model for any FPGA-based SoPC device.

(c) An RTSA-enabled experimental setup is developed that is used to study the thermal behavior of the

SoPC die and to develop the methodology to derive the run-time die temperature estimation model. Xilinx Zynq XC7Z020 SoPC housed on the ZedBoard is used for this purpose. The setup can be used for any RTSA-related experiments along with being used for the context of this paper.

The related literature observation is provided in Section 2. Section 3 briefly describes the SoPC on-chip framework to be configured in the target FPGA to support the RTSA process. The organization of RTSA-enabled experimental setup to study the die temperature behavior in multitask scenario is presented in Section 4. Sections 5 and 6 provide details of the various experiments conducted and results of the investigation of the FPGA die thermal behavior. Section 7 studies the behavior of total power consumption of the die to analyze it with respect to the thermal behavior of the die. Section 8 presents generalized steps for deriving the die temperature estimation model for any SoPC deployed in a partially reconfigurable FPGA platform. The methodology for the model derivation is validated on the Zynq XC7Z020 device in Section 9. Section 10 presents an example of how the die temperature estimation model can be used in run-time to carry out structural adaptation to dynamically maintain the die temperature in the desired range whenever needed, while simultaneously maintaining the performance of the tasks. Section 11 concludes the results of this research.

2. Literature Review

In this paper, we have reasoned why a run-time decision-making mechanism needs to be developed for mitigating thermal dynamics on a SoPC die. This mechanism can dynamically select a suitable system configuration, i.e., a set of ASP circuit variants for the active tasks such that the desired FPGA die temperature is maintained. The major advantage of this mechanism will be the following: once integrated with the decision-making mechanism proposed in [1], it will result in a complete multiobjective mechanism, which will allow autonomous mobile systems to adapt and sustain their dynamic workloads in presence of changing power budgets, system modes, temperature, and available hardware resources, at run-time! We have, therefore, observed the current literature from the same perspective; whether the thermal management methods proposed in recent research works can support multiobjective run-time adaptation in systems. Although the suggested decision-making mechanism for maintaining FPGA die temperature is not in the scope of this paper, we have shown that a requirement for such a mechanism is a model that can estimate the die temperature of any system configuration under consideration at run-time. Based on this necessity, we are proposing a method to derive a mathematical model for run-time die temperature estimation. Hence, along with the above-discussed viewpoint of literature review, we have also studied the recent research efforts for predicting the die temperature.

Dynamic frequency scaling (DFS) is a well-known method for temperature management, where the operating frequency of tasks is increased or decreased to increase or decrease the die temperature. DFS has been used for almost all kinds of application platforms. It has been observed to be used for tasks running on soft-core processors [6], for tasks executing on the FPGA die [7], for multicore processor-based SoCs (MPSoCs) [8], and in NOC-based MPSOCs [9, 10]. Although the method is commonly used, it has an inherent disadvantage of affecting the performance of the executing tasks. It is therefore best suited for applications that are tolerant to changing performance of the tasks. For example, tasks running on processors that may have soft deadlines for completion or which may have enough slack time to allow reduction in their performance can make good use of DFS. However, DFS does not fit well for systems supporting computationally intensive tasks with strict performance constraints, executing as dedicated ASP circuits. Additionally, DFS alone will not be able to support run-time multiobjective adaptation in systems. Although it can be used for both power and thermal management, since the tasks have fixed hardware circuits (no ASP circuit variants), it will not always be able to meet the power budget and temperature conditions. It also cannot support adaptation to varying hardware resources (due to faults) and changing workloads due to changing modes. In order to support these objectives, it will again require ASP circuit variants of tasks such that the performance of tasks can be maintained while changing their frequency using the appropriate task variants. This points to run-time structural adaptation as a solution to develop self-sustaining systems.

Other thermal management methods depend on the tasks being supported. The required temperature profile is achieved by scheduling tasks, managing (suspending or turning on) tasks, allocating task workloads to specific regions, or migrating them between regions of the die accordingly [11–16]. Such methods are dependent on the type of executing tasks and therefore are more suited for processor-based tasks and which have fixed periods of execution. In such cases, it becomes easier to implement the scheduling or allocation policies. However, autonomous mobile systems support tasks which begin or stop execution based on several unpredictable workload or environmental conditions. Task-based thermal management techniques may not always work for such systems. It is because they may not always be able to bring the die temperature at the desired level. They can minimize the temperature by scheduling/allocating the executing tasks, but that minimum temperature achieved may not be the desired die temperature to be maintained at that point of time. Also, with such task-dependent adaptation methods, multiobjective adaptation to varying workload and environmental conditions cannot be achieved.

The use of self-heating circuits is also observed to increase/decrease the die temperature [17–19]. These are dummy circuits which are distributed in different regions of the die; the numbers and location depend on the die temperature to be achieved. A very apparent disadvantage is the use of extra hardware resources to heat up the die. Systems supporting critical multitask multimodal applications face a challenge to manage their ever-increasing workloads on the

limited available resources. In such a case, trying to allocate extra resources just for heating up the die is not an acceptable solution.

All the above observations in a way show that run-time structural adaptation using a decision-making mechanism is the best possible way for developing a completely self-sustaining system. Using ASP circuit variants for tasks and selecting the appropriate variant as and when the set of conditions change can achieve run-time multiobjective adaptation. As discussed in the Introduction section, this calls for a run-time model to predict the die temperature for potential system configurations. However, not many research efforts are seen in the direction of estimating or predicting the die temperature, especially in a run-time scenario. Most methods observed are temperature sensing methods. A variety of ring-oscillator-based sensors and sensors based on other circuits have been developed, which sense the temperature profile of the die. Based on this, different dynamic adaptation methods can be applied [20–29]. The use of sensors assumes the use of task-based thermal management techniques. Based on the temperature distribution obtained from the sensors, the tasks are either rescheduled or reallocated or migrated to achieve the desired temperature profile. Happe et al. [24] discuss a run-time temperature prediction model, whose parameters are learned at run-time from different measurements made using sensors. The model is proposed to be used for mapping task threads between processor cores and hardware circuits based on the temperature predictions. When considering dynamic multiobjective adaptation, the use of sensors is not an effective solution due to the following reasons:

(a) Since several sensors are deployed in different locations of the die, they use up extra hardware resources, which is a scarce resource for autonomous mobile systems.

(b) The sensors need a system configuration to be programmed on the die to observe the die temperature for that configuration. For selecting an appropriate system configuration at run-time, it is not possible to program every possible system configuration, sense the temperature, and then decide a suitable configuration that maintains the die temperature in the required range.

(c) Since the observed sensors are primarily temperature sensors, their use alone cannot help to achieve the goal of run-time multiobjective adaptation.

While observing the existing thermal models, they are not suitable either for run-time use or for FPGA-based devices. Castilhos et al. [30] present a run-time software model that can predict the temperature distribution in MPSoCs. This model is very specific to the instruction-based processors in the MPSOCs and therefore cannot be applied to a FPGA die in general, which supports tasks running in the form of dedicated hardware circuits. Vendor tools like Xilinx Power Estimator [31] and Intel FPGA Power and Thermal Calculator [32] are commonly used for temperature prediction of FPGA/SoC devices. However, these tools can be used offline to predict the temperature for created designs since they use complex and detailed thermal models that require parameters like ambient temperature, switching activity factor of the design and the individual resource types used in the design, board setup, and heatsink parameters. These models, therefore, take time varying from minutes to hours depending on the size of the designs, to predict temperature. Due to this, they are unsuitable for run-time temperature prediction, which requires a model to provide results within fractions of seconds. State-of-the-art tools like HotSpot [33–35] make use of accurate thermal models that predict the temperature for processor chips at the microarchitecture level using the packaging and floorplan information of the chips. Again, such efficient models can help develop temperature-aware systems at design time. The thermal simulations when included in the design flow can result in thermally optimized designs. However, since the models are targeted for processor chips and involve detailed simulations, they cannot be applied for run-time die temperature prediction in SoPC devices. Thermal models have also been developed for SoC devices and 3D integrated silicon chips, as in [36, 37], which predict the temperature at different granularity levels and at different stages of design flow. These models too can only be used to develop thermal-aware or thermally optimized designs at design time and cannot be applied for run-time prediction. Other observed temperature estimation methods are design and device-specific methods [38] and hence cannot be used for run-time temperature prediction.

Luo et al. [39] present thermal models that can predict the temperature of a multi-FPGA system consisting of 4 FPGAs, for a given set of tasks in run-time. Machine learning algorithms are used to derive the thermal models. The models fit well in the context of this paper and are suited for run-time die temperature prediction. A set of 10 benchmarks, allocated in different combinations on the 4 FPGAs, thus resulting in 5040 samples, are used to train and test the machine learning-based thermal models. Analyzing the models from the point of view of this paper, if a run-time die temperature prediction model needs to be obtained for a FPGA device running a multitask multimodal application using a machine learning algorithm, there are no other training samples for the model except for the application itself. This is because every application has its unique set of switching activity factors of its inputs, outputs, and the resources utilized, which affects the power consumption and hence the die temperature. Using other applications/designs for training the model will not lead to accurate model coefficients for the required application. It is also seen in [39] that although the models have a good accuracy when the benchmarks are a part of both the training and testing samples, their accuracy is lower when the training and testing samples use different independent benchmarks. While this is acceptable for the purpose that the models are being used for in [39], a low accuracy in the range of 7–12°C is not suitable for RTSA, i.e., while finding a system configuration that fits in the specified temperature range (which can change dynamically). Additionally, the tuning parameters used for the models in [39] are very specific to the used

multi-FPGA system. To use the models on a different platform, the tuning parameters need to be changed; their values depend on the user and the system. This means there is no generic/standard guideline that can be followed for the machine learning algorithm chosen, when the system changes. Thus, although the thermal models presented in [39] are a very good solution for run-time temperature prediction, they cannot be applied for the purpose of RTSA.

From the discussion in the Introduction section, what is required is a simplified method that can directly derive the coefficients of the run-time die temperature estimation model from data collected from actual measurements. This paper presents such a method to derive a mathematical model for run-time die temperature estimation for a FPGA device and the application being supported on it. Since the coefficients are derived from actual measurements on the FPGA, they correspond to the FPGA device and its associated application and are accurate enough to serve the purpose of RTSA. The proposed method is generic and hence can be used to derive the model coefficients for any FPGA and application pair.

3. Architecture of SOPC Framework Capable of RTSA

A SoPC which can accommodate RTSA must have an underlying architecture that supports the RTSA process. This means, there must be a framework which allows a SoPC to change its architecture (set of ASP variants of active tasks) dynamically. Due to the multitask nature of the considered SoPC, such framework must be able to accommodate multiple ASP circuits of tasks and be able to change architecture of one ASP circuit without interrupting or stalling other ASP circuits. Thus, the entire area of hardware resources should be virtually divided into a number of identical PRRs called as slots. The ASP circuit of a task can be configured in one or multiple slots. The portion of the ASP circuit configured in one slot is called ASP component or Collaborative Macro-Function Unit (CMFU) of the ASP. The number as well as reconfigurable partition boundaries of the slots can be defined during the SoPC design phase based on the largest ASP component that needs to fit in a slot. The method for this design procedure is specific for each type of FPGA and associated CAD system (e.g., [40]). All the above SoPC slots in the framework should have the ability to communicate with each other in two ways: (a) transmission of their data-streams from the ASP component, i.e., CMFU of the upstream task to the CMFU of the downstream task, and (b) to exchange control and synchronization information between different CMFUs of the same task. Furthermore, when ASP circuit of a task needs to be configured/reconfigured, this framework must support the configuration of the required CMFUs in available slots and their self-assembling to form a complete functional circuit. The framework must also allow flexible component relocation between slots. Such a framework called "Multi-mode/task Adaptive Collaborative Reconfigurable self-Organized System (MACROS)" framework has been developed to support RTSA. It has been implemented and tested for multi-video-stream applications on Xilinx Zynq 7000 and Kintex-7 families of FPGA platforms. The detailed description of this concept and MACROS SoPC organization is discussed in [41].

As shown in Figure 1, the entire area of hardware resources available on the target FPGA is divided into several slots (PRRs). ASP circuits can be configured in dedicated slots by loading the partial configuration bit-files of the ASP components to the configuration memory segments corresponding to those slots. All slots are connected to each other through the Distributed Communication and Control Infrastructure (DCCI). DCCI works as a universal Network-on-Chip (NoC) for the SoPC and consists of (a) a crossbar, (b) a set of Local Connection and Control Units (LCCUs), (c) a control data broadcast network, and (d) a system mode broadcast bus. This framework allows any ASP component (CMFU), allocated in any slot of the SoPC to directly communicate to other ASP components of the same or different tasks over associated LCCUs. As well, it is possible for ASP components to change the data transfer link to any other slot of the same ASP circuit or ASP circuits of other tasks as and when necessary. Since all links are direct between components, it is possible to deploy ASP circuits of several tasks to run simultaneously on the SoPC without interference or suspension of any circuits in case of configuration/reconfiguration or switching data links in any of the SoPC slots. For this purpose, the SoPC infrastructure contains a special dedicated circuit, Bit-file and Configuration Management (BCM), to synchronize the slot re/configuration process with the computation processes in other slots accommodating other ASP components. All the details of this framework are presented in [41].

4. RTSA-Enabled Experimental Setup

To develop a mathematical model for predicting the die temperature for a given set of executing tasks, the thermal behavior of the die and what factors the behavior depends on needs to be observed and analyzed. Since the focus is on partially reconfigurable SoPC platforms supporting multitask multimodal workloads, multiple questions need to be answered:

(i) How does the heat spread when tasks are configured in all the PRRs of the die?

(ii) What is the behavior of rise in temperature in such a case?

(iii) What happens to the die temperature when there are spare PRRs along with tasks executing in other PRRs?

(iv) How is the heat distribution and thermal behavior when the multiple executing tasks have the same frequency?

(v) What if the multiple tasks have different frequencies?

(vi) What kind of relation can be developed between temperature and frequency and/or resource utilization and/or power consumption?

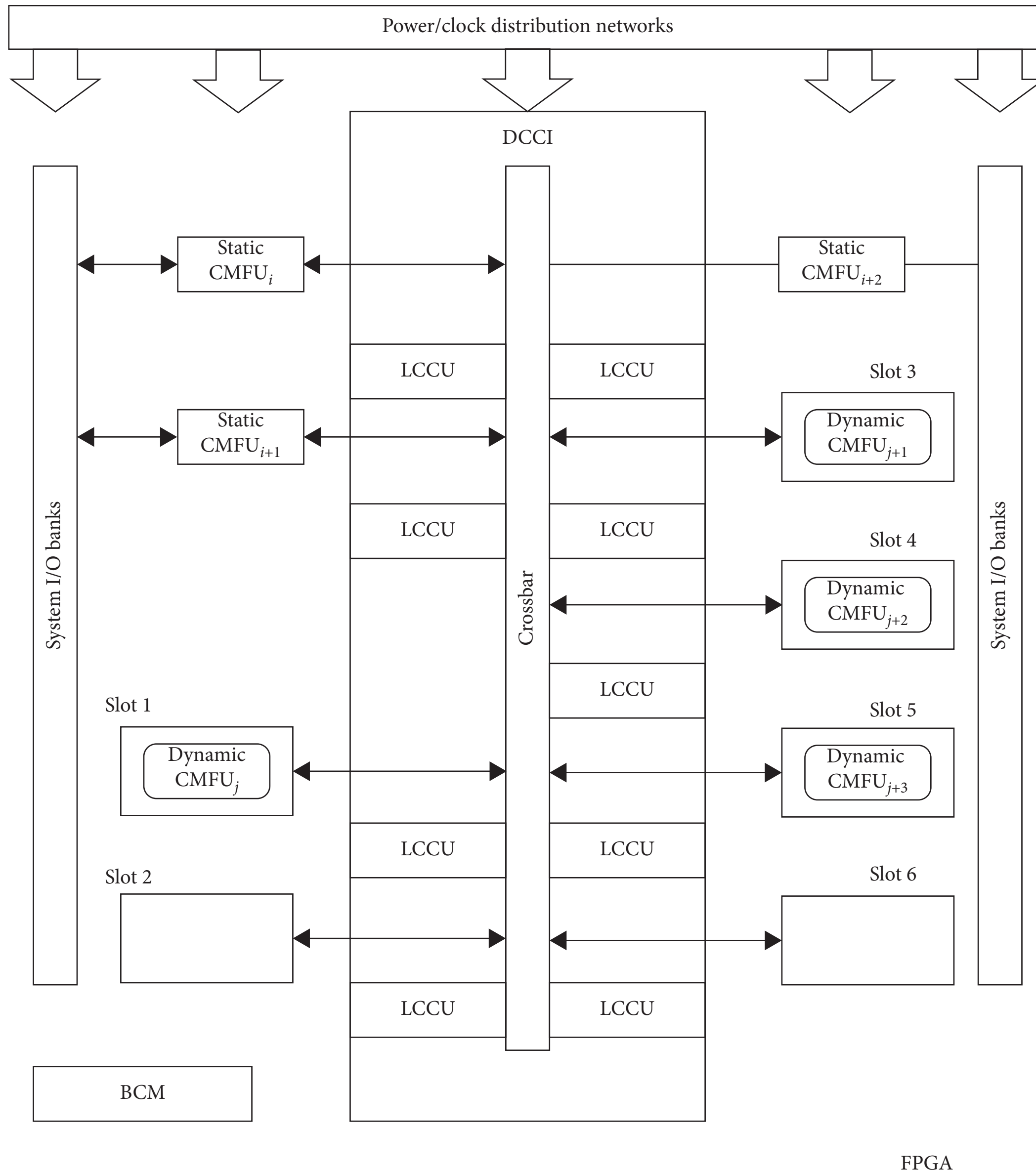

Figure 1: Architecture of MACROS framework.

To find answers for all these questions, several experiments need to be conducted. It is best to have an experimental setup so that all the required experiments can be conducted on the same setup. This way, it would be easier to relate and analyze the results of the various experiments. Also, since this is an ongoing research, it would be ideal to have the same setup even for future experiments. To serve the above purpose, an experimental setup is developed which has an architecture that allows run-time structural adaptation. To start with, the requirements of such a setup are as follows:

(i) A recent SoPC device, capable of DPR, is required for all the experimental evaluations

(ii) There must a mechanism to monitor the die temperature

(iii) There must be a mechanism to measure the total power consumption of the die

(iv) The reconfigurable area of the SoPC die must be deployed with the MACROS framework to support RTSA

(v) There must be test tasks such that their partial bitstreams can be configured in any PRR of the die whenever required

(vi) The tasks must be capable of operating at different frequencies

(vii) There must be a mechanism to dynamically change the operating frequencies of the individual tasks as needed

If the above requirements are met, it becomes possible to conduct experiments using any number of tasks as and when needed. Their locations and frequencies can also be varied dynamically. The MACROS framework can ensure seamless communication and synchronization between the executing tasks. Power consumption and die temperature can simultaneously be measured for any experiment conducted. Thus, a run-time structurally adaptable experimental infrastructure is developed. Based on the above requirements, the RTSA-enabled experimental setup is developed as follows.

The SoPC under test used for the RTSA-enabled experimental setup is the Xilinx Zynq XCZ7020 [42] (referred as Zynq in the rest of the paper), housed on the ZedBoard [43]. The ZedBoard has a means to measure its total power consumption. There is a current sense resistor of 10 mΩ in series with its 12 V power supply. Voltage measured across the resistor can provide the total current consumption, using which the total power consumption of the board can be calculated. If only the on-chip resources are utilized for the experiments, the total power consumption of the board can be considered as the total power consumption of the die. The Zynq device has a temperature sensor that can measure the average die temperature. The sensed temperature can be converted into a digital value by the XADC (analog to digital temperature) hard-core [44] on the Zynq device. The ARM-Cortex A9-based Processing System (PS) can read this value from the XADC core with the help of a C-code on Xilinx SDK and convert the digital value back to temperature. Thus, experiments to observe the thermal behavior and power consumption of the die under multiple scenarios can be conveniently carried out using the ZedBoard. This makes the ZedBoard an appropriate choice for a lab environment.

A partially reconfigurable architecture has been developed on the die with the MACROS framework [41], as shown in Figure 2. The Zynq PL has four PRRs (PRR_1 to PRR_4) where different ASP circuits can be programmed. The PRRs are equal in size and include the same amount of reconfigurable resources. Three partially reconfigurable computational tasks (T_1, T_2, and T_3) have been designed, which are arithmetic functions that include all the types of reconfigurable resources, i.e., Logic Slices, BRAM slices, and DSP slices. The difference between the tasks is the amount of each type of resource utilized and, hence, the area they occupy in a PRR. Three tasks with different resource utilization are developed so that we can have more data to understand the thermal behavior of the die and the relation between temperature and resource utilization, frequency, and power consumption. A task can be programmed to run on any frequency in the range of 30 MHz to 180 MHz, in multiples of 30 MHz. Four clock outputs of a PLL, named, $clock_0$ to $clock_3$, are used for this purpose. Each clock output feeds a task in one PRR. A memory and time optimized C-code developed using Xilinx SDK is used to program the frequency of the task in each PRR. Based on the combination of frequencies provided, the code finds the values of the PLL registers that need to be updated to achieve the required frequency combination, from a lookup table (LUT), and then programs the PLL registers. Instead of involving the PS section of the Zynq device, the clock frequency for a task circuit in each PRR could also be changed using the programmable logic (PL) region of the Zynq device. The Dynamic Reconfiguration Port (DRP) [44] could be used to talk to the XADC. However, that could affect the die temperature being monitored during the experiments. Therefore, the PS part of the device is used to change the clock frequencies. The logic used to optimize the C-code that programs the required frequency combination for the executing tasks is discussed as follows.

4.1. *Description of Memory and Time Optimized Code Logic to Set Operating Frequencies for Tasks in Different PRRs.*

Since a task in each PRR can be configured to run at one frequency from six possible frequencies (30 MHz to 180 MHz), there are $6 \times 6 \times 6 \times 6 = 1296$ possible combinations of frequencies for tasks in the four PRRs. Having a single LUT to store the PLL register values would require 1296 entries for each PLL register. If the clock output phase and duty cycle are kept to default values, there are five 32-bit registers that need to be programmed to configure the frequencies of the tasks in the four PRRs [45].

Clock Configuration Register 0 decides a common divider value for all the output clocks, and Clock Configuration Registers 2, 5, 8, and 11 decide the divider values for $clock_0$ to $clock_3$, respectively. This means the total LUT memory that would be required is 1296 entries × 5 PLL registers/entry × 4 bytes/register = 25920 bytes ≈26 kB.

The LUT size can be reduced by breaking down the LUT into multiple small LUTs, eliminating redundant frequency combinations, and storing only the divider values instead of specific PLL register values. Reducing LUT size saves memory and the search time within the LUT to access the required value.

When deciding divider values for the four clock outputs of the PLL, what matters is the combination of frequency and not the order of frequency according to PRRs. For example, a combination of 30, 60, 90, and 120 MHz means $clock_0$ (which feeds the task in PRR_1) is 30 MHz, $clock_1$ (which feeds the task in PRR_2) is 60 MHz, $clock_2$ (which feeds the task in PRR_3) is 90 MHz, and $clock_3$ (which feeds the task in PRR_4) is 120 MHz. Similarly, a combination of 120, 90, 30, and 60 MHz means $clock_0$ is 120 MHz, $clock_1$ is 90 MHz, $clock_2$ is 30 MHz, and $clock_3$ is 60 MHz. There can be a total of 24 such arrangements of the four frequencies. However, with respect to finding divider values from an LUT, all the 24 combinations are the same. It is because the only difference in these combinations is which register will be programmed with which divider value. In the first case, Clock Configuration Register 2 for $clock_0$ output will be programmed for 30 MHz whereas in the second case, Clock Configuration Register 8 for $clock_2$ will be programmed for 30 MHz. Using the above logic, the LUT size can be reduced as follows.

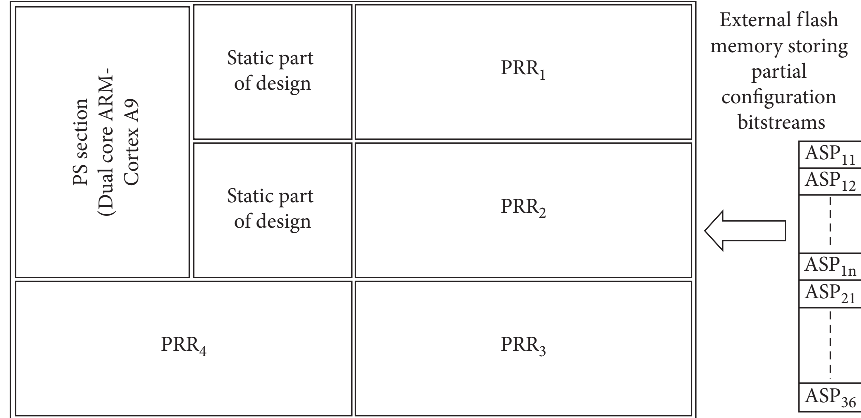

Figure 2: RTSA-enabled experimental platform.

The given combination of frequencies can be sorted in a particular order (ascending or descending) and their corresponding clock output indices (i.e., which clock output they correspond to) can be stored. Divider values can then be obtained from the LUT for the sorted combination of frequencies. Once that is done, the corresponding PLL register values can be mapped using the stored indices and programmed in the PLL. For example, suppose the combination of frequencies is 90, 120, 180, and 60 MHz; the sorted combination in ascending order is 60, 90, 120, and 180 MHz. The corresponding clock output indices are stored as 3, 0, 1, and 2 (means $clock_3 = 60$ MHz, $clock_0 = 90$ MHz, $clock_1 = 120$ MHz, and $clock_2 = 180$ MHz). Let the divider values for the sorted combination found from the LUT be D_0, D_1, D_2, and D_3. These divider values are mapped to the corresponding PLL register using the stored indices. This means D_0 is programmed for $clock_3$, i.e., for Clock Configuration Register 11. Similarly, D_1 is programmed for $clock_0$, i.e., for Clock Configuration Register 2, and so on. For any other combination of the above frequencies from the 24 possible combinations, the only change is the map of clock indices. The divider values from the LUT remain the same. This way, redundant entries can be eliminated from the LUT and its size can be drastically reduced. Thus, the large LUT of 1296 entries can be broken down into the following small LUTs:

(i) LUT_1: all four clock frequencies are different, e.g., 90, 120, 30, and 60 MHz. This combination after sorting in ascending order becomes 30, 60, 90, and 120 MHz. For a frequency range from 30 MHz to 180 MHz, there are $\binom{6}{4}$ such sorted possible combinations = 15 combinations.

(ii) LUT_2: two clock frequencies are the same, e.g., 30, 90, 60, and 30 MHz. This combination after sorting in ascending order becomes 30, 60, and 90 MHz. For a frequency range from 30 MHz to 180 MHz, there are $\binom{6}{3}$ such sorted possibilities = 20 combinations.

(iii) LUT_3: two clock frequencies are the same and the remaining two clock frequencies are the same, e.g., 30, 60, 60, and 30 MHz. This combination after sorting in ascending order becomes 30 and 60 MHz. For a frequency range from 30 MHz to 180 MHz, there are $\binom{6}{2}$ such sorted possibilities = 15 combinations.

(iv) LUT_4: three clock frequencies are the same, e.g., 30, 60, 30, and 30. This combination after sorting in ascending order becomes 30 and 60 MHz. For a frequency range from 30 MHz to 180 MHz, there are $\binom{6}{2}$ such sorted possibilities = 15 combinations.

(v) LUT_5: all clock frequencies are the same, e.g., 30, 30, 30, and 30 MHz. For a frequency range from 30 MHz to 180 MHz, there are only $\binom{6}{1}$ such possibilities = 6 combinations.

Thus, total number of combinations are $15 + 20 + 15 + 15 + 6 = 71$, as against 1296 in the above unoptimized case. In this case, the LUTs need to store only divider values and not actual PLL register values. The number of divider values to be stored in the LUT is equal to the number of different frequencies that the LUT corresponds to. This means LUT_1 has four 32-bit divider values, LUT_2 has 3 divider values, LUT_3 and LUT_4 have 2 divider values, and LUT_5 has only one divider value. Additionally, each LUT also has another 32-bit divider value for Clock Configuration Register 0, for each frequency combination in that LUT. Thus, size of LUT_1 = 15 entries × 5 divider values/entry × 4 bytes/value = 300 bytes. Similarly, size of LUT_2 = 320 bytes, size of LUT_3 and LUT_4 = 180 bytes, and size of LUT_5 = 48 bytes. The total LUT size now becomes $300 + 320 + 180 + 180 + 48 = 1028$ bytes, slightly more than 1 kB. Practically treating it as 2 kB, there is a reduction in the LUT size by a factor of 13!

In the experimental platform used for this paper, the reconfigurable region of Zynq is divided into 4 PRRs, as a result of which the above discussed LUT size reduction is

achieved. The same logic can be used for a larger number of PRRs and a larger range of task frequencies. A much higher LUT size reduction factor will be observed. Thus, the approach used for developing the LUT-based C-code efficiently utilizes the system memory and search time to dynamically program the desired combination of task frequencies in the four PRRs.

To summarize, an RTSA-enabled experimental platform is developed such that three partially reconfigurable tasks can be configured dynamically in any of the four PRRs. Using the memory and time optimized code, the tasks in the four PRRs can be programmed with any frequency combination to conduct the planned experiments, which are discussed in the next section.

5. Investigation of Die Temperature Behavior in Multitask Scenario

The paper [5] observes the thermal behavior of the die when an ASP circuit of a task is configured only in one PRR of the FPGA, when the operating frequency of the task is varied, and when the ASP circuit of the same task is relocated to different PRRs. The following thermal behavior has been noted.

The FPGA die has a very fast heat distribution, as a result of which the PRR location in which a task's ASP circuit is configured plays no role in influencing the die temperature. For a given ASP circuit of a task, the die temperature rises for ≈10 minutes and then saturates in a temperature range which is the same irrespective of the PRR location. The saturation temperature is higher when a task operates at a higher frequency. When a task is no longer running on the PRR, the die temperature cools down to a lower temperature within a minute and then continues to remain at that temperature. The paper [5] thus provides an insight into the heating/cooling behavior when a task's ASP circuit is configured in a single PRR of the FPGA. However, in actual field-deployed systems, there are multiple tasks executing in multiple PRRs at different frequencies. Due to the varying workload, the number of ASP circuits deployed in the available PRRs can also vary, i.e., all the PRRs could be occupied or some could be spare. The thermal behavior of the die in such a scenario where multiple tasks are executing at different frequencies in some or all PRRs still needs to be observed. Also, the thermal behavior needs to be analyzed mathematically from the perspective of a potential model that can estimate the die temperature for any system configuration. Experiments have been planned to analyze the discussed context.

For this set of experiments, ASP circuits of the tasks (T_1, T_2, and T_3) are configured in different PRRs at different frequencies to form different system configurations. A greybox [3] is configured at 30 MHz in the PRR where no task is executing. For each system configuration, the die temperature is observed along with the total power consumption every 30 seconds for a span of 20 minutes. Since the cool-down behavior is already obtained from [5], it would remain the same even for this set of experiments. This is because the cool-down phase has the same configuration of greyboxes deployed in all the PRRs operating at 30 MHz irrespective of the system configuration deployed prior to it. Therefore, observation of the cool-down phase is not repeated in this set of experiments. The following system configurations have been tested:

(1) T_1 in PRR_1 @ 90 MHz, and T_2 in PRR_2 @90 MHz
(2) T_1 in PRR_2 @ 120 MHz, and T_3 in PRR_4 @ 120 MHz
(3) T_1 in PRR_1 @ 90 MHz, T_2 in PRR_2 @ 120 MHz, T_3 in PRR_3 @ 60 MHz, and T_3 in PRR_4 @ 120 MHz
(4) T_3 in PRR_1 @ 90 MHz, T_2 in PRR_2 @ 120 MHz, T_2 in PRR_3 @ 120 MHz, and T_1 in PRR_4 @ 60 MHz
(5) T_2 in PRR_1 @ 120 MHz, T_1 in PRR_2 @ 60 MHz, and T_3 in PRR_4 @ 90 MHz

Although the choice of a task and its operating frequency in a PRR are random, the above configurations are chosen so that we have

(i) Cases of tasks running at different frequencies (configurations 3, 4, and 5) and cases where all tasks are running at the same frequency (configurations 1 and 2)
(ii) Cases where one PRR is spare, i.e., ASP circuits of three tasks are running on three PRRs (configuration 5), two PRRs are spare, i.e., ASP circuits of two tasks are running on two PRRs (configurations 1 and 2), and no spare PRR is available, i.e., ASP circuits of tasks are running on all PRRs (configurations 3 and 4). Note: the case for three spare PRRs, i.e., ASP circuit of a task running on one PRR is already studied in [5].

Figure 3 shows the temperature vs. time curves for each system configuration. The nature of curves obtained for each configuration is the same as the thermal behavior observed in [5]. The temperature rises for around 10 minutes and then saturates within a temperature range. Here, we take a closer look at the curves obtained in Figure 3 to get a detailed idea of how the temperature of the die rises and reaches saturation. The curves can be split into three regions. In the first 1.5–2 minutes, the temperature rapidly rises with an approximate slope (call it $slope_1$) of 3.5 to 4°C per minute. After that, from around 1.5 to 6 minutes, the temperature rise slows down and the slope (call it $slope_2$) changes to ≈1°C per minute. After the 6–6.5 minutes, the temperature saturates at the level reached and stays there within a range of +2°C for the remaining time the configuration is running.

The slopes of temperature rise follow the above-discussed pattern irrespective of the number, type, location, and frequency of the tasks configured on the FPGA die. Consider the following: when system configuration 3 is running on the die, the temperature rise curve has $slope_1 = 4$ and $slope_2 = 1$, as can be seen in Figure 3. When each task of configuration 3 is programmed alone in a single PRR, similar slope values are observed:

(i) T_1 configured in PRR_1 alone @ 90 MHz: $slope_1 = 4$ and $slope_2 = 1$

(ii) T_2 configured in PRR_2 alone @ 120 MHz: $slope_1 = 3.5$ and $slope_2 = 1$

(iii) T_3 configured in PRR_3 alone @ 60 MHz: $slope_1 = 3.5$ and $slope_2 = 1$

(iv) T_3 configured in PRR_4 alone @ 120 MHz: $slope_1 = 4$ and $slope_2 = 1$

As can be observed, the slopes of die temperature rise are the same whether only one task is configured in a PRR or there are multiple tasks running on different PRRs at different frequencies. Thus, the temperature rise slopes for a die remain the same irrespective of the number of tasks (i.e., resource utilization), frequency of operation, and their PRR locations (in our case, $slope_1 = 3.5–4$°C/min and $slope_2 = 1$°C/min for the Zynq device used for the experiments).

From the above observation, if we know the base temperature of the die for a configuration, i.e., die temperature when a configuration is programmed after the FPGA is turned ON, we can predict the saturation temperature of the die using the slopes of temperature rise. However, noting the base temperature of system configurations as an offline process is not feasible. If we have 10 tasks, and 10 ASP circuit variants per task, the base temperature for 10^{10} configurations will need to be noted and stored in a tremendously large LUT. Even if hypothetically, the base temperatures were stored, the search time for the LUT would be too large for run-time temperature prediction. This means although the above results give a complete understanding of the thermal behavior of the die under different scenarios, the behavior cannot be directly used to predict the saturated temperature of a configuration. More experiments are therefore needed to be carried out to achieve this goal, which are discussed as follows.

6. Investigation of Die Temperature Behavior with Changing System Configurations

All the above experiments have been carried out in isolation. This means after the temperature readings for one system configuration are taken, the ZedBoard is turned off until it completely cools down. When tasks are running on the die, the temperature rise is a result of rise in both static and dynamic power consumption. Each experiment carried out runs for a maximum of 20 minutes. After this experiment, if the FPGA is turned OFF and turned ON, the static power consumption (SPC) noted is higher than the usual static power consumption obtained when the FPGA is completely cooled down. This is because the static temperature of the die has increased due to the tasks that have been running on it. As a result, to get accurate observations for the thermal behavior of the die in the above set of experiments, the FPGA is cooled down completely before starting a new experiment. This means the static power consumption of the FPGA at the beginning of every experiment has been the same, i.e., we

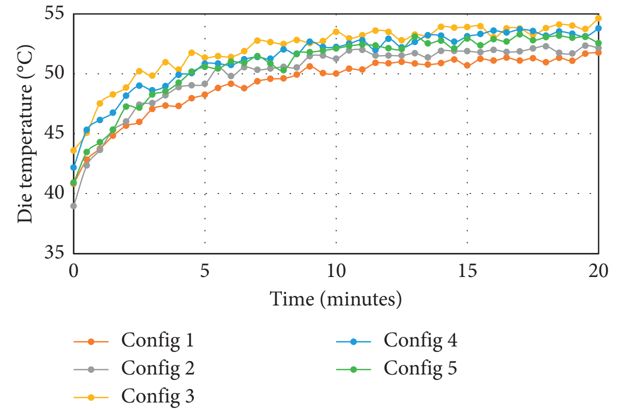

Figure 3: Die temperature vs. time for different system configurations.

have the same starting point. This is required so that all the results obtained can be compared directly without any scaling or modification.

In field-deployed systems, tasks are going to be continuously running on the die. If a new system configuration needs to be programmed, the FPGA is not going to be turned off before programming the configuration. As a result, it is necessary to observe the thermal behavior when different system configurations are programmed one after the other, without allowing any cool-down period. The experiments for this observation are performed as follows: configuration 5 from the above set of experiments is programmed. The temperature rise is observed every 30 seconds for a span of 20 minutes. After 20 minutes, configuration 4 is programmed, and temperature rise is observed for 20 minutes. After that, configuration 2 is programmed and the process repeats, followed by configuration 3. A graph of temperature vs. time is plotted for the entire span of 80 minutes, as shown in Figure 4.

It is observed that, in every 20-minute period corresponding to a configuration, the die temperature settles in the same saturation temperature range that has been observed for that configuration in the set of experiments discussed in Section 5. As can be seen in Figure 4, in the first 20 minutes, the temperature saturates to 52–52.5°C, which can also be observed in Figure 3 for configuration 5. Similarly, from 20 to 40 minutes, the temperature is observed to be saturated at ≈54°C, which is the saturation temperature observed for configuration 4 in Figure 3. From 40 to 60 minutes, the temperature is saturated at ≈53.5°C, which is the saturation temperature observed for configuration 2 in Figure 3. From 60 to 80 minutes, the temperature saturates at around 54°C, which is the saturation temperature observed for configuration 3 in Figure 3.

The above observation once again confirms the rapid heat distribution in the die. As soon as the configuration changes, the corresponding change in die temperature is immediately observed. The heat distribution is fast enough such that the die temperature settles at the saturation temperature of a configuration as soon as it is programmed. From the above obtained results, it is also clear that any configuration that is programmed after a set of tasks has

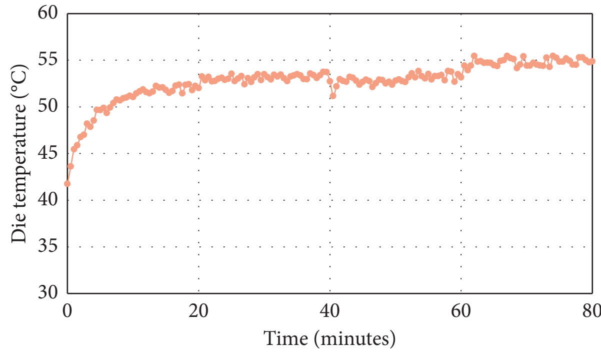

Figure 4: Die temperature vs. time for four consecutive system configurations.

already run for more than 6-7 minutes after starting up and the temperature directly settles at the saturation temperature for that configuration. From this, it can be implied that the die temperature depends on a specific parameter. Every configuration apparently varies in that specific parameter based on which the saturation temperature for that configuration is observed. This parameter needs to be identified.

From all the experiments conducted, it is clear that the die temperature can depend on the following parameters: number of tasks running on the die, the resource utilization of the tasks, the type of resources utilized, location of the tasks, and their operating frequencies. Experimental results in [5] show that, for a given configuration, the die temperature does not depend on the location of the tasks. Kirischian et al. [46] present results which show that die temperature behavior does not depend on the type of resources being utilized, i.e., Logic, BRAM, or DSP slices. We are left with three parameters: number of tasks, their resource utilization, and operating frequencies of the tasks. Every system configuration discussed in the above set of experiments has a different combination of number tasks, their total resource utilization, and their operating frequencies. One specific parameter that can encompass all these parameters is total power consumption (TPC). Power consumption of the die depends on the operating frequencies of tasks, resource utilization of the tasks, and the number of tasks [1, 4]. A relation between saturated die temperature for a given system configuration and corresponding total power consumption must therefore be found. This relation can represent the model to predict the saturated die temperature for any potential system configuration to be deployed on the SoPC die.

7. Analyzing Total Power Consumption Behavior

It is understood from the above section that if we can find a relation between die temperature and total power consumption, which in turn depends on resource utilization and frequency of the executing tasks, all the parameters that can influence the die temperature are taken into consideration. In order to observe the relation between die temperature and TPC, the behavior of TPC must be observed. Multiple experiments have been conducted for this purpose. TPC behavior has been observed for the same period that the die temperature behavior has been observed in the above set of experiments. Initially, TPC behavior is observed for the case when only one task is configured in a PRR. For this observation, the following experiments are carried out.

Task T_1 is configured in PRR_1 at 30 MHz. The other PRRs are configured with greyboxes running at 30 MHz. TPC of the die is noted every 30 seconds for a span of 20 minutes. T_1 is then replaced by a greybox, and the cool-down TPC when no task is running is observed every 30 seconds for a span of 10 minutes. Next, the frequency of T_1 is increased to 60 MHz, and the same 30-minute experiment is repeated. The experiment repeats for a total of six times, where the frequency of T_1 is increased up to 180 MHz. T_1 is then relocated to PRR_2, with greyboxes in other PRRs. Once again, a set of six experiments is conducted, where frequency of T_1 is increased from 30 to 180 MHz, and TPC is read for a span of 20 minutes followed by a cool-down period of 10 minutes. The experiments repeat for T_1 relocated at PRR_3 and PRR_4. Using the TPC results, graphs for TPC vs. time are plotted.

Figure 5 shows a graph of TPC vs. time when task T_1 is configured in PRR_1. The six curves correspond to the TPC when T_1 runs at a frequency of 30 MHz to 180 MHz in steps of 30 MHz. Observing the TPC behavior in 0–20 minutes, TPC rises for around 5 minutes and then saturates at a value, just like how the die temperature behaves. In the cool-down phase (20–30 minutes), the TPC immediately falls to a stable value and stays there for the period observed; again, just like the temperature. Graphs obtained for TPC vs. time when T_1 is configured in PRR_2, PRR_3, and PRR_4 are very similar to Figure 5. This is expected because the difference between each graph is only the location of the task. Since the task is the same, its power consumption in any location should be the same.

The behavior of TPC is also observed for a multitask scenario. As mentioned in the experiments discussed in Section 5, TPC of the die is observed for every configuration along with the die temperature. Figure 6 shows the graph for TPC vs. time for all the 5 configurations discussed in that set of experiments. The curves have the same behavior as can be seen in Figure 5; the TPC rises for a few minutes and then settles at a saturated value for the remaining period of observation.

From these results, it is possible to derive a relation between saturated TPC (STPC) and saturated die temperature (ST), which represents the saturated temperature estimation model (STEM).

From the thermal behavior and TPC behavior noted in all the experiments, a simple methodology to derive the STEM is developed. The derivation steps are presented in the next section.

8. Generalized Steps for STEM Derivation for Any SoPC Device

From Section 7, we know that the STEM will be an equation that predicts the saturated die temperature in terms of STPC.

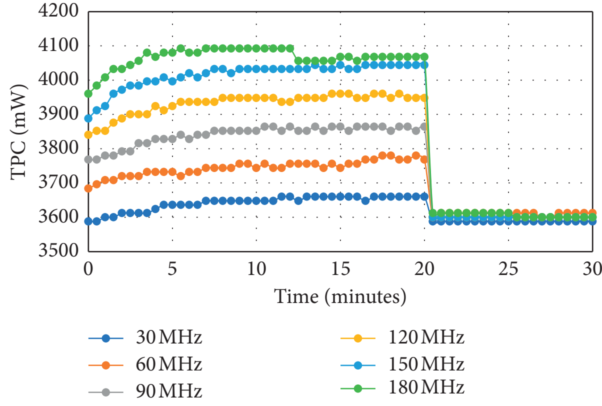

FIGURE 5: TPC vs. time plot when T_1 is configured in PRR_1.

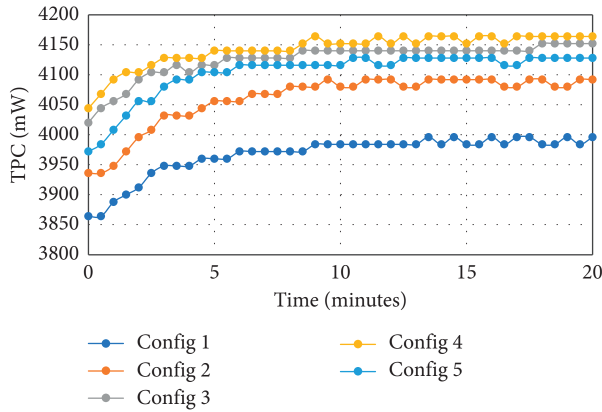

FIGURE 6: TPC vs. time plot for different system configurations.

In a run-time scenario, the smallest time to obtain STPC for a system configuration is run-time prediction. In a system with large number of tasks and their ASP circuit variants, it is not practical to store STPC values for every possible configuration in a LUT. This means once STPC is estimated for a system configuration, it can be used to predict the saturated die temperature for the same configuration.

The authors in [1, 47] present a method to derive the Dynamic Power Consumption Estimation Model (DPCEM) that can predict the DPC for a potential system configuration to be deployed on the SoPC die, at run-time. The DPCEM predicts the DPC that a system configuration would have at around 3 minutes of time post its deployment after system power-up. Referring to equation (7) in [1], the Estimated DPC (EDPC) of a FPGA die is obtained as follows:

$$\text{EDPC}_{(\text{FPGA})}\ (\text{mW}) = \frac{F_{\text{cc}}}{F_{\min}} \times (C_{\text{LS}} \times N_{\text{LS}} + C_{\text{BS}} \times N_{\text{BS}} + C_{\text{DS}} \times N_{\text{DS}} + C_F), \quad (1)$$

where F_{cc} is the current operating frequency of the task and $F_{\min}$ is the minimum operating frequency for the task. C_{LS}, C_{BS}, and C_{DS} are the coefficients representing the slopes of rise in DPC with respect to Logic, BRAM, and DSP slices, respectively. N_{LS}, N_{BS}, and N_{DS} are the number of Logic, BRAM, and DSP slices used by the task, respectively. C_F is a constant.

For a system configuration with N_c tasks, the total resource utilization of the configuration is the sum of the resource utilization of individual tasks. Thus, as presented in equation (13) in [1], the above equation (1) changes to the following:

$$\text{EDPC}_{(\text{FPGA})}\ (\text{mW}) = \frac{F_{\text{cc}}}{F_{\min}} \times \left(C_{\text{LS}} \times \sum_{n=0}^{N_c-1} N_{\text{LS}n} + C_{B_S} \times \sum_{n=0}^{N_c-1} N_{\text{BS}n} + C_{\text{DS}} \times \sum_{n=0}^{N_c-1} N_{\text{DS}n} + C_F \right). \quad (2)$$

By finding a relation between DPC of a system configuration and its STPC, we can estimate STPC. Once that is achieved, the STEM can be derived. The DPCEM derivation requires the development of test designs with varying resource utilization: one reconfigurable resource type at a time [1, 47]. The same test designs need to be used for STPC derivation. Say, there are N_L designs which differ from one another in Logic slice utilization, N_B designs which differ from one another in BRAM slice utilization, and N_D designs which differ from one another in DSP slice utilization. For each design at a specific resource utilization, N_F bitstreams are generated, which vary in the frequency of operation. Thus, the total number of available test bitstreams is as follows:

$$N = (N_L + N_B + N_D) \times N_F. \quad (3)$$

The steps for STPC and STEM derivation are discussed as follows:

(1) Turn on the SoPC device under test and note the static power consumption (SPC).

(2) Configure a bitstream and note the TPC of the SoPC device under test at ≈3 minutes of configuring the bitstream. In this step, there is no temperature monitoring. This means power is not consumed by the hard- or soft-core processor which runs the code to monitor the temperature.

(3) Turn off the SoPC device and let it cool down such that when it is turned on again, it has the same SPC as noted before in Step 1.

(4) Turn on the SoPC device and configure the same bitstream. Note the TPC and the die temperature of the SoPC device under test every 30 seconds for a span of 20 minutes (the resolution of 30 seconds can be further reduced based on the monitoring method available, and the span of 20 minutes can also be reduced or increased as long as a saturated value is possible to be obtained). In this step, the hard- or soft-core processor which runs the code to monitor the temperature also consumes power. The noted TPC therefore will be higher than the one observed in Step 2.

(5) Turn off the SoPC device and let it cool down such that when it is turned on again, it has the same SPC as noted before in Step 1.

(6) Repeat steps 1 to 5 for all the N bitstreams.

Note: the number of bitstreams to be used for these tests can be less than N. This is because, as seen in [4], the die temperature does not depend on the type of resources being utilized. All we need is some designs that vary in resource utilization, so that we have data to relate temperature with power consumption. The more the number of bitstreams used for test, the more data will be available, leading to a more accurate model. Therefore, we suggest using all the N bitstreams that were used for DPCEM derivation.

Note: the coefficients obtained for a DPCEM relate to the switching activity factor of the designs used to derive the model. Since die temperature depends on power consumption, it is advisable to use the same test designs to derive the STEM as well. This way, the STEM and DPCEM will be well in sync. Other test designs can also be used to derive the STEM, as long as their switching activity factors are similar to the ones used for DPCEM derivation. If they have different switching activity factors, an accurate temperature prediction may not be achieved.

(7) Plot the TPC vs. time (0–20 minutes) graphs from the readings noted in Step 4 for all the N bitstreams.

(8) Using the TPC vs. time graphs, note the TPC observed at 3 minutes and STPC (from the 10- to20-minute section) for all N bitstreams. Tabulate the N pairs of TPC (at 3 minutes) and STPC, and find the difference between the two as shown in Table 1.

(9) Average the N readings in the $(\mathrm{STPC} - \mathrm{TPC}_{3\mathrm{min}})$ column of Table 1. This gives a value R, which is the rise in TPC to reach STPC.

(10) Form another table, with one column as the TPC at 3 minutes noted in Step 2 (i.e., without the processor running) and the TPC at 3 minutes noted in Step 4 (i.e., with the processor running). Find the difference between the two for all the N designs as shown in Table 2.

(11) Average the N readings in the $(\mathrm{TPC}_{\mathrm{Step4}} - \mathrm{TPC}_{\mathrm{Step2}})$ column of Table 2. This gives SPP; the static power consumed by the hard- or soft-core processor monitoring the temperature.

(12) From the above obtained values of SPC and SPP, the model equation to estimate TPC (ETPC) becomes

$$\mathrm{ETPC\ (W)} = \mathrm{SPC\ (W)} + \mathrm{SPP\ (W)} + \mathrm{EDPC\ (W)}. \quad (4)$$

SPC and SPP are one-time offline measurements, and the EDPC is obtained using the DPCEM, i.e., equation (2).

Table 1: Relation between STPC and TPC observed at 3 minutes.

Design no.	$\mathrm{TPC}_{3\mathrm{min}}$ (mW)	STPC (mW)	$\mathrm{STPC} - \mathrm{TPC}_{3\mathrm{min}}$ (mW)
Design_1	TPC_1	STPC_1	$\mathrm{Difference}_1$
Design_2	TPC_2	STPC_2	$\mathrm{Difference}_2$
...			
Design_N	TPC_N	STPC_N	$\mathrm{Difference}_N$

Table 2: Calculation of the average value of SPP.

Design no.	TPCstep2	TPCstep4	TPCstep4 – TPCstep2 (mW)
Design_1	$\mathrm{TPCstep2}_1$	$\mathrm{TPCstep4}_1$	$\mathrm{Difference}_1$
Design_2	$\mathrm{TPCstep2}_2$	$\mathrm{TPCstep4}_2$	$\mathrm{Difference}_2$
...			
Design_N	$\mathrm{TPCstep2}_N$	$\mathrm{TPCstep4}_N$	$\mathrm{Difference}_N$

(13) From Step 9, we know the TPC rises by a value R, to reach STPC. Therefore, the model equation to estimate STPC (ESTPC) becomes

$$\mathrm{ESTPC\ (W)} = \mathrm{ETPC\ (W)} + R\ \mathrm{(W)}. \quad (5)$$

(14) Now that we have a model equation for STPC, we can derive the STEM. Plot the temperature vs. time (0–20 minutes) graphs from the readings noted in Step 4 for all the N bitstreams. Observing the section of 10–20 minutes from the temperature vs. time plots and TPC vs. time graphs obtained in Step 7, note the saturated temperature (ST) and the corresponding STPC for all N designs. This way, N ST and STPC pairs are obtained.

(15) Arrange the above N pairs in ascending order and plot a graph of ST vs. STPC.

(16) From the curve obtained, establish a linear relation between ST and STPC. A simple MS Excel plot is enough to obtain the relation. It will be of the following form:

$$\mathrm{ST\ (^\circ C)} = \mathrm{M} \times \mathrm{STPC\,(W)} + \mathrm{C}, \quad (6)$$

where M is the slope which relates the ST and STPC and C is a constant.

(17) For run-time prediction, the STEM equation can be written as follows:

$$\mathrm{EST\ (^\circ C)} = \mathrm{M} \times \mathrm{ESTPC\,(W)} + \mathrm{C}. \quad (7)$$

(18) Using equation (5), equation (7) can thus be rewritten to obtain the model equation to predict saturated temperature as follows:

$$\mathrm{EST\ (^\circ C)} = \mathrm{M} \times (\mathrm{ETPC\,(E)} + \mathrm{R})\,\mathrm{(W)} + \mathrm{C}, \quad (8)$$

where ETPC can be obtained from equation (4) and EDPC can be obtained from equation (2).

Thus, we have a simple method to derive a model that can predict the saturated temperature of any SoPC device for potential system configurations that could be running on the SoPC die. The STEM equation derived using the discussed method incorporates all the possible on-chip parameters that can influence the die temperature. As can be seen, the STEM equation obtained is a linear equation. It can clearly be used for run-time die temperature prediction. Also, since the model is a linear equation, only two die temperature measurements are essentially required to obtain the model coefficients. The above process can therefore be carried out in run-time by systems to self-derive the models and update the coefficients if there are any changes in the SoPC platform or the application being supported (details on the concept of self-derivation of STEM are not in the scope of this paper). The decision-making mechanism can dynamically use the derived model as and when needed to carry out RTSA, i.e., estimate the die temperature for potential system configurations and select the most suitable configuration that can satisfy the existing die temperature constraint.

9. Verification of Methodology for STEM Derivation on Zynq Device

The procedure for deriving the STEM is demonstrated and validated on the Zynq device, housed on the ZedBoard. This device is chosen since it has been used as the SoPC under test for all the above experiments to observe the thermal behavior of the die. Also, a DPCEM has been derived for the device in [1, 47].

A test design, consisting of all reconfigurable resources, namely, Logic, BRAM, and DSP slices is developed. The design is essentially arithmetic functions such that all the reconfigurable resources are utilized. Five variants of the design with increasing resource utilization are generated. For each variant, 6 bitstreams are generated; each one operating at a different frequency from 30 to 180 MHz. Thus, a total of 30 bitstreams are generated. When the ZedBoard is turned on, the noted SPC = 2340 mW (Step 1 of Section 8). For each bitstream, the TPC of the Zynq device alone (without the ARM processor running) is noted at ≈3 minutes of configuring the bitstream (Step 2 of Section 8). Also, for each bitstream, TPC and die temperature (i.e., with ARM processor running) are noted every 30 seconds, for a span of 20 minutes (Step 4 of Section 8). According to Step 7 of Section 8, TPC vs. time plots are obtained for all the 30 bitstreams. The TPC at 3 minutes observation time is noted and tabulated as shown in Table 3. Observing the section from 10 to 20 minutes, STPC for all the 30 designs are tabulated as shown in Table 3. It can be observed from Table 3 that the difference between STPC and TPC observed at 3 minutes is between 60 and 84 mW, with a greater number of readings at 60 mW. This means, on average, the value of R, i.e., the rise in TPC to reach STPC, is 60 mW (Step 9 of Section 8). Next, for all the 30 bitstreams, the average value of the difference, when both TPC and die temperature are noted and when TPC is monitored alone, is observed to be 1176 mW. This means the static power consumed by the ARM core (SPP) is 1176 mW (Step 10 of Section 8). From the above obtained values of SPC, SPP, and R, the model equation to estimate TPC (ETPC for the Zynq device according to Step 12 of Section 8 becomes

$$\begin{aligned} \mathrm{ETPC}_{\mathrm{Zynq}}(\mathrm{W}) &= 2.340(\mathrm{W}) + 1.176(\mathrm{W}) + \mathrm{EDPC}(\mathrm{W}) \\ &= \mathrm{EDPC}(\mathrm{W}) + 3.516(\mathrm{W}). \end{aligned} \tag{9}$$

Based on equation (10) in [1], EDPC for the Zynq device is given as

$$\begin{aligned} \mathrm{EDPC}_{(\mathrm{Zynq})}(\mathrm{mW}) = \frac{F_{\mathrm{cc}}}{F_{\mathrm{min}}} \times \big(0.013 \times N_{\mathrm{LS}} + 1.1 \times N_{\mathrm{BS}} \\ + 0.266 \times N_{\mathrm{DS}} + 23.06\big). \end{aligned} \tag{10}$$

Using equations (2) and (10), the EDPC for a system configuration with N_c tasks running on the Zynq die can be obtained as follows:

$$\begin{aligned} \mathrm{EDPC}_{(\mathrm{Zynq})}(\mathrm{mW}) = \frac{F_{\mathrm{cc}}}{F_{\mathrm{min}}} \times \left(0.013 \times \sum_{n=0}^{N_c-1} N_{\mathrm{LS}n} + 1.1 \right. \\ \left. \times \sum_{n=0}^{N_c-1} N_{\mathrm{BS}n} + 0.226 \times \sum_{n=0}^{N_c-1} N_{\mathrm{DS}n} + 23.06 \right). \end{aligned} \tag{11}$$

Once ETPC of Zynq is obtained using equations (9)–(11), from Step 13 of Section 8, the model equation to estimate STPC (ESTPC) can be obtained using the value of R:

$$\mathrm{ESTPC}_{(\mathrm{Zynq})}(\mathrm{W}) = \mathrm{ETPC}(\mathrm{W}) + 0.060(\mathrm{W}). \tag{12}$$

Using the die temperature readings obtained for the test bitstreams, temperature vs. time graphs are plotted. Observing the section of 10–20 minutes from the temperature vs. time plots, saturated temperature (ST) is noted for all the designs. Arranging the ST and corresponding STPC values in ascending order, a graph of ST vs. STPC is plotted, as shown in Figure 7. The curve obtained can be approximated by a linear equation seen in the graph. Thus, we have

$$\mathrm{ST}_{(\mathrm{Zynq})}(^{\circ}\mathrm{C}) = 11.85 \times \mathrm{SPTC}(\mathrm{W}) + 4.89. \tag{13}$$

Equation (13) represents the model to predict the saturated temperature based on STPC for the Zynq device. It can therefore be rewritten using equations (9) and (12) as

$$\begin{aligned} \mathrm{EST}_{(\mathrm{Zynq})}(^{\circ}\mathrm{C}) &= 11.85 \times \mathrm{ESPTC}(\mathrm{W}) + 4.89 \\ &= 11.85 \times (\mathrm{ETPC}(\mathrm{W}) + 0.060) + 4.89 \\ &= 11.85 \times (\mathrm{EDPC}(\mathrm{W}) + 3.516 + 0.060) + 4.89 \\ &= 11.85 \times (\mathrm{EDPC}(\mathrm{W}) + 3.576) + 4.89. \end{aligned} \tag{14}$$

Comparing the saturated temperature values estimated using (14) and the measured die temperature values for the 30 bitstreams, the model has an accuracy of ±2°C. This is acceptable considering the unavoidable experimental errors

Table 3: Relation between STPC and TPC observed at 3 minutes for Zynq.

	Frequency	TPC_{3min} (mW)	STPC (mW)	$STPC - TPC_{3min}$ (mW)
Design 1	30	3672	3732	60
	60	3780	3840	60
	90	3912	3972	60
	120	4020	4080	60
	150	4116	4176	60
	180	4212	4272	60
Design 2	30	3696	3756	60
	60	3876	3936	60
	90	3984	4056	72
	120	4128	4188	60
	150	4260	4320	60
	180	4356	4428	72
Design 3	30	3720	3780	60
	60	3936	3996	60
	90	4080	4140	60
	120	4236	4296	60
	150	4368	4440	72
	180	4524	4584	60
Design 4	30	3732	3816	84
	60	3972	4056	84
	90	4152	4212	60
	120	4344	4404	60
	150	4500	4572	72
	180	4668	4752	84
Design 5	30	3780	3852	72
	60	3996	4056	60
	90	4224	4284	60
	120	4428	4512	84
	150	4656	4728	72
	180	4872	4956	84

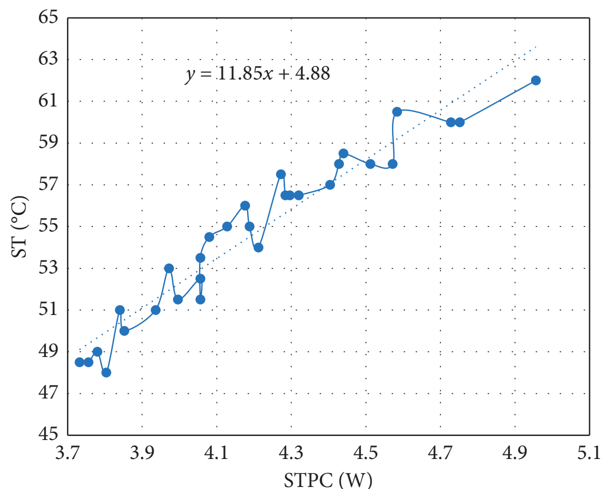

Figure 7: Plot for ST vs. STPC for 30 different bitstreams.

involved while measuring TPC and temperature. Also, in actual systems, the range of temperature predicted by the model will be of more value than the exact temperature. For example, if the specifications require the die temperature to be in the range of 50–52°C, and the STEM estimates 54–56°C for a candidate system configuration, the configuration can easily be rejected. The model has been validated for designs with STPC up to 6 W.

10. Run-Time STEM Application Example

This section presents an example of how the STEM can be used at run-time to carryout RTSA such that the die temperature is maintained in the desired range without affecting the performance of the active system tasks. Since a STEM has been derived for the Zynq XC7Z020 device in Section 9, the same SoPC has been used for the example presented.

Consider a system developed on the Zynq device. The system is deployed with the MACROS framework and is divided into 8 slots to be capable of RTSA, as shown in Figure 8. The system has two tasks T_0 and T_1. Task T_0 performs video acquisition and processing of video frames according to the 720p standard. It has a frame rate of 120 fps. T_1 is a communication and video-transmission task with a performance of 16 Mbps. Each task has 4 ASP circuit variants which have different combinations of operating frequency and resource utilization to provide the same task performance. The two tasks can operate at 30 MHz, 60 MHz, 120 MHz, and 240 MHz. The operating frequency F_{sys}, resource utilization in terms of the reconfigurable Logic, BRAM, and DSP slices used, and the performance of each variant are presented in Table 4.

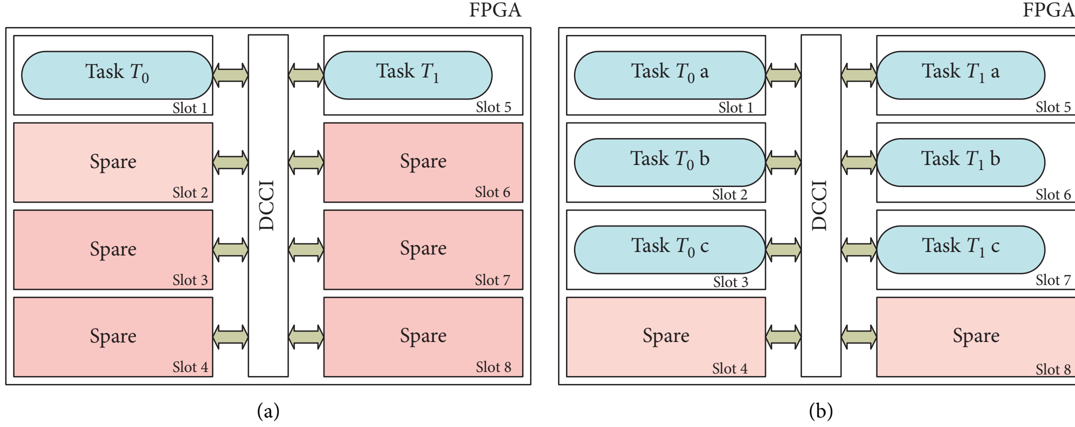

FIGURE 8: Scenarios of run-time adaptation to die temperature changes. (a) $F_{sys} = 240$ MHz and (b) $F_{sys} = 60$ MHz.

TABLE 4: ASP circuit variants of two systems tasks.

Variant no.	No. of slots	F_{sys} (MHz)	Performance	Logic slices	BRAM slices	DSP slices
$T_0 - 0$	1	240	120 fps	3093	43	30
$T_0 - 1$	2	120	120 fps	5897	79	54
$T_0 - 2$	4	60	120 fps	11779	145	97
$T_0 - 3$	8	30	120 fps	23259	254	176
$T_1 - 0$	1	240	16 Mbps	2061	22	82
$T_1 - 1$	2	120	16 Mbps	3893	40	157
$T_1 - 2$	4	60	16 Mbps	7514	58	245
$T_1 - 3$	8	30	16 Mbps	14499	94	447

10.1. Initial State. Assume that, initially, the permitted die temperature range (PDTR) is 50–52°C. As shown in Figure 8(a), the system selects and configures ASP circuit variants $T_0 - 0$ and $T_1 - 0$, which operate at 240 MHz and occupy 1 slot on the FPGA each, as the system configuration that satisfies the PDTR. How the system decides this configuration is not in the scope of this paper; however, the following demonstrates how the selected configuration satisfies the PDTR.

Using equation (11) and Table 4, the DPC of the two tasks running on the Zynq device can be obtained as follows:

$$\text{EDPC}_{(\text{Zynq})}\ (\text{mW}) = \frac{240}{30} \times \left(0.013 \times \sum_{n=0}^{2} N_{\text{LS}n} + 1.1 \times \sum_{n=0}^{2} N_{\text{BS}n} + 0.226 \times \sum_{n=0}^{2} N_{\text{DS}n} + 23.06 \right) = 1494.992\ \text{mW}. \quad (15)$$

From Section 9, the SPC of the Zynq is 2340 mW. In this example, while the system is running, die temperature is not being measured. As a result, the ARM core is not being used; only the FPGA resources are used. Therefore, SPP is not considered in the calculations here. Also, the value of R, the rise in TPC to reach STPC, is observed as 60 mW in Section 9. Using equations (4) and (5), the STPC is estimated as follows:

$$\text{ESTPC}_{(\text{Zynq})}\ (\text{W}) = 2.340\ (\text{W}) + 1.495\ (\text{W}) + 0.060\ (\text{W}) = 3.895\ (\text{W}). \quad (16)$$

From equation (13), the saturated die temperature of Zynq can be estimated as

$$\text{EST}_{(\text{Zynq})}\ (°\text{C}) = 11.85 \times 3.895 + 4.89 = 51°\text{C}. \quad (17)$$

The predicted die temperature falls within the permitted range of 50–52°C. Thus, the selected configuration satisfies the temperature constraint.

10.2. Need to Decrease Die Temperature. Suppose that the external-to-chip temperature changes such that the die temperature needs to be reduced and brought in the range of 46–48°C so that the difference between the external and on-chip temperature is maintained as before to avoid thermal cycling. The above system configuration does not satisfy the new temperature constraint. The system needs to adapt in run-time. It uses the STEM to evaluate potential configurations and dynamically selects and configures ASP variants $T_0 - 2$ and $T_1 - 2$, as shown in Figure 8(b), as the new system configuration. These ASP circuit variants operate at 60 MHz and occupy 3 slots on the FPGA each. Following the same process as in Section 10.1, the STPC of this configuration is estimated to be 3.545 W. Using this ESTPC and equation (13), the die temperature is estimated to be

$$\mathrm{EST}_{(\mathrm{Zynq})}\,(^{\circ}\mathrm{C}) = 11.85 \times 3.545 + 4.89 = 47^{\circ}\mathrm{C}. \tag{18}$$

The predicted die temperature falls within the permitted range of 46–48°C. Thus, by using the run-time STEM and RTSA, ASP circuit variants of tasks that operate at lower frequency and occupy more resources are selected and configured dynamically. The result is that the die temperature is maintained in the desired range and the performance of the tasks is also not affected.

11. Conclusion

The presented research is aimed to find the methodology to derive the thermal dynamic model for partially reconfigurable FPGA-based SoPCs supporting multitask stream-processing applications. The main goal of this research is to allow embedded decision-making systems to dynamically select the most efficient set of task-specific ASP circuits to keep the on-chip temperature in the required saturated level, thus avoiding internal thermal cycling on the flip-chip FPGA package. This is necessary due to the significant influence of on-chip thermal cycling on the reliability of the systems. This new problem can mainly be caused due to the dynamic nature of workload along with their changing operating frequencies and may bring significant thermal instability if not properly managed. Therefore, there is a need to understand the thermal behavior of SoPC dies under multitask workload conditions. From the observed behavior, a methodology for deriving a model to predict the die temperature for a system configuration in any given partially reconfigurable SoPC device needs to be developed. The model will allow the development of a run-time decision-making mechanism, which will evaluate potential system configurations using this model and select a suitable configuration that maintains the required die temperature. This mechanism, when integrated with the run-time decision-making mechanism presented in [1], will provide a complete run-time multiobjective mechanism that can enable run-time adaption to workload, power budget, temperature, and hardware resource constraints. To summarize, to achieve the goal of thermal stability on the die, the first step is to study the thermal behavior of the die and develop a method to derive a model that can predict the die temperature for a given system configuration. This paper presents our research efforts for the same. The main contributions of the paper are as follows: (a) a detailed study of thermal behavior of SoPC die in different workload scenarios, (b) a generic methodology to derive a run-time model that can predict the die temperature for a system configuration running on any FPGA-based SoPC device, and (c) an RTSA-enabled experimental setup on Xilinx ZYNQ XCZ7020 SoPC to study the thermal behavior of the SoPC die and to develop the methodology for deriving the saturated temperature estimation model.

Data Availability

All the necessary data are included in the article.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] D. Sharma, L. Kirischian, and V. Kirischian, "Run-time mitigation of power budget variations and hardware faults by structural adaptation of FPGA-based multi-modal SoPC," *Computers*, vol. 7, no. 4, p. 52, 2018.

[2] V. Dumitriu, L. Kirischian, and V. Kirischian, "Mitigation of variations in environmental conditions by SoPC architecture adaptation," in *Proceedings of the 2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 1–8, Montreal, Canada, June 2015.

[3] Avnet, *Vivado Design Suite User Guide Partial Reconfiguration*, Avnet, Phoenix, AZ, USA, 2018.

[4] D. Sharma, L. Kirischian, and V. Kirischian, "Run-time adaptation method for mitigation of hardware faults and power budget variations in space-borne FPGA-based systems," in *Proceedings of the 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 32–39, Pasadena, CA, USA, July 2017.

[5] D. Sharma, V. Kirischian, and L. Kirischian, "On-chip thermal balancing using dynamic structural adaptation of FPGA-based multi-task SoPCs for space-borne applications," in *Proceedings of the 2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 79–86, Colchester, UK, July 2019.

[6] P. R. Chafi, M. Moradi, N. Rahmanikia, and H. Noori, "A platform for dynamic thermal management of FPGA-based soft-core processors via dynamic frequency scaling," in *Proceedings of the 2015 23rd Iranian Conference on Electrical Engineering*, pp. 1093–1097, Tehran, Iran, May 2015.

[7] S. D. Carlo, G. Gambardella, P. Prinetto, D. Rolfo, and P. Trotta, "SATTA: a self-adaptive temperature-based TDF awareness methodology for dynamically reconfigurable FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 8, no. 1, pp. 1–22, 2015.

[8] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 2, pp. 1–25, 2016.

[9] I. Christoforakis, O. Tomoutzoglou, D. Bakoyiannis, and G. Kornaros, "Dithering-based power and thermal management on FPGA-based multi-core embedded systems," in *Proceedings of the 2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*, pp. 173–177, Porto, Portugal, October 2015.

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

[11] S. Sharifi, A. K. Coskun, and T. S. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference ASPDAC '10*, pp. 873–878, IEEE Press, Taipei, Taiwan, January 2010.

[12] G. Kornaros and D. Pnevmatikatos, "Hardware-assisted dynamic power and thermal management in multi-core SoCs," in *Proceedings of the 21st Edition of the Great Lakes Symposium on Great Lakes Symposium on VLSI GLSVLSI '11*, pp. 115–120, ACM, Lausanne, Switzerland, May 2011.

[13] T. Hashamdar and H. Noori, "Thermal management of FPGA-based embedded systems at operating system level," in *Proceedings of the 2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pp. 1–6, Tehran, Iran, October 2015.

[14] F. Hameed, M. A. A. Faruque, and J. Henkel, "Dynamic thermal management in 3D multi-core architecture through run-time adaptation," in *Proceedings of the 2011 Design, Automation Test in Europe*, pp. 1–6, Grenoble, France, March 2011.

[15] Y.-W. Wang and Y.-S. Chen, "On-line thermal-aware task management for three-dimensional dynamically partially reconfigurable systems," in *Proceedings of the 2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 111–120, Taipei, Taiwan, August 2013.

[16] K. W. Lin and Y. S. Chen, "Online thermal-aware task placement in three-dimensional field-programmable gate arrays," in *Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems RACS*, pp. 412–417, ACM, Prague, Czech Republic, October 2015.

[17] A. Amouri, J. Hepp, and M. Tahoori, "Built-in self-heating thermal testing of FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1546–1556, 2016.

[18] P. Weber, M. Zagrabski, P. Musz, K. K□pa, M. Nikodem, and B. Wojciechowski, "Configurable heat generators for FPGAs," in *Proceedings of the 20th International Workshop on Thermal Investigations of ICs and Systems*, pp. 1–4, London, UK, September 2014.

[19] M. Happe, H. Hangmann, A. Agne, and C. Plessl, "Eight ways to put your FPGA on fire—a systematic study of heat generators," in *Proceedings of the 2012 International Conference on Reconfigurable Computing and FPGAs*, pp. 1–6, Cancun, Mexico, December 2012.

[20] C. Gómez Osuna, M. Sánchez Marcos, P. Ituero, M. Lopez-Vallejo, and M. López-Vallejo, "A monitoring infrastructure for FPGA self-awareness and dynamic adaptation," in *Proceedings of the 2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, pp. 765–768, Seville, Spain, December 2012.

[21] Y. Zhang and A. Srivastava, "Adaptive and autonomous thermal tracking for high performance computing systems," in *Proceedings of the 47th Design Automation Conference DAC '10*, pp. 68–73, ACM, Anaheim, CA, USA, July 2010.

[22] H. Zhou, X. Li, C. Y. Cher, E. Kursun, H. Qian, and S. C. Yao, "An information-theoretic framework for optimal temperature sensor allocation and full-chip thermal monitoring," in *Proceedings of the 49th Annual Design Automation Conference DAC '12*, pp. 642–647, ACM, San Francisco, CA, USA, June 2012.

[23] C. Tradowsky, E. Cordero, T. Deuser, M. Hübner, and J. Becker, "Determination of on-chip temperature gradients on reconfigurable hardware," in *Proceedings of the 2012 International Conference on Reconfigurable Computing and FPGAs*, pp. 1–8, Cancun, Mexico, December 2012.

[24] M. Happe, A. Agne, and C. Plessl, "Measuring and predicting temperature distributions on FPGAs at run-time," in *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs*, pp. 55–60, Cancun, Mexico, November 2011.

[25] L. Vincent, P. Maurine, S. Lesecq, and E. Beigné, "Embedding statistical tests for on-chip dynamic voltage and temperature monitoring," in *Proceedings of the 49th Annual Design Automation Conference DAC '12*, pp. 994–999, ACM, San Francisco, CA, USA, June 2012.

[26] K. M. Zick and J. P. Hayes, "Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 5, no. 1, pp. 1–26, 2012.

[27] S. Velusamy, W. Huang., J. Lach, M. Stan, and K. Skadron, "Monitoring temperature in FPGA based SoCs," in *Proceedings of the 2005 International Conference on Computer Design*, pp. 634–637, San Jose, CA, USA, October 2005.

[28] K. M. Zick and J. P. Hayes, "On-line sensing for healthier FPGA systems," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays FPGA '10*, pp. 239–248, ACM, Monterey, CA, USA, February 2010.

[29] M. Saydé, A. Lakhssassi, M. Bougataya, O. Terkawi, and Y. Blaquière, "SoC systems thermal monitoring using embedded sensor cells unit," in *Proceedings of the 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1052–1055, Boise, ID, USA, August 2012.

[30] G. Castilhos, F. G. Moraes, and L. Ost, "A lightweight software-based runtime temperature monitoring model for multiprocessor embedded systems," in *Proceedings of the 29th Symposium on Integrated Circuits and Systems Design: Chip on the Mountains SBCCI '16*, pp. 4:1–4:6, IEEE Press, Piscataway, NJ, USA, August 2017.

[31] Xilinx Inc., *Xilinx Power Estimator User Guide*, Xilinx Inc., San Jose, CA, USA, 2018.

[32] Intel, *Intel® FPGA Power and Thermal Calculator User Guide*, Intel, Santa Clara, CA, USA, 2020.

[33] K. Skadron, M. R. Stan, W. Huang, S. Velusamy., K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pp. 2–13, San Diego, CA, USA, June 2003.

[34] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, 2004.

[35] W. Huang, K. Sankaranarayanan, K. Skadron, R. J. Ribando, and M. R. Stan, "Accurate, pre-RTL temperature-aware design using a parameterized, geometric thermal model," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1277–1288, 2008.

[36] W. Huang., M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy, "Compact thermal modeling for temperature-aware design," in *Proceedings of the 41st Annual Design Automation Conference*, pp. 878–883, San Diego, CA, USA, June 2004.

[37] R. Zhang, R. S. Mircea, and S. Kevin, "HotSpot 6.0: validation, acceleration and extension," Technical report, University of Virginia, Charlottesville, VA, USA, 2015.

[38] K. Gulati, S. P. Khatri, and P. Li, "Closed-loop modeling of power and temperature profiles of FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays FPGA '09*, p. 287, February 2009.

[39] Y. Luo, X. Wang, S. Ogrenci-Memik, G. Memik, K. Yoshii, and P. Beckman, "Minimizing thermal variation in heterogeneous HPC systems with FPGA nodes," in *Proceedings of the 2018 IEEE 36th International Conference on Computer Design (ICCD)*, pp. 537–544, Orlando, FL, USA, October 2018.

[40] Xilinx Inc., *Vivado Design Suite User Guide Dynamic Function eXchange*, Xilinx Inc., San Jose, CA, USA, 2020.

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

[42] Xilinx Inc., *Zynq-7000 SoC Data Sheet: Overview*, Xilinx Inc., San Jose, CA, USA, 2018.

[43] Avnet, *ZedBoard Hardware User's Guide*, Avnet, Phoenix, AZ, USA, 2014.

[44] Xilinx Inc., *7 Series FPGAs and Zynq-7000 SoC XADC Dual12-Bit 1 MSPS Analog-To-Digital Converter*, Xilinx Inc., San Jose, CA, USA, 2018.

[45] Xilinx Inc., *Clocking Wizard v5.3 LogiCORE IP Product Guide*, Xilinx Inc., San Jose, CA, USA, 2016.

[46] L. Kirischian, V. Kirischian, and D. Sharma, "Mitigation of thermo-cycling effects in flip-chip FPGA-based space-borne systems by cyclic on-chip task relocation," in *Proceedings of the 2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 17–24, Edinburgh, UK, August 2018.

[47] D. Sharma, V. Dumitriu, and L. Kirischian, "Architecture reconfiguration as a mechanism for sustainable performance of embedded systems in case of variations in available power," *Applied Reconfigurable Computing*, pp. 177–186, Springer International Publishing, Manhattan, NY, USA, 2017.