ENABLING EFFICIENT SYSTEM CONFIGURATIONS FOR DYNAMIC WIRELESS BASEBAND ENGINES USING SYSTEM SCENARIOS

Nikolaos Zompakis¹, Antonis Papanikolaou¹, Raghavan Praveen², Dimitrios Soudris¹, Francky Catthoor²
¹ECE School, National Technical University of Athens, 15780 Zografou, Greece
Email {nzompaki, antonis, dsoudris}@microlab.ntua.gr
²IMEC vzw, Kapeldreef 75, 3001 Heverlee, Belgium
Email {ragha, catthoor}@imec.be

ABSTRACT
Next generation mobile wireless systems (4G) support a wide range of communication protocols and services opening new design challenges. The desired flexibility requires an effective utilization of system resources. In this paper, we introduce the concept of system scenarios in wireless communication baseband engines. The scenario methodology classifies the system behavior from a cost perspective and provides the necessary information for an effective system tuning. Apart from the case study, we propose improvements at the detection of the scenarios at run time achieving a better trade-off between cost estimation accuracy and detection overhead. From our case study, a representative example the WLAN communication protocol, we can demonstrate the accurate prediction of the execution time of each packet, which on average is 90% shorter than the worst case, allowing us to use the remaining time for the optimization of specifications like power consumption.

Index Terms — system scenarios, timing slack, wireless communication, baseband engines

1. INTRODUCTION

Modern wireless mobile systems support a wide range of communication protocols and services. The desired flexibility is restricted mainly by their tight requirements. So an efficient utilization of the available resources based on the running situations and with the minimum configuration cost is needed. System adaptation can be implemented either at application level, selecting an effective task mapping technique, or at platform level, applying a dynamic voltage scaling technique (DVS).

Software Defined Radio (SDR) is a well-known example [1], it combines numerous communication standards in a single device. From the user perspective, these systems have stringent requirements on size, performance and energy consumption. Optimizing energy efficiency is key for maximizing battery lifetime between recharges. These requirements should be met in the context of the increasing resource usage dynamism of modern applications. Worst-case platform design leaves a lot of room for optimization if these dynamics can be properly predicted at run-time. New design methodologies are being proposed which can handle the increasing dynamism under tight requirements.

State-of-the-art design methodologies identify use cases and deal with them separately. One of the most promising design methodologies is system scenarios [6]. Scenarios are extracted at an early design phase and can be divided in two categories: use-case scenarios and system scenarios [5]. Use case scenarios cluster possible user actions and system reactions required for the development of a system with predefined functional specifications. But use case scenario behavior still varies significantly in terms of performance metrics/cost. On the other hand, system scenarios cluster system behaviors that are similar from a multi-dimensional cost perspective so that run-time decisions can exploit this cost similarity. As a result, cost (e.g. power, delay) can be traded-off for design time effort.

This paper introduces, for the first time, the concept of the system scenario for wireless communication baseband engines. Our work is not limited to applying system scenarios; we propose novel methodology extensions for the clustering of system behaviors into scenarios and the run-time detection of scenarios. More precisely, we study the main trade-off between clustering overhead, which is correlated with how representative are the scenarios, and the detection overhead, proposing transformations at the implementation of the detection algorithm.

2. RELATED WORK

The scenario concept was introduced fully systematically in [8]. The aim was to capture the data dependent dynamic behavior inside a thread in order to better schedule a multi-thread application on a heterogeneous multi-processor architecture. [9] proposes a scenario-based framework for managing processor resources while achieving QoS on contemporary multi-core processors running media-streaming applications. In [10] scenarios have been used to refine the estimation of the Worst Case Execution Time (WCET). [11] proposes a mapping technique and compiler which identifies the hot path(s) and merges or duplicates the kernels, called packet processing functions (PPF), in order to maximize system throughput. In [12] the authors use scenarios to enable more freedom for Global Loop
Transformations. Compared to [11] the work in [12] focuses on the control-flow graph (CFG) of hot code paths as a whole. Also, their target domain is the multimedia domain, whereas the target domain of [11] is networking. In [12] the authors analyze the loops with varying trip count to design a dynamic loop predictor. In [13] authors propose heuristics to simplify the complexity of control flow graphs (CFGs). In [2] the trip count profiling is used to enable advanced loop optimizations in dynamic translators. In this paper [3] authors propose to utilize the trip count profiling information of the loops with varying trip count and refine the input for the scenario creation.

Generally the systems scenario methodology is particularly beneficial for multimedia that exhibit dynamic behavior with multidimensional costs. Usually, most of these applications are streaming and have to deliver a given throughput which imposes specific time constraints. [4] presents a design methodology that provides a systematic way of detecting and exploiting system scenarios for streaming applications. A scenario is defined as the application behavior for a specific type of input data, i.e. a group of execution paths for that particular group of input data.

### 3. WIRELESS PHY-ORIENTED METHODOLOGY

The target application domain in this work is wireless PHY systems and specifically Software Define Radio (SDR) and Cognitive Radio platforms. Our analysis is based on the concept of Run-Time Situations (RTSs) [5]. Every RTS, or system behavior, is associated with one or several primary cost objectives. These objectives for wireless mobile systems are: 1) throughput, which is related with the performance of the platform, and 2) energy consumption to execute a given task set, which is related with increased battery life. These objectives are cost oriented and they have to be optimized while respecting strict latency constraints.

The active RTS can be identified using specific system parameters, so-called RTS parameters. In wireless applications, RTS parameters can include every variable which leads to a different communication mode. However, the possible values of these parameters may lead to an exponential number of RTSs which in combination with the multi-dimensional cost of every RTS can cause a too complex design process. For that reason, we focus mainly on those parameters which have the highest impact on the most significant costs.

Below, we will apply the system scenario theory to a representative wireless application module. Our case study, will concentrate on the RTSs of the baseband stage (BB) and the FEC stage during a wireless transmission. The goal of our case study is to illustrate how system scenarios are defined in a wireless application, the tradeoffs that must be considered and to quantify the difference between the Worst Case Execution Time and the estimated execution time using scenarios. This difference is idle time for the system that can be used to minimize power consumption.

---

**Step 1. Identification of the RTSs**

Our methodology starts with the characterization of all possible RTSs which occur in a wireless PHY-based system. We identify all the variables (RTS parameters) that affect the state of the system from a functionality or implementation point of view. System variables can be classified in two categories, control and data variables. Control variables define the actual execution path of the application, by determining, for example, which conditional branch is taken or how many times a loop will iterate. Data variables represent the data processed by the application. Control variables have a higher impact on execution time, as they decide how often each part of the program is executed. Hence we focus on them. A typical control variable, from the wireless system domain, is the modulation scheme. Every wireless protocol supports multiple modulation schemes which can change during transmission and they have a pronounced impact on the performance of the platform. A typical data variable for a wireless application is the size of the transmitted packets which has a large impact on the volume of the data that the system has to handle and also on the duration of the application execution.

---

**Figure 1: Gray Box Model, a description of the application that isolates non-determinism and enables the identification of scenarios.**

To extract the RTSs of a wireless PHY-based system we need a detailed description of the targeted application. The left hand side of Figure 1 illustrates a model that splits the
application into deterministically behaving components and shows how they interact non-deterministically with each other, introducing the concept of the Thread Frames (TF). TFs are defined as maximally sized pieces of code without any kind of event driven handling or dynamic task creation. Every application can be split into deterministic parts (e.g. data dependent branches, while loops, conditions, etc) and the TFs and the non-deterministic parts (including all the start up events) which are represented by the event handling schemes. In Figure 1, we present a typical example of a wireless system and the corresponding gray box [14] model which focuses on the critical application parameters splitting the main TF in sub-tasks (ST). The SDR digital processing can be divided into two main TFs (TF1 and TF2). TF1 includes the processes of the Digital Front-End (DFE) and TF2 includes the processes of the Baseband Engine (BBE) and the forward error correction (Flex-Fec). An event handling synchronization between DFE and BBE which introduces non-determinism is the cause of this division into TFs. We focus our scenario analysis on TF2.

For the definition of the RTS parameters, we concentrate on the STs of the gray box model. We see that six STs exist: ST1 is the preamble processing, ST2 includes the necessary FFTs, ST3 implements the scheduling of the multiple antennas, ST4 performs demodulation, ST5 decodes the data and ST6 checks and corrects possible data errors. The STs exhibit branches during their execution which depend on specific application parameters. These branches depend on the following parameters: 1) the number of the antennas, 2) the bandwidth, 3) the number of iterations during decoding and 4) the block size of the data in the decoding. Table 1 outlines the possible values of the RTS parameters. All the possible combinations of these parameters define an exploration space of 72 RTSs.

| Num_antennas | 1, 2, 4 |
| BW (MHz) | 20MHz, 40 MHz |
| Block Size | ≤648 bits, 648 bits < M ≤1080 bits, L=1080 bits |
| Iterations | 2, 5, 6, 7 |

Table 1: RTS Parameters of TF2

Step 2. RTS Characterization

Identification and characterization of RTSs requires knowledge about the operating conditions (signal distortions, noise level) of the system. Characterization of RTSs implies the calculation of an N-dimensional Pareto surface in the cost space, where N is the number of independent cost metrics. It quantifies all the costs for each different platform configuration per RTS. Usually the most critical costs are: 1) packet processing latency and 2) energy consumption. Hence the exploration space is two dimensional and every RTS is represented by a Pareto curve (Figure 2) since alternative platform configurations, e.g. supply voltage level, will result in different costs for the same RTS. The distance between Pareto curves depends on the impact of the respective RTS parameters, e.g. the Pareto curves of two RTSs with different MIMO schemes (e.g. 1×1 - 4×4), will have a significant distance since the number of the active antennas strongly influences the throughput and the energy consumption. In the presented case study we use a single cost metric, namely the packet processing execution time. For the characterization of the RTSs, we use representative results which have been extracted from a real wireless platform which is developed by IMEC [7].

![Figure 2: Pareto curves quantifying the cost impact of different RTSs and scenarios](image)

Step 3. Clustering of the RTSs in System Scenarios

Individually handling all RTSs would lead to excessive overheads at run-time. They must be clustered into scenarios. The similarity between costs of different RTSs or in general sets of RTSs (scenarios) has to be quantified e.g., by defining the normalized, potentially weighted, distance between two N-dimensional Pareto surfaces as the size of an N-dimensional volume that is present between these two sets. Based on this distance, the quality of potential scenario options can be quantified, e.g., to decide whether or not to cluster RTSs in different scenarios [5].

Clustering is implemented using a cost function related to the target objective optimization and takes into account: 1) how often each RTS occurs at run-time, 2) the distance of their Pareto curves 3) the run-time scenario detection and switching overhead. The scenario characterization (Pareto curve) results from taking the worst-case cost point for each different platform configuration among RTSs. For example (in Figure 2), we have a 2 dimensional Pareto set with three RTSs (RTS1, RTS2, RTS3) and the corresponding Pareto curves. The result of the RTS clustering is one system scenario which is characterized by the worst Pareto curve (red Pareto curve - RTS1). Characterizing RTSs with the worst estimation case (WEC) of the scenario inevitably introduces a clustering overhead (light gray and brown area in figure 2). If we want to formulate this problem for every RTS, we can define the clustering overhead (CO) as the distances between the costs of RTS1 and 2 (D1 and D2) from the cost of RTS3 (Eq. 1). This overhead manifests every time that the RTS occurs at run-time. Thus, the total
clustering overhead (TCO) will be proportional to the frequency of the RTS appearance (Eq. 2). TCO represents the difference between our prediction and the actual cost (execution time) and it is always positive since our clustering tends to overestimate the cost.

\[
TCO = \sum_{i=0}^{n} F_{RTS(i)} CORTS(i) + \sum_{i=0}^{n} F_{RTS(i)} DORTS(i) + \sum_{i=0}^{n} F_{S(i)} SO_{Sen(i)}\tag{3}
\]

TO=\sum_{i=0}^{n} F_{RTS(i)} CORTS(i) + \sum_{i=0}^{n} F_{RTS(i)} DORTS(i) + \sum_{i=0}^{n} F_{S(i)} SO_{Sen(i)}\tag{4}

An inherent trade-off exists between the parameters of Eqs. 3 and 4. Keeping the total clustering overhead (CO), low results in many scenarios that contain a few RTSs each which increases the total switching overhead. The policy we follow must depend on the target optimization objective and on the characteristics of the system. During the clustering of the RTSs into scenarios, we consider the switching and the detection overhead. In this case study, we concentrate on the detection and clustering overhead. During clustering we must pay attention to the detection overhead, otherwise we will have to construct a very expensive detection algorithm. To avoid that, we have to consider other RTS distributions and hence other alternative RTS clusters. To achieve an efficient trade-off between clustering and detection overhead we introduce transformations of detection schemes which will be explained later.

**Step4. Detection of the System Scenarios**

After the generation of system scenarios the next step is the realization of a detection algorithm which can recognize at run-time the scenario to be executed. The detection mechanism will be embedded in the middleware (e.g. RTOS) of the targeted platform adding some overhead on both execution time and memory footprint. It is critical to keep this overhead small while maintaining the benefits by exploiting the knowledge from the scenario recognition. The detection is implemented through monitoring the changes of the RTS parameters at run-time. Their value range has great impact on the final overhead. The challenge is to discover heuristic techniques which can detect the scenarios with minimum cost.

A generic implementation of a detection function \(f\), which incorporates requirements like flexibility and small overhead, is a multi-valued decision diagram which consists of a directed acyclic graph \(G = (V,E)\) and a labeling of the nodes and edges [3]. The sink nodes get labels from \(1,\ldots,m\) and the inner (non-sink) nodes get labels from \(\xi_1,\ldots,\xi_n\) being the source node. Each inner node, \(\xi_k\), corresponds to exactly one RTS parameter and has a number of outgoing edges equal to the number of the different values \(\xi_k(1)\) of the associated RTS parameter.

Scenario detection is 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 overheads: (i) code size and (ii) average run-time evaluation cost. To construct a detection algorithm based on a decision diagram for our wireless platform, we have to implement a restricted programming language. So we need to define a set of instructions that will realize the detection mechanism.

In the instruction set, 3 instructions (JEQ, JL, JMP) are used to traverse the decision diagram and 3 other instructions (SEQ, SLE, SBK) for the detection of the final scenario. The instructions JEQ, JL, SEQ, SLE have three arguments: 1) the current node, 2) the value of the RTS parameter that will be used to guide the program to the target node and 3) the next running instruction or the detected scenario. The other instructions are unconditional jumps to the node specified by the argument. These instructions implement the program for the transitions from one node to another by matching the values of the running RTS parameters with the values which appear at the edges of the decision diagram.

Figure 3 illustrates the implementation of a detection algorithm for a wireless application with 3 RTS parameters (bandwidth, number of antennas, coding). The detection algorithm starts from inner node \(\xi_1\), where we check, with the instruction \{JEQ: 1, 20, 3\}, if the current bandwidth is equal to 20 MHz. If the condition is true the detection goes to line 3. At the new instruction line, we are at the inner node 2 and we have a new RTS parameter (number of antennas) to check and a new instruction to run \{JEQ: 2, 1, 5\}. The procedure continues until the decision diagram reaches a detected system scenario.

**Transformations**

Detection overhead is directly correlated with the complexity and depth of the decision diagram. If it is complex, many instructions and significant memory footprint will be required to describe all the cases. Moreover as depth increases, more conditions for the detection of any given scenario are needed. To limit this overhead, we apply transformations which simplify the decision graphs.

Gheorghita et al [15] have proposed transformations that are based on similarities between nodes and outgoing
edges. They also use prediction instead of detection to identify future scenario instances and their transformations impact the quality of the prediction. Thus, they trade off decision overhead for prediction/detection quality. On the other hand, our approach can trade off detection overhead for clustering overhead. It is an extension of the existing techniques.

![Decision diagram of a wireless application](image)

**Figure 3: Decision diagram of a wireless application**

The proposed transformations are: 1) bypass nodes and 2) merge edges of the decision diagram. These actions can affect the optimal RTS clustering into scenarios and alter the clustering overhead. So the parts of the diagrams that we apply these transformations have to be selected carefully. The criterion that we use is the impact of the RTS parameter. Edges and nodes which are correlated with RTS parameter with negligible impact on the cost metric are preferred from others. Thus, we succeed to simplify the detection graph with minimum additional clustering overhead. In the case of Figure 3 for example, if we know that changes at MIMO schemes can cause a variation of the TF execution time by 30%-40% and coding schemes by 5%-6% respectively, we will prefer to merge the edges at the nodes $\xi_4$, $\xi_5$, $\xi_6$ which are correlated with the possible values of the coding. We see that this decision diagram is quite simple to apply transformations, at $\xi_1$ we have just two edges. They become more useful as decisions diagrams grow for more complex cases.

Merging the outgoing edges of a node implies that we cluster RTSs with different values for this parameter into the same scenario. If the impact of the specific parameter is significant on the costs, the clustering overhead will increase. On the other hand if the impact of the parameter is small the introduced clustering overhead will be low. In the extreme case, if the impact of the RTS parameter on the cost is negligible we can ignore it and completely bypass the corresponding nodes. So to simplify the detection mechanism and achieve an efficient trade-off between detection and clustering overhead we have to consider the impact of the RTS parameters and prioritize the application of the proposed transformations on the decision diagram.

To understand the influence of every RTS parameter on the execution time of the TF, we measure the fluctuation of the TF execution times while varying the different RTS parameters. We found that the number of antennas has the most significant impact on the execution time (24%-70%). The decoding block size (25%-45%), the bandwidth (30%) and the number of iterations for decoding (8%-30%) follow.

The number of iterations at the decoding is the parameter with the least influence on the RTS execution time. So the inner nodes related with the decoding have to be transformed first since the overhead will be minimal. During clustering we have considered the detection overhead, but there is no guarantee that these solutions are efficient from a detection perspective. So for our case study, we examine the impact of the transformations on the depth of the decision diagram compared to the worst-case non-optimized decision diagram. For a full detailed decision, we have 103 inner nodes and 174 edges. After the application of the transformations we have the graph in Figure 4. The complexity of the graph is reduced to just 23 nodes and 49 edges. For example, at the inner node $\xi_5$ we see that we can end up with a scenario by completely ignoring the number of the decoding iterations. At the inner node $\xi_4$, by knowing if the number of the iterations is equal or not to 4, we can directly detect the scenario. Similarly, we apply transformations at the inner nodes from $\xi_5$ to $\xi_2$.

The aim of the transformations is both, to simplify the decision diagram and to keep the added clustering overhead low. The first criterion has been satisfied by reducing the number of the inner nodes and the edges. The second goal can be quantified by the execution time variations of different RTS in the same scenario. This maximum variation was initially between 3 and 11%. After the transformations it is between 5 and 16%. So we expect the added clustering overhead to remain quite reasonable. The equation which gives the clustering overhead, related with the average fluctuation of the RTSes in a scenario is the following:

$$CO = \frac{F_{sc} \sum_{i=3}^{PP} F_{av} ET_{av}}{\text{average execution time in a scenario}, ET_{av}}$$

$F_{av}$: it is the average fluctuation in a scenario, $ET_{av}$: it is the appearance frequency of the scenario

4. RESULTS

In a typical run time situation, the platform has to be configured taking into consideration the worst case. Optimization is possible at the end of the execution if timing slack is available. System scenarios provide us with the opportunity to act proactively giving us a pre-estimation assessment of these slacks. Figure 5 illustrates the distribution of execution times for the processing of 1000 WLAN packets. 64% of the packets have execution times 2000-6000 ms and the worst case (128000ms) corresponds to 1% of packets. A lot of slack is available on average.
5. CONCLUSION

In this paper, we introduce the concept of system scenarios in wireless communication baseband engines. We propose methodology extensions which improve the detection of the scenarios at run time achieving efficient trade-offs between cost estimation accuracy and detection overhead. Finally, we examine a comprehensive case study and show that the scenario methodology can predict timings slacks in the execution of very dynamic applications which designers can utilize to apply power optimizations.

6. REFERENCES