IPPC: INTELLECTUAL PROPERTY PROCESSOR COMPONENT APPLIED IN EMBEDDED COMPUTER SYSTEMS

Alexandre Marques Amaral, Márcio Oliveira S. de Souza e Carlos Augusto P. da Silva Martins
alexmarques@ieee.org, marciooss@gmail.com, capsm@pucminas.br

Digital and Computational Systems Group (GSDC)
Graduate Program in Electrical Engineering
Pontifical Catholic University of Minas Gerais

ABSTRACT

Last years, embedded computer systems (ECS) have been evolving in utilization and complexity. The processor core design or redesign produce high efforts, time and costs. This paper presents the reuse of Intellectual Property Processor Component in ECSs, like SoCs and MPSoCs, and the main benefits and advantages.

1. INTRODUCTION

Last years, embedded computer systems (ECS) have been evolving a lot. Their evolution is due to new requirements demanded by current applications and also the development of new technologies, like VDSM (Very Deep Sub-Micron). Besides, there is a wide range of requirements variations that stimulates the design and development of a wide range of ECSs, to fit each of the requirements [1]. In one extreme of this range there are demands for execution of simple operations, low power consumption and performance is not critical (a System-on-Chip applied in mobile applications). In the other extreme of the range there are demands for execution of complex operations, high computational performance and power consumption constraints are not critical (a high-performance Multi-Processor- System-on-Chip).

Considering the wide range of embedded computer systems and their requirements, the processor, used as processing component, may be designed or redesigned during the project of the ECS. This includes the architecture levels (ISA and microarchitecture) and the implementation, requiring a high expertise in microprocessor design, development, implementation and test processes, increasing the project costs. Furthermore, it usually demands a very hard effort and produces a considerable increasing of the time-to-market. Therefore, the component design or redesign generates high efforts, time and costs in ECS design process [2].

Therefore, considering the importance of processor components for embedded computer systems and its design and redesign problems, this research proposed an Intellectual Property Processor Component (IPPC) with a Reduced Instruction-Set (RISC). Hence, the proposed IPPC can be reused in different ECS designs, with independency of scalability requirements and implementation technology. The IPs reuse brings lots of advantages, like reduction of design efforts, time and costs [1,2]. Therefore, the research hypotheses are these advantages of IP processor reuse in different ECS designs. Furthermore, the goals are the design and development of IPPC architecture, implementation and its reuse in different ECSs, like SoCs and MPSoCs.

2. IPPC: IP PROCESSOR COMPONENT

The IP processor component (IPPC) has general purpose architecture, with a reduced instruction-set (RISC), parameterized word bit-width, hardware instruction decoder (microcode absence), register-based data manipulation, Harvard memory model and LOAD/STORE memory access. Thus, the IPPC was designed to be the processing component of ECSs applied in applications with integer basic operations.

As described in the literature [3], the IPPC architecture was designed through the development of two architectural levels: Instruction-Set Architecture (ISA) and Microarchitecture. The following subsections present the architectural levels and the IPPC reuse.

2.1. Instruction-Set Architecture

The ISA was designed through the definition of the basic operations, the instruction formats, the kinds of operand addressing and the instructions and their operation code (op-code). The instruction-set was divided in three classes: arithmetic-logic, branches and data manipulation. Table 1 presents some instructions of the IPPC ISA, and their classes, names, assembly codes and type.

Besides the class-based classification, the instructions were classified according to their types, as presented in table 2. These types were created according to the operand addressing and the amount of addressed registers. Due to design simplification, in this research, two types of operand addressing were applied: direct and immediate [3]. Hence, the instructions can be classified in seven types: 3R, 2RI, 2R, 1RI, 1R, NRI and NR.

Table 1. Some IPPC instructions

Class	Instruction	Assembly	Type
Arithmetic-Logic	Immediate AND	ANDI Rx,Ry,kk	2RI
	Direct OR	ORR Rx,Ry,Rz	3R
	Direct NOT	NOTR Rx,Ry	2R
Branch	Unconditional branch	JUMP aaaaaa	NRI
	Branch if Carry	JMC aaaaaa	NRI
Data manipulation	Immediate Move	MOVI Rx,kkkk	1RI
	Input	INPUT Rx	1R
Other types	End Execution	Halt	NR

Table 2. IPPC Instruction Types

Type	8 Bits	8 Bits	8 Bits	8 Bits
3R	OP code	Reg. 1 (Target)	Reg. 2 (Source 1)	Const. (Source 2)
2RI	OP code	Reg. 1 (Target)	Reg. 2 (Source 1)	Const. (Source 2)
2R	OP code	Reg. 1 (Target)	Reg. 2 (Source 1)	-----
1RI	OP code	Reg. 1 (Target)	Const. (Source 1)	
1R	OP code	Reg. 1 (Target e Source 1)	-----	
NRI	OP code	Const. (Source 1)		
NR	OP code	-----		

2.2. Microarchitecture

The microarchitecture level, that executes the instructions described in ISA level, was designed and developed in a gradual and structural methodology. This means that several sub-microarchitectures were designed, and each of them is specialized in executing one class of instruction. Thus, three sub-microarchitectures were designed and developed. The first is to execute arithmetic-logic instructions; the second is to execute branch instructions and the third is to execute data manipulation instructions. Finally, these sub-microarchitectures were merged, using multiplexers and demultiplexers, to compose the complete microarchitecture (figure 1). This gradual design methodology is described in [3], and is applied throughout MIPS (reduced release) project.

The IPPC has monocycle microarchitecture, due to design simplification and also to support a further pipeline implementation, if necessary. This microarchitecture is composed of datapath and controlpath. The datapath blocks are enhanced in figure 1, while the controlpath blocks are not. The main datapath functional blocks are: Arithmetic-Logic Unit (ALU or ULA in figure 1), Register File (BR) and Data Memory (MD). And the main controlpath blocks are: Control Unit (UC), Program Counter (PC and Som) and Instruction Memory (MP). The block diagram of IPPC microarchitecture is presented in figure 1.

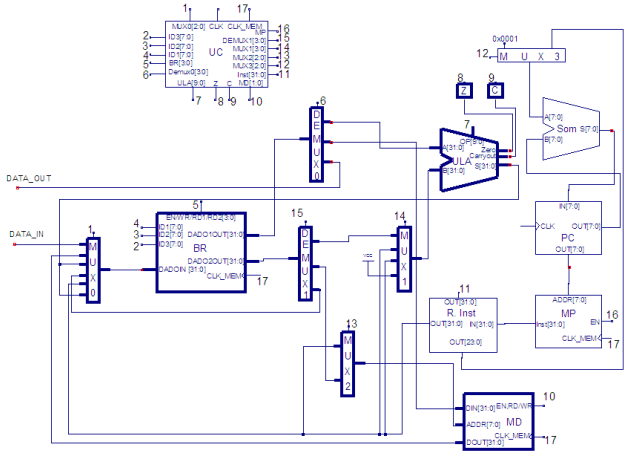

Figure 1. Block diagram of IPPC microarchitecture.

The Arithmetic-Logic Unit (ALU) was designed and developed as a parameterized word-length ripple carry structure. Although ripple carries are not applied anymore, this simplification is due to the ALU design was not the main goal of this research. Among the future works, is the development of a more efficient ALU.

The register file is the highest level of data storage. It is able to execute two reads simultaneously. The lowest level of data storage is the Data Memory (MD). Both levels also have parameterized word-lengths. Others storage blocks are Zero (Z) and Carry (C). These blocks are to store the state of ALU's results, storing high logic level whenever these results has a carry or is equal to zero. The others datapath blocks are multiplexers and demultiplexers to correctly connect functional blocks.

As presented in figure 1, the instruction control and decoding are centralized, performed by a unique functional block, the Control Unit (UC). It generates the control and synchronization signals. The centralized implementation is due to design simplification, and it also targets a further pipeline implementation [3]. The Control Unit was designed as a Finite State Machine (FSM), as presented in figure 2. This FSM has the usual states presented in the literature [3]: Instruction Fetch (IFETCH), Decode (DFETCH), Execute (EXE), Write Back (WB) and Idle (IDLE). The datapath operation of these states is similar to the operation presented in [3].

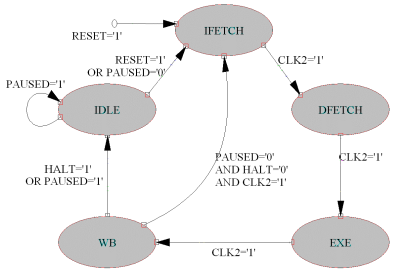

Figure 2. IPPC Control Unit Finite State Machine.

In IFETCH state, the next instruction to be executed is fetched from the Instruction Memory (MP), the Instruction Register (R. Inst.) is loaded, that transmits it to the UC. In DFETCH state, the UC generates the addresses ID1 and ID2 for the registers to be read. In EXE state, the multiplexers and demultiplexers are enabled and their selection inputs are asserted, the ALU and MD are enabled depending on the executed instruction. In WB state, the UC enables the BR or MD and feeds their write address inputs, besides the clock synchronization generation. In IDLE state, the execution is finished or it is waiting an input signal to continue the execution from the instruction it was interrupted.

2.3. IPPC reuse in embedded computer systems

The IPPC was described in VHDL (Very High Speed Integrated Circuits Hardware Description Language) in a structural manner. The description developed represents its microarchitecture with each block of figure 1 described as a component. Furthermore, this description supports synthesis and implementation in different platforms, including FPGAs. This enables the IP processor component to be used and reused in different

systemic projects. In addition, the parameterized feature of the architecture levels (ISA and microarchitecture) is supported by the developed VHDL description. This reduces the efforts for the IP fitting into the system.

After the verification of the IPPC functionalities and behaviors in stand-alone executions [4], the developed VHDL code was wrapped into VHDL descriptions of SoCs and MPSoCs. The developed Systems-on-Chip (SoCs) and Multi-Processor-Systems-on-Chip (MPSoCs) are application-specific integrated systems. Two different applications were chosen, which are: a digital controller and a digital image filter. The controller processing core is called Reconfigurable Electronic Control Unit (RECU) and was the second place in Xilinx Student Contest 2005. The filter processing core is called Image Convolution Circuit (ICC) and its architecture and its implementation was published in [5,6]. These cores were integrated with instances of the IPPC to build integrated systems. Figure 3 presents a parameterized block diagram that represents the organization of developed SoCs and MPSoCs.

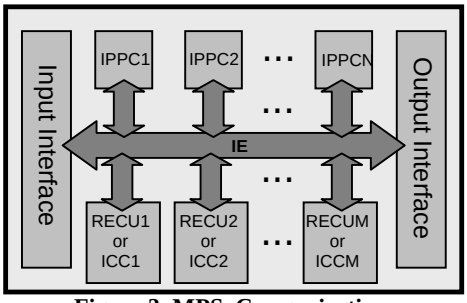

Figure 3. MPSoC organization.

The SoC organization is composed of a unique instance of the IPPC and one or more instances of the application-specific core (RECU). The MPSoC organization has N IPPC and M application-specific core instances (ICC). This possibility of varying the number of IPPC and application-specific core instances makes the organization parameterized, supporting implementation of either a SoC or an MPSoC. The parameters are defined by the system developer, considering tradeoffs between the functional and non-functional application requirements and the implementation costs.

The IPPC instances can work as a data source for the application-specific processing core or even parallel processing core executing an application-specific software. All the implementations developed during this research have their IPPC instances as a data source for the application-specific cores.

Moreover, both organizations have additional blocks of interconnection logic, called *Interconnection Element* (IE) *Input Interface* and *Output Interface*. These blocks can be composed of multiplexers and demultiplexers, a crossbar switch, or being a complete Network-on-a-Chip (NoC) [7]. Using multiplexers and demultiplexers the interface blocks require less silicon resources, although their connections may be time-multiplexed. Otherwise, using a crossbar switch, the connections may be full-time, although the interface blocks require more silicon resources. The interface blocks are implemented as multiplexers and demultiplexers. However, a work in progress is to implement them using the crossbar switch proposed in [8].

Different combinations of IPPC and application-specific core instances, composing VHDL descriptions of different SoCs and MPSoCs were synthesized, mapped, placed and routed, targeting FPGAs. However, only results of the RECU mono-processed SoCs and the ICC multi-processed SoCs were presented in this paper. The target FPGAs are a Spartan-3 XC3S200-4ft256 and a Virtex-II XC2V1500-5ff896, both produced by Xilinx. These devices were chosen since they are soldered on available development boards in Digital and Computational Systems Laboratory (LSDC).

3. EXPERIMENTAL RESULTS

This section presents the results obtained from the implemented IPPC instance applied in the developed SoC and MPSoC implementations. These are results of performance, FPGA resources and power consumption.

Figure 4 presents the response times that a 512x512 image takes to be filtered, with different convolution kernels, executed in ICC-based implementations. Each implementation's maximum operation frequency is presented inside parenthesis. These maximum frequencies were obtained from the Xilinx ISE Timing Analyzer tool.

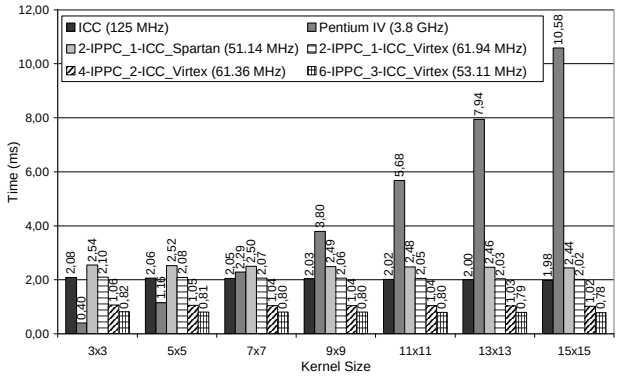

Figure 4. Response times of image convolution in MPSoCs.

Analyzing figure 4, it is noticed that although Pentium IV performs the smaller kernel filtering operations with excellent performance, its sequential execution times increase a lot with the kernel size increasing. Otherwise, the execution using ICC-based systems does not take more pieces of time whenever the kernel size increases. This is due to the parallel architecture and implementation of the ICC core [5,6]. The ICC supports spatial and temporal parallelism in each kernel iteration processing. Despite, figure 4 presents a little decreasing of the ICC-based systems response times with the increasing of the kernel size. This is due to the absence of image border processing. In addition, the response times is decreased with the number of ICC and IPPC instances increasing and also with the upgrading to a more evolved target FPGA family. The former is due to the increasing of

parallelism exploration and the speed-ups are approximately proportional to the increasing of ICC instances. The latter is due to more efficient FPGA resources, providing systems more optimized and with higher maximum operation frequency.

Table 3 presents the percentage of occupied FPGA resources for different implemented systems. These systems differ in the number of IPPC instances, the type and the number of application-specific core instances and the target FPGA. Analyzing table 3, it is observed that RECU-based systems fit both target FPGAs. Otherwise, some ICC-based systems do not fit the Spartan-3 device. This occurs with 2-IPPC_1-ICC_Spartan system, which overmapped the amount of slices and block RAMs. This implies in overmapping of ICC-based systems with more than two IPPCs. Table 3 also presents the high amount of resources of the Virtex-II FPGA. This is showed by the configuration of an MPSoC with six IPPC and three ICC instances, and the verification of unused resources.

Table 3. Percentage of occupied FPGA resources.

System	Regs	Slices	LUT	I/Os	BRAM
1-IPPC_1-RECU_Spartan	14.6%	57.8%	37.8%	42.8%	16.7%
1-IPPC_1-RECU_Virtex	3.7%	14.5%	9.5%	14.0%	4.2%
2-IPPC_1-RECU_Spartan	27.9%	71.7%	54.5%	43.4%	33.3%
2-IPPC_1-RECU_Virtex	7.0%	27.5%	17.7%	14.2%	8.3%
2-IPPC_1-ICC_Spartan	31.3%	115.6%	74.4%	43.4%	108.3%
2-IPPC_1-ICC_Virtex	7.8%	29.2%	18.6%	14.2%	27.1%
4-IPPC_2-ICC_Virtex	15.6%	59.5%	38.3%	15.3%	54.2%
6-IPPC_3-ICC_Virtex	23.4%	88.8%	56.6%	16.5%	81.3%

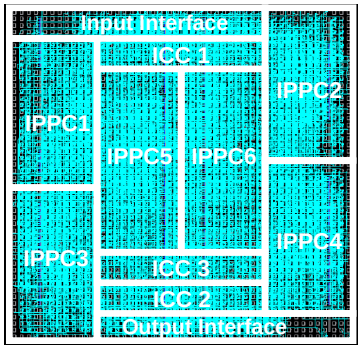

Figure 5. Implemented 6-IPPC_3-ICC system.

Figure 5 presents the 6-IPPC_3-ICC system implemented on Virtex-II FPGA. This figure was generated by the Xilinx FPGA Editor and has details of where the logic of each component and the routed signals are allocated. Applying area constraints during the implementation process enables the component identification among the whole system, as presented in figure 5. The area constraints were defined considering communication efficiency, attempting delay reductions.

According to the Xilinx Xpower tool estimation, the implementations using the Spartan-3 FPGA dissipate about 37 milliwatts, while those using the Virtex-II dissipate about 359 milliwatts. This verifies that for an application that requires low power dissipation, devices with less resources and lower cost may be worthier.

Considering the results and regardless of performance and scalability constraints, it is recommended to implement these SoCs using the Spartan-3 device, reducing costs and power consumption. Furthermore, the IPPC reuse in different SoC and MPSoC provides reduction of the design efforts, time and costs, since the developed systems were designed, with IPPC, ICC and RECU reuse, in three weeks using one FPGAs.

4. CONCLUSIONS

The presented results verified the benefits and advantages of the proposed IPPC reused in systemic embedded computer systems (ECS). This is reached by utilization of IP reuse methodologies [2]. Therefore, the goals were reached and the hypotheses were proved.

Furthermore, the main contribution of this work is the design and development of an IP Processor Component (IPPC) with independency of implementation technology, since it is described in VHDL. Besides, the IPPC reuse in embedded computer systems (SoCs and MPSoCs) is also a contribution. Therefore, reusing the IPPC in systemic designs of ECSs may generate benefits for the academic, industrial and or commercial communities.

This research was developed simultaneously with the graduation conclusion project in Electronics and Telecommunication Engineering [4].

Some future works are: development of a more efficient ALU for IPPC; integration of the reconfigurable crossbar switch [8] as the communication interfaces; design and development of a Reconfigurable IP Processor Component (RIPPC) and its integration in the developed SoCs and MPSoCs.

5. ACKNOWLEDGMENTS

This work was developed with the CNPq (National Counsel of Technological and Scientific Development - Brazil) support.

6. REFERENCES

[1] W. Wolf, “Embedded Computer Architectures in the MPSoC Age”. *32th International Symposium on Computer Architecture*, IEEE Computer Society. Madison, 2005.

[2] S. Sarkar, G. Subash Chandar, S. Shinde. “Effective IP Reuse for High Quality SoC Design”. *IEEE International SoC Conference.* pp. 217-224. 2005.

[3] D.A. Patterson, J. L. Hennessy, Computer Organization and Design: the Hardware/Software Interface. 3rd edition Morgan Kaufmann Publishers, San Francisco, 2004.

[4] A.M. Amaral, R2NPC (Reconfigurable RISC Network Processor Core): Design and Implementation of the R2NP Network Processor Core. Graduation Conclusion Project Report in Electronics and Telecommunication Engineering. Pontifical Catholic University of Minas Gerais. 2006 (in portuguese).

[5] A.M. Amaral, C.A.P.S. Martins, Image Convolution Circuit: Parallel and Parameterized Architecture and FPGA Implementation. *III Student Forum on Microelectronics*, SBC, Florianópolis, 2005.

[6] A.M. Amaral, M.B.Carvalho, C.A.P.S. Martins, Parallel and Parameterized Architecture for Image Convolution. *Workshop on High Performance Computational Systems*, SBC, O. Preto, pp. 195-198, 2006 (in portuguese).

[7] C.A. Zeferino, M.E. Kreutz, L. Carro, A.A. Susin, A Study on Communication Issues for Systems-on-Chip. *15th Symposium on integrated Circuits and Systems Design*. (SBCCI). IEEE Computer Society, Washington, pp. 121-126. 2002.

[8] H.C. Freitas, M.B. Carvalho, A.M. Amaral et al. Reconfigurable Crossbar Switch Architecture for Network Processors. *IEEE International Symposium on Circuits and Systems* (ISCAS), IEEE CAS Society, Kos, pp. 42-45, 2006.