

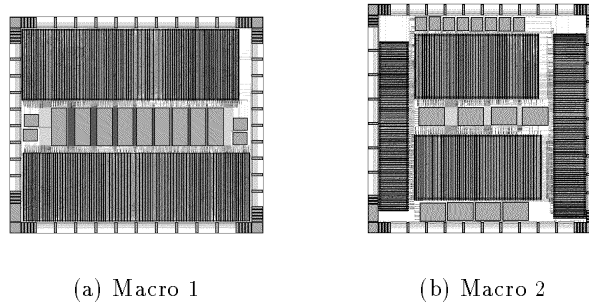
Two different integration methods have been explored and applied on several realistic applications. By using these experimental results we are now about to make a quantitative comparison between the two methods. Their respective advantages will be identified and studied. Results obtained up to now seem to indicate that derivation and synthesis should be strongly coupled into a single integration package within the whole design framework. For each specific application, the integrated ASICs could thus contain both derived and synthesized parts according to the efficiency of each method.

## References

- [1] P.M. Athanas and A.L. Abbott. Image Processing on a Custom Computing Platform. In *Proc. of the 4th International Workshop on Field-Programmable Logic and Applications*, pages 316–322. Springer Verlag, 1994.
- [2] R. Lauwereins, M. Engels, M. Adé, and J.A. Peperstraete. Grape-II: A System-Level Prototyping Environment for DSP Applications. *Computer*, 28(2):35–43, Feb. 1995.
- [3] B.A. Curtis and V.K. Madisetti. Rapid Prototyping on the Georgia Tech Digital Signal Multiprocessor. *IEEE Trans. on Signal Processing*, 42(3):649–62, Mars 1994.
- [4] J. Sérot, G.M. Quénot, and B. Zavidovique. Functional programming on a data-flow architecture : applications in real-time image processing. In *International Journal of Machine Vision and Applications*, volume 7, pages 44–56, 1993.
- [5] I.C. Kraljić, G.M. Quénot, and B. Zavidovique. From Real-Time Emulation to ASIC Integration for Image Processing Applications. In *Proc. of the 8th Annual IEEE International ASIC Conference*, pages 31–4, Sept. 1995. Austin, TX, USA.
- [6] F. S. Verdier and B. Zavidovique. A high level synthesis system for VLSI image processing applications. *To appear in International Journal of VLSI Design*, 1996.
- [7] V. Brecher. New techniques for patterned wafer inspection based on a model of human preattentive vision. *SPIE Journal on Applications of Artificial Intelligence : Machine Vision and Robotics*, 1708:452–459, 1992.

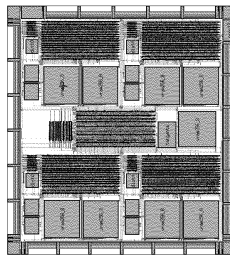
## 4 Example: a real-time defect detector

A real-time defect detector (94 operating DFPs, 800 MOPS) has been successfully emulated and integrated. The algorithm identifies and locates defective regions in strongly patterned images (e.g. wafers) by considering edge directions [7]. The  $1.0\ \mu\text{m}$  Atmel ES2 standard cell technology has been targeted (plus RAM/ROM compiler cells). A 2-chip set implementing the detector has been derived (Fig. 2); it features real-time performance on  $512\times 512$  images. One chip includes eight 512-bytes RAM cells (chip size  $109\ \text{mm}^2$ ), the other chip includes eight  $8\times 8$  multiplier cells (chip size  $111\ \text{mm}^2$ ).



**Figure 2: Layouts of the defect detector chips (different scales)**

The area of the synthesized chip (Fig. 3) is  $83\ \text{mm}^2$  (non-pipelined, real-time on  $228\times 228$  images). The chip includes nine 256-bytes RAM cells. As this is a modified version of defect detector, there are no multiplier cells in the synthesized chip.



**Figure 3: Layout of the synthesized defect detector chip**

## 5 Conclusion

The purpose of this research is to develop a coherent design framework for description, validation and integration of real-time image processing ASICs. The first two steps of the design flow (i.e. functional specification and emulation) are well-adapted to image processing applications. Using the DFFC emulator dedicated to image processing brings many advantages in terms of validation (systematic real-time performance, validation in the target environment).

(I/O FIFOs). These resources are automatically replaced by pipeline registers when possible (i.e. when there is no actual flow delay between neighboring DFPs). The remaining FIFOs perform a true flow delay (e.g. pixel delay), and their depths are reduced to the minimum. Manually collapsing neighboring DFPs into macro-DFPs according to the function they implement is also possible.

The result of derivation is a VHDL netlist at the register-transfer level which is fed into COMPASS' low-level tools (RT to gate level synthesis, floorplanning, place and route). Atmel ES2's 1.0  $\mu\text{m}$  CMOS standard cells technology has been especially targeted. The average gate count for a single derived DFP is 1.5K gates, for an average area of 1.5  $\text{mm}^2$ . Derived circuits are simulated using at most a few hundred input vectors.

### 3.2 High level synthesis

By using only the data-flow graph description of an algorithm as input instead of extensively exploiting the emulator architecture, the synthesis flow can be considered more general. The ALPHA synthesis tool optimizes the potential parallelism between the operations of an algorithm and builds a VLSI circuit with a reduced amount of computing resources. The target architectural model is a fully synchronized machine with one global control part. High speed pipelined architectures as well as highly parallel non-pipelined architectures can be also evaluated. In order to reach near-optimal solutions the synthesis tool implements a minimization of control complexity.

Furthermore, the synthesis method is able to handle hierarchy in the sense that a complex image processing algorithm (which is expressed as a map product of simpler functions) can be synthesized hierarchically. Small parts of the final architecture are synthesized independently first, included in a global VLSI library and then used for the whole chip design as a simple operative module. This point is fully exploited for the synthesis of the defect detector presented in Section 4. The overall synthesis flow is as follows:

1. The first step consists in reading the data-flow graph specification, then propagating data types through the graph (only primary inputs and outputs are explicitly typed in an FP program) and building the control-flow graph. The resulting hybrid representation is a Control-Data-Flow graph where each vertex represents a basic operation (logic, arithmetic) or a complete sub-graph (when hierarchy is used) and edges denotes data or control dependencies
2. The main RTL synthesis step is done thanks to two stochastic optimization algorithms (Simulated Annealing): scheduling and binding. The scheduling step optimizes the allocation of a time stamp for each operation in the graph. Module selection is achieved simultaneously with scheduling. Moreover, by using a new quality measure in the cost function (the CDFG regularity), the control part is also optimized [6]. The second algorithm performs simultaneously operator binding (allocation of functional unit instances to operations in the graph) and transfer resources minimization (registers and multiplexors).

The last step consists in creating the VHDL netlist of RTL components and generating the signal level description of the finite state machine controlling the architecture. These two VHDL descriptions are then fed to the gate-level and layout synthesis tool COMPASS.

The DFFC's inputs/outputs are B/W and color cameras/monitors and low-bandwidth data-flow I/O ports. A SPARC 2 workstation is the host for the DFFC's programming environment. The computer is synchronized with the digital video pixel clock. The 25 MHz maximum operating frequency is not a flip-flop toggle limit but the actual speed at which every part of the DFP – and the whole DFFC – can operate whatever the complexity of the operator assigned to a DFP.

The image processing algorithm to be emulated is expressed in a functional programming language which is automatically compiled and translated into a DFP graph using operators from a database containing 200 operators, from single-DFP to multiple-DFP operators (e.g. convolvers, LUTs, filters). The DFP graph is mapped into a DFFC network configuration ready to be loaded. The mapping implies the placement of each DFP/macro-DFP operator on the 3D mesh and the routing between operators. This can be done manually using an interactive graphic tool. An automatic tool performing the mapping step is currently under development. A complete 1024 DFPs configuration is loaded in about 15 seconds. The algorithm is executed in real-time using the CCD cameras as input flows and sending the output flows to video monitors. Debugging is done by executing the algorithm step-by-step using the low-bandwidth interfaces and the possibility to access any DFP register as well as the contents of the RAMs and FIFOs from the host.

Benefits of DFFC-based emulation include: efficient prototyping, painless debugging and fast integration. Algorithms are fully validated thanks to the total observability and controllability of the emulator. Furthermore, the prototype can be tested with any kind of input sequences in the target environment of the ASICs. The main limitation is that only low- to mid-level image processing applications can be targeted.

Several significant applications have nonetheless been successfully developed, including connected component labeling, colored object tracking, defect detection as well as a tool for interactive satellite image analysis.

### 3 ASIC integration

Two methods for generating a VLSI system after emulation have been developed: derivation and synthesis.

#### 3.1 Derivation from emulation results

During emulation, an architecture implementing the algorithm in real time and on real-life scenes is exhibited and validated. The derivation software analyzes and optimizes this architecture at the register-transfer level [5]. The optimizations performed are:

1. At the DFP level: each DFP is automatically reduced to its minimal equivalent implementation by removing its unused resources and adjusting its remaining resources: bitwidths for datapath resources, depth of each DFP pipeline. As far as control is concerned, an optimized gate-level netlist implementing the logic equations is generated from the state transition graph of each DFP. As an example, a DFP implementing a 16-bit adder whose 8 MSBs are not used is basically reduced to a “one-stage” pipeline containing an 8-bit adder plus 3 I/O FIFOs.
2. At the DFP graph level: each minimal equivalent implementation of a DFP is a data-flow operator, hence it still includes costly resources dedicated to flow management

[1]). Other emulators are dedicated to specific applications: Grape-II [2] and the Digital Signal Multiprocessor [3] (both based on DSP processors) are dedicated to emulating real-time Digital Signal Processing systems.

This paper presents a complete framework for designing image processing VLSI circuits based on an emulator dedicated to processing digital pixel streams on the fly: the Data-Flow Functional Computer. All the design tools are included in this framework: behavioral specification, algorithm validation and ASIC integration. By using the functional paradigm in specifying algorithms [4], the methodology is adapted to the intuitive and informal design of image processing applications.

The design flowchart is described in Fig. 1.

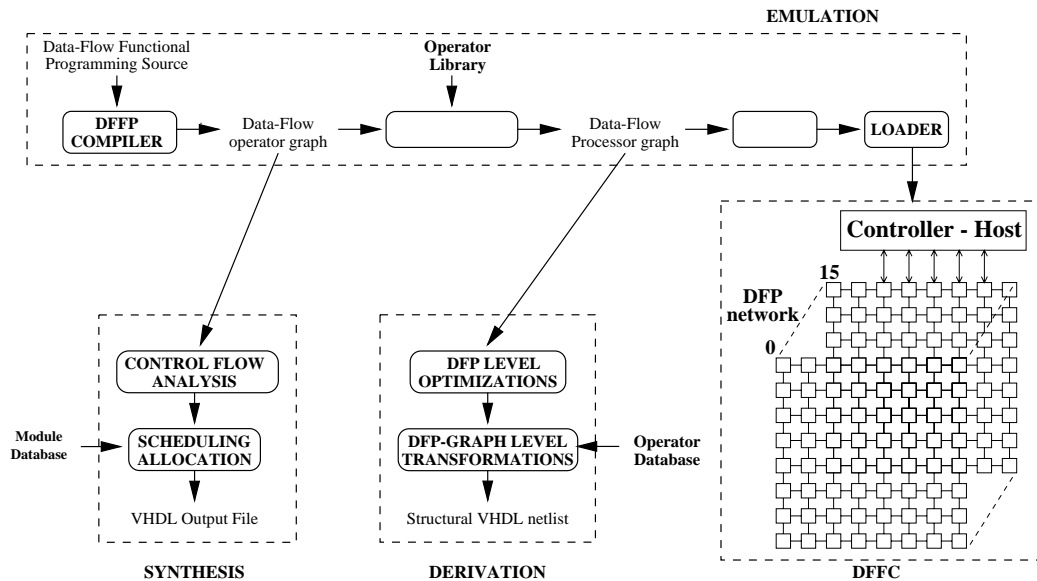


Figure 1: The design flowchart

## 2 Real-time emulation

The Data-Flow Functional Computer is dedicated to processing digital pixel streams on the fly [4]. It is a 3D  $8 \times 8 \times 16$  array of 1024 custom Data-Flow Processors (DFPs) achieving a peak computing power of 50 GOPS. The highly regular structure of the Data-Flow Functional Computer allows only local connections, however long distance connections are possible through DFPs configured as routing elements (an optimized place and route can yield a 50 to 60% DFP operating rate). Each DFP contains a programmable 3-stage pipelined datapath containing 3 input FIFO queues and 3 output queues, an  $8 \times 8$ -bit multiplier, a 16-bit 2901-type ALU, a  $256 \times 9$ -bit RAM and a 16-bit counter. The datapath is configured through a programmable controller (a  $64 \times 32$  bits program is provided for its description).

A chip containing 2 Data-Flow Processors has been fabricated in a  $1.0 \mu\text{m}$  CMOS technology. A single Data-Flow Processor can implement a wide range of low-level image processing operators, e.g. arithmetic/logic operations, counters, comparators, line/pixel shifts, 256-word FIFO, line/column sums, and histograms.

# Systematic Design of Image Processing ASICs through Real-Time Emulation

Ivan C. Kraljić\*, François S. Verdier<sup>◇</sup>,  
Georges M. Quénot<sup>†</sup> and Bertrand Zavidovique<sup>‡</sup>

\* Laboratoire Système de Perception  
DGA/Établissement Technique Central de l'Armement  
16 bis avenue Prieur de la Côte d'Or, 94114 Arcueil FRANCE  
<sup>◇</sup> IUT de CERGY/GRSC  
Université de Cergy-Pontoise, 95014 Cergy FRANCE  
<sup>†</sup> LIMSI-CNRS, BP133, 91403 Orsay FRANCE  
<sup>‡</sup> Institut d'Électronique Fondamentale  
Université de Paris XI, 91405 Orsay FRANCE

## Abstract

A complete environment for designing real-time image processing VLSI circuits is presented. At the core of the methodology resides a dedicated emulator, the Data-Flow Functional Computer (DFFC), whose peak capacity is 20 million gates operating at 25 MHz. Applications are firstly validated in their target environment (real time, real-world scenes) during emulation on the DFFC. Two integration methods are then available: derivation and synthesis. The derivation method optimizes the architecture validated on the emulator, while the synthesis approach is not constrained by the emulator architecture, and thus allows to generate other (optimized) architectures.

## 1 Introduction

The research presented in this paper aims at designing real-time image processing Application Specific Integrated Circuits (ASICs), with emphasis on the need for correct circuits. It is well-known that ASIC validation at each step of the design flow (from algorithm down to VLSI layout) is the main difficulty. The problem resides in the huge amount of input data (20 Mbytes/s. for color video) consumed in real-time operation. As image processing algorithms design (and fine tuning) requires much experimentation and heuristics, the validation of real-time image processing ASICs by simulation is at best extremely time-consuming, at worst prohibitive.

An efficient way to cope with this problem is to use an emulator, i.e. a set of reprogrammable hardware which can be configured by software to implement the ASIC as accurately as possible, at register-transfer or gate levels. Generic emulators are based on Field-Programmable Gate Arrays (FPGAs) using the static RAM technology (e.g. SPLASH-2