Optimization of Motion Estimator for Run-Time-Reconfiguration Implementation.

Camel. Tanougast, Yves. Berviller, Serge.Weber.

Laboratoire d'Instrumentation Electronique de Nancy - Université Henri Poincaré Nancy I
Faculté des Sciences, BP 239
F-54506 Vandoeuvre-lès-Nancy cedex, France
{tanougast, yves.berviller, serge.weber}@lien.u-nancy.fr

Abstract. In this paper, we present a method to estimate the number of reconfiguration steps that a time-constrained algorithm can accommodate. This analysis demonstrates how one would attack the problem of partitioning a particular algorithm into pieces to for run time reconfiguration execution on a Atmel 40K FPGA. Our method consist in evaluating algorithm operators execution time from data flow graph. So, we deduce the reconfiguration number and the algorithm partitioning for RTR implementation. The algorithm used in this work, is a qualitative motion estimator in the Log-Polar plane.

1. Introduction.

The availability of FPGAs which supply fast and partial reconfiguration possibilities, provides a way to dynamically reconfigurable architectures [1]. This new approach enables the successive execution of an algorithms sequence on the same device [2].
This article propose an evaluation method for the determination of the number of successive reconfigurations which can be made for a given algorithm. This evaluation is obtained from the data flow graph in order to optimize its implementation on a run time reconfigurable architecture. This architecture uses Atmel's AT40k FPGAs, which have short configuration times. The evaluation of this number gives us the partitioning of the data flow graph. The aim of this paper is the optimization of hardware resources while satisfying the real time processing constraint. The performances like processing time and resources usage rate of the FPGA are described.
The algorithm is an apparent motion estimator in a Log-Polar images sequence, which estimates the normal optical flow.
Firstly we describe the algorithm. Secondly, we present the method for the determination of the step number for a Run-Time-Reconfiguration (RTR) implementation. Thirdly we give the results compared with a static implementation. Finally we conclude on the contribution of this approach.

2. Qualitative motion estimation in the Log-Polar space.

The Log-Polar images are obtained by remapping the Cartesian coordinate images in a Complex Logarithm Mapping [3]. The advantage of this transformation is that the radial and axial motion in the original space becomes mainly horizontal in the new space. Our solution estimates the horizontal displacements of moving objects edges. The method uses OFC (1) (optical flow constraint) of moving points in image sequence.

$$\vec{V} \cdot \vec{grad}\, I = -\frac{\partial I}{\partial t}. \qquad (1)$$

$\vec{V}$ is the apparent velocity vector of an image point and I the intensity of this point.
From this Optical Flow Constraint we estimate the normal optical flow by dividing the temporal derivative by the spatial gradient:

$$V_n = -\frac{\frac{\partial I}{\partial t}}{\frac{\partial I}{\partial x}}. \qquad (2)$$

V_n is an estimate of normal optical flow in Log-Polar images.
Before this *computation*, two pre-processing are necessary. The first processing is a gaussian filtering in order to guarantee the existence of the spatial derivative of image intensity I(x, y). The second is a time averaging filter to reduce the noise.
Our apparent motion estimator algorithm in Log-Polar plane, is composed of gaussian and averaging filters, followed by temporal and spatial derivatives and arithmetic divider. The datapath of this algorithm is given on figure 1.

3. Determination of the possible number of steps for RTR implementation.

3.1. Evaluation of the possible number of steps.

The images are acquired at a rate of 25 images per second, this leaves us 40 ms to process the entire image. To satisfy the real time constraint we need to process at a faster rate than that of pixels acquisition. The algorithms are partitioned in N steps corresponding to N execution-reconfiguration pairs. The working frequency of each step needs to verify the following inequality :

$$n^2 \times \sum_{j=1}^{N} t\,e_j \leq Ti - \sum_{j=1}^{N} Trec_j\,. \tag{3}$$

Where **n²** is the number of pixels in the image, **N** the number of reconfiguration, **Ti** is the duration of an image (40 ms), $t\,e_j$ is the elementary processing time of a pixel in the jth steps and $Trec_j$ is the reconfiguration time of the jth steps.

The objective is to make an implementation which requires the minimal logical resources and satisfies the real time constraint.

From equation (3) we obtain the minimal number of steps that we can surely implement :

$$N \geq N_{\min} = \frac{T_i}{n^2 \times K \times t\,o_{\max} + k_{rec} \times C_{\max}}\,. \tag{4}$$

$t\,o_{\max}$ is the maximum execution time of an operator of the data flow graph (without routing), K is a coefficient which take into account the routing delay between operators, k_{rec} is a proportionality constant between the configuration time and the number of used logic cells and $C_{\max}$ is the total available logic cells. This evaluation is obtained with the maximal configuration time and the execution time of the slowest operator of each step.

Our method is based on the analysis of the data flow graph of the algorithm in order to deduce the value of these parameters. The determination of $N_{\min}$ gives us the number of partitions of the data flow graph which corresponds to the number of reconfiguration steps.

3.2. Modelling and parameters determination.

AT40K’s technology enables partial reconfiguration. Each configuration time depends on the quantity of logic cells used for each step [4]. We evaluate the configuration time of the jth step by :

$$Trec_j = k_{rec} \times C_j\,. \tag{5}$$

Where C_j is the number of Cells of the jth step. In our case, AT40K20’s capacity of 819 Cells leads to a total reconfiguration time lower than 0.6 ms at 33 MHz with 8 bits of configuration data [5]. We obtain for k_{rec} a value of 733 ns/ cell.

The maximum execution time of an operator depends on the speed grade of the device and the data size to process (number of bits). The following equation gives this time for a cascaded operator :

$$t\,o_{\max} = Dj_{\max} \times (Tc + Tr) + Tsetup\,. \tag{6}$$

Where $Dj_{\max}$ is the maximum data size to process, Tc is the logical function path delay, Tr is propagation delay between logical function and $Tsetup$ is setup time. We evaluate these values to Tc = 1.7 ns; Tr = 0.17 ns and $Tsetup$ = 1.5 ns [5].

The maximum working frequency depends on the slowest operator and the routing delays between operators. We determined experimentally that K is constant for a given occupation rate. This coefficient has a value of 1.5 in our application.

The study of the Cell's structure enables the evaluation of the cell usage for each operator. An n bits adder or substractor, latched or not, require n cells. The same cells number applies for n bits multiplexer or register. This allows the evaluation of logical resources needed for each step of the application from its data flow graph.

4. Results.

From the data flow graph (see figure 1), we obtain the size and type of the different operators used (adder, multiplier, multiplexer...). So, in accordance with the technology used, we deduce the slowest operator execution time. With AT40K, adders are the slowest operators of our datapath if we consider identical size operators (number of bits). In our application, the slowest operator is an 15 bits latched adder. Then, the equation (6), give us a value of $to_{\max}$ of 29.55 ns.

From the equation (4), and the parameters determination, we estimate the minimal number of reconfiguration-execution (steps) $N_{\min}$ = 3.27 for our implementation. This result is obtained with a image size of 512 by 512 pixels.

We deduce the data from the following table for a RTR optimized implementation with constant resources usage rate.

Total estimated number of Cells	**Mean Cells / step number**	**Reconfiguration time / step (ms)**	$te_{\max}$ **(ns)**
690	212	0.16	44.3

The value $N_{\min}$ is calculated by considering that each step require a full device configuration and is executed with a slowest working frequency. In fact, after implementation we obtain reconfiguration and execution time lesser than or equal to evaluated time. That is why four reconfiguration-execution are possible instead of a theoretical value of 3.27.

The partitioning of the data flow graph in four step is made in the following way :

_ first step	: gaussian filter
_ second step	: averaging filter and temporal and spatial derivative
_ third step	: first half of divider
_ fourth step	: second half divider.

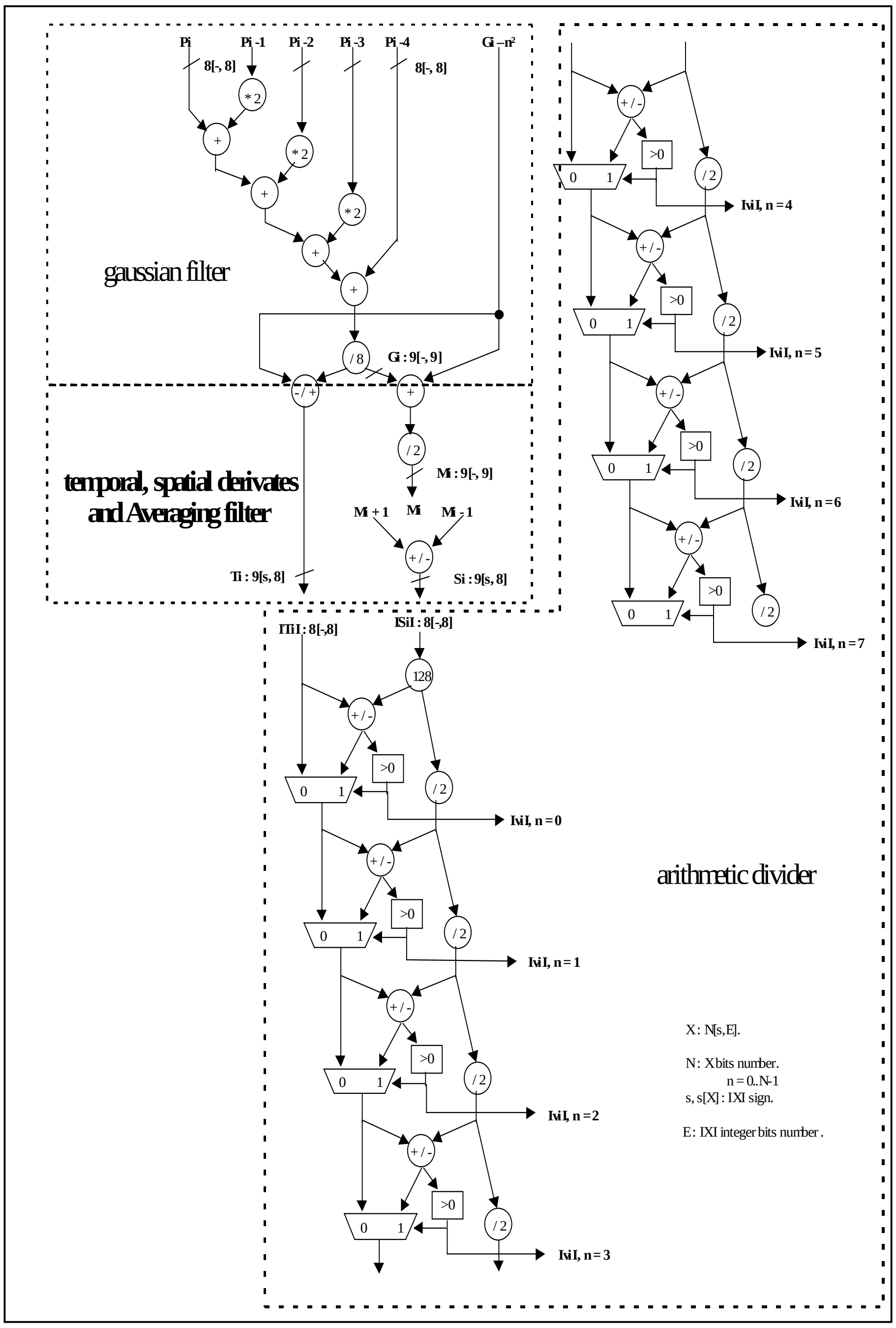

Fig. 1. Data flow graph of the motion estimator.

The divider has been split in two parts in order to homogenize the number of resources for each step. The following table shows results obtained with our implementation.

Operators	**Number of Cells**	**Reconfiguration time / step (ms)**	te_j **(ns)**
Gaussian Filter	106	0.08	27.1
average and Derivatives	103	0.08	26.5
Divider 1	354	0.26	38.7
Divider 2	336	0.25	37.8

We notice that dynamic execution with four steps can be achieved in real time. This is in concordance with our estimation. Indeed, we verify that maximal execution time (38.7 ns) is lesser than the evaluated time (44.3 ns). Moreover, we obtain a global reconfiguration time of 0.67 ms. This value is very inferior to N_{min} multipled by the full device configuration time (1.96 ms).
However, an implementation by partitioning in five steps leads to a critical time very harsh for real time operation. Indeed, in our case we have still 5.22 ms of processing time for a supplementary step. If we consider a configuration time of 0.26 ms (Same number of Cells as for the divider), we obtain a value te_j lower than 19 ns. This is incompatible with our application.
The maximal number of Cells by step allows to determine the functional density gain factor obtained by the RTR implementation [6], [7], [8]. In our example, the gain factor in term of functional capacity is approximately *2.*

5. Conclusion and future work.

We have proposed a method to evaluate the minimum number of reconfiguration-execution (N_{min}). This value depends on resources usage rate (K) for a given algorithm.
From the analysis of the data flow graph, we deduce resources requirement and speed of the various operators. This leads to the determination of total processing time, from which we deduce the optimized partitioning of the data flow graph for RTR implementation.

We illustrate our method with an apparent motion estimation algorithm on log-polar images. The results obtained are in accordance with our estimation. The differences between our estimation and experimental results are mainly due to the variations of K (which depends on routing and actual resource occupation rate). The performances obtained are compatible with the requirements of real time processing.
A partitioning which does not rely on the algorithm's functions, enables an implementation very homogeneous in terms of resource used by each step. This would allow to enhance the functional capacity.

References.

1. D. Demigny, M. Paindavoine, S. Weber : Architecture Reconfigurable Dynamiquement pour le Traitement Temps Réel des Images. Revue technique et Sciences de l'information, Numéro Spécial programmation des Architectures Reconfigurables. (1998).
2. H. Guermoud, Y. Berviller, E. Tisserand, S. Weber : Architecture à base de FPGA reconfigurable dynamiquement dédiée au traitement d'image sur flot de données. 16° colloque GRETSI. (1997).
3. M. Tistarelli, G. Sandini : On the advantage of polar and log-polar mapping for direct estimation of time to impact from optical flow. IEEE Transactions on PAMI, vol 15. (1993). 401-410.
4. ATMEL IDS AT40K User' guide.
5. Atmel. *AT40K FPGA*. Data Sheet.
6. M. J. Wirthlin, B.L. Hutchings : Improving functional density through run-time constant propagation. FCCM97 (1997).
7. H. Guermoud : Architecture reconfigurable dynamiquement dédiées aux traitements en temps réel des signaux vidéo. Thèse de l'Université Henri Poincaré. Nancy 1. (1997).
8. J.G. Eldrerge, B.L. Hutchings : Density enhancement of neural network using FPGAs and run-time reconfiguration . FCCM94 (1994).