QoS driven dynamic partial reconfiguration: Tracking case study

Julien Mazuet, Ill-Ham Atchadam, Dominique Heller, Catherine Dezan, Michel Narozny, Jean-Philippe Diguet

QoS Driven Dynamic Partial Reconfiguration: Tracking Case Study

Julien Mazuet[1,2], Ill-Ham Atchadam[3], Dominique Heller[2], Catherine Dezan[3], Michel Narozny[1], J-Ph. Diguet[2]
[1]*Thales LAS-France*, Élancourt, France
[2]*Lab-STICC, CNRS, Université de Bretagne Sud*, Lorient, France
[3]*Lab-STICC, CNRS, Université de Bretagne Occidentale*, Brest, France
julien.mazuet@univ-ubs.fr

***Abstract*—In hybrid systems on chip (SoC) the algorithms must share limited resources classified as hardware (FPGA logic, memory and DSP blocks) and software (CPU cycle budget). Embedded applications are implemented with concurrent processes, which have to coexist. Therefore, the algorithm performance or precision is often downgraded to fit on a FPGA or to meet timing constraints. One of the solutions to relax these constraints is to reconfigure the SoC according to application requirements observed at runtime. Hardware (HW) reconfiguration can be performed by means of dynamic partial reconfiguration (DPR). Beyond the intrinsic DPR complexity, the most critical aspect is the implementation of the reconfiguration decision. Academic adaptive architectures are usually based on power / performance rules. Nevertheless such solutions are usually application agnostic and so cannot fully exploit the possible adaptation to the environment. So real-life systems require solutions to capture the knowledge of algorithm experts that can be leveraged at runtime to drive the reconfiguration decision according to application scenarios. In this paper we propose a methodology to design a reconfiguration controller based on quality of service (QoS) indicators to be specified by experts. This controller monitors the reconfigurable partitions (RP) and can make DPR decisions to optimize the global QoS. We illustrate our methodology in the Radar domain with a tracking system based on Kalman filters implemented on a Zynq Ultrascale+. This study highlights expected gains and obstacles, it presents the different strategies to cope with the issues and draws perspectives.**

***Index Terms*—QoS, FPGA, DPR, Kalman filter**

I. Introduction

This work is motivated by the highly challenging requirements of future radar applications that are designed by signal processing experts. In this domain, processing tasks provide a high degree of data parallelism and can take full benefit from FPGA implementation. For instance, an active electronically scan array (AESA) radar exhibits multiple channels that can be processed independently and so provide tremendous opportunities for HW accelerations [1]. Besides, it is truly relevant to have different versions of the processing algorithm in order to adapt to the operational context which is changing with the number and types of detected objects as well as their environment. For example, in a target detection setting, in presence of clutter, one can imagine to get two different implementations of a space-time adaptive processing (STAP) algorithm. One implementation would perform very well at the expense of a higher computational complexity, and hence a greater usage of the available hardware resources. The other implementation would not be as efficient as the first one, but would require less computational resources, and therefore could coexist within the available hardware resources with one or many other algorithms like, for instance, a Kalman Filter for tracking. In most of the current implementations, all the possible configurations are specified together in the processing logic (PL), and the system switches between them. This entails limitations on the performances of each algorithm implementation since resources are reserved for nonactive alternative algorithms. A naive solution could be to just use a bigger FPGA, but this is a costly option where unused resources will still exist, which means that the best performances will not be reached. In this case, dynamic partial reconfiguration (DPR) appears as a viable alternative solution. DPR actually allows to dynamically allocate HW resources to the algorithms so that a global performance metric is maximized. However, this approach is possible only if a clear methodology allows signal processing experts, working with Matlab, to efficiently cooperate with reconfigurable SoC designers.

Section II presents a state of the art of DPR controllers which use diverse reconfiguration strategies. In section III we present a methodology to create an adaptive reconfigurable system to reconfigure regarding application QoS feedback. The case study of Tracking method, based on Kalman filters and implemented with the proposed QoS driven reconfigurable architecture, is described in section IV. Section V concludes the paper and draws directions for future work.

II. State of the art

Self-adaptive systems based on HW architectures can be modelled as a feedback control loop, as stated by the MAPE-K model [2]. The MAPE-K model has been introduced by IBM for autonomic computing and is described in Figure 1. With this control model, the reconfiguration manager behaviour is based on the application knowledge, which must be abstracted from the low-level software (SW) or HW implementation.

DPR allows runtime adaptation while exploiting FPGA features, such as speedup and energy efficiency [3]. If the target domain is well suited for FPGA and can benefit from parallel computation, bit-wise operations and fixed-point arithmetic, reconfigurable FPGA give a more efficient solution than GPU and multi-CPU while ensuring flexibility.

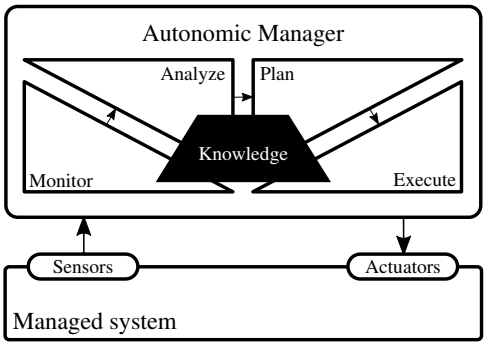

Fig. 1. MAPE-K loop

Several reconfiguration systems are described in the literature. In [4], authors propose to use DPR to switch between radar modes. This is a good example of how to use DPR to adapt to different situations. However, the reconfiguration control is external and not based on any QoS indicator. Authors in [4] also highlight the cost advantage of a reconfigurable FPGA system against a multi-board equivalent solution. Some examples manage reconfigurable modules (RM) like tasks. In [5], the authors use DPR to implement the tasks on the FPGA, in a sequential order given by the application. This kind of application is meant to reduce the global power cost and increase performance using FPGA, but the reconfiguration decision only depends on the system current state. In [6], authors describe a reconfigurable architecture that replaces HW accelerators to provide required functions to the system. This architecture uses a list of functions to be implemented and FPGA available space to decide to reconfigure. Some applications use DPR to enable self-repairing. To do so, they use either scrubbing or duplication / triplication to observe faults [7]. This can be considered a QoS indicator, but the system only uses DPR to rewrite the faulty configuration to recover. By contrast, our reconfigurable system aims to find the most suitable configuration. An approach that considers application QoS is described in [8]. Nevertheless, the reconfiguration decision assumes that the system knows the value of the QoS for a given configuration. In real-life applications this assumption is not always true, besides in this work DPR was not fully implemented. In [9], authors have designed a system which can decide whether to reconfigure a system according to diagnostic indicators that include application QoS values. The decision method is based on Markov decision process and Bayesian networks, and consider available QoS criteria. However, once again, this work doesn't fully demonstrate the full DPR implementation.

Tools have been proposed to support the implementation of DPR-based methodologies, for instance [10] shows how to use DPR in SDSOC. However it is strongly dependent on tools versions and vendors engineering choices. It also implies a complete re-synthesis for each specification modification. Therefore, it is intractable in practice. On the contrary, it underlines the lack of consistent-in-time technical details from FPGA vendors about DPR and CAD tools versions.

Our objective is to optimize performances (QoS, response time, etc.) of a radar-based system according to the context obtained by sensors data. It means that the proposed method must results in configuration control based on a relation between a QoS observed online and the HW configuration. Such an approach is complex since it involves signal processing experts and HW/SW designers who have different concerns and tools.

III. Methodology

A. Separation of concerns

To take advantage of a QoS driven reconfiguration, the signal processing experts must be DPR aware so that they can have in mind the possibility to switch between different algorithm modes or configurations. However, these experts should not have to care about the implementation details. Thus, the design of a QoS driven DPR needs to be based on the separation of concerns scheme depicted in Figure 2. Once the signal processing experts know they can use several HW configurations in one mission, they can explore more ambitious strategies. First, they need to specify the algorithms to implement and the different versions they need, in terms of performances and resources cost. Secondly, they have to define the QoS indicators they need to implement such decisions. The SoC designers then can provide the required feedback in the same way as in a classic development process.

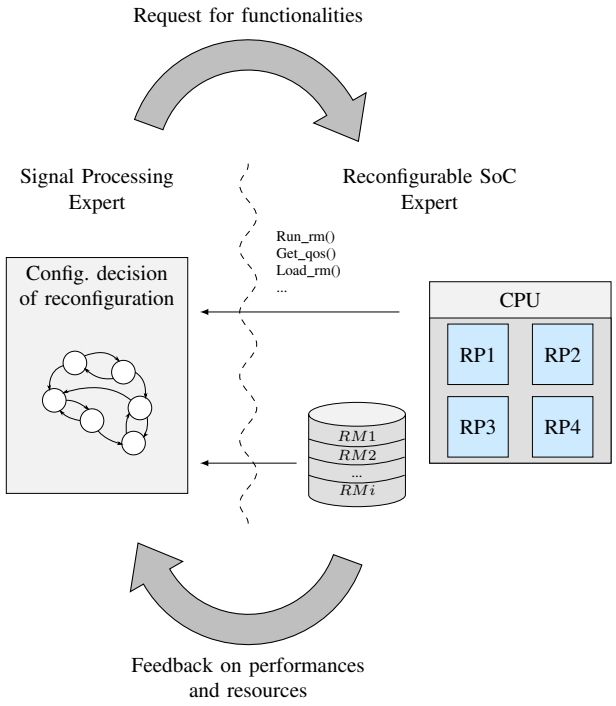

Fig. 2. Schematic representation of the separation of concerns in the QoS driven DPR methodology

After a few cycles, the reconfigurable SoC experts have to bring to the signal processing experts a library of the different

configurations (RMi in Figure 2). They must also provide simple API for the experts to implement the reconfiguration decision system.

B. Reconfigurable architecture model

The architectures considered in this paper target SoC FPGA devices. They are composed of a FPGA (PL) and a processing system (PS). Several partitions are defined to be reconfigurable. The reconfigurable partitions (RP) communicate with one another and with the PS through buses. The more generic the buses are, the more heterogeneous the RM can be. The reconfiguration can be triggered either by the PS or by the PL part. The reconfiguration controller must also access the partial configuration bitstreams that can be stored in the main shared DDR memory or in dedicated one. However, a QoS driven methodology must be accessible to algorithm experts who are not familiar with FPGA. So, if the configuration delay is compliant with the application time constraints, a SW implementation is the best solution. As a SW program, the configuration decision and the configuration control can simply be tested and modified by algorithm experts. Figure 3 illustrates an example of such an architecture with six RPs implemented on a SoC FPGA.

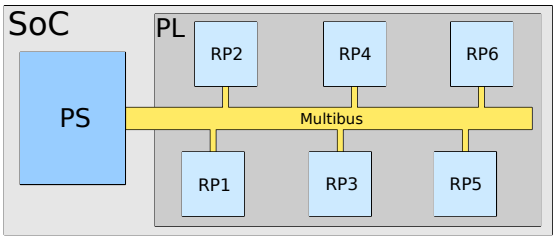

Fig. 3. Example of a SoC reconfigurable architecture.

C. Reconfiguration control system

Besides the execution flow, the QoS-based controller can be decomposed into four parts: 1) Monitor, 2) Analyze, 3) Plan and 4) Execute. The monitor function is in charge of collecting data and computing at least one QoS indicator per RM. These indicators can be computed directly by the algorithm, for example the error in an automation process. If it requires additional computation (e.g. estimation of the next state), the designer can choose to implement it in either the PL or the PS part. The analyze function must determine if a reconfiguration is needed, with regard to the QoS indicators. The plan function decides when to trigger a reconfiguration, and which configurations must be replaced. Finally, the execute function is in charge of loading the right bitstream at the right time according to application execution flows and architecture constraints. In all cases the decision-making based on the indicator is co-designed with application experts and so is preferably implemented in SW. This allows experts to evaluate different strategies. Figure 4 illustrates the DPR control flow, step-by-step.

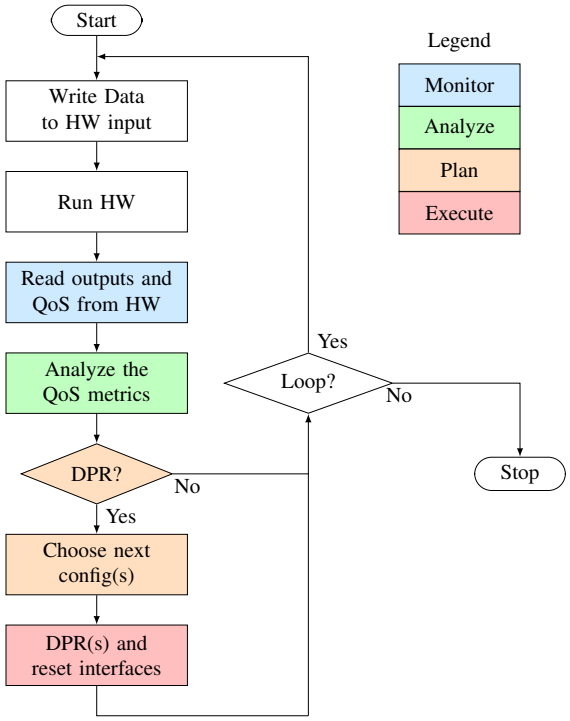

Fig. 4. Diagram of the DPR control system

IV. Case study : Kalman-based tracking

We are interested in optimizing the tracking of a target by selecting the best Kalman Filter (KF) chosen in a library of KFs which differ from each other by their process models, i.e., the way they model the trajectory of the target. Ideally we could execute all the models in parallel and select the best one after each iteration according to QoS indicators. However, our major constraint is that all the KFs available in the library cannot be executed simultaneously because of real-time constraints and hardware resource availability. So, only a subset of the library must be implemented. In our study we consider that two KFs can run simultaneously. KFs can be implemented either in the PS or the PL part. The advantage of the HW version is twofold. First, it improves the execution time and so the response time of the system. Secondly, it allows the execution of multiple KFs in parallel while having the PS available for running other application and system tasks. We choose a model with different KFs running in parallel in separate RPs.

In real tracking systems, such Kalman filters can be very large (more than 10 states in the aircraft dynamic navigation filter in [11]). Hardware optimization based on constant matrix elements is then required. Reconfiguration based only on parameter changes is, therefore, inefficient. Moreover, DPR allows to implement Kalman filters with completely different behaviours (e.g., EKF, UKF and IEKF). In this study, we use simpler Kalman filters to illustrate the method without loss of generality.

A. Reconfigurable architecture model

The architecture used for the study is based on the one defined in III-B. In this example, two RP are defined. These RP can host a KF implementation. Each RP can be configured with seven different state models, each model corresponding to a different trajectory assumption. This means that they do different computations and use different matrix sizes. The KF gets the measurements from the PS through a direct memory access (DMA) interface. This DMA is included into the RP. The buses use a generic AXI4 interface for PL to PS communications. In this example, there is no communication between both RP.

B. QoS estimation and DPR decision

The QoS estimator at time k is chosen as the log likelihood of the measurements collected at time k. This criterion can be used as an efficient estimator of the KF model correctness [12]. It can be computed using the innovation and the innovation covariance computed by each KF under test:

$$f_{QoS}(k) = (\mathbf{i}_k^{\mathrm{T}} \mathbf{\Sigma}_i^{-1} \mathbf{i}_k) \tag{1}$$

where:

$$\begin{aligned} f_{QoS} &= \text{QoS function} \\ i &= \text{innovation vector} \\ \mathbf{\Sigma}_i &= \text{innovation covariance} \end{aligned}$$

We observe that this function requires matrix multiplication, so it can benefit from a parallel HW implementation. This is a typical point of useful discussion that authorize the proposed methodology. The QoS indicator is specified by the application designers and the reconfigurable SoC designers can work on the best implementation. In this case, they can reuse the inverse of the innovation covariance matrix which needs to be computed to get the optimal Kalman gain (equation 2). Therefore, we chose to implement f_{QoS} in hardware, using Newton division to increase performance while reducing PL footprint.

$$K_k = (P_{k|k-1} H_k^{\mathrm{T}} \mathbf{\Sigma}_i^{-1}) \tag{2}$$

where:

$$\begin{aligned} K &= \text{optimal Kalman gain} \\ P_{k|k-1} &= \text{predicted error covariance} \\ H &= \text{observation model} \\ \mathbf{\Sigma}_i &= \text{innovation covariance} \end{aligned}$$

Once this indicator is computed, the IP sends it back to the PS. The application then has access to a QoS value for each of the KF modules. This allows the program to identify which filter is the most adapted to the current situation. In this way, the PS can replace the worst one and implement another IP to test another trajectory hypothesis.

Figure 5 shows an illustrative example of the control principle while considering 2 DPR partitions that can implement different Kalman models. The first and second lines draw the evolution of the configurations implemented on the RP1 and RP2 over time.

The method we chose to control the reconfiguration from the QoS values relies on 4 phases:

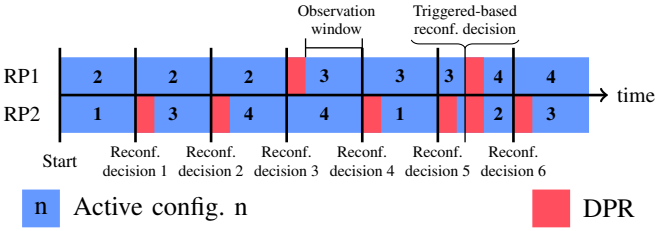

Fig. 5. Example of QoS driven DPR control

1) Initialization: Two configurations are chosen at the beginning of the tracking. This choice can rely on mission indicators. For example, in radar systems, the target trajectory can be classified with the help of a priori knowledge.

2) Observation and decision: QoS-driven adaptation is strongly specific to the application and must therefore be flexible, as a software it can be easily modified according to test results. In our case study different strategies are possible. We have implemented a solution that combines long-term observations and reactivity. The QoS estimator gives a new value at each Kalman call. However, we cannot reconfigure each time we get a new QoS value. Nevertheless, a very bad QoS value means that we need to change the configuration as soon as possible. Therefore, we need to have two kinds of reconfiguration: The first one computes the average QoS over an observation window (shown in Figure 5), the KF with the worst QoS score is replaced by a new one according to a round-robin mechanism (Reconf. decision n in Figure 5). The second one is based on thresholds, when both KF QoS exceed the maximum values then KF are interrupted before the end of the window and replaced by two new ones (triggered-based reconfiguration decision in Figure 5).

3) Reconfiguration: The reconfiguration takes time to perform. In some radar applications, there can be no down time. In our system, the reconfiguration time can be hidden since there are two RP running concurrently and only one will be stopped on reconfiguration. The only situation where a timeout occurs is when the threshold-based reconfiguration is triggered for both RP. In such case, none of the Kalman filters can achieve a good tracking so a HW reconfiguration is required.

4) Convergence: When using a Kalman filter, there is a convergence time. This happens because the starting point of the algorithm is not perfect. This time causes the filter to output inaccurate points and can cause the QoS estimator to output bad scores during the convergence interval. In this experiment, we consider this training phase as an additional reconfiguration time. We don't observe the QoS during this phase to avoid penalizing new implemented filters.

C. Experience setup

1) HW/SW platform: The reconfiguration strategy as well as the KFs are implemented in a Zynq Ultrascale+, a SoC made up of a multi-core ARM and a FPGA that supports DPR. Figure 6 presents the layout of the FPGA, with the two reconfigurable partitions delimited by the purple areas. With

this layout, the reconfiguration of a RP takes 8.7ms using PCAP. This time depends on the size of the partial bitstream as well as the bandwidth of the configuration method. The configuration remains static outside these two boxes. In this experiment, we use only one processor core as well as the PL part. In real-life tracking systems, the other cores would likely be running other tasks such as detection.

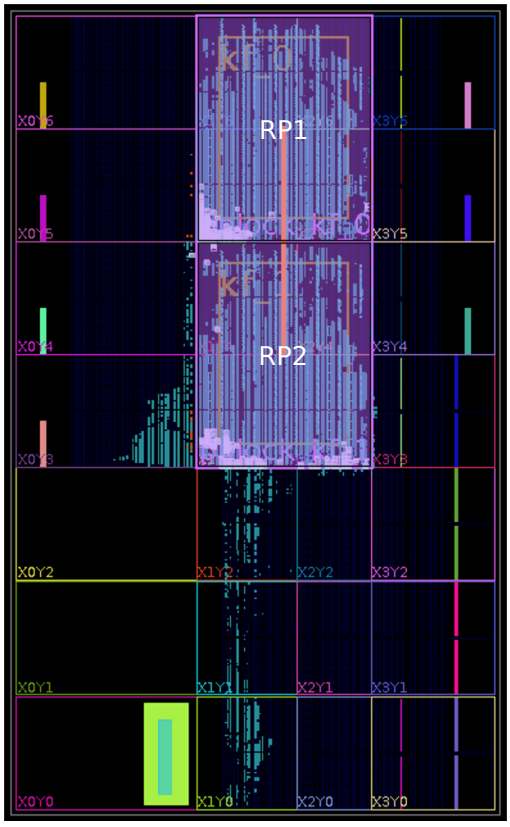

Fig. 6. Layout of the FPGA with the two reconfigurable partitions (RP1 and RP2).

The different Kalman filters are generated using Vivado HLS, a high-level synthesis tool. This allows to generate different configuration quickly, while ensuring that the different configurations are compatible. Indeed, the RM needs to have the same inputs, outputs and operating frequency. In our tracking example, the target lies on a plane whose axis are x and y, $x \perp y$ (Figure 7). The observation vector is $z = [x, y, sx, sy]$, with sx and sy the speeds respectively along the x and y axis.

2) KF / EKF models: The generated Kalman filters are listed below. They correspond to different trajectories assumptions. Note that a state is considered constant when it is not to be seen in the list.

Linear Kalman filters:

KF-1 : State vector $X_n = [x_n, y_n, sx_n, sy_n]$
State transition equations :

$$\begin{cases} x_n = x_{n-1} + sx_{n-1} * dt \\ y_n = y_{n-1} + sy_{n-1} * dt \end{cases} \quad (3)$$

KF-2 : State vector $X_n = [x_n, y_n, sx_n, sy_n, ax_n]$
State transition equations :

$$\begin{cases} x_n = x_{n-1} + sx_{n-1} * dt ; sx_n = sx_{n-1} + ax_{n-1} * dt \\ y_n = y_{n-1} + sy_{n-1} * dt \end{cases} \quad (4)$$

KF-3 : State vector $X_n = [x_n, y_n, sx_n, sy_n, ay_n]$
State transition equations :

$$\begin{cases} x_n = x_{n-1} + sx_{n-1} * dt \\ y_n = y_{n-1} + sy_{n-1} * dt \end{cases} ; sy_n = sy_{n-1} + ay_{n-1} * dt \quad (5)$$

KF-4 : State vector $X_n = [x_n, y_n, sx_n, sy_n, ax_n, ay_n]$
State transition equations :

$$\begin{cases} x_n = x_{n-1} + sx_{n-1} * dt ; sx_n = sx_{n-1} + ax_{n-1} * dt \\ y_n = y_{n-1} + sy_{n-1} * dt \end{cases} ; sy_n = sy_{n-1} + ay_{n-1} * dt \quad (6)$$

Extended Kalman filters:

EKF-1 : State vector $X_n = [x_n, y_n, sx_n, sy_n]$
State transition equations :

$$\begin{cases} x_n = y_{n-1} * y_{n-1} \\ y_n = y_{n-1} + sy_{n-1} * dt \end{cases} \quad (7)$$

EKF-2 : State vector $X_n = [x_n, y_n, sx_n, sy_n]$
State transition equations :

$$\begin{cases} x_n = x_{n-1} + sx_{n-1} * dt \\ y_n = x_{n-1} * x_{n-1} \end{cases} \quad (8)$$

EKF-3 : State vector $X_n = [x_n, y_n, sx_n, sy_n]$
State transition equations :

$$\begin{cases} x_n = {}^{-9.81}/_{2} * (y_n - cst)^2 * {}^{1}/_{(30 * cst)} \\ \qquad + sy_0 * sin({}^{\pi}/_{2}) * (y_n - cst) + cst \\ y_n = y_{n-1} + sy_{n-1} * dt \end{cases} \quad (9)$$

3) Benchmark trajectory: The trajectory used for the tests is based on a combination of sub-trajectories that follow different Kalman models. In this paper, we use a trajectory where the first part corresponds to the equation $y_n = x_n^2$. In the second sub-trajectory, both x and y have constant speed. In the third part, x and y have constant accelerations. The fourth and last part follows the equation of EKF-3. The green curve in Figure 7 shows the complete composed trajectory. Finally, the observed values, which are the blue crosses in Figure 7, result from the addition of the target trajectory and a proportional Gaussian noise. These data are processed by the implemented Kalman filters used for the experiments.

4) Experience procedure: When the designs are synthesized and implemented, the partial bitstreams are loaded in a flash memory. At power-up, the PS loads all the bitstreams in DDR RAM. When a reconfiguration is triggered, the PS feeds the processor configuration access port (PCAP) with the corresponding bitstream.

On power-up, the PS starts a benchmark to test our design. First, the input data is extracted from the SD card and stored in DDR4. Then, the application reconfigures the two RP to set the system into a known state. Depending on the scenario, the two initial configurations may differ. Finally, the processor transmits the data samples and gets the results along with QoS

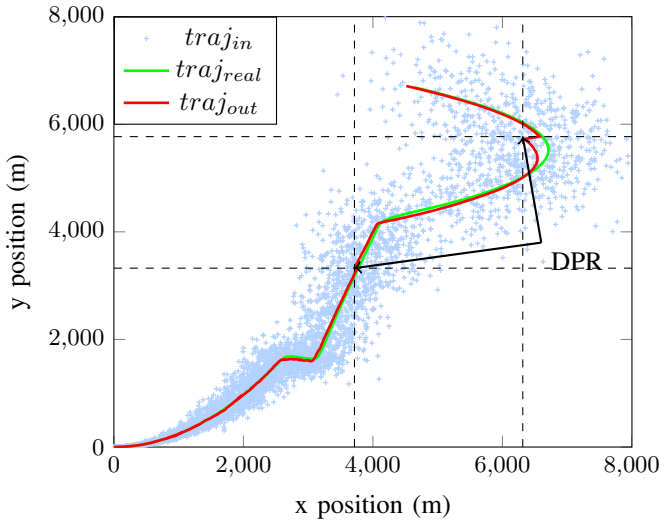

Fig. 7. Tracking positions with DPR (scenario 3)

values. The output of the best KF is first stored into DDR4, and then saved to the SD card to be plotted and analyzed.

In our test bed we consider three different scenarios. In the first one, the two initial configurations are the KF-1 and the EKF-3. Since these systems have strongly different behaviours, the system should easily find the best KF. The second scenario starts with EKF-1 and EKF-3. None of these configurations is a good model at the beginning of the tracking. This scenario can show if the system is able to react fast to a situation where none of the configurations is good and there is a risk to lose the target. The third and last scenario begins with KF-3 and KF-4. As these configurations are close to each other, it will make the configuration choice more challenging.

D. Results and analysis

These tests involve a reconfigurable architecture with active QoS-driven DPR and the best Kalman without reconfiguration (model described in equations of KF-4). The results obtained with the benchmarks are given in Table I. To compare the different approaches, we measure the likelihood and compute its median. We also collect the maximum and more importantly the minimum measurement of the likelihood, as a small value could cause the system to lose the target. We can observe that for the third scenario, the likelihood median is only slightly better when using DPR. However, for the second scenario where the system starts with bad configurations, the QoS value is much better with the reconfiguration system. Hence, the DPR system is better at tracking a target with an a priori unknown trajectory. We also notice that the minimum likelihood measurement is higher with reconfiguration. This implies that the DPR system has less risk of losing the target than the one without DPR. All three scenarios show the same minimum and median values of the likelihood. This shows that our system is robust against its initial condition.

Figure 7 shows the third scenario case of a tracking using DPR. The green curve represents the real trajectory, while

TABLE I
RESULTS OF TRACKING

Tracking Scenario[1]	Sub-trajectory[2]	Likelihood	*DPR*	*No DPR*
Scenario 1	**Part 1**	***Min***	0.4156	0.4232
		Max	0.9970	0.9984
		Median	0.8074	0.8077
	Part 2	***Min***	0.3898	0.3889
		Max	0.9936	0.9938
		Median	0.6915	0.6910
	Part 3	***Min***	0.2580	0.2529
		Max	0.9835	0.9777
		Median	0.6103	0.6100
	Part 4	***Min***	0.2144	0.2128
		Max	0.9939	0.9812
		Median	0.5848	0.5790
Scenario 2	**Part 1**	***Min***	0.4301	0.1640
		Max	0.9939	0.9796
		Median	0.8025	0.4567
	Part 2	***Min***	0.3915	0.1198
		Max	0.9853	0.8620
		Median	0.6968	0.3708
	Part 3	***Min***	0.2582	0.1177
		Max	0.9934	0.9445
		Median	0.6119	0.2874
	Part 4	***Min***	0.2144	0.0868
		Max	0.9894	0.9872
		Median	0.5845	0.2954
Scenario 3	**Part 1**	***Min***	0.4156	0.4227
		Max	0.9976	0.9982
		Median	0.8063	0.8077
	Part 2	***Min***	0.3898	0.3897
		Max	0.9936	0.9966
		Median	0.6915	0.6922
	Part 3	***Min***	0.2580	0.2529
		Max	0.9835	0.9955
		Median	0.6103	0.6098
	Part 4	***Min***	0.2144	0.2114
		Max	0.9939	0.9748
		Median	0.5848	0.5803

[1]Scenarios are described in subsection IV-C4
[2]Sub-trajectories are defined in IV-C3

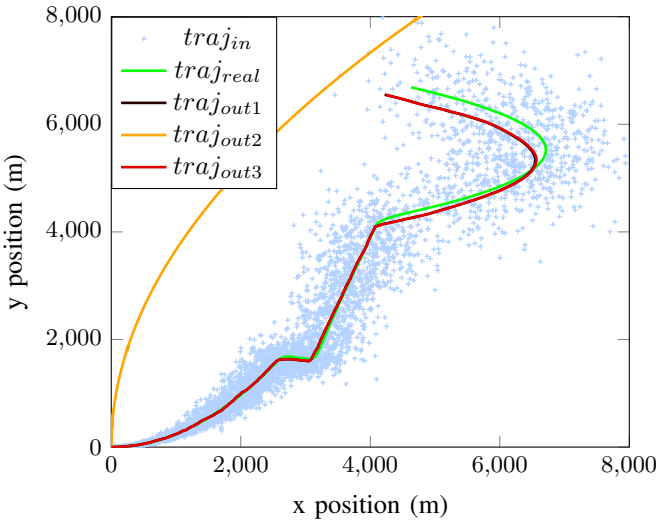

Fig. 8. Tracking positions without DPR (scenarios 1 to 3)

the red one shows the outputs of the reconfigurable Kalman system. In this example the Triggered-based reconfiguration (namely when the distance to the model exceed the threshold value) occurs once at $x = 6314$ when the trajectory radically changes. All other reconfigurations occur periodically at the end of each time-window and can provide the best KF that will be eventually selected according to the QoS comparisons, this is for instance what happens at $x = 3715$. Figure 8 shows the output trajectories for the three scenarios and without DPR. The differences between Figures 7 and 8 highlight the influence of reconfiguration on the Kalman filtering.

These improvements are allowed by the knowledge of the QoS value throughout the process. Thanks to this value, the system can choose the best model at a given time. Moreover, the likelihood also allows to know when the models are not adapted and the system can adapt early. This method requires good knowledge about the application, and extracting a QoS indicator may not always be straightforward.

V. Conclusion

The paper first demonstrates the opportunity of using DPR to virtually extend available HW resource of an embedded system in the domain of Radar. The embedded system can dynamically implement HW accelerators according to application requirements in order to speed-up the application execution while saving CPU time.

The study also shows the importance of considering the QoS to drive the configuration. This point is rarely considered in the DPR literature since it requires transdisciplinarity. However, it is crucial to fully benefit from the expert knowledge (radar, signal processing in our case study). This knowledge can be captured and efficiently used only with a methodology that clearly separates concerns. Based on this methodology, the next steps will address the design of an architecture pattern for the domain of Radar including complex task such as Detection based on Range-Doppler maps and realistic Kalman filters with large state space. This new application context will also require to explore different adaptation strategies.

Finally, we intend to define API to collect QoS and API to select configuration. This interface must be generic for application experts. Nevertheless, the implementation will be specific to the target architecture and, therefore, handled by HW designers.

References

[1] D. Govind Rao, Aalhad P. Deshpande, N. S. Murthy, and A. Vengadarajan. Digital beam former architecture for sixteen elements planar phased array radar. In *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, pages 532–537, Konya, Turkey, May 2013. IEEE.

[2] An architectural blueprint for autonomic computing. White Paper 3rd Edition, IBM Corporation, June 2005.

[3] Russell Tessier, Kenneth Pocek, and Andre DeHon. Reconfigurable Computing Architectures. *Proceedings of the IEEE*, 103(3):332–354, March 2015.

[4] Emmanuel Seguin, Russell Tessier, Eric Knapp, and Robert W. Jackson. A Dynamically-Reconfigurable Phased Array Radar Processing System. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 258–263, Chania, Greece, September 2011. IEEE.

[5] Francisco Fons, Mariano Fons, Enrique Cantó, and Mariano López. Real-time embedded systems powered by FPGA dynamic partial self-reconfiguration: A case study oriented to biometric recognition applications. *Journal of Real-Time Image Processing*, 8(3):229–251, September 2013.

[6] Xuzhi Zhang, Xiaozhe Shao, George Provelengios, Naveen Kumar Dumpala, Lixin Gao, and Russell Tessier. Scalable Network Function Virtualization for Heterogeneous Middleboxes. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 219–226, Napa, CA, USA, April 2017. IEEE.

[7] Norma Montealegre, David Merodio, Agustin Fernandez, and Philippe Armbruster. In-flight reconfigurable FPGA-based space systems. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 1–8, Montreal, QC, June 2015. IEEE.

[8] Jean-Philippe Diguet, Yvan Eustache, and Guy Gogniat. Closed-loop-based self-adaptive Hardware/Software-Embedded systems: Design methodology and smart cam case study. *ACM Transactions on Embedded Computing Systems*, 10(3):1–28, April 2011.

[9] Chabha Hireche, Catherine Dezan, Stéphane Mocanu, Dominique Heller, and Jean-Philippe Diguet. Context/Resource-Aware Mission Planning Based on BNs and Concurrent MDPs for Autonomous UAVs. *Sensors*, 18(12):4266, December 2018.

[10] Tobias Kalb and Diana Gohringer. Enabling dynamic and partial reconfiguration in Xilinx SDSoC. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–7, Cancun, Mexico, November 2016. IEEE.

[11] Lennon Cork. *Aircraft Dynamic Navigation for Unmanned Aerial Vehicles*. PhD thesis, Queensland University of Technology, May 2014.

[12] Pierre Tandeo, Pierre Ailliot, Marc Bocquet, Alberto Carrassi, Takemasa Miyoshi, Manuel Pulido, and Yicun Zhen. Joint Estimation of Model and Observation Error Covariance Matrices in Data Assimilation: A Review. *arXiv:1807.11221 [stat]*, July 2018.