Smart Technologies for Effective Reconfiguration: The FASTER approach

M. D. Santambrogio *, D. Pnevmatikatos †, K. Papadimitriou†, C. Pilato*, G. Gaydadjiev§, D. Stroobandt‡, T. Davidson‡, T. Becker¶, T. Todman¶, W. Luk¶, A. Bonetto*, A. Cazzaniga*, G. C. Durelli*, D. Sciuto*

* Dipartimento di Elettronica e Informazione, Politecnico di Milano

† Foundation for Research and Technology - Hellas

‡ Ghent University

§ Chalmers University of Technology

¶ Imperial College London

Abstract— **Current and future computing systems increasingly require that their functionality stays flexible after the system is operational, in order to cope with changing user requirements and improvements in system features, i.e. changing protocols and data-coding standards, evolving demands for support of different user applications, and newly emerging applications in communication, computing and consumer electronics. Therefore, extending the functionality and the lifetime of products requires the addition of new functionality to track and satisfy the customers needs and market and technology trends. Many contemporary products along with the software part incorporate hardware accelerators for reasons of performance and power efficiency. While adaptivity of software is straightforward, adaptation of the hardware to changing requirements constitutes a challenging problem requiring delicate solutions.**

The FASTER (Facilitating Analysis and Synthesis Technologies for Effective Reconfiguration) project aims at introducing a complete methodology to allow designers to easily implement a system specification on a platform which includes a general purpose processor combined with multiple accelerators running on an FPGA, taking as input a high-level description and fully exploiting, both at design time and at run time, the capabilities of partial dynamic reconfiguration. The goal is that for selected application domains, the FASTER toolchain will be able to reduce the design and verification time of complex reconfigurable systems providing additional novel verification features that are not available in existing tool flows.

I. Introduction

Altering the functionality of hardware so as to adapt to new requirements can offer great advantages in a wide range of application domains. For example, a Network Intrusion Detection System needs to scan all incoming network packets for suspicious content. The scanning has to be carried out at line-speed so that the communications are not slowed down, while the list of threats to check for may be extended and updated on a daily basis. Fixed hardware solutions can achieve high performance, and software solutions can easily adapt to the new set of threats, but neither can achieve adaptivity and high performance at the same time. Reconfigurable logic allows the definition of new functions to be defined in hardware units, combining hardware speed and efficiency, with ability to adapt and cope in a cost effective way with expanding functionality, changing environmental requirements, and improvements in system features. For the Intrusion Detection System the new rules can be hardcoded into the reconfigurable logic, thus retaining the high performance, while providing the necessary adaptivity and extensibility to new threats.

However, designing, implementing and verifying evolving hardware systems is harder compared to static ones. The FASTER project (Facilitating Analysis and Synthesis Technologies for Effective Reconfiguration) [1] aims at introducing a complete methodology to allow designers to easily implement and verify a system specification on a platform that includes one or more general purpose processor(s) combined with multiple acceleration modules implemented on one or multiple reconfigurable devices. Our goal is that for selected application domains, the envisioned toolchain will be able to reduce the design and verification time of complex reconfigurable systems by at least 20%, providing additional novel verification features that are not available in existing tool flows. In terms of performance, for these application domains the toolchain could be used to achieve the same performance with up to 50% smaller cost compared to programmable SoC based approaches, or exceed the performance by up to a factor of 2 for a fixed power consumption envelope.

FASTER will support both region-based [2] and micro-reconfiguration, a technique to reconfigure very small parts of the device [3]. The ability to handle both types of reconfiguration opens up a new range of application possibilities for run-time reconfiguration, as a much broader time frame for the reconfiguration itself is available and the underlying concepts are different for both types of reconfiguration. FASTER will also develop techniques for verifying static and dynamic aspects of a reconfigurable design at compile time using symbolic simulation -a powerful verification approach for static designs, and extending it to support both static and dynamic aspects of a reconfigurable design. We will also explore techniques for verifying selected static and dynamic aspects of a reconfigurable design at run time with a small impact on speed, area and power consumption. Finally, FASTER will provide a

powerful runtime system that will be able to run on multiple reconfigurable platforms and manage the various aspects of parallelism and adaptivity with the least overhead. Within this context, the **novel contributions** of the FASTER project can be summarized in the following:

- Including reconfigurability as an explicit design concept in computing systems design,
- Providing methods and tools that support run-time reconfiguration in the entire design methodology,
- Providing seamless integration of parallelism in the specification that can then be applied to software or (reconfigurable) hardware components,
- Providing a framework for analysis, synthesis and verification of a reconfigurable system,
- Providing efficient and transparent runtime support for partial and dynamic reconfiguration.

In the remaining of this paper, Section II provides the description of the overall problem FASTER is aiming to solve. Related works, with similar objectives, are also presented in this section. Section III describes the FASTER project providing a general view of its organization. Section IV focuses on the description of two *components* of FASTER: the offline analysis and the runtime support for managing dynamically reconfigurable systems. Finally, Section V wraps up the authors conclusions and present future research directions.

II. Context definition

Current and future computing systems increasingly require to be flexible and extensible even after the system is operational, in order to cope with changing requirements and improvements in system features. The FASTER project [1] aims at exploiting the capabilities of dynamic reconfiguration, both at design-time and run-time, by taking as input a high-level description of the application.

A. An Opportunity

In an ever-changing world there is an increasing demand for embedded systems that will be able to adapt to their environment or to meet new application demands. Adaptation by means of changing the software running on processors is not always adequate: many embedded applications require hardware-acceleration and it is imperative that also this hardware can adapt to application changes. Reconfigurable hardware is the key enabler for these systems. Hardware supported adaptation mechanisms provide a cost effective way of coping with changing requirements. This is in addition to providing the flexibility needed to allow functionalities to be defined and easily added or substituted after a system has been manufactured and is already deployed. However, the ability to take relevant reconfiguration issues into account from the initial system specification to the final system design and the mechanisms required to support this additional functionality at runtime are currently lacking.

B. FASTER in the EU Projects Context

Previous research and EU projects such as MORPHEUS [4], hArtes [5], Reflect [6], S4, Acotes [7], and Andres [8] focus on the necessary tool-chain and address similar issues as FASTER, they focus more on system-level or architectural aspects of reconfiguration. The two projects closest to FASTER are hArtes and MORPHEUS. hArtes focuses on the creation of a toolchain for mapping sequential applications in C onto a heterogeneous platform with different processors and possibly also an FPGA (Molen architecture). No dynamic reconfigurability is foreseen and the toolchain is geared towards parallelizing, partitioning, mapping and scheduling C code. The tasks marked for FPGA implementation follow the toolchain for Molen [9]. MORPHEUS addresses solutions for embedded computing based on a dynamically reconfigurable platform and tools. The approach is to develop a global solution for a particular modular heterogeneous SoC platform that provides a software-oriented design flow and toolset. These soft hardware architectures enable better computing density improvements, positioned between general purpose flexible hardware and general purpose processors. Again, runtime reconfiguration is possible with a coprocessor interface and instruction extensions using the TU-Delft Molen approach. While both these projects address similar issues as FASTER, there is no explicit emphasis on design and runtime aspects of partial and dynamic reconfiguration.

This is exactly where FASTER intends to contribute: to introduce partial and dynamic reconfiguration from the initial design of the system all the way to its runtime use. We therefore have to define the design concepts capturing both parallelism and reconfigurability as essential system properties and to provide efficient and user transparent runtime support for dynamic and partial reconfiguration. As reconfigurability increases both the runtime and design time complexity, an adequate framework for verification and analysis is a necessity and the formulation of such framework will therefore also constitute one of the targets of the project. FASTER is focused on exploiting reconfiguration at the hardware description level, so the front-end of these approaches can be used to translate languages such as C or SystemC into task graphs which can then be processed with the FASTER analysis and scheduling. Furthermore, we assume that applications are developed taking into account the dynamic and partial reconfigurability from the start rather than transforming code to take advantage of this feature. However, the tool is not restricted to new applications and existing code can be integrated in two possible ways: (a) using a pre-processing step that will produce the task dependency graphs to be analyzed by the FASTER tool chain so that its results and decisions can be used later on for the synthesis and mapping of the application, or (b) by employing a compilation approach such as the one proposed by the OSMOSIS Capacities project where the high-level description C or System C is transformed into HDL, which can then be processed by the FASTER tool-chain. Therefore, FASTER is complementary to hArtes since it adds an additional degree of flexibility and heterogeneity by adding partial dynamic reconfigurability both as region-based or micro-reconfiguration. The two tool-chains could be usefully integrated together. Within this context, the FASTER tool-chain will provide the design analysis tools to identify the reconfigurability characteristics of the application, determine the best implementation options,

and verify the resulting implementation in order to take advantage of these features from the start, rather than introducing reconfiguration late in the mapping process, or merely by transforming the code.

FASTER proposes a novel approach for computing system design, focusing on dynamically and partially reconfigurable FPGA-based architectures. This project binds classical computing system design and hardware reconfiguration to focus the design attention on the entire platform characterization. FASTER envisions the development of new tools and the identification and formalization of new models and design methodologies that can represent and implement efficient computing systems. The tools will cope with the application requirements as well as the development of area allocation and management algorithms for efficient runtime support of dynamic reconfiguration. FASTER will also extend the reconfiguration options above the traditional region-based design supporting both micro-reconfiguration and region-based reconfiguration in the developed tool set. This broadens the range of reconfiguration time scales that can be effectively used by target applications.

III. FASTER AT A GLANCE

The FASTER project aims to provide the ability to introduce reconfigurability as an explicit design concept. Towards this concept the following five objectives will be addressed:

- ability to explore and specify reconfiguration at any granularity that fits better the application of a designer,
- methods and tools for including run-time reconfiguration in every aspect of the design methodology,
- tools for analysing and verifying both static and reconfigurable parts of the system,
- provide a flexible run-time system and specify the required hardware support that the run-time system needs to efficiently implement the reconfiguration actions and optimize the system operation under the desired criteria, and finally
- integrate seamlessly parallelism and reconfigurability in the specification irrespective of hardware and software implementation.

Based on these objectives, a new model for addressing reconfiguration in dynamically reconfigurable FPGAs and new graph-theoretic algorithms for the temporal and spatial partitioning of a specification on the same architectures will be proposed. We aim to propose a novel approach in defining the clustering of tasks of a system specification by detecting recurrent structures in the specification itself. This will allow to identify modules that can be used more than once during the application lifetime in order to save device resources and reconfiguration time. The intermediate system representation is then analysed in terms of its spatial-temporal constraints. While most of the temporal constraints can be specified in the initial design phase, a more detailed assessment of them will take place in the second step.

This spatial-temporal profile is then verified against the initial application and design requirements. The next step consists of dynamic scheduling where the runtime requirements of the application are taken into account. By providing feedback to the initial design stage certain decisions can be revised. The outcome of this iterative process is a system description that conforms to the application requirements and where dynamic properties are also as much as possible taken into account. The final architecture is then generated along with additional information necessary for the runtime system.

A main innovation of the FASTER project is the support of static, and both micro-reconfiguration and region-based reconfiguration. In micro-reconfiguration, the parts being reconfigured are of fine granularity, such as LUTs (LookUp Tables) or routing switches. Micro-reconfiguration can be fast and configuration generation can take place at run-time, for instance triggered by the change of some parameters [10]. Conceptually, micro-reconfiguration deals with the adaptation of the implemented function by changing a small set of configuration bits only. In contrast, region-based reconfiguration deals with the instantiation of a new function in an entire region of the FPGA. The regions being reconfigured are of much coarser granularity, with large blocks of logic being swapped in and out. Region-based reconfiguration tends to be slow, and traditionally configuration generation takes place at design time. These pre-designed configurations program predefined regions during runtime. The ability to handle both types of reconfiguration opens up a new range of possibilities for run-time reconfiguration, which can offer a versatile framework for serving the design of applications targeting reconfigurable hardware.

Figure 1 illustrates the overall FASTER tool chain. Starting from the left side of the Figure, it begins with the description of the application (in HLL, HDL or in other formats such as task graph, etc.) plus the application requirements and an abstract description of the reconfiguration capabilities of the target platform. When starting from HLL (C/C++ with OpenMP annotations, openCL, etc), FASTER analysis (WP2) can determine which portions of the application can be accelerated in hardware and will be executed in software. In any case, the FASTER methodology then focuses on the portions to be executed in hardware, where the analysis on the corresponding HDLs determines which parts can be reconfigured dynamically and which portions are fixed. Further analysis will estimate the performance to offer feedback to the user. The analysis will also determine the most appropriate granularity for reconfiguration and the required parameters. WP3 performs synthesis, floorplanning, placement and routing for reconfigurable portions that have been selected for micro-reconfiguration. It also performs the necessary check to verify that the partitioned dynamically reconfigurable version of the application is equivalent to the original source. The partitioned application description is annotated with dependency information that will drive the dynamic aspects of the reconfiguration. The final bit files can be produced by vendor-specific tools, and together with the run-time scheduler established in WP4 they form the complete reconfigurable system. WP1 is an initial step in the project and will establish the list of requirements for the tools and determine the best evaluation criteria to assess the results of the project, and WP5 drives the validation and evaluation of the FASTER tools and flow.

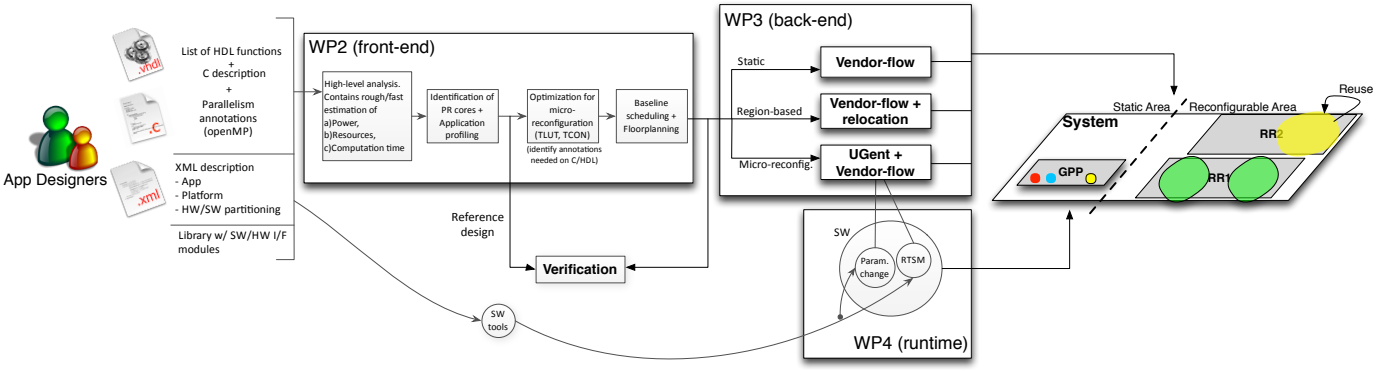

Fig. 1. FASTER design flow broken down in distinct work packages

IV. FASTER Offline and Runtime Support for Dynamically Reconfigurable Systems

A. High-level analysis and reconfigurable system definition

The main goal of this work package is to analyze each application and define its components, estimate its execution time on the target platform, identify the right level of reconfigurability for it (none, region-based or micro-reconfigurable), the power constraints, the floorplanning constraints (size and shape), the placement requirements (type of resources and connectivity among modules) on the target platform and the baseline schedule for its execution. In this work package, the definition of a methodology and workflow to partition a system specification into a task graph (in which every task is to be treated as a region-reconfigurable module or a micro-reconfigurable module) and to schedule it on the same reconfigurable architecture, will be carried out. Figure 2 provides an overview of its structure, showing the different stages from the original high-level specification to the partitioned system (i.e., cores which are ready to be scheduled for configuration and execution). This work package is composed of 5 tasks. The first task involves the use of high-level analytical models in guiding the optimization of the system. The second task involves application profiling for identifying opportunities for reconfiguration. The third task concerns application optimization for implementations involving micro-reconfigurable cores. Based on these three tasks, the fourth task concerns baseline scheduling of applications on the target platform. The output will be the baseline schedule and the description of the application subdivided into components, ready for synthesis using any commercially available tools for the targeted technology. Finally, different implementations of the same design can be verified as being equivalent to one another.

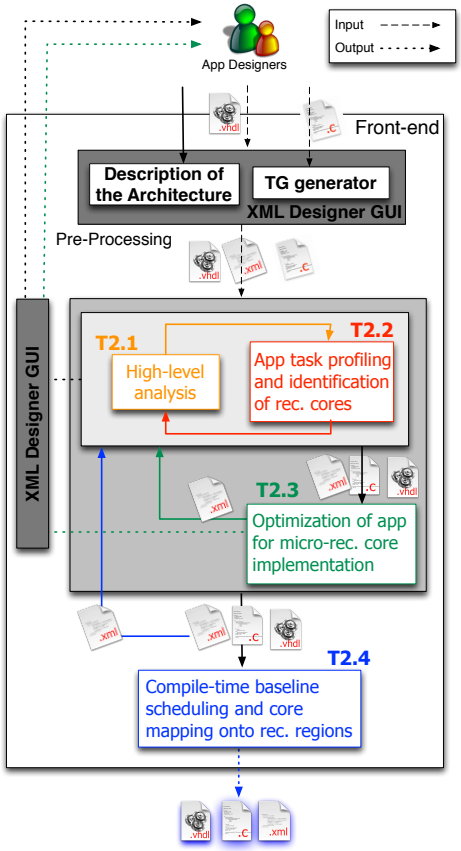

Fig. 2. FASTER high-level analysis and reconfigurable system definition

1) High-level analysis: The purpose of high-level analysis is to provide an analytical model of a reconfigurable design that relates its application attributes to possible implementation parameters, from which metrics such as area, performance and power consumption can be estimated. Based on the above implementation parameters we will then develop a tool for statistical prediction of metrics such as reconfiguration time, area, performance and power consumption, with which each component will be annotated. This process is intended to guide the identification of promising components and reconfiguration techniques at an early stage of the design, and to assess their impact on the implementation. Examples of application attributes include required throughput or execution deadlines, data precision and data set size. Examples of implementation parameters include available resources, their area-delay-power estimates, and the associated reconfiguration overheads. The output of high-level analysis is an estimate of how a design with a given set of application attributes and implementation parameters performs. However, it is not the goal of this task to produce code or to analyze code automatically. Rather, the estimation is intended to guide the identification of features of promising designs, which could benefit other tasks in WP2. Furthermore, the analysis process

continues in parallel throughout the implementation phase. It interfaces with WP3 in that implementation parameters are updated during synthesis and physical implementation. Such updated values are fed back into the model and are used to verify prior calculations. The analysis process also interfaces with WP4 because characteristics of run-time management techniques form part of the implementation parameters used in the analysis. For instance, one would analyze how the response time and reconfiguration time of run-time controllers influence application performance. Likewise the analysis process can be used to improve run-time management techniques by making explicit the requirements for response times, context save and restore procedures.

2) Profiling of applications for identifying region-based, micro-reconfigurable, and static cores: The input of this phase is the original specification, whose analysis results in cores, i.e. groups of operations that compose configurable modules, with optimal sizes. The identification of the region-based reconfigurable cores is performed by analyzing the Control Data Flow Graph of the input application, and trying to identify isomorphic subgraphs in the graph. The choice of finding isomorphic subgraphs is related to the possibility of reusing these components without reconfiguration, thus hiding/minimizing reconfiguration time. Within this context, if distinct components, having the same implementation, are mapped onto the same reconfigurable core, we can execute them on the same physically implemented module with a single initial reconfiguration. Obviously, at the end we have to identify a number of components, isomorphic or not, that cover the entire graph in order to restructure the entire application into a set of interconnected components.

Our first step is the core identification phase. The input of this step is the original specification, whose analysis results in cores, i.e. groups of operations that, reconfigured together as configurable modules, have optimal sizes. The second part of the task is the Partitioning phase. Using the previously computed set of modules as its input, this phase produces a set of feasible covers of the original graph of the specification, following a given policy.

3) Optimization of applications for micro-reconfigurable core implementation: Since every change of a parameter value in the parameterizable micro-reconfiguration results in a reconfiguration of (part of) the implementation, the number of parameter changes should be kept as low as possible. This requires a higher parameter value *locality* in time. The consortium has a profound experience in loop transformations that increase the locality of data in cache based architectures. We will investigate if and how such loop transformations can improve the locality of parameter values and apply these transformations to optimize the applications for the micro-reconfiguration implementation. In this task, we will also research how current applications under consideration can be altered to benefit more from micro-reconfiguration. This will mainly be an investigation on the introduction of parameters in such a way that the overall implementation can be optimized. In this respect, we will also investigate multi-mode applications where the different modes are similar (but not exactly the same) and investigate how such applications can best be represented to benefit from micro- reconfiguration.

4) Compile-time baseline scheduling: Given the information computed in the previous tasks, an integration between a heuristic reconfiguration-aware scheduler and a floorplacer algorithm will produce the baseline schedule of the tasks on the target platform. This scheduler is a reconfiguration-aware scheduler for dynamically partially reconfigurable architectures that can also manage static reconfiguration and multi FPGAs. Its distinguishing features are the exploitation of configuration prefetching, module reuse and anti-fragmentation techniques. It schedules the modules according to the actual hardware composition and availability: given the actual spatial constraints defined by the floorplacer, it schedules the applications trying to reach the minimum scheduling time. The function to be optimized can be chosen based on the user constraints. Output of this task is the list of timings for each module, i.e. start time, and reconfiguration time. However, a static schedule may not be possible for all of the targeted applications. For example, when the execution times of the modules are data-dependent the scheduler will only provide guidelines to the runtime system and the HW scheduler that will manage the system at runtime. A strict interaction with WP4 and the runtime scheduler is therefore foreseen, since the runtime scheduler will receive as input the results produced during the generation phase.

5) Verification: Verification answers the question: how do we know the transformed design preserves the same behaviour? Traditionally, hardware designers have used extensive simulation to verify that their designs implement the desired behaviour, but as designs become increasingly large, the number of test inputs required can become large. Rather than simulating the design logically or numerically, we use symbolic simulation to verify that a transformed version of the design preserves the same behaviour as the original. The symbolic simulation can run at word level, which allows us to verify larger designs, assuming that bit level operator implementations are correct. We combine symbolic simulation with an equivalence checker, based on the Yices tool [11], to check whether different symbolic outputs are actually equivalent or not.

We further extend verification to check the dynamic aspects of designs: the behaviour of reconfiguring designs. We use the concept of virtual multiplexers [12] to represent a reconfigurable region by virtual multiplexer-demultiplexer pairs enclosing all the reconfigurable parts that can be loaded into that region. This allows us to represent reconfigurable regions in our symbolic simulation with multiplexers whose control signals are generated by software parts of the design. Our approach can apply to both region-based reconfiguration and microreconfiguration.

Finally, our approach can extend to dynamic verification: verifying the design at run-time. This entails running our verification flow on the device as it runs.

B. Run-time reconfiguration management

The main goal of this work package is to provide the proper extensions of the runtime system to efficiently handle the

online scheduling and placement of envisioned dynamically reconfigurable system. Depending on the running application the appropriate techniques and module versions will be selected. The online scheduling will consider minimization of reconfiguration overhead, minimization of fragmented area, reduction in power consumption and temperature and inter module wire delay optimization. The basic guidelines for online scheduling are provided by the compilation stage of WP2. To support the above, the architecture will expose some light-weight system monitors and control hooks to the runtime system. The approaches developed in this work package are applicable, without lost of generality, to partially reconfigurable devices as well as to multi-chip systems consisting of several devices that support only complete reconfiguration. Our work for shaping the runtime system is constituted by four main tasks.

1) Evaluation of existing runtime system support for reconfiguration: Within the context of this task we will evaluate existing runtime systems in terms of their capabilities and limitations, and propose a set of features for the efficient support for reconfiguration. So far, work of limited scope has been conducted on runtime support for partially reconfigurable systems with most efforts targeting Xilinx Virtex-II FPGAs that support 1-D reconfiguration only. Some theoretical approaches are addressing newer devices like Virtex-4 and Virtex-5 supporting 2-D reconfiguration but they haven't been implemented, evaluated or verified yet. In either case, the capabilities are limited by the specific architectural features of specific devices with regard to the reconfiguration support. Currently, the FPGA technology imposes the limitation that specific areas for the partially reconfigurable modules should be determined beforehand; then the reconfigurable modules are loaded during execution. Already, research towards trying to overcome this limitation is ongoing. We will evaluate the state of the art looking into issues such as scheduling, fragmentation and area allocation, first as they apply to existing platforms, but also as we envision the operation of future reconfigurable devices.

2) Propose architectural extensions for runtime system: Efficient reconfiguration is crucial for the success of dynamically reconfigurable systems. Reconfiguration overhead is a main shortcoming of the current FPGAs and therefore, the efficient scheduling of the reconfigurations is critical to meet the timing constraints of the applications. Moreover, awareness of the device area constraints will assist decisions targeting the problem of fragmentation. This will allow keeping the number of reconfigurations at low level. Towards the same direction, we will explore other functionalities to improve reconfiguration time such as caching or predicting reconfigurations based on online statistics and prefetching them ahead they are needed. In fact we are planning to incorporate configuration caching and prefetching [13] in our runtime system. To support this functionality dependencies and restrictions are communicated from the synthesis process to the runtime system, along with possible static prefetch hints. In addition, the runtime system should collect online statistics and determine dynamically when to cache and prefetch configurations. A primary issue is the configuration cache size. For example the configuration cache of the internal configuration access port (ICAP) in Xilinx FPGAs is implemented with one Block RAM and is fixed, but the cache of an embedded processor such as PowerPC or Microblaze is implemented with Block RAMs and can be modified. A (double or triple) buffering scheme would extend the ICAP cache and thus increase the reconfiguration bandwidth. To this direction, a prefetching mechanism can load to the cache, or even to the configuration memory, the configuration data that correspond to the circuit that is more likely to execute in the near future.

This task will also address possible light-weight hardware support extensions for efficient FPGA reconfiguration. The set of system monitors and hooks that need to be exposed to the runtime system is one of our concerns. This allows a SW/HW holistic solution to a variety of problems. One such opportunity is bitstream relocation, according to which a single bitstream file can reconfigure different FPGA regions with the same functionality. Another line of research falling under this task is to investigate the hardware support needed for online verification and debug of the reconfigurable units.

Furthermore, information such as thermal data taken with temperature measurement can provide feedback to the runtime system to deal with overheating. Then, re-scheduling can be triggered to reduce temperature and consequently power dissipation. This is of significant and increasing importance mainly for the embedded domain; also it is a concern for the desktop and high-performance computing domains.

3) Integration of micro-reconfiguration in the run-time manager: In this task, parameterizable run-time micro-reconfiguration [10] will be integrated into the runtime system support. The reconfiguration procedure from WP3 should be called by the reconfiguration manager when parameters change value. Also, in scheduling the application tasks, the micro-reconfigurations should be scheduled accordingly.

4) Runtime configuration scheduling and device management: In this task we will investigate first how the runtime system has to manage the directives provided by the compilation/static-scheduling stage. Next, based on the application behaviour the system will be dynamically optimized in order to provide the best service. The runtime system can be part of the OS itself, or software code running underneath the OS and will undertake decisions such as

- the time slot in which the reconfiguration of a module will occur,
- the portion of the FPGA on which the module is going to be placed, and
- the time slot in which its execution will start.

To appropriately cope with reconfiguration, the runtime scheduler will be leveraged with area management capabilities. The input is provided from a dependency/communication graph and based on a list of criteria a decision will be made. Such criteria Are reconfiguration time, execution time of a module, device area constraints, precedence between the modules, and level of fragmentation. A relocation mechanism would allow to defragment the device and consequently trigger fewer reconfigurations. In addition, we will address the problem of scheduling with feedback from thermal data in order to reschedule the modules according to temperature and power dissipation values. To the same direction, the scheduler will

address the problem of keeping close modules that communicate frequently through wide-width buses; this will also allow for low-latency communication. To optimize the on-line scheduling of components for 2-D reconfiguration and minimize the reconfiguration time and improve the resource utilization we will investigate runtime techniques for hierarchically constructing more complex components out of the available simpler regular ones. The algorithms that will be selected depend heavily on whether the scheduling will be preemptive or non-preemptive. In the former case, an Earliest Deadline First (EDF) algorithm has been proven to be a good solution, however in the latter case using a pure EDF algorithm is not the best choice, due to the inserted idle times. Towards this problem heuristics and migration of techniques from the multiprocessor field might offer an appropriate solution. Before this, we will evaluate simpler algorithms like the First-Fit and Best-Fit. Furthermore, we will explore the use of Genetic Algorithms augmented with the capability to terminate upon non-fixed number of generation counts; an implementation in software or hardware (or mixture of both) can be used. We already have a hardware implementation of a GA, which due to its inherent pipelining and parallelism nature is two orders of magnitude faster than its counterpart implemented in C running in an embedded processor. Moreover, although this implementation is complex in logic, it is parameterizable by supporting different cost functions and genetic parameters such as variable population size and members width, and utilizes less than 8% of a Virtex-II FPGA [14]. This hardware-based genetic algorithm can assist the runtime support on scheduling decisions. Before proceeding with this, study is needed to balance the benefits and drawbacks from using such an assistant-circuit lying outside the runtime scheduler with regard to the I/O latency caused to start-up the hardware-based genetic algorithm plus to pass the results to the runtime software code over the speed of finding the solution.

Limitations due to the FPGA architectural features from the reconfiguration perspective might constrain the above solutions. Even the latest Xilinx FPGAs do not allow on-line change of the regions wherein the reconfigurable modules are to be placed. This can limit the ability of making decisions on real-time region allocation. Finally, we are concerned in examining the extent to which our mechanisms will be applied in a transparent manner regardless of the architectural constraints.

V. Conclusions

The FASTER project will enhance the following five aspects of the design of computing systems. The *first objective* is to include reconfigurability as an explicit design concept. Starting from high-level descriptions of the system and its application, our approach will provide a set of different metrics, such as communication and dependency graphs, profiling information, etc, oriented to support system-level synthesis onto a hardware platform supporting dynamic reconfiguration. The *second objective* is to provide the methods and tools for including run-time reconfiguration in every aspect of the design methodology. The methods and tools designed in FASTER will provide the tools necessary for efficient and transparent use of partial and dynamic reconfiguration at different time scales, supporting also both explicit parallelism in the application specification and platforms that combine multicore processors with reconfigurable logic. This will lead to a scenario that will enable efficient interfacing between the parallel software and the (reconfigurable) hardware components. The *third objective* of FASTER is to provide an effective framework for analysis, synthesis, and verification to guarantee that the final implementation corresponds to the application requirements and system specifications. Our focus is on developing an integrated tool-chain that supports the verification of both static and dynamic portions of the reconfigurable design. *Objective four* is to provide efficient and developer/user transparent runtime support for partial and dynamic reconfiguration. Assuming a partially reconfigurable system - either with a single or multiple FPGAs - we will develop a runtime system that can efficiently handle the online scheduling and placement of reconfigurable system components, using dynamically adaptive schemes to optimize the system operation based on different functional and non-functional requirements defined by the user or the earlier tool-chain. Finally the *fifth objective* is to provide seamless integration of parallelism and reconfigurability in the specification, irrespective of whether it applies to software or hardware components. The flow will interface the parallel software to the hardware components, and to the runtime manager responsible for partial and dynamic reconfiguration.

Acknowledgment

This work was supported by the European Commission in the context of FP7 FASTER project (#287804).

References

[1] http://www.fp7-faster.eu/, [Online; accessed May 2012].
[2] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs (Invited Paper)," in *Proceedings of the IEEE Conference on Field Programmable Logic and Applications (FPL)*, August 2006, pp. 1–6.
[3] K. Bruneel, "Efficient Circuit Specialization for Dynamic Reconfiguration of FPGAs," PhD thesis, Ghent University, 2011.
[4] http://ce.et.tudelft.nl/DWB/, [Online; accessed May 2012].
[5] http://hartes.org/hArtes/, [Online; accessed March 2012].
[6] http://www.reflect-project.eu/, [Online; accessed March 2012].
[7] http://www.hitech-projects.com/euprojects/ACOTES/, [Online; accessed March 2012].
[8] http://andres.offis.de/, [Online; accessed March 2012].
[9] http://ce.et.tudelft.nl/DWB/, [Online; accessed May 2012].
[10] K. Bruneel and D. Stroobandt, "Automatic generation of run-time parameterizable configurations," in *Proceedings of the IEEE Conference on Field Programmable Logic and Applications (FPL)*, August 2008, pp. 361–366.
[11] B. Dutertre and L. de Moura, "The YICES SMT Solver," Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 - USA, Tech. Rep., 2006.
[12] W. Luk, N. Shirazi, and P. Y. K. Cheung, "Modelling and Optimising Run-Time Reconfigurable Systems," in *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*. IEEE Computer Society Press, 1996, pp. 167–176.
[13] S. Hauck, "Configuration Prefetch for Single Context Reconfigurable Coprocessors," in *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 1998, pp. 65–74.
[14] M. Vavouras, K. Papadimitriou, and I. Papaefstathiou, "High-speed FPGA-based Implementations of a Genetic Algorithm," in *IEEE International Conference on Embedded Computer Systems, Architectures, Modeling and Simulation (SAMOS)*, July 2009, pp. 9–16.