Two Approaches in One for a Quick and Efficient Design of Low Area Custom Microprocessor Cores

Development of the Ims8NI Core via VHDL and Logic-Synthesis

Peter Manoilov, George Kouzmanov, Todor Stefanov, Vladislava Momtchilova, Angel Popov
Computer Systems Department, Technical University - Sofia
E-mail: anp@vmei.acad.bg

ABSTRACT - *This paper presents a methodology encapsulating the Hardware-Software Co-Design and the Top-Down Approach for developing custom microcontroller cores. Such a core (named Ims8NI), meeting specification requirements, was created in a top-down design style using VHDL models for the hardware part. A set of base algorithms was used as an input, defined by the specific domain of applications, the microcontroller should be used in. These algorithms were partitioned into tasks and the decision for distributing them between the hardware and software part was taken iteratively. An FPGA (Altera-Flex10K10) prototype of the core was implemented for the purpose of functional tests. A simple silicon chip was also implemented and it is intended to make the core public available as a soft- and/or hard-macro for further use in microprocessor based application specific integrated circuits (ASICs). The core Ims8NI is compared with some other kernels of the same class and the results are reported in this paper.*

I. INTRODUCTION. The aim of this work is to present the author's team experience in developing a quick and efficient methodology for low area custom microprocessor cores design. As a result, such a core (called Ims8NI) was created and a simple silicon chip was implemented (as a soft-and hard-macro) for further use in ASIC. They were intended to be controllers in the area of *consumer electronics*. The design process of the core had to meet the following specification requirements:

- the core should be *universal*, but efficient enough to implement a set of *base control algorithms* given by the user.
- the leading criterion (during the design process) should be a *minimal core area* on a silicon chip. This is because the core is supposed to be a part of a more complex *System-on-Chip*. The other purposes are to reach a higher manufacturing yield and to produce the chip using a cheaper transistor technology (say 2μm).
- the instruction set should be *compact* for the target applications.
- *open-core architecture* - it means the possibility of an easy attachment to the core of some peripherals (timers, counters, parallel and serial interfaces, LCD-controllers and etc.).
- *maximum speed* of the core in respect to the achieved minimal area.

The Ims8NI project was partitioned into two parts (hardware and software). In order to reduce the *Time-to-Market* both of them were developed simultaneously and concurrently. The software part included an optimal (with respect to requirements mentioned above) instruction set synthesis. The hardware part of the project included a design, investigation and area optimization of the Control Unit and the Arithmetic-

Logic Unit (ALU) of the Ims8NI core. A special attention was paid to ALU architecture, the control unit type (fixed-logic or micro-program control) and the timing.

II. DESIGN PROCESS. There are many approaches for designing LSI [3,8]. Two of them are the well known *Hardware-Software Co-Design* and *Top-Down Design*. The methodology, presented in this paper, encapsulates these two approaches. The Ims8NI core was developed using this methodology (see Fig. 1).

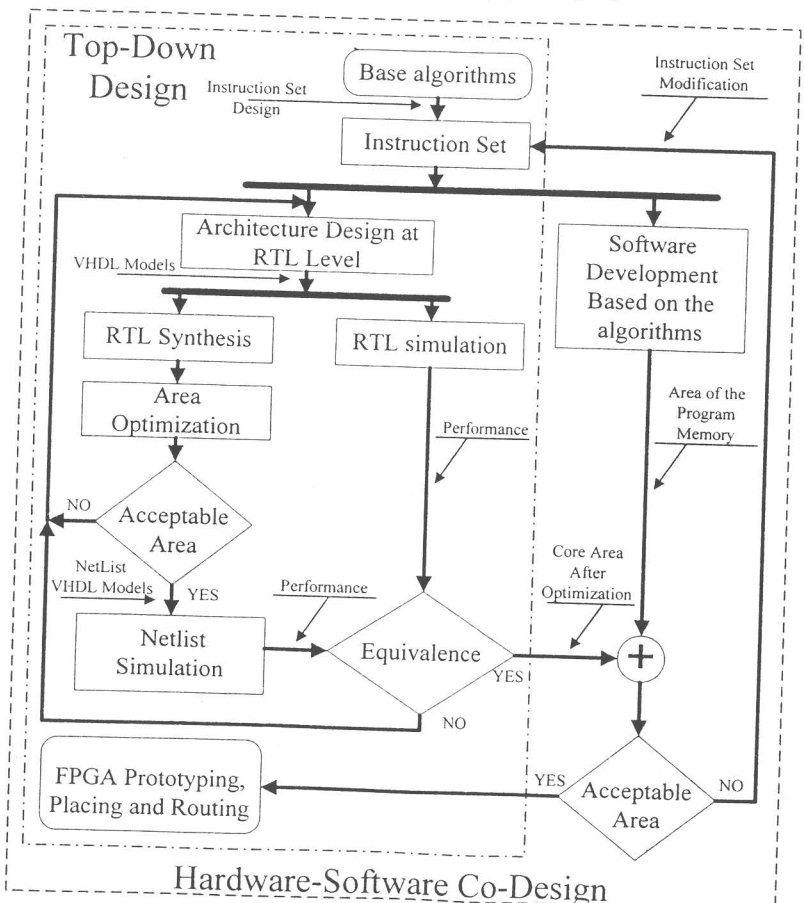


Fig.1. *Hardware-Software Co-Design* with *Top-Down Design* of the hardware part.

The main idea is to use the advantages of the Hardware-Software Co-Design (the very fast achievement of *compatibility* between the hardware and software),

combined with the possibility for an *abstract* top-down design. This combination reduces the *time-to-market* of the device. The abstract top-down design was realized on *VHDL* [2,8,9], used as an input of the *VeriBest* CAE tools [11] (including simulator, synthesizer, optimizer and etc.).

Instruction Set Synthesis. There were not any formal approaches available for an optimal (with specific constrains) instruction set synthesis. So an *heuristic iterative* methodology was used to obtain the Ims8NI instruction set (see Fig. 1). We started with a *minimal compact* instruction set synthesis. Some theoretical researches and instruction sets of existent microprocessors [4,6,10] were used for this purpose. As a result the following set was derived: ***AND, NOT, ADDition, Roll-Right-Carry, Roll-Left-Carry, CALL, RETurn, Skip-if-Zero, Skip-if-Carry, Load-Accumulator, Store-Accumulator***. Some *base algorithms* were realized using this instruction set. Their choice was very important because it defined the specific domain of application the ASIC should be used in. In our case, these base algorithms had been destined for a *sensor and button sampling* and for *displays and actuators control*. Based on the instruction set (mentioned above), the core architecture was designed and the base algorithms were programmed. Further on, the area needed for the core and program memory was calculated, so the results were a *minimal* core area, but a *huge* program memory area. This fact required *multiple instruction set* (including *core architecture*) *modifications* (see Fig. 1) to achieve the *total area* (core + memory) minimization. During this process a lot of instruction sets were obtained and the optimal one in respect to the total area is shown in Table 1.

Table 1. Instruction Set of Ims8NI Core

Mnemonic	Addressing Mode	Description
JMP	extended(addr10)	Unconditional jump to addr10
CALL	extended(addr10)	Subroutine call addr10
AND	direct(addr8)	Logical AND between the accumulator and memory
OR	direct(addr8)	Logical OR between the accumulator and memory
XOR	direct(addr8)	Logical XOR between the accumulator and memory
ADD	direct(addr8)	Add memory to accumulator
ST	direct(addr8)	Store accumulator
LD	direct(addr8)	Load accumulator with memory
LD	immediate(imm)	Load accumulator with value
RET	immediate(imm)	Return from subroutine and load accumulator with value
SETB	bit(addr5)	Set bit
CLRB	bit(addr5)	Clear bit
SB	bit(addr5)	Skip if bit set
LDPC	-	Load program counter with accumulator
RRC	-	Rotate accumulator right
RLC	-	Rotate accumulator left
SC	-	Skip if carry
SZ	-	Skip if zero
HCF	-	Halt and catch fire
RET	-	Return from interrupt or subroutine
NOP	-	No operation

Ims8NI Architecture Design. The architecture of the Ims8NI core corresponding to the instruction set given in Table 1 is depicted in Fig.2. This architecture meets all the specification requirements and has the lowest possible area. It is a *RISC-like Harvard architecture* core without an *Index Register* and *Instruction Register*. *A 3-level hardware stack* is realized, but it is *not a program accessible* one (hidden for the programmer with no PUSH and POP instructions included). Each of these particularities reduces the area without harming the core efficiency.

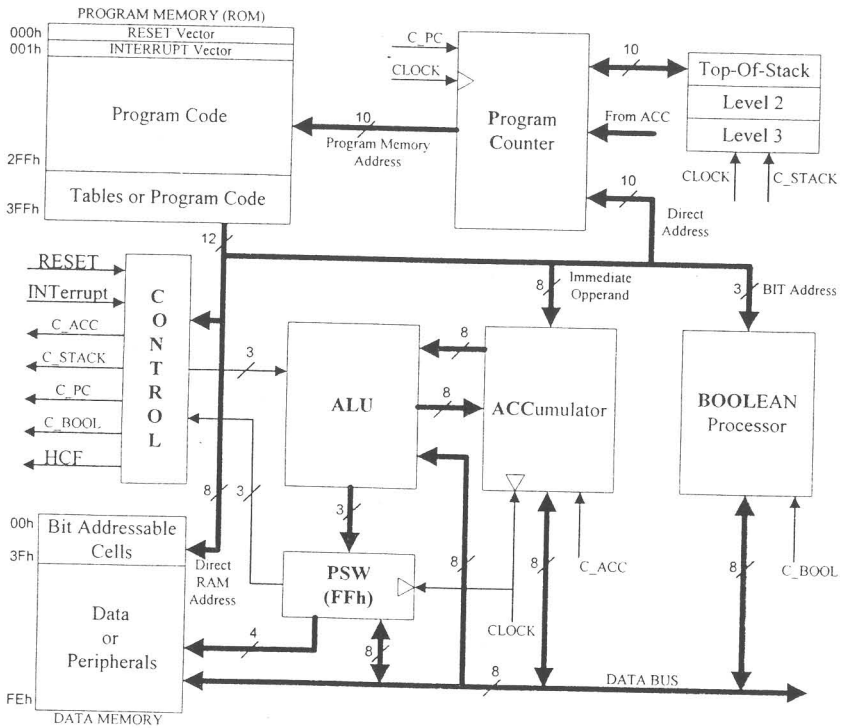


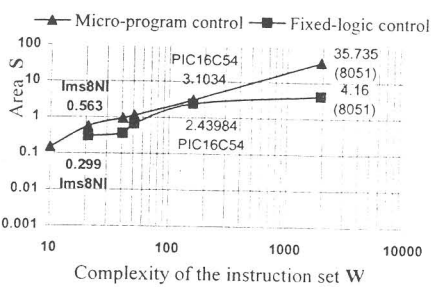
Fig.2. Ims8NI core - Block diagram

We made some analysis of the base algorithms and encountered that there were no operations over *large* data arrays. On the other hand the data memory space (256x8) is very *small*, that means this kind of operations were inconsistent. Therefore there was no need of an Index Register which fact contributed to the core area decrease (about 15%). The architecture (see Fig.2) allows the outputs of the program memory not to change their values while an instruction is running. That is why the Instruction Register was not necessary. The decision for a 3-level-deep stack was taken after an

analysis of the programs which would be running on the core. The choice of a *hardware stack* was predestined from the length differences between the program memory word (10-bit length) and data memory word (8-bit length).

After the decisions at the architecture level (regarding the Ims8NI core structure) had been taken, it was necessary to move at a lower abstract design level in order to minimize the area of the Control Unit and Arithmetic-Logic Unit [5,7].

Control Unit. The *control unit type* (fixed-logic or micro-program control) and the *optimal timing* design were the most important things in respect to the overall area of the core. The choice of the lowest area control unit was made after some investigations and a *W-S curve* was obtained for each type (W denotes a complexity of the instruction set and S - the area). The results (at 2μm technology) are shown in Fig.3. It is obvious that the micro-program control unit needs more area than the fixed-logic control unit. That is why the fixed-logic control unit was chosen and implemented in the Ims8NI core.



$$W = C\sum_{i=1}^{k} C_i \text{ ,where:}$$

C - number of the clocks in the instruction cycle;
C_i - number of the instruction cycles in the i^{th} instruction;
k - number of the instructions.

Fig.3. Area comparison between the fixed-logic control and micro-program control units

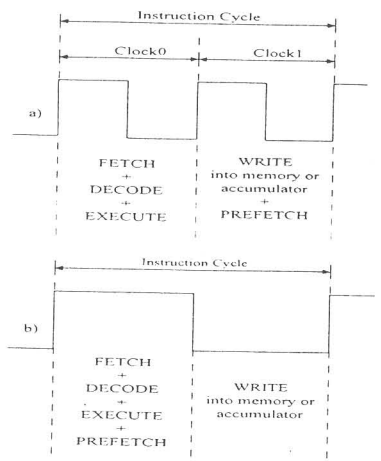


Fig.4. Two possible timing diagrams of Ims8NI

For the lack of any formal theory for an optimal timing design, the following methodology was used. The whole instruction set (see Table 1) was run trough the Ims8NI architecture and it was found that two synchro-events were necessary for each instruction. The first event should enforce the asynchronous execution of the three timing stages known as FETCH, DECODE and EXECUTE. The second should event start the results WRITING (obtained after the instruction execution) into the accumulator or into the data memory. Two rising or two falling clock edges could be

used for synchro-events as it is shown in Fig.4a. We made some researches and another timing solution was proposed (see Fig.4b). This implementation (with *one rising and one falling edge* as synchro-events) is not universal, but in our case it leads to three main advantages: 1) the timing from Fig.4b allows a *lower clock frequency* without violating the instruction execution time; 2) a *lower power consumption* due to the lower switching frequency of the flip-flops; 3) the Control Unit and the ALU can be realized as a *pure combinational* logic circuits. Therefore the Ims8NI core area could be reduced.

Arithmetic-Logic Unit (ALU). The ALU implements one arithmetical operation (Addition without input carry) and five logical operations (see Table 1). This is enough for an efficient realization of the base algorithms. A compact system of logical operations *AND-OR-XOR* was chosen. It is not a minimal basis, but it is optimal in respect to the total area (core + memory) and the base algorithms. A lot of operations in these algorithms check or change a single bit value, that is why we realized an additional *Boolean Processor* in the Ims8NI core (see Fig.2). The area of the core rose a little, but the program memory area decreased, reducing by this means the total area.

Complex arithmetical problems can also be solved satisfactorily enough with the simple set of operations that the ALU of the Ims8NI core implements. For example, the 8-node FFT conversion time (at real clock frequency 15MHz) is about 10μs. It is comparable with the conversion time of some *digital signal processors* (DSPs) like the TMS320C2x family [10].

Prototyping and Manufacturing. An FPGA (Altera - FLEX10K10) [1] prototype of the Ims8NI core was implemented for the purposes of *functional testing, adjustment and real physical parameters measuring*. Also Placing and Routing on a silicon chip using *Standard Cells* technology were performed, in order to estimate the real area of the Ims8NI core.

III. ANALYSIS OF THE EXPERIMENTAL RESULTS. We compared the Ims8NI core with the Ims8BC core (an original product of the InfoMicroSystems Ltd.) and with the Ims16C54 core (functionally equivalent to the famous PIC16C54 microcontroller). The comparison was made at 2μm technology in respect to two criteria: 1) *total area* of the silicon chip; 2) *core speed* measured in length of the instruction cycle. The three cores have a similar domain of application and belong to the same class. It means that each of them has: 1) *Harvard architecture*; 2) *single word instruction length*; 3) *one instruction cycle for each instruction*; 4) *up to 256 bytes data address space*; 5) *up to 1K words program address space*. The facts mentioned above are the main precondition for the comparative analysis given below.

Fig.5 shows the area needed for the implementation of the cores on the silicon chip. The Ims8NI core has the *lowest* total area compared with the other two cores, because some original decisions (stated in the previous section) were realized during the Ims8NI design process. We have to mark the fact that Ims8BC and Ims16C54 cores have *more powerful* instruction sets than Ims8NI core has. But it does not affect the program power of the Ims8NI core, because we found that the *most frequently* used instructions in the base algorithms are included in the Ims8NI instruction set (see Fig.7).

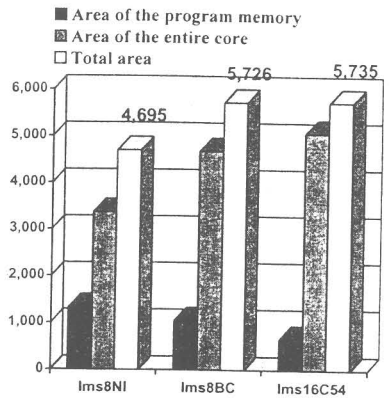


Fig.5. *Area* comparison of the three microcontrollers (at 2μm technology) measured in mm^2

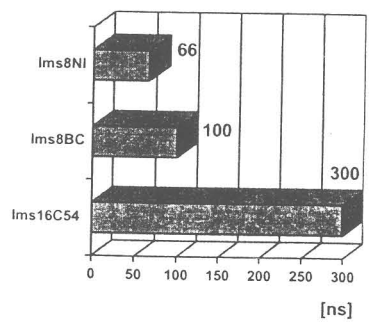


Fig.6. *Time performance* comparison of the three cores (measured in ns) at 2μm technology

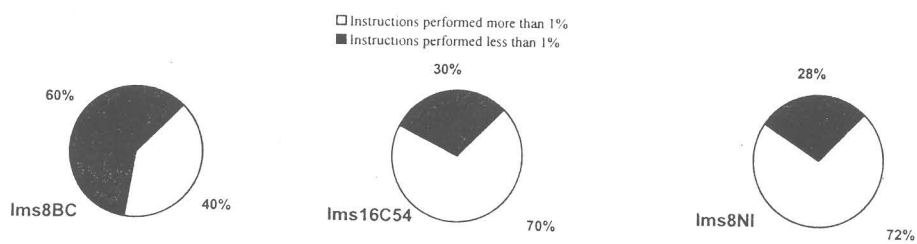


Fig.7. Frequency of the instruction usage

The *speed comparison* between the cores is shown in Fig.6. The parameter "*instruction cycle length*" gives a real information for the speed of these three cores, because it shows the *execution period* for each of their instructions. The good speed results for the Ims8NI core are based on three main things: 1) *the simple (low area) architecture*; 2) *an optimal instruction set*; 3) *the optimal timing* (see Fig.4b) which

allows instruction execution in only one instruction cycle. This cycle equals to *one clock period.*

IV. CONCLUSION. Using the results demonstrated in the previous section, it is worth noting the following advantages of the Ims8NI core: 1) *low area* of the silicon chip; 2) *cheaper transistor technology* (2μm) for manufacturing; 3) *high speed* of the core. These advantages allow the implementation of the Ims8NI core in Systems-on-Chip for consumer electronics control. This chip will have a *high manufacturing yield* and *a low cost* due to its *small size* and it is suitable for *mass production.* The Ims8NI core could also be used in some *real time control* applications due to its high speed performance.

Of course, any electronic device based on the Ims8NI core will work efficiently *only* within the base algorithms defined above. It does not exclude, however, the possibility for using the core in *larger area of applications.*

ACKNOWLEDGMENTS. The authors would like to thank the *InfoMicroSystems Ltd.* We would like to acknowledge the team leader *M. Marinov* and the ASIC designers *G. Gegov, P. Petrov, A. Trifonov* and *B. Bonev* for their support and the practical experience they shared with us.

REFERENCES

1. ALTERA - *Data Book,* Altera Corp., 1998.
2. Bhasker, J., *A Guide to VHDL Syntax,* Prentice Hall, 1995.
3. Gajski, D., Edt., *Silicon Compilation,* Addison Wesley Pbl. Comp., 1988.
4. Microchip, *PIC16C5X - EPROM-Based 8-Bit CMOS Microcontroller Series,* Microchip Technology Inc., 1993.
5. Omondi, Amos R., *Computer Arithmetic Systems. Algorithms, Architecture and Implementation.* Prentice Hall, 1992.
6. PHILIPS, *80C51-Based 8-Bit Microcontrollers,* Philips Semiconductors, 1995.
7. Sandige, R.S., *Modern Digital Design,* McGraw-Hill, New Jork, 1990.
8. Smith, D.J., *HDL Chip Design,* Doone Publications, 1997.
9. Swamy, S., A. Molin, B. Covnot, *OO-VHDL. Object-oriented extensions to VHDL,* IEEE Computer, Oct.95, pp. 18-26.
10. Texas Instruments, *TMS320C2X - User's Guide,* 1993.
11. VeriBest, *VeriBest VB98.0A for Windows NT Intel-User's Guide,* VeriBest, Inc. 1998.