SYSTEM LEVEL SYNTHESIS FLOW FOR SELF-ADAPTIVE MULTI-MODE RECONFIGURABLE SYSTEMS

Stefan Wildermann, Felix Reimann, Daniel Ziener, Jürgen Teich

University of Erlangen-Nuremberg, Germany
{stefan.wildermann, felix.reimann, daniel.ziener, juergen.teich}@cs.fau.de

ABSTRACT

This paper presents a synthesis flow to design self-adaptive multi-mode reconfigurable systems on the system level. Such systems are able to react on environmental changes by switching operational modes through hardware reconfiguration. Thus, they can provide context-aware processing while efficiently utilizing the (constrained and restricted) system resources.

1. INTRODUCTION

Embedded systems of any kind should have low cost, be small and power efficient. This implies design constraints (regarding these objectives but also requirements like real time capabilities) and limited capacity for providing functionality on the one hand. On the other hand, many embedded systems, e.g., embedded smart cameras, are operating in unknown, highly dynamic, and often unpredictable real world environments so that a variety of complex algorithms is required for a robust operation of the system. Due to constraints, only a subset of these algorithms may be used concurrently in a configuration of the system. As a solution for this tradeoff, context-aware and resource-aware adaptation by re-organizing the configuration of algorithms at runtime can lead to a better utilization of the system resources in the presence of constraints and restrictions while retaining and possibly even optimizing the processing quality of the system. Here, reconfigurable hardware is a solution to further increase the flexibility of the system despite these constraints by offering the capability of sharing hardware resources between different configurations mutually exclusive.

2. SELF-ADAPTIVE MULTI-MODE SYSTEMS

In a more abstract view of this system model, the set $\mathcal{G} = \{G_i \mid i = 1, ..., n\}$ denotes all n algorithms which are provided by the designer. During run-time, different combinations of these algorithms can be executed on the available architecture, each representing an *operational mode O* of the system. Ideally, each possible combination of algorithms could be run in the system. However, due to aforementioned constraints, only a small subset of configuration may actually constitute feasible operation modes, and only such modes are allowed to be executed.

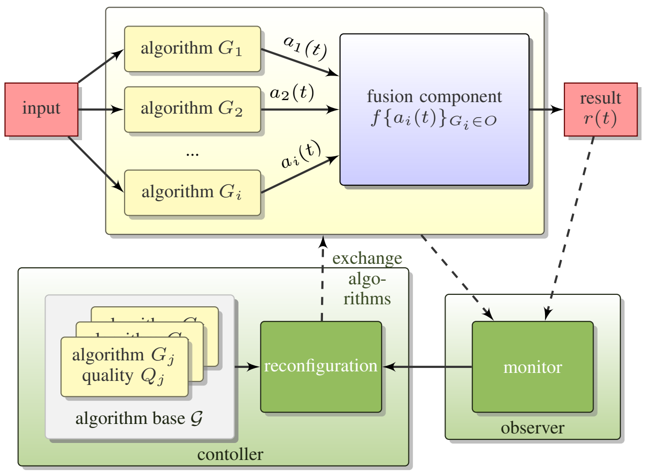


Fig. 1. Self-adaptive system architecture fusing the results of multiple algorithms and adapting this configuration when the current algorithms do not work efficiently.

Therefore, the idea is to additionally equip the multi-mode system with an autonomous *Control Mechanism (CM)* which is able to observe and control the system [1]. The purpose of the CM is to detect environmental changes and degeneration of the system's processing quality (*observe*). It can then react by modifying the system configuration through a transition to a new operational mode (*control*). The system architecture as illustrated in Fig. 1 is described next.

2.1. Self-adaptive System Architecture

The operational mode of the system at instant of time t is represented by the set of active algorithms $O(t) \subseteq \mathcal{G}$, where each algorithm $G_i \in O(t)$ calculates a result $a_i(t)$. These results are fused by a fusion function f to produce the result $r(t)$ of the overall system at time t:

$$r(t) = f\{a_i(t)\}_{G_i \in O} \tag{1}$$

The *observer* evaluates the quality of each active algorithm $G_i \in O(t)$ by an adequate quality function $\tilde{q}(a_i(t), r(t))$, which measures how good a filter is predicting the result $r(t)$:

$$\tilde{q}(a_i(t), r(t)). \tag{2}$$

The *normalized qualities* q_i are then given as

$$q_i(t) = \frac{\tilde{q}(a_i(t), r(t))}{\sum\limits_{G_j \in O} \tilde{q}(a_j(t), r(t))} \tag{3}$$

so that the sum of all qualities sum up to 1. The number N_{eff} of algorithms which are now efficiently contributing to the system output can be calculated based on these qualities as:

$$N_{eff} = \frac{1}{\sum\limits_{G_i \in O(t)} \left(q_i(t)\right)^2} \tag{4}$$

Furthermore, a long term estimate Q_i of the algorithms qualities is generated according to

$$Q_i = (1 - \lambda) \cdot Q_i + \lambda \cdot q_i(t). \tag{5}$$

The *controller* takes these results to test whether a reconfiguration of the system is necessary. If so, the new system configuration $O(t+1)$ has to be determined. This decision is based on a *fitness value* $Z(O)$ used for each mode O. It is calculated as the multiplied qualities of its algorithms according to

$$Z(O) = \prod_{G_i \in O \cap O(t)} q_i(t) \cdot \prod_{G_i \in O \setminus O(t)} Q_i. \tag{6}$$

The fitness value uses the actual qualities of all algorithms which are part of the current mode $O(t)$, and the estimated qualities of inactive algorithms.

Algorithm 1 outlines the decision process for reconfiguration. Adaptation is possible at the earliest θ_{mod} time steps after having performed the previous modification at time step t_{mod} (line 1). This is required to give the system some time to evaluate the quality of the new algorithms in the current context. No modification is necessary if the system efficiently manages to track an object. Therefore, adaptation is only performed if the efficiency N_{eff} is below a predefined threshold (line 2). Note that threshold $\theta_{eff}(O)$ may depend on the configuration since configurations may contain different numbers of filters. For example, in case at least 75% of the active filters should contribute to the result, the threshold would be defined as $\theta_{eff}(O(t)) = 0.75 \cdot |O(t)|$.

Now, one of two behaviors is performed:

- Exploitation: With a probability of $p_{exploit}$, the controller selects that **feasible** mode $O(t+1)$ which has the maximal fitness value.
- Exploration: With a probability of $(1 - p_{exploit})$, the controller selects a **feasible** mode $O(t+1)$ randomly with probabilities proportional to their fitness values.

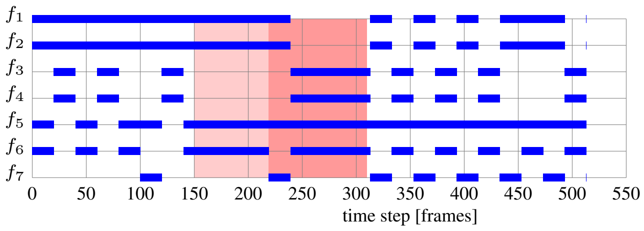
2.2. Smart Camera Case Study

A smart camera application from [2] serves as a case study to illustrate the behavior of systems implemented according to above system architecture. It performs person tracking based on the image processing filters illustrated in Figure 2. The system executes a subset of these filters on the same input image and fuses their results via a tracking algorithm (cf. [2]). The tracking result is used to calculate the filter qualities, indicating how good each filter has predicted this result. The qualities are used to perform the adaptation as described above.

Fig. 3 illustrates the system behavior for a image test sequence. The Gantt chart illustrates the time intervals when each filter is active. In this test sequence, no person is visible between frames 0 and 150, and the system is arbitrarily switching configurations after every $\theta_{mod} = 20$ time steps (frames) according to Algorithm 1. The person appears in the scene around frame 150, and the system is successfully tracking the person with color-based filters and edge-based filters being loaded. A color corruption happens at frame 219 where the input image is switched to gray scale. As the two color-based filters are unable to produce an output, N_{eff} falls below the threshold. The system adapts until the color filters are removed from the system and replaced by the more adequate motion-based filters. When the person leaves at frame 310, all filters fail and the system switches between configurations with after each θ_{mod} time steps.

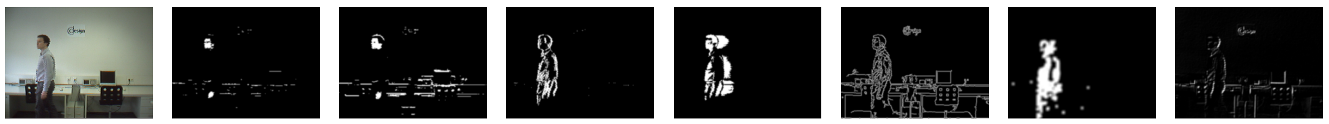
Algorithm 1: Control mechanism for reconfiguration decision, which exchanges algorithms if necessary, either by a behavior performing exploitation or exploration.

```
1 if $t - t_{mod} > \theta_{mod}$ then
2     if $N_{eff} < \theta_{eff}(O(t))$ then
3         generate random number $rnd \in [0, 1]$;
4         if $rnd \le p_{exploit}$ then
5             doExploitation();
6         else
7             doExploration();
8         $t_{mod} = t$;
```



(a) Gantt chart of system setup

Fig. 3. Gantt chart for a test sequence with color corruption. The person is visible in the highlighted interval and color corruption happens in the interval with darker color from frame 219.



(a) input (b) skin color in RGB (f_1) (c) skin color in YCbCr (f_2) (d) motion detection (f_3) (e) background subtraction (f_4) (f) Canny edge detection (f_5) (g) edge background (f_6) (h) Sobel edge detection (f_7)

Fig. 2. Examples of the filters used in the smart camera case study for person tracking. Filters f_1 and f_2 are *color*-based, filters f_3 and f_4 are *motion*-based, and filters f_5, f_6, and f_7 are *edge*-based.

2.3. Design Challenges

For performing the system adaptation, it is necessary to determine the feasible modes of a system. However, system synthesis in the presence of stringent design constraints (restricted bandwidth, reconfigurable hardware, processor utilization, etc.) is known to be NP-complete (cf. [3]). We therefore propose a system level design methodology since it would be too costly or even infeasible to select, verify, and optimize each configuration at run-time.

Furthermore, design constraints limit the amount of combinations of algorithms that can be implemented as feasible modes of the systems. Therefore, resource sharing becomes a key concept to increase the number of feasible operational modes: Even if not all algorithms are able to be executed concurrently, subsets of algorithms can be executed on the same resources as mutually exclusive operational modes. Of course, sharing of computational resources can be achieved by providing a schedule for each mode independently. However, through the use of reconfigurable hardware, it is also possible to share hardware resources between modes. This allows the cost and size to be decreased, while increasing the resource utilization of the system. In this work, Field Programmable Gate Array (FPGA) technology is the implementation target.

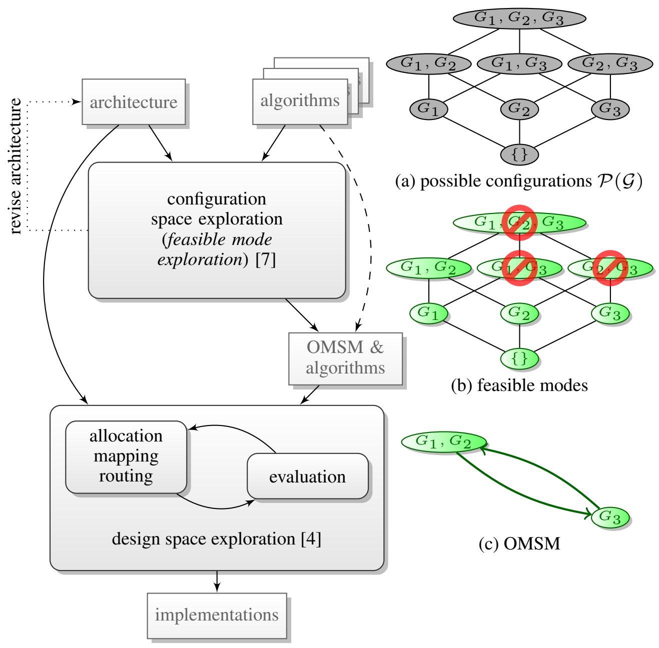
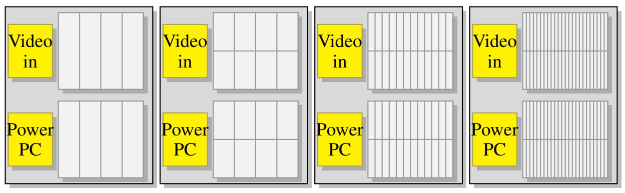


Fig. 4. System level synthesis flow. The first CSE phase identifies the feasible modes for a given specification. In the second DSE phase, the multi-mode reconfigurable system is synthesized through iterative optimization.

3. DESIGN FLOW

The methodology illustrated in Fig. 4 contains two mandatory design phases: The first one is *configuration space exploration* (CSE). For a given specification, the power set of $\mathcal{G}$ constitutes the possible configurations (see Fig. 4(a) for an example with three algorithms). The purpose of CSE is to evaluate which of these configurations can be executed on a given reconfigurable architecture layout as feasible modes despite the constraints (see, e.g., Fig. 4(b)). We define an exploration model in [4] that captures the behavioral aspects of self-adaptive systems and the spatial and technological aspects of FPGA-based reconfigurable system-on-chip architectures. We provide models in [5] for island style reconfiguration as well as 2-dimensional module placement according to [6]. For performing CSE, a symbolic encoding of this model is formulated that specifies the restrictions and design constraints as a *Pseudo Boolean (PB) Satisfiability problem.* By applying PB solvers, this encoding can be tested for satisfiability. We then apply an algorithm called *Feasible Mode Exploration algorithm* [7] that provides a scheme to efficiently apply this test for the exploration of the configuration space. This phase can be performed repeatedly to evaluate and compare different architecture layouts and templates, as indicated in Fig. 4. Based on the result of CSE, the designer can then select the configurations of the system as well as possible transitions between them. This is expressed by an *Operational Mode State Machine (OMSM)* [8], as illustrated in Fig. 4. The OMSM specifies how the CM can then switch between modes at run-time.

The second design phase is *Design Space Exploration (DSE).* DSE is a multi-objective optimization problem with conflicting objectives like cost, area, power consumption, and reconfiguration time. Our DSE approach [4] applies the exploration model to derive several different implementations by (a) allocating resources from the architecture template and remove those not required, which allows the size of the architecture to be further reduced, (b) mapping tasks onto the allocated resources (which also includes the 2-dimensional placement of hardware modules), and (c) routing the communication between tasks. Each implementation is



(a) **A1** (b) **A2** (c) **A3** (d) **A4**

Fig. 5. Schematic outline of architecture options **A1** to **A4**.

evaluated regarding the objectives and iteratively further refined. The result of this phase is a set of non-dominated implementations regarding the specified objectives.

4. EXPERIMENTS

We have implemented and tested our synthesis flow using the publicly available PB solver *Sat4j* [9] and the publicly available optimization framework OPT4J [10]. The flow is applied to implement a self-adaptive reconfigurable smart camera according to the case study from Section 2.2 using image filter algorithms f_1 to f_6 from Fig. 2. The target architecture is a reconfigurable system-on-chip according to [6]. It contains two partially reconfigurable regions (PR regions) for 2-dimensional module placement, as well as a CPU-subsystem using a PowerPC. Four architecture alternatives are generated, the PR regions were divided in different granularities of 4×1 tiles (**A1**), 4×2 tiles (**A2**), 14×2 tiles (**A3**), and 28×2 tiles (**A4**) as illustrated in Figure 5. Partial modules can be placed by occupying one or several contiguous of such tiles.

As the application provides six image filter algorithms, a total of $64 = 2^6$ combinations are possible. The results of CSE for the four architecture alternatives are presented in Table 1. The results show that the finer the tiling, the more efficient resource sharing can be performed, resulting in more feasible modes. Moreover, the execution time can drastically vary depending on the complexity of the exploration model as well as the number of infeasible modes[1]. It shows that such a test could hardly be performed online.

Fig.6 shows the results of performing DSE for the smart camera case study with three operational modes. The optimization objectives are to minimize 1.) the average reconfiguration time, 2.) the power consumption, and 3.) cost of the used resources. Since this is a multi-objective optimization problem, we have chosen the ϵ-dominance where low values $\epsilon \in [0, 1]$ indicate results with higher quality. The symbolic DSE (*dse*) is compared to a state-of-the-art DSE based on an Evolutionary algorithm [8] (*moea*). The results show that the proposed symbolic approach converges much faster to the optimum, while *moea* is trapped in a local optimum.

[1]The latter being the case for **A2**. Testing infeasible modes normally takes longer than testing feasible modes

Table 1. Results of CSE.

architecture	# of feasible modes	avg. execution time [min]
A1	16	0.28
A2	57	204.06
A3	62	3.47
A4	63	12.42

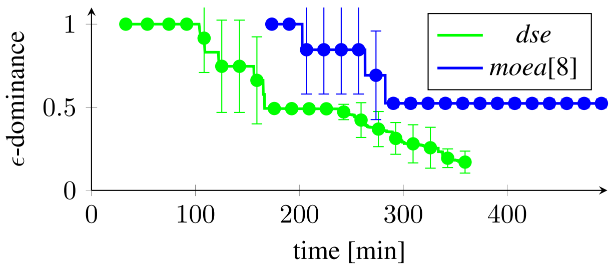


Fig. 6. Result of DSE. Average ϵ-dominance results for test case over the optimization time.

5. CONCLUSION

This paper proposes a system level design methodology for self-adaptive systems which switch between algorithmic configurations to maintain the system efficiency. The analysis, verification, and optimization of such systems at design time, as proposed by our methodology, is mandatory to guarantee feasible and highly optimized implementations.

6. REFERENCES

[1] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter, "Adaptivity and self-organisation in organic computing systems," in *Organic Computing – A Paradigm Shift for Complex Systems*, ser. Autonomic Systems. Springer Basel, 2011, vol. 1, pp. 5–37.

[2] S. Wildermann, A. Oetken, J. Teich, and Z. Salcic, "Self-organizing computer vision for robust object tracking in smart cameras," in *Proc. of ATC*, ser. LNCS. Springer-Verlag, 2010, pp. 1–16.

[3] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," *Design Automation for Embedded Systems*, vol. 3, pp. 23–58, 1998.

[4] S. Wildermann, F. Reimann, D. Ziener, and J. Teich, "Symbolic design space exploration for multi-mode reconfigurable systems," in *Proc. of CODES+ISSS*, 2011, pp. 129–138.

[5] S. Wildermann, J. Teich, and D. Ziener, "Unifying partitioning and placement for SAT-based exploration of heterogeneous reconfigurable SoCs," in *Proc. of FPL*, 2011, pp. 429–434.

[6] A. Oetken, S. Wildermann, J. Teich, and D. Koch, "A bus-based SoC architecture for flexible module placement on reconfigurable FPGAs," in *Proc. of FPL*, 2010, pp. 234–239.

[7] S. Wildermann, F. Reimann, J. Teich, and Z. Salcic, "Operational mode exploration for reconfigurable systems with multiple applications," in *Proc. of FPT*, 2011, pp. 1–8.

[8] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Cosynthesis of energy-efficient multimode embedded systems with consideration of mode-execution probabilities," *TCAD*, vol. 24, no. 2, pp. 153–169, 2005.

[9] D. Le Berre and A. Parrain, "The SAT4J library, release 2.2, system description," *JSAT*, vol. 7, pp. 59–64, 2010.

[10] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J – a modular framework for meta-heuristic optimization," in *Proc. of GECCO*, 2011, pp. 1723–1730.