Friday March 13th, 2009

By Daniel Shapiro

Computer Architecture Research Group

University of Ottawa

# Design Automation for an ASIP Empowered Asymmetric MPSoC

## Introduction

Multiprocessor systems have reached the marketplace and much research is needed on the subject. Multiprocessor System-on-Chip (MPSoC) designs may contain several processors, peripherals, a mechanism for interprocessor communication, and a memory model. MPSoCs are roughly classified into three groups according to the number of processors in the design: "multicore", "manycore", and "massively parallel". Multicore is defined in [1] as an embedded system having a small number of processors, while manycore systems have tens of processors. Massively parallel embedded systems are defined as systems with at least 64 cores. In MPSoC, the challenge is to harness the available processing power without losing too much time on synchronization and communication. This is not only a programmability problem; it is also a hardware design problem. The configuration of a multiprocessor system strongly determines the way that communication and synchronization are carried out. For example, symmetric multiprocessing systems contain a regular repeating structure of homogeneous processors. The advantage to this approach is that the design is simple and although the system performance could be improved with customization, the effort involved would not be worth the gain.

On the other side of this debate are the asymmetric multiprocessor designers who favor complexity in return for better performance, power, and area characteristics. The interconnection networks in asymmetric multiprocessor systems are not regular, and different cores require different compilers in order to be programmed. An asymmetric multiprocessor may have point-to-point communication, busses, network-on-chip, and shared memory communication all in the same design. The argument in favor of asymmetry presented in [1] is that tasks such as cryptography, packet processing, video processing, and many others should not be left to general purpose processors simply because the designed systems perform tasks that are known well in advance, and asymmetric multiprocessors can perform these tasks far faster and with much lower power consumption. Floating Point Units (FPUs) and Digital Signal Processing (DSP) components are two examples presented as evidence that even general purpose processors have required substantial customization since the early days of superscalar processing. Proponents of asymmetric MPSoC propose the development of a standard platform and a standard CPU architecture which would make these systems easier to program [1]. The symmetric and asymmetric multiprocessor design approaches are often referred to as Asymmetric Multi-Processing (AMP) and Symmetric Multi-Processing (SMP).

Research in computer architecture has focused on the automated generation of customized embedded processors. An Application-Specific Instruction Set Processor (ASIP) is a configurable processor for which the instruction set can be configured to best perform the tasks which will run on that core. Often an ASIP is a small single-issue processor extended with a number of customized functional units in the processor ALU. The peripherals and cache size of an ASIP may also be configurable at design time. The fields of MPSoC and ASIP are converging as the requirements for low power and high throughput devices are balanced with the need for flexibility. Complex products such as mobile phones and PDAs often require updates over time to fix bugs in the original design or to add new features. To that end, ASIP processors are more flexible than hardwired logic circuits designed in ASIC because the software they run can be updated. As well, ASIPs process tasks faster than a general purpose RISC processor because they contain hardware customized to the software tasks they will be issued.

Design automation is required in the fields of asymmetric MPSoC design and ASIP design due to the tight time-to-market constraints on product developers, and the large scale and high complexity of the design space. In order to ensure that the complexity of the design does not overwhelm the programmers and designers of the system, toolchains are constructed to automate the design process. There have been several attempts in academia and industry to create a toolchain which can automatically generate an MPSoC composed of multiple ASIPs, but none of them have managed to answer certain fundamental questions. For example, given that hardware area for the designed system is limited, it is not known what the tradeoff is between adding processors to an MPSoC versus adding additional hardware accelerators to the ASIP. And to complicate things even further we must choose a memory model and bus architecture for the MPSoC which also affects the design parameters such as latency of the system, power consumption, and hardware area.

Some problems have been addressed in earnest by industry, such as the need for compiler tools. Processor customization is simpler when it is done by a compiler, and so the software compilation and hardware description generation tools from companies such as Tensilica and CoWare enable embedded system designers to focus on design parameters instead of design complexity. Some researchers would argue that the toolchains provided by Altera (SOPC builder with the NIOS IDE) and Xilinx (EDK with Platform Studio) are also ASIP capable MPSoC toolchains, but these toolchains do not provide easy or automated instruction set configurability. On the algorithms side, there has been a flood of papers from academia describing algorithms for customizing a single RISC processor based on the tasks that it will execute, a small sample of which is contained in [2]--[9]. A survey of the topic can be found in [22]. The shortcoming in academic work has been in tackling the difficult tasks of deciding how and when to customize multiprocessors in an MPSoC. We will go over some of the recent work in the field, and then present some conclusions about the questions that are left unanswered.

In late 2007 Tensilica released a paper detailing the state of the art in configurable MPSoC, and pointed to the open problems which must be addressed in order to move into the massively parallel domain of embedded system design [11]. As of early 2008, many products were being designed with MPSoC and ASIP industrial compiler toolchains [10]. The scale of some of these products was reaching into the manycore domain with a few networking applications in the massively parallel domain, and reaching the limits of the existing toolchain capabilities. In [11], the need for a platform for MPSoC is emphasized. However, the answers to other problems are not clear. Specifically, the five open questions presented were: How do we improve the way that concurrency is extracted from software? How do we find a balance between AMP and SMP approaches by looking at an application? What should the tradeoff be between the number of processors and the number of custom instructions? What is the proper mix of interprocessor communication hardware (point-to-point, busses, network-on-chip, shared memory) given any application? And finally, how should the topology of a massively parallel MPSoC be structured to best perform the tasks of the system?

In [15], two challenges to multicore architectures were presented. First, MPSoCs that perform data-intensive tasks experience a lot of cache replacements in the high-level caches because so many cores have access to the same cache. Second, the disk access times are very slow and must be carefully interleaved to prevent processors from idling. They attempt to solve each of these problems at the operating system level.

Next we will review several approaches in the field of multiprocessor and ASIP. We conclude with some analysis and remarks on the state of the art and possible research directions.

## Tensilica's Xtensa

The Xtensa processor from Tensilica is a configurable and extensible ASIP core. Tensilica has not yet released a fully integrated multiprocessor design toolchain, but a methodology and standards have been put in place by Tensilica to enable multiprocessor system design. A single processor design example using the Xtensa configurable processor is described in ‎[12]. The partition of a system specification into hardware and software is shown for a 2 dimensional Discrete Cosine Transform (2-D DCT).

Tensilica describes in ‎[16] their own methodology and design flow for developing multiprocessor systems with their software and hardware generation toolchain. The design flow starts with the optimization of computation and communication models of the program which will be implemented on a multiprocessor. The program need not be completed, and instead it can represent an approximate model of the system. Next the tasks in the software model are mapped to a hardware architecture and the communication model is analyzed for bottlenecks. This process is repeated in an iterative way until communication in the multiprocessor design meets the high-level design constraints. At this point the architecture of each processor is refined. The single processor design toolchain from Tensilica is used to optimize the power, area, and throughput of each processor to meet the system constraints. Next a system level model is composed from the simulator of each individual processor in the multiprocessor system. The entire system can be simulated using the XTMP modeling protocol to measure the expected performance and to execute software simulations to verify the expected results against an accurate model of the system. After extensive testing of the hardware and software of the design, it can be implemented in an ASIC design flow. Tensilica is expected to release a multiprocessor system design toolchain in the near future, and this toolchain is expected to be much more automated than the current design flow.

In ‎[13] a design flow for the implementation of multiprocessors with ASIPs is presented. In this approach the assignment of tasks to Xtensa processors is interleaved with instruction set extension identification. This approach to high-level system design is focused on the task level of the design space exploration (DSE) problem, which is a higher level of abstraction than other approaches which look at each core individually after task to processor assignment has been completed. The authors show that task to processor assignment is inextricably bound to the custom instruction selection choices which will be made by the system designer. Therefore an iterative improvement algorithm is used to assign tasks to processors and schedule them, while selecting custom instructions which minimize the critical path in the dataflow graph of the task. Their approach includes support for software pipelining, a compiler technique used to improve the execution time of some loops. Using their AMP approach they demonstrate improvements in benchmark execution of 2 times on average over an SMP-based approach. The authors of [13] describe in [14] a methodology for mixing custom instructions with coprocessors. This is further evidence of their strong showing in the ASIP side of MPSoC design. The most important result to come out of [13] is that automated generation of MPSoCs with ASIPs is possible to tackle. They used a shared memory to perform all interprocessor communication. The design flow is presented in the following figure:



## A LISATek Based Approach

In [28] LISATek from Coware was used to generate a compiler toolchain, software, and hardware descriptions from an Architecture Description Language (ADL) specification. SystemC was used with CoWare's ConvergenSC to design the multiprocessor architecture, and each ASIP processor was designed with a SIMD 5-stage pipeline. The goal in [28] was to design using available tools a multiprocessor network of ASIPs which can perform turbo decoding (a digital communication operation) with speed and flexibility. The design was tested for 4, 8, 16, and 32 processors. The main result for this paper was their ability to achieve approximately the same throughput as comparable systems but with higher flexibility.

## Daedalus

The Daedalus system-level toolchain was designed to simulate a multiprocessor system and then implement it in a single tool. Daedalus produces both software and hardware, and has a platform for hardware implementation. The Daedalus system is described in ‎[17] and ‎[18] and is composed of three core tools: KPNgen, Sesame and ESPAM. The GCC compiler is used in Deadalus as the software compiler. KPNgen is used to create a Khan Process Network (KPN) representation of the tasks of the system. This type of representation is used to model distributed or parallel systems. Because only KPN specifications can be processed by the tool, only programs written in the form of static affine nested loops can be used with the tool. The models in Daedalus are stored in XML, as are the parameters, and results. KPNGen sends the resulting KPN into ESPAM. ESPAM also accepts a platform specification of the system hardware and a specification defining a partition between hardware and software in the design. These two XML inputs come from the system-level design space exploration tool called Sesame. Daelalus can target only Xilinx FPGAs, and the exported multiprocessor system can include a mix of PowerPC and MicroBlaze processors.

ESPAM (described in ‎[19] and ‎[20]) is used for processing the system-level specification written in XML. It converts the specification into low-level hardware and software which can be compiled for an FPGA. An ESPAM user can specify a multiprocessor system at a high level of abstraction in a short amount of time, and then the specification can be refined into a real implementation within a few hours. ESPAM has adopted the Eclipse IDE as a graphical frontend for the tool. The user interface can be used to define a scalable architecture template for stream-oriented MPSoCs. When a design is compiled, the Sesame tool explores the design space and then automatically generates the platform and the mapping specification. These can also be written by hand.

ESPAM follows a three step approach for implementing the specification: First, ESPAM constructs a model representing the platform for the implementation using a platform specification, an application specification, and a mapping specification. The platform specification is restricted. It must specify a platform composed of components from a predefined library. The platform describes the topology of a multiprocessing platform. The application specification is a KPN representation of the input program. KPNs are a parallel dataflow model of computation. The mapping specification can specify one-to-one and/or many-to-one processes into processors. This specification describes the relation between all processes in the application specification and all of the components in the platform specification. Second, the abstract platform model is refined into an RTL level model. Third, ESPAM generates software and a synthesizable hardware description for each processor and component in the multiprocessor platform. The output from ESPAM is a hardware description of the MPSoC and software code which can be used to program each processor in the MPSoC. At the RTL level of the design specification the platform topology is described in the form of a netlist. IP-cores are mapped into the netlist in the platform topology where appropriate, and an HDL compiler can be used to create a bit file for programming a Xilinx FPGA.

The applications of ESPAM are broad. The authors of ‎[19] consider in their work data-flow dominated applications in the realm of multimedia, imaging and signal processing. This type of program can be efficiently modeled by using KPNs. The purpose of the ESPAM tool is to bridge the gap between the system level specification and the RTL specification, and finally an FPGA-based implementation. Modules in the multiprocessor system model are called memory components, processing components, communication components, and communication controllers. PowerPC and MicroBlaze processors can be included in the system design. These components communicate using only distributed memory units, and not shared memory. Components are mapped to third-party developed IP-cores including Xilinx cores for processors and memory. The communication component in ESPAM supports peer to peer networks, crossbars, and shared busses. Communication Controllers (CC) are used to synchronize the processors at hardware level of the design. FIFO buffers are generated in order to streamline communication. CCs are composed of an interface, controller and set of buffers. The peer to peer network in ESPAM has the same topology as the topology of the process network passed to ESPAM. Because there is no CB or Bus communication in peer-to-peer, there is no need to grant connection requests and no sharing of communication resources. This will effectively reduce the communication and synchronization delay.

## A NIOS-II Based Approach

In [21] a multiprocessor ASIP compiler was designed using the NIOS-II processor core and the Avalon bus standard. There are three options available in the tool during design space exploration: Do not modify the source code, identify and implement custom instructions, and use a multiprocessor architecture. The partition choice for the multiprocessor architecture is based upon Hierarchical Concurrent Dataflow Graph (HCDFG) representation and profiling information obtained by analyzing the application source code. Custom instructions can be either fine grain small ALU components or coarse grain functions implemented as circuits. The memory model includes a mailbox core and private and shared memories. Power dissipation is analyzed for the implementation on Stratix II and Cyclone II FPGAs. The fine grained and coarse grained approaches are compared for 2CPU and 4CPU designs. The fine grain approach consumed less power and had a higher speedup.

## Srijan

In 2002, the Srijan project was started at IIT in Delhi [31]. In a broad sense the goals of the project were to develop application specific multiprocessor development tools [32]. By 2005 the group had achieved System-Level DSE in the area of MPSoC performance estimation, multiprocessor simulation, and case studies on multiprocessor applications [33]. At the processor level the group was studying memory synthesis, and configurable ASIP generation for VLIW and RISC architectures. They were also making progress in the development of a compiler toolchain to target their processors using the Trimaran compiler infrastructure. The progress report in [33] states that the Srijan project "was started with the objective to develop an integrated framework in which system and sub-system design space can be thoroughly explored for system on a chip multiprocessor architectures."

When [33] was published in 2005 the performance of a processor could be measured in number of cycles executed, memory accesses, stalls, register reads, register writes, and other parameters. Power estimation statistics were also generated for each component in the design such as processor, memory, register file, cache, bus, and custom hardware units. Their ASIP core could be configured to change the number of issue slots, number of registers, and many other parameters in an automated way. At that time, only a uniprocessor VLIW architecture description was implemented, and MDES machine description was used as the architecture description for the compiler. SystemC models of the processor were created in order to accurately simulate the system at the transaction level. Most critically, a single unified representation was used to describe the processor which was then used to generate the MDES and SystemC description. Therefore the model was extensible and platform ready; able to act as a building block to make a multiprocessor system of ASIPs in the future [33].

Dataplane and control plane custom instructions were investigated for VLIW ASIP architectures, along with multi-ported data-caches used to issue Multiple-Input Multiple-Output (MIMO) instructions. An inexact Instruction-Set Extension (ISE) approach was pursued, as were compiler optimizations aimed at finding compile-time optimizations for VLIW processors containing complex register-to-register interconnection networks [33].

More recently, students at IIT Delhi wrote the report "MultiProcessor Systems on Field Programmable Gate Arrays" a capstone undergraduate project on multiprocessor system generation described in [34]. Their tool augments EDK system descriptions to provide the user with a scalable multiprocessor "Resource Sharing and Synchronization Arbiter" (RSSA). The arbiter is composed of point-to-point links that allows resource reservation and arbitration. Other students at IIT Delhi in the Srijan group wrote [35] in the area of power analysis of MPSoCs, and [36] on custom instruction selection and implementation for ASIPs. In [37] an automated DSE algorithm was added to Srijan for mapping processes to processors efficiently, and configuring the register file size and other architecture parameters. The algorithm in [37] is able to optimize (find Pareto points) for power, area, delay, or a combination of these objectives by varying the number of registers, Issue Slots, processors and process mapping. The system works with up to 4 processors.

It is clear that the Srijan group is working on the components of an automated tootchain for the development of an MPSoC with VLIW ASIPs. They are heading in the direction of multiprocessor system design based upon source code analysis. They understand the need for such a toolchain and they have shown the skill needed to develop the various components. However, the final product of their efforts has not yet been completed, and so it sits in two separate pieces as a configurable ASIP tool and a multiprocessor system design tool.

## RAMP

The Research Accelerator for Multiple Processors (RAMP) project is focused on building massive multi-FPGA systems for simulating parallel systems of all kinds in a transparent way. The project is described in brief in ‎[23] and ‎[24], and extensive detail on the project is available from ‎[25]. The RAMP project aims to provide researchers with a platform for designing large embedded multiprocessor systems. The following are some of the goals of the project:

* Provide a transparent system design fabric which hides changes in the underlying FPGA technology
* Provide very fast simulation times (compared to software simulators)
* Support the use of operating systems and software on the designed multiprocessor systems (an important research direction discussed in [1])
* Provide a low cost solution for very accurate modeling of multiprocessor systems
* Expose the visibility of design errors
* Provide a scalable low power consumption platform for prototyping
* Allow for a variety of implementation types and flexibility in what the user will model
* Support a wide variety of software and hardware languages, including MATLAB, C, C++, Java, Verilog, VHDL and other languages

RAMP provides a language for specifying distributed systems called the RAMP Description Language (RDL), along with a compiler for creating cycle-accurate emulation of the distributed system on the RAMP implementation fabric. The implementation layer of the system is compatible with both Xilinx and Altera FPGAs, along with many other hardware platforms.

## MILAN

The MILAN framework described in ‎[26] and ‎[27] is a tool for heterogeneous multiprocessor system design. MILAN takes in user software and real-time requirements, and quickly computes a good set of configurations to implement the requirements. The tool is based upon the Model Integrated Computing (MIC) project which facilitates the creation of domain-specific models, transformations on these models, and validation methods. MILAN also uses GME, a graphical frontend for MIC. Systems designed with MILAN can include MIPS and StrongARM processors. The tool takes a three step approach to the design of heterogeneous systems. First the user defines a set of tasks in a modeling environment, and then a module called DESERT performs symbolic search to enumerate a set of candidate designs which meet the requirements set out by the user. This first step can be tried several times by the user until a satisfactory result in terms of population size and fitness is observed. Next the output of DESERT is profiled by a module called HiPerE, which finds the power consumption and latency data of each component in the design. This second step can also be fed back into the first step to further reduce the number of candidate solutions to the system design problem. Finally, low-level simulations of the power consumptions and latency are performed using MIPS and ARM simulators. MILAN essentially connects DESERT and HiPerE to models and simulators.

Functional descriptions of MILAN models can be translated into C, Java, or MATLAB. Design constraints on the MILAN models are expressed in the Object Constraint Language (OCL). During DSE the constraints on the system may be tightened by the tool in order to reduce the number of results. The main focus in MILAN was to optimize power consumption while maintaining or improving throughput. Constraints such as power consumption, execution time, and implementation cost are defined as constraints on the design space exploration stages.

Hardware components are modeled in great detail. Each memory unit and computation sequence in a candidate system is described using a hardware model. Computation, storage, and communication objects are the basis of the hardware models. Interestingly the configuration of each component is defined in the model.

MILAN can take in an application model and architecture model from the GME and perform two levels of design space exploration which enumerate successively fewer candidate solutions. Hardware descriptions for these candidate solutions can then be generated and synthesized. MILAN was designed in a highly configurable way, so that simulators and exploration tools can be replaced or added to the framework and additional processor and IP core types can be supported. MATLAB and SystemC models can be used in the MILAN framework to verify the functionality of a model. Also, user-specified subsections of the model can be simulated in isolation to save time when simulating.

## Conclusions

Tensilica’s design flow for multiprocessor systems is not yet fully automated. However, they are leading the industry in ASIP design and posing hard questions as open problems. Tensilica has developed some amazing technology, but it is proprietary and only targets Tensilica's proprietary architecture. For this technology to gain momentum over SMP, a free and open source solution is needed so that designers can experience some of the benefits of asymmetric multiprocessing [1]. Patents complicate efforts to deploy open-source solutions. In [10] the issue of patenting compiler and systems technologies illuminates the hardships that would be faced by academics attempting to open-source the research conducted on this topic.

The academic solutions we looked at that are not based on Tensilica's ASIP technology were Srijan, RAMP, Deadalus, MILAN, and [21]. Srijan was very interesting and based on ADL instead of a more abstract ISA description. The project has been well developed over the past several years but there has yet to be completed a bridge between their successful work on ASIP and multiprocessors. The RAMP project is not focused exclusively on efficient multiprocessor system design, and does not support ASIP out-of-the-box. Instead they focus on simulation capabilities, and massively multiprocessor systems. Daedalus is a tool which simulates and generates multiprocessor systems, but it does so only for specific types of programs (KPN representation). MILAN is a toolchain for generating configurable multiprocessor systems by performing DSE on a set of configurable system parameters, but do not have ASIPs in their designs. And so a picture emerges wherein there are good tools for ASIP, good tools for MPSoC design, but no good tools for both. Perhaps the only bright spots are the work in [13] and [21], which contain both ASIP processors and multiprocessor parallelization. [21] is suitable for the multicore domain, and [13] for the manycore domain. The direction in reasearch that looks most promising is that of [13].

The following table summarizes the capabilities of the tools and approaches that were discussed:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Scale (multicore, manycore, massively parallel)** | **MPSoC Automation** | **ASIP Automation** | **Configurable Processor** | **Interconnection Network DSE** |
| **Tensilica** | manycore  | N (user defined)  | Y | Y | N |
| **[13] based on Tensilica tools**  | manycore  | Y | Y | Y | N (shared memory) |
| **CoWare** | manycore  | N (user defined) | N (user defined)  | Y | N |
| **[28] based on Coware tools**  | manycore | N (problem specific topology)  | N (user defined)  | Y | N |
| **Altera** | multicore  | N (user defined)  | N | Y | N |
| **[21] based on Altera tools** | multicore | Y | Y | Y | N (mailbox) |
| **Xilinx** | multicore  | N (user defined)  | N | Y | N |
| **Daedalus based on Xilinx tools**  | manycore  | Y | N | Y | Y |
| **Srijan** | multicore | Y | Y (not integrated) | Y | Y |
| **RAMP based on Xilinx tools**  | massively parallel  | N (user defined)  | N | Y | N (user defined through API) |
| **MILAN**  | manycore  | Y | N | Y | Y |

To build a usable toolchain takes man-years of effort and requires years of sustained effort and funding. "It takes approximately four years and many millions of dollars to prototype a new architecture in hardware, even at only research quality" [29]. And so we must limit our research to solving open problems with the hope that we can bring forward a platform which supports both MPSoC and ASIP technologies. Some open problems in the field of configurable MPSoC are:

* Describe mathematically the tradeoff between the number of processors and the number of custom instructions for an MPSoC with ASIPs.
* Extract concurrency in a general way from application software and apply it to the design of an MPSoC. This will require a balance between compile-time and run-time scheduling.
* Find a balance between AMP and SMP approaches based upon application software.
* Describe the proper mix of interprocessor communication hardware (point-to-point, busses, network-on-chip, shared memory) by analyzing application software.
* Describe the topology and memory hierarchy of a massively parallel MPSoC by analyzing application software.

## References

1. Tom R. Halfhill, "The Future of Multicore Processors," Microprocessor Report [online], Dec. 2007 [cited Mar. 12, 2009], available from World Wide Web: <http://www.mdronline.com/mpr\_public/editorials/edit21\_53.html>.
2. P. Biswas, S. Banerjee, N. Dutt, L. Pozzi and P. Ienne, "ISEGEN: generation of high-quality instruction set extensions by iterative improvement," Design, Automation and Test in Europe, 2005. Proceedings, pp. 1246-1251 Vol. 2, 2005.
3. B. Korte and J. Vygen, Combinatorial Optimization: Theory and Algorithms., 3rd ed. Springer, 2005, pp. 597.
4. C. Ozturan, G. Dundar and K. Atasu, "An integer linear programming approach for identifying instruction-set extensions," Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on, pp. 172-177, 2005.
5. P. Biswas, N. Dutt, P. Ienne and L. Pozzi, "Automatic Identification of Application-Specific Functional Units with Architecturally Visible Storage," Design, Automation and Test in Europe, 2006. DATE '06. Proceedings, vol. 1, pp. 1-6, 2006.
6. C. Galuzzi, E. M. Panainte, Y. Yankova, K. Bertels and S. Vassiliadis, "Automatic selection of application-specific instruction-set extensions," Hardware/software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference, pp. 160-165, 2006.
7. Instruction-set Extensible Processors under Data Bandwidth Constraints," Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, pp. 1-6, 2007.
8. P. Bonzini and L. Pozzi, "Polynomial-Time Subgraph Enumeration for Automated Instruction Set Extension," Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, pp. 1-6, 2007.
9. R. V. Bennett, A. C. Murray, B. Franke and N. Topham, "Combining source-to-source transformations and processor instruction set extensions for the automated design-space exploration of embedded systems," in LCTES '07: Proceedings of the 2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, 2007, pp. 83-92.
10. P. Bonzini and L. Pozzi, "Code transformation strategies for extensible embedded processors," in CASES '06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, 2006, pp. 242-25.
11. Ken Wagner, "In Conversation with Tensilica CEO Chris Rowen," Design & Test of Computers, IEEE, vol. 25, pp. 88-95, 2008.
12. D. Goodwin, C. Rowen and G. Martin, "Configurable Multi-Processor Platforms for Next Generation Embedded Systems," Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific, pp. 744-746, 2007.
13. S. Xu and H. Pollitt-Smith, "Optimization of HW/SW Co-Design: Relevance to Configurable Processor and FPGA Technology," Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on, pp. 1691-1696, 2007.
14. Fei Sun, N. K. Jha, S. Ravi and A. Raghunathan, "Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors," VLSI Design, 2005. 18th International Conference on, pp. 551-556, 2005.
15. Fei Sun, S. Ravi, A. Raghunathan and N. K. Jha, "A Synthesis Methodology for Hybrid Custom Instruction and Coprocessor Generation for Extensible Processors," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 26, pp. 2035-2045, 2007.
16. Xiaodong Zhang, "Research Issues and Challenges to Advance System Software for Multicore Processors and Data-Intensive Applications," Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on, vol. 1, pp. 4-4, 2008.
17. Tensilica Inc., "Design process" [online], [cited Oct. 20, 2008], available from World Wide Web: <http://www.tensilica.com/methodology/design\_process.htm>.
18. M. Thompson, H. Nikolov, T. Stefanov, A. D. Pimentel, C. Erbas, S. Polstra and E. F. Deprettere, "A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs," in CODES+ISSS '07: Proceedings of the 5th IEEE/ACM International Conference on Hardware/software Codesign and System Synthesis, 2007, pp. 9-14.
19. H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu and E. Deprettere, "Daedalus: Toward composable multimedia MP-SoC design," Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE, pp. 574-579, 2008.
20. H. Nikolov, T. Stefanov and E. Deprettere, "Systematic and Automated Multiprocessor System Design, Programming, and Implementation," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 27, pp. 542-555, 2008.
21. H. Nikolov, T. Stefanov and E. Deprettere, "Multi-processor system design with ESPAM," in CODES+ISSS '06: Proceedings of the 4th International Conference on Hardware/software Codesign and System Synthesis, 2006, pp. 211-216.
22. Yassine Aoudni, Mohamed Abid, "A FRAMEWORK FOR ASIP AND MULTIPROCESSOR ARCHITECTURES INTEGRATION WITHIN RECONFIGURABLE SOC AND FPGA DEVICES," Arabian Journal for Science and Engineering, vol. 32, no. 2C, Dec. 2007, pp. 3–12.
23. Carlo Galuzzi, Koen Bertels, "The Instruction-Set Extension Problem: A Survey," in ARC '08: Proceedings of the 4th international workshop on Reconfigurable Computing, pp. 209-220, 2008.
24. J. Wawrzynek, M. Oskin, C. Kozyrakis, D. Chiou, D. A. Patterson, S. Lu, J. C. Hoe, K. Asanovic, "RAMP: A Research Accelerator for Multiple Processors," Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, Report No. UCB/EECS-2006-158, Nov. 2006.
25. G. Gibeling, A. Schultz, K. Asanovic, "The RAMP Architecture & Description Language," presented at WARFP 2005, Austin, TX, March, 2006.
26. G. Gibeling, "RDLC2: The RAMP Model, Compiler & Description Language," M.A.Sc. thesis, Dept. of Electrical and Computer Sciences, University of California at Berkeley, CA, May 2008.
27. S. Mohanty, V. K. Prasanna, S. Neema and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," SIGPLAN Not., vol. 37, pp. 18-27, 2002.
28. A. Bakshi, V. K. Prasanna and A. Ledeczi, "MILAN: A model based integrated simulation framework for design of embedded systems," in LCTES '01: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems, 2001, pp. 82-93.
29. O. Muller, A. Baghdadi and M. Jezequel, "ASIP-Based Multiprocessor SoC Design for Simple and Double Binary Turbo Decoding," Design, Automation and Test in Europe, 2006. DATE '06. Proceedings, vol. 1, pp. 1-6, 2006.
30. J. Wawrzynek, M. Oskin, C. Kozyrakis, D. Chiou, D. A. Patterson, S. Lu, J. C. Hoe, K. Asanovic, "RAMP: A Research Accelerator for Multiple Processors," Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, Report No. UCB/EECS-2006-158, Nov. 2006.
31. Anup Gangwar, "Embedded Systems Group, IIT DELHI" [online]: Embedded Systems and Architecture Group at IIT Delhi, [cited Mar. 13, 2009], available from World Wide Web: <http://www.cse.iitd.ac.in/esproject/>.
32. Anshul Kumar, "Srijan: A Methodology for Synthesis of ASIP Based Multiprocessor SoCs" [online]: Embedded Systems and Architecture Group at IIT Delhi, [cited Mar. 13, 2009], available from World Wide Web:
<http://embedded.cse.iitd.ernet.in/homepage/docs/seminars/anshul/srijan.overview.ppt >.
33. Anshul Kumar, M.Balakrishnan, Preeti Ranjan Panda, Kolin Paul, et.al., "Srijan - A Methodology for Synthesis of ASIP based Multiprocessor SoCs," Embedded Systems Group, Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India, Report No. 2005/1, Dec. 2005.
34. Rajat Sahni, Nilay Vaish, "MultiProcessor Systems on Field Programmable Gate Arrays" B. Tech final report, Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India, 2007.
35. M. Gupta, N. Goel and M. Balaksrishnan, "Energy Based Design Space Exploration of Multiprocessor VLIW Architectures," Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on, pp. 307-310, 2007.
36. N. Pothineni, A. Kumar and K. Paul, "A Novel Approach to Compute Spatial Reuse in the Design of Custom Instructions," VLSI Design, 2008. VLSID 2008. 21st International Conference on, pp. 348-353, 2008.
37. Kaushal Shubhank, Sumit Kumar, "Srijan: Unified Architecture Specification and Design Space Exploration" B. Tech thesis, Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India, 2006.