Cut DSP Development Time – Get High Performance From C, No Assembly Required

Designers are asking their DSP cores to do more and more of the heavy workloads required for highly complex algorithms for filtering, FFT, MIMO, and other signal processing intensive applications. To get high performance from a conventional DSP core, 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 get the performance they need from a general-purpose high-performance DSP.

The magic is in the compiler technology. This white paper explains how Tensilica's ConnX D2 DSP engine coupled with the advanced Tensilica XCC compiler can help you get equivalent or better performance using standard C than other DSPs programmed in assembly code. The result: you can get your project finished much faster.

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 core. Every DSP architecture is different – optimized for a different type of data throughput challenge – and programmers need to understand each underlying DSP architecture in order to optimize the code manually using assembly coding techniques. Thus specialized knowledge is required to achieve effective 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. Why haven't they been staying in C - the language that the algorithm was probably developed in originally?

The answer is simple. Most C compilers cannot efficiently map algorithms to DSP instruction sets aimed at accelerating targeted algorithms. 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.

How can the usage paradigm move from an assembly-level to a C-code level? Let's first examine the most common first step in that evolution – the use of C intrinsics.

Using Intrinsics in C Code for Performance

Because most C compilers can't 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 core instructions, but only uses those special functions 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 applications areas, such as the ITU-T/ETSI modules in voice codec algorithms.

The use of intrinsic functions is certainly an improvement over being required to write all DSP code in assembly, but it does have limitations. First, by using an intrinsic function the code is most likely no longer fully portable, unless a full C function equivalent is provided by the compiler vendor. Further, intrinsic functions 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 answer the question – now what does that function do again? It's easy to get confused and challenging to make sure programmers are using the proper intrinsic for the exact function required.

Making a Better Compiler

Most DSP compilers are well optimized for vectorization and they do a great job accelerating loop-based algorithms, which are typical for most DSP applications.

The problem is that, because of increasing algorithm complexity and multiple standard support, most DSP code sizes keep getting bigger and more difficult to manage. With the increase in algorithm variation, the likelihood of all the code being vectorizable reduces. This is especially true for control code as well as algorithms that are not implemented in a single incrementing loop. In these cases, the SIMD engine cannot be used effectively and most DSPs cores can not deliver optimal performance from C.

Tensilica took a hard look at this when developing the compiler technology used with the ConnX D2 DSP engine. It's based on the same compiler technology behind Tensilica's Xtensa customizable processors. Tensilica has been working on this technology for over 10 years now, and has refined it for both DSP and control code.

Using Two Parallel Execution Routes

The ConnX D2 DSP engine is a 16-bit dual-MAC SIMD core implemented with a VLIW architecture that offers two parallel execution routes. This true VLIW implementation offers parallel performance whatever the algorithm. The ConnX D2 DSP engine is based on the Xtensa customizable processor core, so it includes the

full Xtensa base instruction set that is optimized for efficient control code execution.

In the ConnX D2 DSP engine, when the compiler recognizes loops that are not vectorizable, the SIMD engine resource can be dynamically reallocated across the two parallel execution units within the VLIW architecture. Therefore, arithmetic operations can be executed in parallel, giving up to twice the performance for non-vectorizable code compared to regular SIMD DSP cores.

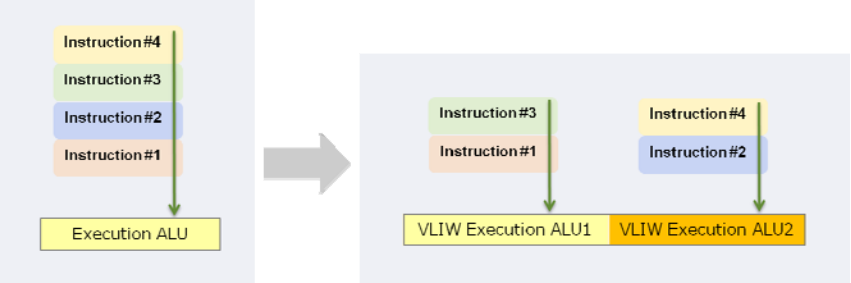

Figure 1. VLIW provides twice the performance by executing two instructions in parallel.

This effectively creates a new type of DSP that can be considered a hybrid between the SIMD and VLIW to give optimum performance, whatever the algorithm.

An Instruction Set Optimized for DSP

The ConnX D2 DSP engine instruction set is specifically optimized for the demanding computations required for digital signal processing. By adding DSP-specific instructions to the Xtensa LX base instruction set, the ConnX D2 DSP engine 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 DSP engine instructions include:

- Add-Bit-Reverse-Base, Add-Subtract – Useful for FFT implementation
- Add-Compare-Exchange – Useful for Viterbi implementation
- Add-Modulo – Useful for FIR implementation

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

The Performance You Can Get

The ConnX D2 DSP engine supports TI C6x and ITU-T intrinsics. In the majority of cases there is a one-to-one mapping of the TI and ITU intrinsic to a ConnX D2 instruction. An example of this is the ITU intrinsic “mult_r” which 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.

Tensilica 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 engine. It required just 27.7 MHz – beating competitor’s DSP cores by as much as 2X compiling the same original source code.

Then we took a 256-point FFT. The ConnX D2 DSP engine got 20% better performance from native C-code compared to an industry standard DSP core running hand-optimized assembly code.

Just take a look at the assembly code generated by the ConnX D2 DSP's compiler for the main loop of this 256-point FFT. The compiler uses the Add-Subtract instruction, with the parallel execution in the two FLIX slots of the ConnX D2 DSP engine. This is nice, efficient code that rivals handwritten code. With this compiler, there's 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
```

A Totally C-Based Design Flow for DSPs

The big benefit of Tensilica's advanced compiler technology is a much faster development time, as shown below:

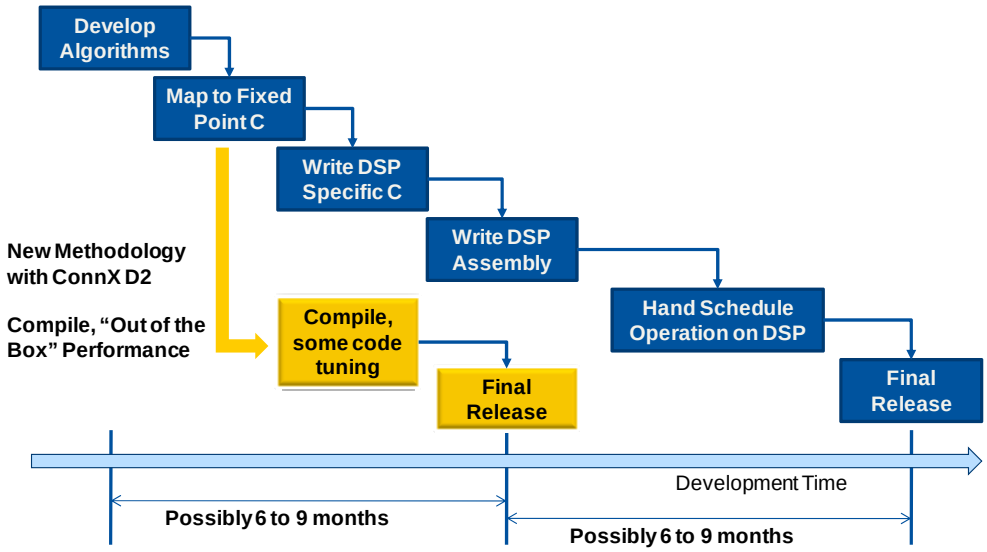

Figure 2. Design Much Faster in C with the ConnX D2 DSP Engine

The time consuming steps involved in writing DSP-specific C and assembly code are eliminated. Standard C code can be used "out of the box" with excellent results.

This greatly reduces development time and helps get products to market much faster.

And if changes are required at the last moment, then the C-code can be changed and updated more easily than assembly code.

Need More than a 2-MAC DSP?

Maybe a 2-MAC ConnX D2 DSP won't give you the DSP boost you need. Check out Tensilica's web site. We have the largest family of architecturally compatible DSP cores in the industry. Or you can use our automated tools and create a DSP all of your own, perfect for your specific application. Our tools will automatically generate a full matching software tool chain so you don't have to.

Conclusion

There's no longer a requirement to program in assembly language or C intrinsics now that Tensilica has introduced the ConnX D2 DSP

engine. Tensilica's advanced compiler can efficiently schedule instructions across the two execution units of its VLIW architecture when an algorithm cannot be vectorized. This gives the benefit of great performance from "out of the box" C code.