Automated Processor Generation for System-on-Chip

Chris Rowen
Tensilica, Inc.
rowen@tensilica.com

Dror Maydan
Tensilica, Inc.
maydan@tensilica.com

Abstract

New application-focused system-on-chip platforms motivate new application-specific processors. Configurable and extensible processor architectures offer the efficiency of tuned logic solutions with the flexibility of standard high-level programming methodology. Automated extension of processor function units and the associated software environment – compilers, debuggers, simulators and real-time operating systems – satisfies these needs. At the same time, designing at the level of software and instruction set architecture significantly shortens the design cycle and reduces verification effort and risk. This paper describes the key dimensions of extensibility within the processor architecture, the instruction set extension description language and the means of automatically extending the software environment from that description. It also describes two groups of benchmarks, EEMBC's Consumer and Telecommunications suites, that show 20 to 40 times acceleration of a broad set of algorithms through application-specific instruction set extension, relative to high performance RISC processors.

1. The SOC Platform Paradox

Two major shifts – one technical, one economic – are changing the design of electronic systems. First, continuing growth in silicon chip capability is rapidly reducing the number of chips in a typical system, and magnifying the size, performance and power benefits of system-on-chip integration. Second, many of the fastest-growing electronics products demand ever-better cost, bandwidth, battery life, and software functionality. These systems – network routers, MP3 players, cell-phones, home gateways, PDAs, and many others – require both full programmability (to manage complexity and rapidly evolving requirements) and high silicon efficiency (for superior application performance per watt, per dollar and per mm^2). In other words, these new systems need new system-on-chip platforms – silicon designs and software environments that are simultaneously flexible and optimal in the application.

The demand for application-specific processors creates a paradox for modern system design: how do architects develop new processors that combine the key benefits of generic programmable chips – longevity, development costs amortized over large volume, adaptability to changing market requirements – with the exactly the instruction set, interfaces and software support optimized for the application. In other words, how do they get the advantages of standard processors without giving up the advantages of custom logic? If the cost of fashioning new optimized processors could be radically reduced, then a much broader array of highly refined processor cores could be used in system-on-chip designs.

Tensilica enables rapid design of highly efficient processor cores by providing a base architecture, a lean core implementation, and an automated method to seamlessly extend the processor hardware and software to fit each system's application requirements. Processors extended by this methodology close the performance gap between high-overhead general-purpose programmable processors and efficient, specialized hardware-only solutions based on hardwired-data-path-plus-state-machine logic functions [1]. This methodology also closes the design gap between the rapid, exponential growth of silicon capacity and the slower growth in designer productivity [2]. This paper outlines the capabilities of Tensilica's Xtensa processor generator [3], including the Tensilica Instruction Extension (TIE) methodology and demonstrates a resolution to the paradox.

2. Configurable Processor Basics

Every Tensilica processor is a superset of the baseline Xtensa instruction set processor. This processor supports a complete 32 bit RISC programming model, implemented with roughly 80 base instructions and requiring just over 25,000 gates for its basic VLSI implementation. The instruction set is encoded in a combination of 16 bit and 24 bit instructions leading to very dense code and low instruction memory requirements. In a generic 180nm CMOS foundry technology, the design runs at over 200MHz under worst-case process and operating conditions.

Configurable processor technology focuses on the automatic generation of new platform-specific processor

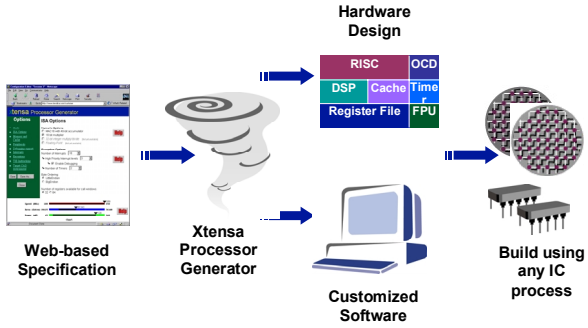
hardware design and software tools from a unified high-level description. By automating hardware and software generation, the platform designer is guaranteed consistency among all the representations of the processor definition – the hardware design, test-benches, simulation models, compilers, assemblers, debuggers and real-time operating systems (RTOSs). All representations are generated together for each architectural variant; all are guaranteed consistent and compatible with one another. This eliminates the potential "Tower of Babel" that surrounds many embedded processor environments today, where subtle differences among members of a processor family – differences in instruction set variations, memory system organization, debug facilities and processor control structures – frustrate the system designer integrating the hardware, verification and software tools. By generating the processor from a high-level description, the platform designer regains control over all the relevant cost, performance and function attributes of the processor subsystem, without having to become a microprocessor design expert. This effectively opens up processor design to a broader population of system and application architects, just as the proliferation of ASICs and logic synthesis tools democratized IC design in the past decade. The four key questions for configurable processors are these:

1. What characteristics of the processor can be configured?
2. How are the characteristics captured by the platform designer?
3. What are the deliverables – the hardware and software components – to the platform designer?
4. What are typical results for building new platforms to address emerging "post-PC" communications and consumer platforms?

The goal for configurability is to allow features to be added or adapted in any form that optimizes the cost, power and application-performance of the processor. In practice, this can be broken into four categories, with examples:

Instruction Set	Memory System	Interface	Peripherals
• ALU functions on general registers • Coprocessors with new application-specific data-types • High performance parallel arithmetic and DSP	• Instruction cache size, associativity, line size • Data cache size, associativity, line size, write policy • Memory protection, translation • Instruction, data RAM, ROM size, address range	• External bus width, protocol, address maps • Direct connection of system registers, queues, multi-ported memories to internal data ports • Multiprocessor interconnect • JTAG debug and trace ports	• Timers • Interrupt controller: interrupt count, priority, type, fast switching registers • Exception vector addresses • Hardware breakpoint controls

The basic processor generations flow is shown below:



The chip designer or application expert comes to a secure web-based generator interface, and selects or describes the instruction set options, memory hierarchy, closely-coupled peripherals and external interfaces required by the application. The generator produces both the complete synthesizable hardware design and the software environment in about an hour. The hardware can be immediately integrated into the rest of the system-on-a-chip ASIC design. It is easily portable to any fabrication process, ensuring optimal performance and silicon leverage. The software environment includes C and C++ compilers, an assembler, a debugger, a cycle-accurate simulator, runtime libraries and popular real-time operating systems. The software development and tuning can also start immediately. With the integrated profiler and one-hour turn-around for software tools and RTOS, the designer can, for the first time, realistically tune the processor to fit the application.

3. Configuration via Web GUI

Two levels of configurability are provided – the web-based graphical user interface (GUI) and the Tensilica Instruction Extension (TIE) language. The Web GUI provides a series of configuration pages through which a wide assortment of processor options can be selected. Typical selections include the following:

- Target process type and operating conditions (e.g. 0.25, 0.18; typical, worse-case
- Optimization priorities (speed, power, area)
- Standard instruction set extensions:
  16b, 32b fast multipliers
  Boolean condition codes
  IEEE floating point coprocessor
  Low-end single MAC DSP coprocessor extensions
  High-end Vectra quad MAC DSP coprocessor
  Extra ALU ops: leading-zeros, sign-extend, min/max
  Size of register file and implementation type
- Number, type and priority of interrupts (up to 32 interrupts on six priority levels)
- Processor read and write bus width (32 to 128b)
- Instruction cache size, associativity and refill size
- Instruction RAM and ROM sizes and base addresses
- Data cache size, associativity and refill size
- Data RAM and ROM sizes and base addresses
- JTAG-based debug control unit

- Real-time trace port
- Hardware data and instruction address breakpoints
- Exception and interrupt vector address
- Target RTL language (Verilog or VHDL)
- Target synthesis, place-and-route, power optimization and test generation tools
- Target real-time operating system
- Target hardware/software co-design environment for Bus Functional Modeling

The GUI provides real-time feedback on estimated gate count, core and memory die area, power and clock frequency for each configuration. This approach gives designers instant access to huge number of possible processor configurations with minimal effort. The configuration GUI is typically used in conjunction with the rapid software simulation and profiling tools bundled with each generated processor. The profiler gives rapid feedback on actual application throughput and helps the system architect select the combination of instruction set, memory system, peripherals and interfaces to meet the system performance requirements at minimal cost and power.

4. Configuration via TIE

The combinatorial possibilities offered by the processor generator GUI are enormous, but many important computational problems offer unique opportunities for application-specific instruction sets. The TIE language lets system designers formally specify their own extensions to the Xtensa core processor. Designer-defined instruction-set extensions captured as TIE work seamlessly with the configuration options selected from the Web GUI.

A TIE description consists of three basic parts:

- **State declarations and types:** Designers may add state registers and register files of any width and number. New C or C++ data-types can be associated with new register files.
- **Instruction encodings and formats:** Designers may specify new formats with up to 6 source and destination registers, including encoded immediate fields. Each new instruction gets a unique encoding
- **Instruction semantics:** For each group of instructions and data-type, designers specify the corresponding transformations from source registers or memory to destinations registers or memory. The designer may optionally specify pipeline latency, used by the compiler, simulator and hardware generator to automatically stretch complex functions across multiple clock cycles.

The TIE language is sufficiently general to efficiently describe almost any instruction set function. In fact, all of the instruction set options available under the GUI, including the sophisticated floating point and Vectra DSP options are implemented entirely in TIE.

A Simple TIE example

Here is a simple, but complete, TIE example of adding a set of instructions to the processor. It defines a new C data-type “long128”, and associates with it a 16 entry register file where is each entry is 128b wide. The instructions that use the new register file include loads and stores, by default, plus one arithmetic instruction “add128”. This instruction is pipelined so the 128b operation does not impact the clock frequency of the processor.

```
;use generator;
;generator::regfile(name => "L",
;                    cname => "long128",
;                    sname => "s",
;                    width => 128);
opcode add128 op2=0 CUST1
iclass ipv {add128} {out sr, in ss, in st}
reference add128 { assign sr = st + ss;}
schedule c2 {add128} { use ss 1; use st 1; def
sr 2;}
```

The software developer might directly use the new data-type and operations as follows:

```
main() {
int i;
long128 source1[256], source2[256], dest[256];
for (i=0; i<256; i++)
  dest[i] = add128(source1[i], source2[i]);
```

5. Hardware Support

The Xtensa processor generator produces a complete hardware design, verification environment and VLSI implementation automation flow. Deliverables include the Verilog or VHDL source RTL for the processor, the hardware test-bench, complete diagnostics, including coverage of TIE extensions, and scripts for synthesis, place and route, test generation, timing and power optimization and simulation.

Support for TIE-based functions goes beyond simple transcription from TIE into equivalent combinatorial RTL functions. Many of the most interesting instruction sets require complex functions. Rather than require the designer to manually pipeline TIE functions, or to slow the entire processor down to longest path in TIE, the hardware generator automatically pipelines the TIE function units. In addition, it also identifies all possible pipeline interactions among instructions and implements pipeline interlocks and result bypass logic to minimize effective latency for complex pipelined functions.

6. Software Support

The Xtensa processor generator produces software support for each configuration as complete as software developed for traditional, non-configurable, systems. In some ways, in fact, it provides better support. In a traditional processor, software developers are often forced to adapt their algorithms to constraints imposed by general-purpose programming languages targeted for

general-purpose hardware. In contrast, configurable processor users can design their hardware and software development system together to better match the underlying algorithm. When a user adds a custom TIE instruction, the Xtensa C/C++ compiler, assembler, simulator, debugger, operating systems and application libraries are all automatically modified to support and exploit the extended architecture.

The Xtensa software system supports configurability from appropriate features in TIE through the generation of dll's (dynamically linked libraries) and Xtensa target code from the TIE description. When a new instruction is added, the TIE compiler generates dll's that describe the TIE instructions, typically in 30-60 seconds for large extensions. This speed is essential to support rapid instruction set architecture tuning and experimentation. Every TIE instruction is directly accessible in C or C++ via an intrinsic function. In addition, the TIE language allows users to define new C datatypes that are mapped to TIE register files along with instruction sequences to load and store these datatypes from and to memory. The C/C++ programmer can use these types as if they were built-in data-types, declaring scalar variables, arrays or structures of them. Data operations are described via intrinsics, but register allocation, instruction sequences for loading and storing new datatypes, addressing arithmetic and control flow generation are all handled automatically just as native datatypes are. The software system automatically extends the cycle-accurate instruction-set simulator via dlls. Similarly, full visibility of all extended register state is incorporated into the debugger and all new instructions are supported by the assembler and disassembler.

An increasing fraction of system-on-chip platforms use standard real-time operating systems. With conventional processor development, the RTOS developer must expend significant time to adapt the runtime environment and development tools to each new architectural variant. Even the addition of a single register or a change in memory organization may trigger a six to twelve month development, validation and release cycle. Xtensa is the first processor technology to fully automate RTOS adaptation for the most important RTOS environments. From the configuration interface, system architects can define the memory map, add new memories, interrupt levels, register files and instructions. The third-party software development environments are automatically extended to support compilation, assembly, and kernel-aware debugging. Interrupt and exception handlers, diagnostic routines and kernel/user code and data segments are all configured and located in the correct regions of the address space. The TIE compiler also automatically generates operating system context switch code using the load and store instruction sequences for new datatypes, mentioned above, further automating software platform delivery. Commercial operating systems, such as Wind River's VxWorks™, are delivered pre-built with hooks in their context switching code to call the routines generated by the TIE compiler. This abstraction of all configuration-specific details makes RTOS porting both simpler and configuration-independent.

The compiler also supports vectorization, which allows ordinary scalar C code to fully exploit SIMD (single instruction, multiple data) or vector extensions, such as the TIE-based Vectra™ DSP engine, a vector DSP coprocessor. The compiler is able to automatically vectorize C code regardless of how many vector elements are configured into each Vectra register. Tensilica also provides optimized hand-tuned application libraries including FFTs, filters, convolution decoders, and other routines. Those routines are also automatically customized for each configuration of the Vectra engine.

7. Application Examples

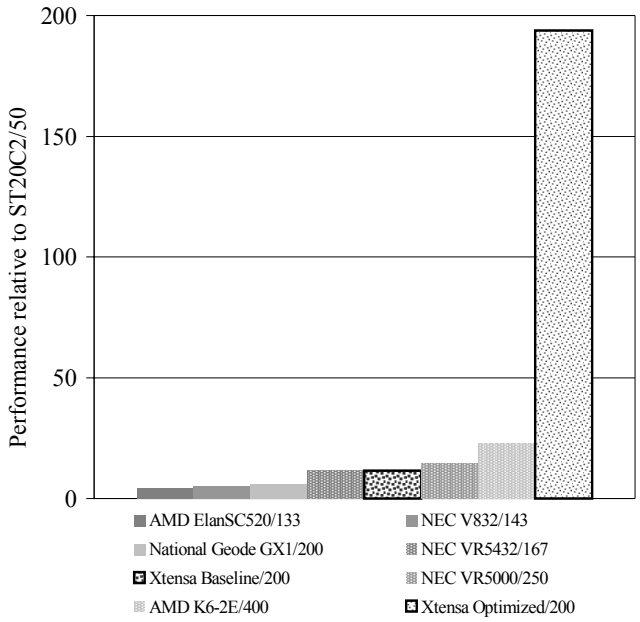
Two sets of representative application kernels – one for consumer devices and one for telecommunications – help illustrate the impact of TIE-based configurability on application performance. The process of tuning these two suites of algorithms parallels the typical use of extensible processors – each of the algorithms is complex and a single processor is tuned for enhanced performance across the whole mix of tasks. Moreover, the twenty separate applications or test-cases in these two suites were all ported, analyzed and used to drive processor configuration over a period of just eight weeks by one engineer, using Tensilica's standard tools.

Example 1: Consumer Multimedia

Video processing lies at the heart of consumer electronics – in digital video camera, in digital television and in games. Common tasks include color-space conversion, two-dimensional filters and image compression. The industry-standard, independently certified, EEMBC Consumer benchmark suite includes a representative sample of all these applications [4]. A baseline configuration of Tensilica's Xtensa processor already includes many appropriate features for these tasks. Even this baseline configuration at 200MHz delivers performance more than eleven times the performance of a basic RISC processor (ST20@50MHz), and on par with popular high-end 32-bit and 64-bit stand-alone processors, where performance is measured as the geometric mean of the relative number of iterations per second through each algorithm. However, when instructions for image filtering and color-space conversion (RGB-to-YIQ and RGB-to-CYMB) are added using TIE, the average performance is increased by a further 17 times, resulting in a processor with almost 200 times the performance of the reference processor as shown in Figure 1. The base configuration was optimized for 200MHz worst-case performance in 0.18µ CMOS technology and utilized 16KB two-way set associative caches, 256KB local data RAM, a 16-entry write buffer and a 32-bit multiplier for a total of 57,600

gates of logic. The optimized results used software that exploited the additional 64,100 gates of extensions implemented in TIE.

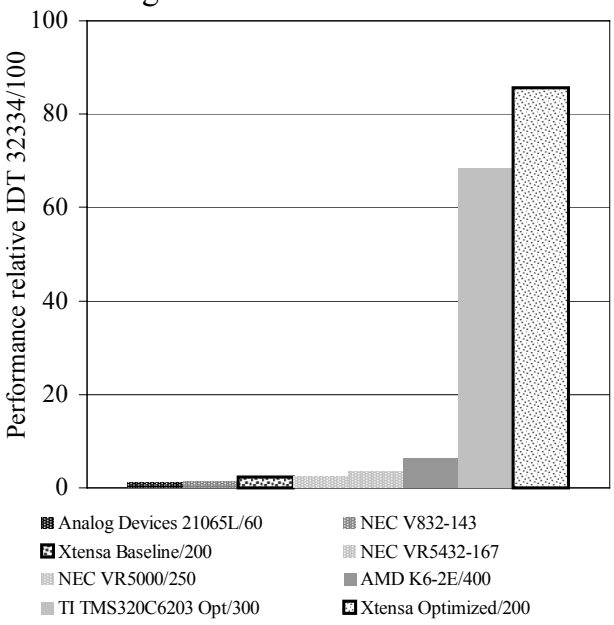
Figure 1: EEMBC Consumer Suite



Example 2: DSP Telecommunications

Telecommunications applications present a different set of challenges. Here the data is often represented as 16-bit fixed-point values, as densely compacted bit-streams or as redundantly encoded channel data. Over the past ten years, standard DSP processors have evolved to address many of the requirements of filtering, error correction and transform algorithms. The EEMBC "Telemark" benchmark includes many of these common tasks. In this case, the applications designer might start with a standard Xtensa configuration. This gives baseline performance on the EEMBC benchmarks that compares well with other leading 32-bit and 64-bit RISC processors, where performance is measured as the geometric mean of the relative number of iterations per second through each algorithm compared to the reference processor (IDT 32334™ – MIPS32 architecture - at 100MHz). However, when additional features are added, including the Vectra DSP co-processor and a few additional instructions in TIE, and the code is re-optimized to exploit the new capabilities of the platform, the performance jumps another 37 times. The overall performance then exceeds that of a high-end Texas Instruments TMS320C6203™ VLIW DSP using hand-optimized code, as shown in Figure 2. The base configuration was again optimized for 200MHz worst case performance in 0.18µ CMOS technology and used 16KB two-way set associative caches, 16-entry write buffer, but not the Vectra extensions. The optimized code utilizes Vectra and 18,000 additional gates of TIE for a total of 180,000 gates.

Figure 2: EEMBC Telecom Suite



8. Conclusion

System-on-chip integration offers significant improvements in system bandwidth, cost and power efficiency, compared to systems built with discrete semiconductor building blocks. These application-specific system ICs can take full advantage of the efficiency benefits of application-specific processors, but only if designers can generate new processors quickly and completely. Automatic generation of optimized processor core hardware and of the associated software tools sharply reduces the time, cost and risk in development of new processor-based platforms, and eases the integration of processors into system-on-chip designs. New methodologies, tools, and processor foundations are required for this shift to application-specific processors, but these methods are triggering an explosion of new silicon platforms that combine the flexibility of standard programming models and the efficiency of application-tuned silicon.

Processor configurability has many dimensions — sizing of memories, addition of specialized external interfaces, and incorporation of closely coupled peripherals. The single most important dimension of configurability is instruction set extension. This paper has outlined the mechanisms for instruction set extensibility, the Tensilica Instruction Extension description format and techniques for full configuration of compilers, simulators, RTOS and RTL implementation. The performance impact is significant, often averaging more than ten-fold that of current

implementations of traditional, fixed-instruction set, general-purpose processors.

9. Acknowledgements

The authors wish to thank the entire technical staff of Tensilica, whose efforts underlie the Xtensa Processor Generator, the foundation of this work. Special recognition is due to Michael Carchia, who did all the EEMBC benchmarking, and to Albert Wang, Earl Killian, Kim Alfaro and Pavlos Konas who reviewed the manuscript.

10. References

[1] N. Zhang and R. W. Brodersen, "Architectural Evaluation of Flexible Digital Signal Processing for Wireless Receivers," Proc. Asilomar Conf., Pacific Grove, CA, October 2000

[2] R.G. Bushroe et al., "CHDS: A Foundation for Timing-Driven Physical Design into the 21st Century," SEMATECH, Inc.

[3] R. Gonzalez, "Configurable and Extensible Processors Change System Design". Hot Chips 11, 1999. ftp://www.hotchips.org/pub/hotc7to11cd/hc99/hc11_pdf/hc99.s4.3.Gonzalez.pdf

[4] http://www.emmbc.org