Network Processors: A Perspective on Market Requirements, Processor Architectures and Embedded S/W Tools

Pierre G. Paulin
STMicroelectronics
Central R&D, SoC Platform Automation
16 Fitzgerald Rd. Suite 300,
Nepean, Ontario, K2H 8R6, Canada
pierre.paulin@st.com

Faraydon Karim
STMicroelectronics
Central R&D, Advanced Design Group
4690 Executive Drive
San Diego, CA 92121, USA
faraydon.karim@st.com

Paul Bromley
STMicroelectronics,
Advanced System Technology
1060 Brokaw Rd.
San Jose, CA, USA
paul.bromley@st.com

Abstract

With the projected explosion of low-cost bandwidth availability, the intensive processing tasks and service hosting will move close to consumers on the "intelligent edge" of the network, where a significant portion of the future storage, processing and network management will take place. We address the rationale for this change, the characteristics of the network processor architecture required to address it, and the software development tools needed in order to improve time-to-market without sacrificing embedded software performance.

1. Network Infrastructure Trends

We have been studying the network infrastructure to understand the technologies required to participate in the considerable future market as the Internet and "Anywhere, Anyhow" communications spread across the globe over the next five years. If we look to a future network based on ultra-high bandwidth fibre communications this will radically shift our preconceptions about where computation and storage take place.

Two simple facts illustrate this shift. In the labs of leading telecom companies, a throughput of 6.4 Terabits/s (6400 Gbit/sec) has been demonstrated using a single fibre strand by means of Wave Division Multiplexing (WDM). The world's total voice traffic in 1999 was 10 Terabits/sec, and the world's transoceanic cable capability will have grown 1000% between 1999 and 2001. In other words, communication bandwidth and the price of that bandwidth will become much less significant in the near future.

This will have a dramatic effect on the complexity and protocols of today's networks. There will be a trend towards much greater simplification and efficiencies through the widespread use of WDM and IP. The "last mile" to the home will remain a challenge, but this is being addressed progressively by xDSL, cable modems, broadband wireless and satellite links.

It is our contention that this bandwidth availability in the wider network will mean that the intensive processing tasks and service hosting will move close to consumers on the "intelligent edge" of the network. It is proposed that this intelligent edge is where a significant portion of the future storage, processing and network management will take place. Therefore, we are actively working at the corporate level on the packet processor core often called a "Network Processing Unit" – or NPU – that is the fundamental core of that intelligent edge.

2. Network Processing Unit Architecture

The projected explosive growth in network bandwidth and services has created the need for a new breed of processors. The rapid introduction of new protocols and applications forces network designers to deal with fast "time-to-market" but also brief "time-*in*-market." An emerging approach to this challenge is the development of

a programmable multi-processor solution that preserves customer investment by keeping up with ongoing changes.

Packet processing poses some challenges in comparison to general data processing. Locality can be poor in network applications. The reason stems from the fact that packet data is essentially data flow. One packet arrival has little in common, generally, with any other packet arrival. This eliminates much of the utility of a traditional data cache for network processors. Furthermore, data structure access per packet also has poor locality. For example, hash and longest prefix match searches are pointer tracing. Thus, the memory data accessed has little necessary inter-relation. Both memory capacity and bandwidth are demanding. Routing look-up tables may not only be huge, but they must also be very fast. Generally, in order to make the timing work, one or more forms of latency hiding techniques are required.

There are some general characteristics of network processors. A network processor is a highly integrated set of micro-coded or hardwired accelerated engines, the memory sub-system, and high-speed interconnect and media interfaces to tackle packet processing close to the wire. It uses pipelining, parallelism, and multi-threading to hide latency. It has good data flow management and high-speed internal communications support. It has the ability to access co-processors and is closely coupled with the media interface.

Network processors present a whole new set of requirements. In our bandwidth hungry world, OC-12 and OC-48 network speeds are becoming common. On the horizon is OC-192 which allows for only 52ns of processing per packet received. After that, OC-768 will soon follow, leaving us only 13ns of processing time per packet. It is clear that traditional microprocessors cannot keep up with the speed and programmability requirements of network processors.

2.1 Illustrative Commercial NPUs

Seeing the unique needs of network processors, many companies such as Intel, MMC and IBM have introduced new designs for this fast paced market.

Intel's IXP 1200 [1] is targeted at LAN-WAN switches operating at OC-48 speeds. The architecture consists of 6 micro-engines sharing a bus with memory. The micro-engines are managed by a StrongARM core processor. It has a PCI bus to communicate with the host CPU, memory controllers, and a bus interface to network MAC devices. The device operates at 162MHz. Each micro-engine supports 4 threads, which helps to eliminate micro-engines waiting for memory resources. Micro-engines have a large register set, consisting of 128 general-purpose registers, along with 128 transfer registers. Shift and ALU operations occur in a single cycle. A hardware hash unit is responsible for the generation of 48 or 64-bit adaptive polynomial hash keys. Multiple IXP 1200's can be aggregated in serial or parallel.

MMC [2] developed the AnyFlow 5000 network processors. These have five different stages: ingress processing, switching, queuing, scheduling, and egress processing. Per-flow queuing is used which allows each flow to be queued independently. Other functions handled on a per-flow basis are queuing control and scheduling.

MMC also developed the nP3400, which integrates a programmable packet processor, switch fabric, and multiple Ethernet interfaces on a single chip. It contains two programmable 200-MHz RISC processors and a 4.4 Gb/s switch fabric. It has policy engines supporting 128 rules.

IBM developed the Rainier NPU [3]. It has sixteen programmable protocol processors and a PowerPC control processor. It has hardware accelerators to perform tree searches, frame forwarding, filtering, and alteration. Each processor has a 3-stage pipeline (fetch, decode, execute) and runs at 133 MHz. Each processor has seven coprocessors associated with it, including one for checksum, string copy, and flow information. Hardware accelerators perform frame filtering and alteration and tree searches.

2.2 NPU Design at STMicroelectronics

We are looking at processor design differently. First, networking is mostly data flow with some unique computational issues. Second, the nature of the computations, branches and pointers require a new set of instructions unique to this type of environment. We present the key characteristics of this configurable and scaleable multi-processor solution designed to address the rapid changes in networking needs and customer requirements.

We address the required performance of network processors by attacking many different fronts. Instruction-set definition, pipelining, parallelism, multithreading, fast interconnect, and semiconductor technology all combine to produce a network processor capable of OC-192 speeds and higher.

Speedup is possible through an enhanced instruction-set which is designed specifically for network-oriented applications. There are specific instructions for field extraction, byte alignment, comparisons, boolean computations, endianess, conditional opcodes used to reduce branches, and more powerful network-specific computational instructions.

Another form of speedup is achieved via pipelining. Our network processor has a chain of pipelined processing units and flexible programmable elements.

A variety of accelerating techniques round out STMicroelectronic's network processor. These include:

hardware multi-threading, which provides latency hiding; hardware accelerators for header parsing, checksums, and other intensive per-packet networking tasks; and finally, a world-class semiconductor technology.

The way in which all the packet-processing engines in the network processor connect to outside resources is crucial. If every packet processing engine is unable to continue work because it is limited by a slow interconnect, then much of the processing power is wasted. To overcome this issue, we went to great lengths to ensure the fastest possible packet processor to resource interconnect. Whether the packet processor is sharing a memory bus resource or special hardware coprocessor, the octagon interface shown in Figure 1 insures that between any two nodes, only two hops are needed, and that multiple packet-processing engines can talk to different resources simultaneously.

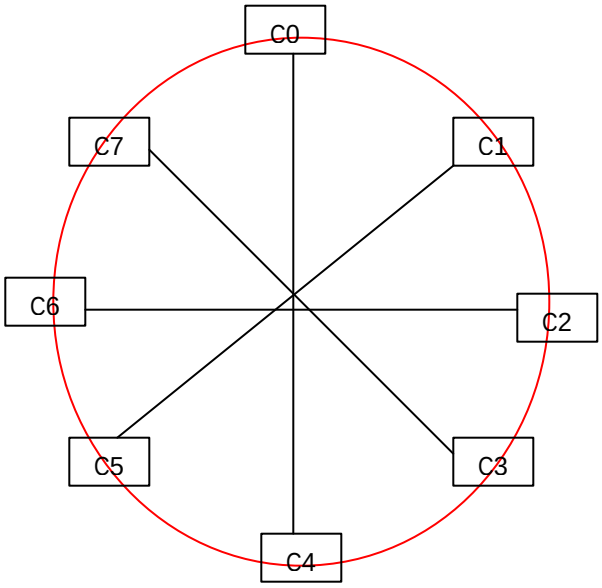

Figure 1. Octagon Interconnect

It is readily apparent that no two network processors are alike. There is still no definitive unified definition for network processors. STMicroelectronics bases its network processor on the functions which it will perform. From the moment a packet or cell arrives, and until it leaves the switch, many functions must be performed on each one. These include framing, classification, encryption, compression, virus scanning, and traffic queuing. With these requirements in mind, STMicroelectronics designed a network processor capable of the task through a combination of speedup techniques, special-purpose instruction-set, and specialized hardware.

Our network processor consists of a few key components, as shown in Figure 2. First, packets arrive through a MAC and get stored into a packet buffer. Only the header is stored here, and the rest of the packet payload is stored in a large external memory. Each packet processing engine will process a packet. When the packet processor is done, it will give back the packet header and pick up the next ready packet header to process.

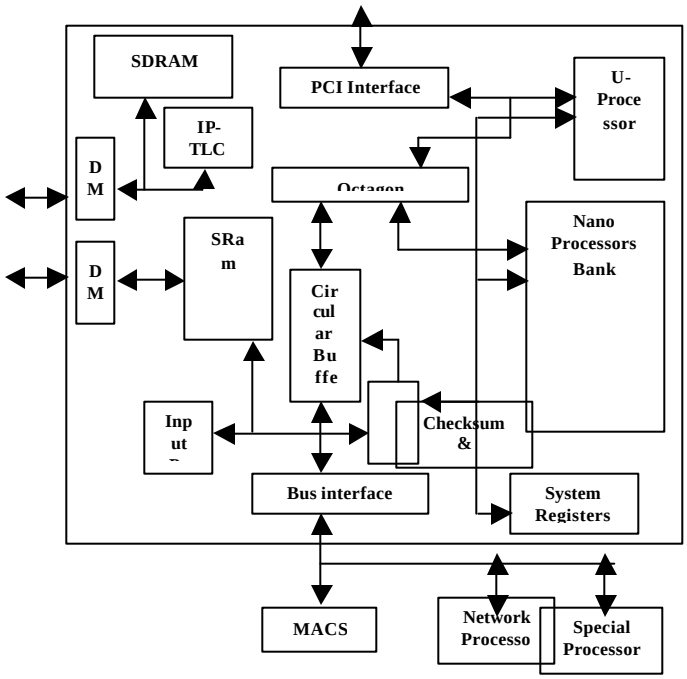

Figure 2. STMicroelectronics NPU Architecture Overview

3. Embedded Software Development Environment

In order to best exploit the inherent capabilities of the optimized network processor architecture, a powerful embedded software development toolset is required in order to support high-level descriptions of the algorithms running on the processor. This ensures time-to-market, code readability and easy portability [4]. Perhaps more difficult is that must be achieved *without* compromising performance [5].

In most of the current offerings on the market, the designer is forced to choose between two alternatives [4]:

1. A standard processor can be chosen in order to ensure good tool support. However, this precludes application-specific instruction-sets to obtain a higher performance solution. In some cases, a standard processor is adopted and extended with specific packet-oriented instructions. In this case, C intrinsics

or in-line assembly are necessary to exploit the new instructions, reducing productivity.

2. Another alternative is a fully dedicated processor offering higher packet processing performance. In this case, only a minimal toolset is usually provided. This has a strong impact on productivity.

Moreover, due to the rapidly evolving nature of the market, where new requirements emerge every few months, the toolset must support rapid processor configuration. Most commercial embedded software environments are developed for a fixed instruction-set architecture (ISA) and are not easily configured to support new instructions, datatypes, memory and/or register file configurations.

3.1 FlexWare: A Retargetable Embedded Software Development Environment

We have based the STMicroelectronics network processor's embedded software toolset on the *FlexWare* embedded software development environment [6], already used within the company for a wide range of general-purpose and application-specific DSPs and MCUs. These are used in consumer audio, wireless telecom, video processing, digital imaging and other applications.

The FlexWare environment is a retargetable suite of four tools:

1. FlexCC: High-performance C compiler technology.
2. FlexSim: High-speed instruction-set simulator model generator.
3. FlexGdb: Source-level debug toolset.
4. FlexPerf: Performance analysis of the embedded S/W and target processor pair.

In addition to FlexWare, the C++-based SystemC simulation environment [14] is used for cycle-accurate micro-architecture models.

In the next sections, we present an overview of the technical characteristics of the FlexWare technologies. In particular, we emphasize the wide range of architectures supported by the FlexWare tools. This allows us to fine tune the processor architecture to our customers' specific needs. This flexibility is a key requirement in the fast changing network processor market environment.

A block diagram of the FlexWare environment is given in Figure 3. A subset of the processor ISA information is captured via the FlexSim tool's graphical user interface. This defines the entire information needed for the instruction-set simulator. It also generates format information for the retargetable assembler/linker, and the performance evaluation tool. The compiler and debugger also require tool-specific targeting files. This targeting information is provided by the tool development experts.

While this approach precludes fully automatic compiler and debugger generation from processor ISA databases, we have found that this is an acceptable price to pay for the resulting high-performance compilers and debuggers, as will be demonstrated in this paper.

All the FlexWare tools are interoperable by construction, but individual tools can be, and have been, linked to other embedded software technologies.

3.2 FlexCC: Compilation Technology

The FlexCC technology is a retargetable C compiler and assembler/linker based on the rule-driven approach described in [7], [8]. The application of this technology to a videophone application [9] was presented in [10]. This involved an application-specific microcontroller and a large instruction word video processor. This technology has since been successfully applied to the following commercial products:

1. A family of application-specific low-cost and low-power DSP's used in low-power telecom, motor servo control and digital consumer applications. Production-strength C compilers have been developed for three members of this DSP family. This has led to the exclusive use of C for all software development. No assembly code has been required.
2. Two generations of a high-performance audio DSP, the MMDSP [11] and the MMDSP+, used for high-volume applications like MP3 players, set-top box, DVD, and more recently, satellite digital radio. See [17] for a description of representative products using this DSP family (e.g STA013, STA014, STA015 audio decoders).

A number of other projects involved the development of beta versions, prototypes and feasibility studies for low-cost general-purpose microcontrollers, DSPs and RISC embedded processors, typically in the early product proposal phase.

Finally, the FlexCC technology was used as a basis for a DSP architecture exploration study. A push-button configurable compiler was developed that supports a number of user-defined parameters. These include register file size and configuration, immediate value formats, and a restricted range of instruction-set permutations. Close to 500 compiler configurations were generated and benchmarked [19].

These applications demonstrate the wide retargetability of the FlexCC technology. The development effort for a compiler ranges between three person-months and one person-year. The bulk of this effort is the validation of the resulting compiler, using industrial ANSI validation suites. The first functional retargeting requires one person-month typically.

Retargetablility is essential, but the performance of the resulting compilers is even more important. A representative benchmark for the FlexCC1 technology is the ETSI C specification of the so-called Enhanced Full-Rate (EFR) codec of the GSM wireless communication standard [15]. This is a complex DSP application of

approximately fifteen thousand lines of C code. The ETSI EFR code was compiled, *without* modifications, on the MMDSP+ single-MAC DSP. The resulting code requires only 53 MIPS to run the application in real time. After seven person-days of optimization of the EFR C code source (without resorting to any assembly code), a figure of 25.5 MIPS was achieved. This is within 15 to 20% of the performance of hand-coded assembly, which can require two or more person years of effort.

3.3 FlexSim: Instruction-Set Simulation

The FlexSim tool takes a high-level specification of the instruction-set written in a proprietary language called IDL (Instruction Description Language), and generates a high-speed functional C model. It also generates the database information needed for the FlexCC assembler, and the instruction-format information needed by the FlexPerf performance evaluation tool. The generated C model is designed for seamless integration with the FlexGdb debugger.

The FlexSim technology has been applied to a wide range of ST processors – close to a dozen in all – ranging from 8 bit microcontrollers to high-performance VLIW DSPs. Retargeting effort varies between two person-weeks for a simple microcontroller, to two person-months for a complex DSP. More importantly, the level of description supported by the IDL language is exactly that of the ISA functional specification. Changes in the ISA specification can be reflected rapidly in the IDL description. Typical model execution speed varies between 200K instructions/sec to over 1.5M instr./sec. These speeds are for an interpreted model, with full interactivity supported (as opposed to a compiled code model which limits interactivity and debug capabilities).

The tool also generates a Tk-based graphical interface for instruction-level interaction with the C model (see Figure 4 in last section for an example). A well-defined API allows to link it with other tools, in particular the FlexGdb source-level debugger.

FlexSim-based models have been used as the 'golden' reference executable specification in most design projects where it was adopted. In the case of an ST advanced DSP design, it was used to validate the C compiler, the applications running on the DSP under design, the hardware implementation in VHDL and was linked to an automatic functional test vector generation tool. Finally, it was linked to the CoWare codesign environment in the context of a rapid prototyping project [18].

3.4 FlexGdb: Source-level Debug Environment

The FlexGdb environment is a source-level debug technology built on the Free Software Foundation GNU public domain debugger [12]. It enriches this technology with support for common extensions needed for embedded processors. These include:

- Multiple memory spaces, such as those used in Harvard-class DSP architectures with X and Y memory spaces.
- Arbitrary width memory addressing unit (MAU). The GNU version supports an 8 bit MAU width only.
- Multiple MAUs on the same processor, for example one for each memory space.
- Paged and segmented memories.
- Special purpose register files, with arbitrary data widths, non-contiguous registers holding long operands, and other application-specific needs.
- In conjunction with the debugger's graphical user interface, support for special purpose datatypes used in embedded applications. An example is the fixed fractional datatype used commonly in DSP's.

The FlexGdb technology has been applied to the MMDSP+ audio DSP. It runs with both the instruction-set simulator and in-circuit emulator. The FlexGdb extensions for complex memories have also been applied to an STMicroelectronics general-purpose microcontroller.

3.5 FlexPerf: Performance Evaluation

The FlexPerf technology was developed to help optimize the embedded-software and target-processor pair. It can be thought of as an instruction-set-aware performance and code profiler. It provides in-depth feedback on the result of the embedded C compiled on the target processor's instruction-set. Two classes of users are supported: the embedded software developer, who needs to understand the relation beween the high-level code and its mapping on the target processor; and the processor architect, in order to fine-tune the instruction-set.

The FlexPerf tool is aware of all the instruction-set, including the instruction word format hierarchy. It is also linked with the debugger symbolic information database. All requests can be done from the C source or assembly source while maintaining the correspondence between the two. For a given program, selected procedure, or set of source code lines, the tool will generate a comprehensive set of information types. These include:

- Instruction count
- Cycle count (for cycle-accurate models)
- Statistics on user-defined classes of instructions
- Waveform analysis
- Processor resource usage analysis

A wide range of graphical feedback utilities have been developed: bar-charts, pie-charts, waveform viewers, and memory access display.

The FlexPerf technology was introduced recently within ST and has been applied to the ST100 advanced DSP [17]. The tool is currently in use by the architecture

team to help evaluate and define future evolutions of this DSP.

3.6 Application of FlexWare to the STMicroelectronics Network Processor

A FlexWare-based embedded software development toolset for the STMicroelectronics NPU is currently underway.

A FlexSim-based model is used as the golden reference functional ISA model. It is the first executable specification of the processor and is used to validate the compiler and run representative applications. It also serves as a functional reference for the cycle-accurate model written in SystemC [14]. The FlexSim graphical front-end allowed to capture a first complete model in less than one person-month. This includes basic unit testing. Furthermore, the high-level constructs allow to track most architecture changes in a day or less. After an additional person-month of validation, this has led to a model robust enough to be used for the evaluation, benchmarking and validation of the C compiler. The FlexSim-based model performance is over 200K instructions/second. In comparison, a hand-coded C++ model runs at 15K instructions/sec. The FlexSim GUI is shown in Fig. 4.

A FlexCC-based compiler has been developed and supports all integer datatypes used in the architecture, including all high-level ANSI C constructs (e.g. structures). An assembler was automatically generated from the FlexSim ISA database. The first working compiler version required less than one person-month effort. This was used as a basis for the first joint architecture-compiler evaluations. This led to a number of enhancements of the processor architecture, as explained below.

The FlexGdb and FlexPerf retargeting process is currently underway.

Lessons learned:

A key lesson so far is that the concurrent development of the C compiler and the architecture has proven essential. This was our experience for other DSP and MCU architectures, and it is reconfirmed for the NPU class. This is especially true when considered in the context of a high-level language driven embedded software development process, where the target is near 100% C code usage.

The *early* availability of a high-performance C compiler raised a lot of issues and allowed us to make the right decisions early on in the design. Many instructions that seemed essential in the specification phase were later demonstrated to be unexploitable by the compiler, or redundant from a compilation perspective.

For example, when two or more code sequences performing the same function have the same performance, the compiler will systematically choose one of them. This often leads to instructions that are never used, and can therefore be removed from the instruction-set without performance penalty. Since instruction-word width has a determinant effect on cost and power, the effect of removing non-essential or redundant instructions can be considerable.

Another change driven by compilation considerations was data memory access granularity. A byte-addressable scheme was chosen over a 32 bit word granularity, based on compiler-driven considerations.

Finally, the key network-specific instructions, which are central to the processor's packet processing performance, are carefully chosen and tuned for the *combined* compiler-processor performance. This involves many trade-offs, some of which are counter-intuitive. Once again the early availability of the compiler is essential in achieving these.

3.6 FlexWare Outlook

We are currently in the final phases of productization of the second generation of the FlexWare environment. On the compilation front, this involves new compilation optimization modules developed by STMicroelectronics. These modules are built on top of the CoSy compiler infrastructure of ACE Associated Compiler Experts bv [13]. The resulting FlexCC2 compiler technology is currently in the final productization phase.

A significant benchmark result for the single MAC MMDSP+ was obtained using the FlexCC2 beta version currently under development. FlexCC2 was applied to the ETSI EFR code example, *without modifications* to the source. The code generated requires 24.5 MIPS of the MMDSP+ to run the application in real time. This is already within 10~15% of the performance of hand-coded assembly on this class of DSP. Moreover, we are currently completing the development of a new register allocation module, which will improve on these already significant results.

On average, for a large number of benchmarks, including those of the EEMBC [16], we have found a 1.4X to 2X performance improvement over FlexCC1, with a 10% code size reduction.

The FlexCC2 technology will be applied to the STMicroelectronics NPU in mid-2001.

A new version of FlexSim is also currently under development. It will support cycle-accurate simulations of the processor micro-architecture and it's peripherals. We will evaluate the use of the FlexSim2 technology in comparison with the existing SystemC microarchitecture model.

4. Conclusion

The fast changing telecom networking environment requires high-performance yet flexible packet-oriented processing capabilities. We have presented the characteristics of some illustrative network processing units (NPU) currently available. We also presented the key features of the NPU developed in the Central R&D organization of STMicroelectronics, where the ISA was developed from the ground up for high-performance packet processing and complex software applications.

The STMicroelectronics *FlexWare* embedded software environment was presented and it's wide range of retargetability demonstrated – ranging from simple microcontrollers, to complex multimedia DSPs. In spite of the wide range of applicability, we have demonstrated state-of-the-art tool performance and functionality. This technology is the basis of our NPU embedded S/W toolset, which will allow us to address the rapid changes in networking needs and customer requirements.

References

[1] see Intel web site: http://www.intel.com
[2] see MMC web site: http://www.mmc.com
[3] see IBM web site: http://www.ibm.com
[4] N. Cravotta, "OC-48, OC-192 and beyond", *EDN Magazine*, Nov. 9, 2000, pp. 61-68.
[5] L. Gwennap, "NPUs tease, don't deliver", *Electronic Engineering Times Magazine*, Nov. 20, 2000, p. 49.
[6] P. .G. Paulin, C. Liem, T. May, S. Sutarwala, "FlexWare: a Flexible Firmware Development Environment", in Code Generation for Embedded Processors, P. Marwedel, G. Goossens (editors), Kluwer Academic Publishers, 1995.
[7] R. P. Gurd, "Experience Developing Microcode Using a High-level Language", *Proc. of the 16th Annual Microprogramming Workshop*, Oct. 1983, pp.2-9.
[8] C. Liem, P. G. Paulin, "Compilation Techniques and Tools for Embedded Processor Architectures", in Hardware/Software Co-Design: Principles and Practice, J. Staunstrup, W. Wolf (editors), Kluwer Academic Publishers, 1997.
[9] M. Harrand et al, "A Single Chip Videophone Video Encoder/Decoder", *International Solid State Circuits Conference*, Feb. 1995, pp. 292-293.
[10] C. Liem, P. G. Paulin, A. Jerraya, "Industrial Experience Using Rule-Driven Retargetable Code Generation for Multimedia Applications", *Intl. Symposium on System-Level Synthesis*, Cannes, Sept. 1995.
[11] L. Bergher, X. Figari, F. Frederikson, M. Froidevaux, "MPEG Audio Decoder for Consumer Applications", *Proc. Of CICC*, 1995.
[12] see Free Software Foundation website: http://www.fsf.org
[13] see ACE website: http://www.ace.nl
[14] see Open SystemC web site: http://www.systemc.org
[15] see ETSI web site: http://www.etsi.org
[16] see EEMBC web site: http://eembc.org
[17] see STMicroelectronics web site: http://www.st.com
[18] R. Hersemeule et al, "Fast Prototyping: A System Design Flow Applied to a Complex SoC Multiprocessor Design", *Proc. Design Automation Conf.*, Jun 2000.
[19] P. G. Paulin, "A Flexible HW/SW Development Environment and its Application to Consumer Multimedia Product Designs", *Pres.CODES/CASHE*, Seattle, Jan. 1999.

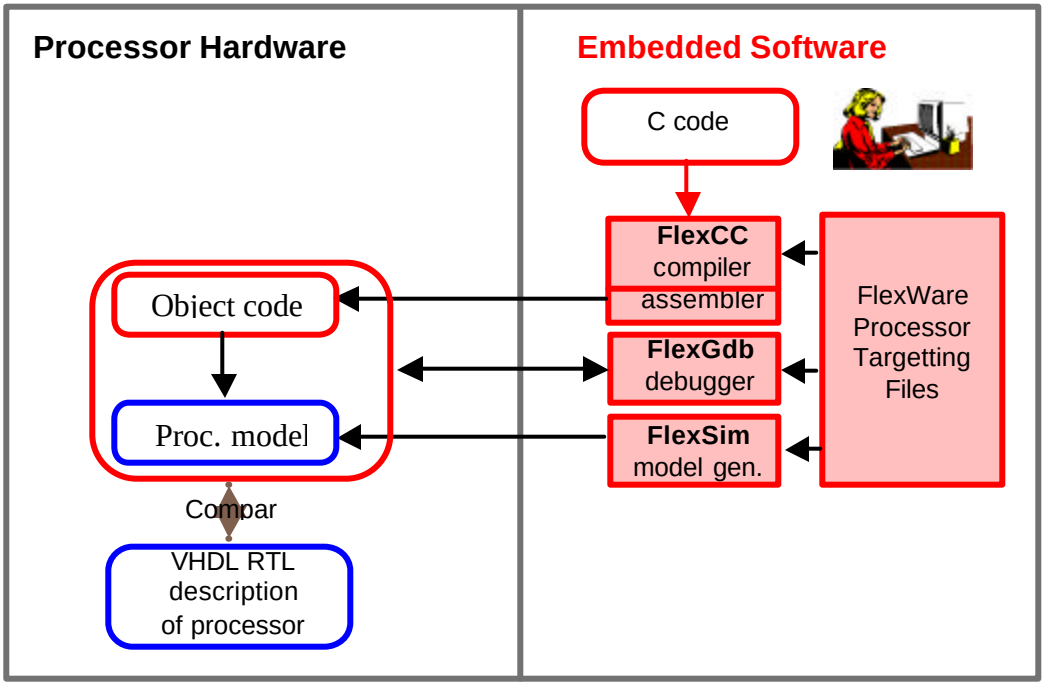

Figure 3. FlexWare Overview

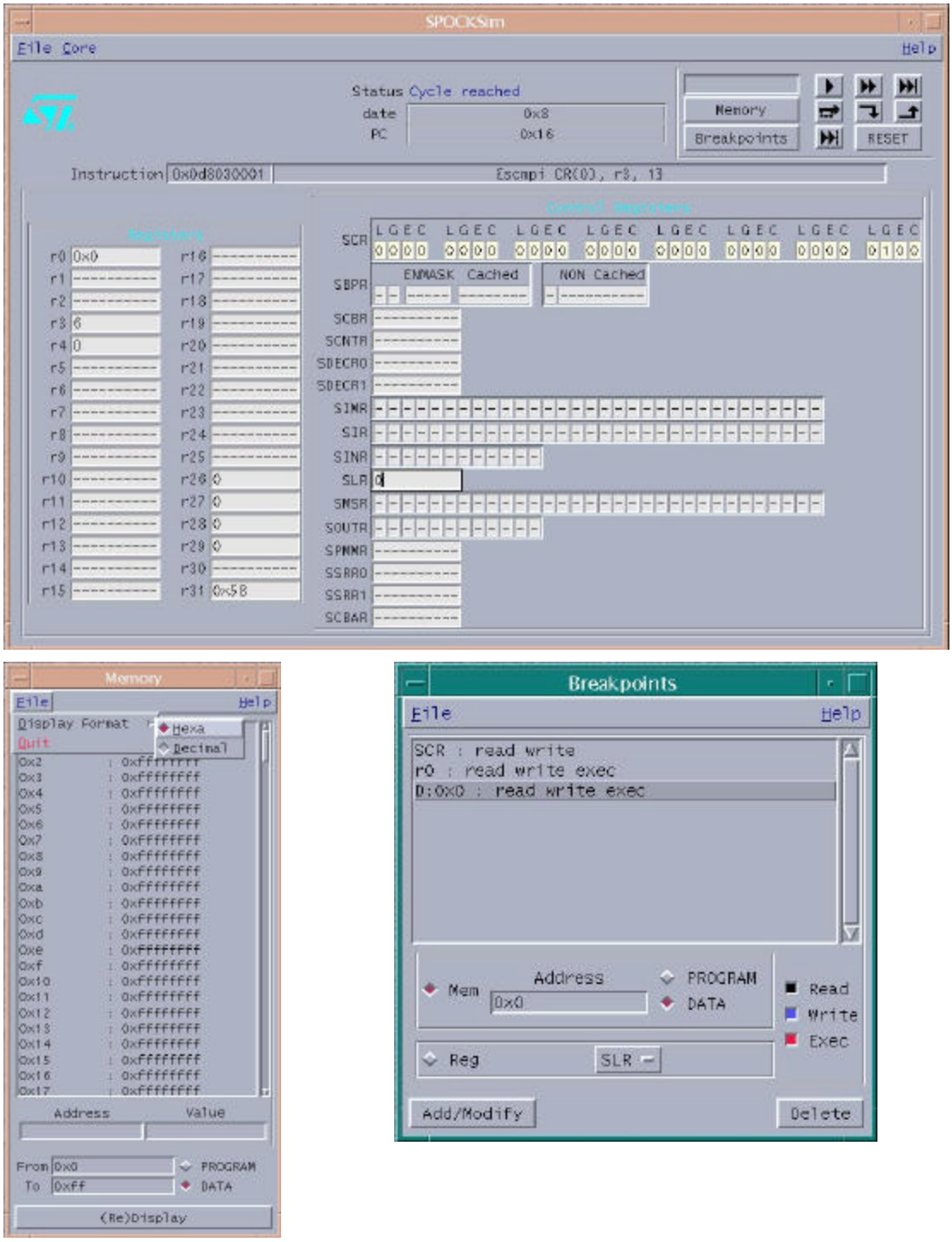

Figure 4. FlexSim user interface for instruction-level debug