DESIGN FLEXIBILITY USING FPGA DYNAMICAL RECONFIGURATION

N.Abel, L. Kessal, D.Demigny

ETIS — UMR 8051 CNRS — ENSEA / Cergy-Pontoise University — France
{abel, kessal, demigny}@ensea.fr

ABSTRACT

In this paper, we detail the implementation of an image processing algorithm on the dynamically reconfigurable architecture ARDOISE. Beyond the complexity of this algorithm, we pay our attention on the possibilities offered by dynamic reconfiguration paradigm. We insist particularly on the software management of reconfigurations and on the flexibility it leads. We spread these concepts, tested practically on ARDOISE, in more complex cases which will be the object of future studies.

1. INTRODUCTION.

For some years, the increasing applications complexity calls for new conception methods. On one hand, the ultra-specific architectures, such ASICs, are the only ones capable of achieving the required performance. On the other hand, their lack of flexibility questions their development cost as regards to their life expectancies on the market. The emergence of new architectures, qualified of *dynamically reconfigurables*, aims to offer a new compromise between flexibility and performance.

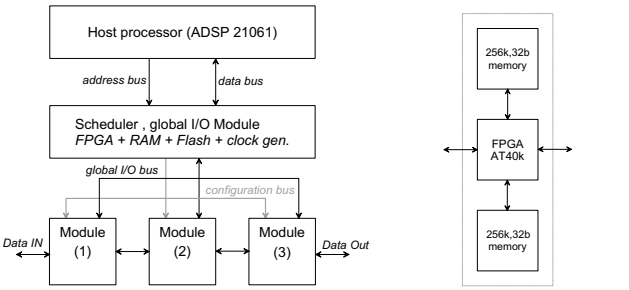
We are interested in the fine grain architectures. Based on the FPGAs (Field Programmable Gate Arrays), they use the fact that the configuration of these components is stored in a memory. Therefore, nothing prevents from changing this configuration, even during calculation, by filling this memory. Studies have already shown that to reconfigure dynamically a FPGA allows to increase its performance. We are going to show that it also brings some flexibility to these architectures.

We begin with a brief description of ARDOISE (sec. 2). This architecture is physically operational (a photo is given on figure 3). We then describe the chosen algorithm (sec. 3). It is to note that this algorithm was chosen, on one hand, because it allowed to operate numerous resultant concepts of dynamical reconfiguration. And on the other hand, because the simplicity of this allowed us to concentrate on the deep of our subject: dynamic reconfiguration. All this concepts are physically implemented on ARDOISE and described in section 4. Finally, we give some possible extensions and we conclude (sec. 5).

2. THE ARDOISE ARCHITECTURE.

2.1. System description.

The basic idea of ARDOISE is to swap algorithms, used in image segmentation, on the same hardware structure, by reconfiguring few devices several times during the processing of each image. This is equivalent to assign to the same hardware resource, an FPGA device, the execution of a sequence of algorithms based on virtual architectures according to a schedule defined before or during execution. We use the AT40K FPGA from Atmel which is particularly suited for run-time reconfiguration because it can be rapidly reprogrammed. This devices which include 45K equivalent gates can be totally configured in less than 1ms. Furthermore, it is possible to configure rapidly an arbitrary subset of the FPGA while the rest of the device is still computing. Such a partial configuration reduces time needed for reconfiguring the device and opens others interesting perspectives.



(a) global view. (b) detail of a module.

Fig. 1. The ARDOISE architecture.

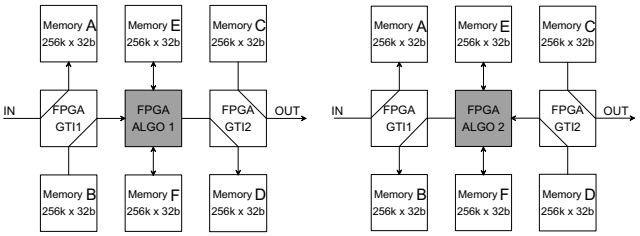
The ARDOISE architecture [1][2] is based on three (or more) identical modules (fig. 1.a). Each one (fig. 1.b) encloses an FPGA connected to two local memories for temporary storage of the local results. The three modules are interconnected by a 48-bits bus which can carry either addresses or data. Free buses on both sides are used for data input and output (for example camera and screen). A *mother module* includes all the hardware for configuration storage in a compress form, for configuration scheduling and for clock

0-7803-8554-3/04/$20.00 ©2004 IEEE.

generation. Its FPGA is used to uncompress the configuration files during the reconfiguration process and to manage the exchanges of the high level results with the host processor. The chosen host processor is a DSP SHARC 21061 by Analog Devices.

2.2. An example.

Figures 2.a and 2.b explain one of the multiple uses of ARDOISE. The two GTI modules are used as frame grabber. During computing of frame n, the frame n+1 is inputted and stored in memory A and the frame n-1 is read from memory C and outputted. The frame n is stored initially in memory B. The first configuration is loaded on the central FPGA (in grey), the first algorithm is then applied to the data previously stored in memory B (fig. 2.a). Then, the FPGA is configured again for computing the second algorithm (fig. 2.b). D and B are used respectively as input and output memories for this algorithm. For successive algorithms, the scheme is repeated. At the end of the frame duration, several algorithms have been computed on the grey FPGA and the role of memories A and B are exchanged for the next frame computation, idem for C and D memories.



(a) First step. (b) Second step.

Fig. 2. An execution scheme with dynamical reconfiguration.

The GTI1 and GTI2 modules allows unsynchronization between the central computing module, UTGV, and the video acquisition system. The different treatments are swapped in the central module. Intermediate results will be stored in the GTIs local memories with different memory models, during computing or reconfiguring. The computing module memories (E and F) are used to store local data inherent to every algorithm: temporary data storage, realization of data structures such as FIFO and various treatment parameters.

3. THE ALGORITHM.

3.1. Description.

Coding grey-scale pictures represents a huge amount of data. In order to reduce the memory space needed for storing it, one can first compute a binary representation of the image. Effective algorithms, as those used in faxes, constitute smart solutions of this problem.

However, we are going to study a much more simple algorithm. All along this study, the reader can be surprised by the disproportion between the implemented solutions and the algorithm simplicity. Nevertheless, architectural concepts are well illustrated by this simple processing. Some useful algorithms (sec. 5), which would have probably hidden these concepts, are proposed later.

Pixels whose intensity is higher than a global threshold will be white in the binary picture. The others will be black. The problem is to find an optimal threshold in order to reduce the original picture distortion. The choice of the threshold is based on the original picture histogram. The binary picture is then computed by comparing the initial picture and the threshold. The whole chain is described on figure 3.

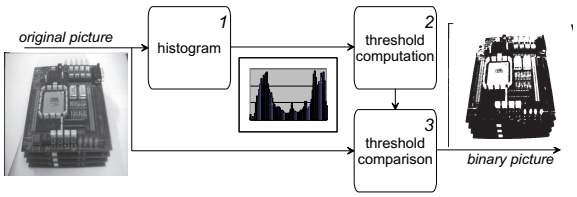


Fig. 3. Successives computing steps for the binary picture processing.

3.2. Hardware and software partitioning.

Algorithms 1 and 3 require a lot of data to be computed: they need to access to all the pixels of the original picture. What is more, these computations are really simple and easy to parallelize: we need only one memory to compute the histogram, and, as we'll show it later, a single comparator for the threshold comparison. Consequently we can take advantage of a hardware implementation of these two algorithms.

On the contrary, the algorithm 2 needn't a lot of data accesses. For example, only $2^8 = 256$ values of the histogram are needed to compute the threshold of an 8-bits coded picture. In addition, this algorithm is much more complicated than the others: the shape of the histogram affects the method for computing the threshold. Consequently, different cases have to be considered, and this makes this algorithm not suited for hardware computing.

In conclusion, for the whole chain implementation on ARDOISE, algorithms 1 and 3 are implemented on the FPGAs, while the algorithm 2 is left to the DSP.

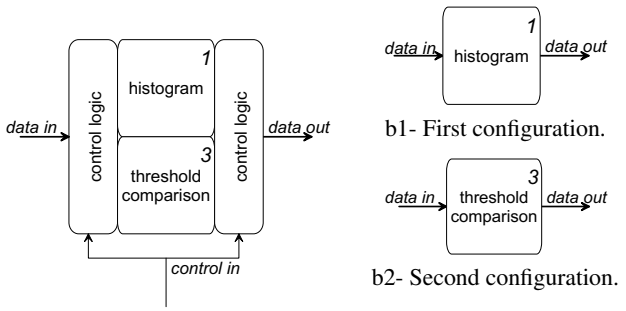
4. TAKING ADVANTAGE OF DYNAMICAL RECONFIGURATION.

4.1. Hardware saving.

Algorithms used for the calculation of the binary image are not used at the same time. The calculation of the optimal threshold needs the histogram results. Similarly, the calculation of the binary picture needs this threshold. Therefore it is necessary to wait until the end of the algorithm 1 calculation before beginning algorithm 3.

Therefore, it is not necessary for the material architecture, foreseen for the calculation of these two algorithms, to process them simultaneously. Dynamical reconfiguration allows to benefit from this situation of two manners (fig. 4):

- the surface necessary for the implementation of the algorithm 1, as that necessary for algorithm 3, is lower than the surface necessary for the implementation of these two algorithms.
- the control logic necessary for the cohabitation of these two algorithms is saved. This reduces also the surface necessary for the implementation of algorithms.



(a) Static scheme. (b) Dynamic scheme.

Fig. 4. Hardware saving using dynamical reconfiguration.

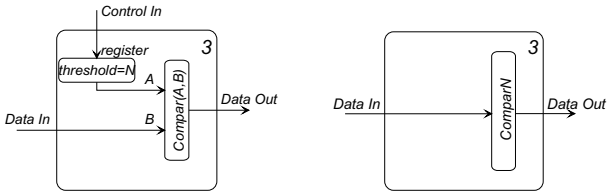
The implications of the second point deserve to be studied in detail. The control logic is not present any more in the hardware part of the architecture. It's the reconfiguration scheduler who controls which algorithm is mapped on the architecture. This presents an advantage which will be widely deepened in the following paragraph. Indeed, the reconfiguration scheduler is controlled via a DSP. Therefore it is programmed with a high-level language. This type of language brings flexibility in the use of the architecture.

4.2. Hardware optimisation.

In this part, we pay our attention on the third algorithm. This last one consists in comparing original image with a threshold. The threshold is not known in advance, so, it is necessary to adapt the algorithm to various possibilities.

Classic solution consists in foreseeing an interface in order to set the algorithm parameter (fig. 5). In this case, an 8-bits register, can latch the threshold value. Interface simply allows to update the register value.

Dynamical Reconfiguration offers another solution (fig. 5). For an 8-bits coded picture, it's possible to develop 256 FPGA configurations. Each one corresponding to a given threshold. The reconfiguration scheduler chooses the appropriate configuration on the fly. Each of 256 configurations is more specific than quoted classic configuration. Thus, configurations can be better optimized.



(a) Classic solution. (b) Dynamic solution.

a generic configuration. 1 of 256 possibles configs.

Fig. 5. Third Algorithm implementation.

What is maybe more important, is that each of 256 specific configurations is more simple than the general configuration. This reduces time necessary for the development of the material part of the algorithm. Once again, control is software managed. Control is moreover easier to implant in a DSP than in a FPGA.

The major inconvenience of this method is the fact it requires an important memory space to store configurations. Memory space needed for the storage of a configuration is given in tab. 1. The memory needs for 256 configurations can be found in tab. 2. The memory space available to store configuration data in the *mother module* of ARDOISE is only of 512kB. Therefore, this solution was not implemented on ARDOISE. When images are coded on a more important number of bits, the necessary space is even more important.

Total reconfiguration	41,0 kB
Partial reconfiguration	60,0 B

Table 1. Memory needs for storing a configuration.

A first solution consists in using partial reconfiguration. In reality, configurations are very similar. On figure 5.b, the change of the threshold only acts upon the *ComparN* module. Let us imagine we begin by reconfiguring totally the FPGA for a constant value, for example *Compar0*, of the threshold. In these conditions, only resources used for *ComparN* have to be reconfigured to process the adequate com-

parison. Table 1 gives the memory space needed to store the partial configuration of the ressources used by *ComparN*. Table 2 gives the memory needs to store the configurations needed to process all 256 possible comparisons when we use partial configuration.

Total reconfiguration	10,2 MB
Partial reconfiguration	60,9 kB

Table 2. Memory needs for storing all the configurations.

4.3. Hardware flexibility.

The results of previous section already allowed us to speak about flexible hardware. Indeed, calculation realized with the FPGA is not predictable in advance. The configuration scheduler chooses the best adapted structure just before the processing.

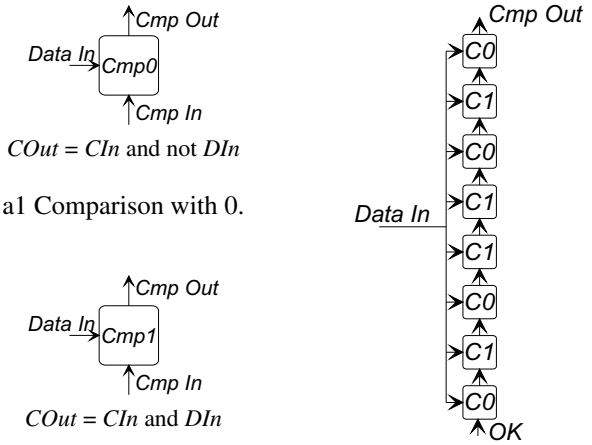
In this section, we are going nevertheless to make a step more towards the flexibility. This time, the designer will not need to envisage all the possible cases. On one hand, this study can be very long in not coarse cases. On the other hand, it is possible to supply in the high-level module enough tools so it generates alone the suited configuration. As one can see it on figure 6, the structure of an 8-bits comparator is very regular. On this figure, a square corresponds to a CLB (Configurable Logic Bloc) of the FPGA. To generate the configurations data of any 8-bits comparator , it is enough to know the configurations data of two basic elements (fig. 6.a), and to copy them, in the configuration memory of the corresponding CLBs. The configuration interface of the AT40K allows to address independently the configuration space of each CLB.

For a given value of the threshold, the DSP generates the corresponding partial configuration. This means, for each CLB of the 8-bit comparator: DSP chooses one of the two 1-bit comparator represented on figure 6.a., and, corresponds the configuration data of the chosen comparator, and the adresses of the CLB.

5. CONCLUSION AND PERSPECTIVES

The implementing of an algorithm on the dynamically reconfigurable architecture ARDOISE showed us to which point the use of this architecture can be profitable. Beyond the hardware saving, the transfer of control towards the software part of these architectures makes flexible the hardware part.

In these conditions, a FPGA is not only adaptable to a given treatment, as it is case in the classic use of these components. It can more adapt itself to the data it treats, and, software management allows to envisage relatively complex mechanisms for the algorithms choice.



(a) 1-bit comparator. (b) 8-bit comparator.

Fig. 6. Comparator generation.

Besides, the removal of control in the hardware part renders their development more simple. The development of new tools will allow us to use effectively dynamically reconfigurable architectures. Thus, one will be able to make FPGAs more powerful, and, as we have shown it, to increase their flexibility.

We plan to use dynamical reconfiguration for processing a complete chain of edge detection. This can consist in the successive processing of: a smoothing filter; the norm of the two directional derivations; and some outlines improvement algorithms.

On this not coarse example, the interest of dynamic reconfiguration is significant. The structure of this complex calculation is going to evolve according to the data it is going to handle. For example, we should be able to choose, at real-time, the best adapted edge improvement algorithm for the treated picture.

6. REFERENCES

[1] L. Kessal, D. Demigny, N. Boudouani, and R. Bourguiba, "Reconfigurable hardware for real time," in *International Conference on Image Processing*, Vancouver, september 2000, IEEE ICIP, pp. 159–173.

[2] D. Demigny, L. Kessal, R. Bourguiba, and N. Boudouani, "How to use high speed reconfigurable fpga for real time image processing," in *International Conference on Computer Architecture for Machine Perception*, Padova, september 2000, IEEE Circuit, pp. 240–246.