Optimizing C Code For Modern DSP Compilers

Numerix Ltd.

May 2005

Numerix Ltd.
7 Dauphine Close, Coalville, Leics, LE67 4QQ, UK
Phone : +44 (0)7050 803996, Fax : +44 (0)7050 803997
Internet : http://www.numerix-dsp.com
Email : support@numerix-dsp.com

Introduction

SigLib is an ANSI C source library of DSP functions that is designed to include a large number of DSP functions and to be portable across as many different architectures and operating systems as possible. While this architecture makes the library idea for prototyping and development, many applications require optimum performance.

This applications note looks at the issues related to using SigLib with modern DSP devices such as the TMS320C67x and SHARC architectures. Modern compilers support functionality to indicate varioius details about the algorithm that are not covered by the ANSI C standard these features can be used to optimise the code for specific architectures.

Implementation Details

This applications note looks at the real and complex dot product operations and also the IIR filter function.

The IIR filter implementation deserves a short description. Diagram 1 shows the standard IIR biquad filter structure. If the C code for this is compiled then there are some dependencies that preclude the compiler from generating optimised code.

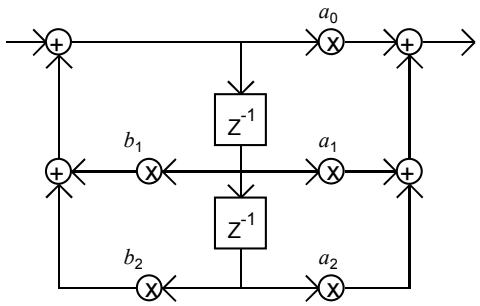

Diagram 1 : The Standard IIR Biquad Filter Structure

The optimised IIR filter structure (shown in diagram 2) utilises a modification of the standard IIR biquad structure, where the feedforward coefficients are scaled so that a0 = 1.0. This results in a structure that can fully utilise the parallelism of the TMS320C67x architecture but a consequence of this is that the output is scaled compared to the standard IIR filter by the value used to scale a0. Being 1.0, the a0 coefficient can be removed from the coefficient array and the new array has the following coefficients for each biquad : A1, A2, B1, B2. To use these functions you must also ensure that the coefficient and state arrays are aligned on a suitable 64 bit memory alignment.

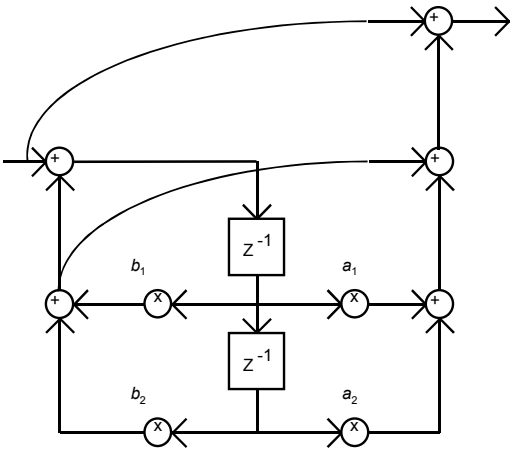

Diagram 2 : The Optimised IIR Biquad Filter Structure

For further details, please refer to the “Using SigLib With Fast IIR Filter Functions for the TMS320C6x DSPs” applications note that is also provided .

The Source Code

The source code for this applications not is included in the file “dspbench.c”. This can be used on the TMS320C67x and SHARC simulators or on real hardware. To compile and link the program for the TMS320C67x, you will neen to have the TMS320C6x tool set installed. Then use the following batch file on the command line :

```
> clrfas FastIIR
```

To compile and link the program for the SHARC, please use the project file “dspbench.dpj”.

Optimisation Features Used

For the TMS320C67x DSP the following optimisation features were used.

Memory alignment to allow the use of the LDDW instruction :

```
#pragma DATA_ALIGN(a, 8);          /* Align arrays on 64 bit boundary for LDDW
*/
```

The `_nassert` to inform the compiler that the data is suitably aligned :

```
_nassert (((int)(a) & 0x7) == 0);  /* Float array aligned on 64-bit boundary */
```

The `MUST_ITERATE` pragma to inform the compiler about the minimum loop count and the modulus of the count :

```
#pragma MUST_ITERATE (16, , 8);    /* Minimum trip count and execute multiple of
8 times */
```

For the SHARC DSPs the following optimisation features were used.

Memory alignment to allow the use of optimised data loads :

```
#pragma align 8
```

Use of program memory for locating data arrays :

```
float pm a[] = {
```

The `SIMD_for` pragma to indicate that SIMD operations should be used :

```
#pragma SIMD_for
```

Benchmark Results

TMS320C67x :

- Real dot product : 4 cycle kernel with 4 MACs
- Complex dot product #1 and 2: 4 cycles per complex MAC
  - Note - The compiler did not like the complex data types
- Complex dot product #3: 4 cycles per PAIR of complex MACs
- Biquad 1 : 8 cycles per biquad
- Biquad 2 : 4 cycles per biquad

ADSP2116x :

- Real dot product : 1 cycle per real MAC
- Complex dot product #1 and 2: 6 cycles per complex MAC
  - Note - The compiler did not like the complex data types
- Complex dot product #3: 4 cycles per PAIR of complex MACs
- Biquad 1 : 4 cycles per biquad
- Biquad 2 : 3 cycles per biquad

Conclusion

An analysis of the assembly code generated by these simple optimization techniques shows that several of the functions are equal to hand optimised assembly code and those that are not as efficient still exhibit a significant performance improvement over the use of standard ANSI C code.

The techniques presented in this applications note can be used to optimise most of the functions available in the SigLib DSP library. The devices covered in this applications note are traditional DSPs however the techniques are equally applicable to C compilers for many modern CPUs.