A System Level Exploration Platform and Methodology for Network Applications Based on Configurable Processors

D. Quinn[1], B. Lavigueur[1],G. Bois[1], M. Aboulhamid[2]

[1]École Polytechnique de Montréal, QC, Canada
{Quinn, Lavigueur, Bois} @grm.polymtl.ca

[2]Université de Montréal, QC, Canada
aboulham@iro.umontreal.ca

Abstract

A recent practice in the development of programmable SoC is the integration of configurable processors, since they offer an interesting compromise between purely software and hardware solutions. This paper proposes an adjustment to the current codesign approach to integrate this opportunity at the partitioning level. Since configurable processors seem to be an interesting option for NPU designs, we integrated into a system level exploration platform the support of an Xtensa processor for more investigation. As case studies, this paper illustrates the methodology for two realistic network-processing applications, for which interesting performances are obtained.

1 Introduction

Up until the late 1990's, most packet processing devices were realized by software. Nearly all of the network tasks such as packet forwarding, filtering, queue management and routing table maintaining were performed by a central processing unit. The advantage of such architecture was the ease to adapt products to new protocols and rapid market changes. But recently, the bandwidth explosion and the continuing growing demand in services have overcome the potential of this conventional solution. On the other hand, it has now been established that programmable system-on-a-chip (SoC) can meet these new challenges. A NPU, that can be defined as a software programmable SoC with architectural features and/or special circuitry for packet processing at wire speed, appeared as the solution.

The range of commercial NPUs comprises all the spectrum of architectures, from fully software programmable to almost completely hardwired [1]. The choice of an architecture may depend on the intended data rate, the targeted application, time-to-market constraints, etc. Tools that efficiency assist and guide architectural designers in the exploration of this broad solution space become a necessity to meet the short development time constraints. Many different tools, both academic and industrial, tackle the task of SoC design automation and try to ease the architectural exploration phase. Teepee is an architecture development framework created by the Mescal group. It can be used to describe an architecture using a formal language and to automatically generate different parts of the desired platform design [2]. Roses, by the TIMA group, is a design environment used to implement a component based methodology to develop SoCs. Roses main goal is to speed up the refinement of the platform by generating the required hardware wrappers and software APIs [3]. StepNP [4], a system-level exploration platform for network processors, is another tool that can help answer this challenge by allowing quick system level exploration of different designs. Finally, commercial tools like CoWare N2C or Cadence VCC can be used to create a platform based design. For this work, three reasons have motivated the choice of StepNP: 1) the architecture under development is easily modified by adding or changing some of its components, 2) it provides the required infrastructure to perform simulation, debugging and performance analysis of the architecture 3) and it is open source.

When developing an NP platform, the exploration phase coarsely consists in two steps. During the first step, two choices must be done: 1) the right processors offering the best-suited instruction set for the targeted application and 2) the interconnections (point-to-point bus or network-on-a-chip NoC) that will satisfy the required bandwidth. The second step, named *partitioning*, is usually required when specification requirements are not met. It consists of replacing time-consuming software parts by specialized accelerating hardwire engines. The tradeoffs are the software flexibility versus the hardware performance.

But in the last few years, one more alternative has appeared to system designers. Between application specific processors (ASP) and purely hardwired solutions, the integration of embedded configurable processors has changed the previous partitioning process. By adding to them specialized instructions to match specific tasks, configurable processors can produce interesting gains similar to hardwired engines. Taking into consideration the fact that bandwidth and latency are key aspects in most of today NPUs, they become an interesting solution for such SoC, avoiding extra communications between processors and coprocessors.

1530-1591/04 $20.00 (c) 2004 IEEE

Therefore, we incorporated into StepNP the necessary logic to support the Xtensa configurable processor provided by Tensilica [5]. This enables the exploration of a wider range of instruction-set architectures. We also propose a modification to the current codesign approach to reflect this new opportunity at the partitioning level. We apply the resulting methodology first for the design of an IPv4 compliant device, and then for the same application supporting encryption. With one processing element, we accelerated these applications significantly, up to 3.25 and 6.92 times faster respectively.

The rest of this paper is organized as follows. Section 2 outlines the improved system level exploration platform used. Section 3 presents the current co-design development methodology and its proposed modification at the exploration phase. Section 4 illustrates a design exploration process for two different applications following the modified methodology. The results are presented in section 5, while section 6 concludes.

2 Components of the Platform

StepNP is a system-level exploration platform for NPUs. Starting with a general architecture composed of standard reduced-instruction-set-computing (RISC) processors and a simple interconnect, StepNP allows easy plug-and-play replacement with more specialized processors, coprocessors, and interconnect; making the exploration task faster and easier. Its main components are a high-level multiprocessor-architecture simulation model; a routing application development kit; and a SoC control, debugging, and analysis toolset. These concepts and the modifications done to StepNP are explained in the next sections.

2.1 The System Level Architecture

A StepNP architecture consists of multiple IP components, including processors, memories, interconnections models from high-level split-transaction channel to more real NoC (e.g. crossbar, mesh, ring) and specialized coprocessors. All these components are SystemC 2.0 simulation models [6] with a standardized interface based on the Open Core Protocol [7]. By specifying the communication mechanisms, we enable plug and play of different SoC IP's at various abstraction levels and facilitate their development and exchange within different working groups.

This work focus on the configurable processor integration, which will permit to measure the impacts of different specialized functional units added into the processor's pipeline on the NPU performance. Different alternatives could be chosen to integrate this feature. For instance, the SimpleScalar tool set [8] can be used to simulate and analyze different microprocessors. It comes with an instruction interpreter for standard instruction sets and could be modified to simulate a more custom set. Lisatek tools [9] based on the LISA language can be used to describe a processor with a specialized instruction set. From this description, simulation tools and a skeleton of the RTL code can be automatically generated. The Xtensa processor can also be customized at the instruction set level with all the required tools automatically generated. Although the Xtensa solution comes with more architectural restrictions than the previous ones, the adopted solution provides a faster way to create custom processors and generates complete tool chains.

The Xtensa configuration process begins by accessing the Tensilica processor generator Web page. The designers can then select the desired features like endianess, basic functional units, bus width, instruction and data cache types and sizes. Using a proprietary language, they can also incorporate application-specific functionality to the processor by adding new instructions named TIE (Tensilica Instruction Extension). After selecting the proper options, the generation process produces the processor's configured RTL description and its configured software development tools, including a complete instruction set simulator (ISS). We wrapped in a SystemC module this ISS to support the StepNP standardized interface. By respecting the proposed interface and SystemC methodology, designers can also easily and quickly develop and incorporate dedicated coprocessors to accelerate specific tasks and evaluate their effects like complexity, speed and communication overhead on their design. Although it is possible to simulate an entire system using the Tensilica framework, creating a SystemC model for the Xtensa holds some advantages. For instance, StepNP contains tools to profile and trace the simulation of the entire system, not only the processor. Furthermore, based on IP reuse approach, it can simulate heterogeneous systems.

2.2 The routing Application Development Framework

At system level exploration, many design styles have to be evaluated in a short period of time. The routing application development framework needs therefore to be easy to use, flexible and modular. Because it will be executed on different types of architecture, the developed application needs also to be high-level straightforward portable code. Click is, by itself, a framework for rapid development of routing applications, which satisfies almost all these requirements. Built in C++, it is a modular software router originally developed by MIT [10]. It was ported on the Xtensa processor and modified to take advantage of the customized instruction set and the coprocessors on the platform.

Click can also serve as a testbench. Using a specialized communication mechanism named SIDL [4], a simple interface definition language used to pass data between two processes, Click can serve as a packets generator to feed the platform; and takes them out to verify if the right behavior is achieved.

2.3 The SoC Toolset

The toolset includes all the specific development tools for the individual processors used in StepNP (compiler, profiler, linker, etc.); and a top-level tool allowing a global visualization of the model execution from a number of perspectives.

3 Methodology

Figure 1 shows the main steps of the current co-design development methodology, with a few modifications applied to it (illustrated by dashed lines). This base design is analogous to other co-design methodologies as presented in [3] which focus mainly on the refinement process; contrarily to our approach that puts more emphasis on the architectural exploration.

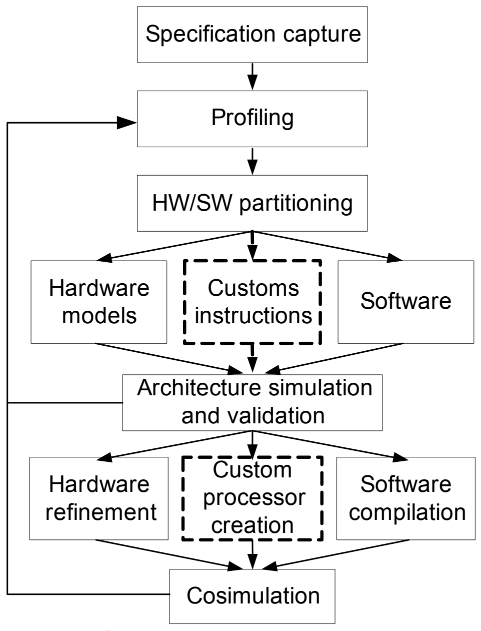

Figure 1. Co-design Proposed Methodology

Our approach starts by considering a complete software implementation on which we apply a software optimization to insure that no poor coded functions are wasting time. After a software analysis, the next step is to choose the best suited processor(s) for the targeted application. In the configurable processors world, it consists in selecting the right architectural features. Next, we propose a modification at the classical partitioning phase. At this level, instead of being completely hardware or software, the functionality to implement can be realized by one or many specialized processor instructions. A custom instruction has the advantage to be fairly easy created, added to the processor core, and integrated into the software that will run on it. Although it can give a significant performance speed-up, one has to be careful to not create too complex instructions that could slow the processor's pipeline to unacceptable speed. Furthermore, the instruction is only described with an instruction extension language [5]. With this description, all the necessary logic to implement the new functional units is automatically generated and added into the processor pipeline. This approach is therefore quicker and less prone to errors than custom hardware creation. As our results will show, custom instructions can be quite beneficial in certain cases, for example when a lot of bit manipulation has to be performed. On the other hand, a custom instruction yields no advantage for certain tasks like the copy of the packets between the input and output interfaces and the memory. In that case, a coprocessor managing the memory is a more appropriate choice.

In summary, our hardware/software partitioning is performed in four steps. First, the application should be run exclusively as software, since it is quicker and easier to develop this way, especially with a development framework like Click. After an initial profiling and the identification of bottlenecks not easily optimized in software, a second step selects the right architectural features, while a third step investigates the introduction of new functional units through the definition of custom instructions. The last step implements specialized hardware to replace the parts ill suited to be implemented directly into the processor.

In addition, sorted by progressive refinements, the following features complete our co-design flow:

- Individual profiling can be done for each processor, while StepNP provides tools for global performance analysis in SystemC.
- The initial and subsequent simulations and validations of the SoC design are done on the StepNP platform. The exploration process usually starts with a transactional channel that can be refined to a more complex communication mechanism afterwards.
- The Xtensa generator web page is used to provide an estimate of the speed, surface and power for a given implementation technology.
- It is possible to synthesize a compiled TIE instruction in order to estimate its critical path and see if it will negatively impact the pipeline speed.
- Hardware modules in SystemC can be refined towards a synthesizable model or described in other HDL languages.

4 Case Study

This section intends to clarify and enforce the proposed methodology by illustrating its use in a step-by-step process. The efficiency of the improved StepNP is also demonstrated. Two speculative but realistic network applications are initially developed with Click for this purpose. Different architectural alternatives are then presented with their associated results.

4.1 Specification Capture

The first specification is an IPv4 router that forwards packets in nearly full compliance with the standards. Such application includes pattern matching, lookup, data manipulation, computation and queue management; which are usual NPU's tasks. We limited the lookups to a small static IP routing table, since many commercial solutions already propose different algorithmic or hardwired approaches to perform searches and updates of huge tables [1, 11].

We included encryption to the previous IPv4 router specification to form the second application. Becoming an immediate necessity for nearly all information exchange systems, data encryption often requires high-speed encryption and decryption. A common algorithm is the Data Encryption Standard (DES) or a straightforward improvement known as Triple-DES. The enhanced version has been specified as an encryption algorithm for both the Secure Shell tools (SSH) and the Internet Protocol for Security (IPSEC). For these reasons, we chose to support the core of the DES encryption algorithm with cipher block chaining (CBC) mode.

4.2 Profiling

The profiling phase usually starts with a basic architecture composed of standard processors and a simple interconnect. We limit here the exploration process to one configurable processor for which we will try to attain the maximum achievable data rates for both applications. Nevertheless, it is possible to duplicate the processor and its coprocessor(s) to reach the required bandwidth by exploiting the inherent parallelism in packet processing. The characteristics of the initial Xtensa processor are: 32 bits bus width, 2 way 8KB instruction and data caches; and no special functional units. At this step, the code is compiled and profiled to identify the bigger time consuming tasks.

The platform is fed with two kinds of packets. First with minimal packets of 64 bytes, usually used as a benchmark standard by the industry; then with a more realistic workload consisting of variable Ethernet packet lengths as recommended in the RFC2544. The total execution time of the IPv4 application is distributed as follow:

1. *Window Overflow and Window Underflow:* Instead of wasting cycles to push and pop registers values at each procedure entry and exit, Xtensa processors use a 'sliding window' mechanism. Nevertheless, when *Window Overflow* and *Window Underflow* occurred, some cycles are necessary to recover.
2. *Push:* A Click router is an interconnected collection of modules called elements; each element performs a precise part of the overall application. Routing applications are built by gluing elements together. The virtual *Push* function is used to drive packets across these elements.
3. *Memory Management:* This category includes all memory management functions like *memcpy*, *malloc*, *memset*, *free*, etc.
4. *Packets In and Out:* Packets are accessible from the processor by a direct memory mapping. The *Packets In and Out* category includes all the functions in charge of these packet exchanges.
5. *CheckIPHeader:* It validates the different IP header fields.
6. *IP Checksum:* Consists on the IP checksum computing and update.
7. *Others:* Includes among other things lookups, fragmentations, Ethernet header settings, etc.

Table 1 gives for each of these categories the fraction of the time consumed. As the results show, there is a substantial price to pay on the performance for the use of a high level and flexible coded application. The overhead almost exclusively comes from a lot of embedded and virtual function calls. It is obvious with the *Window Overflow*, *Window Underflow* and *Push* functions, which spend as much as 39.7% of the total processing time of small packets and 25% for those of variable lengths. For packets of minimal length, the execution time is roughly distributed over all categories. For larger ones, *Memory Management* operations consume the main part, for which the *memcpy* function alone takes nearly half of the entire time. This can be explained by the fact that packets need to be copied from interfaces to the processor's RAM before any actions; and copied back after completed the processing. The *memcpy* function is also used two more times to align packets in memory, when their original 14 bytes Ethernet header is removed after proper treatments and replaced by a new one just before sending it back.

Table 1. Profiling results of the IPv4 application

Functions	**Time Consumed (%)**	
	Min. Length	**Var. Length**
Window Over/Under Flow	22.8	14.6
Push	16.9	10.4
Memory Management	24.9	58.2
Paquets IN / OUT	4.8	3.1
CheckIPHeader	3.4	1.2
IP Checksum	3.6	2
Others	23.6	10.5

Table 2 presents the profiling results of the application supporting DES-CBC encryption. We suppose that all packets need to be encrypted before leaving the router. A new category is then added to the previous ones:

8. *DES-CBC Encryption:* Contains all functions relative to DES-CBC encryption.

The encryption is obviously the main consuming task, reducing significantly the impact of all other categories on the performance. Because the DES-CBC algorithm is performed on the entire IP packet, its work is almost proportional to the packets size, taking from 48.9% of the complete processing time of small packets up to 86.8% for a more realistic workload.

Table 2. Profiling results of the IPv4 application with DES-CBC encryption

Functions	Time Consumed (%)	
	Min. Length	Var. Length
DES-CBC Encryption	48.9	86.8
Window Over/Under Flow	11.7	1.9
Push	8.3	1.4
Memory Management	14.3	7
Paquets IN / OUT	1.6	0.3
CheckIPHeader	0.8	0.1
IP Checksum	1.5	0.2
Others	12.9	2.3

4.3 Exploration Phase (Partitioning)

Guided by the profiling results and the specification constraints, the partitioning steps consist mainly in exploring the solution of the eventual SoC implementation area in order to find the best-suited components (processors, coprocessors, NoC, etc.) and their organization.

Following our methodology (section 3), Click allows for the first step a quick code optimization by means of minimal compilation/linking and devirtualization of the *Push* function. For our applications, these techniques do not significantly change the previous results.

In step 2, the architectural feature selection, it is possible to double the XTensa registers in order to decrease the WindowOverflow and WindowUnderflow functions calls. This reduces the task by almost a factor of two for both applications. For this particular case study, no additional features provide significant speed-ups.

In step 3, the best candidate found for custom instructions is the encryption. Consisting on extensive bit permutations and use of unusual registers lengths, the DES algorithm can easily be speed up with TIE instructions. Few of them accelerate this task by an interesting factor of 8. To illustrate TIE utilization, we also replace the IP checksum computing and update algorithms by two TIE instructions. Table 3 presents the speed-ups provided by different optimizations on the profiling categories for both applications. Since the time to perform most of these tasks is the same regardless the packets size, only the results to process packets of variable lengths are given. These gains are cumulative since there is no reason not to use previous optimizations; and they also include the function calls overhead.

A second candidate to keep improving the overall performance is the *Memory Management* processing, consisted essentially of *memcpy* functions calls. The first half of these function calls can easily be avoided simply by supporting unaligned accesses. The Xtensa processor has an Xtensa Local Memory Interface (XLMI) to connect peripherals, coprocessors or extra memories. We attached to this interface a fast 4K memory and modified the code of the application to move and process the packets in this memory range. We finally added the necessary logic to the XLMI interface (implemented in step 4) to support unaligned access, after that, packets can now be accessed unaligned.

Unfortunately, the other half of the *memcpy* calls cannot be easily avoided. Since this function mainly realized consecutive memory accesses, hardware cannot significantly improve it. So a TIE instruction is not best suited. Nevertheless, the processor can benefit from the use of a coprocessor (also implemented in step 4) to alleviate the memory management tasks. Therefore, we built a simple Direct Memory Access (DMA) controller to exchange the packets between the XLMI memory and the interfaces. Instead of calling the *memcpy* function to copy the packets, the processor now instructs the DMA controller to perform this function. Few modifications to the code are necessary to insure that a packet is always ready to be processed in memory. An additional port needs also to be added to the memory to allow two simultaneous accesses. No special cares for the cache and memory coherency are taken since this memory area is not cached. Together, unaligned access support and DMA controller reduce significantly the memory management processing by an order of magnitude.

Table 3. Gain provided by different optimizations on the profiling categories for a) Ipv4 application b) Ipv4 application supporting DES-CBC encryption

Functions	Gain Speed			
	Software	32->62 reg.	TIE	DMA
Window Over/Under Flow	0.95	1.78	1.76	2.03
Push	1.11	1.10	1.04	1.40
Memory Management	1.00	0.99	0.99	12.84
Packets IN/OUT	1.13	1.13	1.18	1.17
CheckIPHeader	1.09	0.82	1.00	0.99
IP Checksum	1.12	0.98	1.77	2.18
Others	0.95	0.67	1.06	1.53

a)

Functions	Gain Speed			
	Software	32->62 reg.	TIE	DMA
DES-CBC Encryption	1.01	1.00	7.96	7.93
Window Over/Under Flow	0.97	1.91	1.94	2.93
Push	1.11	1.08	1.18	1.55
Memory Management	1.09	1.04	1.10	9.83
Packets IN/OUT	1.01	1.01	1.05	1.06
CheckIPHeader	1.01	1.01	0.99	1.00
IP Checksum	0.97	1.02	1.85	2.22
Others	0.78	1.06	1.13	2.99

b)

4.4 Simulation/Validation and Synthesis

After each architectural modification, we simulate the platform and verify the accuracy of the functionality. By the same occasion, it serves as profiling tests to find the next possible optimizations.

Only the TIE instructions have been synthesized to insure than the processor frequency was not reduced. Since the DMA performance can be easily approximated, no synthesis model had to be done.

5 Overall Results

Figure 2 shows the overall speed-ups achieved after the different steps of our methodology. We obtain a gain of 1.92 for small packets forwarding and 3.25 for real packets with all optimizations applied. When encryption is added, the gains are boosted to 3.57 and 6.92.

Assuming a processor running at 400 MHz (using a typical 0.13µm technology), the obtained data rates are 109 Mbps and 1.16 Gbps for minimal and variable packet lengths processing respectively. They drop to 57.1 Mbps and 193 Mbps when the encryption is considered. Better results should be obtained by reducing the overhead of the high level coded application.

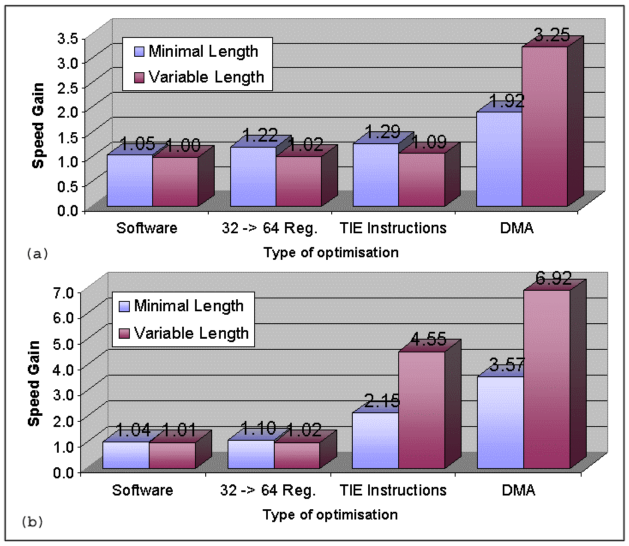

Figure 2. Overall gains for a) IPv4 application b) IPv4 application with encryption

Finally, the time spent on those optimizations can be divided in two parts: TIE instructions creation and co-processor model development. Considering a good understanding of the applications and a basic knowledge of the Tensilica tools, extracting potential instructions and creating TIE instructions have required 2-3 weeks. Note that, once the instructions have been created, integration into the processor is completely automated. On the other hand, developing a behavioral model of the custom co-processor in SystemC has been relatively fast (1-2 week) but the resulting model is farther from a synthesizable model.

6 Conclusion

In this paper we presented a system level exploration methodology for platform-based designs based on configurable processors. The enhanced StepNP development tool was used to develop different architecture prototypes. By using a custom instruction set processor and accelerating engines when appropriate, we obtained significant speed-ups for two different network applications. The main advantage of the proposed methodology is the rapidity to develop different NP architectures, allowing an efficient exploration of a large design space.

In order to improve the presented architectures, more realistic interconnects will be used. To better hide those new bus latencies, we plan to investigate on different hardware multi-threading architectures.

7 Acknowledgments

The authors would like to thank STMicroeletronics SoC platform automation group in Ottawa for their financial and technical support. Also, particular thanks to Tensilica for their support through their tools.

8 References

[1] Shah, N.; “Understanding Network Processors”, Master's Thesis, Dept. of Electrical Engineering and Computer Science, Univ. of California, Berkeley. 2001.
[2] Mihal, A. et al.; “Developing architectural platforms: a disciplined approach”, Design & Test of Computers, IEEE , Volume: 19 Issue: 6 , Nov.-Dec. 2002 , pp. 6-16.
[3] Cesario, W.O. et al.; “Multiprocessor SoC platforms: a component-based design approach”, Design & Test of Computers, IEEE, Volume: 19 Issue: 6, Nov.-Dec. 2002, pp. 52- 63.
[4] Paulin, P.G.; Pilkington, C.; Bensoudane, E.; “StepNP: a system-level exploration platform for network processors”, Design & Test of Computers, IEEE, Volume: 19 Issue: 6, Nov.-Dec. 2002, pp. 17-26.
[5] Gonzalez, R.E.; “Xtensa: a configurable and extensible processor”, Micro, IEEE, vol. 20 Issue: 2, March-April 2000, pp. 60-70.
[6] See Open SystemC web site: www.systemc.org
[7] See Open Core Protocol web site: www.ocpip.org
[8] See the SimpleScalar web site: www.simplescalar.com
[9] See the Lisatek web site: www.coware.com
[10] E. Kohler et al., “The Click Modular Router”, ACMTrans. Computer Systems, vol. 18, no. 3, Aug. 2000, pp. 263-297.
[11] Ji, H.M.; Srinivasan, R.; "Fast IP routing lookup with configurable processor and compressed routing table", GLOBECOM, IEEE, vol 4, Nov. 2001, pp. 2373-2377