Semi-distributed control for FPGA-based reconfigurable systems

Chiraz Trabelsi, Samy Meftali, Jean-Luc Dekeyser

Semi-distributed control for FPGA-based reconfigurable systems

Chiraz Trabelsi*, Samy Meftali* and Jean-Luc Dekeyser*
*INRIA Lille Nord Europe - LIFL - Universite Lille1
Lille, FRANCE
Email:{Firstname.Lastname}@inria.fr

***Abstract*—Due to the growing complexity of the applications targeted by FPGA-based reconfigurable systems, the control design of such systems is becoming one of the main hurdles faced by designers. In this paper, we propose a semi-distributed control model based on the separation between different control concerns (monitoring, decision-making and reconfiguration) and on formalism-oriented design in order to decrease the design complexity of the control, and facilitate design verification, reuse and scalability. This model is composed of distributed controllers handling each the self-adaptivity of a reconfigurable region of the system, and a coordinator that coordinates their reconfiguration decisions in order to respect global system constraints. Implementations on FPGA showed that our semi-distributed control model is more flexible, reusable and scalable than the centralized one, at the cost of a slight increase in required hardware resources.**

I. Introduction

Thanks to their ability to be reconfigured an arbitrary number of times, FPGAs offer a high flexibility for modern embedded systems design. Partial Dynamic Reconfiguration (PDR), supported by several FPGAs, offers more flexibility by allowing portions of the FPGA to be reconfigured at runtime, in order to load different functionalities and adapt to runtime changes, while the rest remains operating [1]. The progress in FPGA technologies has enabled to embed a growing number of computing resources on one chip targeting increasingly sophisticated applications. However, this has led to a growing design complexity since design tools do not evolve at the same pace as hardware technology, resulting in a productivity gap. One of the most complex design tasks for reconfigurable SoC (RSoC) is the control design, since it has to handle different aspects related to runtime adaptivity. In this context, autonomy, modularity and formalism-oriented design can be viewed as an effective combination to deal with the growing RSoC design complexity. In this paper, we propose a semi-distributed control model for FPGA-based reconfigurable systems. This model divides the control problem between autonomous controllers handling each the self-adaptivity of a reconfigurable region of the system through three major tasks: monitoring, decision-making and reconfiguration using three different modules. In order to respect global system constraints and correlations between reconfigurable regions, the controllers' reconfiguration decisions are coordinated by a coordinator before launching reconfigurations. This two-layer decision-making is well adapted to single-FPGA, as well as multi-FPGA systems by implementing the coordinator on a master FPGA. The semi-distributed control is also well adapted to higher hierarchy control architectures by organizing the controllers into clusters coordinated by local coordinators and implemented either on the same FPGA or on different FPGAs. The proposed semi-distributed decision-making model is based on the mode-automata formalism, which allows to abstract the control problem and gives clear control semantics decreasing design complexity. Such a splitting of the control problem offers a high design flexibility facilitating reuse and scalability.

Implementation results showed that our semi-distributed control model is more flexible, reusable and scalable than the centralized one, at the cost of a slight increase in required hardware resources. The rest of this paper is organized as follows. Section 2 gives a summary of the related works. Section 3 illustrates the proposed control model. In section 4, an example of the control implementation for a video processing application is presented. The last section concludes this paper and gives some future works.

II. Related works

Distributed control for FPGA-based systems adaptivity has been proposed by several works. In [2], a hardware controller was allocated to each reconfigurable region in order to control the tasks it runs throughout the application execution. However, reconfiguration decisions were only dependent on a task graph, and the correlation between regions was not treated. In [3], the distributed control was used for the reconfiguration of an organic computing system. However, this work focused more on the distributed access to the configuration port (ICAP), allowing to accelerate the reconfiguration process compared to the centralized access, without giving details about the used control components (monitoring, decisions, etc). In [4] [5], the authors propose a general model of networked entities, handling each computing, monitoring, control and communication. Nevertheless, the decision of reconfiguring such entities is done in a centralized way. Distributed control was also used for multi-FPGA systems, which have started to gain interest and have been investigated in several works. However, only one controller was used per FPGA [6] [7], which implies a high design complexity of controllers, and no formalism was proposed to model the control system.

To master design complexity, formal control models are important since they enable shorter design cycle by improving

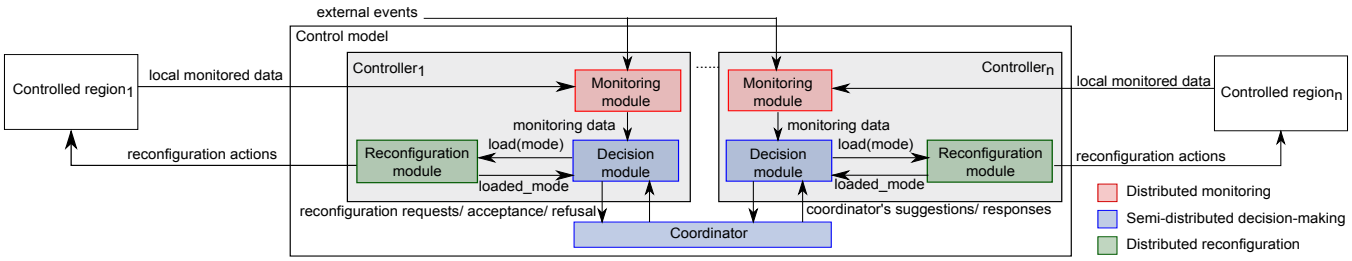

Fig. 1: Overview of the proposed control model

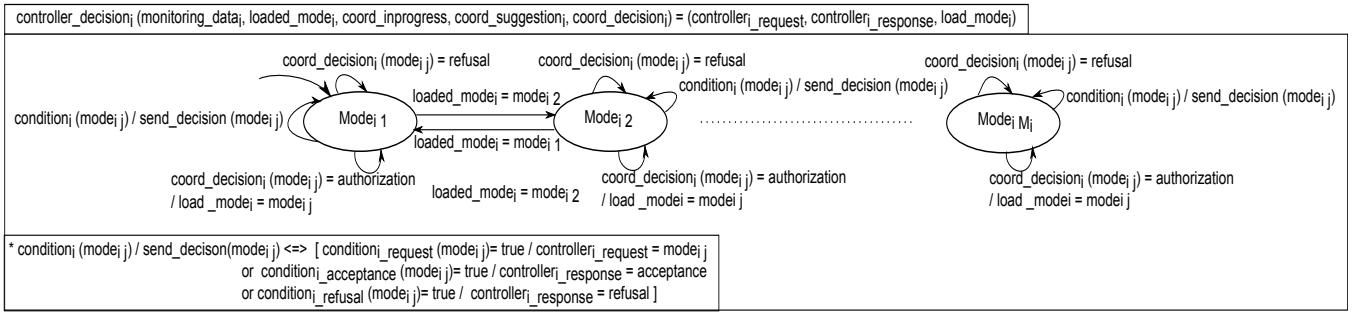

Fig. 2: The controller mode-automaton

design reuse as well as offering formal verification. Mode-automata formalism [8], on which is based the semi-distributed decision-making in our control model, is a simplified version of State charts [9] in syntax. It has been adopted as a specification language for control oriented reactive systems [10]. It has also been used to model control for FPGA-based reconfigurable systems [11]. However, the proposed models targeted a centralized controller.

III. Semi-distributed control model

The main objective of the control model proposed in this paper is to solve design problems related to design complexity, verification, reuse and scalability. In order to achieve this objective, the proposed control model combines three main points: autonomy, modularity and formalism.

A. Autonomous modular distributed controllers for self-adaptivity

Each distributed controller is composed of three modules handling monitoring, reconfiguration decision-making and reconfiguration realization for a given reconfigurable region, as shows Figure 1. The monitoring module collects information from the behavior of the controlled region and other external events such as those sent by sensors, for example. The monitoring data is sent to the decision module, which makes reconfiguration decisions accordingly.

Each decision module makes local decisions about whether or not a reconfiguration of the controlled region is required. Due to the local vision of each controller, launching a reconfiguration of its controlled region without checking whether it can coexist with the current configurations of the other regions might result in problems such as safety problems or might not respect the control global constraints such as those related to performance, temperature, energy consumption, etc. Therefore, before launching a reconfiguration that it estimates required according to the monitoring data, the controller has to send a reconfiguration request to the coordinator. If the coordinator authorizes the requested reconfiguration, the decision module notifies the reconfiguration module in order to launch the required configuration.

The role of the reconfiguration module is to apply reconfiguration actions on the controlled region, which consist in loading the required configuration data (bitstream) in the reconfigurable region through a configuration port, such as ICAP for Xilinx FPGAs. After loading the required bitstream, the reconfiguration module notifies the decision module so that it updates its automaton's current mode.

B. Mode-oriented decision-making

The proposed decision-making model is based on the mode-automata formalism [8]. It is composed of the distributed controllers' automata and the coordinator's automaton. These automata communicate through coordination information whenever one of the controllers estimates that a reconfiguration of its controlled region is required. In this case, it sends a reconfiguration request to the coordinator and waits for its decision. If the request implies also the reconfiguration of other regions in order to respect the global system constraints, the coordinator sends reconfiguration suggestions to the concerned controllers. These controllers can accept or refuse those suggestions. After treating the controllers responses, the coordinator gives its decision, which can be either the authorization or the refusal of the requested reconfiguration.

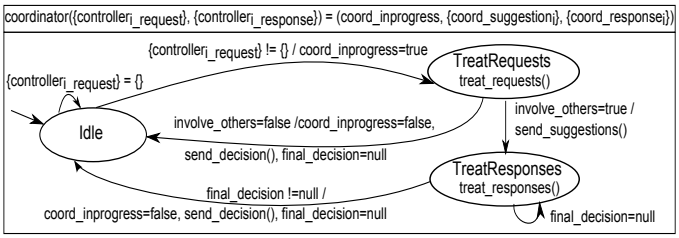

Fig. 3: The coordinator mode-automaton

Reconfigurable regions \ Global configurations	1	2	.....	K
$Region_1$	$mode_{1\ j1.1}$	$mode_{1\ j1.2}$	.....	$mode_{1\ j1.K}$
$Region_2$	$mode_{2\ j2.1}$	$mode_{2\ j2.2}$	.....	$mode_{2\ j2.K}$
.....	.....	.....	.....	.....
$Region_n$	$mode_{n\ jn.1}$	$mode_{n\ jn.2}$	.....	$mode_{2\ jn.K}$

Fig. 4: Global configurations table (GC)

The Controller mode-automaton

The decision module of each controller is modeled by a mode-automaton. Figure 2 shows an example of this mode-automaton. Each mode $Mode_{ij}$ corresponds to a given configuration/mode of the controlled $region_i$, where $i \in [1..n]$, n is the number of the system's reconfigurable regions, $j \in [1..M_i]$ and M_i is the number of the modes/configurations of $region_i$. Here, we assume that each reconfigurable region has a set of configuration possibilities predefined at design-time. Inputs and outputs of the mode-automaton are shown in its header. Inputs are monitoring data ($monitoring\_data_i$) sent by the monitoring module, a reconfiguration success notification ($loaded\_mode_i$) from the reconfiguration module, and coordination information from the coordinator. These information include a notification at the beginning/end of each coordination process ($coord\_inprogress$), reconfiguration suggestions ($coord\_suggestion_i$), and the coordinator decision at the end of a coordination process ($coord\_decision_i$).

Based on monitoring data, the current mode of the controlled region, and on coordination information ($coord\_inprogress$ and $coord\_decision_i$), the controller makes a decision on whether a reconfiguration of the controlled region is required. If yes, it sends a reconfiguration request to the coordinator ($controller_{i\_request}$). During a coordination process, a controller might receive reconfiguration suggestions from the coordination to which it can respond ($controller_{i\_response}$) with acceptance or refusal depending on monitoring data. If the coordinator authorizes a reconfiguration to the controller ($coord\_decision_i(mode_{ij}) = authorization$), it sends a reconfiguration command to the reconfiguration module ($load\_mode_i$), indicating the mode to be loaded. If the coordinator refuses a reconfiguration ($coord\_decision_i(mode_{ij}) = refusal$), it cannot be launched because it cannot coexist with the current configurations of the other regions, according to the global system constraints as it will be explained later.

The coordinator mode-automaton

The role of the coordinator is to coordinate the reconfiguration decisions of the controllers in order to guarantee that the system configuration respects the global system constraints. For this, a table containing the allowed system configurations is used, according to constraints of safety, performance, consumption, etc. This table is determined at design time and can be filled manually by the designer or generated from high-level languages such as those based on contract mechanism [12]. We call this table GC (Global Configurations) shown in Figure 4, where each row corresponds to a global configuration, which is a combination of partial configurations of the reconfigurable regions. This table is defined as follows: $GC[i,k] = mode_{ij_{i.k}}\ \forall i \in [1..n]$, $j_{i.k} \in [1..M_i]$ and $k \in [1..K]$ where, $mode_{ij_{i.k}}$ is the mode of $region_i$ corresponding to the global configuration k; n is the number of reconfigurable regions and of controllers; K is the number of the global configuration possibilities, and M_i is the number of modes/configurations of $region_i$.

The exchanges between the controllers and the coordinator are not continuous in time. They only happen when one of the controllers decides that a reconfiguration of its controlled region is required. This allows to reduce the impact of the communication latency inside the control model on the overall system performance. The coordination algorithm is executed using a three-mode automaton as shows Figure 3. The *idle* mode corresponds to a coordinator waiting for reconfiguration requests coming from controllers. The $TreatRequests$ and $TreatResponses$ modes correspond to a coordinator that is treating the controllers' reconfiguration requests and responses to reconfiguration suggestions, respectively.

The coordinator starts at the *idle* mode. Whenever it receives a reconfiguration request ($\{controller_{i\_request}\} \neq \{\}$), it sends a notification to the controllers indicating that a coordination is in progress ($coord\_inprogress = true$), so that they stop sending reconfiguration requests. It moves then to the $TreatRequests$ mode. Due to its local vision of the system, a request sent by a controller concerns only its controlled region, which corresponds here to a cell of the *GC* table. Note that the number of requests received at the same time depends on the instants reconfiguration decisions are made by controllers, as well as on the communication type between the controllers and the coordinator (through a bus, point-to-point, etc). In this paper, we used a point-to-point communication as will be shown in the case study. This implementation has the advantage of accelerating the coordination process offering the possibility to receive more than one request or response from controllers at the same time. Other communication architectures are still possible here by modifying the communication part of the coordinator without modifying the coordination algorithm.

Once in the $TreatRequests$ mode, the coordinator checks whether the configuration(s) requested by the controller(s) at a given time combined to the current configurations

of the regions handled by other controllers exist as a global configuration possibility in the *GC* table. If no, it checks which other reconfigurations that, combined to the one requested, allow to obtain one or more global configurations that respect the global system constraints. Here, if more than one global configuration satisfies the requested reconfiguration(s), possibilities are ordered according to the control strategy/objective. In the present implementation of the coordinator, we order possibilities in a list according to the number of partial reconfigurations they require, and we give the highest priority to the global configuration that requires less partial reconfigurations in order to minimize reconfiguration time. Different algorithms can also be used here to order configuration possibilities.

In case the first possibility in the ordered list satisfies the requests and doesn't require reconfiguration of other regions ($involve\_others = false$), the coordinator ends the coordination process ($coord\_inprogress = false$) and sends directly its decision ($final\_decision = authorization$) to the controllers requesting the reconfigurations as shows Figure 3. Otherwise, the coordination process is divided into steps. Each coordination step is related to a possibility of the ordered list. The coordinator begins with sending reconfiguration suggestions to the controllers whose regions have to be reconfigured in order to move to the first possibility. Then it goes to the *TreatResponses* mode.

If a coordination step ends with positive responses from all the concerned controllers, the coordination process ends with an authorization ($final\_decision = authorization$). In this case, the coordinator notifies the controllers ($coord\_inprogress = false$) and sends its reconfiguration authorization to the controllers requesting the reconfigurations as well as the other concerned controllers, and goes back to the *idle* mode. Otherwise, the coordinator remains at the *TreatResponses* mode and considers the next possibility. The coordination ends with a refusal ($final\_decision = refusal$) if all the possibilities have been refused by the controllers. In this case, the coordinator notifies the controllers ($coord\_inprogress = false$) and sends a reconfiguration refusal to the controllers requesting the reconfigurations. Then it goes back to the *idle* mode.

Advantages compared to the centralized model

Thanks to the distribution of the control problem between controllers, the proposed control model provides a high design flexibility compared to the centralized control facilitating reuse and scalability. In the case of a centralized decision-making, the decision module of the centralized controller can be modeled by a mode-automaton, where each mode corresponds to a global configuration of the system (a combination of the configurations of the reconfigurable regions). Transitions from a mode to another depend thus on a global vision of the system, using monitoring data as well global constraints (the same as those used by the coordinator in the semi-distributed model). This makes the centralized decision module tightly dependent on the implemented system, which is an obstacle to design reuse. Indeed, when designers want to add other regions to a previous system design, the whole decision model has to be rewritten (both modes and transitions), because each mode has to take into account the new regions. On the other hand, with the semi-distributed decision-making model, adding new regions requires simply adding a controller for each new region. The monitoring and decision modules of the controllers can be easily reused since they depend only on the monitoring data related to the controlled region. The coordination algorithm doesn't change. It has only to increase the number of controllers to be coordinated, and to modify the global constraints checked by the coordinator. This is done simply by modifying the GC table.

IV. Case study

This case study explains the use of the semi-distributed control for a video scaling application, targeting a single-FPGA system. Then it evaluates its efficiency in terms of design complexity, reuse, scalability and resource overhead compared to the centralized model.

A. The application

The application consists of a classical downscaler composed of two main tasks: a horizontal filter and a vertical filter applied to a sequence of video frames. Each filter is composed of a repetition of an elementary task on a block of the frame. An elementary task of a filter is executed by a hardware accelerator in order to guarantee a high performance. Using data-parallelism, each task can be implemented using a number of hardware accelerators performing in parallel the same elementary task on different frame blocks, which allows to reduce execution time. In our case study, we assume that each hardware accelerator is implemented in a reconfigurable region in order to adapt to runtime changes as we will explain later. The variety of parallelism possibilities of the considered application allows to test the scalability of our control model by varying the number of reconfigurable regions and thus the number of controllers. Using similar accelerators to implement data-parallelism for each filter task allows to reuse the same controller for similar regions, reducing thus the design time of the control model.

In our case study, the objective of the control model is to adapt the downscaler application to changes in performance and power requirements. For this, we assume that both elementary tasks of the horizontal and vertical filters are implemented in three versions of hardware accelerators available in an IP (Intellectual Property) library, and different in terms of performance and power consumption. Switching these versions during runtime allows each reconfigurable region to have three different modes ($HFilter\_mode_j$/$VFilter\_mode_j$), $j \in [1,2,3]$. Modes $HFilter\_mode_1$ and $VFilter\_mode_1$ are the modes giving the highest performance but also the highest consumption for the horizontal and vertical filters respectively. $HFilter\_mode_3$ and $VFilter\_mode_3$ are the least performing but the least consuming. Our semi-distributed control model allocates a controller to each region in order to control its

behavior allowing to switch different modes depending on requirements in terms of performance and power consumption. The role of the coordinator is to verify that the global configuration of the system respects the constraints indicated in the *GC* table as we explained previously in section III-B, and this by coordinating the reconfigurable decisions made by controllers.

B. Hardware design of the semi-distributed control

Our control model is wholly designed in hardware in order to avoid the execution time overhead of the software implementation. This design follows the control model described in figure 1. The inputs of this model are performance and consumption information. The objective of the control in this case study is to make a trade-off between the performance and the consumption constraints. We define performance requirements, for both filters, as three performance levels given by the user, where level 1 corresponds to the highest performance. As for consumption requirements, we assume that a battery sensor is used in order to give the battery level at each clock cycle. This information is monitored by the monitoring modules of the distributed controllers in order to be taken into account for reconfiguration decisions.

Provided that all reconfigurable regions have three configuration possibilities, each controller uses a decision module modeled by a three-mode automaton. Figure 5 shows the mode-automaton of the controller related to the horizontal filter, through three different control aspects. The controller related to the vertical filter follows the same concepts. As we said previously, the decision-making of each controller depends on a local vision of the system, which decreases its design complexity. This case study shows how the local-vision decisions of the controllers can be coordinated in order to respect global system constraints. The controller's decisions (reconfiguration requests and responses to suggestions) are based on the following rules:

- No request is sent when a coordination process is in progress ($coord\_inprogress = true$)
- If a request has been refused by the coordinator for $mode_j$ ($refused\_mode_j = true$), no request is sent for the same mode or it will be refused again. Moving to the requested mode is still possible, at a later coordination process, if the coordinator sends a suggestion to the controller related to the same mode and the coordination process ends with an authorization
- Being at $H/VFilter\_mode_{j1}$, a controller decides that a reconfiguration to a less consuming mode $H/VFilter\_mode_{j2}$ is required, only if the user requires a lower performance level, or the consumption constraints do not allow to stay at $H/VFilter\_mode_{j1}$, which is the case when the following condition is valid

$$AB/H_{j1} < a_{j1,j2}.FB/H_1 \quad (1)$$

  as shows Figure 5(a), where H_j (V_j for the vertical filter) is the energy consumption per cycle of the controlled region's mode $H/VFilter\_mode_j$, AB is the available battery energy at a given clock cycle and FB is the energy of a full battery. This constraint allows to check whether the available energy is under a threshold (determined by $a_{j1,j2}$) that allows to stay at $H/VFilter\_mode_{j1}$, taking as a reference the highest consumption (H_1). In Figure 5(a), $a_{1,2} = 75\%$ and $a_{2,3} = 75\%.75\%$, which give the thresholds to move from $H/VFilter\_mode_1$ to $H/VFilter\_mode_2$ and from $H/VFilter\_mode_2$ to $H/VFilter\_mode_3$ respectively.
- In order to move from a $H/VFilter\_mode_{j2}$ to a $H/VFilter\_mode_{j1}$ that consumes more, it is necessary that the user requires a performance level that is higher than the previous one and that the consumption constraints allow to move to the target mode, which is the case when

$$AB/H_{j1} >= (a_{j1,j2} + b_{j2,j1}).FB/H_1 \quad (2)$$

  Note that we add the term $b_{j2,j1}$ in order to avoid that, once in $H/VFilter\_mode_{j1}$, the consumption constraints in (1) lead the controller to decide to go back to $H/VFilter\_mode_{j2}$ so soon, which would lead to an infinite loop. $b_{j2,j1}$ has to be well chosen in order to avoid this problem. In our case study, we take a $b_{j2,j1} = 5\%$ as shows Figure 5(a).
- If the controller receives a reconfiguration suggestion from the coordinator it treats it as follows. If the suggestion requires to move to a less consuming mode, the controller accepts directly. Otherwise, the controller checks the consumption constraints in (2) in order to accept or refuse as shows Figure 5(b).

The coordinator's automaton was implemented according to the description in Figure 3. After a reconfiguration is authorized by the coordinator, the decision modules of the concerned controllers notify the reconfiguration modules. Partial reconfigurations can then be launched either in a parallel or a sequential way. Parallel reconfigurations are only possible for systems having more than one configuration port (ICAP for Xilinx FPGAs), such as multi-FPGA systems but not single-FPGA systems because current FPGAs have only one ICAP. Therefore, in our single-FPGA system, the distributed reconfiguration model was implemented in a way that it realizes reconfigurations in a sequential manner. Each reconfiguration module contains a dedicated register indicating which mode is to be loaded in the controlled region. These registers are then read by a processor, which communicates with the ICAP port in order to load the required bitstreams in the reconfigurable regions. When a required configuration is loaded, the processor notifies the reconfiguration module. The reconfiguration module notifies then the decision module, which updates its current mode in the mode-automaton accordingly.

C. Design reusability and scalability

In order to evaluate the efficiency of our control model compared to the centralized one in terms of design reusability and scalability, we designed the semi-distributed and centralized control models for different numbers of controlled

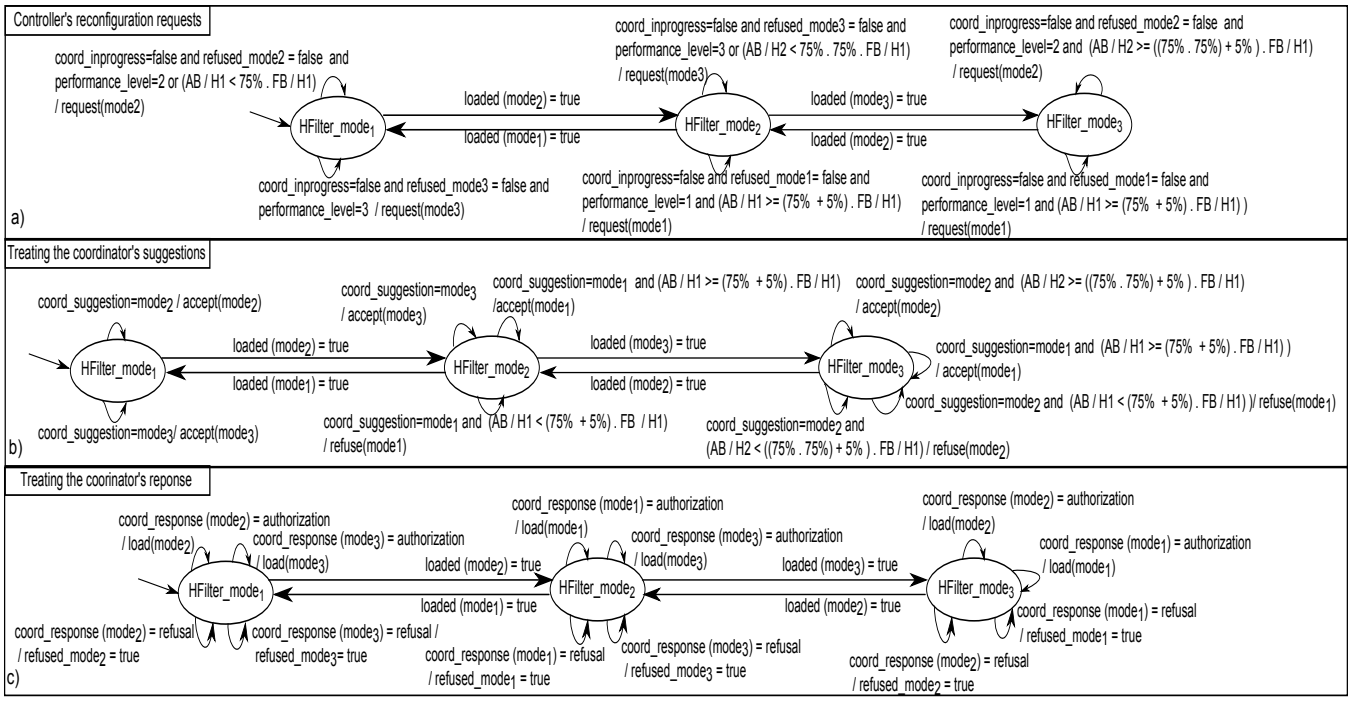

Fig. 5: Mode-automaton of the HFilter's controller

regions (up to $n = 10$ regions, where $n/2$ regions implement the horizontal filter and the rest the vertical filter). We started the semi-distributed model design with one controller for each type of filter. Later, we reused these controllers to compose bigger control models. For the coordination scalability, we only modified the number of coordinated controllers given as parameter to the coordinator, as well as the GC table. The splitting of the control problem between the controllers and the coordinator allowed us to test them separately, which facilitated significantly the design phase. On the other hand, adapting the centralized controller to different numbers of regions was more complicated. The centralized controllers were implemented using one mode-automaton as explained previously. The controller was rewritten each time to adapt to the system implementation, which led to longer design phases.

D. The semi-distributed control through a simulation scenario

Our semi-distributed control model was simulated using ISE 12.4 of Xilinx for different numbers of controllers. In order to explain more the evolution of the semi-distributed decision making at runtime, we will consider a simulation scenario for a control model with 4 controllers and a coordinator. The first two controllers control regions $HFilter_1$ and $HFilter_2$, which implement the horizontal filter. The two others control regions $VFilter_1$ and $VFilter_2$, which implement the vertical filter. In this case study, we suppose that global systems constraints require that all the regions have to implement the same mode number as shows Table I. This choice allows to test both acceptance and refusal in coordination processes by restricting global configuration possibilities. The inputs of the control model are the available battery signal sent by the battery sensor, and the processor commands. These commands allow to send the user required performance level, read reconfiguration registers of the reconfiguration modules and notify them at the end of the reconfigurations. These inputs were simulated using VHDL processes. Here, we suppose that the processor reads the reconfiguration registers after each frame downscaling. Table II represents the characteristics of the simulated system for the studied scenario, in terms of performance (frame/s) and power consumption. These values are used to determine the energy consumption per cycle of the controlled regions, the decrementation step of the battery, and the instants the processor reads the reconfiguration registers.

Figure 6 describes the simulation scenario represented by a chart, where the x axis describes instants when different events occur. These events are related to the current battery level, the performance level required by the user and the coordination processes. The y axis describes the available battery energy with a precision of the thresholds used by the different controllers to make reconfiguration decisions. The chart points are labeled with numbers corresponding to the global configuration at different instants of the simulation.

At $t < t_1$, the current global configuration is number 1. The user required performance level is 1. At $t = t_1$, the available battery energy (AB) reaches 75% of a fully-charged battery (FB). In this case, all controllers send reconfiguration requests to the coordinator asking to move to $H/VFilter\_mode_2$ as we have seen in Figure 5(a). The coordinator notifies the controllers that a coordination process is in progress. Then, it looks for the global configuration(s) that satisfy the received

	Global configuration number		
	1	2	3
Region 1	$HFilter\_mode1$	$HFilter\_mode2$	$HFilter\_mode3$
Region 2	$HFilter\_mode1$	$HFilter\_mode2$	$HFilter\_mode3$
Region 3	$VFilter\_mode1$	$VFilter\_mode2$	$VFilter\_mode3$
Region 4	$VFilter\_mode1$	$VFilter\_mode2$	$VFilter\_mode3$

TABLE I: GC table for the 4-region system

Global configuration	1	2	3
Consumption of H/VFilter (mW)	60/70	40/50	20/30
Performance (frames/s)	10	8	5

TABLE II: Power consumption and performance for different configurations of the 4-region system

requests. Here, only one global configuration satisfies the requests, which corresponds to column 2 of Table I. Since the current coordination process doesn't involve additional controllers to those that sent the requests, a reconfiguration authorization is sent to the controllers with a notification of the end of the coordination process. The decision modules of the controllers send reconfiguration commands to the reconfiguration modules asking them for loading $H/VFilter\_mode_2$ as it was shown in Figure 5(c). Then, when the processor launches its read commands, it finds that the controllers require to move to $H/VFilter\_mode_2$. After loading the required partial configurations, the processor notifies the controllers ($loaded(mode_2) = true$ in Figure 5(a)) so that they update the current modes of their mode-automata to be $H/VFilter\_mode_2$. The whole process ends at $t = t_1 + p_1$, by modifying the global configuration to 2. Note that p_j corresponds to the coordination process number j including the time required to load partial bitstream if the reconfiguration has been authorized.

At $t = t_2$, the available battery is less than $(75\%.75\%).FB.V2/V1$. Controllers 3 and 4 send reconfiguration requests to the coordinator related to $VFilter\_mode_3$. The coordinator sends then reconfiguration suggestions (moving to $HFilter\_mode_3$) to Controller 1 and 2. These controllers accept the suggestions according to Figure 5(b). The coordinator authorizes then the reconfiguration to all the controllers. The whole process ends at $t = t_2 + p_2$ by modifying the global configuration to 3.

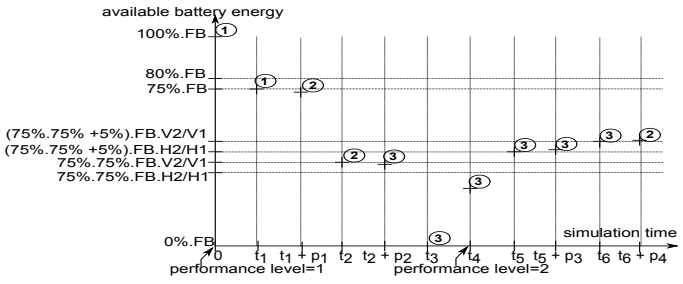

Fig. 6: Simulation scenario

At $t = t_3$, the battery is flat, so it moves to the charging mode. Since the regions implementing the horizontal filter consume less than the other regions, controllers 1 and 2 are the first to reach a battery threshold allowing their regions to move to $mode_2$ if the performance level required by the user is 2. For this, we simulate the change of the user performance level to 2 at $t = t_4$. At $t = t_5$, the battery threshold is reached for Controllers 1 and 2 ($AB > (75\%.75\% + 5\%).FB.H2/H1$). In this case, controllers 1 and 2 send reconfiguration requests to the coordinator in order to move to $HFilter\_mode_2$ at $t = t_5$. The coordinator suggests $VFilter\_mode_2$ to controllers 3 and 4. Since the available battery doesn't allow to move to $VFilter\_mode_2$ ($AB < (75\%.75\% + 5\%).FB.V2/V1$), controllers 3 and 4 send a refusal to the coordinator. The coordination process ends at $t = t_5 + p_3$ with a reconfiguration refusal sent by the coordinator to controllers 1 and 2. Those controllers will not send requests to move to $mode_2$ anymore because they will be refused. They can wait until the coordinator sends them reconfiguration suggestions to move to $mode_2$. At $t = t_6$, there is enough battery to move to $VFilter\_mode2$. Controllers 3 and 4 send reconfiguration requests to the coordinator. The coordinator sends suggestions to controllers 1 and 2 to move to $HFilter\_mode_2$. These controllers accept, and the whole process ends at $t = t_6 + p_4$ by modifying the global configuration to 2.

E. Resource and power overheads

After simulating both control models, we synthesized them for Virtex6-xc6vlx240t in order to estimate their overheads in terms of hardware resources. Figure 7 shows that the overhead of the distributed controllers is linear with the number of reconfigurable regions, because as we explained in section IV-C, the controllers are reused to move from a parallelism degree to another. The overhead of controllers is up to 0.57% of slice registers and 1.36% of slice LUTs, which is an acceptable overhead compared to the overhead of reconfigurable regions as presented in works such as [13] and [14]. The coordinator's overhead is also linear with the number of regions. The main reason to this is that the implemented coordinator uses a point-to-point communication allowing to handle many requests/responses from and to the controllers at the same time, which increases the required resources with the number of distributed controllers. Using different communication types decreases the overhead of the coordinator at a cost of longer coordination processes. However, here also there is an overhead of the communication architecture (overhead of a bus, a NoC, etc.). Up to 10 controlled regions the overhead of the implemented version of the coordinator is acceptable (0.035% of slice registers and 0.29% of slice LUTs).

Table III gives a comparison between the overhead of the semi-distributed and the centralized models. The semi-distributed control model has an overhead that is almost twice the centralized model overhead for different numbers of regions. This difference of overhead is mainly due to the resources required for the coordination between controllers as well as to the higher modularity of the semi-distributed control.

Resource occupation		Number of controlled regions				
		2	4	6	8	10
Semi-distributed control model	Slice registers	375 (0.12%)	744 (0.25%)	1112 (0.37%)	1479 (0.49%)	1847 (0.61%)
	Slice LUTs	474 (0.31%)	907 (0.6%)	1388 (0.92%)	2010 (1.33%)	2499 (1.66%)
Centralized control model	Slice registers	290 (0.1%)	434 (0.14%)	576 (0.19%)	720 (0.24%)	862(0.29%)
	Slice LUTs	286 (0.19%)	545 (0.36%)	736 (0.49%)	964 (0.64%)	1207 (0.8%)

TABLE III: Synthesis details of the semi-distributed and centralized control models

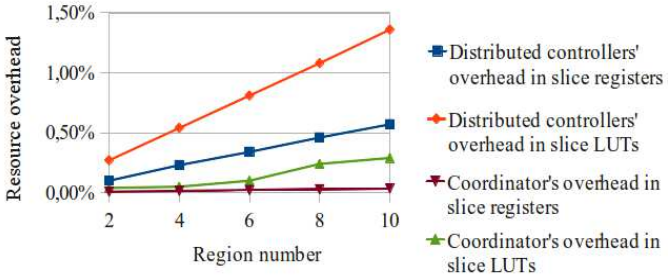

Fig. 7: Resource overhead variation of the semi-distributed model with the number of controlled regions

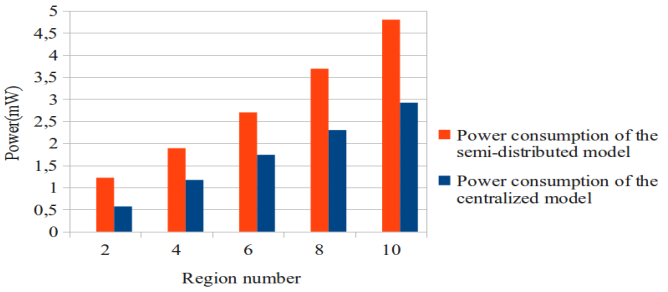

Fig. 8: Power overhead of both semi-distributed and centralized control models

However, the overhead of the semi-distributed model (up to 2499 slice LUTs) is quite acceptable for large FPGAs such as Virtex-6 used here, and the even larger Virtex-7 [15]. The power overhead of the semi-distributed model is also almost twice the centralized model overhead for different numbers of regions for a frequency of 100Mhz as shows Figure 8. Up to 10 regions this overhead does not exceed 5mW, which is considered as an acceptable consumption compared to the consumption of reconfigurable regions.

V. CONCLUSION

In this paper, we propose a semi-distributed control model aiming to decrease the complexity and enhance the reusability and scalability of the control design. This control model is well adapted to both single-FPGA and multi-FPGA systems. It is composed of distributed controllers controlling each the runtime adaptivity of a region of the system, and a coordinator for the controllers reconfiguration decisions. The semi-distributed decision-making model is based on the mode-automata formalism allowing to decrease its design complexity and facilitate its reuse. Implementation on FPGA showed that our decentralized control model is more flexible, reusable and scalable than the centralized one, at the cost of a slight increase in required hardware resources. As future works, we plan to integrate our control model in a whole reconfigurable system in order to evaluate more its efficiency. Our control model can also be used in a Model-Driven Engineering based SoC design flow in order to generate automatically its code, taking advantage of the high abstraction offered by the control formalism.

REFERENCES

[1] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited paper: Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas," in *FPL*, 2006, pp. 1–6.

[2] S. Toscher, T. Reinemann, and R. Kasper, "An adaptive fpga-based mechatronic control system supporting partial reconfiguration of controller functionalities," in *Proceedings of the first NASA/ESA conference on Adaptive Hardware and Systems*, ser. AHS '06, Washington, DC, USA, 2006, pp. 225–228.

[3] C. Schuck, B. Haetzer, and J. Becker, "An interface for a decentralized 2d reconfiguration on xilinx virtex-fpgas for organic computing," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 7:3–7:3, January 2009.

[4] C. Gamrat, J.-M. Philippe, C. Jesshope, A. Shafarenko, L. Bisdounis, U. Bondi, A. Ferrante, J. Cabestany, M. Hubner, J. Parsinnen, J. Kadlec, M. Danek, B. Tain, S. Eisenbach, M. Auguin, J.-P. Diguet, E. Lenormand, and J.-L. Roux, "Aether: Self-adaptive networked entities: Autonomous computing elements for future pervasive applications and technologies," in *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, vol. Chapter 7. Springer, 2011, pp. 149–184.

[5] J.-M. Philippe, B. Tain, and C. Gamrat, "A self-reconfigurable fpga-based platform for prototyping future pervasive systems," in *Evolvable Systems: From Biology to Hardware*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6274, pp. 262–273.

[6] A. Akoglu, A. Sreeramareddy, and J. G. Josiah, "Fpga based distributed self healing architecture for reusable systems," *Cluster Computing*, vol. 12, pp. 269–284, September 2009.

[7] X. Y. Niu, K. H. Tsoi, and W. Luk, "Reconfiguring distributed applications in fpga accelerated cluster with wireless networking," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 545 –550.

[8] F. Maraninchi and Y. Remond, "Mode-automata: a new domain-specific construct for the development of safe critical systems," *Science of Computer Programming*, vol. 46, 2003.

[9] D. Harel, "Statecharts: A visual formalism for complex systems," 1987.

[10] E. Borde, G. Haik, and L. Pautet, "Mode-based reconfiguration of critical software component architectures," in *Design, Automation Test in Europe Conference Exhibition*, ser. DATE '09, 2009, pp. 1160 –1165.

[11] S. Guillet, F. de Lamotte, E. Rutten, G. Gogniat, and J.-P. Diguet, "Modeling and formal control of partial dynamic reconfiguration," *Reconfigurable Computing and FPGAs, International Conference on*, vol. 0, pp. 31–36, 2010.

[12] G. Delaval, H. Marchand, and E. Rutten, "Contracts for modular discrete controller synthesis," *SIGPLAN Not.*, vol. 45, pp. 57–66, 2010.

[13] M. Rummele-Werner, T. Perschke, L. Brauna, M. Hubner, and J. Becker, "A fpga based fast runtime reconfigurable real-time multi-object-tracker," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2011.

[14] J. Huang, M. Parris, J. Lee, and R. F. DeMara, "Scalable fpga-based architecture for dct computation using dynamic partial reconfiguration," *ACM Transactions on Embedded Computing Systems*, vol. V, pp. 1–18, 2008.

[15] Xilinx, "7 series fpgas overview, advance product specification," Tech. Rep. DS180 (v1.8), 2011.