Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

Implementing In-Place FFTs on SISD and SIMD SHARC® Processors

Contributed by Kunal Singh

Rev 1 – March 29, 2005

Introduction

ADSP-21065L and ADSP-21161 processors belong to the SHARC® family of processors. The ADSP-21065L is based on a SISD architecture, and the ADSP-21161 is based on a SIMD architecture. This application note discusses the implementation of in-place FFTs and provides ASM code examples for the SISD and SIMD SHARC processors.

An "in-place" FFT is an FFT that is calculated entirely inside its original sample memory. In other words, calculating an "in-place" FFT requires no additional buffer memory (as do some FFTs). An in-place FFT computation may be used when a system is constrained by memory.

DIT vs. DIF FFT Routines

An FFT decomposed using DFTs of even and odd points is called a decimation-in-time (DIT) FFT. It can also be decomposed using a first-half/second-half approach, which is called a decimation-in-frequency (DIF) FFT.

The DIF FFT algorithm operates on the in-place data (all data points are arranged sequentially in the memory buffer), and the results of the FFT computations are stored in a scrambled fashion (the results are stored in the same buffer, but with *bit-reversed addresses*) and must be un-scrambled.

The given example code uses a radix-2 DIF FFT algorithm for FFT computations.

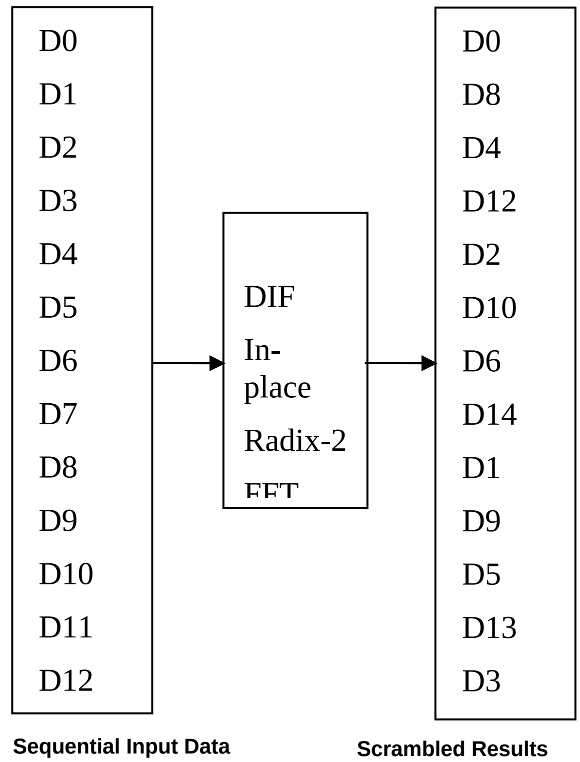

Figure 1. Sequential Input Data vs. Scrambled Results

Figure 1 shows the arrangement of the results in the memory buffer after the DIF FFT computation is performed on the sequential data. The results are bit-reversed.

Since much literature about implementation of radix-2 FFT computation is available, this application note does not discuss the details of FFT algorithm and its implementation. This document discusses three different approaches that can be used to unscramble the data points while implementing the in-place FFTs.

DIF FFT Computation

Consider a 16-point FFT. Figure 1 shows how the input data points and the results are stored in memory for a DIF FFT computation. For an in-place FFT computation, the results would be stored in the same memory buffer in which the input operands are available. No extra memory is utilized as a temporary storage buffer for the FFT computation.

Unscrambling the Results

As shown in Figure 1, the computation results of the DIF FFT computations would be in scrambled fashion and must be unscrambled. Different unscrambling schemes may be adopted providing a trade-off between MIPS and memory usage. Three approaches are described next.

Approach 1

As seen from shown in Figure 1, while unscrambling the results, you must interchange the position of results in the memory buffer. For example, the location of the result D8 would be interchanged with the result D1. While unscrambling the data points, you must ensure that each data point is traversed only once (traversing it twice would nullify any change). A simple approach is to traverse the memory buffer sequentially. The index of data point would be bit-reversed and compared with the actual value of the index. If the bit-reversed value is greater than the actual value of the index, the data points (the data pointed by actual index and the data pointed by the bit-reversed value of the index) would be interchanged. This approach does not require additional memory space, but it would consume a significant number of MIPS and may not be a suitable (practical) implementation (due to the very high MIPS consumption).

Approach 2

As is evident from the discussion in Approach 1, decision-making logic (to decide whether the data at a particular index has to be unscrambled) consumes lots of MIPS. An alternative approach is to pre-compute the logic and store it in the form of a decision table. Thus, a particular entry in the decision table could suggest whether the result data point at a particular location in the memory buffer must be unscrambled. This approach would consume some more memory (to store the decision table), but the MIPS required for decision logic would be saved. The only MIPS overhead would be for accessing a particular word in the decision table, testing a bit field from this word, and then performing a data unscramble operation (which involves address bit-reversal).

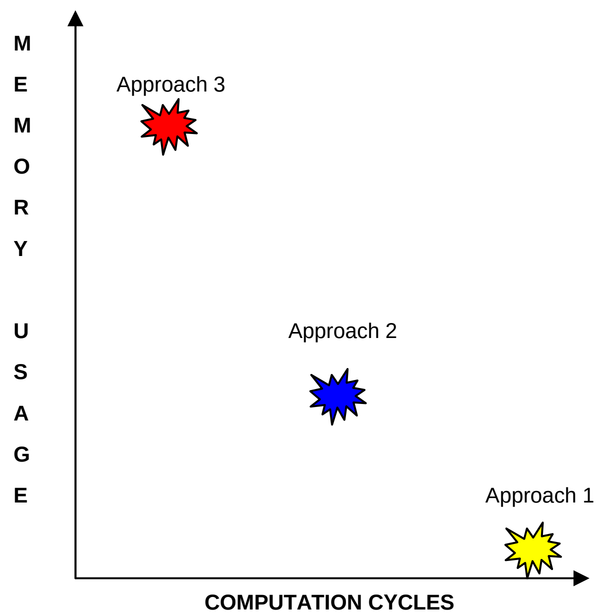

Figure 2. Comparison of Unscrambling Approaches

Approach 3

Another approach for unscrambling the data to is to store the unscrambling addresses in a look-up table. These addresses can be directly read from the look-up table, and data unscrambling from these addresses can be performed. This approach would further optimize the MIPS; however, it would consume more memory (the look-up table would be much larger than the decision table).

Since the SHARC processor architecture supports two parallel data fetch operations, using two look-up tables (one for the actual address and the second for the bit-reversed address) can further optimize MIPS usage.

The example code attached to this application note uses Approach 3 for data unscrambling. A further modification was to store the differences between the consecutive addresses (offset of the two addresses) in the look-up table, rather than storing the addresses. Thus, there is no dependency on the actual physical addresses where data is stored in the memory.

Conclusions

Figure 2 compares the three approaches in terms of MIPS and memory usage. As is evident from the figure, Approach 3 results in optimum MIPS. However, the memory overheads (to store the look-up tables) are comparatively higher in this case.

Table 1 depicts the MIPS count (for FFT calculation and unscrambling) and memory overheads for an N-point FFT on SISD and SIMD SHARC processors.

Sr. No.	N (FFT points)	(ADSP-21161)		(ADSP-21065L)	
		Cycles	Memory (Bytes) For look up tables	Cycles	Memory (Bytes) For look up tables
1.	64	1156	112	1438	112
2.	128	2158	224	2816	224
3.	256	4316	489	5873	480
4.	512	8770	960	12386	960
5.	1024	18288	1984	26563	1984
6.	2048	38158	3968	56772	3968
7.	4096	80188	8064	NA	NA

Table 1. Performance Analysis of Attached Code on SISD and SIMD SHARC Processors

Appendix

The attached ZIP file contains the following code:

1. In-place radix-2 DIF FFT on an ADSP-21161 in assembly language
2. In-place radix-2 DIF FFT on an ADSP-21065L in assembly language
3. Twiddle factor generation
4. Look-up table generation (to be used to unscramble the data)
5. Test data generation (to be used to test the code in items 1 and 2)

References

[1] *ADSP-21161 SHARC DSP Hardware Reference*. Third Edition, May 2002. Analog Devices, Inc.

[2] *ADSP-2065L SHARC DSP User's Manual*. September 01, 1998. Analog Devices, Inc.

[3] *Digital Signal Processing*. Third Edition, 2003. John G. Proakis and Dimitris G. Manolakis

Document History

Revision	**Description**
Rev 1 – March 29, 2005 *by Kunal Singh*	Initial Release