Closed-Loop–Based Self-Adaptive Hardware/Software-Embedded Systems: Design Methodology and Smart Cam Case Study

JEAN-PHILIPPE DIGUET, YVAN EUSTACHE, and GUY GOGNIAT, Lab-STICC, CNRS/
Université Européenne de Bretagne - UBS

This article presents our methodology for implementing self-adaptivness within an OS-based and reconfigurable embedded system according to objectives such as quality of service, performance, or power consumption. We detail our approach to separate application-specific decisions and hardware/software-implementation decisions at system level. The former are related to the efficiency control of applications and based on the knowledge of application engineers. The latter are generic and address the choice between various hardware and software implementations according to user objectives. The decision management is implemented as an adaptive closed-loop model. We describe how each design step may be implemented and especially how we solved the issue of stability. Finally, we present a video-tracking application implemented on a FPGA to demonstrate the effectiveness of our solution, results are given for a system built around a NIOS soft-core with μCOS II RTOS and new services for managing hardware and software tasks transparently.

Categories and Subject Descriptors: C.3 [**Real-time and embedded systems**]: Miscellaneous

General Terms: Design

Additional Key Words and Phrases: Self-adaptive embedded systems, HW/SW codesign, control-theory, QoS, RTOS, image processing

ACM Reference Format:
Diguet, J.-P., Eustache, Y., and Gogniat, G. 2011. Closed-loop–based self-adaptive Hardware/Software-embedded systems: Design methodology and smart cam case study. ACM Trans. Embedd. Comput. Syst. 10, 3, Article 38 (April 2011), 28 pages.
DOI = 10.1145/1952522.1952531 http://doi.acm.org/10.1145/1952522.1952531

1. INTRODUCTION

1.1. Context

The design of embedded systems usually results from a trade-off between antagonist constraints, which are related to design cost (man.months), chip area, power consumption, and real-time or high QoS[1] considerations. The use of a RTOS has become widespread for services it offers in terms of management and implementation of concurrent applications, real-time constraints, and network accesses. When high computing demands are required for application domains, such as image processing, cryptography, source, and channel coding for instance, the final architecture is usually heterogeneous and based on a general, purpose processor (GPP) enhanced with coprocessors dedicated to specific tasks. Some design methodologies, based on design space exploration and

[1]Quality of service.

Authors' address: Jean-Philippe DIGUET, Lab-STICC: UMR 3192 CNRS/Univ. Europenne de Bretagne - UBS BP 92116, F-56321 Lorient Cedex - France.

DOI 10.1145/1952522.1952531 http://doi.acm.org/10.1145/1952522.1952531

power, performance, cost estimation, and tools have been proposed Dave et al. [1999] to solve this issue.

However, the observed trends show that embedded systems tend to have very variable features. The question of unpredictability is first related to the evolution of applications, the optimizations of which are usually based on adaptations to data, leading to strongly data-dependent resource use. The second source of uncertainty is the concurrency of applications sharing resources such a caches, peripherals, and communication media. The traditional approach consists in considering worst cases that lead to overestimations in contradiction to design constraints. With efficient reconfigurable architectures, we can implement the minimum architecture needed to provide the requested performances without excess hardware. This means a minimization of memory accesses and activated hardware, so a reduction of wasted computation.

In this context, self-adaptivity is a promising way to solve the question of optimization at runtime. The first condition is the availability of dynamically reconfigurable architectures. We can reasonably make the assumption that such architectures, which already exist, will be more efficient in the near future in terms of power and reconfiguration time. The second condition is decision capability. This decision can be implemented remotely and provides interesting opportunities for reducing design cost, while offering new services, such as hardware bug fixing or version upgrade. Actually, another challenging issue is the embedding decision within embedded systems to perform hardware (HW)/software (SW) partitioning at runtime. Today, there is no really efficient solution, which can be applied to the design of general embedded systems. This is precisely the point we are addressing in this article.

We consider an architecture model, which fits with a large set of embedded systems. It is based on a GPP with dedicated accelerators that can be dynamically configured. The GPP implements a RTOS managing a set of tasks. Tasks communicate through messages passing for synchronization and shared memories for data, and they can have various possible implementations in HW and/or in SW, which corresponds to different cost/performance trade-offs.

1.2. Objectives and Contributions

Considering the previously described architectural model, the main objective of this work is the implementation of a configuration manager with runtime decision and configuration control capabilities. We focus our work on the following issues, which are the main relevant points from a research perspective.

(1) Separation between decisions to be done at design and at runtime, namely between traditional HW/SW partitioning and online self-adaptivity;
(2) Transparent use of various HW and SW implementations for a given task;
(3) Separation between application-specific and application-independent configuration decisions;
(4) Online decision algorithms and implementation;
(5) Reconfiguration control that must guarantee a safe transition between subsequent configurations;
(6) System stability and avoidance of reconfiguration;
(7) Negligible self-adaptativity cost: no power, area, and performance overhead.

We have concentrated our work on the previous points with the aim to remain, as much as possible, independent from HW platforms and consequently from current partial and dynamic reconfiguration techniques. From a research perspective, our aim is to propose a solution for self-adaptivity based on the assumption that efficient reconfigurable devices are available. Approaches relying on coarse-grain reconfigurable architectures and multiple-context fast switching are some examples of such solutions. So,

we did not base our work on the specific Xilinx framework for partial and dynamic reconfiguration, but we simulate fast reconfiguration with preimplemented HW modules that are clock-gated when unused. We experiment Xilinx solutions in other projects, one of them is for instance dedicated to the domain of networked reconfigurable devices [Bomel et al. 2008].

Our contributions focus on the seven previous items. We propose a design methodology regarding issue 1, we introduce the concepts of unified configuration and control interface (UCCI) and legal representant (LR) to cope with issue 2. Question 3 is solved by means of a hierarchy of local and global configuration managers (LCM/GCM). Our choice for addressing problem 4, regarding self-adaptivity, is based on a closed-loop approach and a Borda vote. We use data-granularity checking and configuration ID broadcasting to deal with problem 5. A proportional integrator (PI) regulator, enhanced with least mean square (LMS) observer, is implemented as a solution to question 6. Finally, point 7 has been our constant optimization criteria.

In this article, we mainly focus on points 1, 3, 4, 6, and 7; details of items 2, 5 can be found in Eustache and Diguet [2008]. First, the whole methodology is presented in Section 3 and a brief description of the configuration management is given in Section 4. Section 5 presents, in detail, the decision concept implementing self-adaptivity. In Section 6, we demonstrate our approach and show how our methodology has been successfully applied for implementing an object tracking application on an FPGA-based smart-cam. The next section first presents the state of the art—the objective is not an overview of the large domain of reconfigurable architectures but how the online reconfiguration decision has been addressed.

2. RELATED WORK

In this section, we focus on the previously quoted contributions. They are not related to reconfigurable architectures in general, but to question of configuration decision from a global perspective including HW and SW aspects. In the domain of adaptive embedded systems, three issues are related to our project: online hardware configuration decision, control-theory for implementing configuration decision, and RTOS for HW and SW task management. This article targets the two first points.

First, a lot of work has been produced in the domain of adaptive architectures. Different techniques have been introduced for clock and voltage scaling (DVS) Wong et al. [2003], pipeline control Manne et al. [1998], cache resource Albonesi [1999], or functional unit Maro et al. [2000] allocation. However, these approaches can be classified in the category of local configurations based on specific aspects, whereas our aim is to provide a global solution to the question of configuration management including both algorithmic and architectural aspects. In Liang et al. [2004], the association between algorithmic and architectural views is relevant for H-264 implementation; however, the hardware controller remains specific and local.

Recently, we observed some first proposals about reconfiguration decisions in embedded systems. In Goddard et al. [2006] a three-level manager is presented. This manager and the associated design space exploration are specific and dedicated to cognitive radio applications based on blind waveform detection. A single decision layer is described and simulated with Matlab in Kaufmann and Platzner [2008]; it is based on an evolutionary algorithm and limited to a single task application and very simple architecture transformations (adder, hash function). Our approach has been driven by the separation between application-specific and system-level decisions, by the choice of a generic methodology and an architecture compliant with energy-limited embedded systems.

Second, controltheory has been used for online decision mainly from a SW point of view. Feedback control has been introduced in the area of soft RTOS to handle the

uncertainty of worst-case execution time (WCET). In Lu et al. [2002], the authors present a complete model for feedback control real-time scheduling. In Li and Nahrstedt [1999], a relevant two-step approach is proposed. However, this kind of technique does not fit for embedded low-cost systems and reconfigurable HW, but the approach of system stability remains interesting. Recently, Gu and Chakraborty [2008] have proposed an efficient DVS technique based on control-theory for PDA power management. The approach, implemented on a Xscale processor running Windows mobile, provides accurate workload predictions in the context of 3D game applications. The PID predictor is then used to decide the voltage/frequency configuration. In this work, control theory is dedicated to a given application and a single system-level knob (DVS).

Our approach has been driven by the necessity of associating application and system-level decisions in a hierarchical, but global, framework with a decision algorithm compliant with embedded system constraints. Contrary to previous work, we consider the whole system as an entity to be controlled.

Finally, the configuration decision and management can be considered as new OS services that rely on other features such as the ability to manage both HW and SW tasks. This aspect has been explored in several projects such as OS4RS [Mignolet et al. 2003], which abstracts task implementations in HW or SW. Bergmann et al. Bergmann et al. [2006] present a solution based on a μcLinux platform implemented on the Xilinx Micro-Blaze soft-core processor, where hardware modules are considered as usual processes with their own address space. This work mainly focuses on HW/SW interprocess communications (IPCs) and provides a transparent use of UNIX pipes, which are efficiently implemented with MicroBlaze FIFO (FSL). The BORPH project [So et al. 2006] introduces a unified interface for SW and HW threads that extend a standard Linux environment, and it is based on the use of Linux pipes for implementing IPC no matter their HW or SW implementation. Another approach proposed in Andrews et al. [2006] is also based on a standard POSIX interface; SW threads and HW threads are managed through a common API called *Hthread* and some time-consuming OS services are implemented into HW. Finally, in RECONOS [Lübbers and Platzner 2007], eCos is selected for the same reasons we have chosen μCOS II, both are low footprint real-time OS. They also propose a common HW interface for HW modules, which are present in the OS task table through SW *delegates* that seems to be equivalent to what we call *legal representatives* and to the concept of *ghost processes* in Bergmann et al. [2006].

We propose a solution that aims at unifying HW and SW task management within an extended OS. In this context, our main contribution is precisely related to the configuration decision process that automatically manages system configuration according to user requirements. This decision process is closely linked to enhanced OS services that are adapted, in our case, to configuration modeling including resource allocation and modified communication and synchronization schemes. From a pure OS point of view, we did not only consider IPC for HW/SW process, our aim is to offer HW tasks access to all OS services. Contrary to Lübbers and Platzner [2007] we distribute IPCs in order to speed up and benefit from concurrency for HW/HW task communications.

In conclusion, even if dynamically reconfigurable coarse-grain architectures have been already proposed, the decision issue for the whole system, including inherent transitions between configurations and stability conditions are usually ignored in the domain of embedded systems. In this area, another strong constraint is cost, which means that implementation of the decision and configuration mechanism must be negligible compared to expected gains in terms of performances and power (and area when partial reconfiguration is available). The stability issue is related to the question of cost and must be jointly considered in order to avoid a repetition of useless and unexpected reconfigurations.

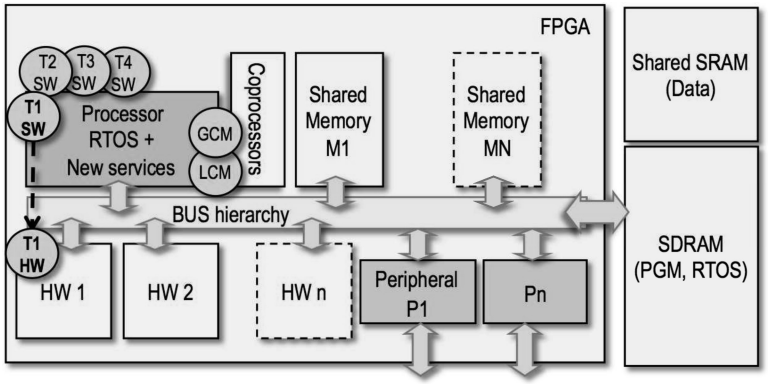


Fig. 1. RTOS-based heterogeneous architecture model for online HW/SW partitioning.

3. METHODOLOGY

3.1. Introduction

The basic problem addressed is the partitioning of a set of tasks over a given architecture model. As for the domain of embedded systems, we consider a model based on a single processor, which can be enhanced with coarse-grain instructions (coprocessors) and hardware accelerators connected to a hierarchy of bus, as shown in Figure 1. The processor implements a RTOS with new services for reconfiguration management, and communications are implemented with shared memories and synchronization signals.

Design space exploration and partitioning decision, especially with heterogeneous HW platforms, form a very complex problem usually solved with heuristics, such as genetic algorithms or simulated annealing. But our objective consists in doing this at runtime. Note that this issue is very different from task migration based on platform-independent SW and homogeneous architecture models, since the point, in this work, is precisely the choice of the HW platform.

Therefore, we have adopted a four-step methodology. The first step is the specification step during which the designer should think in terms of self-adaptivity at functional level. The second step consists in an offline design space exploration (DSE) step, which provides a set of configurations to be selected at runtime by the next two online steps. These two online steps address algorithmic and architectural configurations implemented by local and global configuration managers, respectively.

Note that this approach is compatible with a fully reconfigurable scenario, actually bitstreams stored on chip may be dynamically loaded through a connection to a server providing a remote reconfiguration service as considered in Bomel et al. [2008].

Then, the main objective consists in designing an embedded system able to respect a constraint (namely a reference) specified at runtime by the user, while optimizing secondary magnitudes. In the smart camera example, the constraint can be the QoS and the optimized magnitudes: power and execution time.

Our approach aims to explicitly define roles of SW and HW designers within the whole methodology. The idea is to keep a clear separation, but with a design methodology that provides the HW designer with unambiguous information about application needs in terms of resources.

3.2. Offline Steps: Specification and DSE

3.2.1. Self-Adaptivity Aware Specification. First, we consider that self-adaptativity starts at application design and SW designers are asked to specify the conditions under which

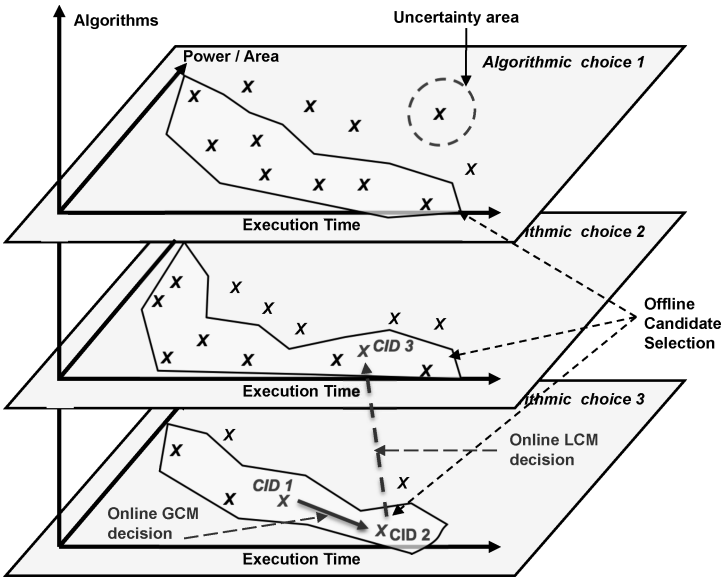


Fig. 2. Configuration space.

Config. ID	LCM Selection	Global Parameters											Local Parameters (Algo. Config)			
		Measures			Data rate	Data Resolution	…	HW/SW Partitioning								
		QoS	P	T				T0	T1	…	TN		T0	T1	…	TN
Ci	✘	Qi	Pi	Ti	Di	Ri		SW	HW		SW		n0i	n1i		nNi
Cj	✔	Qj	Pj	Tj	Dj	Rj		SW	HW		HW		n0j	n1j		nNj

Fig. 3. Configuration table.

an algorithmic configuration must be selected or not by means of application metrics that must be clearly specified and computed if necessary. In practice, the SW designer will follow a methodology in which he will specify lists of algorithmic configurations and transition rules based on metrics. This step is strongly linked to the online adaptation of algorithmic configurations, so an overview of the methodology steps are given in Section 3.3.2.

We believe that our approach is a good way to impose an adaptivity-oriented design discipline on the designer. Moreover, this is not a real design constraint, since this information is usually known by designers at design time, but not explicitly specified.

3.2.2. HW/SW Design Space Exploration. As previously claimed, characteristics of solutions (e.g., power, performances, area) change with data, environment, and architecture hazards. Thus, a selection of promising candidates must be done offline while considering an uncertainty space around each solution, as shown in Figure 2. The description of relevant solutions available on chip is then stored in the configuration table described in Figure 3.

This step can be performed manually or by means of design space exploration tools, providing power, area, and performances estimations that can help to prune the search space. As self-adaptive systems are data-dependent, it is also essential to get test-benches representing data variations or data ranges issued from functional simulations. Moreover, the choice of algorithms by the application designer are usually resource naïve, so feedbacks to the specification step can be necessary if SW implementations are too slow and HW implementation too costly. Regarding the smart camera case study, all tasks have been implemented in various HW and SW versions. Consequently, selected solutions presented in the configuration table rely on real measures.

3.3. Online Steps: Configuration Management

3.3.1. Separation of Concerns. The aim is to implement configuration management for online algorithmic and architectural adaptations. Figure 2 presents the configuration space, where a point means a configuration identifier (CID), and a plan corresponds to an algorithmic choice. Thus, a move within a plan is an architectural reconfiguration (CID1 $\rightarrow$ CID2), and a move between plans means an algorithmic reconfiguration (CID2 $\rightarrow$ CID3). Each line of the configuration table represents a configuration as shown in Figure 3, where global parameters provide the architectural configuration regarding HW/SW partitioning for each task and where local parameters correspond to the algorithmic configuration. The first challenge, when it comes to the configuration decision of embedded systems, is to find a low cost, but efficient solution. The second one is to propose an approach that clearly separate algorithmic configurations, which are intrinsically application specific, and architectural configuration, which mainly deal with trade-offs at system level (e.g., QoS, Performances, and Power).

Our solution, regarding these objectives, relies on a couple of local (LCM) and global (GCM) configuration managers, that can be implemented as SW or HW tasks. There is one LCM for each application and one GCM for the whole architecture. Each LCM is in charge of algorithmic configurations and consequently specified by an application designer. Thus, the LCM first selects the algorithmic configuration without considering implementation issues. The GCM, in charge of architectural configurations, is to be designed by the system designer. Given LCM algorithmic requirements specified notified in the configuration table, the GCM selects the HW/SW implementation that meet specified constraints (QoS, power, performances). Figure 1 shows the smart camera solution, in which LCM and GCM are implemented as SW tasks. The latter is enhanced with wired instructions (coprocessor) implementing basic controller and observer coarse-grain operations.

3.3.2. LCM: Algorithmic Adaptation. The self-adaptivity aware methodology, for application designers, can be organized in five steps.

(1) Task decomposition and specification of possible algorithmic alternatives (plans in the configuration space);
(2) Definition of relevant tasks metrics m_i in the perspective of algorithmic configuration selection;
(3) Functional simulation and specification of algorithmic configurations to be selected according to metric values (LCM rules);
(4) Definition of a global metric providing the GCM with the QoS of the application;
(5) Task and LCM implementation based on communication API for metric delivery.

Thus, the LCM is by definition application specific. It is coded in C as a set of simple logic rules specified by the application designer who knows which are relevant metrics and conditions for algorithmic selection. Here, the worst case corresponds to conditions

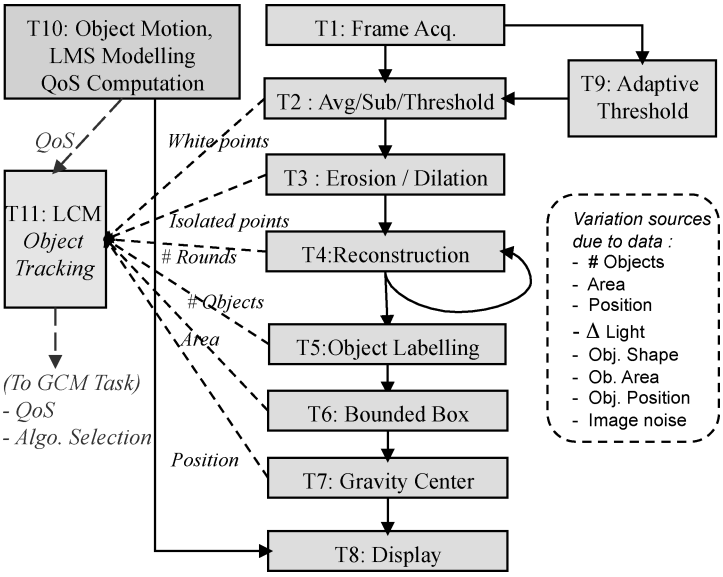


Fig. 4. Object tracking application flow.

where complex algorithms are necessary. For instance, in image processing, the main variables, which drive the algorithmic choices, are the noise levels at different stages of the application flow. These rules are based on metric collected through a set of message passing API, which are used by tasks through a uniform interface described in Section 4. The LCM uses the same API to transmit the QoS metric to the GCM.

The choice of the LCM is then delivered to the GCM though another message passing API: *Apply*(). The selection of the LCM is efficiently implemented by means of masks, which means that the list of valid algorithmic configuration is represented by an array of bits.

Let us consider an example from the smart camera application described in Figure 4. A simple rule consists in activating the adaptive threshold task (T_9) when the noise level is larger then a given threshold (Th_9) specified by the SW designer after a set of PC-based experiments. The noise can be modeled as the difference between two metrics, the first is the number of isolated white pixels (m_3) measured after Erosion/Dilation T_3 and the second is the number of white pixels (m_2) after thresholding (T_2). It will be specified as:

$$if\ m_3 - m_2 > Th_9, then\ Apply(Mask_m[\ldots]),$$

where $Mask_m$ is an array of bits with a length equal to the number of possible algorithmic configurations corresponding to the columns Local Parameters of the configuration table, such as $Mask_m[i] = 1\ if\ CID_i$ is $valid$ and 0 otherwise.

3.3.3. GCM: Architecture Adaptation. The GCM is in charge of architectural implementation decisions. It receives data from sensors (gas gauge, CPU load from the OS, LCM requirements) and from estimators when no measures are available. The GCM decides the new system configuration according to user requirements (e.g., QoS, power, performance references) and configuration solutions issued from the LCM design space restrictions. The decision process is detailed in Section 5.

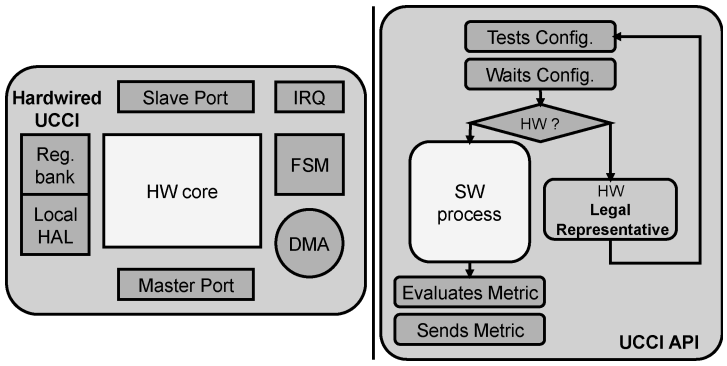


Fig. 5. HW and SW UCCI.

4. CONFIGURATION MANAGEMENT

4.1. Motivations

The reconfiguration of the system at runtime raises two questions. The first is the synchronization of tasks after a reconfiguration has been performed, the second is the problem of interfaces. Both aspects are solved in the context of RTOS, more details can be found in Eustache and Diguet [2008]

4.2. UCCI Interfaces and Legal Representative

We consider applications specified as acyclic task graphs and an architectural model based on message passing and shared memories. This means that a task must indicate to its successors, though mailbox or queuebox mechanisms, the address of data in a shared memory protected with a Semaphore or a Mutex. From a communication point of view, the interface must be unique no matter the implementation of tasks. For this reason, we have developed a UCCI. It is implemented by means of API when the task is running on the processor and as a HDL code container, with hardwired services, when the task is mapped on a HW accelerator. An overview of the UCCI is given in Figure 5 for HW and SW implementations. The interface has to its disposal different API (or equivalent hardwired services), which can be classified in four categories.

—*Synchronization mechanisms*: Abstracted post and pend messages based on mailbox and queue box.
—*Transfer of metrics to the LCM*
—*Configuration mechanisms*: Requests to HAL to get physical addresses, analysis of CID, pending on configuration events, propagation of CID.
—*Memory Accesses*: Update of I/O addresses (SW: variables, HW: registers), soft Mutex for SW tasks and DMA services in case of HW implementation.

The second tedious question is the place of a task from the RTOS point of view. When a task is moving from SW to HW, it still remains alive in the RTOS in a very simple version as a task in charge of RTOS/HW accelerator communications. This concept is called the LR, which is close the ghost process mechanism presented in Bergmann et al. [2006]. Most of the time, a LR is an inactive task pending requests issued from the HW task when a RTOS service is required.

Communications between HW accelerators are direct and, therefore, do not solicit the RTOS. Actually, RTOS communication services are distributed in such a case, since they are implemented in HW within the UCCI.

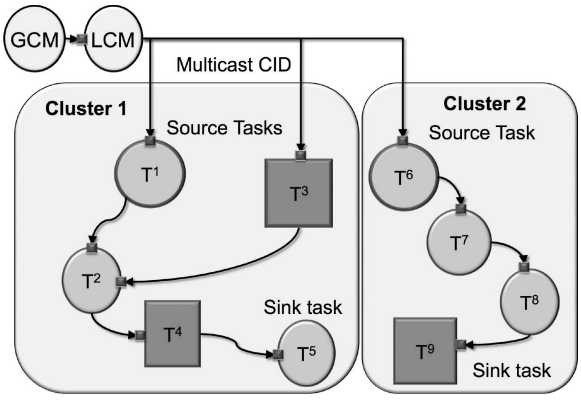


Fig. 6. Configuration synchronization.

4.3. Configuration Control Mechanisms

We solve the synchronization issue by means of diffusion mechanisms as illustrated in Figure 6. Our method reuses existing communication channels, which can be direct for HW to HW communications or based on RTOS services (Pend/Post) for HW/SW and SW/SW communications. These communications are implemented with HW or SW API provided by UCCI.

We, therefore, have developed the following strategy. First, the configuration manager, namely the LCM, sends the CID to all source tasks through a multicast diffusion. Second, the CID is propagated gradually from the source to the sink tasks over data channels, after granularity control to avoid data starving and inconsistency.

The configuration flow, for each tasks is then ordered as follows.

(1) When receiving a new CID, a task runs until it has produced a given amount of data specified offline;
(2) A simple, mask-based CID analysis provides information about the new configuration. Without going in details, four main cases can be distinguished.
   —*Task configuration and I/O addresses unchanged*: Nothing to do;
   —*Task configuration unchanged but new I/O addresses*: A request is sent to the HAL using UCCI API to get new I/O information;
   —*New task configuration HW → SW and CID received from a HW task*: A reconfiguration message, using UCCI, is sent by the HW task to its LR. Then, the LR activates the SW task, that first initializes its I/O configuration using UCCI API;
   —*New task configuration HW → SW and CID received from a SW task*: The LR directly intercepts the new CID and activates the SW task, that first initializes its I/O configuration using UCCI API.
(3) The new CID is propagated to the task successors.

With such diffusion principles, we guarantee that all tasks will be configured starting with the source tasks.

4.4. Reconfigurable Task FSM

Finally, the combination of traditional FSM and UCCI services offers a simple method to implement new task states by means of substates, as shown in Figure 7. Thus, a task can wait on data or on configuration parameters and the run state can be decomposed in three levels. First, we add a substate, where a task computes and transmits metrics

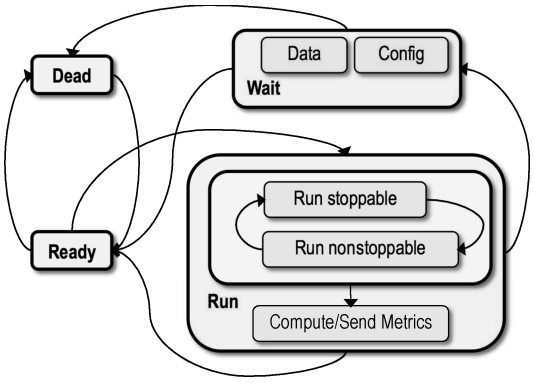


Fig. 7. Configurable task states.

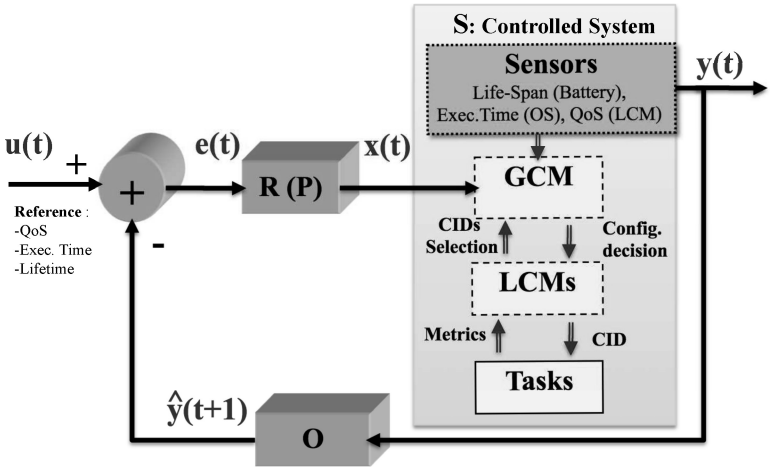


Fig. 8. Generic closed-loop system.

to the LCM. Then, while data are being processed, a task can be in a stoppable substate if enough data have been produced or in a nonstoppable substate otherwise.

5. CONFIGURATION DECISION

5.1. Introduction

We propose an original approach based on a closed-loop model that consists of considering a reconfigurable embedded system as a process to be controlled by means of configurations choices. In the following, we present the model we have adopted and relative issues concerning stability and convergence.

5.2. Closed-Loop Configuration Control

Control theory methodology first requires settling an analytical model close to the real system to be controlled. In our case, the system is composed of a reconfigurable SoC (RSoC) running a set of tasks, which can be implemented with various versions on different HW/SW resources, as well as control, estimation, and configuration tasks. Our model, depicted in Figure 8 is based on three elements. S is the controlled system composed of configuration managers, a task set, and some sensors that provide access to the controlled magnitude $y(t)$.

R is the control function. O is the system observer, which provides estimates for the next time slot. The observer implements a system model that is updated when measures are available.

$u(t)$ is the user reference, depending on priorities a designer can consider. It can be, for instance, application QoS constraint that will be compared to the QoS value provided by the LCM related to the application. It could also be a lifetime threshold that will be compared to a value derived by the GCM from battery level and power consumption or even a time constant that will be compared to real execution times provided by the OS.

In the following, we consider the power P as the controlled magnitude $y(t)$ for the sake of clarity.

Thus,

$$e(t) = u(t) - \hat{y}(t)$$

is the difference between the reference and the observer prediction output, namely the expected average power consumption of the system in the next time slot based on the value provided by the battery controller.

$$\hat{y}(t+1) = a_0 y(t) + a_1 y(t-1) + a_2 y(t-2)$$

produces an estimate of the next average power consumption. In Section 5.5, we detail the estimation model, that is, how the coefficients $\{a_i\}$ are updated.

5.3. Ideal System

In a perfect reconfigurable system, as shown in Figure 8, an infinite configuration space is available, which means that a configuration canceling the error can always be found. In such a case, we could consider the following solution, where x(t) is the output of the proportional regulator R(P).

$$x(t) = k_p e(t)$$

We do not use the derivative effect, which can lead to sudden output variations with this kind of very fluctuating input, nor the integral effect, since it appears in the system modeling as explained hereafter.

y(t) is the output, namely the power consumption, of the system after the reconfiguration adaptation (S). This function introduces an integrator effect that can, in theory, annihilate the steady state error.

$$y(t+1) = y(t) + x(t)$$

5.4. Model for Implementation

In reality, the number of configurations, of course, is limited, which means that a gap will always exist between the controlled magnitude and the user reference. So, we consider the model depicted in Figure 9 for implementation with two modifications regarding the previous ideal view.

The first point is relative to system control. The objective is then to find the closest solution below the user reference. This reference will be p_{max}, since lifespan is the considered constraint in the following. The reference would be the maximum execution time in case of hard or soft real-time applications and the maximum prediction error, namely the QoS, in the context of object tracking.

We, therefore, consider a modified proportional and integrator regulator (PI) with the following definition.

$$x(t) = k_p e(t) + k_i x(t-1) \tag{1}$$

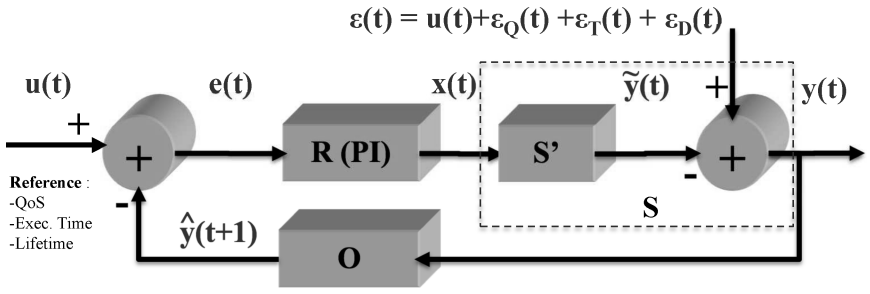


Fig. 9. Implemented closed-loop model.

A classical PI model[2] is not adapted for three reasons. First, the objective is not to annihilate the steady state error but to obtain a gain close, but lower than one. Second, reconfiguration is costly in the domain of embedded systems, so we target a stable configuration that absolutely avoids ceaseless reconfigurations around a magnitude that cannot be reached, since no configuration can be associated. Actually, our solution avoids error accumulation contrary to usual form. Third, this solution is very simple in terms of computing and storage resources.

The second modification is related to the way we model the controlled system S. Basically, the system decides a new configuration with a CID equal to j such that:

$$y_j(t+1) = \max_k(y_k(t+1) \mid y_k(t+1) \leq u(t) + x(t)). \tag{2}$$

Ideally, we should obtain $y(t+1) = u(t)+x(t)$, where the reference is a constant regarding the decision procedure. Thus, we can first assume that the system model is trivial with a continuous configuration space and a reference normalized to zero. In this case, we obtain $\tilde{y}(t+1) = x(t)$. Second, we can consider that the difference between $\tilde{y}(t)$ and $y(t)$ is finally obtained, and can be modeled as a random perturbation $\epsilon(t)$.

$\epsilon(t)$ is composed of a constant value, which is $u(t)$, and three error signals. The first error source ϵ_Q is the quantification effect of the finite configuration table. The second error source, ϵ_T is the load of noncontrolled sporadic tasks that can run on the processor simultaneously and affect performance and power values. The third error source is the data dependency ϵ_D, which implies that the execution delay and power consumption fluctuate with the data values. This aspect is particulary true with applications based on video frame analysis or 3D graphics. Thus, in practice, we obtain

$$y(t) = \tilde{y}(t) + \epsilon(t), \tag{3}$$

where $\epsilon(t) = u(t) + \epsilon_Q(t) + \epsilon_D(t) + \epsilon_T(t)$.

Closed-loop feedback approaches work well for controlling such a changeable environment. If we compute the ratio between the input gain $G(t) = \tilde{y}(t)/u(t)$ and the perturbation gain $G_\epsilon(t) = \tilde{y}(t)/\epsilon(t)$, we obtain a value independent from the observer design

$$\frac{G(t)}{G_\epsilon(t)} = \frac{k_p}{1-k_i}. \tag{4}$$

This ratio can be tuned to reduce the sensitivity of the system to perturbation effects, that is, to the system imperfections.

[2]Usual PI form: $z(t) = z(t-1) + k_p\left(e(t) - e(t-1)\right) + k_i e(t-1)$.

5.5. Observer Design

The observer regularly updates a model that estimates the system behavior. Frist, the aim is to predict the magnitude evolution in order to anticipate the right decision for reconfiguration. Second, sensor acquisition introduces delay and power overheads when a model-based approach enables rapid estimates of the system behavior even when new measures are not available. Basically, it enables the trade-off between accuracy and cost to be adjusted. $y(t)$ can be considered as a noisy signal. The model objective is to predict $y(t+n)$ representing, for instance, the average power consumption of the system at time $t+n$. Various digital signal processing techniques have been defined to solve this kind of problem. However, in the context of embedded systems, the aim is to save power, so the control technique overhead must be lower than the expected gains. It means that only low-cost solutions can be implemented. For these reasons, methods like adaptive least square or Kalman filter are prohibited. Considering the algorithm complexity for adaptation and estimation and the filter length, we have opted for a 3-tap LMS for the observer implementation. Under the assumption that ϵ is a gaussian noise, the stability of the algorithm is guaranteed if K is such as given in Equation (6).

$$\begin{aligned} e_p(t) &= y(t) - \hat{y}(t) \\ \forall i \in \{0,1,2\} \;\; a_i(t+1) &= a_i(t) + \mu * y(t-i) e_p(t) \\ \hat{y}(t+1) &= \sum_{i=0}^{2} a_i y(t-i) \end{aligned} \tag{5}$$

$$0 < \mu < \frac{2}{3 * N_{Taps} * \sigma_\epsilon^2} \tag{6}$$

5.6. Stability Analysis

Equation (6) indicates a classical condition ensuring the stability of the dynamics of the mean of the LMS estimator described in Section 5.5. However, the closed-loop system remains potentially unstable. Conditions of stability must be guaranteed for all a_i possible values; hereafter, we develop the closed-loop function in the Z domain and set the conditions of stability for the feedback control.

Bloc functions

$$\begin{aligned} R(z) &= \frac{k_p}{1 - k_i z^{-1}} \\ S'(z) &= z^{-1} \\ O(z) &= a_0 + a_1 z^{-1} + a_2 z^{-2} \end{aligned}$$

Closed-loop function

$$H(z) = \frac{G(z)}{1 + O(z)G(z)} \tag{7}$$

$$G(z) = \frac{k_p}{z - k_i} \tag{8}$$

From Equation (7), we can derive necessary conditions for stability. Results are summarized in the first column of Table I for two kinds of observers, the first one is the identity function, and the second one is a LMS described in the next section. We also provide, in the second table column, the condition for obtaining a gain $H(\infty)$ lower than one in order to maintain the controlled magnitude under the reference, as explained in Section 5.4.

Table I. Closed-Loop Model Stability Conditions

Observer $\{a_i\}$	Stability	Gain $H(\infty) < 1$
$\{1, 0, 0\}$	$\lvert k_i - k_p\rvert < 1$	$\lvert k_p\rvert < \lvert 1 + k_p - k_i\rvert$
LMS	$k_i - k_p(\sum_i a_i) < 1$ $\lvert a_2 k_p\rvert < 1$ $k_i - k_p(a_0 - a_1 + a_2) > -1$ $\lvert (a_2 k_p)^2 - 1\rvert > \lvert a_2 k_p(a_0 k_p - k_i) - a_1 k_p\rvert$	$\lvert k_p\rvert < \lvert k_p(\sum_i a_i) + 1 - k_i\rvert$

5.7. Decision Implementation

The aim of the decision is to select, regarding a regulated error, the best configuration from the configuration table, which is regularly updated with real measures. Our approach has been driven by a trade-off between efficiency and complexity compliant with embedded systems.

5.7.1. Model. The problem can be formalized as follows: Given i the considered magnitude (Power, QoS, T) and j the CID, $Tab(i, j)$ is the value stored in the configuration table. $X(i, j|k)$ is the estimated controlled error for magnitude i in configuration j, knowing value $X(i, k)$ for current configuration k. For instance, if i represents power, the following linear approximation is used

$$X(i, j|k) = X(i, k)\frac{Tab(i, k)}{Tab(i, j)}$$

$Th(i)$ is a possible tolerance regarding reference and $U(i)$ is the reference set for the magnitude i. All magnitudes are defined in such a way that a configuration meets the constraints if $V(i, j) > 0$.

$$V(i, j) = U(i) + Th(i) - (Tab(i, j) + X(i, j|k)) \tag{9}$$

5.7.2. Decision Algorithm. The main steps of the decision algorithm are given in Figure 10. The frequency of metric transfers is controlled by the configuration period. It means that metrics are transmitted to the LCM after N_e consecutive executions of the application, it means $k.N_e$ executions of the task if k is the number of task iterations within the application period.

Second, the GCM is also pending on a mailbox, waiting for data issued from LCMs regarding algorithmic configurations, meaning that a first decision reduction is obtained through LCM selection. Then, a second restriction is introduced based on a *paying off* delay t_k during which costly hardware reconfiguration is not authorized. T_k corresponds to the minimum delay required to accept the reconfigration overhead compared to expected benefits. $T_k = max\left(\frac{T_R}{G_T}; \frac{E_R}{G_E}\right)$. Where T_R is the reconfiguration delay, G_T the performance gain between the new and the previous configuration, E_R the energy required for a reconfiguration, and G_E the energy gain. Like other configuration parameters, expected gains are estimated, since they are based on fluctuating variables. Time and power consumption are updated in the configuration table but the reconfiguration time may also vary depending the architecture, if for instance a configuration cache is used. Finally, all considered magnitudes are assessed for the final selection regarding a given priority order (e.g., T, QoS, P).

Then, the algorithm runs as follows. First, note that only the first constraint is regulated (e.g., QoS) and considered for selection based on the following condition: $V(1, j) > 0$. Second, other constraints are considered when more than one solution respects the first condition. Otherwise, the candidate providing the smallest error is selected regarding only the first constraint. An important point here is that our objective is to respect user references. It means that in case of multiple solutions, we can relax the constraint on the first reference in order to globally optimize all system

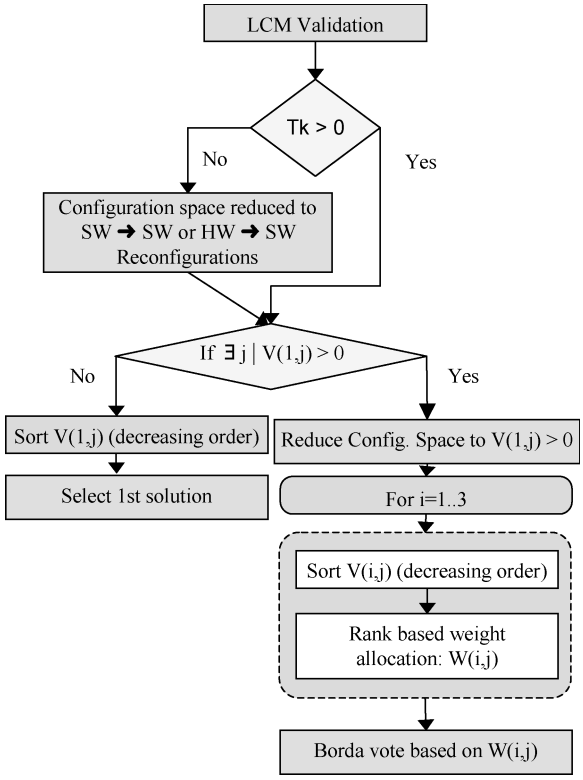


Fig. 10. Decision algorithm.

magnitudes. This optimization is implemented with a simple but efficient vote based on De Borda's method [De Borda 1781], which is processed among survivor solutions. Each magnitude sorts remaining configurations and gives a vote corresponding to the rank. A negative vote means that the constraint is not respected. The closest solution respecting the constraint gets the highest vote. Different weights can be assigned to the different magnitudes if the designer wants to introduce priorities. If multiple candidates obtain the same score, then a Hamming distance with current configuration is used to select minimal SW $\rightarrow$ HW moves. At the end, the GCM selects the configuration with the highest rank and transmits the corresponding CID to the LCM.

5.7.3. Configuration Period. The issue of a sampling period is a critical aspect of controlled systems. In our case, we consider models linear with application iterations; it means that the sampling period is equal to N_e application iterations as depicted in Figure 11, where N_e is an integer and global configuration parameter. It means that metrics are collected every N_e iterations by LCM and GCM that then decide the next configuration. When necessary, this approach can help to minimize the control overhead in terms of communication and computation.

6. OBJECT TRACKING TEST-BED

Before getting into details, let us briefly go back over our objective according to Section 1.2. The goal is to formalize and implement a stable, safe, and nonoscillating self-adaptation mechanism that does not introduce any significant area, power, or performance overhead. Moreover, this mechanism must be easy to implement for both HW

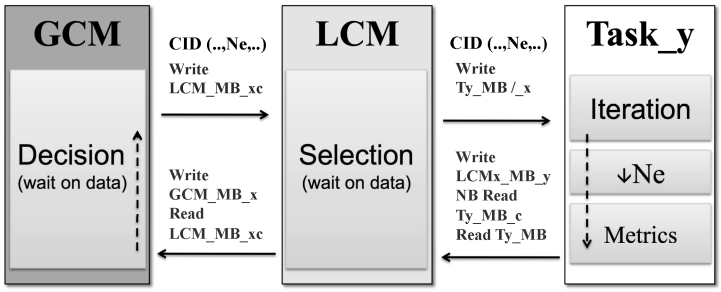


Fig. 11. Control sampling and task period.

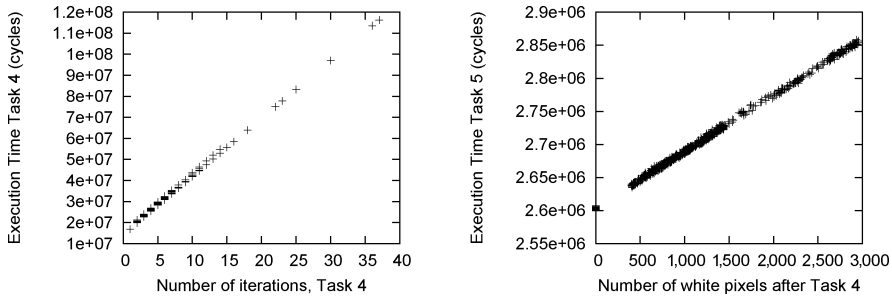


Fig. 12. Example, T_4 metric selection based on application-designer experience and simulation.

and SW designers. The goal of self-adaptation is to converge towards a configuration that respects ordered user references (e.g., QoS, Power, or Performances). The best way to prove the efficiency of our approach, in the absence of equivalent and available approaches to compare with, was the validation of our method on a real-life application, in which self-adaptivity makes sense. In our case, this is an FPGA-based smart camera implementing an object tracking application. In the following, we present how the different design steps of our methodology are applied.

6.1. Application Specification, Metric Selection

The application is composed of 10 tasks (T_1 ... T_{10}) described in Figure 4, which can be implemented in HW or in SW, these tasks are controlled by a LCM implemented as a SW task. The dotted arrows from tasks to the LCM "tracking" indicate metrics to be used for algorithmic configuration. For instance, the number of isolated white points after tasks 2, 3, and 4 the numbers of iterations of object reconstructions (T_{10}) or the number of detected objects (T_5). The selection of the task metrics is based on the application-designer experience and simulation analysis. For instance, we present in Figure 12, the evolution of T_4 and T_5 execution times according to T_4 metrics: the number of reconstruction iterations and the number of isolated white pixels.

6.2. Environment Sensors

The architecture is implemented on a NIOS soft core within an Altera Stratix II 2S60ES FPGA board with a VGA daughter board. In addition, we have plugged in a camera and a battery gauge, providing power consumption features, on FPGA GPIOs.

An important point is to be noticed: The image acquisition rate is controlled by the GCM and follows up the application rate. It means that we avoid useless image storage and costly off-chip memory accesses. The extended RTOS, based on a set of API and an adapted HAL, is built around μCos II and provides the information about task and

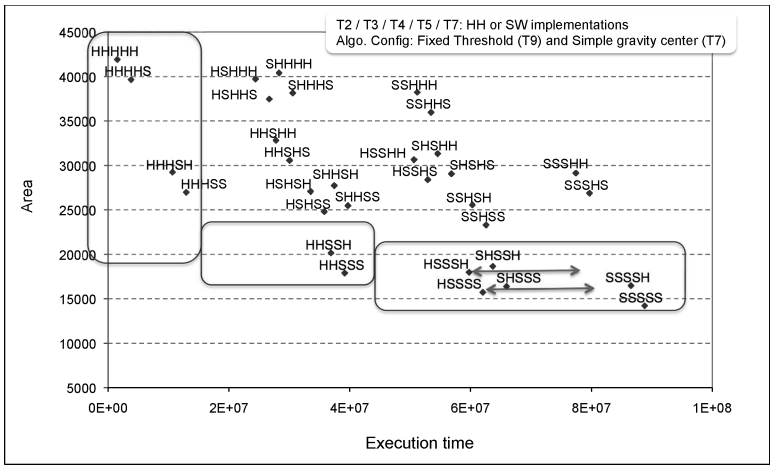


Fig. 13. Smart camera design space exploration.

application execution time. The QoS sensor is realized by Task T_{10} that provides the LCM with a metric, which is the difference between object position based on labeling results and an estimation of object positions based on a LMS algorithm. A value close to 0 but lower than the reference (e.g., 10%) means a very high tracking quality that can be relaxed if the application speed is reduced. However, a value higher than the reference means that the application rate must be increased with a faster configuration.

6.3. Design Space Exploration

The extended RTOS is built with new previously explained capabilities for communication, synchronization, and configuration of HW and SW tasks. HW task modules are connected to the Avalon bus and clocked only when used. A co-processor has been added as a coarse-grain instruction acceded through processor registers for an efficient implementation of the LMS and PI regulator. It is also used for application QoS computation (error between prediction and object position).

Initially, a generic C code of the application was available and various HW modules were designed after a short design space analysis. Major results of real implemented alternatives are given in Figure 13 for a given algorithmic configuration, which is, in this case, the complete application with a fixed threshold (T_9 off) and a gravity center approximated as the center of the bounded box (T_7). As explained in Section 3, measures are fluctuating within a uncertainty area. Thus, data given in Figure 13 correspond to one photography that may vary in time. Thus, the set of selected configurations can include some configurations that don't appear to be Pareto optimal. To illustrate such cases, red arrows indicate the fluctuation of the execution time for two specific configurations.

After this stage, 22 significant configurations are selected with the following algorithmic choices.

—Deep sleep mode: $T_{3,4,5,6,7}$ are inactive;
—Sleep mode: $T_{4,5,6,7}$ are inactive;
—Reconstruction T_4: on or inactive;
—T_2: filter inactive or based on two or four images;
—T_9: on or inactive (fixed threshold).

a) Improve filtering
if $m_4 - m_2 \geq Th_2\_noise\_1$
$NB\_Frames+ = 1$
end if
if $NB\_Frames > 4$
$Adaptive\_Threshold =$ OK for 1 iteration
$NB\_Frame = 0$
end if

b) Reduce filtering
if $m_4 - m_2 < Th_2\_noise\_2$ during N_Frames
$NB\_Frames- = 1$
end if
if $NB\_Frames < 0$
$NB\_Frames = 0$
end if

Fig. 14. Algorithmic configuration, average filtering (T_2).

6.4. UCCI Encapsulation

HW and SW tasks are implemented using UCCI services. Due to length constraint, code details cannot be presented here, but the implementation can be shortly summarized as follows. A SW task is enhanced with three additional stages based on API library. The first one tests if the application is implemented in HW and, if it is true, pends on its control mailbox as the *legal representative* of the HW task. If the CID corresponds to a SW version, then task parameters are updated according to the selected algorithmic configuration. The second stage is implemented after the standard SW task code, this is the metric computation based on task variables. Finally, the third stage is the emission of metric towards the LCM mailbox queues.

A HW task is encapsulated within a HDL container, including communication and configuration supports. The generic shell is adapted with the appropriated number of output and input ports and mailboxes for the control of communication with the OS and other tasks. Data transfers are based on shared memories and dedicated registers are specified, within the UCCI shell, to provide base addresses.

6.5. LCM Implementation

The next step is the LCM specification; in this case, a single LCM is required and a SW version is chosen. Let's first consider I/Os. The right number of mailbox queue instances is defined for the capture of metrics. The number of output mailbox instances is equal to the number of sink tasks, which in this case is 1.

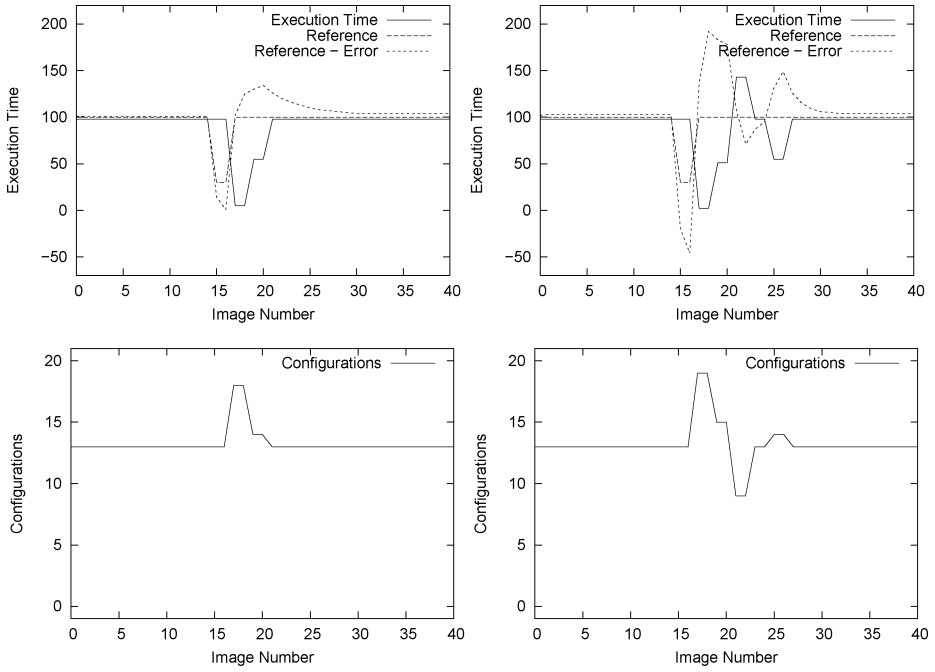
The second point relies on the LCM strategies. Actually, it is currently implemented as rules defined by the application designer according to simulation results. The object tracking application requires six rules. Hereafter, is given, rule 14 for the average filter (T_2), where m_j is the number of isolated white points after task j, $Th_2\_noise\_1, 2$ represents accepted noise levels defined by the SW designer.

Note that the frame acquisition (T_1) is not a periodic task, it is launched only when a frame is required and so is directly dependent on other task configurations. This means that the acquisition rate is controlled and regulated by the GCM.

6.6. GCM Implementation

6.6.1. Introduction. In this section, we focus on the main GCM parameters, which are the PI regulator coefficients: k_i, k_p, and the LMS observer coefficient k_L. A small coprocessor, acceded though processor registers (NIOS custom instruction), has been added to provide basic coarse-grain instructions of PI and LMS algorithms. The cost of the coprocessor is 352 LUT representing 0, 6% of the whole area. Regarding performances, PI and LMS require 55 and 62 cycles, respectively.

Various experiments have been conducted with the smart camera prototype implemented on FPGA. The GCM is implemented as a SW task, so the choice of the coefficients results from a simple variable initialization within the associated C code.



a) $k_p = 0,25$; $k_i = 0,75$ b) $k_p = 0,75$; $k_i = 0,5$

Fig. 15. Execution time regulation: pulse response.

The following sections show how a SW designer can rapidly decide on the correct control parameters as a trade-off between response time and accuracy. This method is equivalent to the way it can be conducted in usual regulated systems.

6.6.2. Observers. Three kinds of observers, corresponding to system magnitudes, are required by the GCM in order to make its choices based on LCM requirements. The first one is the application QoS, which is transmitted by the LCM. In our case study, the QoS is the object tracking error provided by task T_{10} to the LCM as a task metric. The second observer, dealing with execution time, is provided by the RTOS as a simple timer. The GCM starts the application timer when a configuration is decided and the associated CID transmitted and stopped when the QoS metric is received. The third observer is the power consumption delivered by the gas gauge component. The GCM can request data and update the configuration table when a given configuration is running.

6.6.3. PI Specification: Pulse Response Analysis. Figure 15 presents pulse responses for different $\{k_p, k_i\}$ choices, the regulated magnitude is the execution time and the X axis is the time represented as the image number. The reference is initially set to 100 for the first 15 frames, then set to 0 during one frame and set again to 100. For each parameter choice $\{k_p, k_i\}$, we can observe the reference, the execution time $(y(t))$ and difference between the reference and the regulated error $(x(t))$; the configuration selection is given on a separate figure. The increase of the integration factor k_i slows down the adaptation, while the increase of the proportional factor of k_p increases the regulator gain. Since stability constraints are respected, the system returns to the initial configuration in all cases, but the delay and the number of transitional configurations vary according to parameter choices.

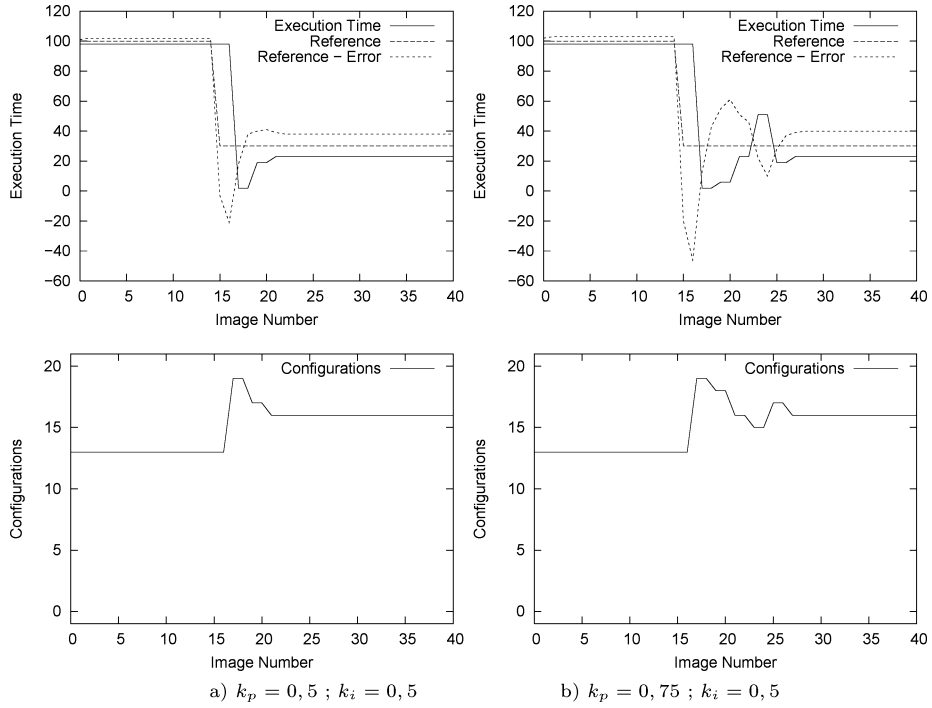


Fig. 16. Execution time regulation (R): step response.

6.6.4. PI Specification: Step Response Analysis. Figure 16 presents step responses with the same experiment conditions described in the previous section. In this set of experiments, the execution time reference is set to 100 until image 15, and then set to 20. Once again, we observe that the transitional phase varies with the parameter choices. Given these features, which can be tested quite easily during the tuning phase, this is a designer decision to specify the PI parameters according to application constraints.

6.6.5. LMS Specification. The third coefficient to be set is the LMS coefficient k_L. The LMS objective is to provide a linear model of the regulated magnitude in order to integrate its evolution within the decision process. The LMS implementation is based on a co-processor, namely a custom instruction, which is initialized with SW instructions.

Figure 17 presents slope responses in the context of execution time regulation and illustrates the LMS effect. The reduction of the LMS coefficient (k_L) slows down the update of the linear model, and we observe a better adaptation with a reduced number of reconfigurations. Both used values for (k_L) are compliant with the LMS stability constraints.

7. TRAIN TRACKING SCENARIOS

7.1. Experiment Conditions

To illustrate the self-adaptivity abilities of the prototype, we propose tracking an electric toy train with a scenario punctuated with various events inducing different configuration decisions. Figure 18 gives the scene captured by the camera. Figure 20 and 19 show the configuration decisions along the execution of this scenario with and without LMS, respectively.

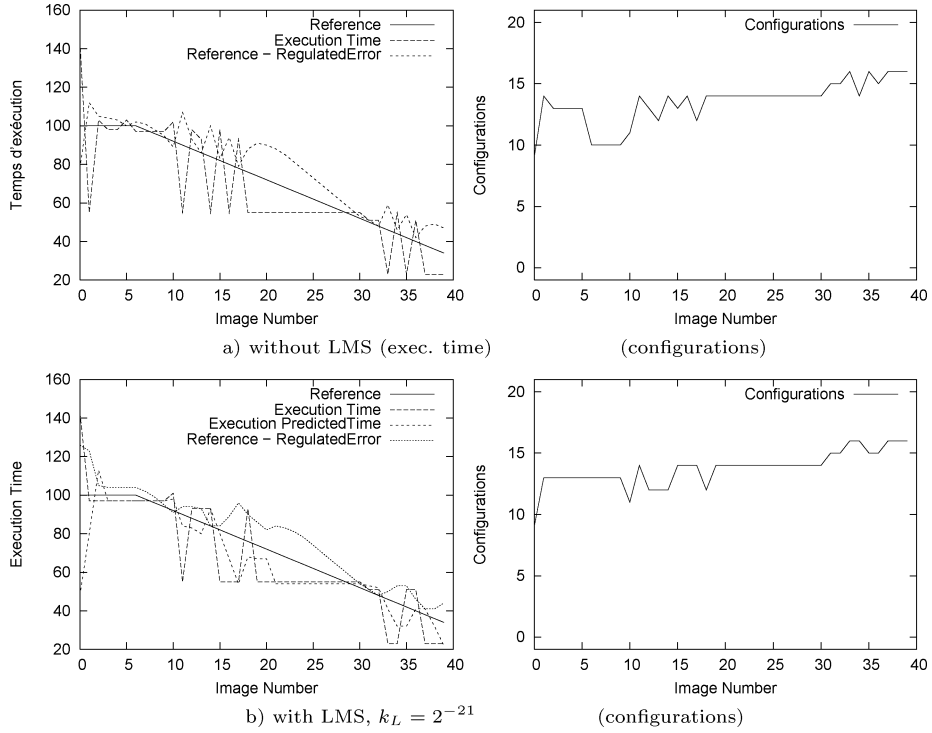


Fig. 17. Execution time regulation: LMS effect.

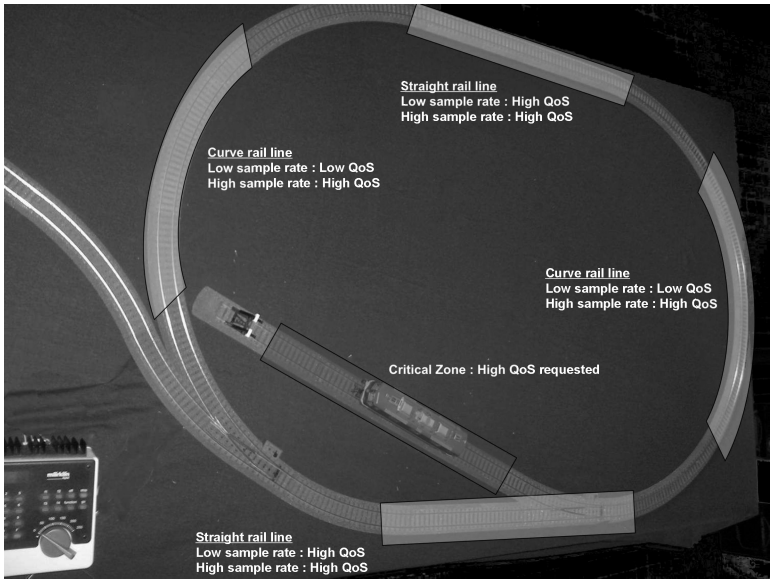


Fig. 18. Train platform camera visual angle.

In this scenario, the regulated magnitude is no longer the execution time but the QoS, namely the tracking accuracy. It means that De Borda's vote is applied to power and execution time values.

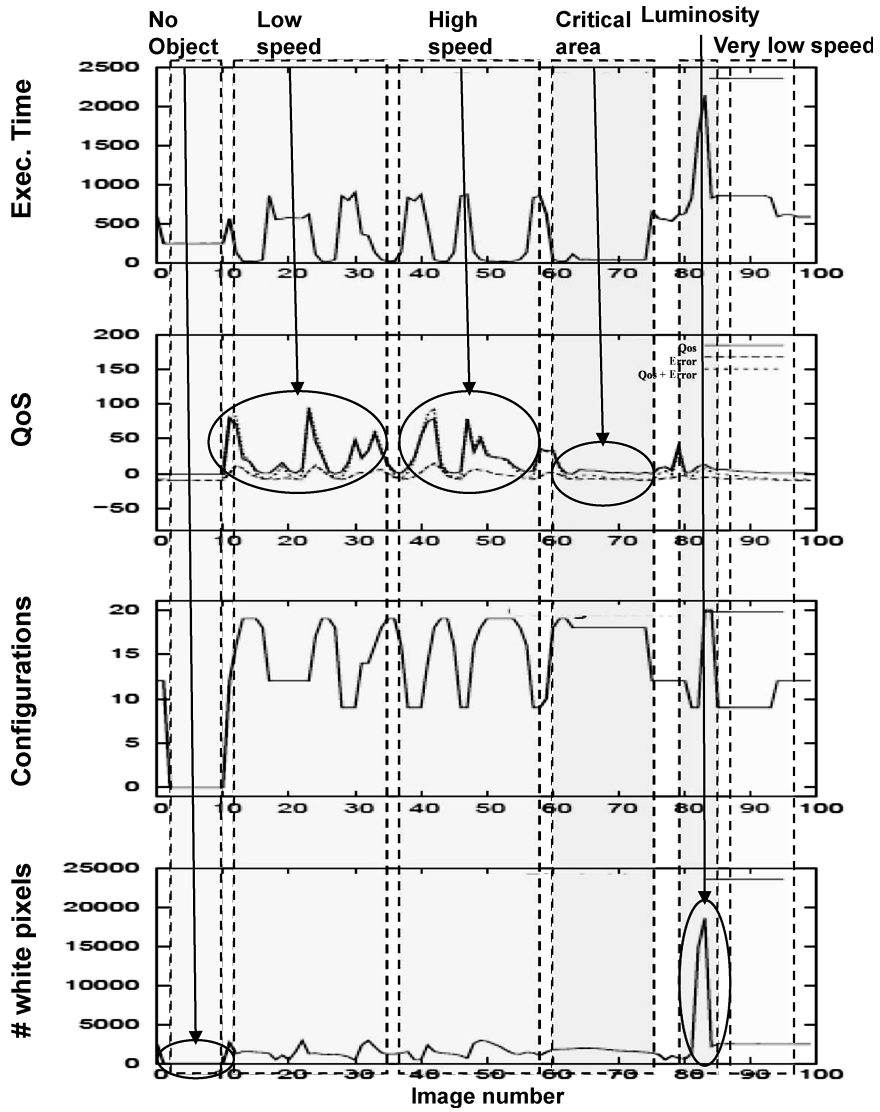


Fig. 19. Self-adaptivity scenario with LMS.

After some experiments and simulations, as is usually done in real automation implementation, we have finally set the following regulator parameters: $\{k_p = 0,25; k_i = 0,25\}$, which provide a good trade-off between stability and reactivity. In the same way, the LMS gain has been set to $k_L = 2^{-21}$. The adaptation rate N_e is set to 1, which means that a new configuration is evaluated after each application iteration.

The QoS reference, namely the tracking maximum error, is set to 10% and reduced to 2% within the critical area.

7.2. Result Analysis

7.2.1. Test Scenario. This scenario is based on a succession of events that highlight self-adaptation capabilities. After several images without any movement, the train

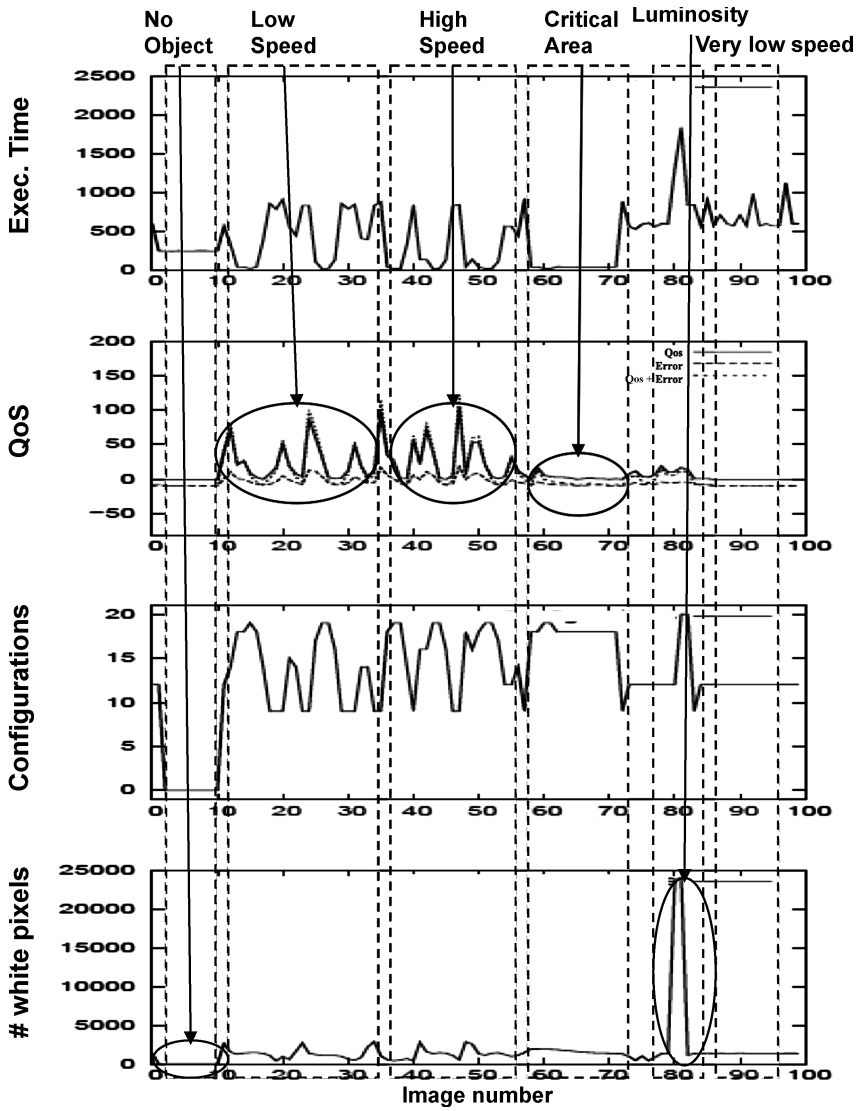


Fig. 20. Self-adaptivity scenario without LMS.

enters the scene at low speed for two circuit rounds. Then, during the stretch of the track, the train speeds up for two others rounds, stops, and goes backward. It then enters a Critical Zone, runs into and leaves the area. Finally, the train continues its path at low speed. All scenario steps are described hereafter and notified in Figure 19 with the LMS support.

Starting point. Initially, no object crosses the scene, the system selects CID #0, where only task 1 is running in SW, the camera frame rate is tuned according to the application execution delay, which is 2 frames/second. This configuration is a "monitoring configuration," where only the first task is active. The regulated magnitude is QoS with a reference set to 10% to guaranty good reactivity.

Table II. Communication Performance Results

	SW ↔ SW	SW ↔ HW	HW ↔ HW
MB Post	517 cy.	2035 cy.	15 cy.
MB Pend	425 cy.	3087 cy.	11 cy.

Object arrival. Then, an object enters the scene at low speed, the system selects CID #12. Due to the initialization phase of the tracking model, a more efficient configuration is selected (CID #19). With all tasks implemented in HW, the tracking computation is running at more than 20 frames/second and the QoS decreases until it goes under the threshold of 10%. Then, De Borda's vote selects the less expensive configuration from CID #16 to CID #9, implementing the last two tasks in SW and all tasks in SW, respectively. Due to curved rail line, the tracking computation is deteriorated and a reconfiguration is requested. At low train speed and low image rate, the damage on QoS has less impact. Moreover, the distance from the camera impacts the tracking efficiency.

Object speed-up. In this example, the train speed is suddenly increased and we observe that the tracking system follows the train during the stretch. However, during the curved lines, the QoS metric skyrockets (error $>$ 100%). The system reacts but no configuration is efficient enough. The CID #19 (all tasks in HW) is nevertheless the best one in such a situation and therefore is selected. The error is reduced until the prediction model is adapted, then a less expensive configuration can be selected.

Critical Zone. We assume that a Critical Zone is defined as where the QoS is increased for security reasons, consequently the reference is set to a lower value, namely 2%. Thus, when the train enters this area we notice that the CID #18 is selected. Due to the decreasing of the QoS reference, only two configurations remain eligible (CID #18 et #19), even if the train stops and QoS converges to zero. As the less expensive, the first one is prefered. When the train leaves the area, the system selects the best configuration from the complete configuration space.

Sudden luminosity change. When the luminosity changes, the threshold no longer fits. For this reason, we notice that the number of white pixels, after the threshold (task T_1) suddenly grows, leads to an increase of the execution time. When such an event appears, an algorithm modification has to be made. A special configuration implements the adaptive computation of a new threshold value, such as a new task based on histogram gradient computation. To readapt the tracking model, the high efficient configuration (CID #19) is selected until the tracking is correct. Then, other configurations may be elected according to De Borda's votes.

Low-speed moving. The scenario ends with a low-speed run around the circuit with CID #12.

7.3. Power and Performance

The CPU time devoted to LCM and GCM task (0.33% in a pure SW solution and 14% when all greedy tasks are implemented in HW) and the HW overhead due to the co-processor (1%) are negligible in such an applicative context where the task granularity is important and reconfigurable architectures make sense. Table II shows the communication performances. The overhead of HW ↔ SW communication is due to context switch and control. One can also observe that we drastically reduce the HW to HW communication time with shared memories directly implemented in UCCI interfaces (shell for HW tasks).

Some architecture configurations may involve significant time overheads (HW and SW tasks switch); however, it is also clear that HW tasks may be considered when computing parallelism is available and relevant speeding up achievable. Such performances can be found in domains such as image processing, encryption, 3D graphics,

Table III. Design Results with a 50MHz Clock

T1,2-T3-T4 T5-T6,7,8	sw-sw-sw sw-sw	sw-hw-hw sw-hw	hw-hw-hw hw-hw
Exec. Time	245.650.000 cy. ±0, 01%	80.800.000 cy. ±0, 04%	1.820.000 cy. ±0, 2%
Frames / sec.	0, 20	0,62	26
Area StratixII S60	19 %	59 %	92 %
Power	137 mW	228 mW	285 mW

Table IV. Fluctuating Execution Time Due to Data

Task	Variation	Exec. Time
Erosion	1 pixel	14.240.816 cy.
	320*240 pixels	146.197.242 cy.
Reconstruction	1 iteration	15.641.863 cy.
	2 iterations	162.476.520 cy.
	3 iterations	172.675.410 cy.

and video encoders. In our case study, for instance, we observe that message passing represents a very low percentage of the entire whole communication.

With different algorithm and architectural configurations, we obtain tracking system performances. Table III provides, for different configuration examples, performance values, FPGA area ratio provided and power consumption measures obtained with the battery gauge (TI Bq2084). Execution time results correspond to a tracking process with a standard input frame. Execution time variation is due first to system architecture (e.g., cache miss, bus collision...) and secondly to data features. Table IV shows examples of such data-dependent performances that can justify the generalization of self-adaptive systems in the future. The reconstruction task, for instance, is a recursive task depending on object complexity; during each iteration, execution time depends on the number of white pixels. In the same way, erosion and labeling execution times depend on the number of white pixels and the number of objects, as well as the number of white pixels and object complexity, respectively.

8. CONCLUSION

Self-reconfiguration is a promising way to improve the efficiency of SoC of MIPS/Watt metric. It also appears to be an economically viable solution for upgrading systems in reaction to HW bugs and fault detection. FPGAs are not yet the sought-after good solution for implementing such systems; however, they propose already available frameworks to study and implement new concepts that need to be designed so that such systems become a reality in the future. Considering the availability of such architectures, in this work, we have proposed a methodology and design solutions to provide embedded system designers with a framework for implementing self-adaptivity when it is necessary for efficiency reasons.

First, we extend the RTOS classical task model in such a way that a given task can be executed transparently with different HW and SW implementations. This evolution also offers HW task access to usual task communication and synchronization schemes. Moreover, its implementation is straightforward because it is based on HW and SW shells that are independent from original C or HDL code, respectively.

Second, we propose an original and lightweight solution for online reconfiguration decisions that can be implemented as a new OS service composed of LCM and GCM managers. This is a learning-based method implemented as a control loop with a simple selection stage. To the best of our knowledge, this is the first time that observer and regulator concepts are applied in the context of RSoC self-adaptation including

both algorithmic and architectural configurations. This study has been conducted in depth including stability and oscillation issues inherent to self-reconfigurable systems. Nevertheless, it is clear that self-adaptivity has cost and gains are obtained if the overhead is negligible compared to the application complexity. Such favorable conditions can be found in domains where task granularity is large enough and parallelism available. This is for instance the case in the domain of smart camera, which has chosen to demonstrate our approach. Finally, a complete proof of concept has been designed with an object tracking application implemented on a FPGA with camera, VGA, and battery peripherals. Self-reconfiguration, according to a QoS reference, has been validated with this demonstrator.

Our model fits with typical architectures of embedded based on a master GPP processor enhanced with (configurable) specific processor and (configurable) hardware accelerators. A possible future work would consist in considering a RSoC with multiple masters competing for system resources.

REFERENCES

ALBONESI, D. 1999. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd Annual International Symposium on Micro-Architecture.* IEEE, Los Alamitos, CA.

ANDREWS, D., SASS, R., ANDERSON, E., AGRON, J., PECK, W., STEVENS, J., BAIJOT, F., AND KOMP, E. 2006. The case for high-level programming models for reconfigurable computers. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems & Algorithms.* CSREA Press, Irvine, CA.

BERGMANN, N., WILLIAMS, J., HAN, J., AND CHEN, Y. 2006. A process model for hardware modules in reconfigurable system-on-chip. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*. Springer, Berlin, 205–214.

BOMEL, P., GOGNIAT, G., AND DIGUET, J-PH. 2008. A networked, lightweight and partially reconfigurable platform. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*. Springer, Berlin.

DAVE, B., LAKSHMINARAYANA, G., AND JHA, N. 1999. Cosyn: Hardware-software co-synthesis of heterogeneous distributed embedded systems. *IEEE Trans. Softw. Engin. 7*, 1.

DE BORDA, J.-C. 1781. Mémoire sur les élections au scrutin (in French). Histoire de l'Académie Royale des Sciences, Paris.

EUSTACHE, Y. AND DIGUET, J.-P. 2008. Reconfiguration management in the context of RTOS-based HW/SW embedded systems. *EURASIP J. Embedded Syst.* (Special Issue on Operating System Support for Embedded Real-Time Applications.)

GODDARD, L., MOY, C., AND PALICOT, J. 2006. From a configuration management to a cognitive radio management system of SDR systems. In *Proceedings of the International Conference on Cognitive Oriented Wireless Networks and Communication Radio.* IEEE, Los Alamitos, CA.

GU, Y. AND CHAKRABORTY, S. 2008. Control theory-based DVS for interactive 3D games. In *Proceedings of the Design Automation Conference.* ACM, New York, 740–745.

KAUFMANN, P. AND PLATZNER, M. 2008. Towards self-adaptive embedded systems: Multi-objective hardware solution. In *Proceedings of the International Workshop on Applied Reconfigurable Computing.* Springer, Berlin.

LIANG, J., LAFFELY, A., SRINIVASAN, S., AND TESSIER, R. 2004. An architecture and compiler for scalable on-chip communication. *IEEE Trans. VLSI Syst. 12*, 7, 711–726.

LU, C., STANKOVIC, J., TAO, G., AND SON, S. 2002. Feedback control real-time scheduling: Framework, modeling and algorithm. *J. Control Theor. Approaches Real-Time Comput.* (Special issue of RT Systems.) *23*, 1/2, 85–126.

LÜBBERS, E. AND PLATZNER, M. 2007. Reconos: An RTOS supporting hard and software threads. In *Proceedings of the International Conference on Field Programmable Logic and Applications.* IEEE, Los Alamitos, Ca.

MANNE, S., KLAUSER, A., AND GRUNWALD, D. 1998. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th International Symposium on Computer Architecture.* ACM, New York, 132–141.

MARO, R., BAI, Y., AND BAHAR, R. 2000. Dynamically reconfiguring processor resources to reduce power consumption in high-performance processors. In *Proceedings of the Workshop on Power-Aware Computer Systems*. Springer, Berlin.

MIGNOLET, J.-Y., NOLLET, V., COENE, P., VERKEST, D., VERNALDE, S., AND LAUWEREINS, R. 2003. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *Proceedings of the Design, Automation, and Test in Europe Conference.* IEEE, Los Alamitos, CA.

SO, H., TKACHENKO, A., AND BRODERSEN, R. 2006. A unified hardware/-software runtime environment for fpga-based reconfigurable computers using borph. In *Proceedings of the 4th International Conference on HW-SW Co-Design and System Synthesis.* ACM, New York.

WONG, J., QU, G., AND POTKONJAK, M. 2003. An online approach for power minimization in qos sensitive systems. In *Proceedings of the Asia and South Pacific Design Automation Conference.* IEEE, Los Alamitos, CA.

Received April 2009; revised August 2009; accepted October 2009