A Reconfigurable Multiprocessor Architecture for a Reliable Face Recognition Implementation

Antonino Tumeo¹, Francesco Regazzoni³, Gianluca Palermo¹, Fabrizio Ferrandi³, Donatella Sciuto¹
¹Politecnico di Milano - DEI, Milan, Italy {tumeo, gpalermo, ferrandi, sciuto}@elet.polimi.it
²Crypto Group, Universite Catholique de Louvain, Louvain-la-Neuve, Belgium, francesco.regazzoni@uclouvain.be
³ALaRI - University of Lugano, Lugano, Switzerland, regazzoni@alari.ch

Abstract—Face Recognition techniques are solutions used to quickly screen a huge number of persons without being intrusive in open environments or to substitute id cards in companies or research institutes. There are several reasons that require to systems implementing these techniques to be reliable.

This paper presents the design of a reliable face recognition system implemented on Field Programmable Gate Array (FPGA). The proposed implementation uses the concepts of multiprocessor architecture, parallel software and dynamic reconfiguration to satisfy the requirement of a reliable system. The target multiprocessor architecture is extended to support the dynamic reconfiguration of the processing unit to provide reliability for the system. Experimental results demonstrate that, due to the multiprocessor architecture, the parallel face recognition algorithm can achieve a speed up of 63% with respect to the sequential version. Results regarding the overhead in maintaining a reliable architecture are also shown.

I. INTRODUCTION

Reconfigurable devices, such as Field Programmable Gate Arrays (FPGAs), have emerged as an interesting alternative target technology for implementing embedded systems, featuring increased flexibility versus relatively lower performance with respect to Application Specific Integrated Circuits (ASICs) [1]. Furthermore, due to their inherent parallelism, embedded systems are nowadays designed in a MultiProcessor Systems-on-Chip (MPSoCs) fashion, where several heterogeneous processing cores interconnected with memories, device controllers and custom accelerators are used to guarantee the high performance required by modern multimedia and communication applications.

An appealing application for such systems is represented by biometric recognition algorithms. Fingerprint and face recognition, in fact, gained much popularity in these last years, becoming one of the hot topics for image analysis and vision research groups. Several research projects [2], [3] focus on the design and the evaluation of new algorithms for face recognition, in terms of effectiveness and performance. Their objective is to develop efficient algorithms, easily integrable in embedded systems, that can be used for identification and security purposes in place or in cooperation with other mechanisms like id cards. Practical applications are, for example, access control or automatic surveillance of sensitive areas and buildings. Since these are very often “mission critical” applications, a high degree of reliability is required. Thus, not only effective algorithms are requested, but also the architectures that run them should tolerate, and eventually correct, possible system/application faults.

In this paper, we present the implementation of a face recognition algorithm on an FPGA multiprocessor architecture, describing its efficient implementation and discussing how we exploit the partial dynamic reconfiguration available in the platform to improve the application reliability. In particular, we describe the mapping of a real-life application on a dynamically reconfigurable multiprocessor architecture where the reconfigurable part is composed by the entire processing unit.

The paper is organized as follows. Section II presents some related works in the area of MPSoC for reconfigurable architectures. Section III describes the multiprocessor architecture implemented on FPGA we adopted, while Section IV and Section V detail respectively the target application and how to increase the system reliability. Section VI summarizes our experimental results, and, finally, Section VII concludes the paper.

II. RELATED WORK

Nowadays MPSoCs, in particular heterogeneous multiprocessors, are the optimal architecture for applications with high processor demand. Because of this, the research community spent a lot of effort in this field, with particular attention to solutions tailored on the needs of multimedia. Addressed problems range from optimized architectures to design methodologies, as in [4], where a complete methodology for designing multimedia MPSoCs is presented. In particular, the authors describe how to use traces and simulation to fully characterize the set of tasks performed by multimedia systems, and demonstrate their methodology to design an experimental SoC for real-time video analysis.

In this paper we concentrate on MPSoC architectures suitable for FPGA, that have recently started to be explored. The majority of past works in fact focus on ASIC implementations, mainly because reconfigurable architectures have just lately reached the capability to hold a complete MPSoC and thus become an attractive platform for multiprocessor architectures.

Regarding MPSoC implementation in general, a number of previous works are related to the task synchronization and the shared memory coherency requirements. In the case of FPGAs Clark et al. [5] and James-Roxby et al. [6] proposed to handle the synchronization of the task and shared resources using ad hoc hardware modules. To address the problem of cache coherency, Hung et al. [7] proposed a dedicated module designed to snoop each memory write operation, invalidating the cache lines with interrupt signals. A multiprocessor platform with FPGA soft cores addressing synchronization and memory hierarchy and multiprocessor interrupt management was proposed by Tumeo et al. [8], [9].

Multiprocessors based on reconfigurable fabrics for image processing have been introduced in [10]–[12]. The multiprocessor systems described in [10], [11] support reconfiguration of the hardware accelerators but not of the processing unit. Moreover, the system described in [10] has only two processors, one for the computation and one for the partial reconfiguration. In [11] the current realization is in a very embryonic phase since the designer is limited to the use of two processors and only one partial reconfigurable module. The RAMPSoC system, described in [12], supports reconfiguration of an FPGA processing unit and of communication network. However, none of those systems considers reliability to processor faults in their design.

III. TARGET ARCHITECTURE

In this section, we detail the basic architecture of the multiprocessor embedded system on FPGA and present the operating system layer that is built on top of it.
The target architecture has been designed with the Xilinx Embedded Developer Kit (EDK) version 8.1 and is shown in Figure 1. In this design, several Xilinx MicroBlaze soft-core processors version 4.0 (3 stages pipeline) are connected to a shared On-Chip Peripheral Bus (OPB). Each processor accesses a local memory, made of Block RAMs (BRAMs, integrated in the FPGA) through the Local Memory Bus (LMB), that allows reading in a single clock cycle and writing in two clock cycles. These memories are used for local and private data, like stack and heap of the threads in execution by the processors. The bus connects the processor to the shared external Double Data Rate (DDR) memory. The 256 MB of external memory have an average latency of 12 clock cycles and they are used both for instructions and shared data. Other elements of the architecture are the UART controller for the serial port, the SysAce controller for reading and writing the Flash Card and the MicroBlaze Debug Module (MDM) for debugging the system. An important component of the architecture is the Internal Configuration Access Port (ICAP) wrapper, which can be connected to the shared bus and allows to the system to perform partial dynamic reconfiguration (a.k.a. self-reconfiguration). Finally, other Intellectual Property (IP) cores, like for example a controller for the image acquisition system can be connected to the shared bus. In our implementation, we read the input and the database data from the Flash Card. This severely limits the performance, since file reading is particularly slow, nevertheless it allows us to build a stand alone system for the evaluation of the application. In the final implementation of the system, database data could be saved in memory at system boot up, while the input image will be loaded from a sensor, eventually with mastering or DMA capabilities, that can directly output the data in the shared memory without intervention from any processors.

To allow multiprocessor aware operations, two ad hoc coprocessors have been designed and connected to each MicroBlaze through the Fast Simplex Link (FSL) point-to-point connections; the Synchronization Engine (SE) and the CrossBar (CB). The SE is a centralized lock and barrier hardware manager, while the CB is a point-to-point data passing mechanism that allows sending small data packets without polluting the shared bus and memory. Thanks to FSL, they can access the processors’ register files with a single cycle latency.

The Early Access Partial Reconfiguration (EAPR) flow from Xilinx easily allows configuring a portion of the architecture as reconfigurable. The flow starts with an initial budgeting, during which the developer decides how to distribute the device area among the static and the reconfigurable parts of the architecture. In this stage, the macros for the clock generators, the reconfiguration port and the communication paths among the basic architecture and the reconfigurable elements (bus macros) are placed. It is followed by the implementation phase, during which the fixed part and the reconfigurable modules are synthesized separately with the previously defined area constraints. If some constraints are too strict, the implementation may violate them, forcing the developer to repeat the budgeting phase. Finally, there is the assembly phase, during which the full bitstreams, composed by the fixed part and all the possible permutations of the reconfigurable modules, and the partial bitstreams are generated.

Reconfiguration is normally driven by a processing element, which loads a partial bitstream from a file or from the memory and writes it to the ICAP, in chunks of 2 KB. For this architecture, we choose as reconfigurable modules the groups made by a processor, its private memory and the LMBs that connect these two elements, and we cache the partial bitstreams in memory at system boot to accelerate the reconfiguration.

B. Software Layer

On top of the architecture, a small multiprocessor-aware operating system layer has been developed. This OS layer supports a fork/join, run to completion thread model with dynamic allocation of the ready tasks to the free processors. This thin kernel is based around two shared tables: the first stores the free processors, the second saves the tasks that have been spawned, executed and joined. When a processor ends its current tasks, it checks if a ready thread is available in the task table. If it is available, the processor starts executing it, if it is not, then the processor updates its record in the processor table and sets itself in blocking wait for a task id on the CrossBar. In this way, if no tasks are available, the processor stops working and does not create contention on the bus and the shared memory. When a new task is created, it is inserted in the task table. If there is a free processor in blocking wait, the id of the new thread is sent to it. If all the processors are busy, it is simply added to the task table as a ready thread. This mechanism is flexible and allows supporting two different models for spawning the threads. The first one is a classic master/slave model, in which the master thread, always running on the master processor, generates the children, performs some work, waits for their termination and can then generate again new threads. Specific create and join primitives, built around the hardware synchronization support, have been implemented. The second threading model is more flexible: after executing the initialization task, the master processor inserts the first set of new tasks and then executes an appropriate primitive that allows setting itself as a worker. From then on, all the tasks can fork new children, inserting them in the task table from where any available free processor will execute them. Father threads do not have to wait for the termination of their children, since each processor automatically checks for new ready tasks as soon as they terminate the previous one. Synchronization among threads for data parallel parts of the program is performed through the CB, at the end of the tasks. The last surviving thread can then insert new tasks to continue execution and eventually start a new parallel part.

Another interesting aspect integrated in this OS layer is the possibility to assign specific threads to specific processors. Through this mechanism, pre-defined tasks will be sent only to a specific set of processors, when they are ready. It is also possible to generate different master threads, and run them on different processors. This mechanism is exploited for the reliable implementation of the target application.

IV. FACE RECOGNITION

Nowadays, face recognition is the least intrusive and fastest biometric technology: the identification process is done in two steps: firstly the face of the person is scanned and secondly the acquired image is compared with the ones stored in a library to find a match.

The application we consider in this work is an adaptation from the Face Recognition benchmark of the ALPbench [13] suite. This benchmark corresponds to Version 5.0 of the CSU Face Identification Evaluation System, which includes four distinct face recognition algorithms and compares them with standard statistical methods. The algorithms are: Principle Components Analysis (PCA), Eigenfaces, which combines Principle Components Analysis and Linear Discriminant
Analysis (PCA+LDA), Bayesian Intrapersonal/Extrapersonal Classifier (BIC) and Elastic Bunch Graph Matching (EBGM). As Figure 2 shows, the CSU Face Identification Evaluation System can be split into four phases: image pre-processing, that reduces unwanted image variation on the starting set of reference images, algorithm training, that is specific to the algorithm, algorithm testing, where the algorithms are performed and the result matrices storing the distances of the images are generated, and analysis of the results, where the previously generated matrices are evaluated to determine which algorithm obtained better results.

We adapted to the APIs of our multiprocessor architecture the source code provided by University of Illinois, that leverages on a master-slave mechanism for task allocation and data handling (i.e., standard pthreads primitives). The parts of the application ported on the target architecture are related to the testing phase. Preprocessing and training phases on the starting set of face images, in fact, can be performed off-line before deploying the image recognition system. The analysis phase, instead, is used to statistically verify which one of the implemented algorithms overall performs better, and in a realistic system it can be interesting only after having processed a great number of images to determine which algorithm is more suitable to the specific environment.

V. RELIABLE IMPLEMENTATION

A valuable aspect for a realistic implementation of a face recognition system is reliability. In fact, it is important to obtain from these systems not only a good performance from the identification algorithms, so to produce less recognition errors possible, but also the lowest probability of corruption of the results. In this paper we explore the use of partial dynamic reconfiguration provided by the FPGA to guarantee such reliability. This mechanism can be applicable in case of environments in which there is probability of repeated Single Event Upset (SEU) errors (e.g. bit flips of Look-up Tables), like space or nuclear power plants or warehouses for chemical waste. There, the high radioactivity levels may influence hardware operations. Furthermore, in many of these environments, there is high risk for intrusions of non-authorized personnel and identification systems may be required. We considered the use of partial dynamic reconfiguration only on the processors and the local memories associated to them, since external memories are selected considering the needed redundancy and thus they do not need additional protection. The synchronization and communication modules, being ad-hoc, have been designed with redundant logic. Furthermore, processors are the components that require the biggest area on our architecture (each processor occupies more than 13% of the total area). Providing them with a fault tolerance mechanism can excessively degrade the performances and increase the area requirements. We thus exploit the concept of modular redundancy allowed by the use of a multiprocessor architecture. The basic idea is to consider the amount of area available on the reconfigurable device, and the performance constraints of the application. Through profiling, we can scale the number of processors in the MPSoC to meet the performance requirements. We can then allocate other processors that executes redundant code until all the available area is filled. The OS layer allows to execute different instances of the same code on different processors with different parallelism granularity. However, to guarantee the same performance from all the redundant instances of the application, it is obviously advisable to use the same number of processors for each one. We then add to our implementation a support for detecting errors, i.e. a voter among the final results of all the instances of the application. If an error is detected, then the reconfiguration is triggered to correct the faulty part. Given a target FPGA device with a fixed area, a trade-off can be found among performance, level of redundancy and reconfiguration latencies.

For our experiments on reliability we evaluated two possible situations: one in which we used four processors for the application and one in which we used two processors for the application. In the first case the reliability is provided through a Dual Modular Redundancy (DMR). DMR allows recognizing the presence of an error, but not to individuate which is the faulty part. With DMR we can use up to four processors for each instance of the application, thus achieving better performances. However, when an error is detected, using this scheme we cannot identify the faulty part. In this situation we can select to reconfigure all the processors in the system, with a considerable amount of reconfiguration overhead, or to stop the system and perform an (usually expensive) integrity check on all the processors to find the faulty one. In the second case, where only two processors are used for the application, the performances are lower, but three parallel instances of the applications can be launched, implementing Triple Modular Redundancy (TMR). With TMR we can detect and localize a single error, and, knowing which pair of processors has generated the error, we can reconfigure only the two of them. Since only two processors are reconfigured, and no integrity check should be performed, the resulting fault correction (reconfiguration) performances are higher.

VI. EXPERIMENTAL EVALUATION

In this section we describe the results we obtained with our parallel implementation of the face recognition application described in IV and we evaluate the trade-off for reconfiguration time vs. reliability. We compiled our software for the MicroBlaze v4.0 CPU and generated the bitstreams for our FPGA device by using the ISE and EDK version 8.1 from Xilinx. The target frequency of the system is 50 MHz. We then run the application using different numbers of CPUs, 1, 2, 4 and 8 respectively.

We firstly concentrate on the analysis of the speed up of the software components executed in parallel, because
no advantage from the parallelism can be achieved by the sequential part of the code. Figure 4 reports, for each parallel sub-function, the speed up as a function of the number of processors present in the system. When the number of MicroBlazes (MB) increases, the speed up generally increases too. The only exception is represented by Projection of the Probe Image (ProjectProbeImage), in which performance decreases when going from 2 to 4 processors and then rises again with 8 processors. This is due to the fact that this function performs synchronization (barriers) without killing the tasks, slightly influencing the access patterns to the bus, the memories and the synchronization module. Looking at the speed up obtained considering all the parallel part (the AllParallelPart line), we can conclude that the best trade-off in terms of speed up and number of processors is achieved with 2 MicroBlazes. In this case the speed up is up to 1.63 with respect to the case with only one CPU, while moving from 2 to 4, and from 4 to 8 processors, although the computation is faster, the slope of the curve is decreased. Shared memory is obviously the constraint here, and even if data are moved to the private memory to accelerate the computation, the contention remains high. DMA based solutions may eventually be explored in place of complex data cache coherency protocols that would require significant area overhead for the non-coherent MicroBlaze processors.

Concerning the area, the architecture with 8 processors occupies almost 100% of the target Virtex-II PRO XC2VP30 (13,696 slices1), while a single processor architecture uses around 11,000 slices. The graph also suggests configurations to use for the reliable implementations could be the 2 processors solution for TMR and the 4 processors solution for DMR. A MicroBlaze configured with basic features has an occupation of around 1,300 slices. To reconfigure that area, around 2,600,000 cycles are required by another processor accessing the ICAP, depending on the placing of the core. In fact, looking at how reconfiguration support is implemented on the Virtex-II PRO, all the FPGA columns in which the reconfigurable module is placed are loaded and configured, but only the behavior of the reconfigured module changes, while the other parts continue to work without glitches. Reconfiguration time for a MicroBlaze is almost four orders of magnitude lower than the execution time for the complete face recognition on a 8 processors architecture.

Table I shows how much the reconfiguration latency for TMR and DRM influences the performance of the 2 and 4 processors architectures. We evaluate the performance and the reconfiguration overhead, supposing that a single error has happened on one of the processors running an instance of the application, and thus reconfiguration is triggered to correct it. We do not take in consideration higher error rates or probabilistic analysis. It is easy to see that, in both the cases, the overhead is lower than 0.1% for the Full Application.

TMR is reconfiguring only 2 processors, while with DMR we are reconfiguring 7 out of 8 processors (after individuating a safe one) and we can see that reconfiguration overhead scales almost linearly. When considering only the Parallel Parts, reconfiguring 7 processors means only little more than 0.3% of overhead. With these reconfiguration latencies (remember that our system caches the partial bitstreams in shared memory), a system with 2 redundant arrays of 4 MicroBlazes each still performs better than a system with 3 modules of 2 processors each, respectively 12% when accounting only for the Parallel Parts and 2% for the Full Application.

VII. CONCLUSION

In this work, we presented and discussed how to develop a face recognition embedded system. We parallelized, implemented, and mapped a Face Recognition benchmark on a realistic FPGA MPSoC. We also provided a quantitative analysis of the proposed implementation, comparing it with a different number of processors. Results show that the parallelized face recognition application running on our architecture is 63% faster than a single processor solution. Additionally we discussed how to exploit partial dynamic reconfiguration to provide reliability, and evaluated some of the implemented reliable solutions. To the best of our knowledge, this paper presents the first implementation for a real-life application of a dynamically reconfigurable multiprocessor architecture where the reconfigurable part is composed by the entire processing unit.

REFERENCES


TABLE I

<table>
<thead>
<tr>
<th>Reconfiguration Overheads for DMR and TMR Implementations with 2 and 4 MicroBlazes (2MB and 4MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Parallel Parts</td>
</tr>
<tr>
<td>Full Application</td>
</tr>
</tbody>
</table>

1 A slice on Virtex-II PRO and Virtex-4 is composed by two 4-input LUTs and two flip-flops.