Diamond Standard Processor Cores

A comprehensive family of software-compatible, preconfigured RISC controller cores for your next ASIC or SOC design

Tensilica's Diamond Standard Series is a broad family of preconfigured 32-bit microprocessor and DSP Intellectual Property (IP) cores based on Tensilica's Xtensa Instruction Set Architecture (ISA). All of these code-compatible processor cores employ a common set of software/firmware development tools making it easy for development teams to move from one processor core to another as design needs change or when your design needs multiple cores to execute different tasks. The base Xtensa ISA employs 24-bit, general-purpose RISC instructions that target a wide range of embedded applications. Most common instructions also have a 16-bit narrow encoding to minimize code footprint and the Diamond Series architecture allows modeless switching between 16 and 24-bit instructions. Consequentially, Diamond Standard Series processor cores achieve the highest code densities among all 32-bit RISC processors while delivering industry-leading performance.

Introduction

Tensilica's Diamond Standard Series is a family of code-compatible, preconfigured 32-bit microprocessor and DSP Intellectual Property (IP) cores based on Tensilica's Xtensa Instruction Set Architecture (ISA). The base Xtensa ISA uses 24 bit instructions that target a wide range of embedded applications. Most common instructions in the Xtensa ISA also have 16-bit narrow encodings as well, and the architecture allows modeless switching between 16 and 24-bit instructions so that the compiler is free to pick the smallest possible instruction at any time with no performance penalty. As a result, the Diamond Series processors achieve the highest code densities among all 32-bit RISC processors.

One of the Diamond Standard processors, the 570T high-performance CPU, adds VLIW-style, 64-bit instructions through Tensilica's innovative FLIX (Flexible Length Instruction eXtensions) technology, which allows the processor to issue multiple operations per instruction. These 64-bit instructions are modelessly mixed with the native 16 and 24-bit instructions increase the processor's parallel-execution abilities and further boosts application performance without the code bloat normally associated with VLIW architectures.

This white paper explores the Xtensa instruction set architecture (ISA) and illustrates the impact of processor architecture on performance. It traces the evolution of modern instruction-set design and compares key features of Tensilica's processor architecture with other ISAs. It provides a detailed explanation and rationale for the major architectural innovations in the Xtensa ISA, on which the Diamond Standard Series processor cores are based.

The first section of this white paper gives a quick overview of the Diamond Standard family. The second section outlines the goals, philosophy, and innovations inherent in the Xtensa instruction set. The third section gives a more detailed description, with a block diagram, for each Diamond Standard processor. Finally, the last section gives more information on strength of the Xtensa-based Diamond architecture, taking a look at the superior benchmark performance of the Diamond Standard Series processor cores.

Diamond Standard Family Overview

Tensilica's Diamond Standard processor core family consists of three general-purpose controller cores, a Linux-compatible CPU core, a high-end static superscalar CPU core, a high-performance audio processor core, and a high-end DSP core. Table 1 lists the various Diamond Standard processor cores and gives a quick summary of each core's characteristics.

Diamond Controller Cores	**106Micro**	Smallest 32-bit, ultra-low power, cache-less RISC controller with local memories
	108Mini	Ultra-low power, cacheless controller core with rich interrupt architecture, minimal gate count for lowest silicon cost
	212GP	Flexible mid-range controller core with instruction and data caches and user-defined local memory sizes
Diamond CPU Cores	**232L**	Flexible mid-range CPU with a Memory-Management Unit (MMU) for Linux OS support
	570T	Extremely high-performance, 2- or 3-issue static superscalar processor

Table 1: Diamond Standard Series Controller, CPU, Audio, and DSP Cores

The Diamond Standard controllers and CPUs are optimized control-plane processors that are industry leaders in area, power consumption, code density and application performance. The Diamond 106Micro and 108Mini enable SOC architects to quickly integrate an efficient CPU into their designs. Both cores are small, low-power 32-bit RISC controllers that achieve the performance levels of much larger, more complex CPUs.

The Diamond 212GP CPU is a small, low-power, high-performance controller core with rich interrupt options and a single-cycle 16x16-bit MAC, which can eliminate the need to include a separate DSP in many system designs. The Diamond 232L adds an MMU (memory-management unit) to support the Linux operating system.

The Diamond 570T is a high-performance, static superscalar processor capable of issuing a 64-bit Very Long Instruction Word (VLIW) bundle consisting of two or three operation slots. The compiler automatically assembles the 64-bit, multiple-operation bundles when it determines that instructions can be issued simultaneously. (The compiler can also create a bundle with just one instruction for performance reasons.) Otherwise, the compiler uses the 16- and 24-bit instructions in the base Xtensa ISA.

This instruction-size flexibility results in very little code expansion from the inclusion of VLIW operations. Normally the 'no-op padding" required with older, fixed-length VLIW ISAs produces "code bloat" because all instructions in these older VLIW architectures must be the same large (or very large) size. Consequently, the Diamond 570T code density remains high. It is at least 20% better than competing RISC architectures on industry standard benchmarks.

Figure 1 shows the five controller and CPU members of the Diamond Standard processor core family, the performance of each core in Dhrystone MIPS/MHz, and the area consumed by the core in a standard 90G IC fabrication process.

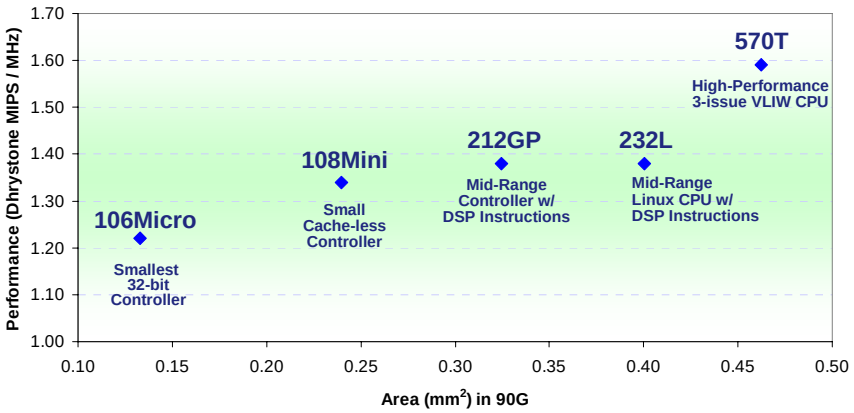


Figure 1: Diamond Standard Processor Cores, Performance, and Area

Software Support

A comprehensive set of software tools is available for the Diamond Standard processors. These tools consist of:

- A software tool suite to match the processor architecture. This tool suite includes the XCC C/C++ compiler, a macro assembler, linker, debugger, and a basic software library. While XCC's operation is similar to the GNU C and C++ compiler (GCC), XCC is an advanced optimizing and vectorizing compiler that provides superior execution performance. XCC generates compact, executable code with a smaller code footprint relative to

other compilers. XCC also bundles multiple operations into VLIW instructions for the multi-issue Diamond 570T processor core.

- Xtensa Xplorer – Diamond Edition (DE), is an integrated development environment based on the industry-standard Eclipse platform. Xplorer DS serves as a cockpit for single- and multiple-processor SOC hardware and software design. Xplorer DS integrates software development and system analysis tools into a common visual design environment that provides powerful graphical visualization abilities and makes creating processor-based SOC hardware and software much easier.
- A cycle-accurate instruction-set simulator (ISS) that enables faster code development, accurate performance modeling, and system-level architectural tradeoffs. The ISS can also be used for code benchmarking.

Feature Summary

All Diamond Standard processors share a common set of 16 and 24-bit instructions that permits code compatibility across the processor core family. Some Diamond processors add VLIW-style 64-bit instructions as discussed above. These VLIW features permit the issue of multiple operations per instructions, boosting the processors' parallel-execution abilities and application performance. The common set of processor core features include:

- Specialized functional units (not on all cores)
- Multipliers: 16-bit MAC, SIMD, VLIW
- Region-based memory protection and a full MMU on Diamond 232L
- Big or little-endian byte ordering
- 5-stage pipeline
- Exceptions:
    - non-maskable interrupt (NMI)
    - nine external interrupts
    - six interrupt priority levels
    - three 32-bit timer interrupts
- 32-entry windowed register file (16 entries for the Diamond 106Micro core)
- Write buffer: 4, 8, or 16 entries (depending on processor)
- Wide variety of interfaces:
    - 32, 64, or 128-bit Processor Interface (PIF) width to main system memory or to an on-chip system bus. Tensilica provides a complete Vera-based tool kit for PIF bridge implementation and verification. (106Micro and 108Mini cores only have local-memory interfaces)
    - Inbound-PIF (e.g., DMA) requests allow external access to the processor's local memories
    - Optional AMBA AHB-Lite interface
    - Direct I/O ports for the Diamond 108Mini, 212GP, and 570T processors
    - Streaming data queue interfaces on the Diamond 570T processor core

- Rich on-chip memory architecture (varies by processor, see Table 2)
- Programmable write-through or write-back cache-write policy
- Cache locking per line for set-associative cache

Memory Type	106Micro	108Mini	212GP	232L	570T
Local instruction RAM* (Kbytes)	1-128	1-128	0-128	N/A	0-128
Local Data RAM0 (Kbytes)	0-128	0-128	0-128	N/A	0-128
Local Data Ram1 (Kbytes)	N/A	0-128	N/A	N/A	N/A
Instruction Cache (set associativity)	N/A	N/A	8KByte (2-way)	16Kbyte (2-way)	16Kbyte (2-way)
Data Cache (set associativity)	N/A	N/A	8KByte (2-way)	16Kbyte (2-way)	16KByte (2-way)
Cache Line Size, I and D cache (Bytes)	N/A	N/A	32	32	32
* Processors with no instruction cache require at least 1KByte local instruction memory since vectors are mapped to local instruction memory due to performance reasons.					

Table 2: Memory Sizes and Details for Diamond Standard Processors.

Processor development and debug capabilities:

- C/C++ callable ISS
- On-Chip Debug (OCD) capability: Trace and instruction/data breakpoint support (two hardware-assisted instruction breakpoints and two hardware-assisted data breakpoints)
- GDB debugger support
- ISS and Co-Simulation Model (CSM) support
- Robust EDA environment support
- Physical synthesis design flow
- Operating system support for Mentor Graphic's Nucleus Plus, Express Logic's ThreadX, Micrium Technologies' µC/OS-II, and Sophia Systems' µITRON.

Code Density

Using the Xtensa ISA, the XCC compiler can generate highly efficient code that has a code footprint that is as much as 50% smaller than with other popular RISC and CISC architectures, as shown in Figure 2. The use of 24- and 16-bit instructions in the Diamond Series processors greatly reduces the size of application code compared to conventional 32-bit RISC code. Small code size helps to reduce on-chip memory requirements.

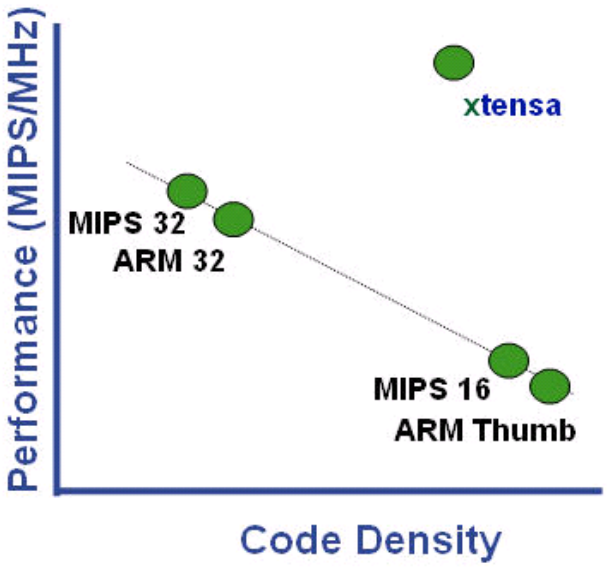


Figure 2. The Xtensa ISA Delivers Smaller Code and Better Performance.

The Xtensa ISA allows the XCC compiler to minimize the program size by reducing both the number of instructions needed to execute the application program and the average number of bits per instruction through the ISA's 16- and 24-bit instruction words, the use of compound instructions, the richness of the comparison and bit-testing instructions, zero-overhead-loop instructions, register windowing, and the use of encoded immediate values.

The Diamond Standard processors have several compound instructions that reduce the instruction count required to encode and execute a program. For example, compare-and-branch instructions constitute the most important class of compound instructions, reducing code size by at least 5%. Other compound instructions include shift, add/subtract, and shift-and-mask.

The Diamond Standard processors (except the Diamond 106Micro and 108Mini cores) employ a feature common to DSPs but not usually offered in general-purpose architectures: zero-overhead loops—the ability to iterate a series of instructions without a branch instruction at the end to loop back. Using this feature, the Diamond Standard processor cores execute loops without stalls caused by mispredicting branches or the need for extra instructions to decrement and test the loop counter. Reducing loop overhead through zero-overhead loops improves performance and reduces code size.

The Diamond Standard processors employ register windows to reduce the number of instruction bits needed to specify a register. Most RISC instructions specify three registers (two source and one destination) and each of these registers must be specified within the instruction. Register windowing results in substantial savings in code size by reducing the number of bits needed to specify a register.

The register windowing in Tensilica's Xtensa ISA support a variable window increment size to allow call levels to completely fit into the Diamond processor's 32-entry general-purpose AR register file, thus minimizing the number of stack operations required to save and restore registers around call sites. Diamond Standard processor cores postpone window overflow operations until absolutely necessary, which results in fewer register-spill traps and smaller code size compared to previous register-window architectures. This feature also produces less memory traffic and smaller code size than architectures that lack register windows.

Principles of Instruction Set Design

After more than half a century of intensive, global research on computer architectures, the design of processor ISAs is a well-established art. Most instruction-set features are not new in themselves, but features can be combined in new and unique ways that advance the state of the art. In particular, significant improvements result when ISAs are optimized for a different use than prior instruction sets.

The ISA designer must balance many competing goals, including:

- The resulting size of the machine code for various algorithms
- The ISA's extensibility and adaptability for new algorithms and applications
- The resulting processor performance
- The resulting power consumption of processors that employ this ISA
- The resulting cost of processors that employ the ISA
- The ISA's suitability for multiple future processor implementations
- The processor's resulting design complexity
- The ISA's suitability as a compiler target for high-level programming languages

The processor's ISA has one direct and two indirect influences on performance. First, the ISA directly determines the number of instructions required to implement a given algorithm. Other components of processor performance include the maximum achievable clock rate and the average number of clocks per instruction. These attributes primarily depend on implementation technology, but ISA features can affect the implementer's ability to simultaneously meet clock rate and clocks-per-instruction goals. For example, a certain choice for instruction encoding might require additional logic in series with the rest of the instruction's execution hardware, which must be addressed either by increasing the clock period or by adding an additional pipeline stage, which increases the number of clocks per instruction (increased instruction latency).

The RISC (Reduced Instruction Set Computing) processor design philosophy emerged in the 1980s. RISC ISAs allow implementers to reduce a processor's CPI (cycles per instruction) count and its clock period significantly. RISC ISAs improve the processor performance, lower design complexity, allow lower cost processor implementations at a given performance level, and are good compiler targets for high-level programming languages.

Curiously, there is no single, completely comprehensive, or even satisfactory definition of the term RISC, but RISC processors typically include:

- Fixed-size instruction words
- 3-operand instruction orientation (two operand sources, one result destination)
- Large uniform register files
- Simple and fixed instruction-field encoding
- Memory access limited to register-to-memory and memory-to-register loads and stores (loads and stores are never combined with computation in one instruction)
- A small number (often 1, usually less than 4) of memory addressing modes
- Avoidance of features that would complicate pipelined instruction execution (no variable instruction latency or microcoded instructions)

Most RISC ISAs are designed for high performance desktop computing environments where a large hard disk storage capacity is a given. These ISAs are not optimized for compact machine code because workstation and server memory is cheap relative to system costs. In particular, most RISC instruction sets usually require more program bits to encode an application than pre-RISC ISAs.

In many embedded applications today, the cost of code storage (on-chip RAM or ROM) is often greater than the cost of the processor (in terms of silicon area or gate count), so the use of RISC processors is sometimes limited in the most cost-sensitive ASIC and SOC designs.

An ISA that combines the advantages of RISC with reduced code size is invaluable in many embedded applications. This combination of goals drove Tensilica's development of the Xtensa ISA.

Why are Xtensa and Diamond Processor Cores Better Suited to Embedded Applications?

The Xtensa architecture, on which the Diamond Standard Series processor cores are based, builds on many of the RISC principles discussed in the previous section, but the Xtensa ISA introduces new ideas and techniques that shrink code footprint by reducing both the number of instructions required to encode a program and the average number of bits per instruction. These techniques improve performance and reduce cost relative to previous architectures.

The Xtensa ISA starts with the premise that it must provide good code density. To achieve exemplary code density, Xtensa processors use a simple, variable-length, instruction-encoding scheme that doesn't compromise performance, which is why RISC processors started using fixed-length encoding in the first place. The Xtensa architecture further optimizes the cost of processor implementation by balancing such features as register files, control-flow operations, arithmetic and logic instructions, and load/store capabilities while favoring operations that occur most frequently in modern embedded software.

Registers

To maintain performance, a RISC ISA must support at least two source register fields and one distinct destination register field. General register instruction sets that optimize only for code density are sometimes designed around two register fields. This design approach can reduce code size, but there is no way to compensate for the resulting increase in the number of instructions required to execute a program. ISAs that specify fewer registers can use narrower register fields and thus save bits per instruction. However, these 2-operand instruction sets increase the number of instructions in the program by forcing more variable and temporary values into the processor's memory, which adds extra load and store instructions for access to operands.

Consequently, this design approach may reduce code size but it increases both the number of cycles for program execution and the power dissipated. As the number of registers in the processor's main register file increases, the marginal benefits of a 2-operand instruction format decline. In particular, many studies have shown that at least 16 general-purpose registers are required for good RISC performance.

The Diamond Standard processor cores employ a windowed general purpose (AR) register file that contains 32 entries (16 entries for the 106micro processor core). Instructions access this physical register file through a sliding 16-register window. Register windowing allows the Diamond processor to have a relatively large number of physical registers while restricting the number of bits needed to encode a source or destination operand address to four bits each. Thus the 3-operand Xtensa instructions need only 12 bits to specify the registers holding the instruction's three operands. This scheme results in compact, efficient instruction encoding while allowing for a larger register file, which results in both compact code and good execution performance.

Register Windows

Register windows reduce code size and improve performance. Register windows are used in a few other processor architectures, such as Sun's SPARC ISA. The name "register window" describes a typical implementation where the register field in the instruction specifies a register within the current window, which is a subset of a larger register file. Register windows avoid the need to save and restore registers at procedure entry and exit. Instead of saving and restoring registers on a stack, a processor with register windows merely changes a register-offset pointer, which

hides some registers from view and exposes new ones. The exposed registers usually do not contain valid data, and can be used directly.

Register windows can overlap in their views of the physical register file, which prevents the argument shuffling that can occur when arguments to procedures are passed in registers. Finally, register windows alter the breakeven point for allocating a variable or temporary to a register, and thus encourage register use instead of needing a memory location, which is slow relative to register storage.

Unlike the SPARC architecture's fixed-window overlap increment, the Xtensa ISA employs a variable increment for register windowing. This feature keeps implementation cost low by allowing a much smaller physical register file to be used for most applications. For example, many Sun SPARC ISA implementations use a physical register file of 136 entries. Xtensa ISA implementations with a 64-entry register file achieve similar performance.

Instruction Width

Prior RISC architectures failed to achieve an appropriate balance between code size and performance because RISC ISA designers felt constrained to certain power-of-two instruction sizes such as 16 and 32 bits. There are indeed advantages to using such instruction sizes. However, relaxing this restriction somewhat has significant advantages that other ISA designers have not explored. Xtensa processors use a 24-bit fixed-length encoding as a starting point; 24 bits are sufficient for achieving high performance while providing extensibility and room for powerful instructions that ultimately decrease the number of instructions required to execute a program. Using 24 bits to encode an instruction instead of 32 and reducing the number of instructions needed to execute a program minimizes code footprint.

The Xtensa ISA's 24-bit encoding represents a 25% reduction in instruction size relative to the more common RISC 32-bit instruction word, which reduces code size requirements relative to most 32-bit RISC instruction sets. In addition, 24-bit instructions are easily accommodated in a processor with 32-bit data-path widths.

The Xtensa architecture uses 4-bit register fields (see Figure 3), the minimum required for acceptable performance and the maximum that fits well within a 24-bit instruction word. Many other RISC ISAs have 32 general-purpose registers and therefore require 5-bit register fields. These processors need 15 bits just to specify two source and one destination operand.

The difference in performance between 16 and 32 general registers is about 5%, which is not as large as the difference between 8 and 16 general registers and is small enough so that other features can be introduced to make up the lost performance (such as the use of compound instructions and register windows, discussed below). The resulting increase in the number of instructions needed to encode a program (also about 5%) is more than offset by the difference between 24-bit and 32-bit encoding (a reduction of 25%).

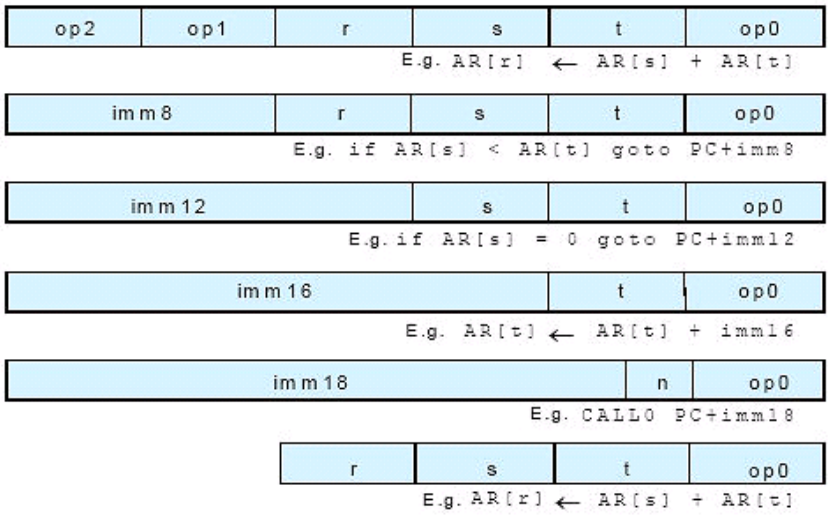


Figure 3: Xtensa Instruction Encoding Formats

Note that many ISAs with 5-bit register fields do not provide 32 general registers for compilation. Most dedicate a register to hold the constant zero, even though the addition of a few extra instruction opcodes can easily eliminate the need for a zero register. (For example, the Xtensa ISA includes a NEG instruction.) Other registers are often given specific uses that can be avoided by including other features in the instruction set. For example, the MIPS architecture dedicates two of its 31 general registers for exception handling and one more register for a global area pointer. So, in effect, the MIPS architecture provides programs with only 28 general-purpose registers for variables and temporary storage, just 12 registers more than an ISA with 4-bit register fields.

The division of general registers into caller and callee saved registers by software convention is common and further restricts the utility of larger register files. The Xtensa ISA includes features that avoid this, which raises the effectiveness of its 16-register window almost to the level of other RISC processors' 32-entry general-purpose register files, but with a considerable savings in silicon area and routing congestion. The Xtensa ISA proves that a 24-bit encoding of a full-featured RISC instruction set is possible.

FLIX 64-bit Instructions

The Diamond Standard 570T processor core takes advantage of Tensilica's unique FLIX technology to schedule multiple independent operations in one 64-bit instruction. These wide-word instruction bundles allow more complex, compound

machine instructions to improve code and application performance without incurring VLIW code bloat.

Unlike fixed-length VLIW ISAs, Tensilica's XCC compiler uses 64-bit FLIX instructions only when needed and only if instructions can be issued simultaneously, otherwise the compiler uses the 16- and 24-bit instructions. This flexibility to intermix 16-, 24-, and 64-bit instructions results is extremely minimal code expansion and avoids the "no-op padding" required with other VLIW ISAs. As a result, code density remains high. The processor modelessly switches between 16-, 24-, and 64-bit instructions, so there is no performance penalty from mode switching because there isn't any such switching.

Compound Instructions

To further improve performance and reduce code size, the Xtensa ISA includes instructions that combine the functions of multiple instructions typically found in RISC and other processor ISAs. For example, one such compound instruction is a simple "left shift and add/subtract." The high-end HP PA-RISC and DEC Alpha ISAs, designed for workstations and servers, also provide such compound operations.

Address arithmetic and multiplication by small constants often use these instruction combinations, so providing these operations reduces instruction count. However, these more complex instructions could potentially increase the processor clock period (reducing the maximum achievable clock rate) because of the logic added to the computation pipeline stage. Various processor implementations have shown that when the shift range is limited 3 bits or less, the extra logic is does not constrain clock frequency.

Right shifts are often used to extract a bit field from a larger word. For unsigned field extraction, two instructions—either left shift followed by right shift, or right shift followed by an AND with a constant—are typically used. The Xtensa ISA provides a single compound instruction, EXTUI (extract unsigned immediate), to perform this function. The EXTUI instruction executes a shift followed by an AND with a specified mask that is encoded in the instruction word as a 4-bit value. The logical AND portion of the EXTUI instruction is so trivial from a hardware perspective that its inclusion in the ISA does not increase the processor's clock period. The same isn't true for an instruction to extract signed fields, so there's no corresponding EXTSI instruction in the Xtensa ISA.

Branches

Most processor ISA, RISC and otherwise (for example ARM, DEC PDP11, DEC VAX, Intel x86, Motorola 68000, Sun SPARC, and Motorola 88000) implement branches with two instructions: a compare instruction that sets condition code(s) followed by a conditional branch instruction that tests the condition code(s) to determine program flow. Conditional branches constitute 10-20% of the instructions

in most RISC instruction sets, and each instruction is usually paired with a compare instruction. This branching style is wasteful. Some instruction sets—including the CDC 6600, the Cray-1, MIPS, DEC's Alpha, HP's PA-RISC, and Sun's SPARC V9—provide an improved method for branching that uses a compound compare-and-branch facility of varying flexibility.

The Xtensa ISA includes the "most useful" compound compare-and-branch instructions. Choosing the exact set of instructions that constitute the "most usefule" set requires balancing the utility of each compare-and-branch instruction with the opcode space that it consumes. This is an especially important consideration when you only have 24 bits (as opposed to 32 bits) for instruction encoding.

Compound compare-and-branch instructions reduce instruction count when compared with instruction sets that have separate compare and branch instructions and even with respect to the partial compare-and-branch instructions in the MIPS and DEC's Alpha ISAs.

The Xtensa ISA's compare-and-branch instructions also support comparisons to immediate values and use clever encoding of constants to increase their utility. The BEQI, BNEI, BLTI, BGEI instructions use a 4-bit field that encodes various common constants. The BLTUI and BGEUI instructions use a different encoding, because unsigned comparisons have a different set of useful values

The Xtensa processor's compound compare-and-branch instructions pack these immediate values, two source-register fields, and an 8-bit PC-relative offset target specifier into a 24-bit instruction word. The 8-bit relative target specifier will be too small in some infrequent cases. In such cases, the compiler or assembler compensates by using a conditional branch of the opposite nature to branch around an unconditional branch instruction, which has a longer range.

The Xtensa ISA also provides a series of compound compare-and-branch instructions that test against zero, the most common case. These compound compare-and-branch instructions have an implicit immediate value (zero) so the four immediate-operand bits are combined with the 8-bit offset to create a 12-bit PC-relative offset field, which provides much greater range.

The Xtensa ISA uses an additional method for allowing coprocessor conditional branches. The Xtensa ISA offers an option that adds 16 1-bit Boolean registers. The Xtensa ISA's BF (branch if false) and BT (branch if true) instructions test these Boolean registers and branch accordingly. Xtensa ISA instructions can set the Boolean registers based on comparisons of their supported data types. All Xtensa processors share the baseline ISA's Boolean register set and the BF and BT instructions. This approach makes efficient use of the Xtensa ISA's short, 24-bit instruction word.

This scheme is a new variant of compare-and-branch condition codes found in many earlier processor ISAs. The use of single-bit (Xtensa, MIPS) instead of multi-bit comparison-result registers (most other ISAs) increases the number of comparison opcodes required but decreases the number of branch opcodes required. This ISA

design approach also makes the introduction of a broad range of application-specific branches and conditional operations simple and efficient for ASIC and SOC designers to implement—a very important feature for an ISA designed expressly for extensibility.

Zero-Overhead Loops

The Xtensa ISA also provides a general-purpose, zero-overhead loop feature similar to that found in some DSPs (digital signal processors). Most RISC processors use their existing conditional branch instructions to implement software loops. However, this opcode economy increases program cycle count and consequently reduces execution speed. For many RISC ISAs, loop overhead consists of three instructions: add, compare, and conditional branch. The loop overhead's performance impact is proportionately higher if the loop body is small. For small software loops, many compilers use an optimization called loop-unrolling to spread the loop overhead over two or more loop iterations, but this approach duplicates the loop body and increases code size.

Many DSPs and some general-purpose processors avoid these problems by providing other ways to perform certain kinds of loops. The first such method is to provide an instruction that repeats the succeeding instruction a fixed number of times (as implemented by TI's TMS320C2x and Intel's x86). For loops containing only one instruction, a repeat prefix instruction eliminates loop overhead and saves power by eliminating the need to repeatedly fetch the same instruction within the loop. Some ISAs with repeat instructions require that the processor not take an interrupt during the loop. This limitation can impose unacceptable interrupt latency because loop execution can consume many machine cycles.

An improvement on simple repeat prefix instructions is the ability to iterate a block of instructions multiple times with reduced or zero loop overhead. The Xtensa ISA provides just such a zero-overhead loop capability via its LOOP, LOOPGTZ, and LOOPNEZ instructions, which are present in all Diamond Standard processor cores except the Diamond 106Micro and 108Mini cores.

The Xtensa ISA's LOOP instructions eliminate instruction-execution cycles required for incrementing the loop index and for comparison and branch operations. It also avoids the taken-branch penalty that is typically associated with a compilation of loops based on conditional-branch instructions. The Xtensa ISA demonstrates how a reduced overhead looping capability can be integrated into a general-purpose processor ISA (as opposed to a DSP) to improve both execution performance and code size.

Overall, the Xtensa architecture makes six important contributions to general branch instructions:

1. A choice of several compare-and-branch instructions in a RISC ISA with the most useful comparisons
2. Compare-and-branch instructions with encoded immediate values, including branch-on-bit instructions
3. Instruction formats with longer target specifiers for common cases (test against zero)
4. The encoding of all branch instructions in 24-bit instruction words
5. Support for branches on coprocessor Boolean registers (condition codes) with logical operations on Booleans
6. Zero-overhead loops that eliminate branch execution delay and reduce code size.

Limited Instruction Constant Width

None of the standard Xtensa instructions is longer than 24 bits, so constant fields in the instruction word are constrained. The Xtensa ISA addresses this issue in several ways. First, the ISA provides small constant fields in many instructions to capture the most common constants. Xtensa instructions encode the constant value rather than specifying it directly. The encoded values are chosen from a wide array of program statistics as the N (e.g. 16) most frequent constants for each instruction type. For example, the Xtensa ISA employs this technique in the ADDI4 instruction, where the 16 values are chosen to be -1 and 1 to 15, rather than 0 to 15. Adding 0 is of no utility (there is a separate MOVE instruction) and adding –1 is common.

The constants used in bitwise-logical operations (e.g. AND, OR, XOR, etc.) represent bit masks of various sorts and, consequently, these masks often do not fit in small constant fields. However, bit patterns consisting of a sequence of 0s followed by a sequence of 1s, and a sequence of 1s followed by a sequence of 0s are quite common. For this reason, the Xtensa architecture has instructions that avoid the need for putting an immediate mask directly into the instruction word. The EXTUI instruction (described above) performs a shift followed by a mask consisting of a series of 0s followed by a series of 1s, where the number of 1s is a constant field in the instruction. This sort of encoding permits large bit-mask fields to be compacted into the Xtensa's 24-bit instruction word.

Xtensa load and store instructions use an instruction encoding format that has an 8-bit constant offset. This offset is added to a base address taken from a register. The Xtensa ISA makes the most of these 8 bits and provides a simple extension method when 8 bits is insufficient. Xtensa load and store offsets are zero-extended rather than sign-extended because the values 128 to 255 are more commonly used by load and store instructions than the values -128 to -1. Also, the offset is shifted left appropriately for the reference size because most references are to aligned addresses from an aligned base register. The offset for 32-bit loads and stores is shifted by 2 bits; the offset for 16-bit loads and stores is shifted by 1 bit; and the offset for 8-bit loads and stores is not shifted.

Most loads and store operations are for 32-bit words so this technique provides 2 additional bits of address range. The Xtensa ISA provides the ADDMI instruction, which adds an 8-bit immediate constant shifted left by 8 bits, when an 8-bit constant offset specified in a load or store instruction (or an ADDI instruction) is insufficient.

Short Instruction Format

The Xtensa ISA consists of a core set of instructions that must be present in all implementations of the instruction set, and a set of optional instruction packages that may or may not be present in a given implementation. One of the most popular packages is the short-instruction-format package, which permits further reductions in code size by further reducing the average number of bits per instruction. When these short-format instructions are present, the Xtensa ISA changes from a fixed-length, 24-bit instruction set to one with both 24- and 16-bit instructions. Note that the Xtensa architecture does not employ modes to add the 16-bit instructions to the ISA as do some other RISC processors. The Xtensa ISA's 24- and 16-bit instruction formats can be freely intermixed and the XCC compiler automatically selects the smallest available instruction for every operation.

Because the Xtensa short instruction forms are optional, these forms are used solely for improving code size; no new capabilities are added by the Xtensa ISA's 16-bit instructions. The set of instructions that can be encoded in 16 bits consists of the most frequent instructions that will fit, based on statistical analysis of a substantial number of embedded programs. The most frequently used instructions in most instruction sets are loads, stores, branches, adds, and moves; these are exactly the instructions in the Xtensa ISA's 16-bit instruction set.

Only the most frequent instructions need short encodings, so three register fields are still available (because the opcode field is small) and narrow, encoded constant fields can capture a significant fraction of the uses. Approximately half of the Xtensa instructions needed to represent an application can be encoded in just six of the sixteen opcodes available in a 16-bit instruction encoding after three 4-bit fields are reserved for register specifiers or constants.

External Processor Interface (PIF)

The PIF connects the core to any proprietary or standard system bus. The PIF width depends on the specific Diamond core:

- 32 bits for the Diamond 106Micro, 108Mini, 212GP and 232L cores
- 64 bits for the Diamond 570T core

The PIF consists of two separate, unidirectional input and output channels. The external interface unit manages data transfers between the PIF and the processor's local instruction memory ports or the data memory ports. In particular, this unit manages data and instruction cache-line requests and provides inbound PIF (external PIF master) capabilities to the processor's local instruction and data RAMS.

Xtensa Local Memory Interface (XLMI) Port

The Diamond Standard 212GP and 570T cores include one 128-Kbyte XLMI port. On the Diamond 212GP, this port is 32 bits wide and on the Diamond 570T, it is 64 bits wide. Unlike the other local-memory ports, the XLMI port is designed to connect to blocks and devices other than memory using signals to indicate when a load has been retired to help ensure that speculative-read effects (present in all pipelined processor cores) do not cause improper operation of devices attached to the XLMI port. Devices with read side effects can be attached to the XMLI bus as long as they adhere to the “load-retired” and “load-flushed” signaling protocols. Devices that have no such side effects can be attached to the XLMI port without the need to pay attention to the “load-retired” and “load-flushed” states.

Ports and Queues for High-Speed I/O

Diamond Standard processor cores offer a unique facility for extremely high-speed input and output using direct ports and queu interfaces that bypass the system bus. Designers can use these features to transfer data between processors and between the processor and other blocks of RTL hardware. Ports are 32 bits wide and are general-purpose input/output wires that can be connected to any part of the system. These wires are connected directly to a special register inside of the processor core. Data on these ports can be read from or written to the port register. Ports are available on the Diamond 108Mini, 212GP, and 570T cores.

Queue interfaces take this idea a bit further. Queue interfaces are 32-bits wide and include flow-control handshake logic, allowing the design team to place high-speed FIFO interfaces between the processor core and other system blocks, as shown in Figure 5. The FIFOs bypass the main system bus and avoid the latency of bus operations. The FIFO interfaces eliminate system-bus data contention, one of the most common problems in complex system-level silicon design today. Queue interfaces are available on the Diamond 570T processor core.

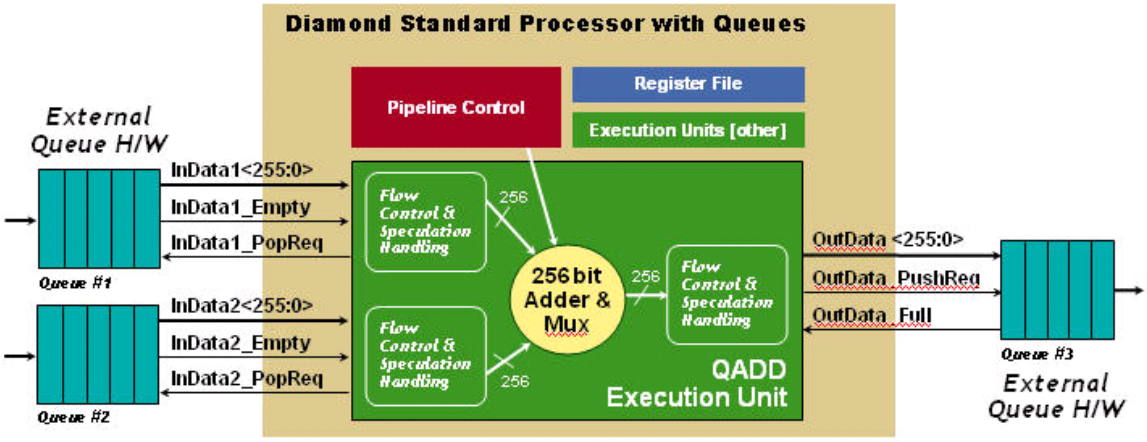


Figure 5. Use of Queues to Speed Data Transfer.

Interrupts and Timers

Diamond Standard processors feature rich interrupt and timer capabilities:

- Nine external interrupts
- Three timer interrupts
- two software interrupts

Architectural Building Blocks

The following blocks are included in all Diamond Standard processors:

On-chip Debug (OCD) – used to access the internal, software-visible processor state through a JTAG port. OCD support includes: debug-mode entry through exception generation, access to all program-visible registers and memory locations, execution of any instruction, modification of the program counter to jump to a desired code location, real-time debug, and a utility for returning to normal operating mode.

Local RAM Interfaces– the local RAM interface or interfaces provide internal memory ports with address ranges within the processor's address space. Local memory is accessed with the same timing as cache. There are two types of optional RAM interfaces: instruction and data.

Timer interrupts – there are three timer interrupts with one 32-bit read/write register that increments every clock cycle and thee 32-bit comparison registers that can generate level-1 interrupts or high-priority interrupts.

The following blocks are included in some Diamond Standard processors:

16-bit multiplier and multiplier-accumulator (MAC16) (offered in the Diamond 323GP, 232L, and 570T processor cores) – adds a 16x16-bit multiplier and a 40-bit accumulator, eight 16-bit operand registers (separate from the main register file), special load instructions for these registers, and a set of compound operations. The MAC16 operand registers can be loaded with pairs of 16-bit values from memory in parallel with MAC16 operations and the MAC16 can sustain algorithms with two loads per multiply/accumulate operation.

32-bit multiplier (offered in the Diamond 570T) – provides instructions that perform 32x32-bit multiplication, producing a 32-bit result.

Low Power – Built Into Each Diamond Processor Core

Clock gating is a very effective power reduction technique that reduces power consumption by stop the clock to parts of the logic that are not in use during a particular clock cycle. Tensilica has designed fine-grained clock gating for every functional element of the Diamond processors cores. As a result, the Diamond Standard Series processors feature dramatically lower power consumption.

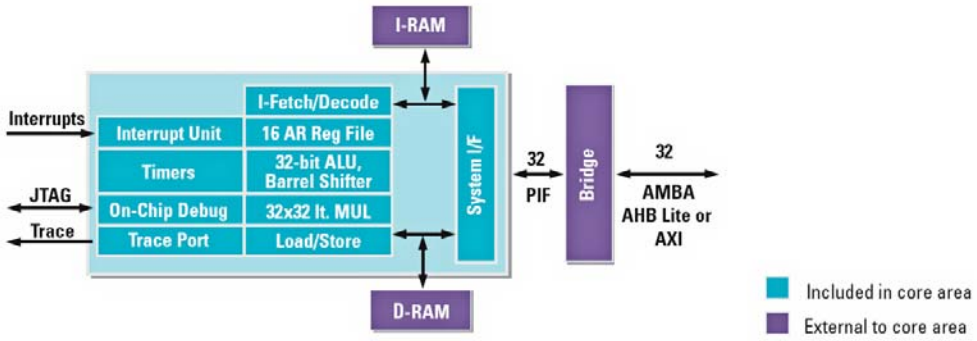
Summary

The Xtensa processor ISA makes a number of fundamental contributions to embedded processor architecture, including:

- A windowed register file with 16 visible entries
- Three-operand programming model using less than 32-bit instruction encoding for performance, generality, and code size
- Rich selection of commonly occurring instruction combinations implemented as compound instructions
- Encoding of common immediate values for improved performance and reduced code footprint
- A powerful branch architecture that includes compare-and-branch, bit-test-and-branch, coprocessor condition code test-and-branch, and zero-overhead loop instructions for increased performance and reduced code footprint
- A 16-bit instruction subset that can be freely intermixed with 24-bit base instructions for further code-size reduction

The Diamond Standard Processors

The Diamond Standard 106Micro RISC Controller Core



The Diamond Standard 106Micro controller core is the smallest 32-bit RISC processor based on an industry-standard architecture. It has been designed to attain the smallest die area and lowest power consumption of any 32-bit processor core. This cache-less controller is ideal for designers looking for a basic 32-bit controller, particularly for those migrating up from an 8- or 16-bit controller. The Diamond 106Micro processor core allows SOC architects to easily integrate an efficient CPU in their designs.

Although the Diamond 106Micro is extremely small, it employs a 5-stage pipeline so it easily achieves 250 MHz in a 130G IC-fabrication process and 400 MHz in a 90G process technology. Modeless switching between 24- and 16-bit narrow instructions permits much higher code density than competing 32/16-bit RISC architectures.

Local, tightly-coupled instruction and data memory use with the Diamond Standard 106Micro can store performance-sensitive code and data to achieve high performance with time-critical code such as interrupt handlers.

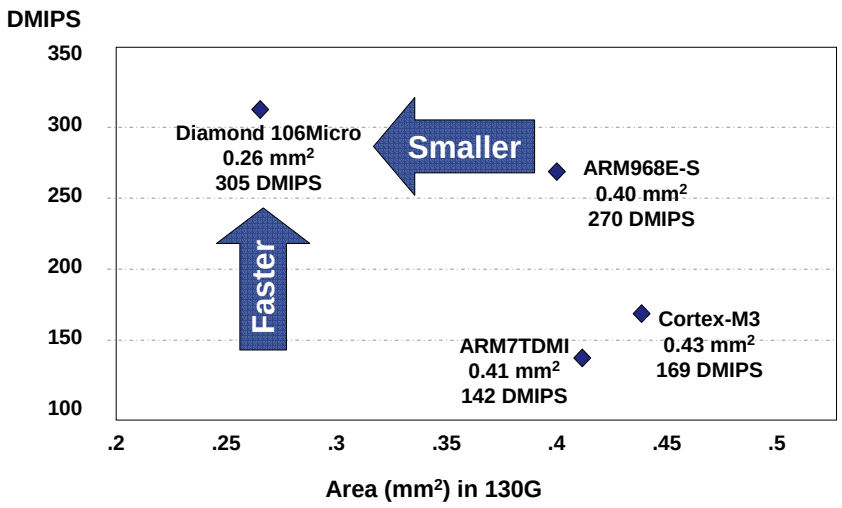
The Diamond 106Micro processor core has an iterative, multi-cycle (non-pipelined) 32x32-bit multiplier that greatly enhances performance of arithmetic and DSP code. The processor core has a non-windowed, 16-entry, general-purpose AR register file to minimize area. This feature potentially provides improved performance on applications that have very deeply nested function calls, because the lack of windowing means that the Xtensa windowing mechanism never throws an exception.

The Diamond Standard 106Micro has a rich interrupt architecture including an integrated interrupt controller with 15 interrupts and a 32-bit timer. These features simplify system design because no external hardware is needed to implement these functions.

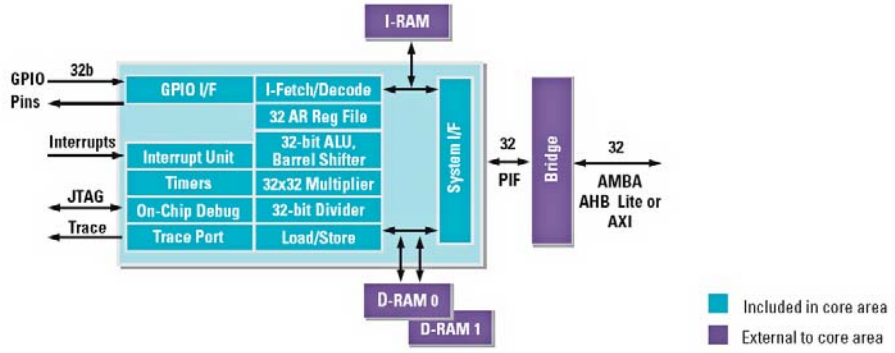
Diamond 106Micro processor core features include:

- Smallest, lowest power 32-bit RISC controller core
- Cache-less processor with memory protection unit
- 5-stage pipeline
- Dhryston MIPS: 1.22 DMIPS/MHz
- 24- and 16-bit ISA with modeless switching
- Iterative 32x32 multiplier
- Separate instruction- and data-memory interfaces
- Integrated interrupt controller with 15 interrupts, 2 priority levels
- Integrated timer
- On-chip debugging hardware
- Embedded trace support
- Comprehensive software development environment
- AHB-lite and AXI bus bridges

The Diamond 106Micro processor core provides better performance than ARM7, ARM9, or ARM Cortex-M3 processor cores while using less power and requiring less die area as shown below:



The Diamond Standard 108Mini RISC Controller Core



The Diamond Standard 108Mini controller core is a fully synthesizable 32-bit RISC processor. It is a small, cache-less RISC controller with tightly-coupled local instruction and data memories, a rich interrupt architecture, and excellent arithmetic and DSP performance. The Diamond 108Mini features low, class-leading power consumption, which is especially important for portable applications.

Although the Diamond 108Mini is smaller in die area than comparable 32-bit processor cores, its performance is extremely high: 420 MHz in a 90nm G IC-fabrication process technology. The Diamond 108Mini processor core achieves 1.34 Dhrystone MIPS/MHz. It also delivers high performance for DSP applications and engine and motor-control applications through its integral 32x32-bit multiplier and 32-bit integer divider.

The Diamond 108Mini delivers fast and flexible interrupt handling with low interrupt latency and a rich interrupt architecture. The cacheless processor provides deterministic behavior for applications with hard real-time constraints. The Diamond 108Mini processor has a 32-entry general-purpose register file with a 16-entry register window, which enables much faster context switching by reducing the need for stack operations. A local, single-cycle instruction memory interface allows time critical code to be placed near the CPU. Two local data memory interfaces allow the processor to access to one bank of local data RAM while an external DMA operation operates on the other bank through the processor's inbound-DMA facility. Separate local instruction- and data-memory interfaces lower bus contention when compared to unified memory-interface architectures.

The Diamond 108Mini achieves the performance levels of much larger, complex CPUs although it is small in size and requires relatively little power.

Diamond 108Mini features include:

- Cacheless design with memory protection unit
- Single-cycle local instruction- and data-memory interfaces
- Two separate local data-memory interfaces for 2-bank system architectures
- Non-maskable interrupt
- 9 external interrupts
- 3 timers
- On-chip debug hardware
- Programmable I/O ports that reduce external control logic and speed I/O
- FPGA system prototyping support reduces design risk
- Deterministic, real-time operation through optional single-cycle local instruction and/or data memory
- Optional AMBA AHB-lite bus bridge

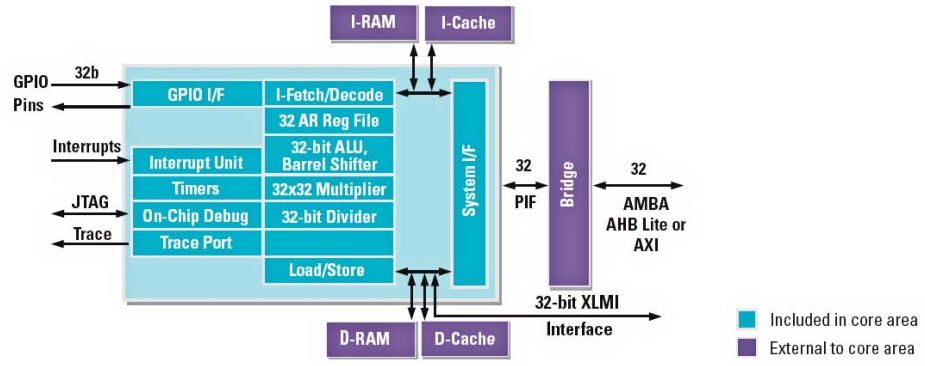
Representative Performance/Area/Power for the Diamond 108Mini processor core:

Maximum Frequency (0.13G worst case)	233-250 MHz
Dhrystone 2.1 MIPS/MHz	1.2
Die Area (0.13G pre-layout)*	0.43 mm^2
Die Area (0.13G post-layout)**	0.51 mm^2
Instruction Width	16/24 bits
mW/MHz (0.13G)**	0.082
All area, power, and frequency numbers are representative only and subject to variation based on each user's chosen process technology, cell library and design tools. * Area is post synthesis, pre-layout ** Area and power are post synthesis, post clock tree insertion, assuming 85% utilization.	

Diamond 108Mini Offers ARM9 Performance at ARM7 Power

Characteristic	ARM 7TDMI-S**	Diamond 108Mini	ARM 968E-S
Max frequency (0.13u G) worst case, optimized for speed	146 MHz	233-250 MHz	240 MHz
Dhrystone MIPS	131	300	264
Power – mW per MHz (0.13G)*	0.10	0.082	0.11
Area – post synthesis	0.24mm^2**	0.40 mm^2	0.40 mm^2
Area – post layout	n/a	0.51 mm2	n/a
Number of interrupts	3	15	3
Timers	No	Yes	No
Direct interface ports/wires	No	32-bit input ports, 32-bit output ports	No
*Power depends on operating conditions, cell libraries, performance targets, and processor load. **Assumes no bus interface, interrupt controller, trace interface, memory protection unit, GPIO. Data on ARM products taken from ARM public website, October 2006, for TSMC 0.13G process. All speed, power and area metrics are subject to variation based on user's design and fab choices.			

The Diamond Standard 212GP Controller Core



The Diamond Standard 212GP controller is a versatile, high-performance, fully synthesizable 32-bit RISC processor core. It is area and power efficient and features a local-memory architecture that provides outstanding flexibility and performance. Designers can take advantage of the processor core's lockable cache and can attach single-cycle instruction or data memories as large as 128 Kbytes to the processor's separate, local instruction- and data-memory ports.

Diamond 212GP target applications are usually controller related so the core's interrupt options are extremely important. The Diamond 212GP includes a non-maskable interrupt to handle critical system events and six interrupt levels consisting of external, software, and timing interrupts. This comprehensive interrupt capability eases the development of software interrupt handlers and interrupt-priority hardware design.

Arithmetic and DSP hardware support within the core reduces the need to include a separate DSP in the system design. DSP support in the Diamond 212GP consists of a single-cycle, 16x16-bit MAC unit, which adds four dedicated 32-bit registers and a 40-bit accumulator. Additionally, there is support for zero-overhead loops, a clamps instruction (for saturating arithmetic). In addition, the processor core has max/min value, normalize, and sign extend instructions for further code streamlining.

The Diamond 212GP has extremely high performance: 400 MHz in a 90nm G-type IC-fabrication process. It can handle nearly any control-plane and many DSP tasks because of its integral 32x32-bit multiplier and 32-bit integer divider.

Diamond 212GP processor core features include:

- Single-cycle, 16x16-bit MAC
- DSP instructions that eliminate need for a separate DSP
- 8-Kbyte, 2-way, set-associative instruction and data caches, programmable as either write-through or write-back
- Local, single-cycle instruction and data memory interfaces
- On-chip debug hardware

- Non-maskable interrupt
- 9 external interrupts
- 3 timers
- Programmable I/O ports that reduce external control logic and speed I/O
- FPGA system prototyping support to reduce design risk
- Optional AMBA AHB-lite bus bridge
- 128-bit XLMI single-cycle bus that can perform transfers much faster than the main bus
- Zero-overhead loops

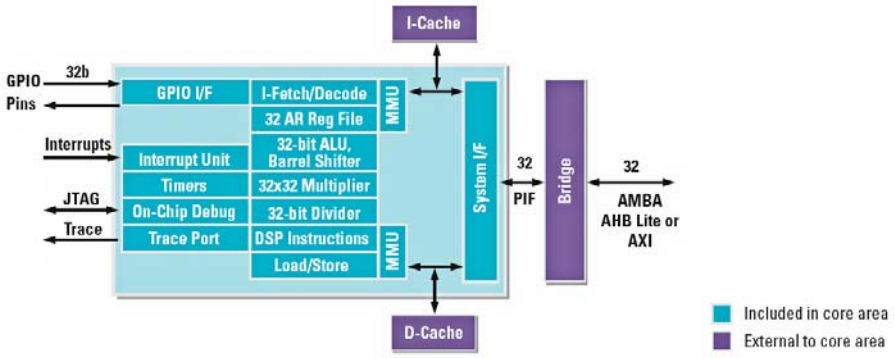
Representative Performance/Area/Power for the Diamond 212GP Processor Core

Maximum Frequency (0.13G worst case)	233-250 MHz
Dhrystone 2.1 MIPS	325
Die Area (0.13G pre-layout)*	0.56 mm^2
Die Area (0.13G post-layout)**	0.77 mm^2
Instruction Width	16/24 bits
mW/MHz (0.13G)** (power)	0.116
All area, power, and frequency numbers are representative only and subject to variation based on each user's chosen process technology, cell library and design tools. * Area is post synthesis, pre-layout ** Area and power are post synthesis, post clock tree insertion, assuming 85% utilization.	

Diamond 212GP Provides Better Performance than ARM9 with Lower Power and Smaller Area

	ARM 946E-S	Diamond 212GP
Max frequency (0.13u G) worst case, optimized for speed	210 MHz	233-250 MHz
Dhrystone MIPS	231	335
Power – mW per MHz (0.13G)*	0.31	0.116
Area – post synthesis	0.97 mm^2	0.56 mm^2
Area – post layout	n/a	0.77 mm^2
Zero-overhead looping	No	Yes
Number of interrupts	3	15
Timers	No	Yes
Direct interface ports/wires	No	32-bit input ports, 32-bit output ports
*Power depends on operating conditions, cell libraries, performance targets, and processor load. Data on ARM products taken from ARM public website, October 2006, for TSMC 0.13G process. All speed, power and area metrics are subject to variation based on user's design and fab choices.		

The Diamond Standard 232L RISC Controller Core



The Diamond Standard 232L CPU core is a versatile, high-performance, fully synthesizable 32-bit RISC processor core. It is area and power efficient with a local-memory architecture that provides outstanding flexibility and performance. The Diamond 232L CPU core features a full-featured Memory Management Unit (MMU) to support operating systems such as Linux. The processor core's separate 8-Kbyte instruction and data caches are 4-way, set-associative.

The Diamond 232L's MMU includes instruction and data TLBs (translation look-aside buffers) that perform virtual-to-physical address mapping. In addition to address translation, the MMU provides four different privilege levels (for memory protection), variable page sizes, and multiple access modes. With its MMU, flexible interrupt architecture, and high performance, the Diamond 232L can easily meet the needs of a complex ASICs and SOCs that must perform many simultaneous operations.

Arithmetic and DSP hardware support in the Diamond 232L processor core reduce the need to include a separate DSP in the system design. Arithmetic support is provided by a built-in 32x32-bit multiplier and 32-bit integer divider. DSP support in the Diamond 232L consists of a single-cycle, 16x16-bit MAC unit that adds four dedicated 32-bit registers and a 40-bit accumulator. Additionally, there is support for zero overhead loops and a clamps instruction for saturating arithmetic. The processor core also has max/min value, normalize, and sign-extend instructions.

Diamond 232L processor core features include:

- Single-cycle, 16x16-bit MAC
- DSP instructions that eliminate need for a separate DSP in the system
- 8-Kbyte, 2-way set associative instruction and data caches, programmable as either write-through or write-back
- On-chip debug hardware
- Non-maskable interrupt

- 9 external interrupts
- 3 timers
- Programmable I/O ports that reduce external control logic and speed I/O
- FPGA system prototyping support to reduce design risk
- Optional AMBA AHB-lite bus bridge
- 128-bit XLMI single-cycle bus that can perform transfers much faster than the main bus
- Zero-overhead loops
- Linux-compatible MMU

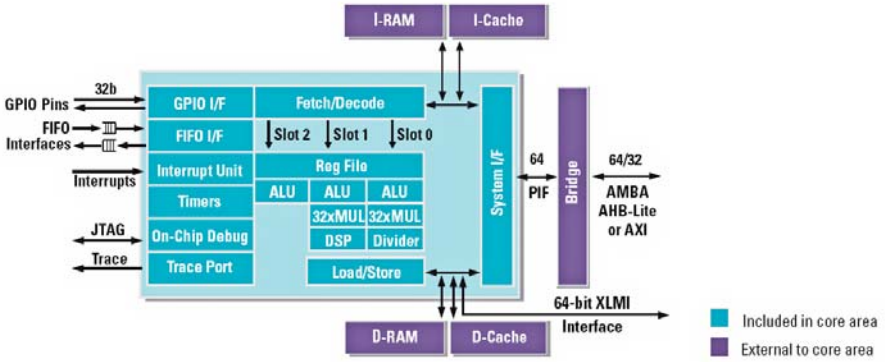
Representative Performance/Area/Power for Diamond 232L

Maximum Frequency (0.13G worst case)	233-250 MHz
Dhrystone 2.1 MIPS	300
Die Area (0.13G pre-layout)*	0.70 mm^2
Die Area (0.13G post-layout)**	0.81 mm^2
Instruction Width	16/24 bits
mW/MHz (0.13G)** (power)	0.189
Area, power, and frequency numbers are representative only and subject to variation based on each user's chosen process technology, cell library and design tools. * Area is post synthesis, pre-layout ** Area and power are post synthesis, post clock tree insertion, assuming 85% utilization.	

Diamond 232L Offers More Linux-Ready Features at Half the Power and Area

	ARM 926EJ-S	**Diamond 232L**
Max frequency (0.13u G) worst case, optimized for speed	250 MHz	233-250 MHz
Dhrystone MIPS	275	300
Power – mW per MHz (0.13G)*	0.36	0.189
Area – post synthesis	1.45 mm^2	0.70 mm^2
Area – post layout	n/a	0.77 mm^2
Zero-overhead looping	No	Yes
Number of interrupts	3	15
Timers	No	Yes
*Power depends on operating conditions, standard cell libraries, performance targets, and processor load. Data on ARM products taken from ARM public website, October 2006, for TSMC 0.13G process. All speed, power and area metrics are subject to variation based on user's design and fab choices.		

The Diamond Standard 570T Static-Superscalar CPU Core



The Diamond Standard 570T CPU core is among the highest performance, highest throughput licensable processor cores available. It's efficient 5-stage pipeline and 3-issue VLIW architecture delivers industry-leading performance levels for both control and DSP code when benchmarked using industry-standard EEMBC benchmarks.

The Diamond 570T CPU core's 16-, 24-, and compound 64-bit VLIW instruction bundles can be freely intermixed in the instruction stream with no processor mode switching, thus boosting performance while minimizing code size. The XCC C/C++ compiler automatically creates 64-bit VLIW instruction bundles if instructions can be issued simultaneously; otherwise, the compiler selects 16- or 24-bit instructions. This capability increases code density and reduces the amount of on-chip cache or memory required for instruction storage.

The Diamond 570T CPU core includes many standard DSP instructions to increase performance of numerically intensive applications. It also has a 32x32-bit multiplier and 32-bit integer divider. Example DSP instructions include: zero-overhead loops, clamps (saturating arithmetic), max/min value, normalize, and sign extend. Additionally, a MAC unit enables high performance on inner loops requiring fast multiplication.

The Diamond 570T CPU core also includes 32-bit input/output GPIO ports and 32-bit input/output FIFO interfaces, which can be used to connect to standard FIFOs for fast communication with other RTL blocks, devices, and processors without using the system bus for maximum system throughput.

Diamond 570T CPU core features include:

- Three-issue, static superscalar VLIW CPU
- Modeless switching between 16-, 24-, and 64-bit VLIW instructions
- 64-bit local-memory interfaces to instruction and data cache and to local instruction and data memories
- Single-cycle, 16x16-bit MAC and two 32-bit multipliers
- 16Kbyte, 2-way set-associative instruction and data caches, programmable as write-through or write-back
- Single-cycle local instruction- and data-memory interfaces
- On-chip debug hardware
- Non-maskable interrupt
- 9 external interrupts
- 3 timers
- Programmable I/O ports that reduce external control logic and speed I/O
- FPGA system prototyping support to reduce design risk
- Optional AMBA AHB-lite bus bridge
- 128-bit XLMI single-cycle bus that can perform transfers much faster than the main bus
- 64-bit peripheral interface (PIF) bus
- High-speed 32-bit input/output FIFO queue interfaces that eliminate system-bus contention

Representative Performance/Area/Power for Diamond 570T

Maximum Frequency (0.13G worst case)	200-233 MHz
Dhrystone 2.1 MIPS/MHz	1.52
Die Area (0.13G pre-layout)*	1.03 mm^2
Die Area (0.13G post-layout)**	1.58 mm^2
Instruction Width	16/24/64 bits
mW/MHz (0.13G)** Power	0.087-0.410
mW/MHz (90nm typical max frequency)** (power)	0.155
All area, power, and frequency numbers are representative only and subject to variation based on each user's chosen process technology, cell library and design tools. * Area is post synthesis, pre-layout ** Area and power are post synthesis, post clock tree insertion, assuming 65% utilization.	

Diamond 570T Uses Less than Half the Die Area and Power of ARM 1136/1156

	ARM 1156T2-S	Diamond 570T	ARM 1136J-S
Instruction Issue (per cycle)	1	3	1
Dhrystone MIPS	402	354	396
Dhrystone MIPS/MHz	1.20 (est)	1.52	1.20
Power – mW per MHz (0.13G)*	0.24	0.08	0.24
Area – post layout	0.90 mm^2	0.48 mm^2	0.90 mm^2
Number of pipeline stages (deeper pipelines are less efficient)	9	5	8
Instruction width	16/32 bit	16/24/64 bit 3-issue	16/32 bit
High throughput data Queues	No	Yes (input and output)	No
Direct interface ports/wires	No	32-bit input ports, 32-bit output ports	No

*Power depends on operating conditions, standard cell libraries, performance targets, and processor load.

Data on ARM products taken from ARM public website, October 2006, for TSMC 90nmG process. All speed, power and area metrics are subject to variation based on user's design and fab choices.

Benchmarks – The Diamond Standard Processor Family Leads the Industry

In benchmark after benchmark, the Diamond Standard processor family comes out on top. Tensilica used its Diamond 570T high-performance CPU in the popular industry EEMBC benchmarks, with the following results:

EEMBC (Embedded Processor Benchmark Consortium) Benchmarks

No single benchmark can accurately capture the full range diversity of embedded applications. In an effort to create an embedded benchmark that would be more informative than the Dhrystone, EDN Magazine sponsored the creation of a comprehensive suite of representative embedded applications. From that start grew the industry benchmarking consortium called EEMBC. More than 40 leading processor and software companies initially joined EEMBC and together, they developed both a set of benchmarks and a fair process for running, measuring, certifying, and publishing test results. These benchmarks cover a wide range of embedded tasks, but the bulk of the certified results are available for four suites: networking, consumer, telecommunications, and office automation.

The data in this section is taken directly from the certified results on the EEMBC website at www.eembc.com, as of October 2006. In each the Diamond Standard 570T processor core's performance is compared to the published results for ARM cores

The ARM architecture is represented in the EEMBC benchmarks by the ARM1026EJ-S, the only core that has been benchmarked by ARM. We also compare it to the ARM1136JF-S, which was benchmarked as an implemented chip in a Freescale IMX31 device. No certified ARM11 EEMBC results have been published as of October 2006.

Each EEMBC test suite consists of a number of different programs, written in C. The EEMBC Netbench 1.1 benchmark suite approximates the task load of a low-end network router. It consists of three benchmark kernels. One implements the Dijkstra shortest-path-first algorithm, which is widely used in routers and other networking equipment to find the shortest or least-cost path from a specific router to all other routers. The packet-flow benchmark indicates the potential performance in an IP router with four network interfaces. In the route-lookup benchmark, performance is measured on the fundamental operation used by IP-datagram routers, including receiving and forwarding IP datagrams and implementing an IP-lookup mechanism based on a Patricia Tree.

The EEMBC Consumer benchmark suite is a compilation of five separate benchmark kernels that are representative of consumer digital imaging applications. The high-pass grey-scale filter benchmark demonstrates performance in front-end processing of digital still cameras. This benchmark tests 2D-array and multiply/accumulate capabilities. The JPEG compression and decompression benchmarks take still images from full source data captured from a sensor, compress the data using the

JPEG file format, and then decompress the JPEG image back to the full image representation. These are a common set of tasks found in digital still cameras and digital video camcorders. The RGB to CYMK conversion benchmark demonstrates a common conversion used in color printing. The RGB to VIQ conversion benchmark demonstrates a conversion used by NTSC encoders for digital video processing.

The EEMBC Office Automation benchmark is a suite of benchmarks that approximate the roles of processors in printers, plotters and other office automation systems that handle text and image processing. This benchmark suite includes a dithering benchmark that evaluates how the processor handles indirect references (used for managing internal buffers), how it manipulates large data sets, how it manipulates packed-byte quantities (used to hold gray-scale pixel data), and how it performs four byte-wide multiply accumulate operations per pixel.

An image rotation benchmark uses a bitmap rotation algorithm that rotates a complete binary image 90 degrees clockwise, testing bit manipulation, comparison, and indirect reference capabilities. A text processing benchmark exercises the processor's byte-manipulation, pointer-comparison, and stack-manipulation capabilities and its indirect-reference handling.

The EEMBC Telecom benchmark suite approximates modem, xDSL, and related fixed-telecom applications. It includes five kernels that represent traditional DSP algorithms:

- The autocorrelation benchmark is based on a mathematical tool frequently used in signal processing for analyzing functions or series of values, such as time domain signals.
- The convolutional encoder benchmark, useful for cellular and modem applications, adds redundancy for error checking and explores the processor's ability to perform bit-wise exclusive ORs and table lookups.
- The bit-allocation benchmark tests the ability to stream data over a series of buffers, which it then modulates and transmits over a telephone line in ADSL applications.
- The Inverse Fast Fourier Transform benchmark tests the processor's ability to convert frequency-domain data into time-domain data while the Fast Fourier Transform benchmark tests the processor's ability to convert time-domain data into frequency-domain data.
- The Viterbi decoder benchmark tests the processor's ability to recover an output data packet from an encoded input data packet in embedded IS_136 channel-coding applications.

The following chart shows how the Diamond Standard 570T performs compared to the tested ARM processors.

Diamond 570T Performs 2.3X BETTER than ARM1136JF-S on EEMBC Benchmarks

	ARM 1136JF-S*	ARM 1026EJ-S (certified as a core)	Diamond 570T
NetMARK	1.0	1.29	2.55
ConsumerMARK	1.0	1.47	2.91
OfficeMARK	1.0	1.19	1.64
TeleMARK	1.0	1.06	2.28
Geometric Mean	1.0	1.24	2.30
*Results extrapolated from Freescale IMX31 device. No certified ARM1136JF-S EEMBC results have been published as of October 2006.			

The chart shows how the Diamond Standard 570T performs significantly better than the ARM processors on all EEMBC benchmarks with a geometric mean score that's 2.3x better than and ARM 1136JF-S and nearly twice as fast as an ARM 1026EJ-S.

An Open Invitation

If your design team needs one or more small, high-performance, low-power processor cores (controllers, CPUs, or DSPs) for your next SOC design, contact Tensilica for assistance. For more information on the unique abilities and features of the Diamond Standard family of processor cores, see www.tensilica.com, email sales@tensilica.com, or contact Tensilica directly at:

US Sales Offices:

Santa Clara, CA office:
3255-6 Scott Blvd.
Santa Clara, CA 95054
Tel: 408-986-8000
Fax: 408-986-8919

San Diego, CA office:
1902 Wright Place, Suite 200
Carlsbad, CA 92008
Tel: 760-918-5654
Fax: 760-918-5505

Boston, MA office:
25 Mall Road, Suite 300
Burlington, MA 01803
Tel: 781-238-6702 x8352
Fax: 781-820-7128

International Sales Offices:

Yokohama office (Japan):
Xte Shin-Yokohama Building 2F
3-12-4, Shin-Yokohama, Kohoku-ku,
Yokohama
222-0033, Japan
Tel: 045-477-3373 (+81-45-477-3373)
Fax: 045-477-3375 (+81-45-477-3375)

UK office (Europe HQ):
Asmec Centre
Eagle House
The Ring
Bracknell
Berkshire
RG12 1HB
Tel : +44 1344 38 20 41
Fax : +44 1344 30 31 92

Israel:
Amos Technologies
Moshe Stein
moshe@amost.co.il

Beijing office (China HQ):
Room 1109, B Building, Bo Tai Guo Ji,
122th Building of Nan Hu Dong Yuan, Wang Jing,
Chao Yang District, Beijing, PRC
Postcode: 100102
Tel: (86)-10-84714323
Fax: (86)-10-84724103

Taiwan office:
7F-6, No. 16, JiHe Road, ShihLin Dist,
Taipei 111, Taiwan ROC
Tel: 886-2-2772-2269
Fax: 886-2-66104328

Seoul, Korea office:
27th FL., Korea World Trade Center,
159-1, Samsung-dong, Kangnam-gu,
Seoul 135-729, Korea
Tel: 82-2-6007-2745
Fax: 82-2-6007-2746