Specification and OS-Based Implementation of Self-adaptive, Hardware / Software Embedded Systems

Yvan Eustache
Université Européenne de Bretagne
UBS LabSTICC Lorient, FRANCE
yvan.eustache@univ-ubs.fr

Jean-Philippe Diguet
Université Européenne de Bretagne
UBS LabSTICC Lorient, FRANCE
jean-philippe.diguet@univ-ubs.fr

ABSTRACT

This paper presents our solution for specifying and implementing self-adaptivness within an OS-based and reconfigurable embedded system according to objectives such as quality of service (QoS), performance or power consumption. More precisely, we detail our approach to separate, at runtime, application-specific decisions and hardware/software implementation decisions at system level. The first ones are related to the control of the efficiency of applications, they are specified in Local Configuration Managers (LCM) based on the knowledge of application engineers. The second ones are generic and address the choice between various hardware and software implementations according to observations of the gap between online measurements and objectives set by the user, these decisions are implemented in the Global Configuration Manager (GCM) as an adaptive close-loop model. We have designed a video tracking application on an 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 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms

Management, Design

Keywords

Self-adaptive embedded systems, HW/SW codesign

1. INTRODUCTION

The introduction of hard and soft processor cores into reconfigurable circuits has changed the job and horizon of designers by providing them with complete Reconfigurable System On Chip (RSoC) providing hardware (HW) reconfiguration and the availability of (real-time) operating systems (RTOS). Dynamic HW adaptation to application requirements is a real issue that enables architecture specialization to optimize performances/cost/power trade-offs and a solution to perform online optimization that cannot be decided offline.

An (RT)OS for RSoC is an absolute necessity for handling complexity and providing the synchronization and communication abstraction of complex systems implementing heterogeneous applications where behavior is far from being deterministic.

A lot of work has been done in the domain of reconfigurable architectures, however there are still two issues that have been more or less left on the sidelines: the transparent management of both HW and/or software (SW) tasks and the configuration decision.

Here, we have put the HW target on the side due to the fact that dynamic reconfiguration of FPGA with current Xilinx tools is a specific problem that we have already addressed in other projects.

In this paper, our solution for configuration decision is presented in five points. We first place our approach among the existing work. Then in section 3, we detail an original strategy for online configuration management in the context of an RTOS-based embedded system. In section 4, we describe and formalize decision mechanisms, while in section 5, we show our implementation on a smart camera. Finally, in section 6, we present results of our case study, before concluding.

2. RELATED WORK

When it comes to the issue of (re)configuration decisions of embedded systems, two domains are related to our work. Firstly, a lot of work has been produced in the domain of adaptive architectures (clock and voltage scaling, pipeline control, etc.), classified in the category of local configurations based on specific aspects. Our aim is to add global configuration management including both algorithmic and architectural aspects. Secondly, from the SW point of view, QoS management has been explored in-depth for web service applications. In [2], authors present a complete model for feedback control real-time scheduling. In [1], a relevant two-step approach is proposed. However, this kind of technique is not fit for embedded low cost systems.

The idea of improving an RTOS has already appeared to be able to manage both HW and SW tasks. First proofs of concepts have been exhibited in [6] and [5]. In [5], the RTOS is dedicated to the management of HW tasks. In [6], the

CODES+ISSS'08, October 19–24, 2008, Atlanta, Georgia, USA.

OS4RS layer abstracts task implementations in HW or SW, and is mainly based on message passing API to associate logical and physical addresses on the reconfigurable hardware. Bergmann et al. [7] present a solution based on a μcLinux platform implemented on the Xilinx Micro-Blaze processor where hardware modules are considered as usual processes with their own address space. This work mainly focuses on HW/SW inter-process communications (IPC). Another approach, also based on a standard POSIX interface, has been chosen in [3]. The applied method is the migration of the main time consuming services of OS into hardware. SW and HW threads are managed through a new concept which is a common API called *Hthread* combined with a generic hardware interface. Finally, RECONOS, described in [4], is a solution based on the same assumptions that we set when starting our project. They selected eCos which is a low footprint real-time OS compliant with embedded system requirements; we have selected μCos and rejected Linux for the same reasons and for its availability on Xilinx and Altera platforms. They also propose a common hardware interface for hardware modules, which are present in the OS task table through SW *delegates* that seems to be equivalent to what we call a *Legal Representative* and to the concept of *Ghost Processes* in [7]. Our main contribution is precisely related to the configuration decision process that automatically manages system configuration according to user requirements. Contrary to [4], we distribute inter-process communications in order to speed up and benefit from concurrency for HW/HW task communications.

3. CONFIGURATION MANAGEMENT

The first question raised by the reconfiguration process is the locality. A reconfiguration can be decided at the application level or system level. Based on application specific data, a local decision provides a short reaction delay and metrics to compute the QoS. However, some decisions must be considered globally when a trade-off must be found over the complete system between power consumption and computation efficiency. An application is a set of inter-dependent tasks, so the choice also impacts the application specification by means of task and intra-task control definitions. Another pertinent point is the complexity of the decision implementation. In the context of embedded systems, only low cost solutions must be considered. The third point is the reconfiguration type, actually we consider a configuration space with 3 dimensions:

- **hardware** reconfiguration;
- **algorithmic** reconfiguration regarding different combinations of tasks with various algorithmic schemes for executing the application;
- **communication scheme**.

Each configuration is a point of the configuration space, which is identified as a configuration identifier (CID) with characteristics such as QoS, performances and power that are updated according to online measurements. Three dimensions of the configuration space are a good compromise between accurancy and complexity.

3.1 Two-step configuration strategy

To cope with these issues, we implement a two-step configuration management. The Local Configuration Manager (LCM) is the first level of management. There is one LCM per application; it is, therefore, application-specific and designed as a kernel task by the application designer. The second level is the Global Configuration Manager (GCM), there is one unique and generic GCM for the whole system. It is also implemented as a kernel task independently from embedded applications.

The LCM is in charge of the algorithm selection for all the tasks of the application it controls. This selection is based on data retrieved through a standard interface. Actually, the LCM is pending on mailboxes fulfilled by application tasks that can provide relevant metrics for the LCM to select right configurations (see Fig.1-a). The choice of these metrics falls to the competences of the application designers.Then, the LCM restricts the global configuration space regarding algorithmic decisions and transfers this choice as a mask of CIDs indicating which configurations are valid. Secondly, the LCM transforms application specific metrics into a normalized metric providing the GCM with the QoS of the application it controls. This is a kind of "application sensor" for the global manager. Finally, the LCM controls the application configuration while applying the GCM decisions as described in section 3.3.

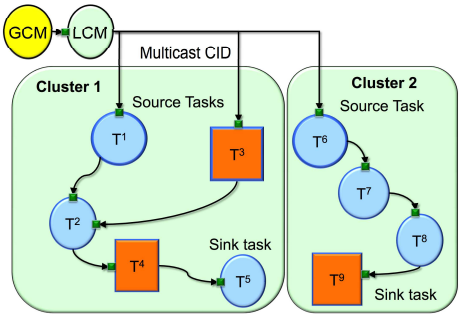
The GCM is pending on mailboxes fed by LCMs, it is in charge of global system parameters (e.g. video rate) and HW/SW implementation decisions. It receives data from sensors (gas gauge, CPU load, application QoS).The GCM chooses the new system configuration according to user requirements (system references) and configuration solutions issued from the LCM design space restrictions. The decision process is detailed in section 4. Finally, the GCM transfers its configuration decisions to the LCMs, namely the CID it has selected (see Fig.1-b).

3.2 Adaptivity-oriented design framework

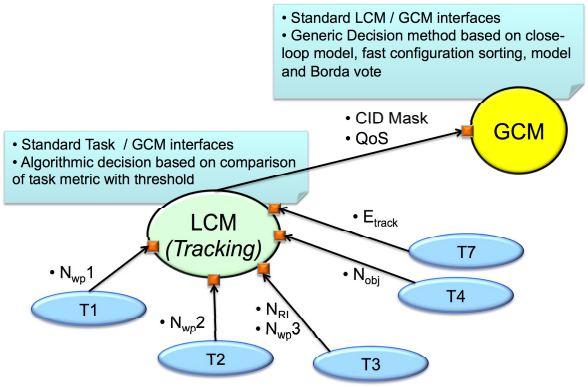
We believe that our approach is a good way to impose an adaptivity-oriented design discipline on the designer, the framework is composed of clear steps with associated API and Interfaces. The bottom flow can briefly be structured with the six following steps.

1. Specification of the application with algorithmic configuration in mind. The objective is to define, for the application tasks, the relevant metrics to be transmitted to the Manager for algorithmic decisions.
2. Specification of sensors (Power, Temperature, ...) providing information about environment and system status.
3. Design space exploration according to addressed magnitudes (e.g. Power, Exec. Time, QoS).
4. Encapsulation of task for configuration management: at the software level by using LCM API to send metrics and check configuration orders, and UCCI [1] API for inter-task and task-OS communication, at the hardware level by using UCCI VHDL shells and interrupt handlers.
5. LCM implementation: specification of LCM input mailboxes, implementation of decision rules as a FSM, namely the internal program of the LCM.
6. Definition of application QoS to be controlled by the GCM, use of LCM API to communicate this metric and CID mask from the LCM to the GCM.
7. GCM implementation and tuning of GCM policy: PI control and observer parameters.

[1] Uniform interface for communication, configuration control



a) (Top-Down) Configuration propagation



b) (Bottom-up) Metric propagation

Figure 1: LCM/GCM/Task interfaces

3.3 Configuration control mechanisms

We solve synchronization 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. UCCI has been defined for both HW and SW tasks and implemented as generic HW shells and SW API respectively. This aspect, which was previously published in [8], will not be detailed here.

We have, therefore, developed the following strategy. Firstly, the configuration manager, namely the LCM, sends the CID to all source tasks of all clusters through a multi-cast diffusion. Secondly, the CID is gradually propagated from the source to the sink tasks over data channels. With such diffusion principles, we guarantee that all tasks will be configured starting with source tasks. Note that, the propagation principle is included in UCCI services since HW tasks receive and send configuration IDs via OS communication services or directly according to the implementation of other tasks. Moreover, each HW task has a SW legal representative (LR), that for instance must be informed when the CID is directly received from another HW task.

4. CONFIGURATION DECISION

The GCM takes a configuration decision based on a feedback controller that adapts the system configuration to user requirement according to environment (data, system). This idea is to implement a smooth control, as is currently done for car engine power which can be dynamically adapted to a road profile for a given speed reference. The main differences with this domain are firstly the control cost which can not be neglected and secondly the control frequency which is much higher in embedded systems.

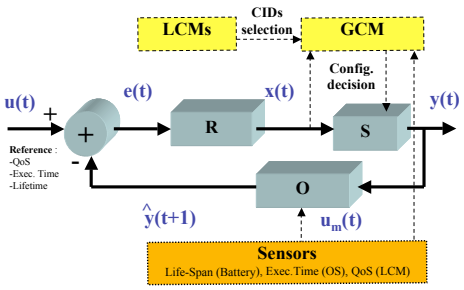


Figure 2: Close-loop Self Reconfiguration

4.1 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 control, estimation and configuration tasks and a set of application tasks, that can be implemented with various versions on different HW/SW resources. Our model, depicted in Fig.2 is based on three elements. R is the control function and S the configuration adaptation. 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 QoS, Lifetime (battery gauge), or Time (OS).

$e(t) = u(t) - \hat{y}(t)$ is the difference between the reference and the observer prediction output.

$x(t) = ke(t)$ is the output of the proportional regulator (R). The derivative effect is not used and the integral effect appears in the system modelling.

$y(t) = y(t-1) + x(t)$ is the output of the system after the reconfiguration adaptation (S). This function introduces an integrator effect.

$\hat{y}(t+1) = a_0 y(t) + a_1 y(t-1) + a_2 y(t-2)$ produces an estimate of the next average power consumption.

$y(t)$ is a noisy signal, since the power and delay in the configuration table are approximated. The first error source is the load of non controlled sporadic tasks and the second is the fluctuation of load that depends on nature of data.

The observer regularly updates a model that estimates the system behavior. The aim is firstly to anticipate decisions for reconfiguration and secondly to avoid costly sensor accesses when unnecessary. Considering algorithm complexity for adaptation a 3-taps LMS (Least Mean Square Algo.) has been chosen. Its stability is guaranteed under some conditions which have been defined in another publication, these details are out of the scope of the paper.

4.2 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.

The Decision Algorithm is described in Fig.3. 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 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 number of application iterations 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 in clock cycles, G_T the performance gain (the execution time difference 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 respects 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 is processed among survivor solutions, each magnitude sorts remaining configurations and attributes a vote corresponding to the rank. 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 → HW moves.

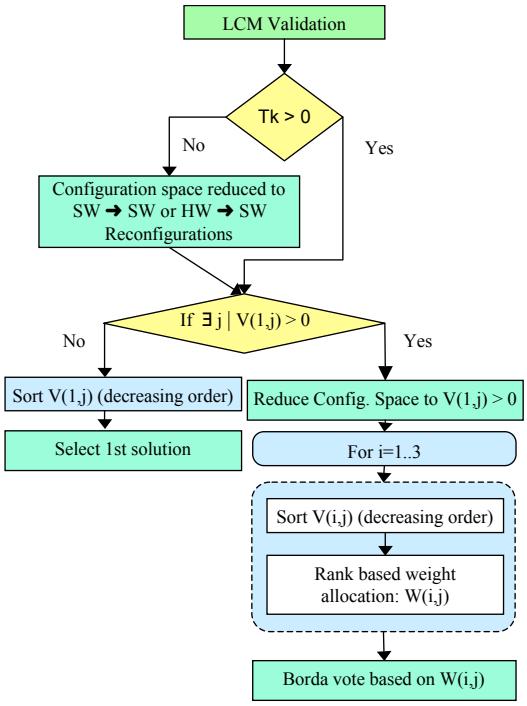


Figure 3: Decision algorithm

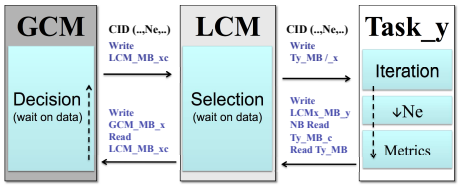


Figure 4: Control sampling and application period

Configuration period: the issue of a sampling period is a critical aspect of controlled systems, in our case system we consider models linear with application iterations which means that the sampling period is equal to N_e application iterations as depicted in Fig.4, where N_e is an integer and global configuration parameter. The approach minimizes communication and computation overheads.

5. SMART CAMERA IMPLEMENTATION

We will now illustrate the different steps of the design methodology proposed in section 3.2 with an embedded smart camera for object tracking implemented on a FPGA. The prototype has been tested for tracking an electric toy train.

5.1 Application specification, metric selection

Our object tracking application is composed of 9 HW or SW tasks (T_1 ... T_9) described in Fig.5. These tasks are controlled by a LCM implemented as a SW task.

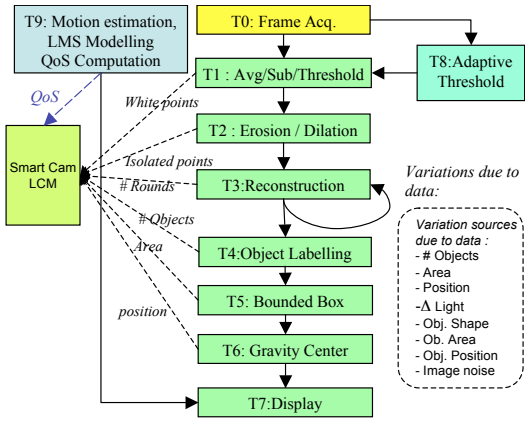


Figure 5: Application flow

The metrics provided by tasks to the LCM "tracking" are indicated in Fig.1-a) the number of isolated white points (N_{wp}) after tasks 1, 2 and 3, the number of iterations of object reconstructions (N_{RI}), the number of detected objects (N_{obj}) and the error of tracking (E_{track}) between prediction based on motion estimation and object extraction from real frames. On a global level, the GCM is also implemented as a SW task. Additionally, some specific SW tasks are implemented for VGA, battery and camera peripheral controllers, they include a minimal LCM integrated in each task for configuration communication with the GCM.

5.2 Environment sensors

The architecture is implemented on a NIOS soft core within an Altera Stratix II 2S60ES FPGA board with a camera and a VGA daughter board. We also plugged the camera and gas gauge (power consumption measures) into the FPGA GPIO. The extended RTOS is built around μCos and provides application execution time measures. The QoS sensor is provided by Task 10, it gives the difference between object position based on labeling results and an estimation of object positions based on a LMS algorithm. A value higher than the reference means that the application rate must be increased with the fastest configuration.

5.3 Design Space Exploration

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 for the LMS and PI regulator implementations. It is also used for application QoS computation (error between prediction and object position).
Significant algorithm configurations are selected with the following algorithmic choices:
1. Deep sleep mode: $T_{2,3,4,5,6}$ are inactive;
2. Sleep mode: $T_{3,4,5,6}$ are inactive;
3. Reconstruction T_3: on or inactive;
4. T_1: filter inactive or based on two or four images;
5. T_8: on or inactive (fixed threshold).

With Pareto methods, we restrict the system configuration space with architecture configurations.

5.4 UCCI encapsulation

The RTOS is extended with capabilities for communication and configuration of HW and SW tasks. A software task is enhanced with configuration updates, metric computation capabilities based on C code functions, and metric emissions (towards the LCM mailbox queues) based on API.
A hardware task is encapsulated within a VHDL shell, 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.

5.5 LCM and GCM implementation

The first point relies on the LCM strategies. Actually it is currently implemented as rules defined by the application designer according to simulation results. The number of mailbox queue instances is defined for the capture of metrics.

The second point focuses on the main GCM parameters, which are the PI regulator coefficients: k_i, k_p and the LMS observer coefficient k_L. Various experiences have been conducted with the smart camera prototype implemented on FPGA. The objective at this stage is to select coefficients in order to decide a good trade-off between reaction speed and error smoothing.

6. CASE STUDY

One of the main objectives of the project was the design of a real demonstrator. This embedded system, supplied by a battery, was mounted over a platform for tracking toy trains. We applied a two-step strategy, the first step of which was a design space exploration resulting in 58 configurations with 9 algorithmic combinations.

To illustrate the second step, which is self-adaptation, we present a comprehensive example, the results of which are summarized in Fig.6 for CID, execution time (cycle budget), the application QoS and the metric sent by the first task T_1 to the LCM, namely the number of white pixels. The priority order is QoS (the regulated magnitude), execution time and power. This scenario highlights self-adaptation capabilities. After several images without any movement, the train enters the scene and executes two rounds at low speed and two rounds at high speed, 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. All the scenario steps are described hereafter, and noted in Fig.6.
Starting point: Initially no object crosses the scene, the system selects CID #0 where only task one is running int SW, the camera frame rate is tuned according to the application execution delay, which is 2 frames/s.
Object arrival: Then, an object enters the scene at low speed, the system selects CID #12 and then a more efficient configuration (CID #19) due to the initialization phase of the tracking model (all tasks in HW, processing at more than 20 frames/s.) and the QoS decreases until it goes under the threshold of 10%. Then the less expensive configurations from CID #16 to #9 are selected according to Borda's votes. Tracking QoS mainly depends on train speed and direction, image rate, and distance to the camera.
Object speed up: we observe that the tracking system follows the train during the straight line. However, during curved lines, the system reacts, but no configuration is efficient enough (error $> 100\%$). CID #19 (all tasks in HW), the best one in such a situation, is selected. The error is reduced until the prediction model is adapted, then a less expensive configuration can be selected.
Critical Zone: The QoS into the Critical Zone is increased for security reasons, the reference is set at 2% and consequently, only CID #18 and #19 remain eligible.
Sudden luminosity change: The threshold no longer fits and the number of white pixels after the threshold (task T_1) suddenly grows and leads to the increase of the execution time. A special configuration implements the adaptive computation of a new threshold value based on the histogram gradient. To readapt the tracking model, the high efficient configuration (CID #19) is selected. The scenario ends with a low speed run around the circuit with CID #12 according to Borda's vote.

6.1 Power and performance

The CPU time devoted to LCM and GCM tasks (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. Tab.1-a) shows the communication performances. The overhead of HW $\leftrightarrow$ SW communication is due to context switch and control. One can observe that it is drastically reduced for HW to HW communications directly implemented in generic shells for HW tasks. Some architecture configurations may involve significant time overheads (HW and SW task switch), however it is also clear that HW tasks may be considered when computing parallelism is available and relevant speeding up achievable (e.g. image processing, encryption). In our case study, we observe that message passing also represents a very low percentage of the whole communication.
With different algorithm and architectural configurations, we

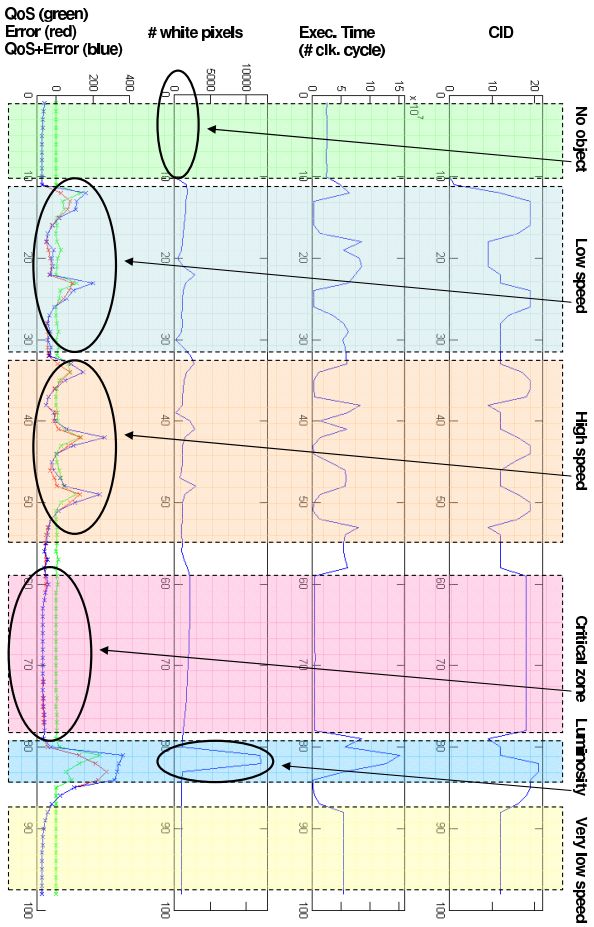


Figure 6: Decisions along scenario execution

obtain tracking system performances Tab.1-b). Execution time results correspond to a tracking process with a standard input frame. Execution time variation is first due to system architecture (e.g. cache miss, bus collision...) and secondly to data features. Tab.1-c) shows examples of such data-dependent performances that can justify the generalization of self-adaptive systems in the future. For instance, erosion and labeling execution times depend on the number of white pixels and the number of objects, and the number of white pixels and object complexity, respectively.

7. CONCLUSION

Self-reconfiguration is a promising way to improve efficiency of the SoC of the 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 already propose available frameworks to study and implement new concepts which need to be designed so that such systems become a reality in the future. In this work, we have proposed and fully implemented a solution that relies on three main contributions. 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.

a) Communication performances :

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

b) Examples of implementation results @ 50MHz:

T1-T2-T3	sw-sw-sw	sw-hw-hw	hw-hw-hw
T4-T5	sw-sw	sw-hw	hw-hw
Exec. Time	245.650.000 cy.	80.800.000 cy.	1.820.000 cy.
	±0, 01%	±0, 04%	±0, 2%
Frames/s.	0,20	0,62	26
Area	19 %	59 %	92 %
Power	137 mW	228 mW	285 mW

c) 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.

Table 1: Implementation Results

Moreover, its implementation is straightforward, because it is based on HW and SW shells that are independent from original C or VHDL code, respectively. Secondly, we propose an original and lightweight solution for online reconfiguration decisions; this is a learning-based method implemented as a control loop. To the best of our knowledge, this is the first time that observer and regulator concepts are applied in the context of RSoC self-configuration. Finally, a complete proof of concept has been designed with an object tracking application implemented on an FPGA with camera, VGA and battery peripherals. Self-reconfiguration, according to a QoS reference, has been validated with this demonstrator.

8. REFERENCES

[1] B.Li and K.Nahrstedt. A control-based middleware framework for quality of service adaptation. *IEEE Journal on Selected Areas in Communication*, 1999.

[2] 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*, 23(1/2):85–126, july/september 2002.

[3] D.Andrews, R.Sass, E.Anderson, J.Agron, W.Peck, J.Stevens, F.Baijot, and E.Komp. The case for high level programming models for reconfigurable computers. In *ERSA*, June 2006.

[4] E.Lűbbers and M.Platzner. Reconos: An rtos supporting hard- and software threads. In *FPL*, August 2007.

[5] H.Walder and M.Platzner. Reconfigurable hardware operating systems: From design concepts to realizations. In *ERSA*, Las Vegas, USA, June 2003.

[6] J-Y.Mignolet, V.Nollet, P.Coene, D.Verkest, S.Vernalde, and R.Lauwereins. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *DATE*, March 2003.

[7] N.W.Bergmann, J.A.Williams, J.Han, and Yi.Chen. A process model for hardware modules in reconfigurable system-on-chip. In *ARCS*, 2006.

[8] Y.Eustache, J-Ph.Diguet, and M.El Khodary. RTOS-based hardware software communications and configuration management in the context of a smart camera. In *ERSA*, June 2006.