

An Asynchronous 16*16 Pixel Array-Processor for Morphological Filtering of Greyscale Images

F. Robin, M. Renaudin*, G. Privat

France Telecom, CNET-Grenoble, * Telecom Bretagne
BP 98, 38243 Meylan Cedex, France
e-mail : robin@cns.cnet.fr

Abstract: We present a fine-grain asynchronous 16*16 VLSI array processor. It demonstrates how asynchronism can be exploited both at functional and architectural levels. Our design flow is based on a standard cells approach that combines Differential Cascode Voltage Switch Logic blocks and standard CMOS gates. The chip has been fabricated using the CNET/SGS-Thomson 0.5 μm CMOS triple metal layer technology. It includes 800 000 transistors in an area of 8*9 mm^2 . It allows real-time iterative morphological filtering of greyscale 256*256 pixels images at a 30 Hz frame rate using a single chip.

1. Introduction

1.1. Morphological image filters

Morphological filters are becoming popular in the image processing field, especially for segmentation purposes [Sal92]. They are useful for simplifying an image while still preserving the contour information. They are based on the greyscale erosion and dilation operators. The application of iterated geodesic dilations of size 1 after an erosion of size n , the reference signal r_i being the original image, defines an opening by reconstruction, according to the following equation :

$$x_i(n) = \text{Min} (\text{Max} \{ x_{i-k(n-1)} \}, r_i), \text{ with } x_i(0) = \text{Min} \{ r_{i+k} \}, \quad (\text{Eq. 1})$$

where the index i is the coordinate vector of pixel x and N_n the n -th order neighborhood. This operator is applied iteratively until global convergence is reached.

1.2. Functional asynchronism

The cellular iterative model aims at performing global processing tasks through the propagation of local dependencies. The canonical and parallel way to compute these iterative-convergent operators is through a globally synchronous updating mode : all variables are updated once, from the values calculated at the previous step, before another iteration step is initiated. A recursive updating mode, where the most recent value computed for a variable is used within the same iteration step by the next variables during the image scanning, is known to improve convergence speed, but is inherently sequential and directionally asymmetric.

In between, and retaining advantages from both, stand the asynchronous updating modes that have been studied for relaxation algorithms [Bau78]. The idea is to allow at each iteration step the updating of a random subset of state variables from "delayed" values of other variables dating back to past iterations rather than the immediately preceding one. Thus it corresponds to a less constrained updating order. In fact, a local calculation can be completed for each variable, whatever the iteration numbers of the adjacent processes are. This concept defines a functional asynchronism, at the level of algorithmic dependencies. We have studied this scheme for morphological filtering [Rob95]. It can be applied by a transformation of the global iterative algorithm into local pixel-based iterative processes and then by relaxing some indexing constraints. Instead of using the same iteration index n for all pixel processes, a local iteration step n_i can be introduced, leading to (Eq. 2), where each process has its own evolution speed.

$$x_i(n_i) = \text{Min} (\text{Max} \{ x_{i-k}(n_{i-k}), k \in N_1 \}, r_i) \quad (\text{Eq. 2})$$

While being a fully parallel approach, this updating scheme takes advantage of the recursive scheme properties because when a value is updated, it can be used by its neighboring processes as soon as they start a new calculation. Simulations have shown that such an asynchronous mode can almost double the convergence speed for the opening by reconstruction, compared to the globally synchronous mode [Rob95].

2. A functionally asynchronous cellular architecture

We have designed a fully parallel asynchronous processor array to implement the functional model described in the previous section. Each processing element (PE) updates its private pixel value in an unsynchronized manner with respect to its neighbors.

2.1 Processor-array architecture

The processor array is organized as a two dimensional square mesh-connected processor array, in which each processor communicates with its four nearest neighbors (Fig. 1). The processor array communicates with the environment using three serial links per line of PEs for pixel I/O, and an additional twelve-bit input for program instructions. Pipeline registers (PR) are added to ensure that the PEs can compute while the next image to process is entered and the previously calculated image is output. The computation starts when the Request line is activated, the Acknowledge line is driven active when the computation is completed.

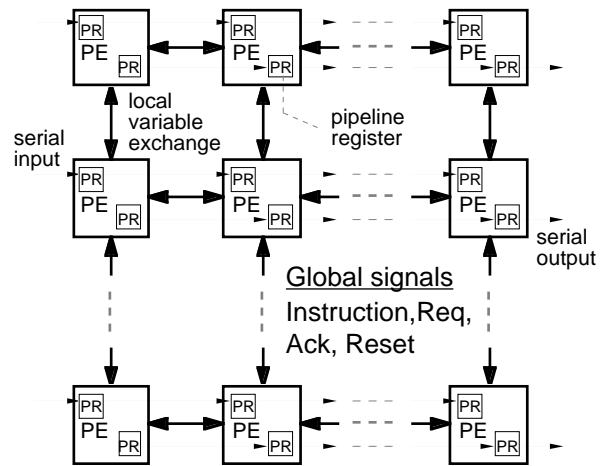


Fig.1:Architecture of the processor array core.

2.2. Programmability of the processing elements

The processing element is configured by an instruction word (Fig. 2), so that different morphological operators can be computed by the array. A bit indicates whether a Minimum or Maximum operation between the neighboring values is to be computed, which corresponds respectively to an erosion and a dilation. Another bit specifies if a geodesic operator is chosen (e.g. Eq. 2). A 5-bit field specifies the actual neighborhood. Another 5-bit field gives the number of times the operator should be locally iterated. The instruction word is thus equivalent to a small program, containing loops and conditional executions, which corresponds to a SPMD model since no synchronization between processors is used during the whole execution of the instruction. The opening by reconstruction can then be programmed using two instructions, corresponding to the erosion and the iterated geodesic dilation.

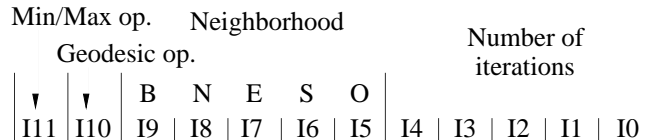


Fig. 2 : Instruction format.

2.3. Architectural level features

Implementing such an array with unsynchronized asynchronous processors brings up interesting features. First of all, program execution is performed on a mean time basis, rather than a worst time basis because the PEs do not have to wait for the slowest among them at each iteration, as would be the case with a synchronous update mode. This full parallelism combined with an improved convergence, due to the functional asynchronism, leads to higher performance when compared to a synchronous implementation.

Secondly, power efficiency is increased in three ways. Each PE can be powered down automatically at no extra cost, as soon as needed. There isn't any power peak at clock transitions, the electrical activity being spread along the time, and no power consumption is devoted to drive clock signals in the PEs. Moreover, due to the robustness of the asynchronous implementation, the supply voltage can be decreased while preserving the functional correctness. Therefore, depending on the application, the computation power may be tuned by modifying the supply voltage, leading to a minimal power consumption.

In the third place, we obtain a perfectly local implementation, that dispenses with global clocks and uses purely self-timed building blocks, that ensures modularity and scalability.

3. Circuit design

3.1. Implementation of functional asynchronism

As explained above, functional asynchronism does not require any synchronization between computations performed by different PEs. Hence, it has to be handled at the processor array level by a specific inter-PE exchange mechanism. In this scheme, each processing element should be able to download the most recent state values from neighboring PEs at any time, regardless of their mutual phase-shift and without prior synchronization signal. At the hardware level, this corresponds to enabling a PE to sample the neighbors' state variables as soon as it is needed. The adopted communication scheme waives classical request-acknowledge sequencing rules, that are in fact not strictly necessary. A direct sampling of values from the output registers of neighboring processors is enforced unconditionally. We designed a solution based on QFlop circuits, i.e. latches that are able to asynchronously sample an input signal on request of a local clock [Ros88].

3.2. Data path of a processing element

We have chosen to use precharged function blocks implemented with the so called "Differential Cascode Voltage Switch Logic" (DCVSL) to design data paths [Ren94]. The processing element must be able to store three 8-bit pixels, one for the current iteration value that is visible from the neighbors, one for a temporary value within an iteration step, and one for the reference pixel. The data path includes the corresponding registers (respectively L, B, and I), as well as serial/parallel registers for the data to be input/output independently from the computation (E, R, and S) (cf Fig. 3). A multiplexer can select a value among the reference pixel and the four neighbors' values. The operative part must compute a minimum or maximum between the temporary value stored in B and the selected neighboring value. As the temporary value B must be initialized with the input pixel value stored in E, a multiplexer is used to select the output of register E or the output of the operative part. In fact, Figure 3 shows that B is not stored in a register but in the multiplexer itself, which is designed with a latched DCVSL logic. As the ALU is also implemented with LDCVSL, it makes up a self-timed ring with only two stages [Ren96]. The ALU is made of a carry-ripple adder and multiplexers that can take advantage of dynamic critical paths and computation time dispersions [Ren94].

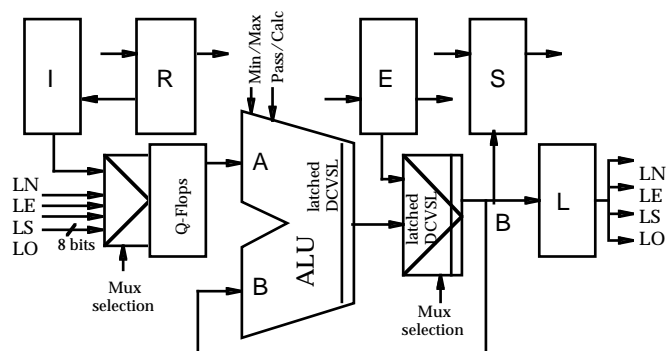


Fig. 3 : PE data path.

3.3. Asynchronous control

The control scheme mainly consists in waiting for an external request, copying the last result B into the parallel/serial output register S, copying the serial/parallel registers' input values (E and R) into the second-level registers (B and I), and then start the iterative

computation. This computation contains two nested loops: the outer loop counts the number of iterations, the inner loop copies the previous iteration result into the current iteration register L, and sequentially combines the neighboring values with the temporary value B.

A modular approach has been used to implement the asynchronous control. Each control module has been specified as an asynchronous automaton with a signal transition graph (STG) [Hau95]. Our synthesis procedure has been inspired by the technique proposed in [Chu94]. The boolean functions are directly mapped onto a set of standard CMOS gates.

3.4. Test results

The complexity of the processor array reaches 800 000 transistors and a global area of $8 \times 9 \text{ mm}^2$, in a $0.5 \mu\text{m}$ CMOS triple metal layer technology (Fig. 4). It has been tested and is fully functional. The peak power consumption is about 1 W at 3.3 V. The maximal execution time of an instruction performing a single iteration step, with the complete neighborhood and the dual geodesic operation, is 250 ns. It allows real time processing of 256×256 pixels images at 30 Hz. The chip is still functional at 1.8 V.

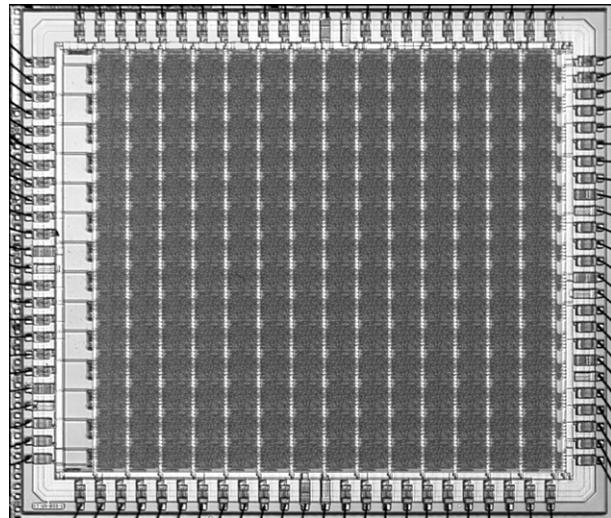


Fig. 4 : Chip microphotograph.

4. Conclusion and perspectives

We demonstrated on an image processing application that asynchronous circuits widen the spectrum of the available solutions when jointly designing algorithms and architectures. We exhibited the possibility of applying functional asynchronism to the morphological filtering problem and found an elegant hardware solution based on a structural asynchronism. It strongly illustrates how the asynchronous concept can be used at different levels.

The hardware overhead for asynchronism is certainly non-negligible, but has to be weighted against the benefits spelled out before: higher convergence speed, modularity, scalability and robustness of the structure. We believe that the asynchronous implementation style will make the integration of very complex parallel systems possible, while keeping the design effort acceptable.

References

- [Bau78] G. Baudet, "Asynchronous iterative methods for multiprocessors", *Journal of the Association for Computing Machinery*, Vol. 25, No. 2, April 1978, pp. 226-244.
- [Chu94] T.A. Chu, "Synthesis of hazard-free control circuits from asynchronous finite state machines specifications", *Journal of VLSI Signal Processing*, Vol. 7, 1994, pp. 61-84.
- [Hau95] S. Hauck, "Asynchronous design methodologies : an overview", *Proceedings of the IEEE*, Vol. 83, No. 1, January 1995, pp. 69-93.
- [Ren94] M. Renaudin, B. El Hassan, "The design of fast asynchronous adder structures and their implementation using DCVS logic", *Proceedings of the ISCAS*, London, 1994.
- [Ren96] M. Renaudin, B. El Hassan, A. Guyot, "A new asynchronous pipeline scheme: application to the design of a self-timed ring divider", to appear in *IEEE Journal of Solid-State Circuits*, special issue on 1995 ESSCIRC, June 1996.
- [Rob95] F. Robin, G. Privat, M. Renaudin, "Asynchronous relaxation of morphological operators: a joint architecture-algorithm perspective", *Proceedings of the International Workshop on Parallel Image Analysis*, Lyon, France, December 1995.
- [Ros88] F.U. Rosenberger, C.E. Molnar, T.J. Chaney, T.P. Fang, "Q-modules : internally clocked delay-insensitive modules", *IEEE Transactions on Computers*, Vol. 37, No. 9, September 1988.
- [Sal92] P. Salembier, J. Serra, "Morphological multiscale image segmentation", *Proceedings SPIE on Visual Communications and Image Processing*, Vol. 1818, 1992, pp. 620-631.