Real-Time Imaging 8, 277–289 (2002)
doi:10.1006/rtim.2001.0274, available online at http://www.idealibrary.com on IDEAL®

Real Time Image Rotation Using Dynamic Reconfiguration

Field programmable gate array (FPGA) components are widely used nowdays to implement various algorithms, such as digital filtering, in real time. The emergence of dynamically reconfigurable FPGAs made it possible to reduce the number of necessary resources to carry out an image-processing task (tasks chain). In this article, an image-processing application, image rotation, that exploits the FPGAs dynamic reconfiguration method is presented. This paper shows that the choice of an implementation, static or dynamic reconfiguration, depends on the nature of the application. A comparison is carried out between the dynamic and the static reconfiguration using two criteria: cost and performance. It appears that, according to the nature of the application, the dynamic reconfiguration can be less or more advantageous. In order to be able to test the validity of our approach in terms of algorithm and architecture adequacy, we realized an AT40K40-based board "ARDOISE".

© 2002 Elsevier Science Ltd. All rights reserved.

E. Bourennane, C. Milan, M. Paindavoine and S. Bouchoux

Laboratory LE21, University of Burgundy, France
Aile des sciences de l'Ingénieur, B.P. 47870 21078 Dijon Cedex, France
E-mail: ebourenn@u-bourgogne.fr

Introduction

Many applications in signal and image processing were born with the arrival of the Field programmable gate array (FPGAs) [1–4]. Since then, FPGAs have become more efficient in terms of integration density and running frequency [5, 6]. To satisfy the real-time constraint, some complex processing required the realization of multi-FPGA boards and more often heterogeneous boards based on FPGAs and DSPs. In these cases, the different algorithms are decomposed into tasks where each, according to its size, is assigned to an FPGA for its execution. We thus obtain concurrent tasks by using parallelism by functions. The goal of the designer was mainly to insure the real-time constraint. Nowdays, this technology has evolved so much that the real-time aspect is not the only objective of the designer. While some user needs have remained the same, like defects inspection, pattern recognition, filtering and image compression, FPGAs have become more powerful in calculation. For some processing, the constraint of 25 images per second is, by far, insured by the new generation of FPGAs. Consequently, these components are under-employed leading to a decrease in the efficiency of the architecture. FPGAs remain inactive during a great fraction of the image acquisition time. Once received, the image is processed at higher rate as compared to pixel acquisition frequency and then the FPGA sits idle waiting for the next image. This lost time is due to the fact that until now, it was impossible to reconfigure FPGAs to a new task during their inactivity. This is due to two main reasons: (i) FPGAs require a great deal of time to be reconfigured; (ii) to reconfigure the FPGA it was necessary to reset the whole component and consequently lose all the internal register contents.

1077-2014/02/$35.00 © 2002 Elsevier Science Ltd. All rights reserved.

Traditionally, once the configuration is optimized for a well-defined application, it is loaded into the FPGA and remains there until the end of the processing. This kind of configuration is called the static configuration. The arrival of the dynamically reconfigurable FPGAs in 1989 [7] has allowed the optimization of the FPGA's temporal efficiency [8]. Thus, it is possible to reduce the number of necessary logic gates to perform an image-processing application consisting of a succession of algorithms by time sharing between the different tasks with the available resources. When a cell (or a block of cells) has finished the execution of a task, it can be reconfigured to accomplish another task. An FPGA is said to be dynamically reconfigurable when it is possible to reconfigure only a part of the component while other parts are executing the algorithm.

Most of the works met in literature [9–12] made the hypothesis that the user of the FPGA has chosen the solution based on the dynamic reconfiguration and they only tried to help him in the optimization of his implementation. Thus, the paper [9] examined in detail the optimization of the implementation of a pipelined application by using run-time reconfigurable FPGAs. That study led to a striped FPGA architecture. This architecture permits an optimal implementation of pipelined applications, which integrates an on-chip configuration cache memory. However, the paper did not undertake a comparison between the static-configuration-based solution and the run-time-based one, and it did not consider the case where the application does not admit a pipeline. The paper [10] introduces morphing, a technique for enhancing the efficiency of reconfigurable pipelines at run-time. It describes the use of morphing in the emulation of large virtual pipelines by small physical pipelines. The Xilinx 6200 PCI is given as good and flexible platform for implementing virtual pipelines. Morphing works best when reconfigurable time is comparable to the pipeline computation time. The paper [11] treats in addition to the partitioning, the scheduling of the tasks. The advantage of the approach presented in this paper is the capability to model communication between nonadjacent on-chip configurations and multiple levels of logic. Reference [12] introduces SCORE. The SCORE permits partitioning FPGA configurations into parallel and fixed-size pages. The paged model provides a framework for device size abstraction, automatic dynamic reconfiguration and automatic performance scaling on larger devices, without recompilation. SCORE aims to automate the partitioning process.

Paper [13] highlights the diversity and multiplicity of systems and applications to which configurable computing can be used. Various systems such as SPYDER, RENCO, Firefly and BioWatch are given to show well the differences between the static- and dynamic-reconfiguration-based systems. This paper concludes by indicating that the static reconfiguration is used to improve the performances in terms of speed and resource utilization, however, the dynamic reconfiguration has the advantage of being adapted to the dynamic environment. Even though this paper has given in a clear manner the definitions and the differences between the static and dynamic reconfigurations, it did not propose any method to make a choice between the two reconfiguration modes.

Our work comes before these works that have as objective the optimization of partitioning and of scheduling for an implementation based on the dynamic reconfiguration. Indeed, we notice nowadays that many applications are still based on the static reconfiguration. So, based on this remark, it appears that it is worth to ask the following question: in which case we use the static reconfiguration and in which case we use the dynamic reconfiguration?

Often, the designer has to decide at the beginning of his project on which solution to retain: based either on the static reconfiguration or on the dynamic one. Then, he starts the optimization of the chosen implementation. With equal performances, the designer has to choose the cheapest solution. Our paper classifies the different applications based on their nature and evaluates their costs for the two cases: dynamic and static implementations. These implementations are evaluated on a given FPGA. For each case, examples are given to support our analysis. At the end of this paper, we evaluated real-time image rotation implementation on a board that we have realized using the two reconfiguration types.

It is difficult to choose an application that highlights both the advantages and limitations of the dynamic reconfiguration. The image rotation is a typical example where the dynamic reconfiguration (DR) can be exploited in a very interesting manner. Indeed, the algorithm that we have chosen requires operator reconfiguration for each image row. The choice of this algorithm has been motivated by its structure IIR and FIR filters. The achievement of such a task by a conventional FPGAs would require the use of several FPGAs that increases the number of necessary

resources. For this application, we have realized a board based on the AT40K40 FPGA (ATMEL).

This article is structured as follows. The following section gives a brief review of the structures of the static and dynamic reconfigurable FPGAs. The next section presents the analysis criteria of the two implementations, static versus dynamic reconfiguration. The subsequent section studies the dynamic reconfiguration case-by-case, depending on the nature of the application, The next section describes the image rotation algorithm that was implemented. The following section deals with the analysis of the two implementation approaches (static and dynamic) of the image rotation algorithm according to the criteria defined in the section which presents the analysis criteria. Finally, the last section gives a comparison between the results obtained by the dynamic and static approaches.

Field-Programmable Gate Arrays

Static reconfigurable FPGAs

Conventional FPGAS form the basis of many applications [2–5]. These components have allowed the association of the flexibility and the specificity. Several applications can be realized by specialized architectures by simply configuring the FPGAs each time the FPGA-based board is supplied. An FPGA is composed of configurable logical blocks (CLBs), in/out blocks (IOBs), which connect the logic cells to external signals, in/out (IO), and programmable routing network interconnecting the cells.

Dynamic reconfigurable FPGAs

Compared to traditional FPGAs, dynamically reconfigurable FPGAs offer the possibility of sharing in time the available resources in the FPGA between the different tasks of an application. This can be accomplished by using either total or partial dynamic reconfiguration. For an application and at a given time, one or several parts of the FPGA can be inactive. It is then possible to reconfigure them for other tasks by using partial reconfiguration. Some components allow a direct access to the resources of configuration by simple addressing. For example, the ATK40 series (ATMEL) allows the reconfiguration of any area of the component by modification of the SRAM configuration contents. The FPGA is seen, by the user, as a memory for which one can modify the content by putting data (8/16 bits) in the necessary addresses (24 bits) for the reconfiguration. Hence, it is possible to modify the content of a logic block independent of the others.

Criteria Analysis of the Static and the Dynamic Implementations

The analysis of the static implementation will be carried out through the evaluation of two criteria: the implementation cost and the processing performance. The cost will be designated here by the number of CLBS (N_{Stat_CLBS} = computation cells + control cells) occupied by the algorithm. The smaller is this number, the better is the implementation. Here, we do not take into account the energy consumption nor the hardware complexity of the peripherals, such as external memories, data bandwidth, and address bus. The complexity and the variety of peripherals make difficult the estimation of their costs. The processing performance criterion is given by the necessary execution time T_{Stat_exec} to accomplish the application. T_{Stat_exec} is expressed only by the product of the number of pixels N (image size) to be processed and the iteration period $T_{Stat_iteration}$ of the algorithm. In the same manner, we define the dynamic cost by N_{Dynam_CLBS} and the performance criterion by T_{Dynam_exec} With equal performances the contribution of the dynamic reconfiguration will be given by the ratio $[(N_{CLBs_Stat} - N_{CLBs_Dynam})/N_{CLBs_Stat}] \times 100\%$. For some applications using dynamic reconfiguration, it may be preferable to have less performances in order to benefit from reductions in the number of CLBS used. It is necessary, in this situation, to compare the reduction in cost to the loss in performances.

The Dynamic Reconfiguration Case-by-Case

Up to date, many investigations have been conducted in order to optimize implementation, partitioning, and scheduling tasks using the dynamic reconfiguration. However, few works have been devoted to help the user to choose between an implementation that uses the static configuration and an implementation based on the dynamic reconfiguration. Our work deals with this subject and gives an approach that allows, from the start, the user to make a choice between the two implementations. In the following, we are going to enumerate different cases in which dynamic reconfiguration can be more or less interesting.

Pipelined applications

Often, image-processing applications consist of a succession of tasks, $Task_i$ $(i = 1 \ldots M)$. This type of applications can be implemented following either a dynamic or static configuration as shown in the diagram given in Figure 1. An example of such kind of applications is given in [8]. It deals with video coding and it consists of four stages that can be pipelined: discrete wavelet transform, quantization, run-length coding and entropy coding. Here, the authors have succeeded in reducing the necessary resources (CLBs) needed to implement this encoder. They used only one FPGA, CLAY31 of National Semiconductor, instead of three FPGAs of the same type. This is achieved by using rapid run-time reconfiguration instead of static configuration. Indeed the implementation that does not employ run-time reconfiguration can present, when needed, better performances, an execution time which is three times smaller than that obtained when using runtime reconfiguration. The authors, however, preferred a reduction in the number of gates for congestion and power consumption reasons because the device was aimed for portable handheld wireless transceivers.

Discussion of the static implementation. In this case, the user selects the FPGA component that best fits his/her needs at the lowest cost. The optimum is obtained when the processing frequency of the designated FPGA corresponds exactly to the real-time constraint. An economy, in internal resources of the FPGA, can be obtained by choosing the execution time ($NT_{Stat_iteration}$) equal to the image acquisition time T. Thus, one obtains a static implementation having a maximum spatiotemporal efficiency of 100%. This means that at any time all FPGA resources are utilized (active). The cost and performance of such an implementation are given by

$$
\begin{aligned}
{}_{Stat} &= N_{Stat_CLB} \\
&= \sum_{i=1}^{M} N_{Task_i_CLB} + N_{Controle_CLB} \text{ and } Perf_{Stat} \\
&= T_{Stat_exec} = NT_{Stat_iteration} = T \qquad (1)
\end{aligned}
$$

Discussion of the dynamic implementation. In the example given above, the implementation is done by using global reconfiguration with K partitions (here $K = 2$), and each partition is composed of 2 tasks. The processing has been divided into two partitions (taskl + task2) and (task3 + task4) of equal areas (N_{CLBS}). The iteration period of each partition depends on the implemented algorithm. Although the areas of the two partitions are equal, their iteration periods can be different. The time duration T allocated to the processing is identical to that of the static case. One can easily find that the real-time constraint

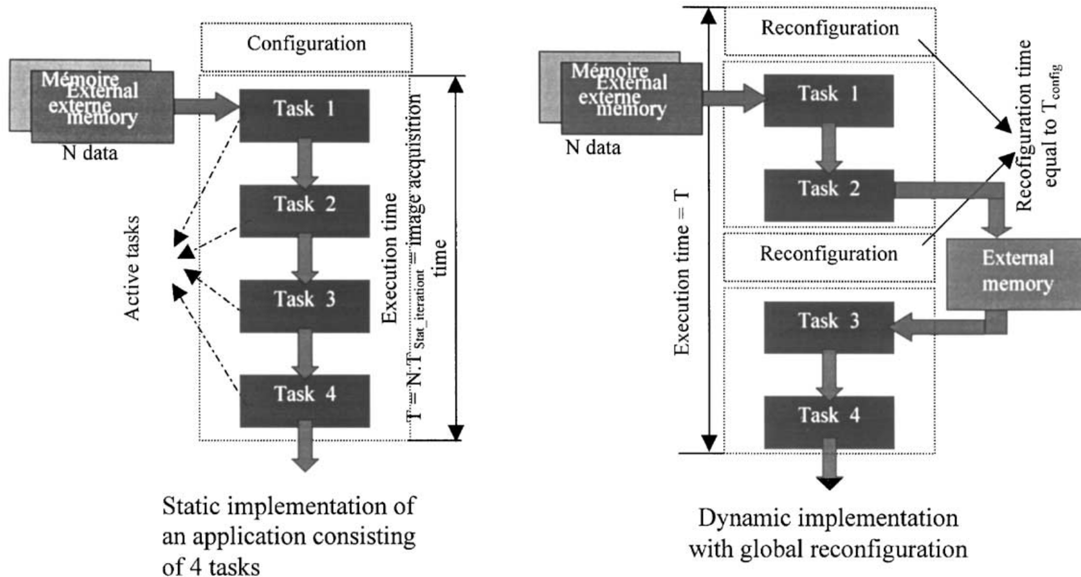


Figure 1. Static and dynamic implementations.

leads to

$$T = NT_{Stat_iteration} = N\sum_{i=1}^{K} T_{iteration_i} + KT_{config} \quad (2)$$

Since we are dealing with a global reconfiguration, the reconfiguration times are equal: $T_{config_1} = T_{config_2} = T_{config_K} = T_{config}$. We assume, without loss of generality, that the two partitions (taskl + task2, task3 + task4) run at the same frequency ($f_{iteration_1} = f_{iteration_2} = f_{iteration_K} \Rightarrow T_{iteration_1} = T_{iteration_2} = T_{iteration_K} = T_{Dynam_iteration}$). Therefore, from Eq (2), we obtain

$$NT_{Stat_iteration} = KNT_{Dynam_iteration} + KT_{config} \quad (3)$$

From where we can deduce the maximum period (the minimum frequency) with which the dynamic FPGA can run in order to reach the same performances as the static FPGA:

$$T_{Dynam_iteration} = \frac{T_{Stat_iteration}}{K} - \frac{T_{config}}{N} \quad (4)$$

If we want to obtain a gain in the area of the order K the dynamic FPGA has to be (as well as the associated external memories) at least K times faster than the static FPGA.

$$Cost_{Dynam} = N_{Dynam_CLB} = \underset{j=0}{\overset{j=(K-1)/2}{MAX}} \left(\sum_{i=2j+1}^{2j+2} N_{Task_i_CLB} + N_{Control_CLB} \right)$$

and

$$Perf_{Dynam} = T_{Dynam_exec} = T_{Stat_exec} = KNT_{Dynam_iteration} + KT_{config} = T \quad (5)$$

In this case, the question arises as to what implementation to choose?

If it is clear that with equal performances, we have obtained by using the DR a diminution of the cost (the area is reduced by a factor K); however, the answer to the question is obvious because we have to keep in mind that the dynamic FPGA has to be K times faster than the static one. Hence it would be judicious to take into account the following points before making any choice:

- costs comparison:
  - — In terms of development time.
  - — Price of the components.
  - — External resources: memories and bandwidth.
- reuse and flexibility of the board in other cases and applications
- power consumption

In summary, if the static implementation presents a spatiotemporal efficiency of 100%, it is difficult (perhaps even expensive) to replace it by a dynamic implementation.

Case where the application does not present parallelism by functions

In this case, if the nature of the application does not allow us to have the pipeline the static implementation will present inevitably a spatial efficiency lower than 1. In Figure 1, it is sufficient to leave inactive one to three tasks among the four (see Figure 2). Such cases can appear when the processing to be performed is conditioned by the nature of the data. The example of neural

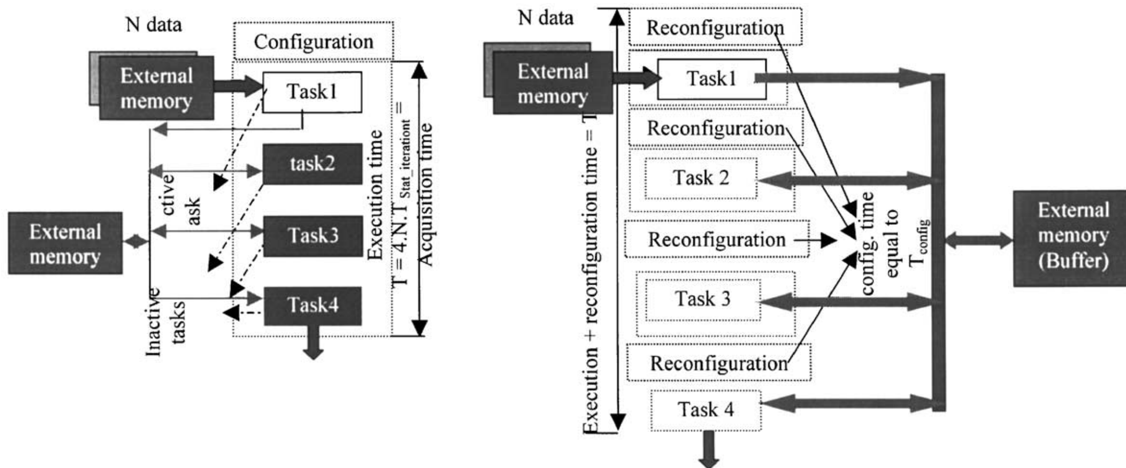


Figure 2. Static implementation with spatial efficiency lower than 100% and its corresponding dynamic implementation.

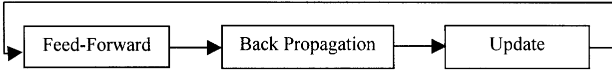


Figure 3. The three stages of the back propagation training algorithm.

networks using run-time circuit reconfiguration (RRANN: run-time neural artificial reconfigured network) is a good example to illustrate such cases [14–16]. The back propagation learning algorithm is partitioned into three sequentially executing stages: feed-forward, back propagation, and update (see Figure 3). Only one of them is loaded into the FPGA at a time. These three stages are mutually exclusive in time i.e. the computations in any stage cannot proceed until the previous stage finishes the execution and the computation of all the data present at its input. This process of reconfiguration and execution continues until the training algorithm converges. The implemented neural network is formed by 60 neurons, equally repartitioned over four layers with a total of 10 930 connections. In order to determine the gain brought by the dynamic reconfiguration, this algorithm has been implemented for both the static and run-time reconfigured systems on an FPGA CLAY31 of National Semiconductor. With equal performances, the convergence is obtained in 1.93 ms for the two implementations, the run-time implementation has necessitated 18904 CLBS while the static implementation has consumed 50764 CLBS, so the gain in area is about 62.76%. The example given in [17] shows the case where the process to be executed depends on the nature of the data, and it concerns adaptive wavelet packet applications. It is shown that using dynamically reconfigurable logic is very suitable for adaptive wavelet packet applications.

Let us analyze costs and performances of the two implementations given in Figure 2.

Discussion of the static implementation. The spatial efficiency being weak (at any time, there exist some parts that are inactive in the FPGA), the cost due to the number of CLBS used will certainly be high compared to the dynamic reconfiguration. The cost is given by Eq (1). The performance is also given by Eq (1) but needs to be explained. Assume in Figure 2, that there is only one task (among $M = 4$) that is active at a given time and that the whole processing requires the execution of M tasks. The allowed time to achieve the processing is always equal to the image acquisition time T. The iteration period of the static implementation has to satisfy the real-time constraint and leads to the following equality:

$$Perf_{Stat} = T_{Stat_exec} = MNT_{Stat_iteration} = T \tag{6}$$

The processing frequency must be higher than in Pipelined applications by a factor M to satisfy the real-time processing constraint.

Discussion of the dynamic implementation. If the tasks are exclusive in time, only one task is loaded and executed at a time, both the cost and the performance are given by

$$\begin{aligned} Cost_{Dynam} &= N_{Dynam_CLB} = MAX(N_{Task_i_CLB} \\ &\quad + N_{Control_CLB}) \quad \text{for} \quad i = 1 \ldots M \\ \text{and } Perf_{Dynam} &= T_{Dynam_exec} = T_{Stat_exec} \\ &= MNT_{Stat_iteration} = MNT_{Dynam_iteration} \\ &\quad + MT_{config} = T \end{aligned} \tag{7}$$

$$gain_{CLBs}\% = \frac{N_{Stat_CLBs} - \underset{i=1}{\overset{i=M}{MAX}} X_{Dynam_CLBs}(task_i)}{N_{Stat_CLBs}} \times 100$$

With equal performances, the dynamic reconfiguration allows us to have a gain in the number of CLBS. Eq (7) shows the necessity to have a dynamic FPGA faster than the static one.

$$T_{Dynam_iteration} = T_{Stat_iteration} - \frac{T_{config}}{N} \tag{8}$$

According to relation (8), the execution frequency of the dynamic FPGA can be reduced by increasing the amount of data (N) to be processed. However, it is necessary to raise the external memory capacity.

General case

The application may or may not present a pipeline. Furthermore, its execution time on a static FPGA (in static mode) can be less then the image acquisition time ($MNT_{Stat_iteration}T$). This case corresponds to the static implementation having a spatiotemporal efficiency lower than 1. The spatial efficiency is lower than 1 because all the tasks active or inactive are loaded into the FPGA in the beginning of the application, but only one is active at a time. Moreover, if the static FPGA used is too fast compared

with the application requirements, it will be inactive during a fraction of the image acquisition time. Thus, we obtain a temporal efficiency less than 1.

Discussion of the static implementation. The cost and the performance are given by

$$Cost_{Stat} = N_{Stat_CLB} = \sum_{i=1}^{M} N_{Task_i_CLB} + N_{Control_CLB}$$
$$Perf_{Stat} = T_{Stat_exec} = MNT_{Stat_iteration} + T_{inactivity} = T \quad (9)$$

We can notice the presence of $T_{inactivity}$ in the relation $Perf_{Stat}$. This inactivity time is taken into account in the static performance because it corresponds to the mismanagement of the available time.

Discussion of the dynamic implementation. In this case, we will show that we can obtain by using DR a gain in area and in execution frequency while keeping the same performance as in the static case. The iteration frequency of the dynamic implementation $f_{Dynam_iteration}$ can even be lower than the iteration frequency of the static implementation. We can also choose for such application a faster dynamic FPGA so that the number of necessary CLBs will be reduced.

$$gain_{CLBs}\% = \frac{N_{Stat_CLBs} - \underset{i=1}{\overset{i=M}{MAX}} X_{Dynam_CLBs}(task_i)}{N_{Stat_CLBs}} \times 100$$

$$Cost_{Dynam} = N_{Dynam_CLB} = MAX\left(N_{Task_i_CLB} + N_{Control_CLB}\right) \quad (10)$$

for $i = 1 \ldots M$

$$Perf_{Dynam} = T_{Dynam_exec} = T_{Stat_exec} = MNT_{Stat_iteration} + T_{inactivity} = MNT_{Dynam_iteration} + MT_{config} = T$$

The necessary iteration period is such that

$$T_{Dynam_iteration} = T_{Stat_iteration} + \frac{T_{inactivity}}{MN} - \frac{T_{config}}{N} \quad (11)$$

The inactivity time (idle time) of the static implementation has a positive effect on the iteration period of the dynamic implementation. If relation (12) is satisfied, then the dynamic iteration frequency can be lower than the static iteration frequency.

$$(T_{inactivity})_{Stat} \geq MT_{config} \Rightarrow T_{Dynam_iteration} \geq T_{Stat_iteration} \quad (12)$$

Thus we obtain a gain in area and in frequency by the use of the dynamic reconfiguration.

Image Rotation Algorithm

We often meet the problem of image interpolation when we want to restore an analog image from its samples. The analog form of the image allows us thereafter to carry out operations of up-sampling (super-resolution, zooming) and to shift the image by noninteger values. Image rotation is based on B-Spline interpolation used in image compression. It is very helpful for character recognition. We have chosen as application the image rotation whose principle, according to the algorithm of Unser, is based on three translations : translations following rows then columns and again following rows [18,19]. The translation is different for each row (or column). The proposed implementation hereafter can be easily adapted to up-sampling.

B-Spline functions

B-Spline functions $\beta_n(x)$ are polynomials of order n, continuous and continuously differentiable up to order $(n-1)$. A B-Spline is defined by the following convolution:

$$\beta_n(x) = \beta_{n-1}(x) * \beta_0(x) \quad (13)$$

with $\beta_0(x)$ the B-Spline of order zero:

$$\beta_0(x) = \begin{cases} 1 & \text{for } |x| < 0.5 \\ 0 & \text{elsewhere} \end{cases} \quad (14)$$

Then, we deduce the next equation

$$\beta_n(x) = \sum_{j=0}^{n+1} \frac{(-1)^j}{n!} C_{n+1}^{j} \left(x + \frac{n+1}{2} - j\right)^n \times u\left(x + \frac{n+1}{2} - j\right) \quad (15)$$

where

$$C_{n+1}^{j} = \frac{(n+1)!}{j!.(n+1-j)!} \text{ and } u(x) = \begin{cases} 1 & si\, x \geq 0 \\ 0 & si\, x < 0 \end{cases}$$

The representation of the gray levels $I(k)$ by a continuous function $I(x)$ is given by

$$I(x) = \sum_{i=1}^{N} C(i)\beta_n(x-i) \quad N \text{ is the number of pixels} \tag{16}$$

$I(x)$ is such that

$$\begin{aligned} I(x)_{x=k} &= I(k) = \sum_{i=1}^{N} C(i)\beta_n(k-i) \\ &= \text{original digital image} \end{aligned} \tag{17}$$

Knowing the sequence of pixels $I(k)$ and coefficients $\beta_n(k{-}j)$, it is possible to deduce $C(j)$ from Eqn (17) and to substitute it into (16).

In the case of a B-Spline of order 3 ($\beta_3(x)$), we obtain

$$\begin{aligned} I(k) &= \frac{1}{6}C_{k-1} + \frac{4}{6}C_k + \frac{1}{6}C_{k+1} \Rightarrow C(z) \\ &= \frac{6}{z^{-1}+4+z} I(z) \\ &= \frac{1.6}{(1+0.26z^{-1})(1+0.26z)} I(z) \end{aligned} \tag{18}$$

The image rotation

The image rotation is a geometrical problem that consists of pivoting each point of the space around an axis. As shown in Figure 4, given a point p of coordinates (x, y) in an orthonormal reference frame of origin O. Assume another point p' in the same reference frame such that the vectors Op and Op' are of equal magnitudes and form an angle.

The coordinates of p can easily be determined from those of p' by using the rotation matrix $R(\theta)$:

$$\begin{aligned} R(\theta) &= \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \\ &= \begin{pmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \\ &\times \begin{pmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{pmatrix} = ABA \end{aligned}$$

with

$$A = \begin{pmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \tag{19}$$

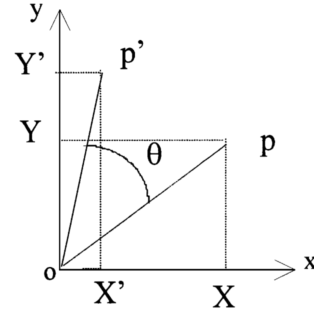


Figure 4. Representation of p and p' in (O, x, y) plane.

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = R(\theta) \begin{pmatrix} X \\ Y \end{pmatrix}$$

The rotation matrix $R(\theta)$ can be decomposed into a sequence of unidimensional translations along the directions x and y. Matrices A and B are translation matrices following x and y, respectively. The image rotation is thus obtained by a series of translations by non integer values. The shifted image by a distance Δ is obtained by

$$I(k-\Delta) = \sum_j C_j B_3(k-\Delta-j) \tag{20}$$

$$B_3(x) = \begin{cases} \frac{(2+x)^3}{6} & \text{for } -2<x<-1 \\ \frac{(4-6x^2-3x^3)}{6} & \text{for } -1<x<0 \\ \frac{(4-6x^2+3x^3)}{6} & \text{for } 0<x<1 \\ \frac{(2-x)^3}{6} & \text{for } 1<x<2 \end{cases} \tag{21}$$

and the up-sampled image by

$$I(k/m) = \sum_j C_j B_3(k/m - j) \tag{22}$$

Analysis of the Static/Dynamic Implementations

In Figure 6, the image rotation algorithm is decomposable into a succession of three translations: following the rows, then columns and again rows. These three phases cannot be fully implemented in one AT40K40 FPGA. The inherent sequential aspect of this application already gives a first temporal repartition of the different subtasks. The study presented in this paragraph enables the examination of the performance and cost of such a decomposition. Tests are done on the ARDOISE board (Architecture Reconfigurable Dynamically Oriented Image and Signal Embarcable).

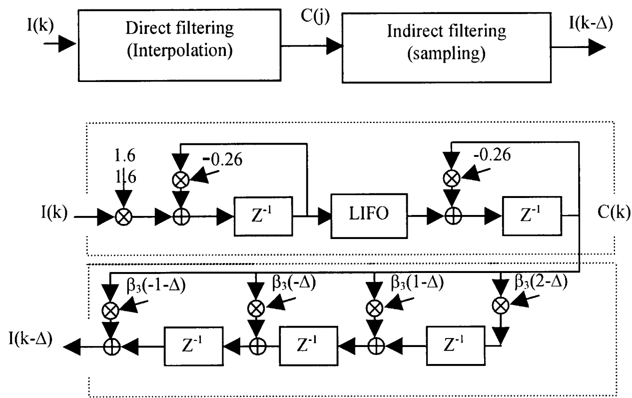


Figure 5. The structure of the filter.

Figure 7(a) gives the block diagram of the board prototype. The main task of the DSP is to load the bitstream in parallel into the FPGA.

The internal data coding and arithmetic computations of the filter (Figure 5) are performed with 13-bit resolution [20]. The results of the implementation presented here are based on the global reconfiguration. The two SRAMs function in "Ping-Pong" mode; i.e. when one is in the writing mode the other is in the reading mode and vice versa. The solution retained uses a data parallelism of order 4; four data points are processed at a time. At every memory access, two data points (2×16bits) are read. The memory access frequency is 40 MHz (twice the processing frequency). In the FPGA, two samples are processed simultaneously and the following two samples are processed with a 25 ns time delay (1/2 cycle of the processing clock period).

At the output of the FPGA, the data are read out at a frequency of 40 MHz (see Figure 8). The anticausal part of the filter is implemented in the same manner. One reconfiguration is necessary between the causal part and the anticausal part of the filter, it is possible to carry out only a partial reconfiguration (see Figure 5).

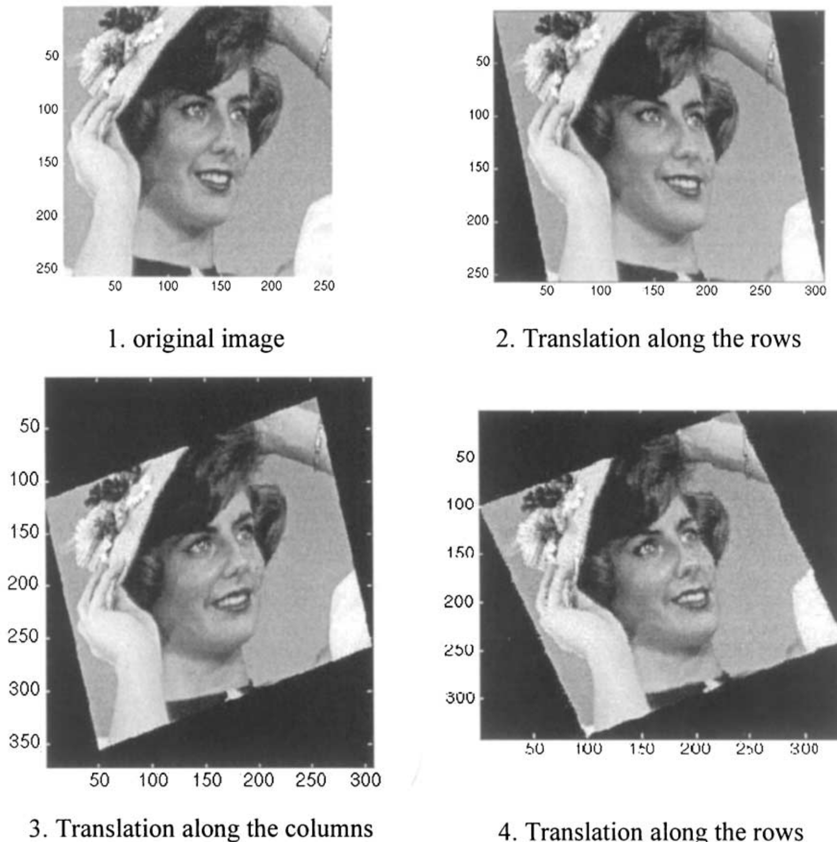
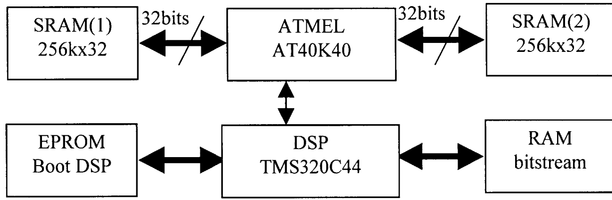


Figure 6. Image rotation with three consecutive translations.



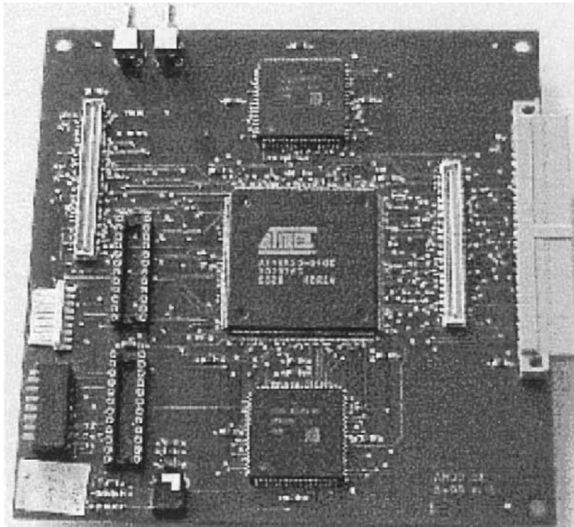


Figure 7. (a) Block diagram of the prototype board. (b) ARDOISE prototype board.

This filter is of first order with constant coefficients (see Eqn. (18) and Figure 5). The images to be processed have 8-bit resolution (256 gray levels) and are of size 256×256 pixels. The rotated image is of size 362×362 pixels. The multipliers limit the processing frequency to 20 MHz.

The nonrecursive part of the filter is loaded into the FPGA at the end of the execution of the anti-causal IIR filter (Figure 9). The four coefficients of this filter vary from one image row to another. Therefore, it is

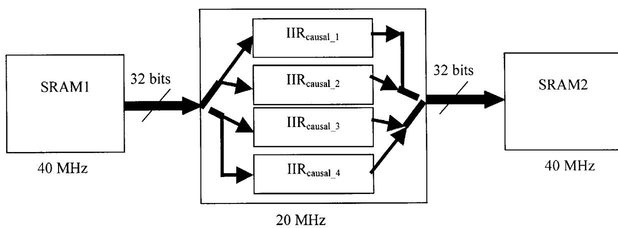


Figure 8. Parallel implementation of the causal part (idem for anticausal) of the IIR filter.

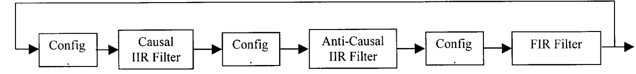


Figure 9. Temporal partitioning of image translation.

necessary to update them for each image row. The long-term objective is of course to use multipliers with propagated constants. This will allow us to perform the dynamic reconfiguration of the FIR filter from one row to the next by substituting the old multipliers by more recent optimized and simplified multipliers for the next coefficients. In this first version, we use multipliers with variable coefficients. Since it is necessary to change their values from one row to the next, the coefficients are loaded into the internal registers at the end of each image row. To load eight coefficients, four clock cycles are required at the end of each image row. The size of the FIR filter has forced us to use a data parallelism of order 2. Two FIR filters occupy 1011 CLBS.

Discussion of the static implementation. In order to carry out a comparison on the same basis, the evaluation of the cost (in CLBS) is done by implementing the image rotation algorithm in a static manner on the AT40K40-based board. This application belongs to the general case, which is studied earlier. The three tasks that compose it are concurrent (parallelism by functions). The running frequency of this component is higher than the application needs. As a result, the component remains inactive for a fraction of the image acquisition time. Moreover, we are going to see that the static implementation requires several AT40K40 FPGAs, and hence additional resources.

One translation is achieved by two IIR filters and one FIR filter. It occupies 1600 cells from a total of 2024. To accomplish an image rotation it is necessary to perform three translations requiring $3 \times 1600 = 4800$ cells. This practically takes 3 AT40K40 FPGAs. If the size of FPGAs was adapted to the image rotation needs, the static cost would be: $Cost_{Stat} = N_{Sta_CLBt} = 4800$ *CLBs*.

However, in reality we choose an FPGA with *a priori* knowledge that among the cells constituting it, there will be some that will not be used. Since we possess only one card based on AT40K40 FPGA, the experiments have been done on this unique board for the three translations. First, the image is translated along the rows, then

along the columns and finally along the rows again.

$$Cost_{Stat} = N_{Stat_CLB} = 3 \times 2024 = 6072CLBs$$
$$Perf_{Stat} = T_{Stat_exec} = NT_{Stat_iteration} + T_{inactivity} = (20 + 20)\,\text{ms} = 40\,\text{ms} = T \quad (23)$$

We can note in Eqn (23) the presence of an idle time of 20 ms. The times indicated in this section are obtained by performing tests on the board "ARDOISE" (see Figure 7(b)).

Discussion of the dynamic implementation. The dynamic implementation has been tested at the same rate (20 MHz) as the static implementation and on the same component AT40K40. Nine global reconfigurations are necessary to do the three translations (an image rotation), that is six reconfigurations for IIR filters and three reconfigurations for FIR filter. The average time delay of each global reconfiguration is approximately 0.5 ms. So the total reconfiguration time per image is $T_{total\ reconf.time/image} = 9 \times 0.5\,\text{ms} = 4.5\,\text{ms}$. The processing time of an image (by the IIR causal filter) is given by

$$T_{IIR_causal} = T_{Dynam_iteration} \times \frac{image\ size}{data\ parallelism\ ratio} \times 3\,Translations = 50\,ns \times \frac{362 \times 362}{4} \times 3 = 5\,\text{ms}$$

We obtain the same execution time for the anticausal part of the IIR filter: $T_{IIR_anticausal} = 5\,\text{ms}$. The data parallelism ratio of the FIR filter being equal to two, T_{FIR} will be twice the T_{IIR_causal}: $T_{FIR} = 10\,\text{ms}$.

To accomplish an image rotation, the whole algorithm execution time is 20 ms. The global time of the application is $T_{IIR_causal} + T_{IIR_anticausal} + T_{FIR} + T_{total\ reconf.time/image} = 24.5\,\text{ms}$.

The obtained performance is identical to that obtained for the static case. The dynamic cost is given by

$$Cost_{Dynam} = N_{CLB_Dynam} = \underset{i=1}{\overset{i=3}{MAX}} \left(N_{Task_i_CLB} + N_{Control_CLB}\right) = 2NCLB_{FIR} = 1011CLBs \quad (24)$$

$$Perf_{Dynam} = T_{Dynam_exec} + T_{total\,reconf.time/image} + T_{inactivity} = (20 + 4.5 + 15.5)\,\text{ms} = 40\,\text{ms} = T$$

The exact cost is equal to the size of the used FPGA (AT40K40) and is 2024CLBs. Task 1 (four IIR causal filters) and task 2 (four IIR anticausal filters) occupy an identical number of CLBS that is 800 CLBS. Task 3 (two FIR filters) requires 1011CLBS.

This first approach, which does not exploit the optimized multipliers, already shows the gain in area obtained by the dynamic reconfiguration 66.7%.

It is to note that the use of partial reconfiguration to mask configuration times will make only a maximum gain in performance of 11% of the image duration (the global reconfiguration time by image is 4.5 ms). Therefore, a more significant improvement can be obtained only by using optimized arithmetic operators.

Comparison Between the Implementation in Static Configuration and the Implementation in Global Dynamic Reconfiguration

Although the contribution of the dynamic reconfiguration is certain, this is only to discuss two points that seem, for us, to be advantages of the static configuration.

- We have presented the image rotation for an image of a size of 256×256 pixels. In the dynamic reconfiguration case, any increase in the processed image size (by increasing the pixel acquisition frequency) can require a total recasting of the partitioning and sequencing of the tasks. It may also be necessary to increase the resources (perhaps changing the FPGA) to insure the real-time constraint. In our example, changing the image to 512×512 multiplies the dynamic execution time by four and thereby causes the violation of the real-time constraint.
  The static implementation does not need any changes when increasing the processed image size, the only limits in this case will be the external memory (buffers) sizes.
- The increase of the input pixel frequency is supported by the static configuration, limited by the maximum processing frequency, without any modification of the architecture. In the case of the DR, the input pixel frequency does not have to exceed the processing frequency divided by the partitioning rate K (number of partitions). It will be even lower if we take into account the reconfiguration time.

If we leave aside these two points, the DR is by far the one which brings higher flexibility and requires less area

and therefore avoids crowding (more reliability and less energy consumption). A dynamically reconfigurable FPGA is more flexible than a static-configuration-based FPGA due to the fact that it is possible to realize a succession of image-processing algorithms on one dynamically reconfigurable FPGA while it would be necessary to have, for the same task, several static configurable FPGAs. Any modification of the number and the nature of algorithms to perform can be easily achieved when using a dynamically reconfigurable FPGA. Large problems are broken down and partitioned temporally into stages, each of which tits onto an array. As an example, one can read the paper [21] which describes a solution for automatic target recognition based on dynamically reconfigurable FPGA.

Conclusion and Prospects

This first real-time image rotation implementation by using dynamic reconfiguration has allowed us already to highlight some problems specific to dynamic reconfiguration. Through this article we have focused on the impact of the application nature on the cost and the performance of the implementation. We have clearly shown that the implementation of a pipelined application (with a spatiotemporal efficiency equal to 100%) using the DR cannot be justified by the cost criterion alone. Indeed, for such applications, except for the crowding problem, the dynamic implementation can be more expensive than the static implementation. In general, a gain in number of CLBs of a factor k is obtained by the use of the DR, but this is at the price of a processing frequency that increases by the same factor. An example of a concrete realization is given to illustrate our theoretical developments. We have implemented real-time image rotation in static and dynamic manners. The dynamic implementation shows an improvement in the used area of 67% compared to the static case.

Currently, we are dealing with the image rotation implementation by using local reconfiguration. Indeed, as it is stated at the end of Analysis of the static/dynamic implementations, a significant improvement in the execution time (when using DR) can only come by the use of optimized multipliers.

References

1. Carter, W.S., Duong, K., Freeman, R.H., Hsieh, H.-C., Ja, J.Y., Mahoney, J.E., Ngo, L.T. & Sze, S.L. (1986) A user programmable reconfigurable logic array. In: *IEEE Proceedings 1986 Custom Integrated Circuits Conference*, IEEE, May, pp. 233–235.
2. Dick, C. & Harris, F. (1998) Virtua1 signal processors. *Microprocessors and Microsystems* **22**: l35–148.
3. Waldemark, J., Millberg, M., Lindblad, T. & Waldemark, K. (2000) Image analysis for airborne reconnaissance and missile applications. *Pattern Recognition Letters* **21**: 239–251.
4. Park, S.W., Seo, Y. & Hong, K.S. (2000) Real-time camera calibration for virtue studio. *Real-Time Imaging* **6**: 433–448.
5. Caffrey, M., Szymanski, J.J. & Begtrup, A. (1999) High performance signal and image processing for remote sensing using reconfigurable computers, *SPIE*, 19–21 Juillet 1999, Denver, CO, pp. 142–149.
6. Atmel Co., AT4OK FPGAs with FreeRAM, htttp://www.atmel.com/
7. Gray, J.P. & Kean, T.A. (1989) Configuration hardware: new paradigm for computation. In: Proceedings *Decennial caltech Conference on VLSI*, Pasadena, CA, March 1989, pp. 277–293.
8. Villasenor, J., Jones, C. & Schoner, B. (1995) Video communications using rapidly reconfigurable hardware. *IEEE Transactions on Circuits and Systems for Video Technology* **5**: 565–567.
9. Schmit, H. (1997) Incremental reconfiguration for pipelined applications. *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 47–55.
10. Luk, W., Shirazi, N., Guo, S.R. & Cheung, P.Y.K. (1997) Pipeline morphing and virtual pipelines. In: Luk, W. & Cheung, P.Y.K. (eds), *Field-Programmable Logic and Applications*, Lecture Notes in Computer Science, Vol. 1304, London, England: Springer, pp. 111–120. http://www.doc.ic.ac.uk/̃wl/papers/
11. Chang, D. & Marek-Sadowska, M. (1999) Partitioning sequential circuits on dynamically recontigurable FPGAS. *IEEE Transactions on Computers* **48**: 565–578.
12. Caspi, E., Chu, M., Huang, R., Yeh, J., Wawrzynek, J. & DeHon, A. (2000) Stream computations organized for reconfigurable execution (SCORE): introduction and tutorial. In: *Conference on Field Programmable Logic and Applications*, FPL '2000, August 28–30, 2000, 10pp. http://brass.cs.berkeley.edu/documents/score tutorial.html.
13. Sanchez, E., Sipper, M., Haenni, J.O., Beuchat, J.L. & Perez-Uribe, A. (1999) Static and dynamic configurable systèmes, *IEEE Transactions on Computers* **48**: 556–564.
14. Wirthlin, M.J. & Hutchings, B.L. (1995) Improving functional density using run-time circuit reconfiguration, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **6**: 247–256.
15. Wirthlin, M.J. & Hutchings, B.L. (1997) Improving functional density through run-time constant Propagationin, *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 86–92.
16. Hutchings, B.L. & Wirthlin, M.J. (1995) Implementation approaches for reconfigurable logic applications, In: 5th *International Workshop on Field Programmable Logic and Applications*, Oxford, England; August 1995, pp. 419–428.
17. Graves, C., & Gloster, C. (1998) Use of dynamically reconfigurable logic in adaptive wavelet packet applica-

tions. In: *Proceedings of the 5th Canadian Workshop on Field-Programmable Devices,* June 1998.

18. Unser, M., Aldroubi, A. & Eden, M. (1993) B-Spline signal processing: part I —theory. *IEEE Transaction on Signal Processing* **41**: 821–832.
19. Unser, M., Thevenaz, P. & Yaroslavsky, L. (1995) Convolution-based interpolation for fast, high-quality rotation images. *IEEE Transaction on Signal Processing* **41**: 834–848.
20. Berthaud, C., Bourennane, E., Paindavoine M. & Milan, C. (1998) Implementation of a real time image rotation using B-spline interpolation on FPGA's board. *SPIE*, San Diego, pp 512–519.
21. Villasenor, J. et al. (1996) Configurable computing solutions for automatic target recognition. *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa, CA, April 1996, pp. 70–79.