A New Concept for System-Level Design of Runtime Reconfigurable Real-Time Systems

Arno Luppold, Benjamin Menhorn, Heiko Falk and Frank Slomka
Institute for Embedded Systems/Real-Time Systems
Ulm University
{arno.luppold|benjamin.menhorn|heiko.falk|frank.slomka}@uni-ulm.de

***Abstract*—This concept paper proposes a new system-level design methodology for runtime reconfigurable adaptive heterogeneous systems in a real-time environment. Today, among those approaches dealing with runtime reconfiguration and hardware/software co-design, compliance with hard real-time conditions is not guaranteed. Our approach will fill this gap. In contrast to other approaches, we apply methods of real-time analysis to embedded reconfigurable systems. An extended compiler and a runtime resource manager guarantee both synthesis and reconfiguration in a (hard) real-time environment. With this approach, the system can adapt to changes in requirements and operational environments during runtime.**

I. Introduction

Modern embedded systems have to execute multiple tasks while often meeting hard real-time constraints and coping with a restricted chip size and power budget. To efficiently fulfill their tasks, such heterogeneous systems often consist of both software tasks and dedicated programmable hardware logic. The Ptolemy project was one of the first projects, which uncovered the difficulty of an early design partitioning between hardware and software [1]. In most cases, the decision whether a given function should be realized in software or hardware is highly dependent on a system's other tasks.

An additional challenge occurs with changes during runtime. When considering safety-critical infrastructure, shutting down a system may not be feasible. Apart from safety-critical systems, shutting down industrial production systems for maintenance work will often result in substantial financial losses. This leads to the need for run-time reconfigurable systems. However, when functions are added, resources are changed, bugs fixed, or if the environmental conditions change, a system's timing behavior will vary. In this case, finding an efficient and effective hardware/software partitioning on the fly becomes even more complex. But even then it has to be ensured, that operational constraints are not violated. These considerations lead to the idea of a holistic development process on the system level. The scope of our work covers hardware/software co-design of embedded systems and their runtime reconfigurability. We will accomplish two goals: first, we will guarantee specific requirements during runtime even after the system was reconfigured. In our case, we focus on hard real-time systems. Second, in contrast to most other approaches, which base their decisions for hardware/software partitioning on simulations, we rely on analytical methods to guarantee compliance with all requirements. This way, we can guarantee safe upper bounds for all response times, while simulation delivers only unsafe estimates [2].

This work was partially supported by Deutsche Forschungsgesellschaft (DFG) under grant FA 1017/1-1.

The key contributions of this paper are:

- We propose a new concept for a methodology to design heterogeneous embedded real-time systems on system level.
- We show how we will ensure that hard real-time constraints are met even in runtime reconfigurable systems.
- We will base our work on analysis rather than simulations.
- We propose a future implementation of our methodology.

This paper is organized as follows: Section II gives a brief overview of related projects in the field of reconfigurable platforms and hardware/software co-design. Section III shows our underlying approach. With this approach, Section IV describes our concept for a holistic methodology. This paper closes with a conclusion and future challenges.

II. Related Work

The design of runtime reconfigurable embedded systems has been addressed by several projects. In [3], the authors propose a synthesis process in which the system's hardware capabilities are expressed in a global resource library. A system's functions are modeled by using acyclic task graphs. To each graph a set of attributes is assigned. These sets contain properties such as a task's worst-case execution time (WCET), deadline and period. Based on a heuristical optimization algorithm, task graphs which are not executed at the same time are then pairwise merged to common hardware resources. After a merge, the system's real-time deadlines are verified, and the optimization flow is terminated when deadlines cannot be hold.

In [4], the authors propose an object-oriented method for hardware/software co-design on reconfigurable platforms. A so-called *primitive object* can be specified. It can either be realized by hardware or software. Systems are subsequently modeled as composite objects, containing hardware and software primitive objects. Objects may then communicate with each other using various communication mechanisms.

The ERA project [5] focuses on developing a platform consisting of reconfigurable very large instruction word processors on field programmable gate arrays (FPGA). Those processors are connected through a network on chip to memory subsystems. The number of processors as well as their shared resources may be instantiated during compilation time. They may also be configured at runtime by both software and hardware schedulers. Software reconfiguration can be initiated by the programmer using a given application programming interface (API). Alternatively, a platform's operating system can also automatically reconfigure its system using information like the platform's workload or temperature.

The ANDRES project [6] provides a SystemC framework to create an adaptive heterogeneous system consisting of software and digital hardware components. ANDRES uses the OSSS+R SystemC library [7] to divide a system's hardware in a static and a reconfigurable area. The reconfigurable area is modeled as a polymorphic object. This can be compared to polymorphism in object-oriented software. This allows to switch between different implementations during runtime when maintaining a fixed interface. Based on the SystemC description, the ANDRES workflow automatically generates a hardware VHDL description and embedded code in C++. A set of hierarchically organized controllers is then used on the embedded platforms to reconfigure the system at runtime. Based on the ANDRES project, COMPLEX is one of the most recent projects [8]. It develops a framework for a platform-based design space exploration to find an optimum between system performance and power consumption. Its holistic approach uses UML and SystemC to design hardware and software at an algorithmic level. Resulting systems are then simulated in a cycle-accurate simulator to determine an optimal hardware/software partitioning for each use case. At runtime, a global resource manager can switch between these configurations at dedicated switching points [9].

The iLAND project realizes a resource manager for reconfigurable networked embedded systems [10]. Real-time guarantees are established by decoupling all distributed elements. Although focussing on soft real-time purposes, iLAND systems can be runtime-reconfigured in bounded time. However, iLAND does not focus on hardware/software partitioning but uses general purpose CPUs and mainly focusses on problems arising from the system's distribution over a network [11].

[12] describes the Java based language LIME. LIME programs are either compiled completely to java bytecode, or can be translated into OpenCL for GPUs or VHDL for FPGAs as target systems. OpenCL or FPGA sources are then compiled using the target's native tool chain. A runtime manager implemented in Java can move application parts from software to the FPGA or GPU at runtime. Due to the Java approach, the runtime manager's real-time behaviour cannot be determined. Additionally, the LIME approach does not integrate any real-time oriented optimization techniques. Instead, the programmer has to manually adapt his LIME code until the resulting output serves his needs.

Although some of the related projects consider real-time systems, none of them uses real-time analysis to decide on a system's partitioning. Instead, all partitioning approaches rely on simulations or heuristics, leading to an unsafe real-time behaviour. Although not focusing on reconfigurable systems, Wandeler [13] shows a possibility to efficiently analyze complex real-time systems. However, Wandeler does not consider any secondary objectives like energy consumption and provides no methods to analyze capabilities and requirements of hardware logic. Based on his findings, our approach will use analysis for all its decisions regarding hardware/software partitioning. This way, we can guarantee that a system's partitioning will be optimized towards its worst-case behavior both at the development stage and at runtime.

III. Underlying approach

Most existing research projects use SystemC [14], a C++ class library, for modeling and simulating designs at system level. A set of possible operational scenarios is applied to a SystemC model, and its behavior is obtained by running SystemC simulations. SystemC code can then be transferred to C/C++ code or to a hardware description language (HDL) like VHDL or Verilog. Finally, the resulting code can be compiled or synthesized and each module's properties like its WCET can be calculated. These results are fed back into the SystemC model as basis for the decision which module should be implemented in which architecture. Transforming SystemC models to software source code and HDLs is usually automated by tools [15], [16].

We do not base our decision for hardware/software partitioning on simulations or heuristic algorithms but aim at a precise analysis with regards to minimizing modules' WCET. Therefore, we do not need the simulation capabilities of SystemC. Instead, we will base our design flow on the relatively new standard OpenCL [17]. OpenCL provides an open standard for programming on any heterogeneous multi-threaded platform. Companies like Altera provide OpenCL support for their FPGAs [18], NVIDIA for some of their graphics chips [19], Intel and AMD for some of their CPUs [20], [21]. In [22], OpenCL is used to develop a design methodology for application-specific processors.

In OpenCL, systems are described as a set of computing units called *kernels*, similar to modules in SystemC. Each kernel is a stand-alone unit, allowing full task and data parallelism. Kernels can communicate with each other, e.g. by using shared memory, and can be synchronized by using synchronization points. Kernels are managed by a resource manager called *host*. Hosts may run on external systems or on their target systems. During runtime, they can freely determine which and how many instances of a kernel are created or destroyed, and on which physical resources they run. Also, hosts are allowed to locally compile and execute new kernels at runtime. Hosts themselves are developed as individual programs, communicating with kernels by the OpenCL API. Therefore, the compiler may define an initial mapping of kernels to hardware resources, or leave it as a runtime decision for the host. Compared to SystemC, OpenCL natively distinguishes between kernels used for computation and supervising hosts. This makes OpenCL superior when directly designing runtime reconfigurable systems. In consideration of the previous discussed, we will build up our approach on hosts and kernels from OpenCL.

IV. Methodology

Our approach bases on two main approaches: An extended compiler and a runtime resource manager (RRM). In the following, both approaches will be further explained.

A. Extended compiler

One essential part of our proposed methodology is our extended compiler. Its main tasks are the clustering into modules, the generation of executable code and the evaluation of the modules. We have illustrated our approach in Figure 1. The input for our compiler is a given system description in OpenCL. The main steps of our extended compiler are:

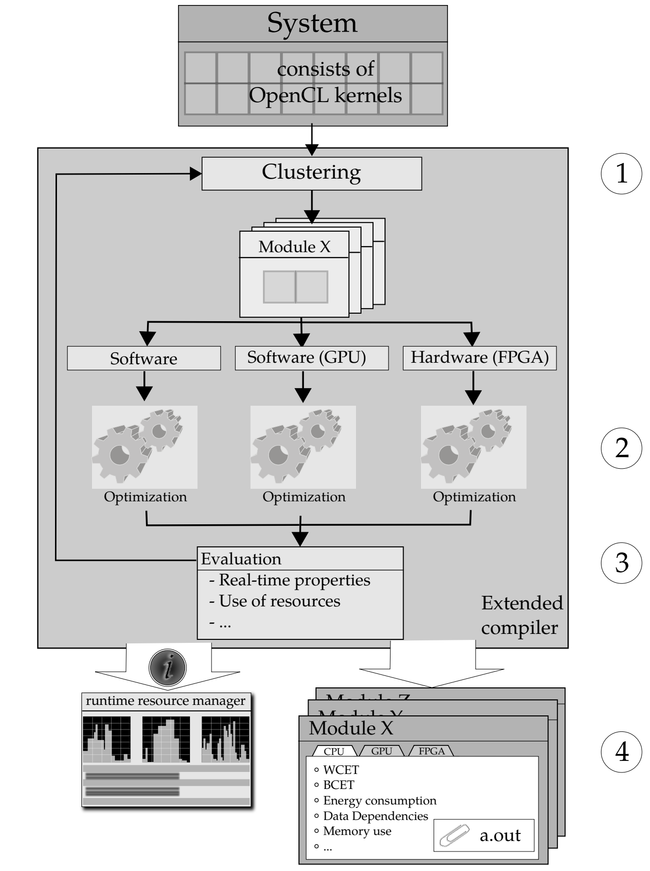

Fig. 1. Extended Compiler

① The System is realized in OpenCL. It consists of single kernels. In the first step, our compiler decides how those kernels will be clustered. Each module can then consist of one or more kernels and can run on different resources such as a CPU, FPGA or GPU.

② Each module is compiled and optimized for various target resources such as general purpose processors, graphics processors and FPGAs. Here, the respective module also undergoes an optimization. The optimization may target on energy savings, reducing chip area or cost reduction. We mainly want to apply methods of runtime improvement as our target is the compliance with real-time bounds. In doing so, our main focus does not lie on the development of these optimizations. Instead, we will make use of existing previous work here. The modular structure of the compiler allows to integrate real-time oriented optimizations for software on CPUs, GPUs and programmable hardware on FPGAs from other research groups and fields. In the field of software optimization, for example, it would be possible to use techniques from the WCET aware compiler WCC [23], which uses the proprietary WCET analyzer aiT [24].

③ In the evaluation part, each compiled and optimized module is analyzed for its properties. To correctly analyze response times of systems' functions and to guarantee all real-time constraints, it will be necessary to not only analyze each module's WCET, but also model dependencies like mutual exclusions or resource conflicts between all modules. However, due to the adaptive behavior of our approach, it is not possible to compute the final worst case response time of each module during the compilation. To solve this problem, we are building upon the *limiting event streams model* provided by [25]. In this model, calculating dependencies between modules is orthogonal to the actual real-time analysis. This enables us to model dependencies independently of the specific implementation and perform significant parts of the real-time analysis without the need to fully know how the final system will be realized. Evaluation may also indicate that a current clustering cannot meet certain requirements, or that a found clustering will not allow any reconfigurations during runtime. Then, the current set of modules will be rejected and a new clustering into modules will be performed. This may also be the case, if modules have certain dependencies and clustering of several OpenCL kernels to separate modules will not improve performance. Our iterative process will use information gathered from modules' optimizations to decide if clustering the system into modules in a different manner will result in a better reconfigurability or tighter WCETs.

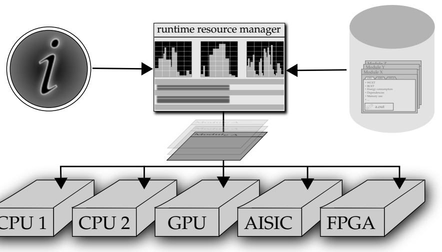

Fig. 2. Runtime resource manager

④ The extended compiler outputs two items. One is information used by the RRM, and the other is a set of modules with meta information. The information for the RRM comprises all information to allow the reconfiguration of the system during runtime. Information for the RRM includes the system's hardware capabilities like interconnect bus speed as well as runtime dependencies between different modules, such as mutually excluded execution of different modules. The second output is a set of modules with execution information. Each module contains meta information about the WCET, its best case execution time and data dependencies on other modules. It also contains information about other needed resources such as chip area, program memory or energy consumption. However, all these values differ for each single resource. Therefore, there is separate meta information for every resource. With every module also comes the binary code for a CPU/GPU implementation and netlists for hardware implementations, respectively. Those module sets are then used by the RRM to (re-)configure the system.

B. Runtime resource manager

As Figure 2 illustrates, the main task of the RRM is the distribution of the modules to the available resources. Our development process is based on a timing model and ensures that no reconfiguration will degrade the system's WCET performance. Therefore, the RRM has three sources of information: one is the information generated by the extended compiler. The second is the meta information from the set of modules. And the last one is status information from the system itself like sensor output or current workload. Most of the time, several modules will share resources, especially CPUs. Due to the number of possible combinations and the uncertainty introduced by the runtime reconfiguration, it is infeasible to determine all possible resource conflicts during compilation time. The RRM has the responsibility to account for penalties due to resource conflicts like e.g. cache misses or limited communication between the modules to provide safe response times. However, it is also impossible to calculate all these data on an embedded platform, where not only the modules themselves but also the RRM has to deal with limited resources. Our approach conducts as much of the response time analysis as possible in the previously described extended compiler. All information which can be calculated offline, e. g. the modules' WCETs, or logical dependencies between modules is calculated by the compiler. Based on these data, the compiler can partially predict the modules' behaviour regarding memory and bus accesses. At runtime, the RRM computes safe upper bounds for modules' worst case response times based on these pre-generated offline data and online data as e. g. the current workload. We will investigate efficient analyses methods like e.g. [26], [27]. Those methods will be adapted to integrate the pre-compiled offline data and to generate a fast and secure online approximation of the modules' worst case response time, instead of focussing on as tight as possible bounds. This way, the RRM can safely determine, which module should be placed on which resource.

We base our RRM on the hosts concept from OpenCL. We build upon standard hosts and will extend them by the required functionality described above which will be in accordance with the OpenCL Specification [17].

V. Conclusion and Future Challenges

We presented a new concept to fill a gap in reconfigurable embedded systems. Our concept will ensure that all real-time bounds will always be met. In this paper, we presented our concept in order to gather feedback on our approach.

The main challenge will be to provide a method for computation-efficient calculation of penalties due to resource conflicts between different modules. This part of the real-time analysis can only be carried out at runtime by the RRM which has limited computation power on embedded systems. Another challenge is to provide an efficient interconnect between host and kernels. This interconnection network must be able to guarantee real-time constraints for communication between kernel and host as well as between different kernels with data dependencies between each other.

A successful implementation of our approach will allow to design adaptive safety-critical real-time systems as runtime reconfigurable heterogeneous systems. At the same time, it will allow system engineers to concentrate on systems' functions, while leaving the concrete partitioning and optimization to our compiler throughout the development process.

References

[1] A. Kalavade and E. A. Lee, "A Hardware-Software Codesign Methodology for DSP Applications," *IEEE Des. Test*, vol. 10, no. 3, 1993, pp. 16–28.

[2] S. Samii *et al.*, "A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems." in *Proceedings of DATE*, 2008, pp. 556–561.

[3] B. Dave, "CRUSADE: hardware/software co-synthesis of dynamically reconfigurable heterogeneous real-time distributed embedded systems," in *Proceedings of DATE*, 1999, pp. 97–104.

[4] P. Green and M. Edwards, "Object oriented development method for reconfigurable embedded systems," *IEE Proceedings Computers and Digital Techniques*, vol. 147, no. 3, 2000, pp. 153–158.

[5] S. Wong *et al.*, "Early Results from ERA – Embedded Reconfigurable Architectures," in *Proceedings of INDIN*, 2011, pp. 816–822.

[6] A. Herrholz *et al.*, "The Andres Project: Analysis and Design of Run-Time Reconfigurable, Heterogeneous Systems," in *Proceedings of FPL*, 2007, pp. 396–401.

[7] A. Schallenberg, W. Nebel, and F. Oppenheimer, "OSSS+R: Modelling and Simulating Self-Reconfigurable Systems," in *Proceedings of FPL*, 2006, pp. 1–6.

[8] "COMPLEX - COdesign and power Management in PLatform-based design space EXploration." [Online] Available: https://complex.offis.de/

[9] C. Ykman-Couvreur *et al.*, "Run-time resource management based on design space exploration," in *Proceedings of IFIP*, ser. CODES+ISSS, 2012, pp. 557–566.

[10] M. García-Valls, I. Lopez, and L. Villar, "iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, feb. 2013, pp. 228–236.

[11] M. García-Valls, P. Basanta-Val, and I. Estevez-Ayres, "A Codesign-mponent Model for Homogeneous Implementation of Reconfigurable Service-Based Distributed Real-Time Applications," in *In Proceedings of NOTERE*, 2010, pp. 267–272.

[12] J. Auerbach *et al.*, "A compiler and runtime for heterogeneous computing," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 271–276. [Online] Available: http://doi.acm.org/10.1145/2228360.2228411

[13] E. Wandeler, "Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems," Ph.D. dissertation, Swiss Federal Institute of Technology Zurich, Zurich, 2006.

[14] "SystemC." [Online] Available: http://www.accellera.org/downloads/standards/systemc/

[15] F. Herrera *et al.*, "Systematic embedded software generation from SystemC," in *Proceedings of DATE*, 2003, pp. 142–147.

[16] K. Grüttner, F. Oppenheimer, and W. Nebel, "OSSS methodology - system-level design and synthesis of embedded HW/SW systems in C++," in *Proceedings of ISABEL*, 2008, pp. 1–5.

[17] A. Munshi, "The OpenCL Specification V 1.2," Khronos OpenCL Working Group, Tech. Rep., 2012.

[18] Altera Corp., "Implementing FPGA Design with the OpenCL Standard," White Paper, San Jose, CA, 2011.

[19] "NVIDIA OpenCL SDK." [Online] Available: https://developer.nvidia.com/opencl

[20] "The Intel SDK for OpenCL." [Online] Available: http://software.intel.com/en-us/vcsource/tools/opencl-sdk

[21] "Accelerated Parallel Processing (APP) SDK." [Online] Available: http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/

[22] P. Jääskeläinen *et al.*, "OpenCL-based Design Methodology for Application-Specific Processors," *Transactions on HiPEAC*, vol. 5, no. 4, 2011, pp. 1–20.

[23] H. Falk and P. Lokuciejewski, "A compiler framework for the reduction of worst-case execution times," *Real-Time Systems*, vol. 46, no. 2, 2010, pp. 251–300.

[24] AbsInt Angewandte Informatik GmbH, "aiT: worst-case execution time analyzers." 2013. [Online] Available: http://www.absint.com/ait/

[25] S. Kollmann, "A context-sensitive real-time analysis with event streams," Ph.D. dissertation, Ulm University, 2012.

[26] K. Albers, "Approximative real-time analysis," Ph.D. dissertation, Universität Ulm, 2011.

[27] N. Fisher and S. Baruah, "A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines," in *Proceedings of the 17th ECRTS*, 2005, pp. 117–126.