SECTION 5

Optimizing Performance of the FFT

5.1 Optimization

"Optimization saves . . . 2067 instruction cycles which equals about 10% cycle time of the optimized code."

Judging the performance of any program requires the consideration of both its time and space complexity. There is always a trade off between these two aspects. Time complexity indicates how fast an algorithm can be implemented on a specified microprocessor, while space complexity tells how much memory may be required. Optimization can either reduce memory requirement or minimize runtime of an algorithm. Since memory costs are continually decreasing, time optimization becomes more and more important.

One way to evaluate the time complexity of an algorithm is to compare its theoretical complexity, ideal implementation complexity, and practical complexity. Theoretical complexity refers to the number of additions and multiplications required by the given algorithm, independent of the microprocessor's architectures. This type of evaluating is only good for high-level comparison among algorithms and does not reflect the real performance of the algorithm on a given microprocessor. Not surprisingly, an algorithm that retains a lower theoretical complexity has a higher ideal

implementation complexity. Ideal implementation complexity considers only the implementation of the core algorithm by the given microprocessor's instruction capabilities, such as available instruction type, addressing mode, parallel data move, etc. Ideal implementation complexity indicates the non-overhead performance of a given algorithm on a microprocessor, and always provides an optimistic estimation of an algorithm's performance. Practical complexity denotes the ideal implementation complexity plus the structure overhead of the microprocessor. (Structure overhead includes all required instructions not associated with the core algorithm.) Moving pointers, setting up DO loops, jumps to subroutines, and conditional jumps are typical structure overhead in microprocessors.

By distinguishing the different complexities, one can easily determine which microprocessor is competent for each aspect, and which instruction or address mode is critical to the specific algorithms. Also, chip designers may derive clues from the complexity analysis for determining which instruction or address mode should be added to the next revision. For example, the DSP96002 supports FMPY||ADD||SUB — an instruction with two parallel moves. The theoretical complexity of a radix-2 butterfly is four real multiplications and six additions or subtractions. Thus, the ideal implementation complexity of a radix-2 FFT on the DSP96002 is four instruction cycles. If each butterfly needs an average of 0.25 instructions to set up a pointer or DO loop, etc., the practical complexity of radix-2 is 4.25 instructions. The ratio of ideal implementation com-

plexity to practical complexity reflects the efficiency of a microprocessor to perform a specific function. For example, the efficiency of the DSP96002 performing a radix-2 complex FFT could be:

$$\text{efficiency} = \frac{\text{ideal implementation complexity}}{\text{practical complexity}} = \frac{4}{4.25} = 0.94$$

Eqn. 5-1

In other words, the structure overhead for this particular example is about 6%. For FFTs implemented on programmable DSPs, the structure overhead should be between 3% and 15%. If a DSP has structure overhead higher than 15%, it can not be called a DSP. If one claims a structure overhead lower than 3%, it is probably an application specific integrated circuit (ASIC).

5.1.1 Minimum Memory Requirement — In-Place Calculation

Although each radix-2 butterfly has two complex input data and two complex output data, calculation of the butterfly can be done by using only one memory set called in-place calculation. Memory requirements may be minimized by:

- ***Reordering data into bit-reversed order.*** This can be done in-place since data is interchanged by pairs, as seen in Figure 4-9. Thus, only 2N real data locations are required.

- ***Reducing the size of the twiddle factor table*** from N real locations to N/2 real locations for normal order input DIT FFT (see reference 8). Notice that in normal order input DIT FFT the order that accesses the twiddle factor table is bit-reversal, i.e.

$$W_N^0, W_N^{(N/2)-1}, W_N^{N/4}, W_N^{N/8}, W_N^{(3N)/8}, \ldots, W_N^{(N/4)-1}$$

  N/2 complex numbers can be combined in pairs of two, which differ by a factor $W_N^{N/4} = -j$. In other words, the second twiddle factor in the pair can be obtained by multiplying -j with the first twiddle factor. In fact, this optimization can be implemented with a minor modification to the previous butterfly core. All odd indexed groups will use negated, real and imaginary exchanged twiddle factors from the previous even indexed groups. Therefore, the number of groups in a pass is reduced to half of the previous one and the access time of twiddle factors is also reduced to half of the previous one.

- ***Using a triple-nested DO-loop FFT*** to minimize the program memory space (as seen in Figure 4-5). Items 1 and 2 above save data memory space for the FFT calculation only.

5.1.2 Optimization for Faster Execution

Although the previously discussed program executes very efficiently, some applications may impose less stringent requirements on program memory size, but demand even faster execution. Faster execution can be obtained by further optimizing the previous algorithm. The following pages present several steps to achieve this optimization.

1. Since the first and second passes have trivial twiddle factors:

$$W_N^0 = 1, \text{ and } W_N^{N/4} = -j$$

it is common to combine the first and second passes as one radix-4 pass by calculating N/4 butterflies in the following equations.

$$\begin{aligned}
Ar' &= Ar + Cr + Br + Dr \\
Br' &= Ar + Cr - (Br + Dr) \\
Cr' &= Ar - Cr + (Bi - Di) \\
Dr' &= Ar - Cr - (Bi - Di) \\
Bi' &= Ai + Ci - (Bi + Di) \\
Ci' &= Ai - Ci - (Br - Dr) \\
Ai' &= Ai + Ci + Bi + Di \\
Di' &= Ai - Ci + (Br - Dr)
\end{aligned} \qquad \text{Eqn. 5-2}$$

Notice that there are eight additions and eight subtractions in . A DSP that has a multiplication and accumulation instruction with one or two parallel moves (type A DSP) may take at least sixteen instructions to do . A DSP that has a FMPY||ADD||SUB instruction with two parallel

data moves (type B DSP) can do in eight instructions. After combining the first two trivial passes as a radix-4 pass, the number of instructions required in the radix-2 DIT complex FFT becomes:

$$(TRIV \times N/4) + [(m-2) \times N/2 \times BFLY]$$

where: TRIV is the number of instructions necessary to perform a trivial butterfly

Theoretically, for the DSP56001/2, the DSP96002, and the DSP56156, TRIV may be 16, 8, and 16 instruction cycles, respectively. Therefore, a 1024-point complex FFT on the DSP96002 can be done in (8 x 256) + (8 x 512 x 4) = 18,432 instruction cycles. This is a lower boundary of the radix-2 complex FFT. In fact, TRIV is 17, 8, and 22 on the DSP56001, the DSP96002, and the DSP56156, respectively. Cycle time of the FFT can be reduced further by exploring the simple relations among the remaining passes.

2. Trivial twiddle factors exist in the remaining passes as well. Special butterflies can take advantage of those simple relations. There are two types of trivial twiddle factors:

Type I $\quad W_N^0 = 1, W_N^{N/4} = -j$

Type II $\quad W_N^{N/8} = -W_N^{(3N)/8} = 0.707 - j0.707$

Type I trivial factors don't involve multiplications as already shown in Eqn. 5-2. To utilize these simple relations in the remaining passes, different butterflies must be inserted in one

pass. This change results in longer program code and some structure overhead, such as updating address registers, different DO loops, and modulo addressing.

Type II trivial factors are not really trivial for either type A or type B DSPs. Type II trivial factors reduce the theoretical complexity of a radix-2 butterfly to two real multiplications and six real additions or subtractions. With only one adder on type A DSPs, six instructions are required as before. The ideal implementation complexity could be 3 for type B DSPs, but unfortunately each radix-2 butterfly deals with four real inputs and four real outputs. Type B DSPs have only two parallel data moves, and each radix-2 butterfly still takes at least four instruction cycles for type II trivial factors. The type II trivial factor issue is addressed here because this is probably the last chance for further optimizing radix-2 FFTs.

Each group in the last pass consisted of a single butterfly. A triple nested DO loop is thus no longer required in this pass: it can be split and handled by a single DO loop.

3. Another alternative is to combine the last two passes into one radix-4 pass. Since each butterfly in the last pass requires a different twiddle factor, one instruction to fetch a twiddle factor must be appended in the butterfly core. The same fetch occurred in the second to last pass in every two butterflies. Combining four radix-2 butterflies into one radix-4 butterfly may save four multiplications but a special twiddle

factor table has to be created for the radix-4 butterfly.

4. For longer FFTs (>256 points), internal memory in the DSP56001/DSP56002 is not sufficient to contain the complete data set. Consequently, the butterflies execute more slowly when the processor needs to fetch a data value in external X and in external Y memory in the same instruction cycle. This causes the instruction cycle to be “stretched”, resulting in slower execution time. Through intelligent memory usage, however, this effect can be minimized. In a further optimized routine (see ***Appendix A***), the first two passes are combined into a single pass. Next, separate 256-point FFTs are computed, whereby the data is moved into internal memory, and the results are not moved to external memory until the final pass. This process avoids the stretching of the instruction cycle on the middle passes, and makes optimal use of the available internal memory.

With these optimizations, a significantly faster routine is obtained. For instance, a 1024-point optimized complex FFT routine is available for DSP96002 which executes in 0.94 ms at 40MHz clock (see Fully Optimized Complex FFT in ***Appendix A***). A fully optimized complex FFT routine for DSP56001/2 is also listed in ***Appendix A*** (CFFT56.ASM). 0.704ms is needed to calculate a 512-point complex FFT at 40 MHz clock, which is 8.7% faster than an optimized complex FFT. For more benchmarks see ***SECTION 8***. Note, however, that “straight-line” code always results in longer programs.

5.2 Example of Optimization

5.2.1 Fully Optimized Complex FFT for the DSP56001/2

Program CFFT56.ASM in ***Appendix A*** is a good example of optimizing complex FFTs on the DSP56001/2 for fast execution time. Figure 5-1 shows passes, groups and butterflies for a 512-point complex FFT. There is a total of 9 passes. The number of groups in each pass doubles from pass to pass, while the number of butterflies in each group halves from pass to pass. Each pass has the same number of butterflies,.i.e. N/2=256 butterflies.

CFFT56.ASM takes advantage of the trivial twiddle factors in all the passes. Note that pass 0 and 1 can be done by simple radix-4 butterflies. A radix-4 butterfly has been coded by 17 instructions, which is the best case on the DSP56001/2. The parallel data move in this radix-4 butterfly has been deliberately arranged to avoid a dual data move involving external memory, although the first and next to last instruction may result in cycle stretch in some cases. Since half of the 512 data are in external memory, one instruction cycle is stretched, and 18 instruction cycles are used for a 512-point complex FFT. This equals 4.5 instruction cycles per radix-2 butterfly. The same radix-4 butterflies are also applied to passes 2, 3, 4, and 5. Note that in Figure 5-1, the groups highlighted by cross lines are trivial butterflies too, and are not covered by the simple radix-4 butterflies. These data points are calculated by 5-instruction radix-2 butterflies. As shown in Figure 5-1,

each pass has 256 radix-2 butterflies and the first seven passes have 860 trivial butterflies. 772 of these radix-2 butterflies require 4.5 instruction cycles (simple radix-4 butterflies) while 88 of them require 5 instruction cycles. Therefore, the total cycle time for trivial butterflies is 772 x 4.5 + 88 x 5 = 3,914 which means a savings of 860 x 6 - 3914 = 1,246 cycles when compared to a non-optimization case. For program simplicity, the above calculation does not utilize the trivial butterflies in passes 7 and 8.

CFFT56.ASM uses N/2 real twiddle factors. This scheme reduces the data memory requirement and also reduces the structure overhead on group DO loops, because the group number in each pass changes to half of the previous scheme.

CFFT56.ASM fully utilizes internal memory to avoid cycle stretch when the DSP56001/2 accesses two data. A 512-point complex FFT is divided into two 256-point parts. The first 256-point part remains in internal memory until the last pass. The second 256-point data loads into internal memory after the first pass and stays there until the last pass.

The last two passes are implemented by two separate single loops to avoid the penalty of DO loop set-up. Each group has four radix-2 butterflies in the next-to-last pass, and two in the last pass. If group DO loop is still used, then each butterfly may take 6.75 and 7.5 cycles in the next-to-last pass and the last pass, respectively. The cycles saved from the separated DO loops are 256 x 6.75 + 256 x 7.5 - 512 x 6 = 576.

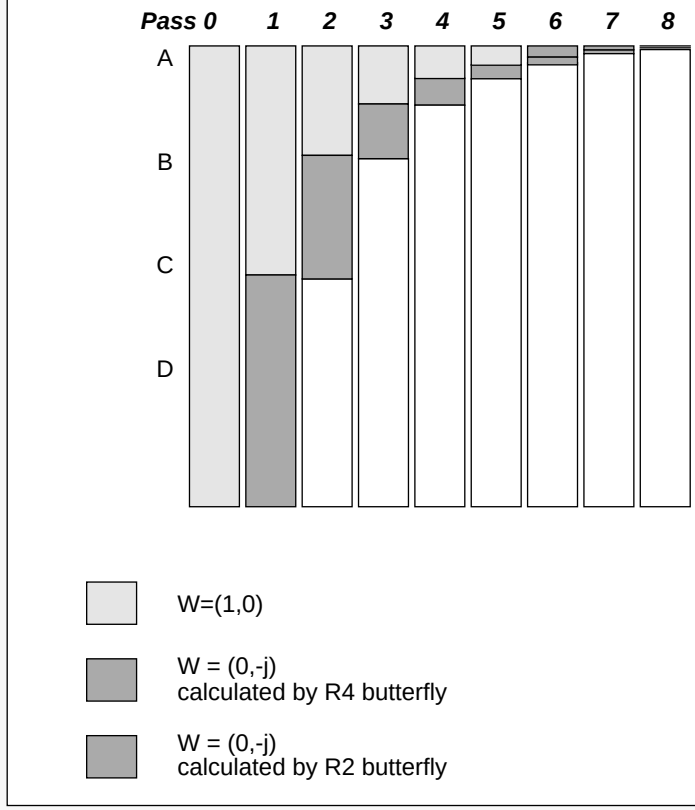

Figure 5-1 Trivial twiddle factors in a 12-point complex radix-2 DIT FFT. The butterflies in highlighted groups can be calculated without multiplications. A, B, C, and D are radix-4 butterfly pointers.

5.2.2 Fully Optimized Complex FFT for the DSP96002

APPENDIX A presents a fully optimized program for 1024-point complex input FFT for the DSP96002. Like the fully optimized program for the DSP56001/2, this program takes advantage of trivial twiddle factors in all of the passes as follows:

- Naturally, the first and second passes are combined into a radix-4 pass with each radix-4 butterfly requiring 8 instruction cycles. This is equal to 2 instruction cycles per radix-2 butterfly.

- All trivial butterflies in the middle passes are calculated by a separate routine.

- Each pass is written in a separate section to reduce the DO loop overhead. To reduce the program length, the special radix-4 and normal radix-2 butterflies are programmed in subroutines. Only two-nested DO loops are used for each pass.

- The last two passes are also combined into a radix-4 pass. After the combination, the number of instruction cycles per radix-2 butterfly is decreased from 5 to 4.25 instruction cycles. Because radix-4 butterflies are used in the last two passes, an extra set of 256 complex twiddle factors must be present in the external memory. These twiddle factors are generated off-line by MATHLAB software.

The fully optimized 1024-point complex FFT uses 18891 instruction cycles; while the optimized 1024-point complex FFT program (seen on the Motorola DSP bulletin board; Dr. BuB) uses 20958 instruction cycles. Optimization saves 20,958-18,891=2,067 instruction cycles which equals about 10% cycle time of the optimized code. Also note that the fully optimized code only works with fixed data length. ■