HARPA

Harnessing Performance Variability

Project ref.	FP7-612069
Call ref.	FP7-ICT-2013-10
Activity	ICT-10-3.4

HARPA RTE: Interfaces and Requirements

Dimitrios Rodopoulos,
Nikolaos Zompakis and
Dimitrios Soudris
ICCS, GR

Report Number:	D2.2
Version:	v4.14.b
Date:	April 3, 2014

Revisions List

Date	Version	Author(s)	Description
April 3, 2014	v4.14	Dimitrios Rodopoulos	First Draft
April 10, 2014	v4.14.b	Dimitrios Rodopoulos	Including feedback from internal Reviewer

Executive Summary

This report deals with the requirements and interfaces of the HARPA Run Time Engine (RTE) for the mitigation of performance variability. For that purpose, the largest part of this report covers the concept of system scenarios, which is the main control principle that will be used for performance variability mitigation. Initially, examples are provided for key methodology terms. Excerpts of system operation (represented by Run Time Situations – RTSs) are to be clustered into representative states of system operation (represented by Scenarios), so that the degrees of performance variability are fully covered. Throughout this report, the predominant control principle assumed is that of system scenarios. Throughout the illustration of such terms, examples are provided drawn primarily from the Bias Temperature Instability (BTI) degradation mechanism. This constitutes a representative example at the silicon device level, which has direct implications for the yield and functionality of greater digital systems. Furthermore, examples are also drawn from other domains such as cloud computing or wireless radios. There is limited reference to dynamic scenarios, which are more complicated and are addressed in future HARPA deliverables (D2.6).

Assuming that a system scenario inventory is present, the HARPA RTE must be able to detect and switch between system scenarios, while signaling the platform knob accordingly to obtain optimal platform configurations. The HARPA RTE is to interact with platform monitors, so that it can detect when a switch between scenarios should materialize. In view of this dynamism, we reflect on the requirements for an efficiently hardcoded scenario inventory. The capability of scenario refinement (or calibration) is mentioned, as a major requirement to cover time-zero performance variability issues. The testing phase of a digital system is considered as the most appropriate stage for the HARPA RTE calibration.

Finally, certain observations are made regarding the effectiveness of scenario detection and switching. The various overheads associated with these functions are analyzed and knowhow from the literature is drawn in order to minimize them. For example, assuming that a directed acyclic graph is used to represent the scenario detection scheme (starting from RTS parameters and leading to the actual scenarios) graph transformations can minimize the length of the paths that have to be followed in order to conclude to a currently applicable scenario.

Table of Contents

Table of Figures 4

1 Introduction 5

2 Theoretical Background.................... 7

2.1 Run Time Situations 7

2.2 Cost Metrics 9

2.3 System Scenarios 11

3 Introduction to System Scenarios 13

3.1 Identification 13

3.2 Characterization 14

3.3 Clustering 15

3.4 Detection 16

3.5 Switching 17

4 Interfaces & Requirements for the HARPA RTE running System Scenario Mitigation 19

4.1 Performance Variability Model 19

4.2 Pre-stored System Scenario Information 20

4.3 The Actual HARPA RTE 20

4.3.1 Monitor-Detection Interface 21

4.3.2 Switching-Knob Interface 21

5 Conclusions 23

6 Works Cited 24

Table of Figures

Figure 1: An example of an RTS for a single FET device [5] 7
Figure 2: Three RTS parameters for a single device [5], recorded over an interval of roughly three years; the frequency of the Vgs signal is assumed constant at 50 MHz. 8
Figure 3: An example of RTS parameters from the Cloud Computing domain [6] 8
Figure 4: RTS parameter of packet size communicated through a NoC buffer per unit time [7]........... 8
Figure 5: An example of an RTS parameter for the case of device-level BTI modeling [5] 9
Figure 6: An example of a Cost Metric for the case of device-level BTI modeling [5]........................... 9
Figure 7: Evolution the delay cost metric for a production cloud service [6]....................................... 10
Figure 8: Execution time for a series of WLAN packets.. 10
Figure 9: Scenario-based design flow [8] 12
Figure 10: The components of the system scenario methodology in view of the performance variability mitigation problem; components illustrated with a dashed line correspond to HARPA Work Packages other than the current (Knobs & Monitors → WP3, Platform Degradation Model → WP4) 13
Figure 11: RTS Identification & Characterization.. 15
Figure 12: Front of Pareto optimality 15
Figure 13: Application of constraints 15
Figure 14: Clustering overhead representation for the clustering of three RTSs in a single system scenario; WEC represents the Worst Estimation Case [13].. 16
Figure 15: A direct acyclic graph is used to formulate the detection sequence of occurring system scenarios [13]; certain optimizations are in order, so that the detection complexity becomes more affordable.. 17
Figure 16: Highlighting the interfaces and general requirements of the system-scenario-based HARPA RTE 19

1 Introduction

The purpose of this report is to provide insight into the interfaces and requirements that should enable mitigation of performance variability using system scenarios. The concept of *system scenarios* is already established in the literature and used in many contexts. The abstract nature of this methodology has made it very suitable in dealing with control problems exhibiting dynamic behaviour in the problem itself, or variability in the platform on which the problem is executed. Important case studies in the past have covered especially the energy efficient execution of dynamic multimedia [1] or wireless [2] applications on single and multi-core SoC platforms. More recently, system scenarios have also been utilized to mitigate time-zero variability in digital circuits [3]. However, there has never been a unified application of system scenarios on the fine-grain time-dependent variability of digital systems. In that respect, the HARPA project will apply the concept of system scenarios on the mitigation of performance variability, throughout the entire system lifetime, focusing both on time-zero and time-dependent mechanisms. We will use a fine-grain workload-dependent model developed in [4] to enable this. This technique will be applied both at the operating system (OS) and the run time levels. The current report focuses on the latter aspect, namely the HARPA Run Time Engine (RTE).

The system scenario methodology is based on the identification of different platform operation states. These may strongly vary because of the workload that is processed or because of inherent platform degradation. The idea is to group similar platform behaviour states to a single scenario, thus being able to limit the total number of different scenario classes and hence to reduce the storage overhead and to simplify the run-time platform control (e.g. with respect to power management). This grouping is achieved by bottom-up clustering or top-down partitioning behaviour-platform pairs which exhibit a small distance in the N-dimensional parameter space. And at run time we can then identify the active scenario by monitoring key platform and application variables and checking in which scenario class they reside. This scenario grouping can be performed fully at design time (system scenarios) or at least partly at run time (dynamic scenarios). This report will cover only the former case, since the latter is the subject of a later deliverable.

Given the design time analysis, in the case of system scenarios, it is possible to pick the optimal platform configuration for each scenario, which will satisfy the timing and quality cost constraints of the system. Thus, if such information is initially loaded and stored on the platform, the switching between platform configurations can take place, each time a different scenario is detected. Already, we have created a requirement for platform components enabling controllability and observability. This links directly to WP3 of the HARPA project which is entirely devoted to this purpose. In the current report, we will not reflect on specific knobs and monitors that are required to mitigate performance variability with the proposed system scenario methodology.

For the sake of simplicity, we will use a single example of performance variability in the motivational examples presented in this report. The phenomenon of our choice is Bias Temperature Instability (BTI), as described by state-of-the-art atomistic models [5]. This mechanism is responsible for the absolute increase of a Field Effect Transistor’s threshold voltage (V_{th}) under certain operating conditions. At the device level, this is apparently a strong source of performance variability, since the boundary between the cutoff and saturation regions becomes dependent of the device's gate voltage (V_{gs}). Especially the device V_{th} level is shifted by this BTI impact. At the circuit and architecture level, this phenomenon has direct implications on the functionality and intrinsic parametric behaviour of a platform, since it may cause functional breakdown and delay or leakage variation of the system under certain operating conditions.

Without any loss of generality, we will present the interfaces that are required in order to enable mitigation of BTI-induced performance variability. Assuming a different source of performance variability, the system scenario methodology specifications can be seamlessly applied. That is why, throughout this report, we will provide examples from other domains where performance variability is featured (e.g. cloud computing or networks on chip).

The current report is organized as follows: Section 2 provides some initial concepts related to system scenarios. We introduce terms such as that of Run Time Situations (RTSs), Cost Metrics and Scenarios, while providing a link towards the issue of performance variability. In Section 3, we give a brief overview of the system scenarios methodology. Section 4 casts this methodology to the domain of performance variability, exploring the requirements and interfaces that are expected. Finally conclusions are summarized in Section 5.

2 Theoretical Background

With the system scenarios concept being relatively new, it is very relevant to introduce some key terms before illustrating its projection to the performance dependability domain. In this Section, we will present three key terms, each in one of the following Subsections: Run Time Situation (RTS – Subsection 2.1), Cost Metric (Subsection 2.2) and Scenario (2.3). In each case, we will “land” the respective terms to the domain of performance variability and will present some examples.

2.1 Run Time Situations

Assuming a system with variable performance, it is very important to isolate observable windows of its operation to identify the degree of variability. These *excerpts of system operation are called Run Time Situations (RTSs)*. Traditionally, an RTS can be easily derived at design time, assuming some sort of platform simulation/emulation. In Figure 1, we can see a Run Time Situation for the V_{gs} workload of single FET device. The duration of this RTS is 10 μs.

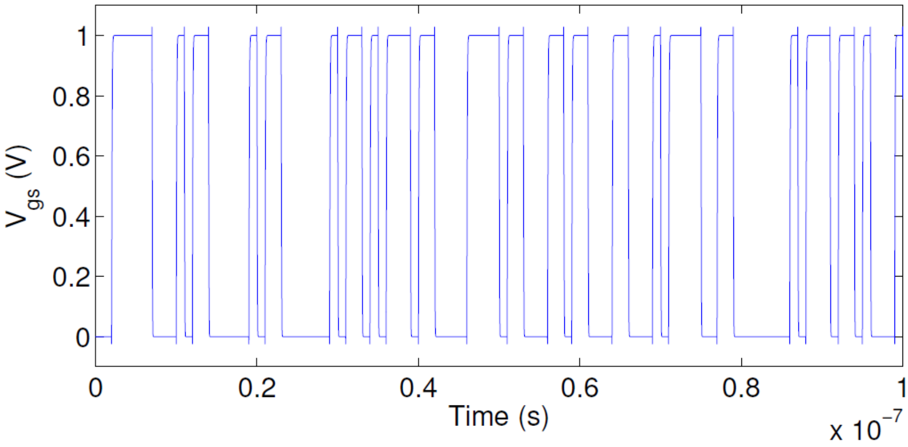

Figure 1: An example of an RTS for a single FET device [6]

Having covered the duration of an RTS, it is very important to note which system parameters are encapsulated in an RTS. *The designer is free to choose the system parameters that need to be stored in an RTS, which will be referred to as RTS Parameters*. These may come from any abstraction of the system. The typical course of action is to tailor the RTS parameters to the type (application or platform level) of system abstraction. As a result, if we remain at the abstraction level of a single device, we could assume an RTS of 3 years and record the RTS parameters displayed in Figure 2: the signal probability (or duty factor) of the device’s V_{gs} signal, the V_{dd} configuration and the operating temperature.

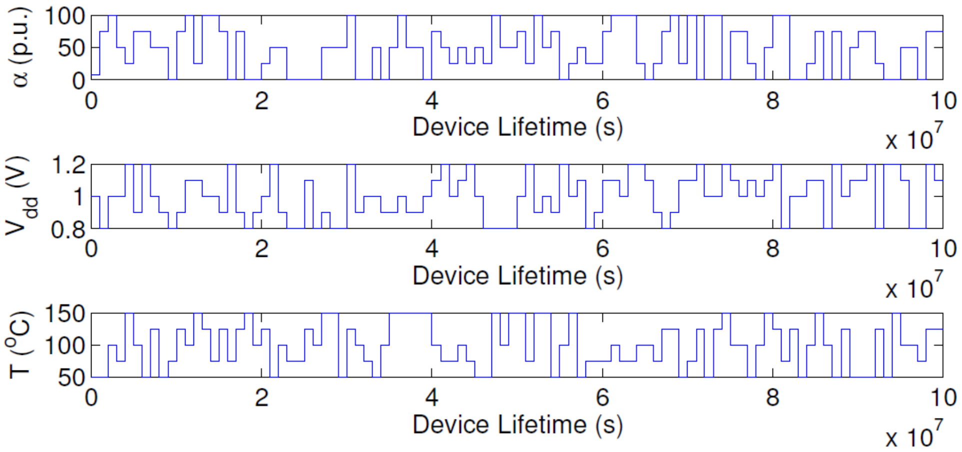

Figure 2: Three RTS parameters for a single device [6], recorded over an interval of roughly three years; the frequency of the V_{gs} signal is assumed constant at 50 MHz.

At this point, we have clarified on what constitutes an RTS. Such excerpts of system operation can be very useful for the identification of the various degrees of performance variability of a system. Furthermore, the term can be reused in a variety of contexts, on condition that the RTS parameters in question are strictly defined. In, we can see two other examples of RTS parameters from two different contexts, namely Cloud Computing and Networks on Chip (NoCs). In any case, it is straightforward that, in order to gain understanding of performance variability, *it is necessary to be able to extract RTSs for a given system, based on the observation of specific RTS parameters*.

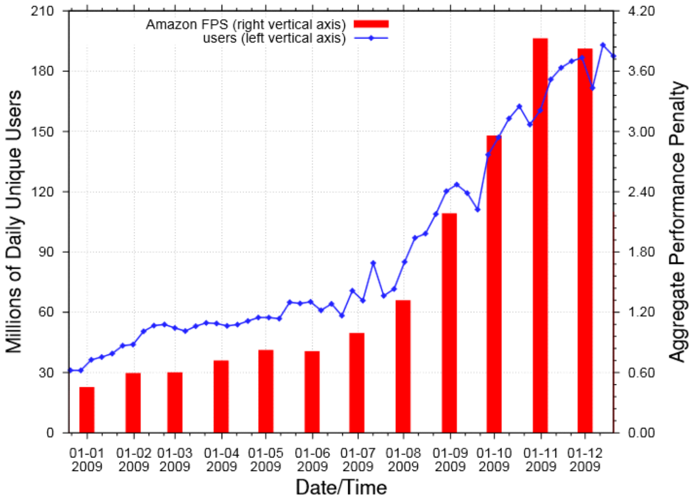

Figure 3: An example of RTS parameters from the Cloud Computing domain [7]

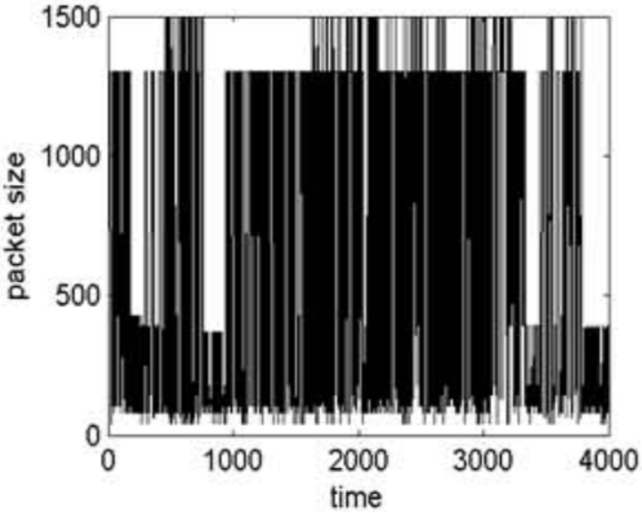

Figure 4: RTS parameter of packet size communicated through a NoC buffer per unit time [8]

2.2 Cost Metrics

In the previous Subsection, we have illustrated the concept of RTSs, as excerpts of system operation, extracted for specific RTS parameters. However, in view of performance variability, apart from RTS parameters, we need to monitor system metrics that demonstrate the target objectives, which in HARPA will be related to harnessing variability. Hence, the metrics of interest that accompany an RTS will be referred to as Cost Metrics. It is important to distinguish that an RTS is not a Cost Metric. The RTS's belong to the system parameter space which determines the system setting or state .For any of these states, we can then evaluate the metrics in the objective space. In the case of BTI, we have already presented an example of an RTS parameter, which is also shown in Figure 5. Assuming the target source of variability, we also have to record its impact, namely the V_{th} variation. In the specific case, this V_{th} variation is considered as the Cost Metric and is illustrated in Figure 6. Similarly, for the case of an embedded Static Random Access Memory (eSRAM) we can identify the functional yield of the circuit as a cost metric and its operating V_{dd} as the respective RTS. The functional yield is the percentage of eSRAM samples that satisfies the criterion of Figure 7. It is calculated throughout the lifetime of the eSRAM circuit, for different V_{dd} configurations (see Figure 8) and various degrees of confidence, based on a set of 150 samples of the target eSRAM circuit. Different cost metric evolution can be identified for different RTSs, as illustrated in Figure 11. Finally, we can observe that the spread of the yield estimators (i.e. confidence interval – CI) is increasing, when we specify a stricter degree of confidence, as illustrated in Figure 12 through.

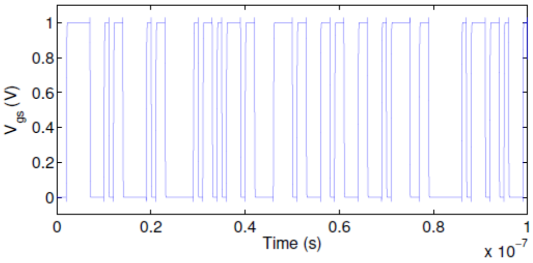

Figure 5: An example of an RTS parameter for the case of device-level BTI modeling [6]

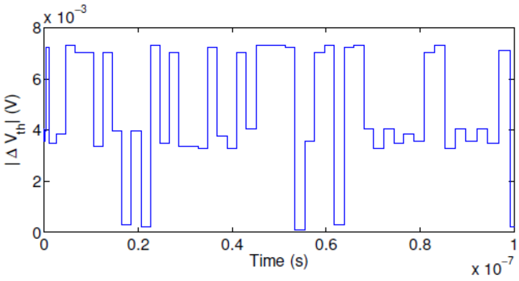

Figure 6: An example of a Cost Metric for the case of device-level BTI modeling [6]

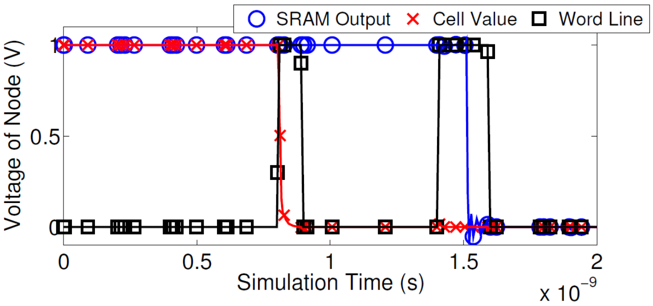

Figure 7: The functionality criterion of the target eSRAM [6]

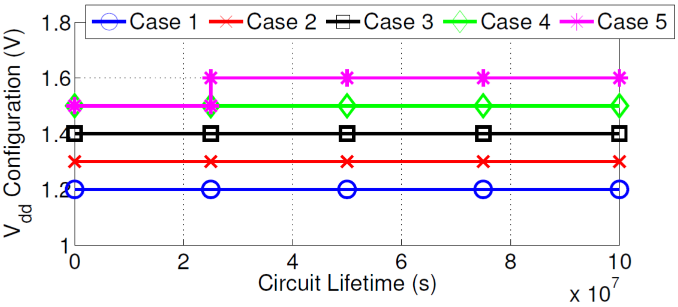

Figure 8: An example of an RTS at the circuit level for an embedded SRAM circuit [6]

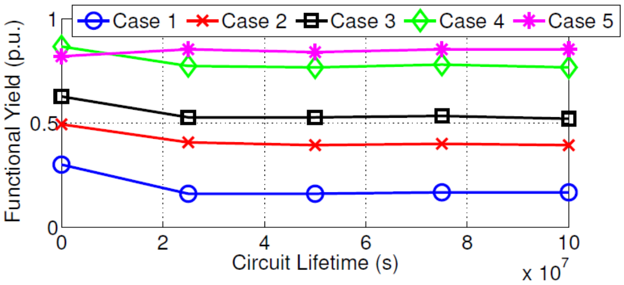

Figure 9: An example of an RTS at the circuit level for an embedded SRAM circuit [6]

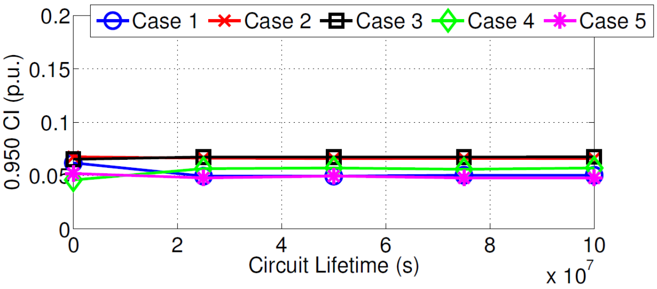

Figure 10

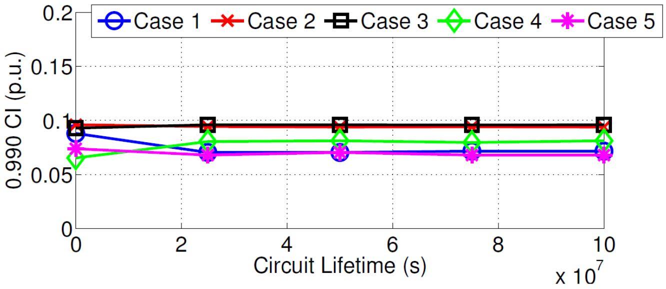

Figure 11

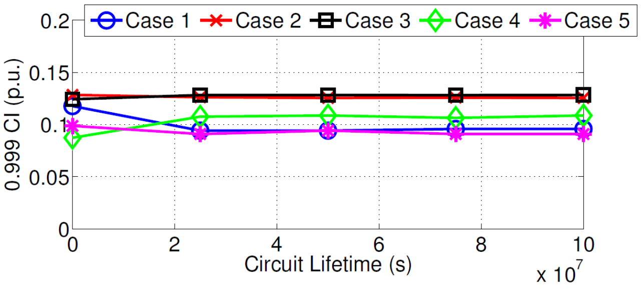

Figure 12

As a result, it becomes apparent that, in the context of performance variability, the cost metrics are the ones that contain all the variability impact information. Similarly to the above case, we can distinguish cost metrics in various domains that experience performance variability. In the cloud computing domain, the delay of a service is representative example of a cost metric (see Figure 13). Similarly, in the wireless radio domain, the execution time for a series of Wireless Local Area Network (WLAN) packets is another representative example (see Figure 14).

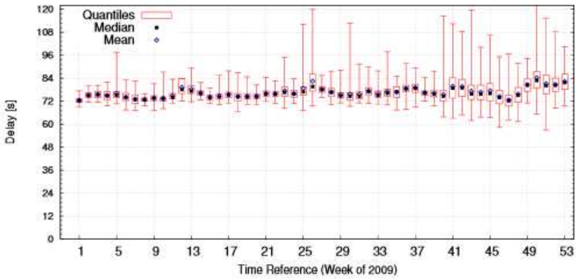

Figure 13: Evolution the delay cost metric for a production cloud service [7]

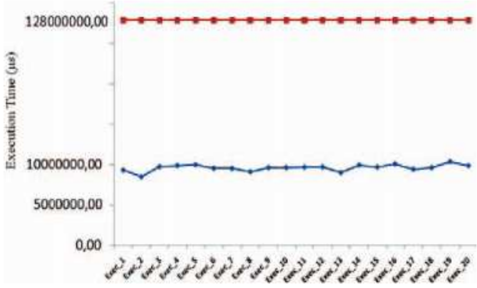

Figure 14: Execution time for a series of WLAN packets

2.3 System Scenarios

It should be clear, at this point, that a set of RTSs may not necessarily evaluate similarly for specific cost metrics. In other words, *different instances of system operation (RTSs) may display identical or similar values for certain cost metrics*. Here lies a very interesting opportunity to group RTSs that are similar or identical in terms of cost metrics to a single set. Hence, *a Scenario is a set of RTSs that exhibit similar values for the cost metrics under investigation*. The way that scenarios are derived, based on a set of RTSs, will be explained in Subsection 3.2.

Scenarios are, in general, extracted at the design phase and these can be distinguished in two main categories: (a) use case scenarios; and (b) system scenarios. Earlier we have also introduced a third class, namely dynamic scenarios which are partly determined at run-time. *Use Case Scenarios* cluster possible user actions and system reactions. The user’s actions are presented as different operation modes at system level. The use case scenarios can differ greatly in terms of performance. *System Scenarios* classify from cost perspective the several operation modes and they relate a behaviour (with modes) and an implementation (e.g. an SoC platform in our case). More precisely, system scenarios are clusters of system operations, which appear cost similarities at a multi-dimensional space. For example, a user makes use of a wide variety of network services like web browsing, VoIP connection, and video communication. Also he/she can activate specific communication links like Wi-Fi, WiMAX, Bluetooth, etc. All this activity is part of the use case scenarios. On the other hand, the signal power, the modulation scheme, the spectrum area and all technical details, which are defined internally by the system implementation on the SoC platform and which are transparent to the user, are potentially part of the system scenario domain. To summarize, every use case scenario can correspond to one or several system scenarios without excluding the possibility to be overlapped. System-scenario is the kind of scenarios that will be the focus of our study.

Figure 15 presents the generic Scenario Methodology design flow and shows the hierarchical differentiation between the use-case scenarios and system scenarios. All start with a product idea, which describes as use case scenarios the desirable operation functionality of a platform. This represents a manually high level description of the product’s features. For example in a wireless device the use-case scenarios describe the kind of the networks (e.g. cellular networks, Wi-Fi, WiMAX) and the kind of the applications (e.g. Video streaming, audio streaming, image capturing, GPS navigation) that the device will support. This description is used, as input at the next step that defines which part of the functionality will be implemented in software and which in hardware. At this step, several simulation tools are required to characterize the resulting operation modes at different design approaches. The classification of these operation modes in the multi-dimensional cost space leads to the definition of the systems scenarios. The extraction of the system scenarios respects a number of criteria the basic of which is the utilization of the system resources. For example in a wireless device, the broadcasting in two different communication modes (e.g. using different modulation schemes BPSK and QPSK in a Wi-Fi link) has close operation costs. These modes will be included at the same system scenario. The design trajectory is concluded with the coding of the system scenario to be easy detected and exploited at run time and the realization of the final product.

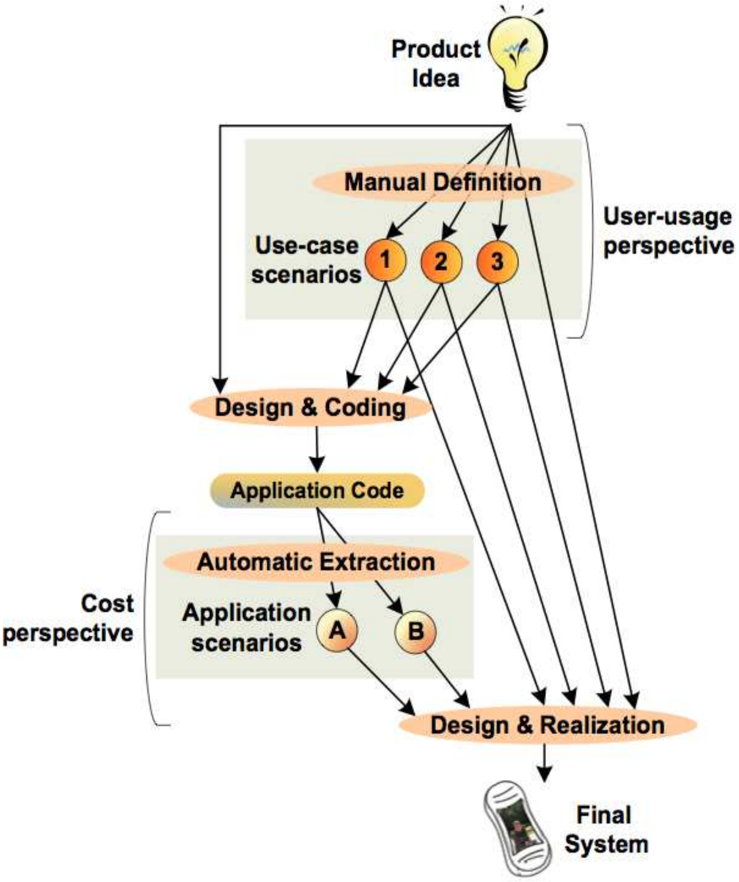

Figure 15: Scenario-based design flow [9]

3 Introduction to System Scenarios

This Section will provide the basic steps of the generic system scenario methodology. These contain the main reasoning behind the optimal/near-optimal[1] control of a digital system that exhibits certain performance variability. A summary of these steps is presented in Figure 16, already foreshadowing the requirements and interfaces of the HARPA RTE engine, which will be covered in Section 4. A good summary is also provided in [1] and [2].

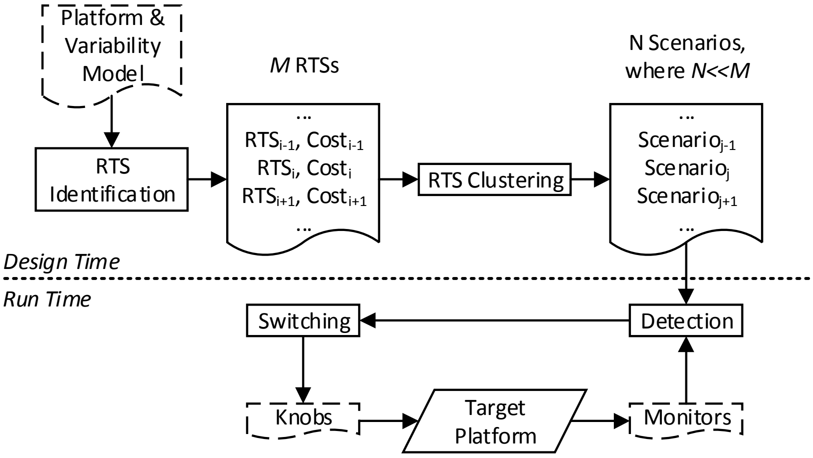

Figure 16: The components of the system scenario methodology in view of the performance variability mitigation problem; components illustrated with a dashed line correspond to HARPA Work Packages other than the current (Knobs & Monitors → WP3, Platform Degradation Model → WP4)

3.1 Identification

The first step is to identify all the possible RTSs. As we have referred the definition of the RTS exploration space requires the identification of the parameters, which differentiate the state of the system from functionality perspective. These parameters are distinguished (1) at internal and external variables based on the source of the functionality differentiation and (2) at control and data variables based on the kind of control that they apply. As internal, we assume every variable, which has a direct impact on the system, is located inside the application and is related with functionality changes. As external we assume the variables, which have indirect influence on the system response, start from the external environment and are in a continuous interaction with the system. The control variables define the different paths of the application execution. For example they determine how the conditional branches will be executed or how many times loop iterations will be repeated. Thus, the control variables influence the execution time deciding how often each part of the application will be executed.

In the context of performance variability, it will in general be too difficult to isolate all possible individual RTSs of a target digital system. For instance, in the case of the BTI phenomenon that we

[1] Optimal/near optimal in terms of the trade-off between cost metrics such as performance and quality cost.

use as a test case, it is rather difficult to enumerate all possible degradation states at which the digital circuit may reside. More specifically, given that the responsible BTI defects have two states (occupied from a minority carrier or free), the average number of possible states that a single device may be in is 2^{η}, assuming an average of η defects per device [4]. Projecting this to a realistic circuit with millions of FETs, it is clear that enumerating all possible RTSs is indeed infeasible. As a result, it is important to engage reliability analysis techniques that will alleviate this complexity. Such techniques are to be invoked in the context of the respective modeling Work Package of the HARPA project, namely WP4.

3.2 Characterization

Characterization represents the evaluation process of the cost metrics of every RTS. In most cases, this is not a simple determination of one cost metric; it leads to a Pareto surface in the multi-dimensional exploration space [10]. During the cost evaluation all the RTSs are examined for different platform configurations. Each configuration is evaluated for specific cost metrics. The optimal trade-offs between the cost metrics in question create a Pareto surface. Each RTS can be characterized by many cost metrics obtained by using high-level cost estimators. The aim is that all costs to be quantified for every RTS for each different platform configuration. The two typical cost metrics are the energy consumption and the performance that is expressed as total delay (latency). The notion of the Pareto optimality [10] is useful for handing the RTS cost dimensioning giving a set of optimal configurations. A designer can make trade-offs within this set without considering the full range of every parameter combination. Pareto analysis [11] has been used in many engineering practices and especially in design-space exploration problem [12]. A design space represents a cloud of design points with multiple cost metrics. As design point is defined a run time situation (RTS) under a specific system mode. The Pareto boundary is the curve, which includes only the points, which are closer to the cost axes giving the best tradeoff in comparison with the rest points. The meaning of a Pareto curve is not a single optimal solution for every cost dimension separately, but a space of partially optimal solutions based on the running requirements. Thus, a Pareto curve gives the intermediate dominant solutions between the individual costs. To summarize, the Pareto curves describe the trade-offs of the possible system configurations, giving a classification of the optimal design points. Upon these trade-offs, constraints may be applied. That way, points that may be otherwise optimal may become prohibitive. The above concepts are illustrated in Figure 17 through Figure 19.

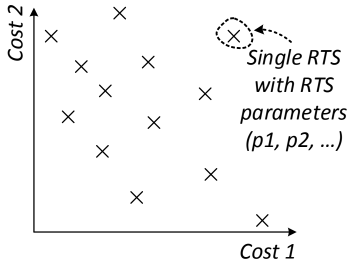

Figure 17: RTS Identification & Characterization

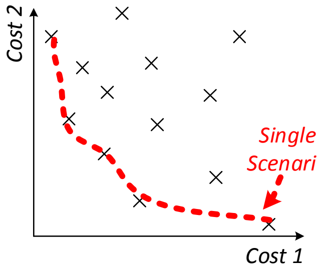

Figure 18: Front of Pareto optimality

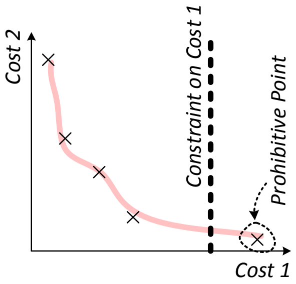

Figure 19: Application of constraints

3.3 Clustering

As we have mentioned the individually handling of each RTS would lead to excessive overheads at run-time, since the source code and all configuration settings would need to be stored for each RTS and applied at run-time. Thus, RTSs are clustered into groups, called system scenarios. So the system reactions are realized in a system scenario basis making the implementation of a potential run-time scheduler much simpler. But the clustering process introduces inevitably an overestimation, termed *clustering overhead*, caused by the deviation between the real cost of the RTS and the estimated cost which is the representative cost for the system scenario of the RTS. The idea behind clustering is instead of many cases we assume fewer cases paying a price on the estimation accuracy. For example, we suppose one hundred RTSs whose cost variation is 5%. If we regard, that the most cost consuming case represents all the others we have an overestimation at the system scheduling process 5%. If we take an average case the overestimation will be lower but the resource distribution will not satisfy some deadlines so it is not recommended for hard deadline systems as we examine in our study. We see that the variation cost in a group of RTS defines the accuracy of the run-time resource assessment. Thus the first criterion for the RTS classification into system scenarios is the cost similarity of the RTSs exploiting the short cost distance between the clustered RTSs and the worst case RTS.

The overestimation is incurred in every appearance of this RTS. Thus, the total overestimation will be proportional not only to the distance between RTS cost and system scenario cost but also to the frequency of every particular RTS. The similarity between costs of different RTSs or in general sets of RTSs (system scenarios) has to be quantified, e.g. by the normalized, potentially weighted, distance between two N-dimensional Pareto surfaces. Based on this distance, the quality of alternative system scenario options can be quantified, e.g. how to cluster RTSs in different scenario [1].

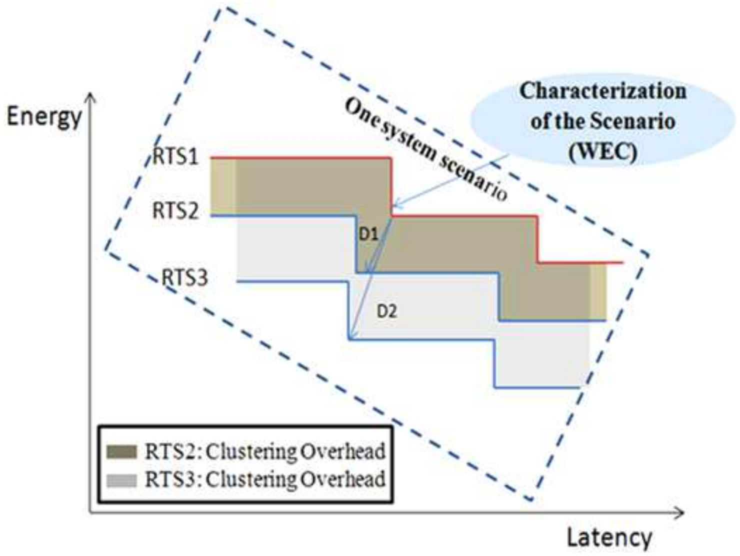
Energy
Latency
Characterization of the Scenario (WEC)
One system scenario
RTS1
RTS2
RTS3
D1
D2
RTS2: Clustering Overhead
RTS3: Clustering Overhead

Figure 20: Clustering overhead representation for the clustering of three RTSs in a single system scenario; WEC represents the Worst Estimation Case [13]

3.4 Detection

In order to mitigate performance variability, we need to implement a platform controller/scheduler which will navigate through the available scenario inventory. A resource scheduling based on the running system scenario requires a detection implementation to identify the running situation and recognize the corresponding scenario. This mechanism can be embedded in the middleware, e.g. RTOS, of the target platform. In the context of the HARPA project, we opt for splitting the detection effort between the OS and the middleware. Based on difference in responsiveness of these two abstraction levels, decisions and countermeasures will be invoked accordingly. In the current report, we focus on the faster run time component, since the OS is covered by WP1 of the HARPA project. We intend to cover detection speeds up to the millisecond range to capture dynamic effects that happen quite fast. That is opposed to the OS level where the update rate will be more in the order of a second.

A run time system scenario detection engine has two basic functionality parts. The first is the detection of the state change by monitoring the values of the RTS parameters and the second is the system scenario determination, which required a predefined decision tree. The sensing of the value changes can be easily implemented by installing a profiling tool at the application source code, which updates the system. At a lower abstraction level, hardware monitors can provide relevant information. The realization of the decision tree is a more demanding process, which adds overhead on both execution time and memory footprint. The critical point is to keep the overhead in low levels maintaining the benefits by exploiting the scenario recognition. A decision tree is synthesized by the nodes, which represent the variables and by the edges, which correspond to the variable values. A wide range of variables and values can create a very complex decision tree. The challenge is to discover heuristic techniques to traverse the decision tree with the minimum cost.

A generic implementation of a detection function f, which incorporates requirements like flexibility and small overhead, is a multi-valued decision diagram comprising a directed acyclic graph $G = (V, E)$ and a labeling of the nodes and edges [9]. A source node exists getting label ξ_1, inner nodes get labels from 2 to n (where n is the number of RTS parameters) and the sink nodes get labels from 1 to s (where s is the number of system scenario). Each inner node ξ_k corresponds to exactly one RTS parameter and has a number of outgoing edges equal to the number of the different values the associated RTS parameter. Scenario detection constitutes a traversal of this graph. On each path from the source node to a sink node each RTS parameter is encountered at most once. When a detection mechanism is used, it introduces two main costs related with: (i) the code size overhead and (ii) the execution delay. The scope is to construct a detection algorithm with the minimum implementation cost. This already reveals a major requirement for the detection of system scenarios related to performance variability of digital systems. In, we can see an example of the aforementioned detection tree. Transformations have been applied on the DAG (merging certain nodes), in order to minimize the overhead of detecting system scenarios at any point.

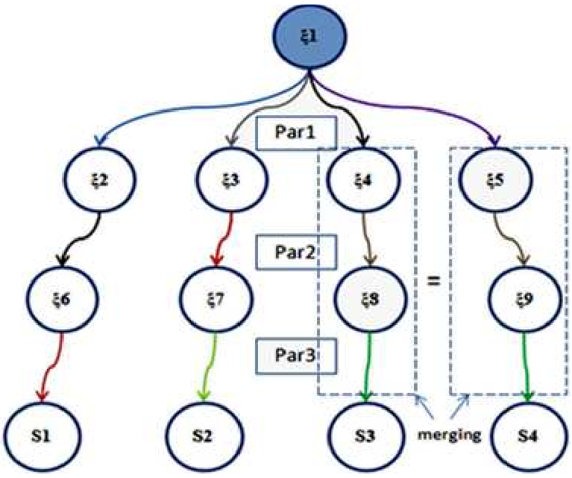

Figure 21: A direct acyclic graph is used to formulate the detection sequence of occurring system scenarios [13]; certain optimizations are in order, so that the detection complexity becomes more affordable

3.5 Switching

Having identified the system scenarios and the suitable detection approach, the next step is the implementation of a run-time algorithm to decide for the system reaction and the switching of the platform configuration in real time. At the identification step, we characterize the scenarios so we can estimate, at design time, the tuning configuration for every system scenario that respects the application constrains with minimum energy cost. The tuning configurations can be related with voltage scaling and frequency scaling or other power saving techniques like processor resizing [14] or cache resizing [15]. Each system scenario corresponds to an optimal set of system configurations, e.g. an energy-delay Pareto curve of potential working points stored in the system scenario list.

What we need at this point is the implementation of an engine, which will react to the detection of a new system scenario being triggered and decide whether to switch from the current scenario or not, while exploiting this information and taking into consideration the switching cost. If the new system scenario is not expected to last very long and the gain is limited then we cannot afford a high switching cost because that will probably be lower. As *switching overhead*, we define the cost for the switching from one scenario to another. This cost will normally depend heavily on the initial and final state.

4 Interfaces & Requirements for the HARPA RTE running System Scenario Mitigation

In the previous Section, we have illustrated the basic steps of the system scenario methodology. In view of the performance variability problem, we can now list the interfaces and requirements, so that this methodology can be applied on performance variability mitigation. Furthermore, a major constraint is the responsiveness of the mitigation engine, which, given the run time context, should be in the order of milliseconds. To illustrate the interfaces and requirements, we repeat the high-level schematic of system scenario application on performance variability, seen in Figure 22. The annotated (Sub)subsections analyze the displayed requirements in more detail.

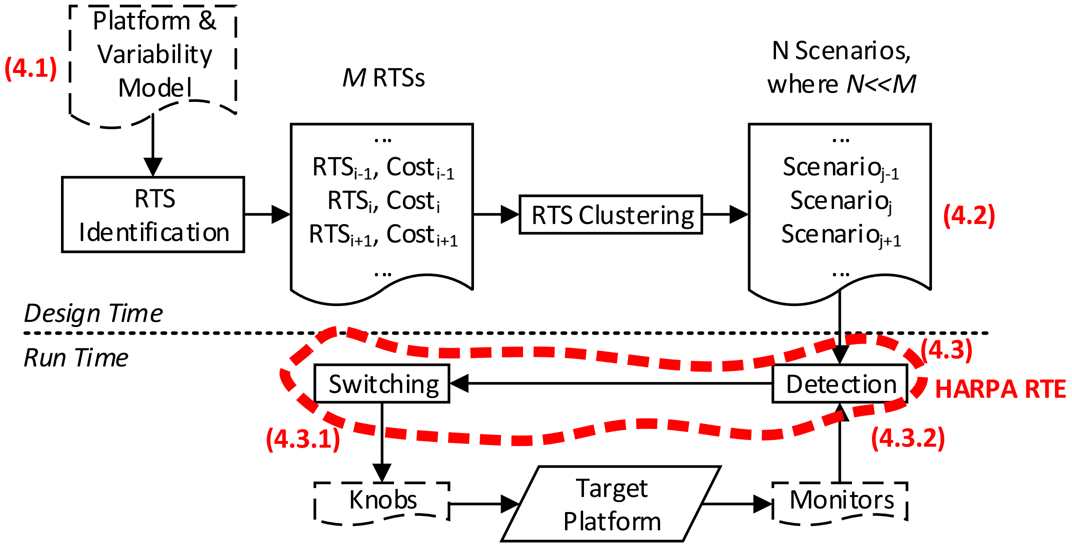

Figure 22: Highlighting the interfaces and general requirements of the system-scenario-based HARPA RTE; again dashed components correspond to Work Packages 3 and 4 of the HARPA project

4.1 Performance Variability Model

A major prerequisite is a model that will describe the performance variability of the target platform. In the general case, this model should be time, temperature and workload dependent, so that the impact of platform reconfiguration can be realistically captured. In the case of the BTI phenomenon, which is used as a rough test case in this report, such models exist and have been propagated to the IP block level [6]. Finally, this model should capture both functional and parametric aspects of system dependability, since it has been already demonstrated that both aspects are tightly entangled in complex digital systems [16]. Creation of this model requires the propagation of knowhow from low abstraction levels, as in the case of TCAD simulations [17], to the middleware level where the HARPA RTE will operate. WP4 of the HARPA project is focusing on the provision of such a model.

4.2 Pre-stored System Scenario Information

In Subsection 2.3 and throughout Section 3, we have presented the concept of system scenarios and how they can be used to optimally configure a platform that exhibits certain variability (either inherent, or in terms of workload). In any case, it is very important for system scenario information to be available in a flexible way. It is unavoidable that this information will be hardcoded to the platform and specifically in the HARPA RTE. However, we need to reflect on the derivation of such information as well as the way that it is going to be stored.

We have already mentioned that system scenarios are typically derived at design-time. However, it is very important to factor in time-zero variability, which is heavily present in modern digital systems. As a result, time-zero calibration of the system scenarios is required in order for the scenario information to be relevant for each platform sample. We can assume that this process occurs during the testing phase of the platform, which is typically performed for frequency and voltage binning [18] [19]. Already in the case of time-zero variability, such calibration has already been proposed [3].

Regarding the storage of the scenario information, it is very important to conform it to the detection phase of the system scenario methodology. Previous work has suggested the use of a DAG, which captures all RTS parameters and their possible values, finally leading to the available scenarios. Theoretically, this is a very suitable formulation. However, with a real platform in mind, the scenarios should be pre-stored and pre-coded in an effective way so that they have minimal memory footprint and so that the juxtaposition between RTS parameters and scenarios can be easily derived. A Look Up Table (LUT) is an easy implementation that serves both purposes. More intelligent implementations include a traditional Neural Network, which also serves the purpose of refining/altering the scenario information. The latter capability is relevant in the case of Dynamic Scenarios, which will be addressed in future stages of the HARPA project (D2.6).

4.3 The Actual HARPA RTE

According to Section 3 and from a functionality perspective, the system-scenario-based HARPA RTE should perform two main tasks: (1) *detection* of the current scenario for the platform and (2) *switching* between scenarios when the necessary conditions apply. In the simplest case, this can be implemented as a Finite State Machine the branches of which correspond to switching upon the detection of certain RTS parameters (or more simple cost metric values). This implies that the HARPA RTE must be a programmable module with low latency (response time up to or just below 1ms), directly connected to knobs and monitors available on the platform. Alternatively, the HARPA RTE could be implemented on one of the nodes of the broader HARPA target platform. These connections are addressed in detail in Subsubsections 4.3.1 and 4.3.2. A significant concern corresponds to the overheads involved in detection of and switching between system scenarios.

Already in Subsections 3.3 through 3.5, we have discussed that the detection of scenarios is associated with an overhead, which is positively related to the size of the scenario inventory. Similarly, a switching overhead is associated with the available platform knobs and also depends on the frequency of occurrence of RTSs belonging to different scenarios. Finally, the clustering of RTSs to scenarios also involves a certain overhead, related to the distance of an RTS from the worst case RTS of the parent scenario. In view of these overheads, it is very important to quantify them as early as possible, so that they can be taken into consideration during the HARPA RTE implementation. Table 1 summarizes the implementation concerns of the HARPA RTE regarding the above overheads.

Table 1: Implementation concerns regarding the overheads of the system scenario methodology

System Scenario Methodology Overheads	**Depending on …**	**Ways to Address/Minimize**
Detection Overhead	Number of Scenarios	Quality of RTS clustering
	Variability of RTS Parameters	
	Scenario DAG Implementation	Scenario DAG Transformations [13]
	Monitor Responsiveness	Available from monitor implementation
Switching Overhead	Frequency of inter-scenario occurrence	Quality of RTS clustering
	Knob Responsiveness	Available from knob implementation
Clustering Overhead	Trade-off of Number of Scenarios vs Worst estimation cost (see Figure 20)	Improve Quality of RTS clustering technique

4.3.1 Monitor-Detection Interface

The concept of the HARPA RTE, requires that it is fed with platform state information at all times. The source of such information is a set of monitors that are placed across the target platform. Depending on the target system, these monitors can be implemented in hardware (e.g. thermal sensors) or software (e.g. monitoring threads in a many-core system). In any case, the information contained in the monitors should be directly available to the HARPA RTE, either in the form of a periodical update or an asynchronous fetch. In the hardware case, memory mapped registers are typically used for the storage of sensor values that are placed within an integrated system [20].

4.3.2 Switching-Knob Interface

In the case of the HARPA RTE-Knob interface, it is very important to have a quantified view of the response time and phase lag, from the signaling of the knob up to the respective change taking effect. That way, the switching overhead can be realistically estimated and the decision for a change between platform configurations can be more credible.

5 Conclusions

This report has presented the interfaces and requirements of the HARPA RTE, assuming the usage of the system scenario methodology for the mitigation of performance variability. The system scenario scheme is a rather recent approach towards optimized system design, aiming at the balance of a variety of cost metrics regarding system operation.

Initially, we have given some introductory information on system scenarios and defined key terms. In each case, we have presented relevant examples, in the context of performance variability. The main concept vehicle of this report has been the phenomenon of Bias Temperature Instability (BTI), which affects the threshold voltage of FET devices.

Moreover, we have presented the steps of the system scenario methodology: Excerpts of system operation are initially identified and characterized based on certain cost metrics. These constitute the *RTSs* of the target system. RTSs with similar cost metric values are clustered into *scenarios*. The run time engine that mitigates performance variability is responsible for detecting and switching between scenarios, so that the target platform is optimally configured at all times. Based on these steps, we have identified requirements and interfaces of the HARPA RTE.

Initially, the provision of a model for performance variability is of major importance, since it will allow the identification of a significant number of RTSs. Enumeration of all possible RTSs is deemed infeasible, due to the extreme complexity of performance variability in modern digital systems (in the case of BTI the number of possible RTSs is exponentially exploding). Assuming that identification, characterization and clustering of RTSs is complete, the HARPA RTE should have a scenario inventory at its disposal from where to draw optimal platform configurations in a demand-driven way.

The two main functionalities of the HARPA RTE will be the detection of and switching between scenarios (and their respective platform configurations). This will be performed in tight collaboration with the available platform knobs and monitors. As a result, we have identified the following enables for proper operation of the HARPA RTE: (1) “Smart” clustering of RTSs, so that the clustering, detection and switching overheads are kept relatively low. (2) Transformations of the scenario DAG, to enable faster detection. Alternatively, a detection scheme based on a fast (and trainable) utility (such as a neural network) is a very relevant implementation. (3) Selection of knobs and monitors that are conformed to the millisecond response time of the HARPA RTE. Assuming memory-mapped registers, the handover of information should be of that order.

6 Works Cited

[1] S. V. Gheorghita, et al., "System-scenario-based design of dynamic embedded systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 1, Jan. 2009.

[2] N. Zompakis, A. Bartzas, F. Catthoor, and D. Soudris, "System scenarios-based architecture level exploration of SDR application using a network-on-chip simulation framework," *Microprocessors and Microsystems*, vol. 37, no. 6, pp. 544-553, Aug. 2013.

[3] C. Sanz, et al., "Combining system scenarios and configurable memories to tolerate unpredictability," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 13, no. 3, Jul. 2008.

[4] M. Toledano-Luque, et al., "Response of a single trap to AC negative Bias Temperature stress," in *Reliability Physics Symposium (IRPS), 2011 IEEE International*, Monterey, CA, 2011, pp. 4A21-4A28.

[5] T. Grasser, et al., " The Paradigm Shift in Understanding the Bias Temperature Instability: From Reaction–Diffusion to Switching Oxide Traps ," *Electron Devices, IEEE Transactions on*, vol. 58, no. 11, pp. 3652-3666, Nov. 2011.

[6] D. Rodopoulos, P. Weckx, M. Noltsis, F. Catthoor, and D. Soudris, "Atomistic Pseudo-Transient BTI Simulation with Inherent Workload Memory," *Device and Materials Reliability, IEEE Transactions on (to appear)*, 2014.

[7] A. Iosup, N. Yigitbasi, and D. Epema, "On the Performance Variability of Production Cloud Services," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011, pp. 104-113.

[8] N. Zompakis, V. Tsoutsouras, A. Bartzas, D. Soudris, and G. Pavlos, "Dynamic Frequency Scaling for MPSoCs based on Chaotic Workload Analysis," in *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*, Hannover, Germany, 2010, pp. 1-8.

[9] V. S. Gheorghita, "Dealing with dynamism in embedded system design," Technische Universiteit Eindhoven Doctoral Dissertation, 2007.

[10] M. Geilen, T. Basten, B. Theelen, and R. Otten, "An Algebra of Pareto Points," *Journal Fundamenta Informaticae*, vol. 78, no. 1, pp. 35-74, Sep. 2007.

[11] M. A. Yukish, "Algorithms to identify pareto points in multi-dimensional data sets," The Pennsylvania State University Doctoral Dissertation, 2004.

[12] M. Gries, "Methods for evaluating and covering the design space during early design development," *Journal Integration, the VLSI Journal*, vol. 38, no. 2, pp. 131-183, Dec. 2004.

[13] N. Zompakis, A. Papanikolaou, R. Praveen, D. Soudris, and F. Catthoor, "Enabling efficient system configurations for dynamic wireless baseband engines using system scenarios," in *Signal Processing Systems (SiPS), 2011 IEEE Workshop on*, Beirut, 2011, pp. 305-310.

[14] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," in *ISCA '03 Proceedings of the 30th annual international symposium on Computer architecture*, 2003, pp. 336-349.

[15] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," in

Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on, Haifa, 1999, pp. 248-259.

[16] D. Rodopoulos, A. Papanikolaou, F. Catthoor, and D. Soudris, "Demonstrating HW-SW Transient Error Mitigationon on the Single-Chip Cloud Computer Data Plane," *IEEE TVLSI (to appear)*, 2014.

[17] S. M. Amoroso, L. Gerrer, and A. Asenov, "3D TCAD statistical analysis of transient charging in BTI degradation of nanoscale MOSFETs," in *Simulation of Semiconductor Processes and Devices (SISPAD), 2013 International Conference on*, Glasgow, 2013, pp. 5-8.

[18] V. Zolotov, C. Visweswariah, and J. Xiong, "Voltage binning under process variation," in *ICCAD '09 Proceedings of the 2009 International Conference on Computer-Aided Design* , 2009, pp. 425-432.

[19] A. Datta, S. Bhunia, J. H. Choi, S. Mukhopadhyay, and K. Roy, "Speed binning aware design methodology to improve profit under parameter variations," in *Design Automation, 2006. Asia and South Pacific Conference on*, Yokohama, 2006.

[20] Intel Labs, "Using the SCC Sensor Registers," Intel Corporation, 2010.