Applications of scenarios in early embedded system design space exploration
van Stralen, P.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
One of the challenges during embedded system design is the application driven design. Due to the application driven design, the objectives that are steering the design of an embedded system are mainly based on the needs of the application(s). Examples of embedded system objectives are performance, power, but also battery lifetime and security. Such, potentially conflicting, objectives severely complicate embedded system design. The system level design methodology reduces the complexity of embedded system design by providing a structured approach to reduce the implementation efforts.

Part of the system level design is the early design space exploration. At a high level of abstraction, the design space of potential designs is partially explored to make early design decisions that reduce the effort that is spent in later design phases. Chapter 2 introduced the design space exploration as it used in this thesis. In our case, a design is defined by a mapping of a multi-application workload onto an MPSoC architecture. This mapping is evaluated using our high-level simulation framework Sesame (Section 2.1) that is capable to determine the quality of a single mapping in the order of seconds. There are many different ways of mapping a multi-application workload onto the architecture. As a result, there is a huge design space of potential mappings that cannot be exhaustively explored. One of the techniques to efficiently search such a large design space is a multi-objective evolutionary algorithm (MOEA). A MOEA uses a genetic algorithm (Section 2.2) that mimics the natural evolution process by keeping a population of mappings and to evolve them over a number of generations. As high quality mappings have a higher probability to reproduce, the fittest mappings will survive.

Although early design space exploration is already complex, it is becomes even more complex due to the growing dynamism in the embedded systems. In this thesis, two sources of dynamism are investigated: at the application side (Part II) and at the architecture side (Part III). At the application side, the reason for the increasing dynamism is twofold: at first the number of applications on a single embedded system increase and, secondly, the behavior of applications themselves become more dynamic. Next, at the architecture side the decreasing technology scale leads to less reliable computation. The smaller the technology, the more susceptible the ar-
Architecture is to faults. These faults may corrupt the output of an application that is running on the architecture. Irrespective of the dynamism inside the embedded system, it must meet the objectives of the system. For that purpose, this thesis captures the dynamism of the embedded system in scenarios and uses them to statically search for a design that is capable of meeting the objective of the embedded system in all cases.

Our problem definition (Section 1.4) concluded with the following research question: "How can scenarios be used to enhance the design space exploration of dynamic embedded systems". This question was answered in Chapters 4, 5 and 7 where a scenario-based design space exploration was described. Figure 8.1 shows the general technique of the scenario-based DSE that aims at finding static mappings that perform well for all potential scenarios. This set of potential scenarios, however, is large and, therefore, it is infeasible to exhaustively evaluate all the explored mappings with the full set of potential scenarios. Our solution is to use a representative subset of scenarios that predicts the quality of a mapping by a representative subset of scenarios. For this purpose, our scenario-based DSE has two components: a design explorer and a subset selector. In the design explorer there is searched for optimal mappings. To perform the search as efficient as possible, a representative subset of scenarios is used to evaluate the mappings. For this purpose, the subset selector is responsible for selecting a representative subset of scenarios. A part of this selection problem is defining the metric that is used to determine if a subset is representative. One of the challenging issues is that the representativeness of a scenario subset is dependent on the current set of mappings in the design explorer (Chapter 4). Therefore, the trainer will contain a set of mappings that is used to train the scenario subset. While exploring the design space, the design explorer will send the selected mappings to the subset selector. These selected mappings will be used to keep the trainer up-to-date with the design explorer. On top of that, mappings in the trainer will be evaluated more thoroughly than in the design explorer. Therefore, the trainer is not only used to contain the mappings that are used to train a representative subset of scenarios, but also to contain the set of currently best mappings. Hence, at

Figure 8.1: A generalized technique to perform scenario-based DSE.
the end of the scenario-based DSE the trainer will contain the final population of (sub-)optimal mappings.

8.1 Application Scenarios

At first, we described how application scenarios were used in scenario-based DSE to describe the dynamism in a multi-application workload. An application scenario considers dynamism of two sources: an inter-application scenario that describes which applications can be active simultaneously and an intra-application scenario that describes the dynamism within applications. Identification of application scenarios is done partly automatically and partly manually. Intra-application scenarios are detected automatically using a profiling-based method. This method is optimized to store the application scenarios as efficient as possible using a so-called scenario database. Next, inter-application scenarios need to be defined manually.

Chapter 3 showed that objectives like execution time and power differ per application scenarios. For early design space exploration, the absolute difference between the obtained non-functional requirements (i.e., mapping fitness) and the value of the non-functional requirement in the final embedded system is not relevant as long as the ordering of mapping is correct. If early design space exploration predicts that a mapping has a higher performance than another mapping, this should really be the case. The difference between the real and predicted fitness, however, does not really matter. In the experiments of Chapter 3, it turned out that the relative ordering of mappings could also vary along the different application scenarios. Therefore, it is of uttermost importance that subset of application scenarios is used that is representative for the complete set of application scenarios to compare the different mappings.

Therefore, Chapter 4 introduced a scenario-based DSE to search for a mapping that, on average, optimizes the non-functional requirements (i.e., metrics like execution time and power of the mapping when the application workload is executed on the system) given all the application scenarios in the scenario database. More specifically, a co-exploration is performed where both the design space of potential mappings and scenario subsets are explored simultaneously. This means that, in comparison to a traditional DSE a part of the computing power is not spend on the search to the optimal mapping, but the search to the best representative scenario subset.

Our experiments showed that for our multi-application workloads this investment in training a representative subset of scenarios pays off. Using stochastic applications, both multi-application workloads with a low and high dynamism (i.e., the difference between the values of the non-functional requirements when the different workloads are executed on the system) were used to compare the scenario-based DSE with a traditional DSE. By using a stochastic workload a wide range of application workloads could be tested. In the results, scenario-based DSE outperformed both a
traditional exhaustive DSE where each mapping is evaluated for all the application
scenarios and a traditional DSE with a statically selected scenario subset to predict
the fitness of the individual mappings. As could be expected, scenario-based DSE
was much faster than an exhaustive DSE. On top of that, the quality of the res-
ulting Pareto front of the scenario-based DSE was better or equal to the quality of
the Pareto fronts of the DSE with a statically selected scenario subset. With a low
diversity in the multi-application workload, the result of scenario-based DSE was
comparable, but over different runs the results of scenario-based DSE where much
more stable than the statically selected scenario subset. Still, with a low diversity
in the multi-application workload a random scenario subset may, by coincidence, be
of a relatively high quality. Hence, the outcome of the traditional DSE may also be
of relatively good, but this highly depends on the subset that initially selected. The
scenario database is more stable, as the subset selector continuously tries to improve
the subset. With an higher diversity in the multi-application workload, however, it
becomes harder to select a representative subset of scenarios and, as a result, the
scenario-based DSE starts to deliver better Pareto fronts than the traditional DSE.

Nevertheless, we should comment about the selection of the representative subset
of scenarios. As was illustrated by Figure 8.1, scenario-based DSE needs to divide its
computational resources between the design explorer and the subset selector. The
gain of a better representative subset of scenarios is that a design explorer is able
to provide better fitness predictions and, thus, a potentially lower search time for
the design explorer. Important, however, is that the investment in computational
resources for the subset selector does not exceed the gain in computational resources
at the design explorer. The opposite is the case for the size of the representative
subset of scenarios. Generally, a larger representative subset of scenarios is easier to
find. Hence, the computation time required at the subset selector decreases. A large
representative subset of scenarios, however, leads to a larger evaluation time at the
design explorer.

As the representative subset of scenarios is of significant importance to the
scenario-based DSE, Chapter 5 takes a more detailed look on the subset selector.
As a first step, the representativeness of a scenario subset must be defined. For this
purpose, two metrics are used: misclassification ratio and the number of misclassi-
fied relations with respect to the mappings in the trainer. The misclassification ratio
determines the capability of a scenario subset to correctly predict the Pareto rank.
In case two subsets have the same misclassification rate, the number of misclassified
Pareto dominance relations is used. These metrics, however, require that the map-
pings in the trainer are exhaustively evaluated. Although it is infeasible to evaluate
all the mappings in the design explorer, it is feasible to exhaustively evaluate the
small amount of mappings that are present in the trainer.

Furthermore, Chapter 5 also introduces three different selection techniques for
representative subset of scenarios: 1) a genetic algorithm, 2) a feature selection
algorithm and 3) a hybrid method combining the two aforementioned approaches.
Experiments show that the hybrid method is capable of exploiting the benefits of the two other approaches. In the genetic algorithm the design space of potential scenario subsets is quickly explored, whereas in the features selection algorithm a more systematical exploration of the local neighborhood of a scenario subset is performed. Especially for larger scenario subsets, the hybrid method can first quickly prune the design space using the genetic algorithm. This pruning delivers the first decent scenario subsets must faster than the feature selection algorithm that has a relatively slow convergence. A genetic algorithm, however, is much less effective to explore the local neighborhoods of the scenario subsets. In this case the feature selection is applied.

Finally, the sensitivity of the scenario-based DSE to the size of the scenario subset is investigated. It turns out that there is an accuracy / overhead trade-off. A larger representative subset of scenarios results in a longer evaluation time for the design explorer. Still, a large subset of scenarios is also potentially more accurate. Our experiments showed that there is an accuracy threshold with respect to the subset size. If a smaller scenario subset size is chosen, the outcome of the scenario-based DSE will be affected by the inaccurate fitness predictions. When a scenario subset is larger than the accuracy threshold, the outcome of the scenario-based DSE will not be affected. The only aspect that is affected is the convergence time of the scenario-based DSE. Therefore, the scenario-based DSE is not highly dependent on the subset size. As long as it exceeds the accuracy threshold, the outcome of the scenario-based DSE will eventually converge to an optimal set of mappings for a dynamic multi-application workload.

8.2 Architecture Scenarios

Another source of dynamism in embedded systems that is discussed in this thesis are the transient faults that may occur in unreliable MPSoC architectures. One of the sources of these transient faults is a single upset event that is caused by cosmic rays. It used to be the case that transient faults were only an issue in embedded application that were used in space, but with the decreasing technology scale it also becomes an issue at ground level. Due to these transient faults, the outcome of the computation from the processors can be corrupted. This may lead to incorrect outcomes of applications that are running on these processors. To model these unreliable architectures, architecture scenarios are used. Basically, architecture scenarios provide a sequence of transient faults that occur on the architecture with a given time and place. As transient faults are independent and infrequent, a Poisson distribution is used to model the probability of an architecture scenario.

As a first step towards a fault-tolerant DSE, Chapter 6 presents the Sesame Automated Fault-tolerant Explorer (SAFE). SAFE is an extension of Sesame that facilitates the fault-tolerant design of embedded systems. Completely in line with the separation of concerns within Sesame, SAFE has an additional pattern layer
that describes the automatic transformation of a normal application into a fault-tolerant application. To this purpose, the pattern layer consists of multiple fault tolerance patterns that define how an application is transformed into a fault-tolerant application and which policies are used for fault detection and handling. One of the examples of a type of fault tolerance pattern is active redundancy where multiple replicas (two for DMR and three for TMR) are used to run the application. All the outgoing data of the application is verified by comparing the outputs of the different replicas using a voter. In this way, corrupt data is detected as long as the majority of the replicas have the correct data. Depending on the policy, the current frame of the application must be dropped or restarted in the absence of a majority.

Based on the available fault tolerance patterns, a fault-tolerant mapping can be defined. Fault-tolerant mappings perform three different steps: 1) patternization, 2) binding and 3) dispatch. Firstly, the patternization segregates the applications into different subnetworks. Each of the subnetworks is made fault tolerant by selecting a fault tolerance pattern. By applying this fault tolerance pattern, all the applications can be transformed from a normal application into a fault-tolerant application. Secondly, the fault-tolerant application is bound onto the architecture. Per subnetwork, the binding selects the architectural resources that are required to all the replicas of the subnetworks and the additional processes to verify the computation (like the majority voter for the active redundancy). Finally, the dispatch generates the routing for the additional communication that is required to implement the fault tolerance patterns.

Using the architecture scenarios, SAFE can simulate a fault-tolerant mapping and obtain the non-functional properties of a fault-tolerant embedded system. During the simulation, the transient faults that are encoded in the architecture scenario are injected using software initiated fault injection (SWIFI). The detection and handling of these faults is completely modeled up to the restart of frames and the procedure to make explicit checkpoints. As a result, the obtained mapping fitness completely takes the fault tolerance patterns into account. On top of that, additional metrics like frame drop ratio can be introduced to make reliability a first class citizen of the DSE. This exploration of the fault-tolerant mapping is important as our experiments in Section 6.5 show that optimal type of fault tolerance pattern is completely application dependent. On top of that, the fault tolerance patterns have a non-trivial effect on system objectives like power, performance and frame drop ratio. Therefore, it is necessary to already incorporate the reliability during the early design space exploration.

Chapter 7 shows a first implementation of a fault-tolerant DSE framework. It provides an efficient early DSE of optimal fault-tolerant mappings taking into account the wide range of potential architecture scenarios. Accordingly, the scenario-based DSE framework that is shown in Figure 8.1 is perfectly applicable to the fault-tolerant DSE: the set of architecture scenarios is too large to straightforwardly evaluate all the mappings exhaustively. To discriminate between the scenario-based DSE for a
dynamic multi-application workload and the fault-tolerant DSE, the design explorer of the fault-tolerant DSE is called FMapping DSE and the subset selector is called subset DSE. As expected, the subset DSE searches for a representative subset of architecture scenarios and the FMMapping DSE uses this subset to predict the fitness of the evaluated fault-tolerant mappings.

In the fault-tolerant DSE reliability classes are introduced to be able to analyze how the optimal fault-tolerant mappings behave under a different number of transient faults in the architecture. The benefit of combining the fitness of different reliability classes is shown using two case studies. Among other things, the fault sensitivity of the fault-tolerant mappings can be easily observed.

8.3 Future Work

There are many potential directions for future research based on the contributions in this thesis. At first, the contributions of this thesis purely exploit scenarios in a static fashion. Given the application or architecture scenarios in the system, the best mapping was identified that was optimized for the complete set of potential scenarios. A static mapping is used during the complete lifetime of the embedded system. Scenarios, however, can be exploited to improve the performance of the system by switching mappings during runtime based on the current scenario. In case of application scenarios, for example, starting an application may result in the activation of an additional processor or the increase of the frequency on a specific processor. Similarly, an architecture scenario can be used to optimize the performance by migrating processes from a faulty processor to another processor. The first steps for this future work are already taken in [59, 60].

To go from static mappings to dynamic mappings that can be changed during runtime, the embedded system must be able to detect scenarios. Currently, the scenarios as described in this thesis cannot easily be detected except for the inter-application scenarios that describe which applications are active simultaneously. For architecture scenarios, the closest architecture scenario can easily be defined by a distance metric and, potentially, this can also be done for intra-application scenarios. In this way, the next scenario can be predicted based on the current scenario. Next, the scenario-based DSE must be extended such that the DSE not comes up with a single optimal static mapping, but a set of mappings that is as a basis for the dynamic switch of mappings based on the scenario. This could be analogue to the reliability class of Chapter 7. Just as a fixed number of reliability classes, a fixed number of scenario clusters can be defined for which a mapping is identified. This means, however, that there must also be a technique to cluster mappings.

Apart from researching the possibilities for dynamic mappings, the work on the fault-tolerant DSE needs to be extended. Currently, the work in Chapter 7 is quite preliminary. Several steps need to be taken to deepen the research on the fault-tolerant DSE. At first, the fault-tolerant DSE should be extended to incorporate
permanent faults. This should be done alongside the dynamic mapping, as a permanent fault on an architectural resource should result in a remapping that is able to efficiently map the multi-application workload without using the defective processor. More importantly, the subset DSE of the fault-tolerant DSE should be fully analyzed. Similarly to the approach where the selection of the representative subset of application scenarios was analyzed in Chapter 5, the selection of architecture scenarios must be looked into. In the case study of Section 7.4.1 there was already emphasized that it is important that the subset of architecture scenarios is representative. However, in contrast to the subset of application scenarios, it is infeasible to exhaustively evaluate a mapping for all architecture scenarios. This complicates the subset DSE as the real fitness of a training mapping can only be approximated. Hence, other subset selection techniques may be work more efficient than the genetic algorithm that is currently used. Additionally, more extensive experiments must be done in order to study the selection of subset of architecture scenarios. Important to know, for example, is the optimal size of the representative subset. Another aspect is if there is a minimal size of the subset of architecture scenarios that is required for accurate results of the fault-tolerant DSE.