

RETARGETING FIELD-PROGRAMMABLE OPERATOR ARRAYS

Ivan C. Kraljić, Georges M. Quénot*,
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

Email: ik@etca.fr

ABSTRACT

The Field-Programmable Operator Array (FPOA) is a very coarse grain Field-Programmable Device designed to implement a wide range of low- to mid-level image processing operators. An FPOA contains 2 basic blocks; each basic block includes a configurable datapath (containing 8/16-bit data-flow operators and a 256×9 RAM), a configurable controller and six 10-bit-wide I/O ports. The Data-Flow Functional Computer emulator contains 512 FPOAs in an $8 \times 8 \times 8$ array and is aimed at rapid prototyping of real-time vision automata. This paper presents a methodology for retargeting Field-Programmable Operator Array based designs into VLSI circuits consisting of standard cells and datapath (compiler) cells. The retargeting process is highly facilitated by the 8/16-bit operator-level granularity of the FPOA; it includes removal of the unused resources and optimization of the remaining resources. A retargeted extraction macro (edge detection and computing of their directions) is presented.

*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: zavido@etca.fr

INTRODUCTION

The interest in ASIC emulation [2, 5] has been growing since a few years, mainly because Field-Programmable Gate Arrays (FPGAs) have become powerful enough in terms of speed and density to emulate realistic designs. Once a design has been validated on the emulator, a corresponding cheaper version of the design has to be derived in the form of a set of VLSI circuits. Two methods are available for retargeting FPGA-based designs:

1. A direct conversion of an FPGA implementation into its mask-programmed equivalent. This process is performed by the FPGA vendors: Xilinx offers such a conversion path [6]. The transformation consists in replacing the programmable elements with fixed metal connections.
2. Conversion of the FPGA implementation into another technology. This requires an actual resynthesis of the design since the timing of signal paths is altered.

The Data-Flow Functional Computer (DFFC) is an emulator developed at ETCA dedicated to real-time image processing. It contains 512 very coarse grain FPGAs called Field-Programmable Operator Arrays. After emulation on the DFFC the FPOA-based applica-

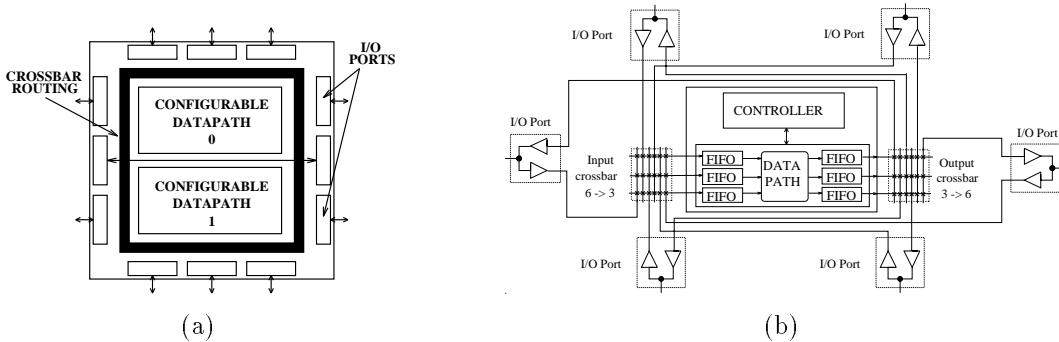


Figure 1: Architecture of the FPOA (a) and its basic block (b)

tion has to be retargeted into a cheaper implementation. This paper presents a retargeting methodology for Field-Programmable Operator Arrays that is intermediate between the two conventional processes cited above. Indeed, the dedication of the FPOA to real-time image processing (including its 8/16-bit operator-level granularity) allows an automatic conversion including optimizations while avoiding a full resynthesis.

EMULATION FLOW

The Field-Programmable Operator Array (FPOA) [3] is a very coarse grain FPGA whose granularity is at the operator level instead of the gate level. Where the FPGA's basic blocks include a few gates, the FPOA's basic blocks include a few 8/16-bit data-flow operators; and where the FPGA's I/O blocks and switch matrices handle 1-bit signals, the FPOA's I/O ports and switch matrices handle 10-bit buses. The counterpart of the very coarse grain choice for the basic blocks is that rather little of them can be integrated in a single chip (our current implementation packs 2 basic blocks in a $8.9 \times 9.6 \text{ mm}^2$ chip processed in a $1 \mu\text{m}$ CMOS technology for a total of 33,000 gates). However, the FPOA is designed in such a way that assembling them in three-dimensional meshes automatically constitutes a bigger-size FPOA with the same block struc-

ture and without functional discontinuity or performance loss. Indeed, the FPOA is intended to be used in large arrays.

Each FPOA includes two basic blocks (fig. 1). A basic block contains a configurable datapath (datapath, controller, 6 I/O FIFOs) and 6 I/O ports. The 3-stage pipelined datapath is connected to 3 input FIFO queues and 3 output queues, it contains an 8×8 -bit multiplier, a 16-bit 2901-type ALU, a 256×9 bits RAM, a 16-bit counter as well as multiplexors, shifters and absolute value operators. The datapath is configured through a programmable controller (a 64×32 bits program is provided for its description). The architecture of the basic block has been designed to implement efficiently a wide range of low level image processing operators (arithmetic/logic operations, 8/16-bit histogrammer, line/pixel delays or sums).

The Data-Flow Functional Computer emulator contains 512 FPOAs in an $8 \times 8 \times 8$ array and is fully equivalent to a monolithic FPOA containing $8 \times 8 \times 16$ basic blocks. The emulator is synchronized with the digital video pixel clock (up to 25 MHz) and it operates at the actual speed of the emulated hardware since it is optimized at the operator level. The DFFC emulator is able to emulate several millions of gates for a billion of cycles within a minute.

The DFFC is programmed in a functional programming language where each functional primitive represents an FPOA basic block or

a macro of basic blocks taken from an operator database (200 elements). The algorithm is executed in real time on the DFFC; input flows come from B/W and color cameras, and output flows are sent to monitors. All the operations corresponding to a single pixel are carried out in parallel at each time step (functional parallelism or MISD) which is the opposite of data-parallelism where a single operation is performed for all the pixels at each time step (SIMD). The machine clock is exactly the digital video pixel clock. Since pixels are processed on the fly within digital video streams, the number of basic blocks in the array roughly determines the number of operations per pixel that can be computed in real-time. Several applications have been successfully prototyped including colored object tracking, main direction follower, contrast enhancement [4] and defect detection. *During the emulation both specification and an architecture suitable for integration are validated simultaneously.*

RETARGETING METHODOLOGY

The retargeting process takes as input the actual FPOA graph that has been validated on the DFFC emulator. This is a multi-level description that can be seen as: 1) a data-flow graph, 2) a Finite State Machine (FSM) network and 3) a Register Transfer Level (RTL) netlist. In the following, any element of the RTL netlist (any datapath operator, any I/O port, the controller) is called *a resource*. The retargeting process consists in the optimization of the actually used resources of the emulator. Note that *the lowest abstraction level of this description is limited to the RT level, which has two advantages:*

1. Resource optimization is highly facilitated. Furthermore, the relative simplicity of the FPOA architecture allows a

deterministic knowledge of whether a resource is used or not.

2. The validation of the retargeted implementation, although it must be performed at the structural level, requires a far smaller effort than what is the case with conventional FPGA resynthesis.

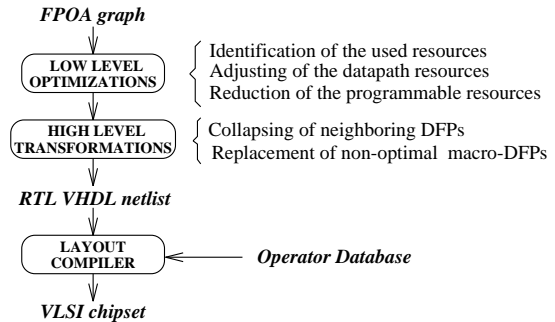


Figure 2: The FPOA retargeting process

Fig. 2 depicts the FPOA retargeting process. The optimizations/transformations are all defined at a basic block level.

LOW LEVEL OPTIMIZATIONS

They are performed on the RTL netlist with input from the Finite-State Machine definition of each basic block. They are applied to the datapath and the I/O ports. They reduce each basic block to its minimal equivalent implementation by 1) removing the unused datapath resources, 2) adjusting the bitwidths of datapath resources and depth of the pipeline, and 3) reducing the programmable resources (“freezing” the controller and I/O ports to their specific function).

Removing the unused resources is performed by analyzing the FSM description of the basic block operator: the state transition graph defines an active sub-path in the (configurable) datapath (the data will flow only through this sub-path). The resources present on the active sub-path are the only ones that are kept (see fig. 3). Each one of these resources is then reduced to its smallest implementation: for instance in the **add** operator of

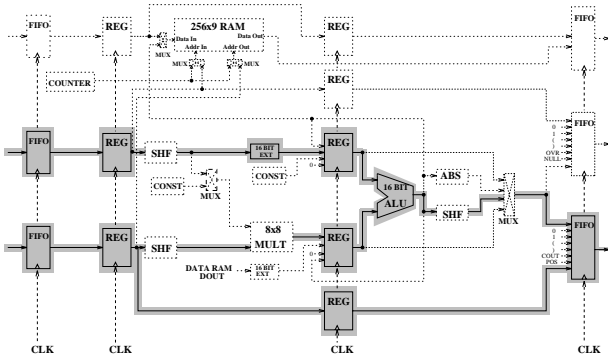


Figure 3: Active sub-path of the add operator (in shaded areas)

fig. 3, the 16-bit ALU is reduced to an 8-bit adder (since the 8 MSBs are not needed).

The controller reduction is straightforward: a basic block operator is defined by a Finite State Machine whose state transition graph is available. In a basic block the controller is programmable hence the FSM is implemented with a RAM (fig. 4a). In a derived basic block the controller need not be configurable: the FSM can be implemented with a ROM, a PLA or in standard cells. Currently only the standard cells implementation is available (fig. 4b). The average utilization rate of the 64×32 RAM computed on several applications is only 5% (only a small number of basic block operators are configured through an FSM of more than 10 states) which means that the controller reduction will yield a dramatic area reduction of the controller.

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. Fig. 5a illustrates the way basic blocks communicate through output FIFOs and ports from the *father*-block connected to input ports and FIFOs from the *son*-blocks. In derived circuits (fig. 5b), 2 cases can occur. If the connection is from one output FIFO (and port) to a single input FIFO (and port), then they are replaced by a single FIFO whose depth is adjusted to the

requirements determined by the basic blocks' function. If the connection is from one output FIFO (and port) to more than one input FIFO (and port), all the involved ports are merged into a replicate port which implements the necessary control logic (a few gates). The depths of the FIFOs are adjusted as required.

HIGH LEVEL TRANSFORMATIONS

The low level optimizations reduce each basic block 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 basic blocks into macro-blocks according to the function they implement or to user specification. For instance, the `atan` arc tangent operator of the defect detector application required a 1024×8 Look-Up Table (LUT) and was emulated using 9 basic blocks (4 LUTs, 5 selectors) due to the limitations (a basic block contains a 256×9 RAM). This operator was derived into a macro-block containing a 1024×8 ROM.

Derived circuits consist of single FPOA basic blocks and collapsed macro-blocks. Their architecture is Finite State Machine with Datapath and their execution model is data-flow. Retargeted circuits are compatible with the FPOA I/O protocol and hence they can be connected to the DFFC. They can also be used as stand-alone autonomous circuits.

The output of the retargeting process is a VHDL netlist at the RT level which is fed into COMPASS' low level tools. Only the I/O FIFOs can currently be implemented using COMPASS' Datapath Compiler, the remaining resources are implemented in standard cells. Circuits featuring FIFOs implemented in datapath cells are smaller than circuits featuring FIFOs implemented in standard cells at the expense of a more complicated (and not fully automatic) placement and routing step. The use of the Datapath Compiler will be generalized for all the datapath.

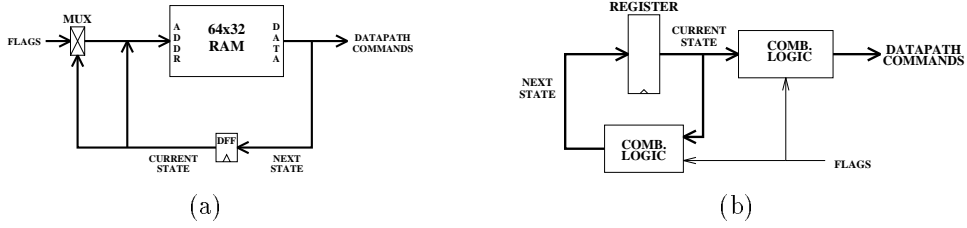


Figure 4: Controller implementation: initial (a) and derived (b)

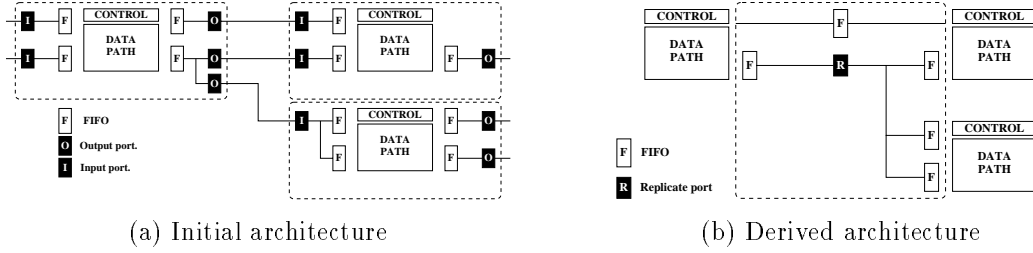


Figure 5: Communication between basic blocks

RESULTS

Derived basic blocks (featuring low level optimizations) are shown in table 1. They do not include any I/O port since the I/O port configuration is application-dependent (this is not significant since a derived I/O port contains a few gates). With a $1\mu\text{m}$ CMOS technology the average area of a basic block is 2 mm^2 . The appliance of high level transformations yields an area reduction of about 60% for retargeted circuits. Hence a maximum of 100 derived basic blocks can be integrated on a single chip. The use of a $0.7\ \mu\text{m}$ technology is expected to raise this density up to 200 blocks/chip.

Derived circuits are validated by simulation, firstly at the basic block level (if the basic block has not already been derived and validated), then at the circuit level. As it is actually a validation of optimized (validated) circuits, the amount of simulation needed is restricted to a few dozens input vectors.

A defect detector has been successfully emulated on the Data-Flow Functional Com-

puter. The algorithm [1] consists in identifying and locating defective regions in strongly patterned images (fig. 6a). The defect detector has then been retargeted into a $1\mu\text{m}$ CMOS technology. Fig. 6b presents the retargeted extraction macro (edge detection and computing of their directions). The chip includes 26 basic blocks (21K gates) in 50.7 mm^2 (core size 35.7 mm^2), it has been validated by simulation using a few dozens input vectors. The chip can process images with at most 512 pixels/line due to the on-board 512-word FIFO; but larger images can be processed by connecting an external FIFO. The computing power of the chip is about 110 MIPS.

CONCLUSION

A retargeting methodology for Field-Programmable Operator Arrays (FPOA) has been presented. After emulation on the FPOA-based Data-Flow Functional Computer emulator, a vision application is derived into a VLSI chipset consisting of standard and

| Operator | Initial block | abs | add | max | Pixel delay | Line delay | 8-bit Histo | 512-word FIFO |
|-------------------------|---------------|------|------|------|-------------|------------|-------------|---------------|
| 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 |

Table 1: Initial and derived basic blocks in a 1 μ m CMOS technology

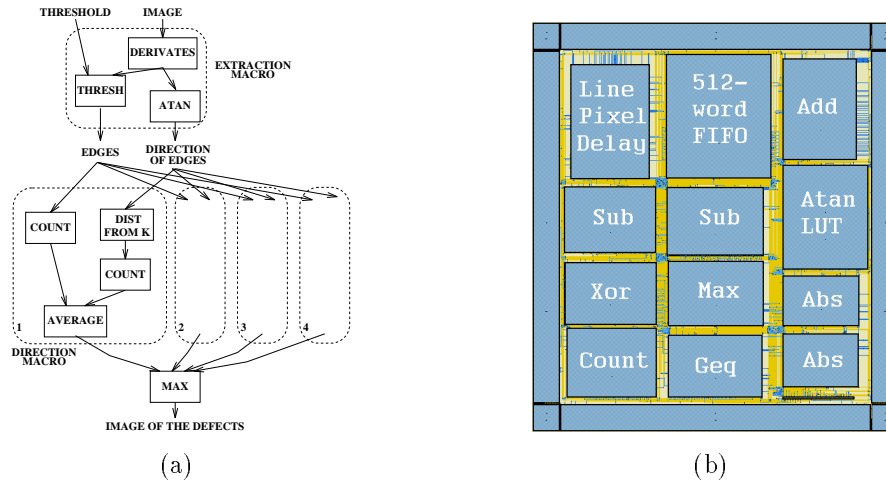


Figure 6: Data-flow graph of the defect detector (a) and floorplan of the extraction macro (b)

datapath cells using the retargeting methodology. Thanks to the 8/16-bit operator-level granularity of the FPOA the retargeting process optimizes the resulting ASICs while at the same time it minimizes the amount of simulation needed to validate it. Retargeted circuits are intended to be autonomous vision automata but they are compatible with the FPOA (and hence with the DFFC).

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] K. Öner, L. Barroso, S. Iman, J. Jeong, K. Ramamurthy, and M. Dubois. The U.S.C. Multiprocessor Testbed: A Rapid Prototyping Engine. In *Proc. of the 1995 ACM Third Int’l Symp. on FPGAs*.
- [3] 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.
- [4] J. Sérot, G. Quénot, and B. Zavidovique. Functional programming on a dataflow architecture: applications in real-time image processing. *Machine and Vision Applications*, 7(1):44–56, 1993.
- [5] R. Tessier, J. Babb, M. Dahl, S. Hanono, and A. Agarwal. The Virtual Wires Emulation System: A Gate-Efficient ASIC Prototyping Environment. In *Proc. of the 1994 ACM Second Int’l Symp. on FPGAs*.
- [6] Xilinx. *HardWire Data Book*, 1994.