High Level Synthesis by Systematic Derivation of Vision Automata from Emulation Results

I.C. Kraljić, G.M. Quénot and B. 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

Tel: +33-1-42 31 97 07 Fax: +33-1-42 31 99 64

Email: ik@etca.fr

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. Indeed, the same architecture which emulated the application is the entry point of the derivation process. Thus, the “synthesis” is restricted to the optimization at the Register-Transfer level of the validated emulator architecture, as a consequence the validation of derived ASICs is easier. The derivation process is demonstrated on the direction extraction macro of a defect detector application.

Keywords

High level synthesis, VLSI, image processing, emulation.

1 INTRODUCTION

High level (or architecture) synthesis takes as input an algorithm expressed in a high level language and generates an RTL netlist implementing the algorithm according to explicit constraints. The resulting netlist must be thoroughly simulated in order to validate its behavior. The increasing interest in rapid prototyping, formal verification or FPGA-based emulation indicates that simulation does not cover all the aspects of design validation. When real-time vision applications are concerned, the specification of the application is vague and imprecise and the enormous amount of input data requires weeks of traditional simulation (the computational requirements of real-time image processing range up to

*G.M. Quénot is currently at LIMSI-CNRS, BP133, 91403 Orsay Cedex FRANCE. Email: quenot@etca.fr

Figure 1 Emulation flow.

billions of operations per second). Furthermore, the application should be validated in its environment (a problem seldom addressed by simulation).

This paper describes an approach called **derivation from emulation results** where the validation step precedes the synthesis step: the algorithm is firstly emulated in its environment (with input from video cameras) on a general image processing computer (the Data-Flow Functional Computer), then the actual computer configuration is analyzed, optimized and finally implemented as a VLSI chipset. The RTL netlist implementing the algorithm is thus directly derived from the hardware which emulated the algorithm: **emulation provides simultaneously the synthesis of an RTL netlist and its validation**. Hence the post-emulation “synthesis” step is reduced to the optimization of the emulator’s active resources. Since data dependency is preserved, total equivalence of the derived RTL netlist’s behavior with the results obtained during emulation is achieved (functional behavior and performance). As a consequence the amount of simulation required to validate the derived ASICs is restricted.

Firstly, the application is emulated in real time on the DFFC (Figure 1) [Quénot *et al.* 1994] [Sérot *et al.* 1993]. 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). Figure 2 summarizes the DFP’s architecture. The programmable 3-stage pipelined datapath contains 3 input FIFO queues and 3 output queues, an 8×8 -bit multiplier, a 16-bit 2901-type ALU, a 256×9 bits RAM and a 16-bit counter. The datapath is configured through a programmable controller (a 64×32 bits program is provided for its description). The DFP architecture has been designed to implement efficiently a wide range of low- to mid-level image processing operators (arithmetic/logic operations, 8/16-bit histogrammer, line/pixel delays and sums). Several significant applications have been successfully emulated, including colored object tracking, main direction follower, contrast enhancement and defect detection. In the following, we call a resource of a Data-Flow Processor any DFP datapath element, the DFP controller and its I/O ports.

Figure 2 Architecture of the Data-Flow Processor.

2 DERIVATION FROM EMULATION RESULTS

Principle

The actual DFFC configuration implementing the algorithm is used as an input to the derivation process. The process is based on a processor-level analysis: identification of the effectively used resources and optimization of these resources. The resulting automaton is implemented as a VLSI chipset whose architecture is pipelined Finite State Machine with Datapath. The VLSI approach to derivation is highly facilitated by 1) the granularity of the Data-Flow Processor which is at the 8/16-bit operator level and 2) the relative simplicity of the DFP architecture which allows a deterministic knowledge of whether a resource is used or not. Derivation is similar to retargeting an FPGA implementation into an ASIC: the conversion of the design into a different technology (*e.g.* gate array) yields a smaller and faster implementation. Thanks to the coarse grain architecture of the Data-Flow Processor derivation is far more easily automated.

Justification

One question that immediately arises is whether one could not reduce a graph of several dozens processors into a few chips. Table 1 shows the average (static) resource utilization (and actual area) of DFP resources computed on several non-trivial algorithms (3×3 convolution, Nagao-like filter, erosion, defect detector...). It is clear that the resource utilization of the presented algorithms is low. Furthermore, the average area of a DFP after resource reduction (*i.e.* removal of the unused resources) can be estimated. For each resource i , let us call u_i its average static utilization rate and a_i its actual area (in mm^2). The estimated average area of a DFP after resource reduction is

$$A = \sum_{i \in \{\text{resources}\}} u_i \times a_i.$$

Using Table 1 yields $A = 7.35 \text{ mm}^2$ ($1\mu\text{m}$ technology). Thus, considering only the lowest-level optimization step (discarding the unused resources) shall yield a theoretical reduction in area of 80%.

Table 1 Average static utilization rate and area of DFP resources

Resource	I/O Ports	I/O FIFOs	Data-Path	Data RAM	Controller
Utilization rate	42.4%	42.2%	22.2%	20.0%	4.9%
Area (mm ²)	9.34	3.74	4.85	1.60	8.55

Figure 3 Derivation flow.

3 THE DERIVATION PROCESS

This Section explains the optimizations performed during derivation. In accordance to the emulation flow steps, 3 different representations for an algorithm are available (from high to low level of abstraction): 1) data-flow graph, 2) Finite State Machine (FSM) network and 3) RTL netlist. Whereas most traditional high level synthesis systems use only the data-flow representation (e.g. [Verdier & Zavidovique 1993]), derivation will also use the other 2 representations: low level optimizations are performed on the RTL netlist and high level transformations are performed on the FSM representation and data-flow graph*. The block diagram of the derivation process is shown in Figure 3.

The low level optimizations

Their aim is to reduce each DFP to its **minimal equivalent implementation** by discarding its unused resources and adjusting its remaining resources. These optimizations are performed on the RTL netlist with input from the Finite State Machine definition of each DFP. They are applied to each DFP and to connections between adjacent DFPs.

1. **Removal of the unused resources:** an analysis of the static and dynamic (FSM) definitions of the DFP operator determines precisely whether a datapath resource (ALU, MUX, FIFO...) is used or not. A resource is unused if it produces no output flow or if its output flow is equal to its input flow or if it produces a constant flow. Unused resources are removed or replaced by buses.
2. **Adjusting bitwidths and depths:** the bitwidth of each datapath resource and the depth of each I/O FIFO and of the DFP pipeline are restricted to their minimum value.

*Note that more and more high level synthesis systems perform post-synthesis optimizations on the synthesized RTL netlist [Eveking & Höreth 1993] [Papachristou *et al.* 1993].

3. **Reduction of the programmable resources:** reducing a programmable resource (controller, crossbar or I/O port) means “freezing” it to its specific function in each DFP operator. The controller which was implemented with a 64×32 program RAM is implemented in standard cells. The I/O ports of a DFP were connected to its FIFOs through a crossbar; in a derived DFP the (fixed) connections are implemented statically.

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 two-word I/O FIFOs. These immediate optimization steps bring huge area reduction.

The high level transformations

The low level optimizations reduce each DFP to its minimal equivalent implementation. This implementation is a data-flow operator, hence it still includes costly resources dedicated to flow management (I/O FIFOs and ports). These resources are eliminated through collapsing of neighboring DFPs into macro-DFPs according to the function they implement or to user specification. For instance, the `atan` arc tangent operator of the extraction macro (Figure 4a) required a 1024×8 Look-Up Table (LUT) and was emulated using 9 DFPs (4 LUTs, 5 selectors) due to the DFP limitations (a DFP contains a 256×9 RAM). This operator was derived into a macro-DFP containing a 1024×8 ROM.

The derived ASICs are validated by simulation as follows: 1) each derived DFP (or collapsed macro-DFP) is validated **if it has not already been derived and validated** and 2) the whole derived ASIC is validated. Typically a few dozens input vectors are enough to validate the function of each derived circuit.

4 RESULTS

Output of the derivation process is a VHDL netlist at the Register-Transfer Level which is fed into a commercial layout compiler (COMPASS). Derived circuits consist of both single and macro Data-Flow Processors implemented in standard cells and datapath compiler cells. Considering a $1\mu\text{m}$ CMOS technology the average size of a derived DFP featuring only low level optimizations is 4.4 mm^2 (Table 2). This implies the 3 low level optimizations, thus it explains the difference with the theoretical area computed in Section 2 which included only the first low level optimization. The I/O FIFOs occupy a significant part (about 43%) of the derived processors' area, this justifies the need for high level transformations that deal with this problem. The high level transformations decrease the area by about 60%. Hence 50 derived DFPs can be integrated on a single chip. The use of a $0.7\mu\text{m}$ technology shall raise this density up to 100 DFPs/chip. If the derived circuit must be the smallest possible for a large application, one can envision the use of a multichip module technology. Currently only the I/O FIFOs are implemented using COMPASS' datapath compiler, while the rest is implemented in standard cells. Subsequent use of the datapath compiler shall further reduce the datapath's area [Leveugle & Safinia 1993].

The direction extraction macro of the defect detector has been successfully derived using the derivation tool and COMPASS' back-end tools. The direction extraction chip (Figure 4) contains 26 initial DFPs in 50.7 mm^2 (21K gates, core size 35.7 mm^2 , $1\mu\text{m}$ CMOS technology). The chip has been derived with both low and high level optimizations, it features a gain in area of 60% compared to the extraction chip derived with only low level

Table 2 Initial/derived Data-Flow Processors in a 1 μ m technology (areas in mm²)

Operator	Initial DFP	abs	add	and	max	256-w FIFO	Pixel delay	Line delay	Histogram
Total area	35.64	2.6	4.8	3.6	3.9	4.9	4.1	3.9	7.1
Control/DP	1.76	0.24	0.21	0.17	0.12	0.07	0.43	0.47	0.28
FIFO/Op	0.11	0.48	0.52	0.50	0.47	0.25	0.45	0.47	0.26

(a) Data-flow graph.

(b) Derived chip.

Figure 4 The direction extraction macro.

optimizations. The chip has been validated by simulation using only a few dozens input vectors at a clock frequency of 12 MHz. The chip alone can process images with at most 512 pixels/line due to the on-board 512-word FIFO; but the connection of an external FIFO relaxes this constraint. The computing power of the chip is about 110 MIPS.

5 CONCLUSION

An approach to high level synthesis called derivation from emulation results has been presented. It exploits the results of the emulation of an algorithm in real time and in its environment. The emulation step provides simultaneously the synthesis of an RTL netlist (the emulator's active resources) and its validation; then the derivation step optimizes this RTL netlist and a commercial layout compiler is used to derive a chipset implementing the algorithm. The derivation concept has been implemented for real-time vision applications: a general vision computer has been built (the 1024-processor Data-Flow Functional Computer), the computer is programmed in a functional programming language.

Our current work consists in improving the derivation process: apart from the already cited generalization of datapath compiler cells in derived circuits, an area of research is the optimal partitioning of Data-Flow Processor graphs into derived chips. The natural par-

tioning of DFP graphs (according to the data-flow graph hierarchy) does not necessarily yield an optimal result in terms of area/performance of the derived chipset.

6 REFERENCES

- Allart, E. and Zavidovique, B. (1987) Functional Image Processing through Implementation of Regular Data-Flow Graphs. In *21st Annual Asilomar Conference on Signals, Systems on Computers, Pacific Grove, CA*.
- Eveking, H. and Höreth, S. (1993) Optimization and Resynthesis of Complex Data-Paths. In *Proc. of the 30th ACM/IEEE Design Automation Conference*, pages 637–41.
- Leveugle, R. and Safinia, C. (1993) Generation of optimized datapaths: bit-slice versus standard cells. In G. Saucier and J. Trilhe, editors, *Synthesis for Control Dominated Circuits (IFIP Transactions A-22)*, Elsevier, pages 153–66.
- Papachristou, C. and Harmanani, H. and Nourani, M. (1993) An Approach for Redesigning in Data Path Synthesis. In *Proc. of the 30th ACM/IEEE Design Automation Conference*, pages 419–23.
- Quénot, G.M. and Kraljić, I.C. and Sérot, J. and Zavidovique, B. (1994) A Reconfigurable Compute Engine for Real-Time Vision Automata Prototyping. In *IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, USA, pages 91–100.
- Sérot, J. and Quénot, G.M. and Zavidovique, B. (1993) Functional Programming on a Data-flow Architecture: Applications in Real-Time Image Processing. *International Journal of Machine Vision and Applications*, pages 44–56.
- Verdier, F. and Zavidovique, B. (1993) A Complete Environment for Global Architecture Synthesis. In *Proc. of Computer Architectures for Machine Perception*, pages 77–81.

7 BIOGRAPHY

I.C. Kraljić was born in Paris, France, February 1970. He is a Ph.D. student in the Laboratoire Système de Perception at the Établissement Technique Central de l'Armement. He received an M.S. degree in Electrical Engineering from University Paris XI, Orsay, France in 1992. His current research interests include high level synthesis, rapid ASIC prototyping and image processing.

G.M. Quénot was born in Casablanca, Morocco, May 1960. He received the degree of Engineer from École Polytechnique in 1983, and a Ph.D. in Computer Science from University Paris XI, Orsay, France, in 1988. From 1988 to 1991 he was in charge of the hardware activities for real-time image processing at ETCA. He is now a researcher at LIMSI, a French National Agency for Scientific Research (CNRS) Laboratory. He is mainly interested in parallel computer architectures, VLSI circuit design and image processing.

B. Zavidovique is professor in the EE Department at University Paris XI, and a scientific advisor at ETCA for the problems of real-time processing and robotics. He was invited at UC Berkeley-EECS, in 1986. His research interests include perception systems, real-time architectures and programming methods within the frame of circuit integration. Since his Doctorate of Sciences in Robot Vision from the University of Franche Comté (France) in 1981, he has published more than two hundred scientific papers in image processing, sensor fusion, computer architecture, intelligent control and learning.