

From Real-Time Emulation to ASIC Integration for Image Processing Applications

Ivan C. Kraljić*, Georges M. Quénot[†] and Bertrand Zavidovique[‡]

Laboratoire Système de Perception

DGA/Établissement Technique Central de l'Armement

16 bis avenue Prieur de la Côte d'Or, 94114 ARCUEIL Cedex FRANCE

Abstract – A methodology for deriving image processing ASICs from the results of their real-time emulation on the Data-Flow Functional Computer is presented. The aim of the method is to reduce the time and effort required for synthesizing and validating ASICs after emulation. This is achieved by optimizing the architecture validated on the emulator and integrating the optimized resources. The results of the derivation of a defect detector are presented.

I. INTRODUCTION

The automation of Application Specific Integrated Circuit (ASIC) design has been an active field of research for many years. On one hand, a whole range of tools aimed at helping the traditional ASIC designer has been developed (e.g. combinational logic optimization, automatic placement/routing...); on the other hand high-level synthesis tools expected to generate a Register-Transfer and/or gate level netlist from a behavioral description of the design have been studied. Despite all these efforts, designing a correct ASIC (which will function as expected in its target environment) remains a far from trivial task. One effective way of ensuring that the ASIC will behave correctly once plugged in its environment is to emulate it. Emulated designs are also of a higher-quality since the number of cycles that can be run are several orders of magnitude more than with simulation (even accelerated). Once successfully emulated, the design has to be retargeted and integrated into one or more ASICs.

This paper presents a methodology for deriving image processing ASICs from the results of their real-time emulation on the Data-Flow Functional Com-

puter (DFFC) emulator. The design is firstly emulated in real time on the DFFC emulator dedicated to image processing [2, 3]. The DFFC is an $8 \times 8 \times 16$ array of Data-Flow Processors (DFPs) and it processes digital video streams on the fly at a rate up to 25 MHz pixel. The algorithm is expressed in a functional programming language which is translated into a DFP graph using operators from a database (200 operators). The function of each DFP is specified by a finite-state machine definition and a DFP implements a low-level image processing operator (e.g. adder, line/pixel delay, histogrammer...). During emulation, an architecture implementing the algorithm in real time on real-life scenes is exhibited.

II. DERIVATION FROM EMULATION RESULTS

A. Basic concept

Emulation on the Data-Flow Functional Computer yields a multi-level *validated* description of the design: at the highest abstraction level the design is represented by a data-flow graph (DFG), then it is defined as a network of finite-state machines (state transition graphs), finally it is specified by a Register-Transfer/gate level netlist. Each description can be independently used for generating the ASICs. Synthesis from the data-flow graph or state transition graphs yields optimized ASICs at the expense of a costly validation. Retargeting the RT/gate level netlist yields sub-optimal ASICs but minimizes the validation effort.

The approach to ASIC generation after emulation presented in this paper is an attempt to exploit all the results of the emulation: each one of the 3 different descriptions of the design is used in the derivation process at its best advantage. The derivation process consists in optimizing the validated Register-Transfer level description of the design in order to obtain a reasonably cheap (i.e. small silicon area) integrated design in the least amount of time.

*Email: ik@etca.fr

[†]G.M. Quénot is currently at LIMSI-CNRS, BP133, 91403 Orsay Cedex FRANCE. Email: quenot@limsi.fr

[‡]B. Zavidovique is also with the Institut d'Électronique Fondamentale, Université de Paris XI, 91405 Orsay Cedex FRANCE. Email: zavidov@etca.fr

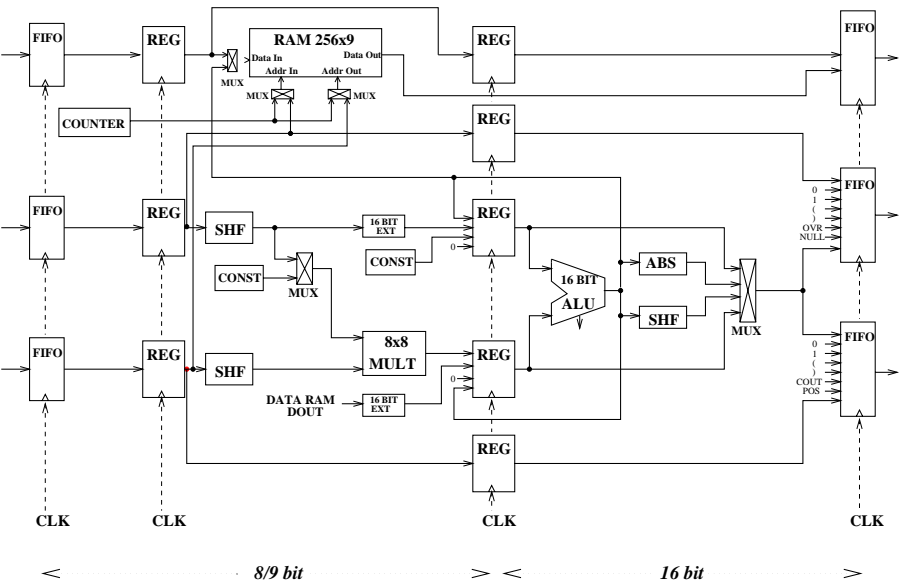


FIGURE 4: THE DFP'S DATAPATH AND ITS I/O FIFOs

has been made not to use the controller implementation, but its specification i.e. the state transition graph. This choice allows for several architectures such as ROM-based , PLA-based or standard cells-based. Currently only the last option is available: the derivation software generates an optimized gate-level netlist implementing the logic equations.

- *DFP I/O ports.* The I/O port reduction consists mainly in replacing the crossbars (between FIFOs and I/O ports) by static connections implementing only the needed (application dependent) communication links.

At this point in the derivation process the design is represented by a graph of “data-flow specific processors”, e.g. data-flow adders, data-flow line delays, data-flow histogrammers... They still include costly flow-management resources (I/O FIFOs and ports). They are dealt with by the high level transformations.

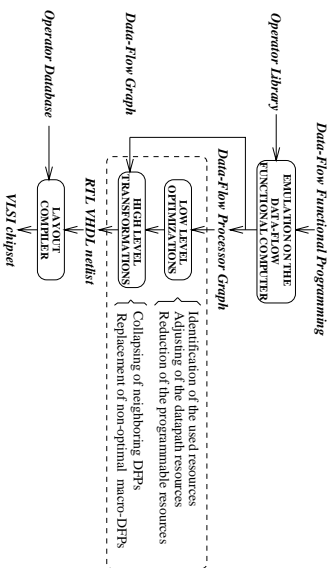


FIGURE 5: DESIGN FLOW

B. High level transformations

The flow-management resources are eliminated through collapsing neighboring DFPs into macro-DFPs according to the function they implement or to user’s specifications. For instance, the `atan` arc tangent operator of the defect detector required a 1024×8 Look-Up Table (LUT) and was emulated using 9 DFPs (4 LUTs, 5 selectors) due to the limitations (a DFP contains a 256×9 RAM). This operator was derived into a macro-DFP containing a 1024×8 ROM.

The architecture of the derived ASICs is finite-state machine with datapath and their execution model is data-flow. The ASICs are stand-alone autonomous circuits but they remain compatible with the DFFC.

C. Output of the derivation process

The output of the derivation process is a VHDL RT/gate level netlist describing the optimized design. The hierarchy of the netlist is shown in figure 6. A VHDL package provides the netlist generation with all necessary RT level operators. The “DFP CORE” VHDL entity can be a single DFP core or a collapsed macro-DFP core. Note that the derived design can be described either as a re-usable macro (without pads) or as an actual chip (with pads).

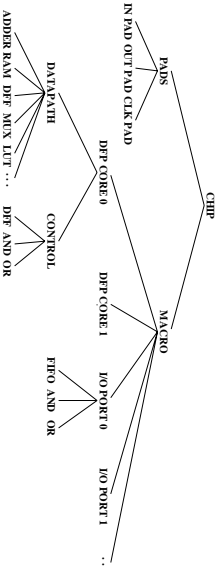


FIGURE 6: HIERARCHY OF THE VHDL NETLIST

IV. IMPLEMENTATION ISSUES

The VHDL netlist contains both Register-Transfer level (datapath) and gate level (controller and I/O

control) primitives. The layout compilation is performed on COMPASS' back-end tools. The netlist is technology-independent; theoretically the corresponding design can be implemented in any technology, e.g. gate array, standard cells, datapath and memory compiler cells, or even field-programmable gate array. Two implementation options have been especially explored: full standard cells and mixed standard/datapath cells (plus RAM/ROM/multiplier compiler cells for both options) in ESS's 1.0 μm CMOS technology. The main differences between the two options are summarized in table I. In practice the only possible way of manually placing mixed implementations is to firstly place and route each DFP or macro-DFP *individually*, then to assemble (place and route) the resulting blocks. This is an incremental procedure: the floorplan is refined as long as the final layout is not satisfactory, the designer has to modify the placement of the full design as well as the placement of each (macro-)DFP. A gain of 10% between a poorly planned placement and a carefully studied one is easily achieved. One interesting feature of this method is the high re-usability of the (macro-)DFPs. Each placed and routed (macro-)DFP can be used for any design where it is needed.

TABLE I: IMPLEMENTATION OPTIONS

	Full standard cells	Mixed standard/datapath cells
Placement	Automatic	Manual
CPU power required	Huge	Low
Re-usability of macros	Difficult	Easy
Area	Larger	Smaller
Operating frequency	Lower	Higher

The layout generation is the most time-consuming step in the derivation process. Whether the designer chooses a full standard cells implementation or a mixed one, the generation of a satisfactory layout can take up to a few days. This is to be compared to the few minutes necessary to derive the netlist of the design after emulation. In order to cope with this problem, we plan to generate automatically placement constraints for cells and connectors and guidances for buses and signals. Furthermore, a trade-off between the quality of the layout and the generation delay shall be studied.

V. RESULTS

Typical results of derived Data-Flow Processors (featuring low level optimizations and mixed standard/datapath cells) are shown in table II. The ap-
 plication of high level transformations yields a further area reduction of about 60% for derived circuits. With the 1.0 μm technology used, a maximum of 100 DFPs can be safely integrated on a single ASIC. In the future a 0.7 μm technology will raise the density to 200

DFPs per ASIC. Derived circuits are validated by simulation, firstly at the (macro-)DFP level, then at the circuit level. Since it is actually a validation of optimized (validated) circuits, the amount of simulation needed is restricted to a few hundreds input vectors.

The defect detector algorithm consists in identifying and locating defective regions in strongly patterned images (e.g. wafers) by considering edge directions. It is based on a model of human vision presented in [1]. Basically the detector identifies the regions where the edges have directions that are weakly represented in the whole image (a pattern defect is defined as a difference of the local mean edges direction compared to a global measure of these directions). The data-flow graph of the defect detector is shown in figure 7. The algorithm involves about 70 instructions/pixel, thus at a rate of 25 images/second (considering 572×768 images) the computing power required is about 800 MIPS. The defect detector has been emulated on the Data-Flow Functional Computer in real time using 108 Data-Flow Processors.

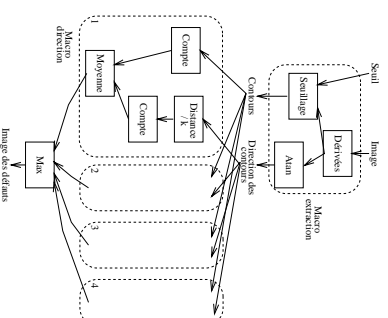
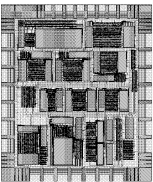


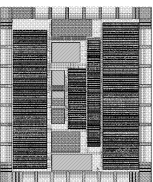
FIGURE 7: DATA-FLOW GRAPH OF THE DEFECT DETECTOR

The defect detector has been derived by respecting the hierarchy of the data-flow graph: each macro-operator has been derived in a single ASIC. The technology used was the CMOS 1.0 μm ESS2. The extraction macro has been implemented in both standard and datapath cells. The area of the chip is 36.24 mm^2 (core area 23.59 mm^2). The direction macro has been implemented in standard cells. The chip area is 47.79 mm^2 (core area 34.09 mm^2). The layouts of the chips are displayed in figure 8. This is obviously not the least silicon-greedy way of deriving the defect detector. It should indeed be derived as a single DFP graph in order to apply the optimizations to the whole graph and not independently on each macro. Both datapath and standard cells should be used. We estimate that such a derived defect detector could be integrated onto a single chip whose core area would be about 100 mm^2 .

Operator	DFF			max	Pixel delay	Line delay	8-bit Histo	512-word FIFO
	Full	abs	add					
Area (mm ²)	35.64	1.12	1.39	1.91	1.44	1.50	3.91	4.49
FIFO/Operator	0.11	0.51	0.58	0.42	0.65	0.62	0.24	0.17



(a) EXTRACTION



(b) DIRECTION

FIGURE 8: LAYOUTS OF THE DERIVED DEFECT DETECTOR MACROS (DIFFERENT SCALES)

VI. CONCLUSION

A methodology for deriving image processing ASICs from the results of their real-time emulation on the Data-Flow Functional Computer has been presented. By exploiting the 3 different descriptions of the design validated during emulation, the time and effort required for synthesizing and validating ASICs after emulation are reduced. The derivation process optimizes the architecture validated on the emulator at the register-transfer level.

Future work consists in studying the automatic generation of placement constraints for cells and connectors and guidances for buses and signals in order to improve the layout generation.

REFERENCES

- [1] 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.
- [2] G.M. Quénot, I.C. Kraljić, J. Sérot, and B. Zavidovique. A Reconfigurable Compute Engine for Real-Time Vision Automata Prototyping. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 91–100, April 1994. Napa, CA, USA.
- [3] J. Sérot, G. Quénot, and B. Zavidovique. Functional programming on a dataflow architecture: applications in real-time image processing. *Machine Vision and Applications*, 7(1):44–56, 1993.