ECE503: The FFT & Number Representation
Lecture 10

D. Richard Brown III

WPI

02-Apr-2012

Lecture 10 Topics

1. A brief introduction to asymptotic complexity
2. Complexity analysis of the DFT
3. The Cooley-Tukey radix-2 decimation in-time fast Fourier transform (R2DIT-FFT) (Textbook 11.3)
4. Number representation (Textbook 11.8)
5. Fixed-point basics

Asymptotic Complexity of Algorithms

Suppose you have an algorithm $\mathcal{A}$ that operates on discrete-time sequences of length N and requires $g(N)$ "operations" to complete. Operations could be things like:

- Multiplications
- Multiply+accumulates (MACs)
- Comparisons
- Memory accesses
- etc.

We say algorithm $\mathcal{A}$ has asymptotic complexity of $\mathcal{O}(f(N))$ if

$$\boxed{\lim_{N\to\infty} \frac{g(N)}{f(N)} = c > 0}$$

where c is a positive constant (not a function of N).

Some Asymptotic Complexity Examples

Suppose you have an algorithm $\mathcal{A}$ to sort a sequence of N numbers and you determine it requires $g(N) = 6N + 12$ comparisons to complete. What is the asymptotic complexity?

What if you improved the algorithm so that it requires $g(N) = N/2 + 5$ comparisons?

Some real world examples:

- Finding an item in a sorted array with a binary search: $\mathcal{O}(\log N)$.
- Matrix inversion via Gaussian elimination: $\mathcal{O}(N^3)$.
- Solving the traveling salesman problem via dynamic programming: $\mathcal{O}(a^N)$ with $a > 1$.

Asymptotic Complexity of the DFT

To determine the asymptotic complexity of the DFT, recall

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \text{ for } k = 0, 1, \dots, N-1.$$

The most relevant measure of “operations” here is multiplies or multiply+accumulates (MACs). In our analysis, let’s assume the W_N^{kn} are all computed in advance (and, hence, are not included in the complexity analysis). Your textbook discusses how these can be computed on the fly via Goertzel’s Algorithm.

How many (complex) multiplies to compute the DFT?

How many (complex) additions to compute the DFT?

If we define “operations” as multiplies (or additions, or MACs), what is the asymptotic complexity of the DFT?

The Cooley-Tukey Radix-2 Decimation-in-Time FFT

Main idea: Suppose you are given a length-N sequence $\{x[n]\}$ (with N even) and wish to compute the DFT $\{X[k]\}$. What if we did this?

1. Split $x[n]$ into two length-$\frac{N}{2}$ sequences
$$\{x[n]\} = \{x[0], x[2], \dots, x[N-2]\} + \{x[1], x[3], \dots, x[N-1]\}.$$
2. Reindex and call the first sequence $\{x_{even}[n]\}$ for $n = 0, \dots, N/2 - 1$.
3. Reindex and call the second sequence $\{x_{odd}[n]\}$ for $n = 0, \dots, N/2 - 1$.
4. Compute the length-$\frac{N}{2}$ DFTs: $\{X_{even}[k]\} = \mathrm{DFT}\{x_{even}[n]\}$ and $\{X_{odd}[k]\} = \mathrm{DFT}\{x_{odd}[n]\}$. Each of these DFTs has $\approx \frac{N^2}{4}$ operations.
5. Assemble the length-$\frac{N}{2}$ DFTs into the desired length-N DFT $\{X[k]\}$ via
$$X[k] = \begin{cases} X_{even}[k] + W_N^k X_{odd}[k] & k = 0, \dots, \frac{N}{2} - 1 \\ X_{even}[k - \frac{N}{2}] + W_N^k X_{odd}[k - \frac{N}{2}] & k = \frac{N}{2}, \dots, N-1 \end{cases}$$

How many total operations? $\approx \frac{N^2}{4} + \frac{N^2}{4} + N$. Still $\mathcal{O}(N^2)$. But what if we repeated this “divide and conquer” approach all the way to one-point DFTs?

FFT Complexity Analysis: $N = 1$

When $N = 1$, there is nothing to divide into even and odd parts, so we will just use the DFT. The DFT equation:

$$X[k] \;=\; \sum_{n=0}^{N-1} x[n] W_N^{kn} \qquad \text{for } k = 0, 1, \dots, N-1$$

When $N = 1$, we have

$$X[0] \;=\; x[0]e^0 = x[0]$$

No multiplies or additions. Hence a one-point DFT/FFT has no MACs.

FFT Complexity Analysis: $N = 2$

Two-point FFT computed using radix-2 decimation in time: $\{x[n]\} = \{x[0], x[1]\}$.

$$\begin{aligned} X[0] &= X_{even}[0] + W_2^0 X_{odd}[0] = X_{even}[0] + X_{odd}[0] \\ X[1] &= X_{even}[0] + W_2^1 X_{odd}[0] = X_{even}[0] - X_{odd}[0] \end{aligned}$$

Note that $X_{even}[k]$ and $X_{odd}[k]$ for $k = 0, 1$ are just one-point FFTs.

$$\begin{aligned} X_{even}[0] &= \text{DFT}_1\{x[0]\} = x[0] \\ X_{odd}[0] &= \text{DFT}_1\{x[1]\} = x[1] \end{aligned}$$

Hence,

$$\begin{aligned} X[0] &= x[0] + x[1] \\ X[1] &= x[0] - x[1] \end{aligned}$$

How many MACs in the R2-DIT two-point FFT?

Note that the two-point DFT and FFT are identical.

FFT Complexity Analysis: $N = 4$

Four-point FFT computed using radix-2 decimation in time:

$$\begin{aligned}
X[0] &= X_{even}[0] + W_4^0 X_{odd}[0] = X_{even}[0] + X_{odd}[0] \\
X[1] &= X_{even}[1] + W_4^1 X_{odd}[1] = X_{even}[1] - jX_{odd}[1] \\
X[2] &= X_{even}[0] + W_4^2 X_{odd}[0] = X_{even}[0] - X_{odd}[0] \\
X[3] &= X_{even}[1] + W_4^3 X_{odd}[1] = X_{even}[1] + jX_{odd}[1]
\end{aligned}$$

Computation of $\{X_{even}[0], X_{even}[1]\} = \mathrm{FFT}_2\{x[0], x[2]\}$ requires how many MACs? 2

Computation of $\{X_{odd}[0], X_{odd}[1]\} = \mathrm{FFT}_2\{x[1], x[3]\}$ requires how many MACs? 2

How many more MACs are required to assemble the results into a four-point FFT? 4

Hence, the total MACs needed to compute a four-point FFT is 8.

FFT Complexity Analysis: $N = 8$

Eight-point FFT computed using radix-2 decimation in time follows the same accounting:

$$X[k] \quad = \quad X_{even}[k] + W_8^k X_{odd}[k] \qquad \text{for } k = 0, 1, \dots, 7$$

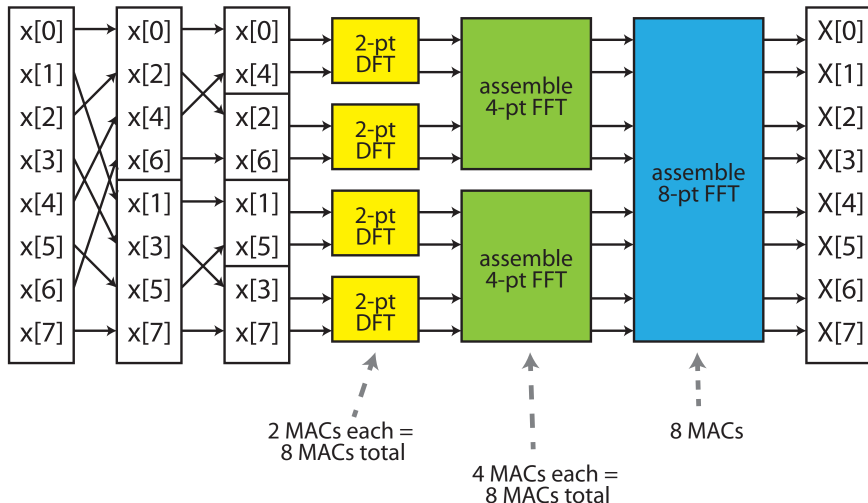
Computation of $\{X_{even}[0], X_{even}[1], X_{even}[2], X_{even}[3]\} = \mathsf{FFT}_4\{x[0], x[2], x[4], x[6]\}$ requires how many MACs? 8

Computation of $\{X_{odd}[0], X_{odd}[1], X_{odd}[2], X_{odd}[3]\} = \mathsf{FFT}_4\{x[1], x[3], x[5], x[7]\}$ requires how many MACs? 8

How many more MACs are required to assemble the results into an eight-point FFT? 8

Hence, the total MACs needed to compute an eight-point FFT is 24.

FFT Complexity Analysis: $N = 8$

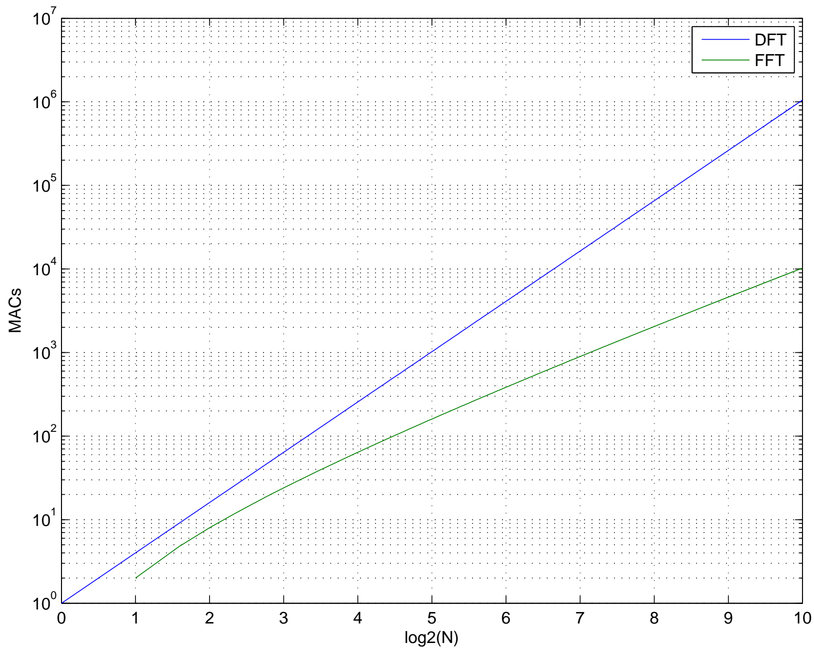


See also Figure 11.21 in your textbook.

FFT Complexity Analysis: General N

Operation	MACs
Computation of N one-point FFTs	0
Assembling $\frac{N}{2}$ two-point FFTs from N one-point FFTs	$\frac{N}{2} \cdot 2 = N$
Assembling $\frac{N}{4}$ four-point FFTs from $\frac{N}{2}$ two-point FFTs	$\frac{N}{4} \cdot 4 = N$
Assembling $\frac{N}{8}$ eight-point FFTs from $\frac{N}{4}$ four-point FFTs	$\frac{N}{8} \cdot 8 = N$
$\vdots$	$\vdots$
Assembling one N-point FFT from two $\frac{N}{2}$-point FFTs	$1 \cdot N = N$

Hence, the total MACs needed to compute an N-point FFT is ______.



DFT
FFT
MACs
log2(N)
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
0
1
2
3
4
5
6
7
8
9
10

Floating-Point vs. Fixed-Point

Computing the output of an IIR filter (direct form I example):

$$y[n] = b_0 x[n] + b_1 x[n-1] - a_1 y[n-1].$$

When implementing DSP algorithms like filters, there is a fundamental choice between using **floating-point** or **fixed-point** math.

Floating-Point Advantages	Fixed-Point Advantages
More accurate	Less expensive
Easier to code	Typically faster
	Typically lower power

Floating-point datatypes in C include `float` (32 bit IEEE) and `double` (64-bit IEEE). Matlab defaults to double-precision floating-point math.

You can buy DSPs that have built-in hardware support for floating-point math, but these have limited application because they are expensive, slow, and tend to be less power efficient than fixed-point DSPs.

Fixed-Point Numbers (part 1)

First, recall M-bit binary unsigned integer representation:

$$q = \{\mathrm{MSB}, \dots, \mathrm{LSB}\} = \{b_{M-1}, b_{M-2}, \dots, b_0\} \overset{\text{e.g.}}{=} 10110$$

The base-10 value of q is easily computed as

$$q = \sum_{m=0}^{M-1} b_m 2^m \overset{\text{e.g.}}{=} 0 + 2 + 4 + 0 + 16 = 22$$

Fixed point numbers are just a generalization of this idea where we add a **binary decimal point** denoted by Δ. For example:

$$q = 10_{\Delta}110$$

In this example, there are $b = 3$ "fractional bits". We can compute q as

$$q = \sum_{m=0}^{M-1} b_m 2^{m-b} \overset{\text{e.g.}}{=} 0 + \frac{1}{4} + \frac{1}{2} + 0 + 2 = 2.75 = \frac{22}{8} = \frac{\text{integer value}}{2^b}$$

Fixed-Point Numbers (part 2)

Negative fixed-point numbers are just like negative integers. Typically these are stored in two's complement representation.

Example: Write an 8-bit binary two's complement representation of -1.375 given a fixed-point representation with four fractional bits.

Step 1: Determine the 8-bit binary representation of $+1.375$: $q = 0001_{\triangle}0110$.

Step 2: Invert the bits: $q = 1110_{\triangle}1001$.

Step 3: Add 1: $q = 1110_{\triangle}1010$.

Step 4: Check (in this case, b_{M-1} is called the "sign bit"):

$$
\begin{aligned}
q &= -b_{M-1}2^{M-b} + \sum_{m=0}^{M-2} b_m 2^{m-b} \\
&= -8 + 0 + \frac{1}{8} + 0 + \frac{1}{2} + 0 + 2 + 4 = -1.375
\end{aligned}
$$

Fixed-Point Addition

Just like binary integer addition except

- you must make sure the decimal points are aligned and
- you have to check for overflow.

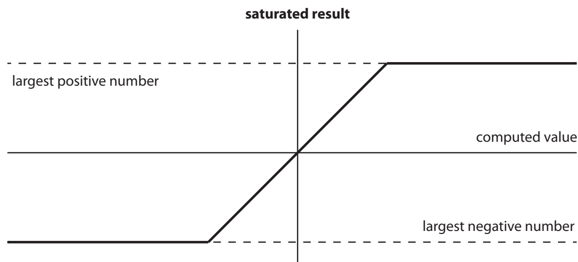
Example 1: $q1 = 0011_\Delta 0110$ and $q2 = 0001_\Delta 0011$, compute $q1 + q2$. The decimal points are aligned, so we can add these to get $q1 + q2 = 0100_\Delta 1001$. No overflow. You can check this result in base-10.

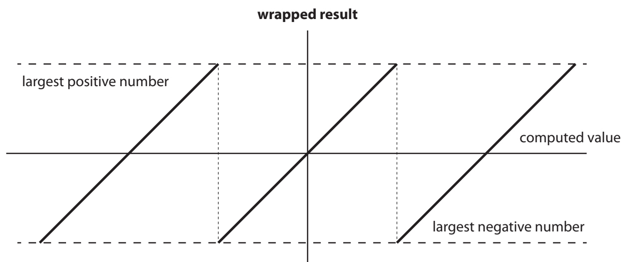
Example 2: $q1 = 0011_\Delta 0110$ and $q2 = 000_\Delta 10011$, compute $q1 + q2$. The decimal points are not aligned here, so we either need to convert $q1$ to a representation with 5 fractional bits or convert $q2$ to a representation with 4 fractional bits before we can add.

- First option: $q1 = 011_\Delta 01100$. Then $q1 + q2 = 011_\Delta 11111$.
- Second option: $q2 = 0000_\Delta 1001$ (note we've lost some precision here). Then $q1 + q2 = 0011_\Delta 1111$.

Neither option overflowed. Option 1 is more accurate.

Types of Overflow





Overflow Examples & Remarks

In these examples, we assume $q1$ and $q2$ are signed fixed-point numbers.

Example 1: $q1 = 0111_{\Delta}0110$ and $q2 = 0001_{\Delta}0011$, compute $q1 + q2$ assuming saturation overflow. The decimal points are aligned, so we can add these to get $q1 + q2 = 1000_{\Delta}1001$. Note the sign bit is one, which shouldn't happen when we add two positive numbers. Hence we overflowed. The saturated result is $q1 + q2 = 0111_{\Delta}1111$.

Example 2: $q1 = 0111_{\Delta}0110$ and $q2 = 0001_{\Delta}0011$, compute $q1 + q2$ assuming wrapped overflow. The decimal points are aligned, so we can add these to get $q1 + q2 = 1000_{\Delta}1001$. This is the wrapped result.

Remarks:

- Your textbook calls wrapped overflow "two's complement overflow".
- Generally, wrapped overflow is worse than saturated overflow, but sometimes it works out when you are adding a sequence of numbers that the final result is correct with wrapped overflow (see homework).
- Overflow is usually devastating to filtering.

Quantization Error

We define the quantization error as the difference between the number we are trying to represent and the number we actually get in our fixed-point/quantized representation:

$$\epsilon = \mathcal{Q}(x) - x$$

where $\mathcal{Q}(x)$ is the fixed-point/quantized value and x is the unquantized value.

Example: $x = \pi$ and we quantize x to a signed 8-bit fixed-point datatype with 3 fractional bits. We can compute the closest quantized value in this case falls at $\mathcal{Q}(\pi) = 3.125$, which can be written as $q = 00011_{\Delta}001$ in binary. The quantization error is then $\epsilon = \mathcal{Q}(\pi) - \pi \approx -0.0166$.

Quantization errors tend to be much bigger if we have overflow.

Example: $x = \pi$ and we quantize x to a signed 8-bit fixed-point datatype with 6 fractional bits. We can compute the closest quantized value in this case falls at $\mathcal{Q}(\pi) = 3.140625$, but this value is above the largest positive value of our datatype. With saturation overflow, we would have $\mathcal{Q}(\pi) = 1.984375$ and the quantization error is then $\epsilon = \mathcal{Q}(\pi) - \pi \approx -1.1572$.

Conclusions

1. Asymptotic complexity basics
2. The DFT and the FFT (Section 11.3, skimming Goertzel's algorithm)
   - The FFT and the DFT give the same results.
   - The only difference is that the FFT is much faster than the DFT for large N. The FFT has asymptotic complexity $\mathcal{O}(N \log_2(N))$ whereas the DFT has asymptotic complexity $\mathcal{O}(N^2)$.
   - The FFT forms the basis for lots of fast algorithms like "fast convolution".
   - The R2DIT Cooley-Tukey FFT is just one approach.
3. Finite-precision signal processing (Section 11.8, skimming floating point representation)
   - Fixed-point number representation
   - Fixed-point addition
   - Overflow
   - Quantization errors