Networked self-adaptive systems: an opportunity for configuring in the large

J-Ph. Diguet, L. Ye, Y. Eustache, J. Crennes, P. Bomel, G. Gogniat, J. Vidal, F. De Lamotte
Lab-STICC, CNRS / Université Européenne de Bretagne, UBS, Lorient, France

Abstract— *Self-adaptivity is a solution to give decision intelligence to embedded systems in order to dynamically adapt HW / SW architectures to environment variations, data changes and user requirements according for instance to energy efficiency. In our approach, this choice is done at run-time and based on a set of embedded configurations. This choice offers fast reconfiguration but can also mean a restricted configuration space regarding multiple application systems. However usual network capabilities can enlarge this configuration space, so we propose a solution based on a hierarchy of configuration servers from which configuration, based on predefined architecture models, can be downloaded. In this paper we present a combination of two techniques to propose a global configuration management.*

Keywords: Self-adaptivity, network, reconfigurable architectures

1. Introduction

The key argument for reconfigurable architecture is probably "obtaining a more effective architecture by means of specialization". This idea is not new if we refer to Estrin paper in 1963 [1], where it was already described as such, however we are now reaching borders, where it can become an on-chip reality. Many multiprocessor, heterogeneous and configurable machines have already been developed in the 90's, they were for instance based on transputers, DSP and FPGAs. Obviously, things have changed with the integration progress on a single chip providing fast communication capabilities that were the main limits of previous attempts. The second aspect of evolution is the possibility to access, at run-time, to configuration memories and consequently to reconfigure architectures dynamically. Combined with the first goal of reconfigurable computing, it enables to consider the question of hardware specialization as one new dimension of programming. Thus, as it is today for compilation and scheduling, the management of configurations may also result from a balance between static (design time) and dynamic (run-time) decisions.

If we look back again in the past, Von Neumann brought the concept of instruction stream in the 40's, then Harvard architecture has been proposed in order to parallelize both streams. Finally in the 60's, the concept of the cache memory has been introduced to speed up stream outputs.

A configuration stream as specified in Fig.1 can now be considered as the third kind of stream in the context of reconfigurable architectures. Consequently, this new stream should now go with data and instruction ones and must be also speed up with a cache hierarchy considering voluminous files but moderate update frequencies.

On the application side of embedded systems, we can observe the same evolution in terms of complexity. Multiple and heterogeneous applications can share resources and stress hardware in different ways such as typical networking, signal/image processing, encryption functions. In case of personal devices, 3D graphics, display and GUI can be added. An OS can then become a necessity for hardware abstraction and resource allocation.

Both architecture and application evolutions lead to highly dynamic and data or context dependent behaviors of embedded systems. In these changing conditions, the problem of optimization, regarding for instance energy efficiency, can hardly be handled without any configuration considerations.

Finally, any design methodology targeting embedded systems is strongly constrained by development cost. The use of IPs and standard API are tracks to reduce the effort cost, this option must also be considered in the domain of reconfigurable architectures. This approach makes sense in the domain of mass market products and ambient intelligence, based on standard set of application and basic functions.

In this context, self-adaptivity applied on well-defined architecture models, is a promising way to solve the question of optimization at run-time. 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 a near future in terms of power and reconfiguration time. The second condition is decision capability.

Given this situation and the assumption that a majority of embedded systems will be somehow connected to a network, we propose in this paper an overview of a global approach for the design of self-adaptive systems targeting effective architectures according to applications demands. The methodology is first locally based on a combination of static and dynamic configuration decisions for both hardware and software aspects, the objective is to get a trade-off between decision complexity and configuration storage cost. Secondly, the idea is to increase the local configuration space by means of a shared and distributed hierarchy of configuration caches. The approach is based on two assumptions. First some generic architecture models have been previously defined and secondly applications are based on an intensive use of an evolutionary library of API and standard coprocessors or IPs. Section 2 presents the global approach. Section 3 illustrates local self-adaptivity with a smartcam demonstrator and section 4 introduces global-self adaptivity with a MP3 example. Finally we conclude.

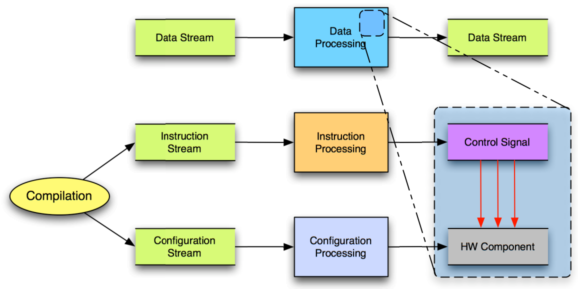


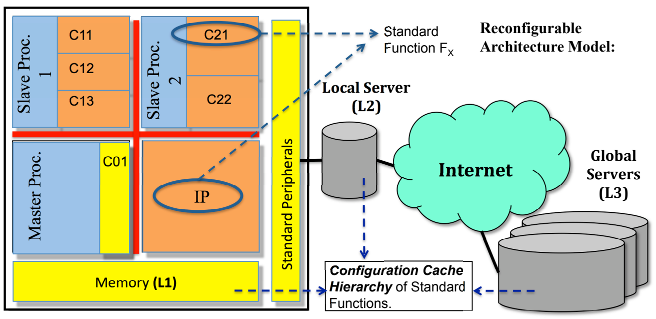
Fig. 1: R-MPSoC: Data, Instruction, Configuration Streams

2. Global Methodology

The basic problem addressed is the partitioning of a set of tasks over a given architecture model, the aim is to specialize at run-time the architecture according to application needs. As for the domain of embedded systems, we consider today an architecture meta-model based on a master processor (MP) and a set of hardware accelerators (IPs) or slave processors (SP) connected to a hierarchy of bus as depicted in Fig.10. Depending on the application domain and performances requirements, different models can be selected.

The main idea is to reduce the design space to be explored at run-time in order to introduce short and low cost decision overheads. Thus, the approach consists first in loading a reduced pre-defined set of configurations used for fast local adaptations according to context and data variations. Then, this set can be updated at run-time through a network connection if better configurations are necessary or if new applications, requiring new configurations, are started.

Therefore, we have adopted a methodology that can now be decomposed in five steps. The first one is the specification step during which the designer should think in terms of self-adaptivity at functional level. The second one consists in an offline design space exploration (DSE) step, which provides a set of possible configurations to be selected at run-time by the next two online steps. These two online steps address algorithmic and architectural configurations implemented by local and global configuration managers respectively. Finally the last step is also handled online by the global manager, that can update available configurations in the local configuration memory. It is based on a hierarchy of caches (Fig.2) and has two objectives. The first one is to extend the accessible configuration space for a given application regarding on-chip / on-board space storage restrictions. The second one is the update of software and hardware versions.



Self-adaptivity Methodology:
1. Offline Self-adaptivity oriented specification
2. Offline Design-Space Exploration (On-chip / On-board Configuration Space)
3. Online Algorithmic Configuration (Local Manager)
4. Online Architecture Configuration (Global Manager)
5. Online Configuration Update (Hierarchy of Configuration Caches: chip, board, network)

Fig. 2: Global Framework

3. Local Self-adaptivity

3.1 Introduction

Actually, another challenging issue is the embedding decision within embedded systems to perform hardware (HW) / software (SW) partitioning at run-time. 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 section.

We consider an architecture meta-model, which fits with a large set of embedded systems. Then, considering the smartcam example, we select a model based on a GPP with dedicated accelerators that can be dynamically configured (M01 on Fig.10). 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.

3.2 Objectives and contributions

Considering the previously described architectural model, the main objective of this work is the implementation of a **configuration manager** with run-time **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 at design and at run-time;
2) Transparent use of various HW and SW implementations;
3) Separation between application-specific and application-independent configuration decisions;
4) Online decision: algorithms and implementation;
5) Reconfiguration control, transition reliability;
6) System stability, avoidance of configuration oscillations;
7) Negligible self-adaptativity overhead (power, area, time).

Our contributions focus on these items, with the objective to remain, as much as possible, independent from hardware platforms since we consider that current dynamically reconfigurable devices and tools (e.g. Xilinx), are temporary solutions that may strongly evolve in the near future. We propose a design methodology regarding the issue 1) and we introduce the concepts of UCCI (unified configuration and control interface) and LR (legal representant) 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 close-loop approach and a Borda vote. We use data-granularity checking and configuration ID broadcasting to deal with problem 4). A PI (proportional integrator) regulator, enhanced with LMS (least mean square) observer, is implemented as a solution to question 5). Conditions on control parameters have been derived to guide designers regarding issue 6). Finally, point 7) has been our constant optimization criteria. In this paper we give an overview, however some details can be found in [2] regarding points 1), 3), 5), 6) and 7); details of items 2) and 4) can be found in [3] and [4] respectively.

3.3 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 hardware and software aspects. A lot of work has been produced in the domain of adaptive architectures and different techniques have been introduced for clock and voltage scaling [5], cache control [6] or functional unit [7] 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. In [8], 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 [9] 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 [10]; it is based on an evolutionary algorithm and limited to a single task application and very simple architecture transformations. 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. Finally, 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 [11], authors present a complete model for feedback control real-time scheduling. In [12], a relevant two-step approach is proposed. However, this kind of technique does not fit for embedded low cost systems and

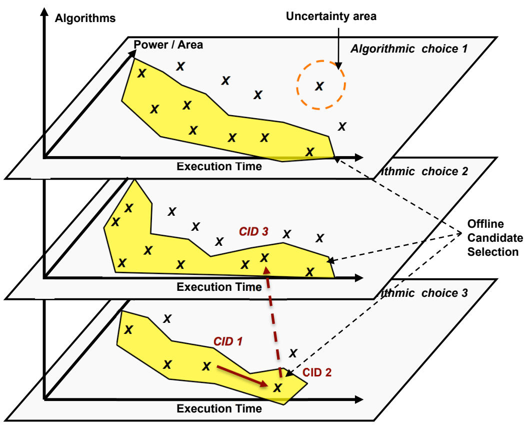


Fig. 3: Configuration Space

reconfigurable hardware, but the approach of system stability remains interesting.

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.

3.4 Offline steps: specification and DSE

Self-adaptivity aware specification First, we consider that self-adaptativity starts at application design and software designers are asked to specify the conditions under which 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 software designer will follow a methodology in which he will specify lists of algorithmic configurations, and transition rules based on metrics. 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.

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 depicted in Fig.3.

3.5 Online steps: configuration management

Separation of concerns The aim is to implement configuration management for online algorithmic and architectural adaptation. Fig.3 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 algorithmic reconfiguration (e.g. CID1 $\rightarrow$ CID2) and a move between plans means an architectural reconfiguration (e.g. CID2 $\rightarrow$ CID3).

The challenge, when it comes to the configuration decision of embedded systems, is to find a low cost, but efficient solution. Our solution, regarding these objectives, relies on a couple of local (LCM) and global (GCM) managers, that can be implemented as software or hardware tasks. In the smartcam demonstrator (§3.8), GCM and LCM are implemented as SW tasks using some wired instructions (coprocessors).

LCM: algorithmic adaptation Basically, the local configuration manager (LCM) is a set of rules specified with simple API by the software designer and efficiently implemented by means of masks, e.g. $if\ F\{m_i\}\ true\ then\ apply\ Mask_m[]\ to\ Configuration\_list$ where $Mask_m$ is an array of bits with a length equal to the number of possible configurations, such as $Mask_m[i] = 1\ if\ CID_i$ is $valid$ and 0 otherwise. A rule is a simple logic function based on metrics issued from tasks.

GCM: architecture adaptation The global configuration manager (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 3.7.

3.6 Configuration management

Concept This point is not the main concern of this paper, however to make it easier to understand we briefly give an overview of the main concepts. The reconfiguration of the system at run-time raises two questions. The first one is the synchronization of tasks after a reconfiguration has been performed, the second one is the problem of interfaces. Both aspects are solved in the context of RTOS, more details can be found in [4].

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 indicates to its successors, through mailbox or queuebox mechanisms, the address of data in a shared memory protected with Semaphore or 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 Unified Configuration and Communication Interface (UCCI). It is implemented by means of API when the task is running on the processor and as a HDL code container when the task is mapped on a hardware accelerator. The interface is in charge of synchronization mechanisms (e.g. mailbox), transfer of metrics to the LCM, configuration mechanism and memory accesses (DMA in case of HW implementation). When a task is moving from SW to HW, it still remains alive in the RTOS as a sleeping task in charge of RTOS / HW accelerator communications, this concept is called the Legal Representative (LR). Communications between HW accelerators are direct and, therefore, do not solicitate the RTOS (RTOS communication services are distributed in such a case since implemented in HW UCCI).

Configuration control mechanisms We solve the synchronization issue by means of diffusion mechanisms. Our method reuses existing communication channels, which can be direct for HW to HW communications or based on RTOS services for HW/SW and SW/SW communications. We, therefore, have developed the following strategy. Firstly, the configuration manager, namely the LCM, sends the CID to all source tasks through a multi-cast diffusion. Secondly, the CID is propagated gradually from the source to the sink tasks over data channels, after granularity control to avoid data starving and inconsistency. With such diffusion principles, we guarantee that all tasks will be configured starting with the source tasks.

3.7 Configuration Decision

Introduction We propose an original approach based on a close-loop model that consists in 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.

Close-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 running a set of tasks, that can be implemented with various versions on different HW/SW resources, and control, estimation and configuration tasks. Our model, depicted in Fig.4 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.

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.

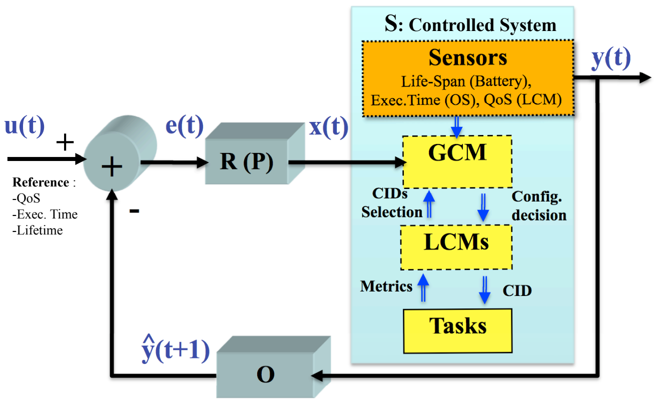


Fig. 4: Generic Close Loop System

Observor design The observer regularly updates coefficients $\{a_i\}$, this is a model that estimates the system behavior. The aim is firstly to predict the magnitude evolution in order to anticipate the right decision for reconfiguration. Secondly, 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. Considering the algorithm complexity for adaptation and estimation and the filter length we have opted for a 3-tap LMS for the observer implementation.

Configuration decision 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 tradeoff between efficiency and complexity compliant with embedded systems.

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 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. All magnitudes are defined in such a way that a configuration meets the constraints if $V(i,j) > 0$.

$$V(i,j) = U(i) + Th(i) - (Tab(i,j) + X(i,j|k)) \qquad (1)$$

Decision Algorithm. The main steps of the decision algorithm are given in Fig.5. 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.

Secondly, the GCM is also pending on a mail box, 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 reconfiguration 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. Finally, all considered magnitudes are assessed for the final selection regarding a given priority order (e.g. T, QoS, P).

The algorithm runs as follows. First, note that only the first constraint is regulated (e.g. T) and considered for selection. Secondly, other constraints are considered when more than one solution respect the first one (namely $V(1,j) > 0$), otherwise the candidate providing the smallest error is selected regarding only the first constraint. Then a vote based on Borda's method [13] 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 multiple candidates obtain the same score then a Hamming distance with current configuration is used to select minimal SW $\rightarrow$ HW moves.

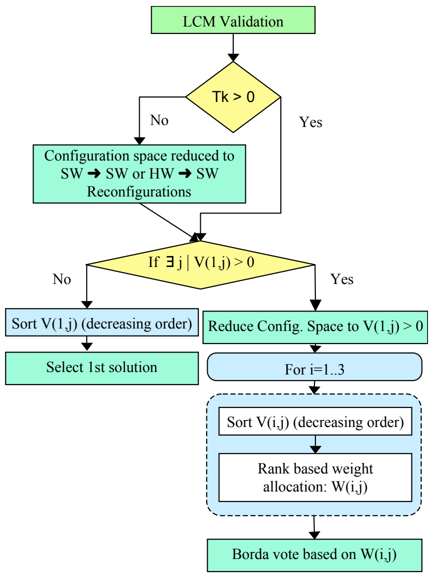

LCM Validation
Tk > 0
No
Yes
Configuration space reduced to SW ➜ SW or HW ➜ SW Reconfigurations
If ∃ j | V(1,j) > 0
No
Yes
Sort V(1,j) (decreasing order)
Reduce Config. Space to V(1,j) > 0
Select 1st solution
For i=1..3
Sort V(i,j) (decreasing order)
Rank based weight allocation: W(i,j)
Borda vote based on W(i,j)

Fig. 5: Decision algorithm

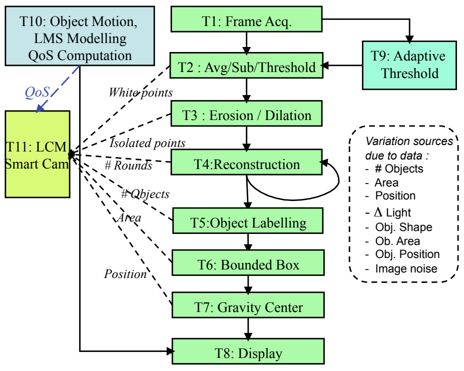


Fig. 6: Application flow

3.8 Object tracking test-bed

The best way to prove the efficiency of our approach, in the absence of equivalent and available approaches to compare with, is the validation of our method on a real-life application, in which self-adaptivity makes sense. Hereafter we present 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.

a) Application specification, metric selection The application is composed of 10 tasks (T_1 ... T_{10}) described in Fig.6, 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 Fig.7 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.

b) 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. The image acquisition rate is controlled by the GCM and follows up the application rate to avoid useless image storage. The extended RTOS is built around μCos II and provides the information about task and 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.

c) Design Space Exploration The extended RTOS is built with new previously explained capabilities for communication, synchronization and configuration of HW and SW tasks. Hardware 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 hardware modules were designed after a short design space analysis. In this case study, the algorithmic configuration is the complete application with a fixed threshold (T_9 off), and a gravity center is approximated as the center of the bounded box (T_7). After this stage, we have limited the search space to 22 significant configurations with the following algorithmic choices: 1) Deep sleep mode: $T_{3,4,5,6,7}$ are inactive; 2) Sleep mode: $T_{4,5,6,7}$ are inactive; 3) Reconstruction T_4: on or inactive; 4) filter inactive or based on two or four images; 5) T_9: on or inactive (fixed threshold).

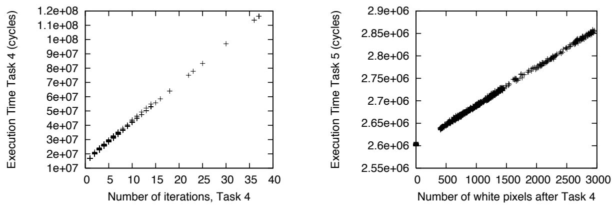


Fig. 7: T_4 metric selection based on designer experience

d) UCCI encapsulation Hardware and software 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 software 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 software 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 hardware 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 memory, dedicated registers are specified within the UCCI shell to indicate their base address.

e) LCM implementation The next step is the LCM specification, in this case a single LCM is required and a software 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 one. 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. For instance, rule 1 relies on the selection of fixed or adaptive threshold computation and the number of frames in T_1 according to the number of isolated white points after T_2 and T_4.

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.

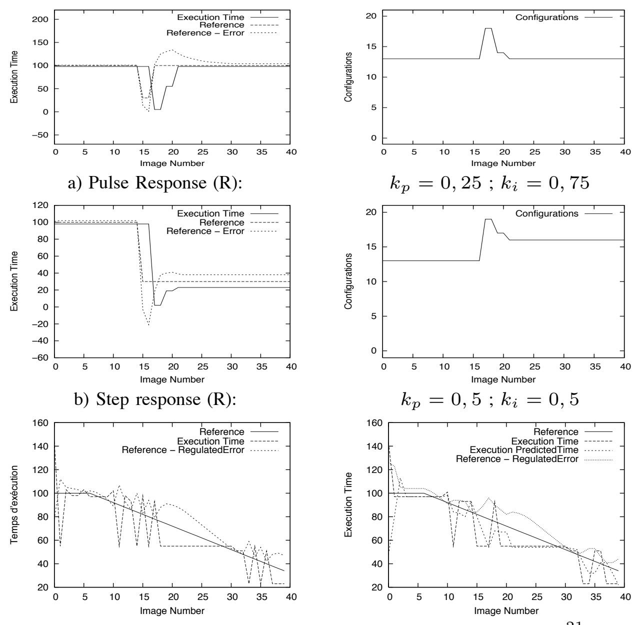
f) GCM implementation In this section, we focus on the main GCM parameters, which are the PI regulator coefficients: k_i, k_p and the LMS observer gain k_L. Various experiments have been conducted with the smart camera prototype implemented on FPGA. The GCM is implemented as a software task, so the choice of the coefficients results from a simple variable initialization within the associated C code.

The following sections show how a software 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.

g) PI and LMS parameter specification Fig.8 presents some pulse, step and slope responses for different choices of R ($\{k_p, k_i\}$) and LMS parameters (k_L). In this case the regulated magnitude is the execution time, the X axis is the time represented as the image number. 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. The LMS implementation is based on a co-processor, namely a custom instruction, which is initialized with software instructions. 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. These example show, how the software designer can set the right parameters according to application requirements.

3.9 Train tracking application

a) Scenario To illustrate the self-adaptivity abilities of the



a) Pulse Response (R): $k_p = 0,25$; $k_i = 0,75$

b) Step response (R): $k_p = 0,5$; $k_i = 0,5$

c) Slope Response: No LMS With LMS: $k_L = 2^{-21}$

Fig. 8: Exec. time regulation

prototype, we propose tracking an electric toy train with a scenario punctuated with various events inducing different configuration decisions. Fig.9 shows the configuration decisions along the execution of this scenario. In this scenario, the regulated magnitude is no longer the execution time but the QoS, namely the tracking accuracy. It means that Borda's vote is applied to power and execution time values.

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 tradeoff between stability and reactivity. In the same way, the LMS gain has been set to $k_L = 2^{-21}$. The adaptation rate is set to one, 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. This scenario is based on a succession of events that highlight self-adaptation capabilities. After several images without any movement, the train 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 backwards. It then enters a Critical Zone, runs into and leaves the area. Finally, the train continues its path at low speed.

b) Result analysis The CPU time devoted to LCM and GCM task (0.33% in a pure SW solution) and the HW overhead due to the co-processor (1%) are negligible in such an applicative context where reconfigurable architectures make sense.

With different algorithm and architectural configurations, we obtain tracking system performances. Tab.1 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. Tab.7 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.

Some architecture configurations may involve significant time overheads (HW and SW tasks switch), however it is also clear that hardware 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, video encoders. In our case study, for instance, we observe that message passing represents a very low percentage of the entire whole communication.

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

Table 1: Design results with a 50 MHz clock

4. Global Self-Adaptivity

4.1 Introduction

Embedded systems require fast and low cost self-adaptivity, we reach this objective by reducing the configuration space based on a set of promising solutions around an average configuration. However such a solution may be too limited if the variations of application conditions lead to very large configuration space or if multiple complex applications are running concurrently. For cost and technology reasons, the answer cannot be limited to the increase of embedded memories even if RAM and Flash resources enable to store large amounts of configuration files. The question of firmware update is another important issue with a great impact on development cost. However, most of embedded systems can access to the network and so to huge and shared configuration (bitstream, binaries) repositories. Thus we can extend the available reconfiguration space by means of hierarchy of caches as depicted in Fig.2, which for instance can be composed upper the configuration memory of an

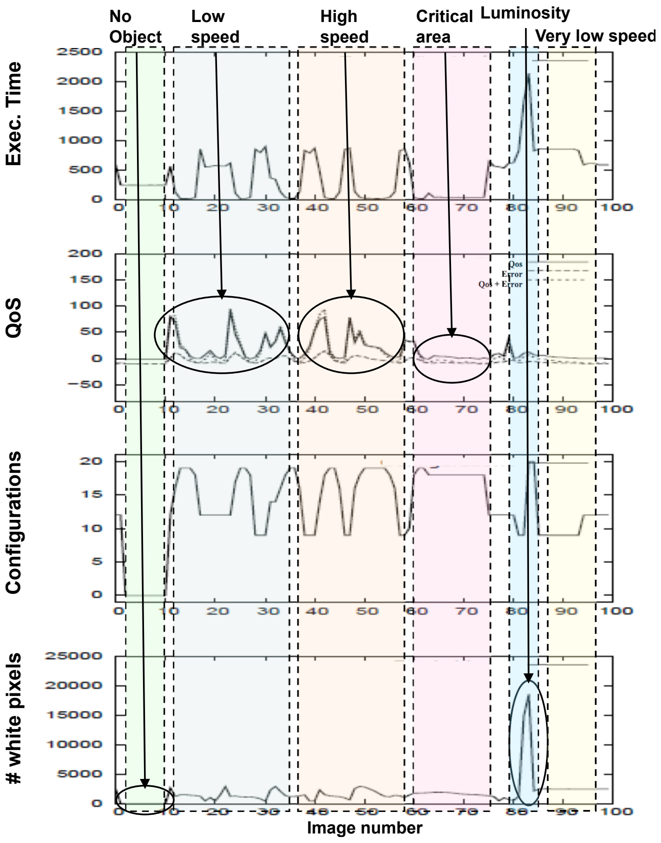


Fig. 9: Self-Adaptivity scenario with LMS

on-board DDR memory, a local W/LAN server and some Internet Servers providing new releases and applications. Such an approach raises two additional questions. First, architectures must be defined as instances of a registered model, in order to classify configurations available to a set of identical reconfigurable embedded systems. It also means a new flexible programming model based on standard interfaces and API compliant with the architecture model. The second main point relies to the cache policy, namely the way configurations are stored and updated at each level of the hierarchy.

4.2 Networked reconfigurable systems

A lot of future applications, based on distributed embedded systems, are expected in the domain of intelligent environment (smart cameras) or transportation (smart sensors) and in the context of nomadic devices (software defined radio, computing). It means that wired and wireless local area networks can be considered as an available solution. Given this assumption, we have implemented different architectures providing a networked-reconfiguration service for Xilinx FPGA. Note that the objective was to implement partial and dynamic reconfiguration directly from a remote and local server. In other words, we bypass the DDR level of the cache hierarchy when the requested configuration is not available on board. Previous work have considered such a network approach, William and al. [14] have developed a μClinux device driver on top of ICAP, enabling bitstreams downloading. The system is based on a Microblaze implemented on Virtex2p. As no measures are provided, estimations done in a similar context, lead to a transfer speed ranging from 1.6 Mb/s to 3.2 Mb/s. Lagger et al. [15] also propose a solution based on μClinux for cryptographic application in the context of pervasive applications, authors indicate datarate ranged from 240 Kb/s for HTTP and 480 Kb/s for FTP. This shows that "TCP + μClinux" is very flexible and widely accepted as a universal platform, however it also means an overhead that can be an issue in case of real-time applications. Not strictly dedicated to DPR, the XAPP433 [16] application note from Xilinx, describes a HTTP server performing 4Mb/s built around a Microblaze implemented on a Virtex4 FX12 running at 100 MHz and based on the lwIP [17] protocol stack and the Xilinx XMK OS. We have implemented different stand-alone solutions tested with different kind of bitstreams, A1 is a customized IP protocol implemented on a PowerPC, A2 is an enhanced version based on A1 and improved with a DMA [18]. Considering the use of standard protocols and very low error rates observed in usual W/LAN implementations, we have also implemented solutions based on UDP [19] in case of wired (A3) and wireless (A4) networks. Table 2 shows the throughputs and memory footprint we obtained. These results and recent experiences based on Virtex V indicate that we can expect to reach the maximum LAN bandwidth with FPGA, namely 100Mb/s and more in a near future. We can conclude that LAN capacities will fix L1 cache bandwidth. Regarding L0 in the case of FPGA, it is bounded by ICAP bandwidth that can theoretically raise up to 3,2Gb/s on Virtex V, which corresponds to a DDR range.

	Lagger [15]	Williams [14]	Xilinx [16]	A1 (ad hoc IP)	A2 (A1 + DMA)	A3 (wired UDP)	A4 (Wifi UDP)
Throughput (Mb/s)	1.7	3.2	4	40	80	60	30
Memory (bytes)	$> 1M$	$> 1M$	$< 100K$	$< 100K$	$< 100K$	$200K$	$200K$

Table 2: Throughputs and memory footprints

4.3 Modeling and standards

If we consider now that an embedded reconfigurable system can efficiently download a set of configurations from global servers, it means that open source programs extended to hardware components can provide designers with infinite possibilities of (re)configuration and upgrade, it also means very exciting future prospects. Depending on technologies and devices families, different solutions can be proposed. If we consider for instance R-MPSoC based on Xilinx FPGA and Microblaze cores, we can propose a meta-model from which can be derived various architecture models. In this context, Model Driven Engineering tools can usefully help designers to generate codes and compilation or synthesis scripts automatically, in [20] we present such an approach, based on MARTE/UML meta-model.

Fig.10 shows how a meta-model is progressively specialized in order to provide a given reconfigurable architecture. We start from a meta-model (M01) of a general architecture composed of a manager processor controlling some slave processors with a given number of co-processors, and various other components such as shared memories, IP and peripherals. Then an instance of the selected model is specified by the designer. In this case M11 corresponds to the previous smartcam architecture, M10 is the model we adopted for the second example for audio coding. It is composed of a manager processor and two slave processors that can then be dynamically configured by means of two co-processors. Thus, servers of configuration can host database where configurations are identified and classified according to instances of architecture models.

The programming model is another important issue for the design of applications over a given model of reconfigurable architectures. Our approach, depicted in Fig.11, is an extension of the model we developed for self-adaptivity, where the granularity level is a task (namely a thread). Each application is composed of a set of tasks and a local configuration manager (algorithmic configurations), a descriptor file provides the configuration manager (GCM) with meta-data including the list of used standard functions. A unique GCM, implemented on the master processor (MP), decides the global architectural configuration, it is also in charge of configuration downloading through the cache hierarchy. Each task is specified with a UCCI interface, which is in charge of configuration test, synchronization and communications mapping. A task can be mapped as software task on a slave with or without specific co-processors or as hardware task as a specific IP. As explained in introduction, we observe that applications in embedded systems share a large set of common basic functions, we believe that these functions can be called through common API adapted to the different architecture models. The hardware or software implementation of these functions don't need be redesigned but could available on configuration servers.

4.4 MP3 Case study

To illustrate the global configuration concept, we have built networked self-adaptive architectures based on available dynamically configurable processors on FPGA. The XPSoC-V2 architecture, for instance, is based on model M11 in Fig.10 and implements, on a Xilinx ML410 board (Virtex4 FX60), two microblazes. The first one (MP) is the manager and the second one (SP) is a slave with configurable coprocessors connected to FSL interfaces. MP runs a petalinux

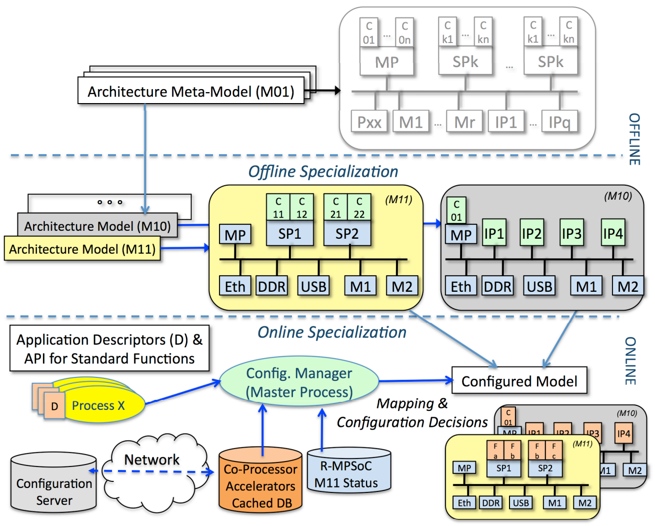


Fig. 10: Global model views

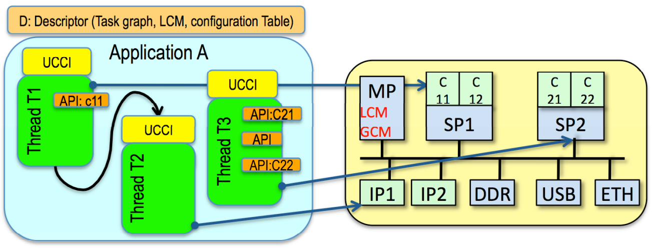


Fig. 11: Heterogenous mapping based on standard interfaces

[21] operating system. In this case study, SP runs an MP3 decoder application. Two types of coprocessors can be downloaded from remote configuration server, a IMDCT coprocessor and a MUL16 coprocessor. Table 3 presents performances and area for the different versions.

	Software	HW-MUL16	HW-IMDCT
Slice used for coprocessor	0	161/ 25280	482/25280
DSP used for coprocessor	0	4/128	4/128
LUT used for coprocessor	0	86/50560	521/50560
Execution TimeS	307.2	232.9 (24%)	167.7 (45%)

Table 3: Mp3 decoding on XPSoC-V2

5. Conclusion

In this paper we have presented a global methodology for the design of self-adaptive systems. It is first based on a close-loop approach for deciding, at run-time, among a finite set of configurations the best one according to user references. Then we extend this solution by implementing a hierarchy of remote configuration servers providing an access to a huge configuration space. Finally, we propose to open the design of standard configurable multiprocessor architectures, by means of model specification that enables to a large set of hardware and software engineers to share applications and standard function designs.

References

[1] G.Estrin, B.Bussell, R.Turn, and J.Bibb, “Parallel processing in a restructurable computer system,” *IEEE Transactions on Electronic Computers*, no. 6, pp. 747–755, Dec. 1963.
[2] Y.Eustache and J-Ph.Diguet, “Specification and OS-based implementation of self-adaptive, hardware software embedded systems,” in *6th Int. Conf. Hardware/software Codesign and System Synthesis (CODES-ISSS)*, Atlanta, USA, 2008.
[3] Y.Eustache, J-Ph.Diguet, and M. Khodary, “RTOS-based hardware software communications and configuration management in the context of a smart camera,” in *Proc. of the Int. Conf. on Engineering of Reconfigurable Systems & Algorithms (ERSA)*, June 2006.
[4] Y.Eustache and J-Ph.Diguet, “Reconfiguration management in the context of RTOS-based HW/SW embedded systems, *special issue on Operating System Support for Embedded Real-Time Applications*,” *EURASIP Jour. of Embedded Systems (JES)*, Jan. 2008.
[5] J.L.Wong, G.Qu, and M.Potkonjak, “An on-line approach for power minimization in qos sensitive systems,” in *ASP-DAC*, 2003.
[6] D.H.Albonesi, “Selective cache ways: On-demand cache resource allocation,” in 32nd *Annual Int. Symp. on Microarchitecture*, 1999.
[7] R.Maro, Y.Bai, and R.I.Bahar, “Dynamically reconfiguring processor resources to reduce power consumption in high-performance processors,” in *Work. on Power-Aware Computer Systems*, 2000.
[8] J.Liang, A.Laffely, S.Srinivasan, and R.Tessier, “An architecture and compiler for scalable on-chip communication,” IEEE *Trans. on* VLSI *Systems*, vol. 12, no. 7, pp. 711–726, July 2004.
[9] L.Goddard, C.Moy, and J.Palicot, “From a configuration management to a cognitive radio management system of SDR systems,” in *CROWNCOM*, Greece, June 2006.
[10] P.Kaufmann and M.Platzner, “Towards self-adaptive embedded systems: multi-objective hardware solution,” in *Int. Work. on Applied Reconfigurable Computing (ARC)*, London, UK, Mar. 2008.
[11] C.Lu, J.Stankovic, G.Tao, and S.Son, “Feedback control real-time scheduling: Framework, modeling and algorithm,” *special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, vol. 23, no. 1/2, pp. 85–126, july/september 2002.
[12] B.Li and K.Nahrstedt, “A control-based middleware framework for quality of service adaptation,” *IEEE Journal on Selected Areas in Communication*, Sept. 1999.
[13] J-C.De Borda, “Mémoire sur les élections au scrutin (in french),” *Histoire de l’Académie Royale des Sciences, Paris*, 1781.
[14] J.Williams and N.Bergmann, “Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip,” in *The Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, USA, 2004.
[15] A.Lagger, A.Upegui, E.Sanchez, and I.Gonzalez, “Self-reconfigurable pervasive platform for cryptographic application,” in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, Aug. 2006.
[16] Xilinx, “Xapp433. web server design using microblaze soft processor,” October 2006.
[17] A. Dunkels, “lwip,” *Computer and Networks Architectures, Swedish Institute of Computer Science, http://www.sics.se/ãdam/lwip/*, 2001.
[18] P.Bomel, J.Crenne, L.Ye, G.Gogniat, and J-Ph.Diguet, “Ultra-fast downloading of partial bitstreams through ethernet,” in *Proc. of the Int. Conf. on Architecture of Computing Systems (ARCS)*, ser. Lecture Notes in Computer Science, Delft, The Netherlands, March 2009.
[19] J.Crenne, P.Bomel, G.Gogniat, and J-Ph.Diguet, “UDP partial bitstreams diffusion through WLAN,” in *Int. Conf. on Design, Archi. for Signal, Image Proc. (DASIP)*, Sophia Antipolis, France, Sept. 2009.
[20] G.Vidal, F. Lamotte, G.Gogniat, P.Soulard, and J-Ph.Diguet, “A codesign approach for embedded system modeling and code generation with uml and marte,” in *DATE*, Nice, France, Apr. 2009.
[21] Petalinux. [Online]. Available: http://developer.petalogix.com/