Instruction Customization: A Challenge in ASIP Realization

Deepti Shrimal
Research Scholar
Department of Computer Science
Mohan Lal Sukhadia University

Manoj Kumar Jain, Ph.D
Associate Professor
Department of Computer Science
Mohan Lal Sukhadia University

ABSTRACT

An Application Specific Instruction set Processors (ASIP) or alternatively known as customized processor is a processor designed for a particular application or for a set of applications. Earlier surveys show that though a significant research has been done for this most promising processor design technology, still approaches used in them are lacking in methodologies to define processor configuration based on the requirements of the applications. There are number of approaches claiming to design and synthesize ASIPs but they are facing many challenges. This paper is an attempt to find major challenges faced by them as well as the current state of these promising techniques adopted by the industry. This paper also analyzed their effort to know really what they could have achieved so far and identified what should be done to make these techniques successful. If some limitations can be removed soon, these techniques are going to expand in an explosive manner.

General Terms

Architecture, Design techniques, Instruction set customization, Processors.

Keywords

Application Specific Instruction set Processor (ASIP), Custom Processor, Embedded System, Micro-architecture, Simulation, and Synthesis.

1. INTRODUCTION

An Application Specific Instruction set Processor or ASIP is generally designed for a set of applications for a particular domain. Special characteristics of these applications guide us about the target processor. Earlier survey in [1] shows that significant research has been done for ASIP synthesis and more research is in progress. With the advent of soc's, the need to design a synthesizable processor or programmable core has become more popular among embedded system designers now-a-days. These processors offer several advantage such as reduce cost, high flexibility and high performance over the ASIC (Application Specific Integrated Circuit). The main steps in designing ASIP involves application analysis, design space exploration, instruction set generation, software tool set generation (including tools for code generation, operating system, debugger etc) and hardware synthesis which provides synthesizable processor descriptions using which ASIP chip is fabricated.

Since the base ISA of these processors can be extended and customized, they are becoming more popular in academia and industry. The base instruction set of these processors can be extended with custom instructions with the help of tightly integrated functional units [2] as shown in Figure 1.

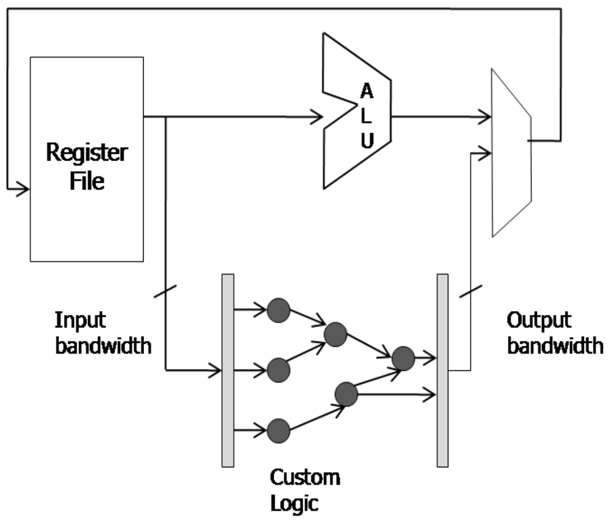

Fig 1: Base ISA extension using custom logic

There are many commercial processors available in the market such as Tensilica Xtensa [3], Altera Nios II [4] and Xilinx MicroBlaze [5] which support extensibility using such type of functional units. The instruction set and micro-architectural parameters are changed for different application or for a set of applications based on these functional units for target architecture easily. The primary goal of IS customization in such processors is to design an instruction set for ASIP that satisfies the hard constraints and also optimizes the performance of micro-architectural parameters like register file size, cache memory, functional units bandwidth etc. Figure 2 shows the overall process to add custom instruction into ASIP.

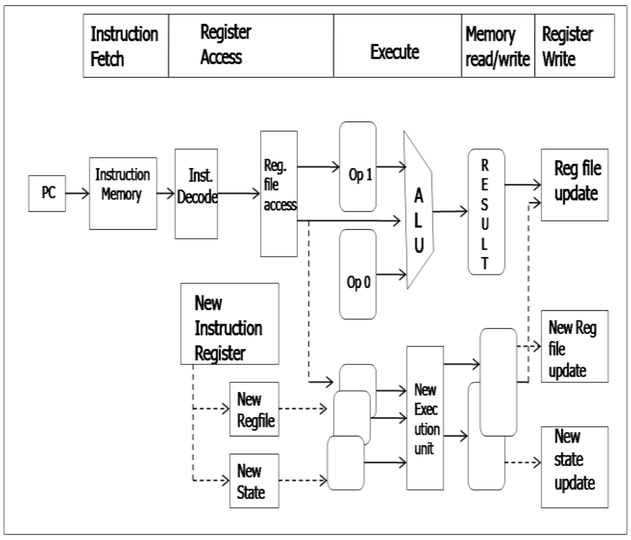

Fig 2: Custom instruction integration in ASIP

This identification of custom instruction for instruction set extension can be classified as complete customization or partial customization depending on whether a new processor is generated or existing processor is going to be modified. This extension of an instruction set normally considered for identify the degree of human efforts to be required and then accordingly automatic extension or manual extension is done. Xtensa, Nios II and MicroBlaze are examples of processors that allow designers to customize instruction set of the processor automatically rather than manual extension.

2. INSTRUCTION SET CUSTOMIZATION

There are numerous techniques proposed by many researchers for automated synthesis of custom instructions of customizable processors [6] but in last decade, high level descriptions of application has been received much more attention than others. This technique uses compiler infrastructures [7] to find source level DFG's and then data flow sub-graphs are evaluated for custom instruction candidates followed by synthesis of customized hardware and software components which is depicted in Figure 3.

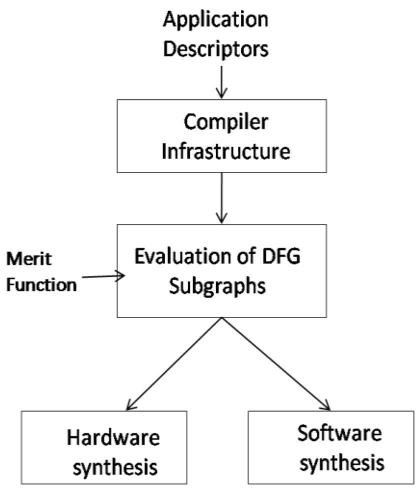

Fig 3: synthesis of the customized hardware and software components

Each custom instruction candidate usually comprises the computation of a frequently executed sub graph of the application DFG's and is usually implemented as CFU in the data path of the processor. To find optimal set of sub graphs from this DFG's involves two different subtasks: (i) identification of custom instruction using which all candidate sub graphs from the application DFG's are generated (ii) selection of custom instruction which is used to evaluate the performance of each candidate and then optimal candidate is chosen.

There are several approaches used by many researchers for instruction set identification and then custom processor has been designed to meet user requirements. A design proposed in [8] applied to the base processor of Tensilica Xtensa and MicroBlaze processor to identify optimal instruction set from a give high level descriptions. This approach was based on ILP model and relaxes designer from the data bandwidth which is usually limited by FSL channels. In the same context [9] used SFU (specialized functional units) in customized processor to support extension in parallel fashion.

An approach described in [10] is a mix of subgraph enumeration and ILP formulation to generate fast custom instruction synthesis. Apart from this, there are some techniques/ algorithms proposed by some researchers that only focused convex maximal subgraph for the same purpose. An algorithm used in [11] accelerates the entire design space optimally by considering all legal patterns of subgraph which satisfies few architectural constraints. Similarly an approach [12] also generates custom instructions without imposing any restrictions on the number of input and output operands on enumerated convex subgraph. In the similar context [13] considered only independent graph for hardware – software partitioning at instruction level within basic block and updated that block to speed up the performance of the processor. All these approaches discussed above are based on graph enumeration and their overall description for instruction set extension is depicted in the Figure 4.

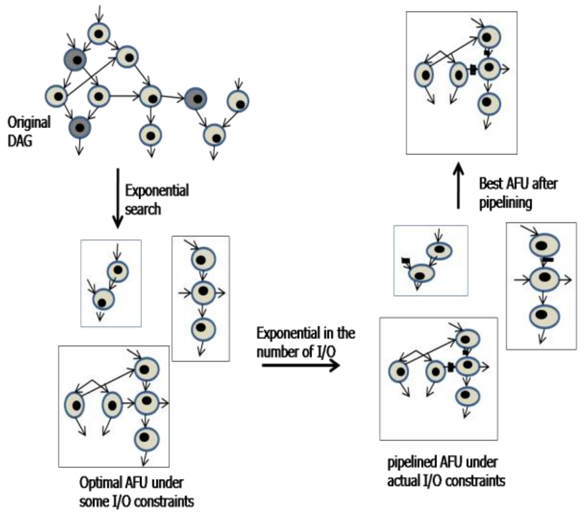
Original DAG
Exponential search
Best AFU after pipelining
Exponential in the number of I/O
Optimal AFU under some I/O constraints
pipelined AFU under actual I/O constraints

Fig 4: ISE identification approaches

The following section of this paper is discussing about Xilinx MicroBlaze, Altera Nios II and Tensilica Xtensa processors which are RISC based and support automatic extension of instructions in both partial and complete manner using maximal subgraph to speed up their process of designing custom processor to achieve better performance. The configuration based comparison among these custom processors is shown in Table 1. Both Altera and Xilinx products require the user to define custom instructions using an HDL whereas Tensilica allows developers to produce custom instructions in one of two ways. First, using Tensilica Instruction Extension (TIE) language and second using Xpress compiler.

Table 1: Comparison of custom processors

	MicroBlaze	**Nios II**	**Xtensa**
ASIC/FPGA Tech.	Virtex	Stratix	0.13 micron
ISA	32 bit RISC	32 bit RISC	32 bit RISC
Cache Memory (I/D)	Up to 64 KB	Up to 64 KB	Up to 64 KB
Floating Point Unit (Optional)	IEEE 754	IEEE 754	IEEE754
Pipeline	3 Stages	6 Stages	5 Stages
Custom Instructions	None	Up to 256 Instructions	Unlimited
Register File Size	32	32	32 or 64
Implementation	FPGA	FPGA	FPGA/ASIC

3. MICROBLAZE PROCESSOR

Xilinx MicroBlaze processor has 3 stage pipelines with variable length instruction latencies ranging from one to three cycles. It is capable of including maximum of 8 inputs and 8 outputs FSL [14]. The FSL is a fast and dedicated interface which is used to implement customization of instructions. MicroBlaze has its own ISA which consists of instructions for blocking and non-blocking reads and writes to the FSL. Two instructions put/get are used for sending and receiving data in blocking mode where as nput/nget is used in non-blocking mode.

MicroBlaze claims to combine all the flexibility advantages of SCP. Generally, there are two ways to integrate a customized IP core into a MicroBlaze-based embedded soft processor system. One way is to connect the IP on the On-chip Peripheral Bus (OPB). The OPB is part of the IBM Core ConnectTM on-chip bus standard. The second way is to connect the user IP to the MicroBlaze dedicated Fast Simplex Link (FSL) bus system to move operands to the CPU.

MicroBlaze provides flexibility to the designer during the design process to configure the processor according to their needs and quickly integrate the processor within any FPGA [15] as depicted in the basic block architecture in Figure 5. Xilinx platform studio is used as a tool to define set of libraries and customizes hardware divider and barrel shifter rather than performing custom logic in software part.

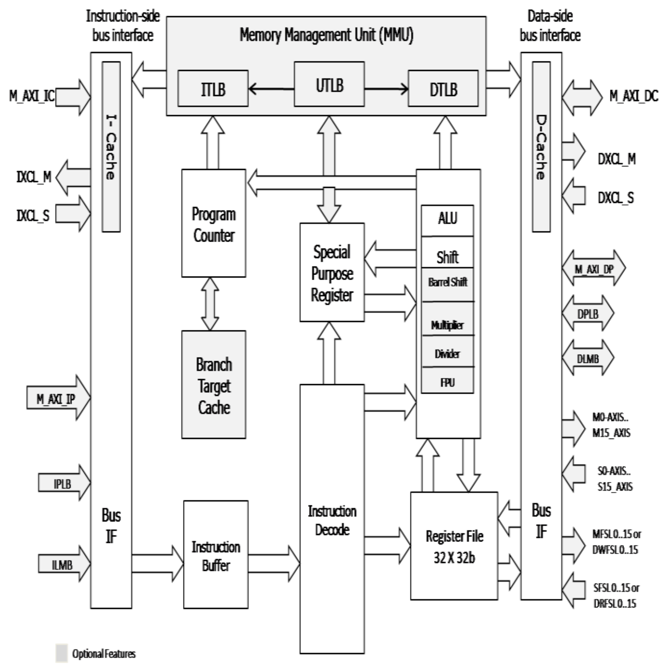

Fig 5: Core processor of MicroBlaze

Although the MicroBlaze processor has a three-stage pipeline, during the Execute phase of the pipeline, instructions have different latencies which are the main disadvantage of it. Another potential drawback of the MicroBlaze processor is the non availability of floating point instructions therefore software routines are required to perform these operations.

4. NIOS II SOFT CORE PROCESSOR

The *Nios II Embedded processor* is the most versatile and popular configurable soft processor that supports all Altera Soc's, FPGA and hard copy ASIC devices and widely used in the design of embedded systems and digital signal processing to system control. Altera Nios II processor support 32 bit instruction set, 32 general purpose registers, 32x32 multiply and divide operations , 64 bit dedicated instructions and 128 bit product of multiplication. Nios II can be implemented in the programmable logic [16] and memory blocks of Altera FPGA's. The multi master Avalon switch fabric allows it's user to accelerate and optimize their design. Nios II offers the possibility to integrate 5 custom instructions using 3 main registers (dataa[0..31], datab[0..31]) as inputs and (result[0..31]) as output as shown in Figure 6. The following steps are to be performed to add custom instructions in Nios II processor

1. Design HDL description of custom instruction
2. Verify the design using VHDL/ Verilog
3. Update the application code with custom instruction opcode
4. Integrate custom instruction in Nios II core
5. Compile custom instruction opcode
6. Generate custom Nios II bit stream.

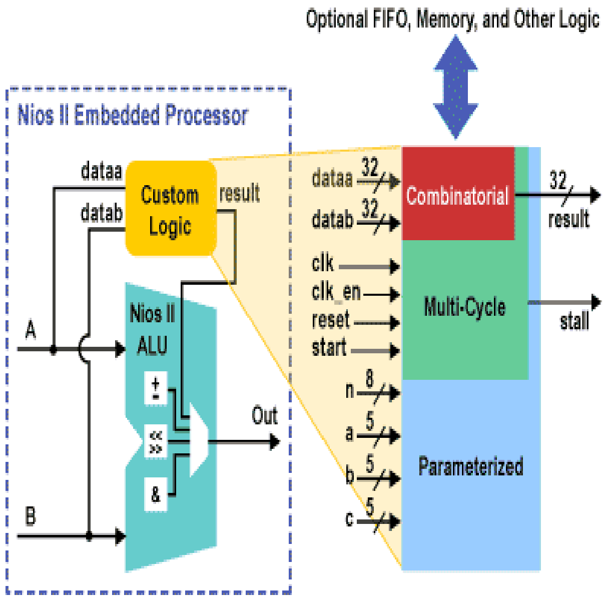

Fig 6: Custom instruction logic of Nios II

The soft core processor of Nios II is designed for 5 stage pipelines with separate data and instruction Harvard structure. Users can customize up to 256 instructions according to their needs in Nios processor core. Although the size of the register file is configurable, only 32 registers are visible to programs at any given time and instruction set of Altera Nios II supports only the basic arithmetic and logic instruction which is one of the main limitations of it.

5. TENSILICA XTENSA PROCESSOR

Tensilica Xtensa has five stage pipelines, up to 64 general purpose physical registers, 6 special purpose registers and 80 base instructions including 16 bit/24 bit RISC instruction encoding with modeless switching. It allows instruction set customization using processor configuration and processor extension. In processor configuration, usually suitable combinations of predefined set of existing architectural features such as size of register files, ALU units, single cycle or multi cycle multiplier [17] etc are selected for customization where in processor extension designer allows to add their own functionality to the processor using TIE language which is independent of the processor pipeline. Xtensa processor generator [18] uses the input from the designer to configure software tools, ISS and hardware HDL shown in Figure 7. The output produced by generator is used for simulation, debugging, profiling, synthesis, placement and routing, verification etc.

Since there are differences in the instruction set of one customized core to another core, it is more difficult for third party software and tool providers to support Tensilica cores as compared to fixed architecture cores and hence is less likely to be used across a range of applications.

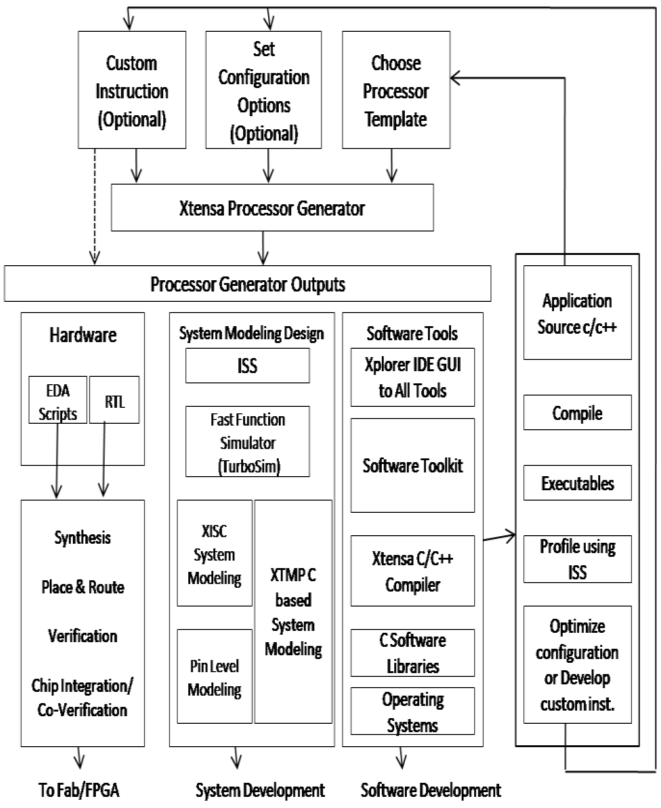

Fig 7: Design Flow of Xtensa Processor

6. CONCLUSION

Many companies till today provided their customizable processor to add flexibility to the base processor by adding new instructions or hardware description language but they support only few micro architectural variations. Also, these processors suffer from communication overhead because of use of interfaces with specialized hardware. Particularly we have analyzed processors namely, Xtensa, MicroBlaze, and Nios II and observed that these processors are far behind the expected ASIP processor. Though, they have provided development environment with simulation capabilities to the designer to give their design features as input but all the approaches are lacking in defining processor configuration based on the requirements of the applications. Furthermore they have not yet getting energy benefits from such ISE customization and if these types of problems remain then ASIPs are going to boom in the future.

7. REFERENCES

[1] Jain M.K., Balakrisnana M. and Kumar A. 2001. ASIP Design and Methodologies: Survey and Issues. Proceedings of IEEE. VLSI. pp. 76-81.

[2] Ye Z. A., Moshovos A., Hauck S., and Banerjee P. 2000. CHIMAERA: High- Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit. In Proc. 27th Annual International Symposium on Computer Architecture. pp. 225-235.

[3] Tensilica Inc. homepage, Available: http://www.tensilica.com

[4] Altera Corp. homepage, Available: http://www.altera.com.

[5] Xilinx Inc. homepage, Available: http://www.xilinx.com.

[6] Cong J., Fan Y., Han G., and Zhang Z. 2004. Application- specific instruction generation for configurable processor architectures. In *Proc. FPGA*, Monterey. CA. pp 183-189.

[7] Atasu K., Pozzi L., and Ienne P. 2003. Automatic application- specific instruction-set extensions under microarchitectural constraints. In *Proc.40th DAC*. pp 256-261.

[8] Atasu K., Mencer O. and Luk W. 2008. Fast Custom Instruction Identification by Convex Subgraph enumeration. pp 1-6.

[9] Chen L, Tarango J., Mitra T. and Brisk P. 2013. A Just-in-Time Customizable Processor. IEEE. pp 524-531.

[10] Chen X., Maskell D. L., and Sun Y. 2007. Fast identification of custom instructions for extensible processors. IEEE Trans. Computer-Aided Des (CAD) Integrated Circuits. pp 359-368.

[11] Pothineni N., Kumar A. and Paul K. 2008. Exhaustive Enumeration of Legal Custom Instructions for Extensible Processors. 21st International Conference on VLSI Design pp 261-266.

[12] Pozzi L., Atasu K., and Ienne P. 2006. Exact and approximate algorithms for the extension of embedded processor instruction sets. IEEE Trans Computer Aided Des. pp 1209-1229

[13] Biswas P., Banerjee S., Dutt N., Ienne P and Pozzi L. 2006. Performance and energy benefits of instruction set extension in an FPGA soft core. VLSI Design. pp 651-656.

[14] Lazányi J. 2005. Instruction Set Extension Using MicroBlaze processor. FPL. IEEE. pp 729-730.

[15] Hamblen J. O. 2006. Using System-on-a-Programmable-Chip Technology to Design Embedded System. IJCA. pp 142-152.

[16] Brown S. and Rose J. 1996. FPGA and CPLD Architectures: A tutorial. IEEE, pp. 42-46

[17] Peddersen J. 2005. Rapid Embedded Hardware/Software System Generation. Proceedings of the 18th International Conference on VLSI Design. pp 111-116.

[18] Cheung N. 2003. Rapid Configuration & Instruction Selection for an ASIP: A Case Study. IEEE. pp 802-807.