System-Level Runtime Mapping Exploration of Reconfigurable Architectures

Kamana Sigdel[†] Mark Thompson[‡] Andy D. Pimentel[‡] Carlo Galuzzi[†] Koen Bertels[†]

[†] *Computer Engineering Laboratory*
EEMCS, Delft University of Technology
The Netherlands
Email: {K.Sigdel, K.L.M.Bertels, C.Galuzzi}@tudelft.nl

[‡] *Computer Systems Architecture Group*
University of Amsterdam
The Netherlands
Email: {M.Thompson, A.D.Pimentel}@uva.nl

Abstract

Dynamic reconfigurable systems can evolve under various conditions due to changes imposed either by the architecture, or by the applications, or by the environment. In such systems, the design process becomes more sophisticated as all the design decisions have to be optimized in terms of runtime behaviors and values. Runtime mapping exploration allows to explore reconfigurable systems at runtime to optimize task mappings in order to adapt to the changing behavior of the application(s), the architecture, or the environment. Performing such explorations at runtime enables a system to be more efficient in terms of various design constraints such as performance, chip area, power consumption, etc. Towards this goal, in this paper, we present a model that facilitates runtime mapping exploration of reconfigurable architectures. A case study of an MJPEG application shows that the presented model can be used to perform runtime exploration of various functional and non-functional design parameters.

1. Introduction

Reconfigurable computing is gaining popularity in the embedded system domain due to its ability to provide high computational power with great flexibility. A reconfigurable architecture is a flexible architecture able to adapt to the application - the structure of the architecture can change at start-up time or at runtime (in case of dynamically reconfigurable systems) to match the new application. These architectures are typically formed with a combination of a General Purpose Processor (GPP) and a reconfigurable hardware component such as an FPGA. The reconfigurable component is used to accelerate code fragments from applications in a processor/co-processor fashion [1],[2],[3].

The ever increasing intricacy of the functionalities as well as increasing use of reconfigurable heterogeneous resources significantly complicate the design of modern embedded systems. As a result, to create a good design, it is crucial to perform Design Space Exploration (DSE) at various design stages in order to efficiently appraise several design choices. DSE at the early design stages helps designers to systematically explore trade-offs between various design choices such as hardware/software partitioning, architecture–to–application mapping, task scheduling, task allocation, etc. As the design progresses, the design space can be gradually trimmed and pruned of unsuitable design alternatives until the most suitable solution is obtained.

Towards this goal, we are interested in designing a system-level modeling and simulation framework for high level runtime mapping exploration of dynamically reconfigurable architectures. Dynamic systems evolve under various conditions to adapt to any changes imposed by the architecture or by the applications or by the environment. In such systems, the design process becomes more sophisticated as all the design decisions have to be optimized in terms of runtime behaviors and values. *Runtime Mapping Exploration* allows to explore various task mappings in reconfigurable systems at runtime in order to adapt to the changing behavior of the application(s), the architecture or the environment. Performing such exploration at runtime enables a system to be more efficient in terms of various design constraints such as performance, chip area, power consumption, etc. In this paper, we present a framework for a two level DSE approach for runtime mapping exploration. In the first level exploration, we explore the system under static system conditions and find the most suitable mapping or set of mappings to address such conditions. Second level exploration optimizes the mapping(s) at runtime in order to satisfy the changing system conditions by learning from its current system information (the application, the architecture or the environment), from its historic data or even from the predicted behaviors/conditions of the system.

In [4], we presented a model for system-level DSE of reconfigurable architectures. This model can simulate and estimate the performance of a reconfigurable architecture at a high level of abstraction for the applications from the multimedia domain. In this paper, the model is extended to support the runtime mapping explorations in order to address dynamic and adaptive behavior of reconfigurable systems. We use the Sesame framework [5] as a modeling and simulation platform for system level DSE. The Molen architecture [6] is used as an example of a dynamic reconfigurable architecture. For this purpose, the Sesame framework has been extended to support dynamically reconfigurable behavior of the Molen architecture [4]. The main contributions of this paper are the following:

- The design of a system level modeling and simulation framework that supports runtime mapping exploration of reconfigurable architectures.
- A two level DSE approach for the runtime mapping exploration of reconfigurable architectures.
- An initial case study to demonstrate the runtime mapping exploration to address the dynamic conditions of the system using an MJPEG application.

The paper is organized as follows: Section 2 reviews some related works. Section 3 presents the two level mapping exploration approach using an example. Section 4 provides a brief description of the Molen architecture and the Sesame simulation framework. Section 5 describes the model and its implementation details. Section 6 provides a case study and some preliminary results. Finally, Section 7 presents conclusions and future work.

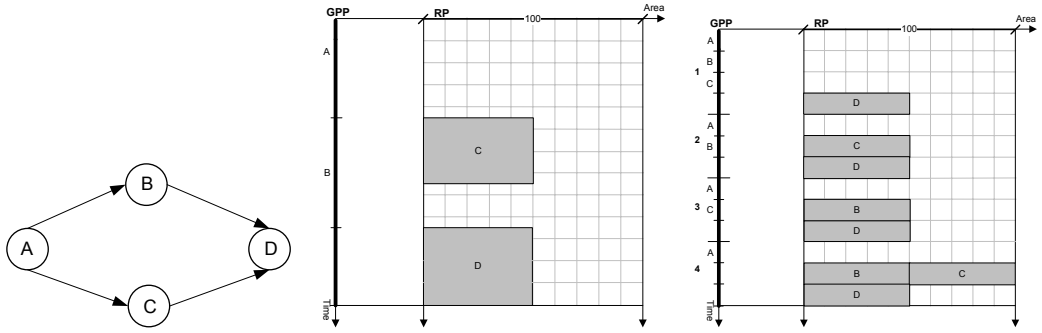
2. Related Work

System level modeling and simulation for DSE of reconfigurable systems has been touted for quite some time. However, there are not many system-level DSE tools available for reconfigurable architectures, which can assist designers at very early stages of a design. Several techniques for high level DSE, partitioning and mapping of applications on reconfigurable architectures have been presented in literature for performance evaluation and rapid exploration of different reconfiguration alternatives such as [7],[8],[9],[10]. Most of these approaches basically focus on the static nature of the architecture and/or applications. Furthermore, these techniques cannot address the adaptive and dynamic system nature of reconfigurable systems either from the architectural perspective and/or from the application point of view. In [11], a simple runtime DSE approach is discussed in which a DSE module dynamically explores the most frequently executed tasks and maps them onto a reconfigurable hardware. However, this work is focused at lower design levels and targets few specific tasks. Similar approaches for high-level runtime mapping of the application tasks onto heterogeneous MPSoC are presented in [12] and [13]. Close to our approach is the approach proposed in [14] [15], which present customized runtime management for heterogeneous MPSoC that combines a design time exploration together with runtime manager. Unlike them, our focus is to develop a simulation framework for runtime mapping of the application(s) onto the dynamic reconfigurable architecture at early design stages.

3. Dynamic Application Mapping

Dynamic systems can change under various conditions, while maintaining both functional consistency and non-functional design constraints such as power consumption, performance, redundancy, etc. These changes can be imposed and initiated by an architecture, by an application or by the environment. For example, an architecture can change under the following conditions: an increase or decrease of the available or connected resources, a requirement to temporarily switch-off one or more parts of the hardware to reduce the power consumption or a necessity to achieve high fault tolerant behavior for certain tasks. At the same time, the application can change to maintain a specified Quality of Service (QoS) for variable processing load of one or multiple applications, or due to the arrival of some additional sporadic tasks. In all these cases, it is necessary or beneficial to change the mapping of tasks on the various resources at run-time. Dynamic mapping allows a task to be executed on various resources at different time intervals during the execution of the application. As a result, a system needs to support task migration from one resource to another. Moreover, a system needs to implement some policies to define which mappings are used under certain conditions and when a mapping needs to be changed. Typically, a trade-off has to be made between the cost and benefits of the migration. In this work, we do not consider the implementation details of a reconfigurable system, rather we discuss the modeling and exploration framework to assist system designers with various design decisions.

In the presented framework, applications are represented as a Kahn Process Network (KPN) at the granularity of coarse-grain tasks [16]. A KPN consists of concurrent processes with explicit communication over FIFO channels. KPNs are deterministic and can capture the parallel and dynamic nature of streaming applications in the multimedia domain. Therefore, KPNs are suitable for our purpose. For this case, we only use acyclic KPN application graphs. The architecture model considered is the Molen architecture [6] [17] which consists of a GPP and a dynamically Reconfigurable Processor (RP). Application tasks can be executed on the GPP, on the RP or on both. Tasks run on the GPP as regular (compiled)



(a) KPN Graph of an Application (b) Static Mapping (c) Dynamic Mapping

Figure 1. Tasks Mapped onto the RP and the GPP with Static and Dynamic Mapping

microprocessor code or on the RP as a hardware IP core. We define three types of tasks in the system:

- *Software tasks* are those tasks that are executed on the GPP,
- *Hardware tasks* are those tasks that are executed on the RP,
- *Pageable tasks* are those tasks that can switch between these two processors and thus can be executed on both.

Our previous model presented in [4] assists designers to identify a set of hardware tasks and a set of software tasks such that the design constraints are satisfied. The mapping is determined statically (at design time) and, as a result, each task is mapped onto only one resource (either GPP or RP) during the entire runtime of the application. An example of such mapping behavior is shown in Figure 1(b). For the KPN graph given in Figure 1(a), the set of SW tasks is $\{A, B\}$ and the set of HW tasks is $\{C, D\}$. The dynamic mapping enables identification of the hardware and software task sets at runtime. An example of dynamic mapping behavior is shown in Figure 1(c). For time intervals 1 to 4 the sets are: $SW_1 = \{A, B, C\}$ $SW_2 = \{A, B\}$ $SW_3 = \{A, C\}$ $SW_4 = \{A\}$ and $HW_1 = \{D\}$ $HW_2 = \{C, D\}$ $HW_3 = \{B, D\}$ $HW_4 = \{B, C, D\}$ respectively. As can be inferred from the figure, for this particular example, task B and C change their mapping during execution from GPP to RP and vice versa. Therefore the set of pageable tasks is: $\{B, C\}$.

3.1. Two Levels Mapping Exploration

Using the model presented in the paper, mapping exploration can be carried out at two levels. At the first level, we address exploration of the system under *static* conditions. Static, here, refers to the condition of the system where applications, architectures and the environment do not change during the exploration. The system constraints imposed in such conditions are fixed and known during the exploration. For instance, a condition where a single application is mapped onto a fixed architecture is an example of a static condition. In this case, the exploration is performed at design time by iterative evaluation of different mappings until the most suitable task sets satisfying the given system constraints are identified. This is used as a starting point for the second level exploration.

At the second level, mapping exploration is carried out at runtime to address the runtime conditions of the system. In this case, we optimize the pageable task set at runtime by dynamically changing the mapping to satisfy the given design constraints. For example, in order to meet the real-time constraints of a sporadic task with faster execution, some tasks can be migrated from RP to GPP in order to give priority to the sporadic task. Similarly, if a sudden increase in the application load is detected, performance can be enhanced by moving tasks from the GPP to the RP. These decisions are made based on the condition and requirement of the running system.

To perform the runtime adaptation of the task sets, one or more components in the system have to make "intelligent" mapping decisions at runtime. Most importantly, the system should be able to identify *which* tasks to migrate and *when* to change the mapping to perform the migration. Various information about the runtime condition of the system can be used: application information such as task priority, real time constraints and architecture information such as free resources, timing information, etc. Note that this decision making process can be implemented in the actual system in any number of ways: e.g. in the application, in the hardware or as an operating system policy.

To illustrate the two levels of exploration, let us consider a scenario with two applications App1 and App2. Application

App2 in this case is sporadic and as a result can go in and out of the system at random time. The first level exploration identifies hardware, software and pageable task sets for each application separately. The exploration starts with an initial random solution and different task sets are identified by searching the design space with iterative simulations. Let's denote the best task sets for application App1 and App2 as: App1$_{SW}$, App1$_{HW}$, App1$_{page}$, App2$_{SW}$, App2$_{HW}$ and App2$_{page}$. In the second level exploration the application model App3 is the combination of App1 and App2. The initial task sets for the second level exploration is defined as:

$$App3_{HW} = App1_{HW} \cup App2_{HW}$$
$$App3_{SW} = App1_{SW} \cup App2_{SW}$$
$$App3_{Page} = App1_{Page} \cup App2_{Page}$$

In order to address the "dynamic behavior" of the application model App3, a number of explorations have to be carried out at runtime in order to find the best mapping for the pageable task set. With the new system condition, at certain times when application App2 starts to execute in the system, the tasks in $App3_{Page}$ have to compete for resources. As a result, some subset of $App3_{Page}$ may predominantly be executed either on hardware (P) or on software (Q); then the task sets can be changed accordingly:

$$App3_{SW} = App1_{SW} \cup App2_{SW} \cup Q$$
$$App3_{HW} = App1_{HW} \cup App2_{HW} \cup P$$
$$App3_{Page} = (App1_{Page} \cup App2_{Page}) - P - Q$$
$$\text{where} \quad |P| \geq 0, \quad |Q| \geq 0$$

Note that although not all the tasks in the HW task and/or pageable set may fit on the RP at the same time, all tasks can be executed with a delay after reconfiguring them on the RP. In such cases, in order to avoid such reconfiguration delays, some tasks from a HW task set can also be moved to a pageable task set. At any point during the exploration process the designer can influence the process by manually setting up these task sets using his domain knowledge or heuristics. For instance, he may decide to manually fix some tasks from one task set to another in order to reduce or enlarge the design space for iterative simulation.

4. Tools and Platforms

The modeling and simulation framework presented in the paper is not restricted to a particular type of architecture. For an evaluation purpose, in this paper, we use the Molen architecture as an example of a dynamically reconfigurable architecture. We use the Sesame framework as a modeling and simulation platform for system level DSE. The Sesame environment [18][5] is extended in order to model and simulate the dynamic reconfigurable behavior of the Molen architecture [4].

4.1. Reconfigurable Architecture

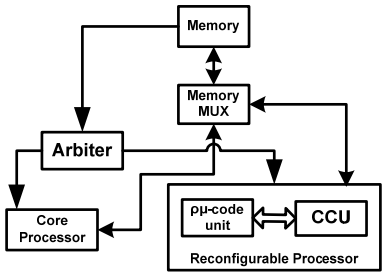


Figure 2. The Molen Architecture

The Molen polymorphic processor is established on the basis of the tightly coupled co-processor architectural paradigm [6][17]. The two main components in the Molen machine organization are the *Core Processor*, which is a GPP and the *Reconfigurable Processor* (RP) such as an FPGA. The GPP and the RP communicate through a shared memory. The RP is further subdivided into the $\rho\mu$-code unit and one or more *Custom Configured Units* (CCU) (see Figure 2). These two processors are connected to an arbiter [6][17]. The latter controls the co-ordination of the GPP and the RP by directing program instructions to either of these processors. In order to speed up program execution, parts of the code running on the GPP can be implemented on the CCU(s). The code to be mapped onto the RP is annotated with special *pragma* directives. When the arbiter receives the *pragma* annotated instructions, it initiates the CCU unit, gives data memory control to the RP and drives the GPP into a wait state. An operation executed by the RP is divided into two distinct phases: *set* and *execute*. In the *set* phase, the CCU is configured (i.e. a configuration bitstream is loaded onto the FPGA) and in the *execute* phase the CCU(s) starts its execution. When the RP finishes its execution, it acknowledges the arbiter. The arbiter releases the data memory control back to the GPP and then the GPP can resume its normal program execution.

4.2. Sesame Framework

The Sesame modeling and simulation environment [18][5] is geared towards fast and efficient exploration of embedded multimedia architectures, typically those implemented as heterogeneous MPSoCs. Sesame adheres to a transparent simulation methodology where the concerns of application and architecture modeling are separated via a mapping layer. It uses the KPN model of computation [16] for application

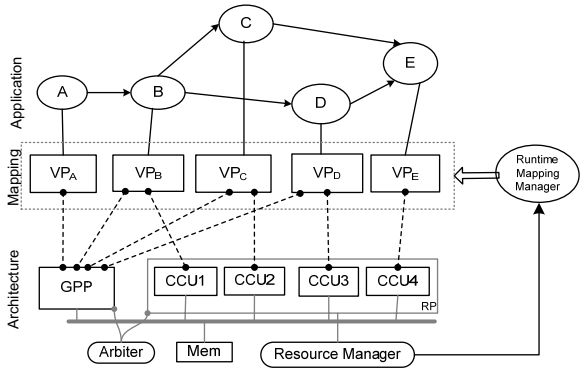


Figure 3. Sesame Model to Facilitate Runtime Mapping of Tasks from GPP to RP and viceversa

modeling. This model is suitable for modeling stream-based (multi-media) applications. A KPN consists of concurrent processes that communicate using blocking read/non-blocking write synchronization over unbounded Kahn channels. The application model generates event traces containing Read (R), Write (W) and Execute (EX) events. R and W are communication events that describe the communication over the Kahn channels. EX is a computation event that describes the computation behavior of a Kahn process (typically at the function granularity). These events are collected into event traces and mapped onto an architecture model using an intermediate mapping layer. The mapping layer consists of Virtual Processors (VP), which can be regarded as representatives of application processes at the architectural level. The main purpose of the mapping layer is to schedule the event traces from the application processes onto the architectural components. In the architecture model, the architectural timing consequences of the events are modeled. The processor components model the processor utilization of the studied application to architecture mapping. This is done using a lookup table that contains execution latency for each EX event. The interconnection and the memory components model the utilization and the contention caused by the communication events (R and W).

5. The Model

The model presented in this paper is an extension of our previous model for dynamically reconfigurable architectures as presented in [4]. This model could not perform dynamic mapping of tasks to architecture components. In this section, we describe the new model and the added modeling capabilities.

Figure 3 shows an overview of the components in the different layers of the model. The architecture model consists of the actual system components - Processors, Memory and Interconnection Networks and the components such as Arbiter, Resource Manager (RM) and Runtime Mapping Manager (RMM), which implement some model behavior. In the real system, RM and RMM can be implemented either at operating system level or at architectural level. There is one GPP and one RP which, in turn, may consist of multiple CCUs. The GPP and all CCUs are connected to the shared memory by a bus network component. A CCU represents the hardware implementation of a task. Since the RP is dynamically reconfigurable, the number and type of CCUs on the RP may change at runtime. The RM component is connected to all the CCUs. Its basic functionality is to keep track of which CCUs are *configured* on the RP at any given time. A CCU can only process events when it has been configured; if it is not, then it requests the RM to be scheduled for configuration. The RM configures the CCUs according to the policy implemented (eg. as soon as area becomes available). The Arbiter component performs synchronization between the GPP and the RP. It supports either mutual exclusive operation of GPP and RP (traditional co-processor model) or parallel operation.

Note that, in the mapping layer the communication channels are not shown in order to simplify the picture. The mapping layer – among other things – implements an elaborate synchronization mechanism for the events generated by the application layer, e.g. it makes no sense to un-configure a CCU from the RP after it has read data, but before performing a function on it (because the data would be lost). Therefore specialized synchronization in the mapping layer groups R, W and EX events for processing on a CCU. For more details we refer to [4].

5.1. Runtime Application Mapping

In order to facilitate runtime application mapping onto the architecture, the VPs can be connected to multiple processor resources. The trace events from the application model can now be forwarded to any connected processor in the hardware layer. Figure 3 shows an example where a virtual processor VP_B is connected to the GPP and CCU1. A hardware or a software task requires only one connection to a CCU or GPP respectively. However, a pageable task has a connection to both processors. In many scenarios, it is conceivable that there exists multiple CCU implementations for one pageable task. However, for the sake of simplicity, we currently restrict it to one. The VP forwards events, but does not make the mapping decision by itself. This is done by the RMM to which all VPs are connected.

5.2. Runtime Mapping Manager

The RMM is the central intelligent component which performs dynamic mapping decisions. It can use all information available in the model (application, mapping and architecture) as inputs for its decision making policy. For example, Figure 3 shows the connection between the RM and the RMM

since the availability of resources on the RP is an important consideration for the RMM.

The complexity of RMM implementations can be relatively simple as well as extremely complicated. RMM can use any available method or heuristic to perform the mapping decision based on current, historic or even predicted dynamic conditions of the system. The RMM bears the ability to learn from its environment, from its previous data, or the current situation. The application information, e.g. priority of tasks, real time constraints and the architectural information e.g. resources or timing information can be used as an input information for the RMM. Based on these values and behaviors, the RMM can make mapping decision for each task. Since the dynamic mapping policy is implemented as a separate component, it is easy to plug-in different RMM policies for different experiments.

5.3. Component Interaction

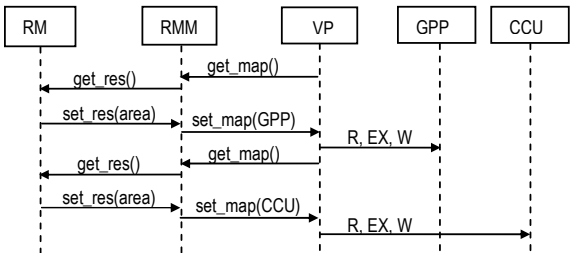


Figure 4. Interaction Between Various System Components to Enable Runtime Application Mapping

To summarize the interaction between different components, consider Figure 4. It shows an example time-interaction diagram of the components that are involved in processing application trace events for the GPP and a CCU. Before forwarding any events, the VP asks the RMM onto which processor the trace event has to be executed. Based on the policy implemented, the RMM returns a target processor identifier (either CCU or GPP) and the VP forwards events accordingly. To support its decision making process, the RMM may request additional information about the system or the environment. In this case it requests the current available area from the RM. If there is available area on the RP, RMM maps the task onto the CCU otherwise to the GPP.

6. Case Study and Preliminary Results

In this section, we present a case study using the presented model and discuss some preliminary results. The main focus of this case study is to demonstrate an experimental validation of two level mapping explorations using the presented model. Therefore, the actual calibration of the model or the evaluation of several mapping algorithms is left as future work.

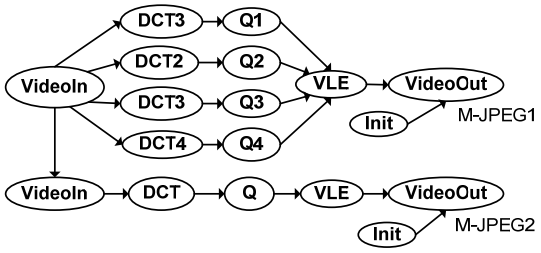


Figure 5. Application Model

We use an MJPEG application as an application model. Frames are divided in blocks and each task performs a different function on each block as it is passed from one task to another. There are two implementations of the model: MJPEG1 operates on the blocks (partially) in parallel, MJPEG2 operates on the blocks sequentially (see Figure 5). MJPEG1 and MJPEG2 are combined together in order to create an example of dynamic application behavior in the system. MJPEG2 can be considered as a sporadic application that appears in the system randomly and competes with MJPEG1 for the available resources. This behavior is implemented such that at unspecified times MJPEG2 starts encoding a frame simultaneously with MJPEG1. Figure 5 shows the application model used for this case study.

We instantiate an architecture model with 10 CCU units. For illustration purposes, we use estimated values of computational latencies for the GPP and CCUs. Similarly, we also use estimated values for the area occupancy on the FPGA and the reconfiguration delay for each task. We have the assumption that, no task can have a size larger than the total FPGA area. For carrying out these experiments, we fix the size of each task to 50% of the available area on the FPGA. Therefore, two tasks can execute on the FPGA at the same time, however, after reconfiguration, more than two tasks can execute on the FPGA, as a result, there can be more than two tasks in the HW and/or pageable task set. Our final assumption is that there is no delay associated with the dynamic mapping such as task migration time, context switching time, delay associated with the RMM, etc.

The first level of static exploration is carried out with the model we presented in [4]. MJPEG1 and MJPEG2 are separately mapped onto the given architecture and a range of different mappings are evaluated. At the end of this exploration, the most suitable mapping is recorded which has the following set of HW, SW and pageable tasks for each application. For MJPEG1: $HW_1 = \{DCT_1, DCT_2, DCT_3, DCT_4, Q_1, Q_2, Q_3, Q_4\}$, $SW_1 = \{VideoIn_1, VLE_1, VideoOut_1, Init_1\}$ and $Page_1 = \{\emptyset\}$. Similarly, for the MJPEG2, $HW_2 = \{DCT, Q\}$, $SW_2 = \{VideoIn, VLE, VideoOut, Init\}$ and $Page_2 = \{\emptyset\}$. Note that, only a subset of these tasks can fit on the FPGA at the same time, but other tasks can also be executed

Pageable Task Set used as input to the Runtime Exploration	Tasks added to SW task set after the Runtime Exploration	Elapsed Time		
		Dynamic Mapping	Static Mapping (GPP)	Static Mapping (CCU)
DCT1	None	250299094	255014646	219436393
DCT1, DCT2	None	289329064	292241251	219436393
DCT1, DCT2, DCT3	DCT2	292183820	335035856	219436393
DCT1, DCT2, DCT3, DCT4	DCT2	346752251	374238416	219436393
DCT1, DCT2, DCT3, DCT4, DCT	DCT2, DCT4	357931106	449243536	219436393
DCT1, DCT2, DCT3, DCT4, DCT, Q	DCT2, DCT4	334587257	482804304	219436393
DCT1, DCT2, DCT3, DCT4, DCT, Q, Q1	DCT2, DCT3, DCT4	377277865	501284688	219436393
DCT1, DCT2, DCT3, DCT4, DCT, Q, Q1, Q2	DCT2, DCT3, DCT4, Q1	392565665	519765072	219436393
DCT1, DCT2, DCT3, DCT4, DCT, Q, Q1, Q2, Q3	DCT1, DCT2, DCT3, DCT4, Q1, Q2	407599184	538245456	219436393
DCT1, DCT2, DCT3, DCT4, DCT, Q, Q1, Q2, Q3, Q4	DCT1, DCT2, DCT3, DCT4, Q1, Q2, Q3	425876433	556725840	219436393

Table 1. Experimental Results

on the FPGA after the reconfiguration of the FPGA for these tasks. The second level of exploration can help to optimize these task sets and can find an efficient set of tasks at runtime. This can possibly also improve performance by avoiding the reconfiguration delay for a task running on the GPP.

In order to carry out the second level of mapping exploration at the runtime, the combined application model shown in Figure 5 is considered. The initial task sets used as input for this exploration are the combination of task sets from the first level exploration as discussed in Section 3.1. These task sets are as follows: HW = $\{DCT_1, DCT_2, DCT_3, DCT_4, Q_1, Q_2, Q_3, Q_4, DCT, Q\}$, SW = $\{VideoIn_1, VLE_1, VideoOut_1, Init_1, VideoIn, VLE, VideoOut, Init\}$ and Pageable = $\{\emptyset\}$.

Based on analysis of the previous result, we choose some of the HW tasks as pageable tasks for runtime exploration. Note that, it is possible to mark all the tasks as a pageable tasks. However some tasks are better fixed either as hardware or as software to avoid several delays (reconfiguration delay and/or change in mapping delay, etc) associated with them. First level exploration provides such indications for creating an efficient task set for runtime mapping exploration. At the same time, the designers can impact these decisions based on his/her knowledge and preferences. During the second level exploration, the behavior of the pageable tasks at runtime is noticed. For this, in each successive simulation, one task from HW task set is marked as a pageable task and the performance numbers are recorded. This result is listed in Table 1. The first column in the table shows the pageable tasks before performing the runtime mapping exploration. Note that, the software task set given in this case is fixed. The second column in the table shows the pageable tasks which change their execution behavior and become SW tasks after performing the runtime exploration, other tasks than these tasks stay pageable. The third column in the table shows the simulated execution time when the task mapping is changed at runtime between the GPP and the RP. The last two columns of the table show the simulated execution time when the tasks are statically mapped onto the GPP (column 4) and the RP (column 5) respectively.

The mapping policy implemented by the RMM is very simple. The manager tries to allocate tasks to the RP, if area is available, otherwise the tasks are allocated to the GPP. As it can be inferred from Table 1, after performing runtime exploration, the pageable tasks either move to the GPP or to the RP based on the availability of chip area on the latter. This process is highly affected by number of pageable tasks specified in the system. When there are less pageable tasks, these tasks are mapped onto the RP, however when this number increases, many of these tasks are pushed onto the GPP. In the first experiment, as shown in the first row of the table, we consider only DCT_1 from HW task set as a pageable, the rest of the tasks in the HW set are fixed to be HW. In this case, the DCT_1 stays pageable and executes on the GPP and the RP. However, in the last experiment, as shown in the last row of the table, we consider all the tasks of the HW set as pageable tasks and, as a result, many tasks move to the GPP and become SW tasks. This is due to the fact that the RP has limited area.

Another observation that can be made from Table 1 is in terms of execution cycles. When all the tasks are mapped onto CCUs, the performance is higher (column 5 of the table) than when they are mapped on to GPP (column 4 of the table). This happens because of the lower execution latency of CCUs as compared to the GPP. With the dynamic mapping, the performance has ranged between these two values (column 3 of the table). Note that, executing all the tasks on CCUs can obtain the best performance but CCUs are limited in terms of area and they also have penalty associated with reconfiguration overhead. With the runtime mapping, we can obtain performance improvement if RMM takes into account these factors. Nevertheless, in this case study, we do not see any performance improvements due to two main reasons. First, the policy implemented by RMM is very simple and it does not take into account the performance evaluation, configuring delay, etc., for mapping decisions (it only looks at available area). Second, for this particular experiment, we used a fixed and very roughly estimated reconfiguration

delay for all the application tasks. Another reason for not measuring speedups might be the fact that the area of all tasks is fixed as 50% of the FPGA. A larger diversity of size may give other results. We believe that, in the future, we can see real performance improvements when realistic values of reconfiguration delays/area are used and/or the RMM implements intelligent policies based on a learning algorithm.

7. Conclusion and Future Work

In this paper, we presented a framework for two level DSE for runtime mapping explorations of the reconfigurable architecture. A case study of an MJPEG application demonstrated that the model can be used to explore the pageable task set and optimize the task set at runtime by either moving them to the HW set or the SW set. Due to the area constraints of the RP, when the number of pageable tasks in the system increased, more and more of them moved onto the GPP. The model presented in this paper is flexible and can be easily extended to incorporate various system information and evaluate different mapping exploration methodologies. As a result, in our future work, we will implement and evaluate various runtime mapping exploration methodologies in order to evaluate dynamic reconfigurable systems. Moreover, we will validate the model against a real implementation to allow for final calibration of the model in order to increase its accuracy.

Acknowledgement: This research has been funded by the hArtes project EU-IST-035143, the Morpheus project EU-IST-027342 and the Rcosy Progress project DES-6392.

References

[1] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable computing: architectures and design methods", *IEEE proceedings of Computers and Digital Techniques*, vol. 152, June 2005, pp. 193–207.

[2] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software", *ACM Computing Surveys*, vol. 35, June 2002.

[3] S. Vassiliadis and D. Soudris, *Fine- and Coarse-Grain Reconfigurable Computing*, S. Vassiliadis and D. Soudris, Springer, 2007, vol. XVI.

[4] K. Sigdel, M. Thompson, A. Pimentel, T. P. Stefanov, and K. Bertels, "System-level design space exploration of dynamic reconfigurable architectures", *Proceeding of International Symposium on Systems, Architectures, MOdeling and Simulation*, July 2008, pp. 279–288.

[5] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels", *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, 2006.

[6] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The molen polymorphic processor", *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.

[7] J. Noguera and R. M. Badia, "System-level power-performance trade-offs in task scheduling for dynamically reconfigurable architectures", *Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems*, pp. 73–83, October 2003.

[8] P.-A. Hsiung, C.-S. Lin, and C.-F. Liao, "Perfecto: A systemc-based design-space exploration framework for dynamically reconfigurable architectures", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 3, pp. 1–30, 2008.

[9] T. Rissa, M. Vasilko, and J. Niittylahti, "System-level modelling and implementation technique for run-time reconfigurable systems", *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp 295.

[10] Y. Qu and J.-P. Soininen, "Systemc-based design methodology for reconfigurable system-on-chip", *Proceedings of the 8th Euromicro Conference on Digital System Design*, 2005, pp. 364–371.

[11] G. Still, R. Lysecky, and F. Vahid, "Dynamic hardware/software partitioning: A first approach", *Proceedings of the Design Automation Conference*, 2003.

[12] A. Kumar, B. Mesman, B. Theelen, H. Corporaal, and H. Yajun, "Resource manager for non-preemptive heterogeneous multiprocessor system-on-chip", in *Proceedings of the Workshop on Embedded Systems for Real-Time Multimedia*, 2006, pp. 33–38.

[13] O. Moreira, J. J.-D. Mol, and M. Bekooij, "Online resource management in a multiprocessor with a network-on-chip", in *Proceedings of the 2007 ACM symposium on Applied computing*, 2007, pp. 1557–1564.

[14] C. Ykman-Couvreur, E. Brockmeyer, V. Nollet, T. Marescaux, F. Catthoor, and H. Corporaal, "Design-time application exploration for mp-soc customized run-time management", *Proceedings of International Symposium on System-on-Chip*, 2005, pp. 66–69.

[15] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-time management of a mpsoc containing fpga fabric tiles", *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 1, pp. 24–33, 2008.

[16] G. Kahn, "The semantics of a simple language for parallel programming", in *Proc. of the IFIP Congress 74*, 1974.

[17] S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. M. Panainte, "The molen programming paradigm",*Proceeding of International Symposium on Systems, Architectures, MOdeling and Simulation*, July 2003, pp. 1–10.

[18] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures", *EURASIP J. Embedded Syst.*, vol. 2007, no. 1.