cadence®

Cut DSP Development Time—Use C for High Performance, No Assembly Required

Digital signal processing (DSP) IP is increasingly required to take on complex processing tasks in signal processing-intensive applications. While designers have traditionally used assembly code programming to ensure high performance, advances in DSP architectures and compilers make it possible for developers to keep their algorithms in C and still meet performance requirements. This white paper illustrates how a Cadence® Tensilica® ConnX D2 DSP coupled with the advanced Cadence Tensilica Xtensa® C/C++ Compiler (XCC) achieves better DSP performance using standard C rather than assembly code, helping you finish your project much faster.

Contents

Overview

Application requirements are driving designers to ask their digital signal processing (DSP) IP to perform more and more of the heavy workload required for highly complex algorithms in fast Fourier transform (FFT) filtering, multiple-input and multiple-output (MIMO) processing, and other signal processing-intensive applications. To ensure high performance from a conventional DSP, developers have traditionally used assembly code programming, which is time-consuming and difficult to maintain.

Advances in state-of-the-art DSP architectures and companion compilers now make it possible for developers to keep their algorithms in C and still achieve the performance they need from a general-purpose high-performance DSP.

The magic is in the compiler technology. This white paper shows how the advanced Cadence Tensilica Xtensa C/C++ Compiler (XCC) can yield equivalent or better performance using standard C compared to other DSPs programmed in assembly code. This allows you to complete your design significantly faster and gives your design additional flexibility for future implementations. This white paper illustrates how a Cadence Tensilica ConnX D2 DSP—based on the Xtensa architecture—coupled with the XCC achieves higher performance from native C-code compared to an industry-standard DSP core running hand-optimized assembly code.

DSP software developers have traditionally converted key performance-critical portions of their algorithms to assembly language because that was considered the only way to achieve high performance when using a DSP. Every DSP architecture is different—optimized for a different type of data throughput challenge—and programmers need to understand each underlying DSP architecture to optimize the code manually using assembly coding techniques. Thus, specialized knowledge is required to achieve optimal results.

Assembly programming also locks code into a specific DSP platform by targeting that DSP's specific instruction set architecture (ISA). The developer loses flexibility in choosing cores for future projects that need to reuse code.

Most developers use C to quickly create and test software—the language that most algorithms are originally developed in—but when using DSPs this is often not possible because most C compilers cannot efficiently map algorithms to DSP instruction sets. Designers today expect to get better out-of-the-box performance from their DSPs to reduce the amount of tuning needed to achieve acceptable performance. If the amount of code that needs precise tuning is small, assembly coding can be an acceptable solution. But as application programs have become larger and more complex and as the number of industry standards has multiplied over the years, the need for a purely C-based solution has escalated.

The most common first step in the evolution from an assembly-level to a C-code level is the use of C intrinsics.

Using Intrinsics in C Code for Performance

Because most C compilers cannot take full advantage of high-performance DSP architectures, intrinsic instructions can be used within C to represent assembly instructions. A compiler maps the intrinsic instructions to specific DSP instructions when the intrinsics are used in the code. Otherwise, the compiler falls back to a default implementation provided by the language run-time environment. Intrinsics have proven popular for specific application areas, such as the International Telegraph Union (ITU) Telecommunication Standardization Sector/European Telecommunications Standards Institute (ITU-T/ETSI) modules in voice codec algorithms where designers have successfully alleviated machine dependencies.

The use of intrinsic functions is an improvement over being required to write all DSP code in assembly, but has limitations. First, by using an intrinsic function, the code is most likely no longer fully portable to other DSP architectures unless a full C function equivalent is provided by the compiler vendor. Further, intrinsic function names for the same logical function are not the same in different architectures, thus requiring significant recoding for portability.

Second, an intrinsic function's name may not be the best possible match to what that function does. Therefore, code management with extensive documentation is required to keep track of what functions do. It is a challenge to make sure programmers are using the proper intrinsic for the exact function required.

Making a Better Compiler

Most DSP compilers are optimized for vectorization and are good at accelerating loop-based algorithms, which are typical for most DSP applications. However, because of increasing algorithm complexity and multiple standard support, most DSP source code keeps getting bigger and more difficult to manage. With the increase in algorithm-variation, it is less likely that all the code is vectorizable. This is especially true for control code as well as algorithms that are not implemented in a single incrementing loop. In these cases, the single instruction, multiple data (SIMD) engine cannot be used effectively and most DSPs cannot deliver optimal performance from C.

The compiler technology behind the Xtensa customizable processors is optimized for both DSP and control code.

Using Two Parallel Execution Routes

The Tensilica ConnX D2 DSP based on the Xtensa processor is an ISA option that adds a 16-bit dual multiplier accumulator (MAC) SIMD and dual-instruction issue functionality, using a form of VLIW with variable slot widths called flexible length instruction extensions (FLIX). This architecture enables two execution routes in parallel for instruction processing, offering performance from parallelism whatever the algorithm. The Xtensa processors allow you to define a FLIX implementation that is best suited for your application needs. The ConnX D2 DSP adds instructions to the Xtensa base ISA, which is optimized for efficient control code execution.

When the compiler recognizes loops that are not vectorizable in the ConnX D2 DSP, the SIMD engine resource dynamically reallocates across the two parallel execution units within the FLIX architecture. Therefore, arithmetic operations are executed in parallel, giving up to twice the performance for non-vectorizable code compared to non-VLIW DSPs. This effectively creates a multi-way data parallelism between the SIMD and FLIX approach to parallelism giving optimum performance, whatever the algorithm.

Refer to the Increasing Processor Computational Performance with More Instruction Parallelism white paper for more details.

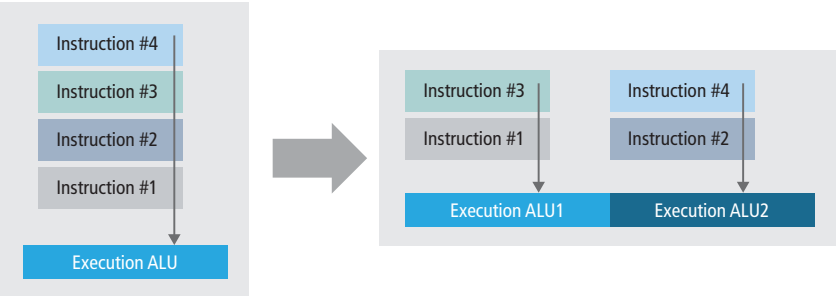

Figure 1: FLIX Provides Twice the Performance by Executing Two Instructions in Parallel in the ConnX D2 DSP

An Instruction Set Optimized for DSP

The ConnX D2 DSP instruction set is specifically optimized for the demanding computations required for digital signal processing. By adding DSP-specific instructions to the Xtensa base instruction set, the processor efficiently performs 16-, 32-, and 40-bit fixed-point additions, subtractions, and multiplications 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 DSP instructions include:

- Add-bit-reverse-base, add-subtract—Useful for FFT
- Add-compare-exchange—Useful for Viterbi
- Add-modulo—Useful for finite impulse response (FIR)

When used in conjunction with a bit reverse addressing scheme, this instruction set executes FFT algorithms quickly. All of these DSP-centric features are automatically utilized by the compiler to further increase the performance of DSP-centric algorithms.

Improved Performance

The ConnX D2 DSP supports TI C6x and ITU-T intrinsics. In most cases, there is a one-to-one mapping of the TI and ITU intrinsic to a ConnX D2 instruction. For example, the ITU intrinsic "mult_r" maps to the XD2_QMUL16_RND() instruction, which performs multiplication with rounding in one instruction. The mapped TI C6x instrinsics are bit-for-bit equivalent to the TI C6x. Therefore, ported algorithms running on ConnX D2 should give the same bit functionality with no need for code tweaking.

Cadence took the original ITU/ETSI source C reference code for the AMR-NB (VAD2) audio codec (encoder plus decoder) algorithm and compiled it for the ConnX D2 DSP. It required just 27.7MHz, beating competitive DSPs by as much as 2X when compiling the same original source code.

With a 256-point FFT, the ConnX D2 DSP supplied 20% better performance from native C-code compared to an industry-standard DSP running hand-optimized assembly code.

Assembly code generated by the compiler for the main loop of this 256-point FFT is shown below. The compiler uses the parallel execution in the two FLIX issue slots of the processor for results that rival handwritten code. With this compiler, there is no need to re-write the assembly code to get the performance you need.

```
loopgtz a7, label
  xd2 _ l.d16x2s xdd0,a2,4;      xd2 _ cmul.rfs.dc16s  xdd3,xdd3,xdd0
  xd2 _ l.d16x2s xdd1,a3,8;      xd2 _ sra.d16x2s      xdd4,xdd1,xdd7
  xd2 _ l.d16x2s xdd2,a4,8;      xd2 _ cmul.rfs.dc16s  xdd6,xdd6,xdd2
  xd2 _ l.d16x2s xdd3,a5,8;      xd2 _ sra.d16x2s      xdd5,xdd3,xdd7
  xd2 _ s.d16x2.iu xdd4,a3,4;    xd2 _ addsub.d16x2s   xdd0,xdd4,xdd0,xdd2
  xd2 _ l.d16x2s xdd2,a6,8;      xd2 _ sra.d16x2s      xdd6,xdd6,xdd7
  xd2 _ s.d16x2.iu xdd5,a4,4;    xd2 _ sub.d16x2s      xdd5,xdd3,xdd1
  xd2 _ s.d16x2.iu xdd6,a5,4;    xd2 _ addsub.d16x2s   xdd1,xdd3,xdd1,xdd3
  nop;                           xd2 _ addsub.d16x2s   xdd1,xdd0,xdd0,xdd1
  xd2 _ vsel.d16x2 xdd3,xdd5,xdd3,9; xd2 _ sra.d16x2s xdd1,xdd1,xdd7
  xd2 _ s.d16x2.iu xdd1,a2,4 ;   xd2 _ addsub.d16x2s   xdd6,xdd3,xdd4,xdd3
  xd2 _ l.d16x2s xdd0,a6,4 ;     xd2 _ cmul.rfs.dc16s  xdd1,xdd0,xdd2
  xd2 _ l.d16x2s.iu xdd2,a6,12; nop
```

Complete C-Based Design Flow for DSPs

The Cadence advanced compiler technology enables a significantly faster development time, as shown below.

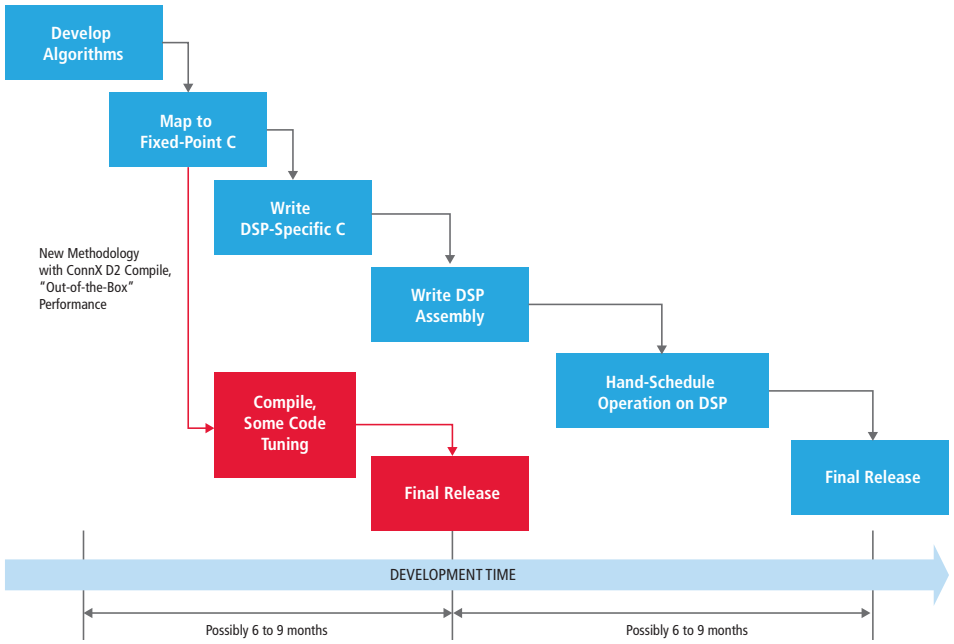

Figure 2. Faster Design Performance in C with the ConnX D2 DSP

This process eliminates the time-consuming steps involved in writing DSP-specific C and assembly code. Standard C code can be used "out-of-the-box" with excellent results, greatly reducing development time and helping get products to market much faster. Further, if changes are required at the last moment, the C code can be changed and updated much more easily than assembly code, which reduces cost and risk.

Additional Resources

Cadence has the largest family of architecturally compatible DSPs in the industry. The Xtensa processor development tools allow you to choose from existing DSP options or create a custom processor that can be optimized for your specific application. Our tools make it easy to create the ideal instruction set and will automatically generate fully synthesizable RTL with scripts for fabrication and the full matching software development tool chain to speed your time to market.

Conclusion

You no longer have to resort to assembly or C intrinsics to achieve best results for DSP applications—Xtensa processors, such as the Cadence ConnX D2 DSP based on the Xtensa architecture, provide better performance from native C-code compared to hand-optimized assembly code. Our advanced compiler can efficiently schedule instructions across multiple FLIX issue slots when an algorithm cannot be vectorized, giving the benefit of great performance from "out-of-the-box" C code.

Additional Information

For more information on the unique abilities and features of Cadence Tensilica Xtensa processors, see ip.cadence.com.