

ConnX D2 DSP Engine

Dual-MAC, 16-bit Fixed-Point Communications DSP

FEATURES

- Both SIMD and 2-way FLIX (parallel VLIW) operations
- Optimized, vectorizing XCC Compiler
- High-performance DSP instruction set
- Dual write ports compute up to three results/cycle
- Supports TI (C6x) and ITU-T C intrinsic code base
 - Bit-for-bit compatible with TI C6X code
- C-centric programming model supports standard C 16-bit, 32-bit and 40-bit data types

BENEFITS

- Outstanding “out of the box” performance on compiled C source
- Reduces or eliminates the need for assembly code
- Performance acceleration for vectorizable code
- VLIW parallel execution for non-vectorizable code
- Large base of pre-optimized C code
 - Quick and easy compiling of C code optimized with TI C6x intrinsics
 - Quickly leverage all ITU-T reference code using ITU C intrinsics

A Flexible 2-MAC DSP

The explosive growth in next-generation wireless communications, disk drives, home entertainment devices, and computer peripherals is driving demand for 16-bit fixed-point DSPs. Stand-alone DSP chips are no longer cost effective for most of these price-sensitive applications. Instead, there's growing demand for general-purpose 16-bit DSP engines that can be easily designed into highly integrated system-on-chip (SOC) silicon.

At the same time, the growth of multiple standards and the complexity of these standards is driving developers away from traditional assembly-code programmed DSPs towards integrated architectures that combine excellent DSP performance with generalized high performance when developing with compiled native C control code.

The market needs a DSP engine that can easily be customized if necessary, integrated into a SOC design, and programmed most often in C, rather than assembly code. This will help speed new products to market as quickly as possible.

The Flexibility of C Programming with Assembly Level Performance

The ConnX D2 DSP Engine, used with Tensilica's Xtensa LX processor core, provides approximately 20% higher performance than similar dual-MAC architectures.

ConnX D2 DSP Engine

The ConnX D2 DSP engine is a click-box option for Tensilica's benchmark-breaking Xtensa LX processor technology. The ConnX D2 option adds dual 16-bit multiply-accumulate (MAC) units and a 40-bit register file to the base RISC architecture of the Xtensa LX processor. The ConnX D2 engine utilizes two-way SIMD (single instruction, multiple data) instructions to provide high performance on vectorizable C code. It also delivers dual-MAC performance using 64-bit VLIW (very long instruction word) instructions for code that cannot be vectorized.

The ConnX D2 DSP engine delivers outstanding 16-bit fixed point “out of the box” performance on compiled C code, without the need for assembly code optimization. This allows SOC development teams to have greater flexibility in resource allocation as well as the ability to quickly change algorithms. C code optimized with TI C6x or ITU C intrinsic functions compiles directly to the ConnX D2 instruction set, allowing developers to benefit from pre-existing TI and ITU code bases.

The ConnX D2 engine is supported by the comprehensive Eclipse-based Xtensa Explorer™ software development environment containing everything from a source code editor, debugger, and ISS to the highly optimized Xtensa C/C++ (XCC) compiler that provides excellent code density.

Out of the Box Performance

The ConnX D2 DSP engine is tightly integrated with the advanced Tensilica XCC compiler technology. The XCC compiler can efficiently map C algorithms to the ConnX D2 ISA (instruction set architecture) from native C and C intrinsic code, removing the need for time-consuming assembly code optimization. An example of this “out of the box” performance is the AMR-NB (VAD2) algorithm (encoder + decoder), which requires just 21.6 MHz. This beats equivalent competitive DSP cores by as much as twice the performance in some comparisons. Another example is the ConnX D2 running an 256 point complex FFT (fast Fourier transform), which gives 20% better performance than a competitive DSP core running hand optimized assembly code.

The ability to do optimized software design in C means that there is a faster response time to changing algorithms as well as less dependence upon key programming resources.

Consistent Performance – Even when Vectorization is Not Possible

Many high-performance DSPs are large SIMD engines that run vector data through at maximum bandwidth.

These DSPs rely upon compiler vectorization of C code to hit their peak performance. However, if a loop is not vectorizable, then the SIMD engine degenerates into a single-MAC DSP. The major shortcoming of these DSPs is that non-vectorizable code is commonplace. One example: a loop where non-single-integer address increments are used. The ConnX D2 engine solves this problem by using the parallel computation of the FLIX (VLIW) architecture. The dual MACs in the ConnX D2 engine can be fully saturated with either SIMD instructions or VLIW instructions, delivering maximum performance on all types of C code.

Figure 1 shows a simplified block structure of the ConnX D2 engine with two MAC units as well as register banks. Within the ConnX D2 data registers (XD_DR) the following data types are supported:

- 16, 32 and 40-bit integer
- 16, 32 and 40-bit fixed point
- 16-bit vector x 2
- 8-bit vector x 4
- 16-bit complex number

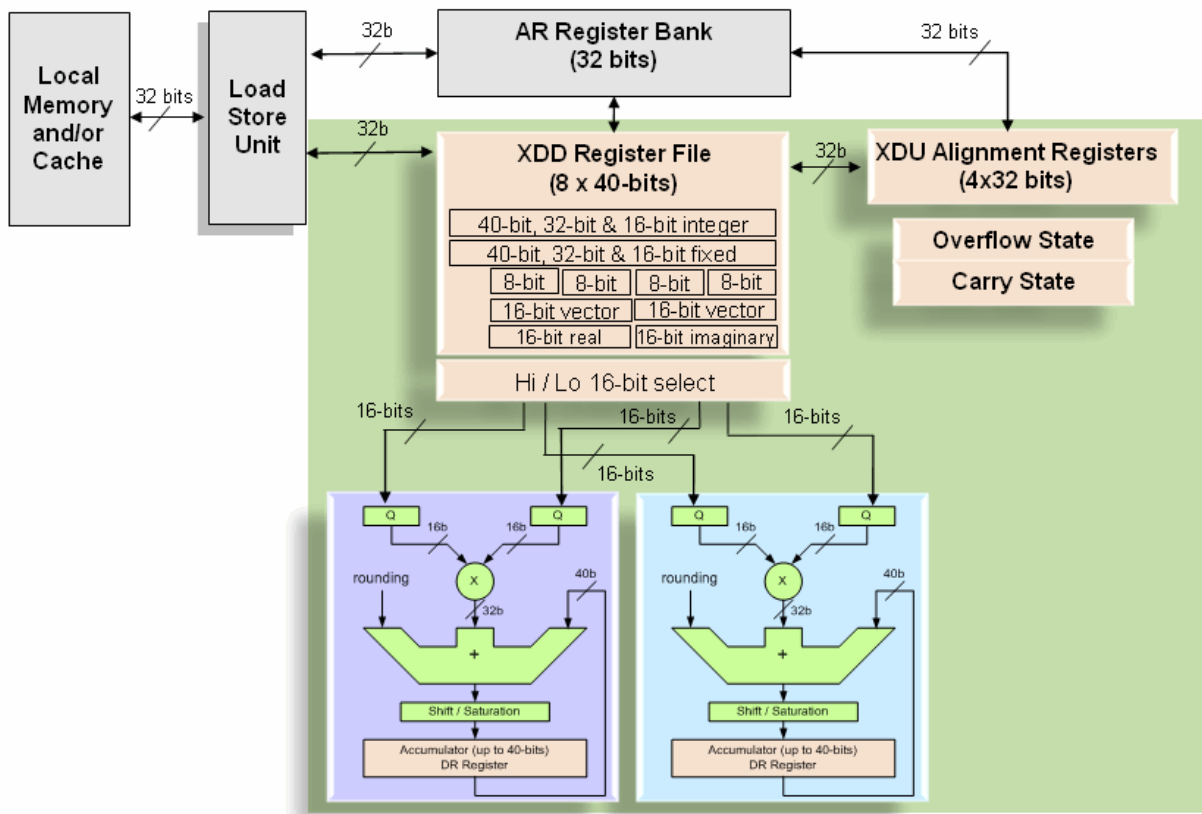


Figure 1. A simplified block structure of the ConnX D2 DSP Engine

Advanced DSP Instruction Set

The ConnX D2 instruction set is specifically optimized for the demanding numeric computations required for digital signal processing, and it is a natural complement to Tensilica's Xtensa LX base instruction set. By adding 275 DSP-specific optimizing instructions, the resulting processor (Xtensa LX plus ConnX D2) efficiently performs 16-, 32-, and 40-bit fixed point additions, subtractions, and multiplies with rounding and saturation. It uses seven DSP-centric addressing schemes and adds data manipulation instructions, including shifting, swapping, and logical operations to provide outstanding performance on DSP algorithms. Supported DSP addressing modes include:

- Immediate
- Immediate updating
- Indexed
- Indexed updating
- Aligning updating
- Circular (instruction)
- Bit-reversed (instruction)

For specific DSP algorithm acceleration the ConnX D2 engine instructions include:

- Add-Compare-Exchange (used within Viterbi algorithms)
- Add Modulo
- Add Subtract

Used in conjunction with a bit reverse addressing scheme, this instruction set executes FFT algorithms very fast.

These instructions are complemented with the dual-port write technology within the ConnX D2 engine. This allows two results to be written to the register files in one instruction. This can give a maximum of three writes to the register files per cycle within the VLIW implementation.

The ConnX D2 SIMD unit is supported by a comprehensive set of instructions for vector base loads and stores to support multiple data widths and SIMD data register loading orders, which can be aligned or unaligned.

Tensilica's optimizing compiler automatically packs instructions for optimum parallel execution, maximizing the use of the built-in VLIW capabilities. The compiler can also reduce down to 24-bit instructions when parallel operation is not possible.

If designers have specific optimizations in mind that are not included in the ConnX D2 and Xtensa LX instruction sets, they can easily add multi-cycle execution units, registers, register files, and much more using the Tensilica Instruction Extension (TIE) methodology (details available at www.tensilica.com). It's much easier and faster to add new instructions to an Xtensa processor than to design a Verilog hardware block to perform that function.

Operation	Instructions	Description
Arithmetic		
Absolute value	XD2_ABS.NS.D16S, XD2_ABS.NS.D16X2S, XD2_ABS.NS.D32S, XD2_ABS.S.D40S	Saturating signed absolute value
Add	XD2_ADD.A16S, XD2_ADD.D16X2S, XD2_ADD.D32S, XD2_ADD.D40, XD2_ADD.NS.D16X2S, XD2_ADD.NS.D40S, XD2_ADD.S.D16S, XD2_ADD.S.D16X2S, XD2_ADD.S.D32S, XD2_ADD.S.D40S, XD2_ADD32.D16X2S, XD2_ADDX2.D32S	Various addition (signed, unsigned, saturating)
Add with carry	XD2_ADDC.D32S	Add with carry
Add immediate	XD2_ADDI.A16S, XD2_ADDI.D32S	Add immediate
Add shifted	XD2_ADDX2.D32S	
Add-Subtract	XD2_ADDSUB.D16X2S, XD2_ADDSUB.D32S	2 result add subtract generation. FFT acceleration
Add-Compare-Exchange	XD2_ADDCXCHG.D16x2S	Viterbi
Add-Modulo	XD2_ADDMOD.A16S.D16X2S	Circular buffer addressing
Add Bit Reversed Base	XD2_ADDBRB.A32S.D32S	FFT algorithm, bit reverse base addressing
Min/Max	XD2_MAX.D16X2S, XD2_MAX.D40S, XD2_MIN.D16X2S, XD2_MIN.D40S	
Negate	XD2_NEG.NS.D16S, XD2_NEG.NS.D16X2S, XD2_NEG.NS.D32S, XD2_NEG.S.D40S	Saturating negation
Normalized shift amount	XD2_NSAZ.D16S, XD2_NSAZ.D32S, XD2_NSAZ.D40S, XD2_NSAZ.F.D40S	Normalized shift amount

Operation	Instructions	Description
Arithmetic		
Subtract	XD2_SUB.A16S, XD2_SUB.D16X2S, XD2_SUB.D32S, XD2_SUB.D40S, XD2_SUB.NS.D16X2S, XD2_SUB.NS.D40S, XD2_SUB.S.D16S, XD2_SUB.S.D16X2S, XD2_SUB.S.D32S, XD2_SUB.S.D40S	Various subtraction (signed, unsigned, saturating)
Subtract with carry	XD2_SUBC.D32S	Subtract with carry
Logical Operations	XD2_AND.D40, XD2_NAND.D40, XD2_OR.D40, XD2_XOR.D40	Bitwise logical operations on Data registers
Multiplies		
16-bit Fixed Point Multiply	XD2_MUL.F40.D16S.LL{_S1}, XD2_MUL.F40.D16S.LH, XD2_MUL.F40.D16S.HH, XD2_MUL.FS.D16S.LL{_S1}, XD2_MUL.FS32.D16S.HH{_S1}, XD2_MUL.FS32.D16S.LH{_S1}, XD2_MUL.FS32.D16S.LL{_S1}, XD2_MUL.FS32.D32S.D16S, XD2_MUL.F40.D32S.D16S, XD2_MUL.RFS.D16S.LL{_S1}, XD2_MULA.FS32.D16S.HH{_S1}, XD2_MULA.FS32.D16S.LL{_S1}, XD2_MULA.FS40.D16S.LL{_S1}, XD2_MULA.FS40.D16S.LH, XD2_MULA.FS40.D16S.HH, XD2_MULA.RFS16.D32S.D16S.LL{_S1}, XD2_MULS.FS32.D16S.HH{_S1}, XD2_MULS.FS32.D16S.LL{_S1}, XD2_MULS.FS40.D16S.LL{_S1}, XD2_MULS.FS40.D16S.LH, XD2_MULS.FS40.D16S.HH, XD2_MULS.RFS16.D32S.D16S.LL{_S1}	Fixed point multiply with options of asymmetric rounding, saturation, overflow
16-bit x 32-bit Fixed Point Multiply	XD2_MUL.F40X2.D16X2S, XD2_MUL.FS.D16X2S, XD2_MUL.FS32X2.D16X2S, XD2_MUL.RFS.D16X2S, XD2_MULAA.FS32.D16S.LL.HH, XD2_MULSS.FS32.D16S.LL.HH, XD2_MULSS.FS40.D16S.LL.HH, XD2_MULSS.FS40.D16S.HL.LH, XD2_MULSS40.D16S.HL.LH, XD2_MULSA.FS40.D16S.LL.HH, XD2_MULSA40.D16S.LL.HH, XD2_MULZAA.F40.D16S.HL.LH, XD2_MULZSS.F40.D16S.LL.HH, XD2_MULZSS.F40.D16S.HL.LH, XD2_MULZSS40.D16S.LL.HH, XD2_MULZSS40.D16S.HL.LH, XD2_MULAA40.D16S.LL.HH, XD2_MULSS40.D16S.LL.HH, XD2_MULSS40.D16S.HL.LH, XD2_MULZAA40.D16S.LL.HH, XD2_MULZAA40.D16S.HL.LH, XD2_MULZAA.F40.D16S.LL.HH, XD2_MULZAA.FS32.D16S.LL.HH, XD2_MULAA.FS40.D16S.LL.HH, XD2_MULAA.FS40.D16S.HL.LH, XD2_MULZAS.F40.D16S.LL.HH, XD2_MULZAS.F40.D16S.HL.LH, XD2_MULZAS40.D16S.LL.HH, XD2_MULZAS40.D16S.HL.LH, XD2_MULAS.FS40.D16S.LL.HH, XD2_MULAS.FS40.D16S.HL.LH, XD2_MULAS40.D16S.LL.HH, XD2_MULAS40.D16S.HL.LH	Fixed point multiply with options of asymmetric rounding, saturation, overflow
16-bit Integer Multiply	XD2_MUL32.D16S.HH{_S1}, XD2_MUL32.D16S.LH{_S1}, XD2_MUL32.D16S.LL{_S1}, XD2_MUL32.D16U.LL{_S1}, XD2_MUL40.D16S.LL{_S1}, XD2_MULA40.D16S.HH{_S1}, XD2_MULA40.D16S.LL{_S1}, XD2_MULA40.D16S.LH, XD2_MULS40.D16S.HH{_S1}, XD2_MULS40.D16S.LH, XD2_MULS40.D16S.LL{_S1}	16-bit Integer multiple to 32-bit, 40-bit
32-bit x 16-bit Multiply	XD2_MUL32X2.D16X2S	
Complex Multiply	XD2_CMUL.FS.DC16S, XD2_CMUL.RFS.DC16S	Fixed point complex multiply (imaginary portion on the low)
Saturation and Rounding		
Rounding	XD2_ROUNDASYM16.D40S, XD2_ROUNDASYM16X2.D40X2S, XD2_ROUNDASYM16.D40S, XD2_ROUNDASYM16X2.D40X2S	Asymmetric and symmetric rounding
Saturation	XD2_SAT16.D40S, XD2_SAT32.D40S, XD2_SATOVFL.D32S	Saturate with overflow
Clamping	XD2_CLAMP.S.D40S	Signed clamp with no overflow
Sign Extension	XD2_SEXT.D32S	Sign extend
Extraction	XD2_EXTUI.D32U	Extract unsigned immediate

Operation	Instructions	Description
Shifting		
Left/Right saturating	XD2_SHL.S.D16S, XD2_SHL.S.D16X2S, XD2_SHL.S.D32S, XD2_SHL.S.D40S, XD2_SHLI.S.D16S, XD2_SHLI.S.D16X2S, XD2_SHLI.S.D32S, XD2_SHR.S.D16S, XD2_SHR.S.D16X2S, XD2_SHR.S.D32S, XD2_SHR.S.D40S, XD2_SHRI.S.D16S, XD2_SHRI.S.D16X2S, XD2_SHR.NS.D40S	Bidirectional saturating signed shifting, 40-bit, 32-bit, 16-bit and 16x2-bit
Right with rounding	XD2_SHR.RS.D16S, XD2_SHR.RS.D16X2S, XD2_SHR.RS.D32S, XD2_SHR.RS.D40S, XD2_SHRI.RS.D32S	Bidirectional saturating signed fixed point with rounding, 40-bit, 32-bit, 16-bit and 16x2-bit
Non-saturating	XD2_SLL.D16X2S, XD2_SLLI.D16X2S, XD2_SLL.D32S, XD2_SLLI.D32S, XD2_SRA.D16X2S, XD2_SRAI.D16X2S, XD2_SRA.D32S, XD2_SRAI.D32S, XD2_SRL.D16X2U, XD2_SRL.D32U, XD2_SRLI.D32U	Unidirectional, 40-bit, 32-bit, 16-bit and 16x2-bit
Unsigned	XD2_SHR.D40U	Bidirectional unsigned
Vector Select	XD2_VSEL.D16x2	Vector select two 16-bit elements
Byte exchange	XD2_BXCHG.D16XS2	Xchange bytes within 16-bit vector (xd2_int16x2d)
Comparison		
Comparison	XD2_EQ.D40, XD2_EQ2.D16X2S, XD2_LE.D40S, XD2_LE.D40U, XD2_LE2.D16X2S, XD2_LE2.D16X2U, XD2_LT.D40S, XD2_LT.D40U, XD2_LT2.D16X2S, XD2_LT2.D16X2U, XD2_NE.D40, XD2_NE2.D16X2	Compare of vector elements, with Boolean register update
Register Move		
Unconditional	XD2_MOV.A32.D32, XD2_MOV.A32.U32, XD2_MOV.A8.D8HS, XD2_MOV.A2.B2, XD2_MOV.B2.A2, XD2_MOV.D32.A32S, XD2_MOV.D32.A32U, XD2_MOV.D40, XD2_MOV.D40.A40, XD2_MOV.D8X4.D8, XD2_MOV.U32, XD2_MOV.U32.A32, XD2_MOV16X4.D8X4S, XD2_MOV16X4.D8X4U, XD2_MOV8X4.D16X4.H, XD2_MOV8X4.D16X4.L, XD2_MOV.D40.D32.D8	Unconditional register moves
Conditional	XD2_MOVEF.D16X2, XD2_MOVEF.D40, XD2_MOVELTZ.D40S, XD2_MOVEVT.D16X2, XD2_MOVEVT.D40, XD2_MOVEQZ.D40S, XD2_MOVEGEZ.D40S, XD2_MOVEVEZ.D40S	Conditional register moves
Immediate	XD2_MOVEI.D40, XD2_MOVEI.U32	Move immediate to DR
States	XD2_MOVE.CARRY.A1, XD2_MOVE.OVERFLOW.A1, XD2_MOVE.A1.CARRY, XD2_MOVE.A1.OVERFLOW, XD2_MOVE.FLAGS.A1, XD2_MOVE.A1.FLAGS	Writing and reading CARRY and OVERSTATE state
Vector Loads		
Aligned	XD2_L.D16X2S, XD2_L.D16X2S.R, XD2_L.D16X2S.IU, XD2_L.D16X2S.RIU, XD2_L.D16X2S.X, XD2_L.D16X2S.RX, XD2_L.D16X2S.XU, XD2_L.D16X2S.RXU, XD2_L.D16X2S.PXU, XD2_L.D16X2S.RPXU, XD2_L.A16S.X	Loading of vector register (with reversed)
Unaligned	XD2_LA.D16X2S.P, XD2_LA.D16X2S.IU	Loading to and from alignment buffer
Scaler	XD2_LS.D16, XD2_LS.D16.IU, XD2_LS.D16.X, XD2_LS.D16.XU, XD2_L.D16S, XD2_L.D16S.IU, XD2_L.D16S.X, XD2_L.D16S.XU, XD2_L.D32U	Loading of vector from scaler and loading of scaler (16-bits and 32-bits)
Special	XD2_L.U32, XD2_LH.D40	Special loading operations
Vector Stores		
Aligned	XD2_S.D16XS, XD2_S.D16X2.R, XD2_S.D16X2.IU, XD2_S.D16X2.RIU, XD2_S.D16X2.X, XD2_S.D16X2.RX, XD2_S.D16X2.XU, XD2_S.D16X2.RXU, XD2_S.D16X2.PXU, XD2_S.D16X2.RPXU, XD2_SP.D16x2.IU	Storing from vector register (with reversed)
Unaligned	XD2_SA.D16X2.F, XD2_SA.D16X2.IU	Storing from alignment buffer
Scaler	XD2_S.D16, XD2_S.D16.IU, XD2_S.D16.X, XD2_S.D16.XU	Storing of scaler (16-bits)
Special	XD2_S.U32, XD2_SH.D40S, XD2_SH.D40U	Special storing operations

Existing DSP Code Base Support

The ConnX D2 DSP engine includes support for TI C6x C-intrinsics and ITU C-intrinsics. Compiling an application written with TI or ITU C intrinsics merely requires inclusion of a C header file that maps these intrinsics to ConnX D2 instructions. This means that existing code bases (TI or ITU) can be ported quickly to the ConnX D2 DSP engine.

Both the ITU and TI C6x intrinsics are mapped in most cases to a single instruction in the ConnX D2 instruction set, providing for the most efficient execution of the intrinsics. A good example of this is the ITU "mult_r" intrinsic, which is implemented in one MAC-with-rounding instruction of the ConnX D2 ISA.

Tool Support for Processor Designers

Every Xtensa LX processor with (or without) the ConnX D2 DSP Engine is automatically generated with a complete set of software development and modeling tools tailored to the exact Xtensa LX configuration.

Tensilica's [Xtensa Explorer](#) GUI serves as the cockpit for the entire design experience. From Xtensa Explorer, designers can profile their application code, identify "hot spots" that can benefit from acceleration, and make the changes necessary to speed up that code. Using a check-box menu within the GUI, designers can configure processors to include the ConnX D2 DSP Engine and other features they need including options for processor interface, memories, operating system support, EDA scripts, debug and trace, and much more.

Designers can quickly extend the processor's instruction set by adding new instructions using the Tensilica Instruction Extension (TIE) language, a hybrid of C and Verilog, which is the easiest-to-use method for saving power, increasing performance, and reducing clock frequency.

Designers can profile, compare and save many different processor configurations, so they can pick the right one for their application. The ISS or TurboXim can be used for simulations. Also, designers can model and simulate multiple processor subsystems in this environment using Tensilica's XTensa Modeling Protocol (XTMP) or Tensilica's Xtensa SystemC (XTSC) modeling.

Xtensa Explorer serves as the gateway to the Xtensa Processor Generator. Once a processor configuration is finalized, the Xtensa Processor Generator creates the automatically verified Xtensa processor to match all of the configuration options and extensions required in less than an hour. A software tool chain is also created matching all processor modifications that have been made.

Tool Support for Software Developers

The Xtensa Software Developer's Toolkit (SDK) provides a comprehensive collection of code generation and analysis tools that speed the software application development process. Tensilica's Eclipse-based Xtensa Explorer graphical user interface (GUI) serves as the cockpit for the entire development experience and also provides powerful visualization tools to aid application optimization.

The entire Xtensa software development tool chain (the compiler, linker, assembler, debugger, instruction set simulator, etc.), along with simulation models, RTOS ports, optimized C-libraries, etc., are automatically configured for the tailored processor hardware.

Xtensa Core Architecture

The ConnX D2 engine is a configuration option within the Xtensa architecture. This is one of the many DSP-centric configuration options. Other DSP architectural features available include:

- Zero overhead loops. Hardware support for implementing zero latency on loops.
- Dual load/store unit. For connectivity for separate memory units as well as increased data bandwidth
- Data and instruction cache
- Local instruction and data memories

Additionally, Xtensa LX processors can be configured with other DSP engines, including:

- HiFi2 : 24-bit dual MAC for audio applications
- Single-precision floating point unit
- Double-precision floating point calculation assist
- QuadMAC ConnX Vectra LX DSP engine
- ConnX BBE (3.9G and 4G Baseband DSP engine)

Tensilica also offers the ConnX 545CK pre-configured DSP core that uses the QuadMAC ConnX Vectra LX engine for outstanding DSP benchmark results (see www.tensilica.com for details).

Conceptual Block Diagram

Figure 2 shows a simplified example of the Xtensa LX core configured with the ConnX D2 engine.

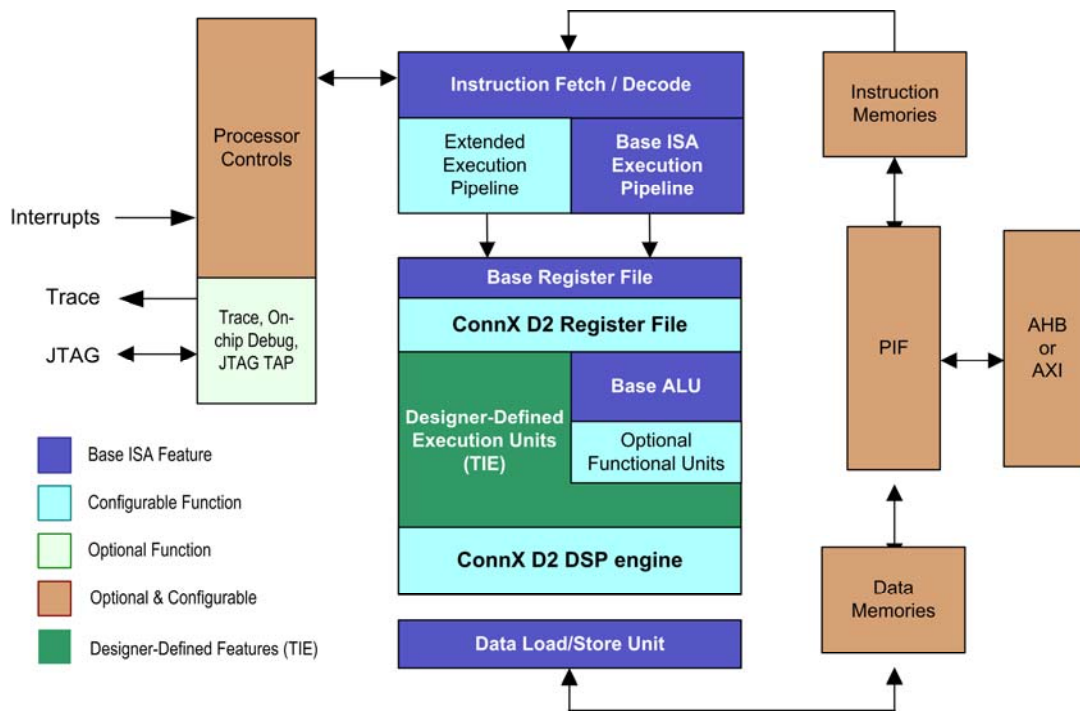


Figure 2. Simplified block diagram of Xtensa LX with ConnX D2 DSP engine.

Complete with Reference Cores

We've put together two reference configurations of the Xtensa LX with ConnX D2. These reference cores are available within the Xtensa Xplorer tool suite, letting you get up and running in a very short time.



Tensilica, Inc.

3255-6 Scott Blvd., Santa Clara, CA 95054-3013, USA
 Tel: 1-408-986-8000 Fax: 408-986-8919 Website: www.tensilica.com

©June 2012, Tensilica and Xtensa are registered trademarks of Tensilica, Inc.
 The Tensilica logo, Xenergy, Xplorer and TurboXim are trademarks of Tensilica, Inc. All other trademarks are the property of their respective owners.