FFT 电路设计规范

1. FFT 简介：

傅里叶变换是信号分析的基本方法之一，它能够将信号从时间域转化到频率域，进而研究信号的频谱结构和变换规律。傅里叶变换从形式上可分为离散傅里叶变换（Discrete Fourier Transform，DFT）和连续傅里叶变换（Continuous Fourier Transform，CFT）。

快速傅里叶变换 (Fast Fourier Transform，FFT) 是一种高效计算离散傅里叶变换的方法。采用这种算法能使得计算离散傅里叶变换所需要的计算量显著减少。

FFT 算法应用于音频处理、图像处理等多个领域中，这些领域往往要求芯片具备对信号进行实时处理的能力，因此设计出一个计算效率更高的高能效 FFT 电路结构就变得至关重要。

2. 课程设计目标：

面向高能效计算需求，设计兼顾性能、面积和功耗开销于一身的 FFT 电路，完成电路的架构设计、Verilog HDL 代码设计、逻辑仿真、性能分析、逻辑综合、时序分析与验证和物理设计，进行结果分析比较。

3. 课程设计指标及应用要求：

（1） FFT 电路的基本要求：16 点 FFT，每个输入数值均为复数，分为实部和虚部，实部和虚部的长度均为 17bit，最高位（第 16 位）为符号位，第 0 位至第 7 位为小数位，第 8 位至第 15 位为整数位；

（2） 可采用任意 FFT 算法，但保证 FFT 功能正确；

（3） 芯片设计工艺：0.18um 工艺；

（4） FFT 电路的评价指标：完成一次 16 点 FFT 的能耗、单位面积单位功耗时的 FFT 操作数、芯片的工作频率和面积开销；

4、FFT 算法介绍

4.1 DFT 的基本概念

DFT 算法将输入的时域序列 x(n)转化为频域序列 X(k)输出,数学表达式如下：

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{nk}\quad（k=0,1,2,\cdots\cdots,N-1）,\ W_N^p = e^{-j\frac{2\pi}{N}p}$$

从公式中可以看出每计算一个 X(k)需要 N 次复数乘法和 N-1 次复数加法，

因此计算所有的 X(k)需要 N^2 次复数乘法和 N(N-1)次复数加法，DFT 的计算复杂度为 $O(N^2)$。

4.2 基-2 FFT 算法

FFT 算法能够大大降低 DFT 算法的计算量，常用的基-2 FFT 算法需要输入序列 x(n)的长度 N 为 2 的整数次幂即 $N=2^M$（M 为正整数）。若不满足可将序列补零延长，使其满足长度要求。输入序列长度为 N 的 FFT 运算称为 N 点 FFT 运算。

基-2 FFT 算法可分为按时域抽取的 FFT 算法（Decimation in Time, DIT）和按频域抽取的 FFT 算法（Decimation in Frequency, DIF）。

DIT 算法是按照 n 的奇偶性将原序列分成 2 个长度为 N/2 的序列，即:

$$x_1(r) = x(2r)$$
$$x_2(r) = x(2r+1) \quad （r=0,1,\cdots\cdots,N/2-1）$$

则$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} = \sum_{r=0}^{N/2-1} x_1(r)W_N^{2kr} + \sum_{r=0}^{N/2-1} x_2(r)W_N^{2kr}\,W_N^k$

因为$W_N^{2kr} = W_{N/2}^{kr}$，所以可得：

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x_1(r)W_{\frac{N}{2}}^{kr} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x_2(r)W_{\frac{N}{2}}^{kr}$$

$$= X_1(k) + W_N^k X_2(k), (k = 0,1,\dots,\frac{N}{2}-1)$$

$$X\left(k+\frac{N}{2}\right) = X_1(k) - W_N^k X_2(k), (k = 0,1,\dots,\frac{N}{2}-1)$$

上述公式可表示为下图所示的蝶形运算：

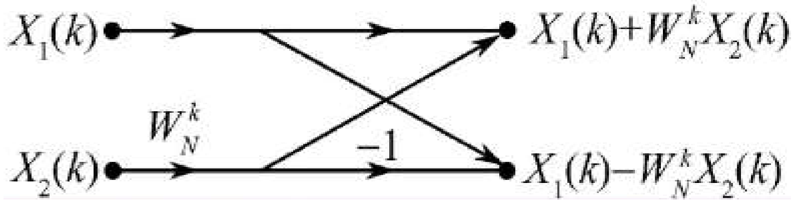

图 4.1 蝶形运算图

对于 N 点的 DIT-FFT 运算来说，共有 $\log_2 N$ 级，每级都有 N/2 个蝶形运算，每次蝶形运算有一次复数乘法，两次复数加法。因此复数乘法的运算总次数为 $\frac{N}{2}\log_2 N$，复数加法的运算总次数为 $N\log_2 N$,总的运算复杂度为 O(NlogN)，相比 DFT 的 $O(N^2)$的运算复杂度而言运算量大大降低。

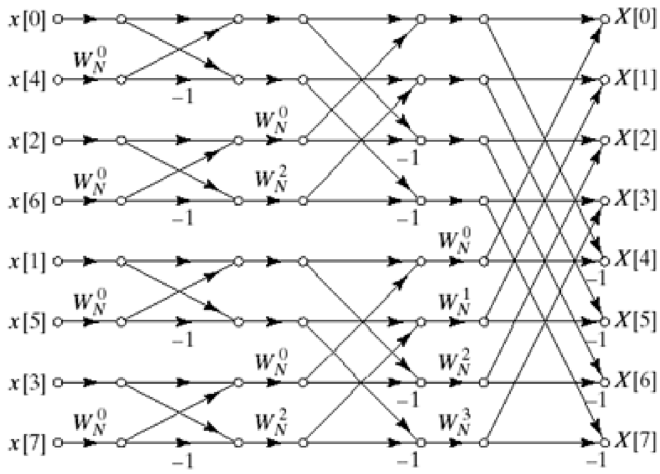

图 4.2　8 点 DIT-FFT 算法运算流图

而不同于 DIT 算法，另一种 DIF 算法是将输入数据对半分为两组进行如下的推导运算：

$$
\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{nk} \\
&= \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=N/2}^{N-1} x(n)W_N^{nk} \\
&= \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=0}^{N/2-1} x(n+N/2)W_N^{(n+N/2)k} \\
&= \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)W_N^{Nk/2}]W_N^{nk} \\
&= \sum_{n=0}^{N/2-1} [x(n) + (-1)^k x(n+N/2)]W_N^{nk}
\end{aligned}
$$

比较 8 点 DIF-FFT 算法和 8 点 DIT-FFT 算法的运算流图，不难发现两者的运算复杂度均为 O(NlogN)，只在输入数据和输出数据的顺序编排上有所差别。如果要得到输入序列长度更长的 FFT 运算流图，可通过公式推导进一步得到。这里给出了 16 点的 DIF-FFT 算法运算流图。

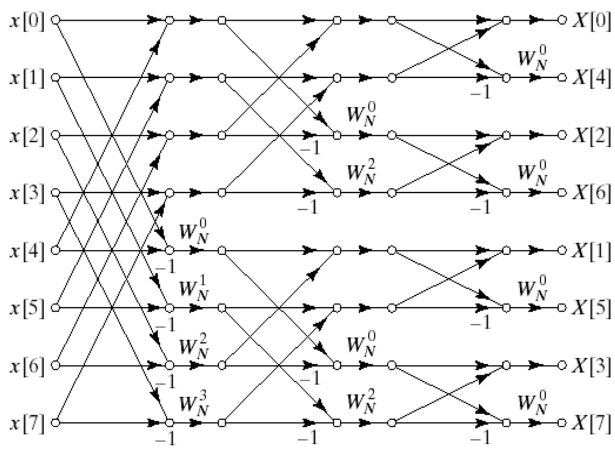

图 4.3　8 点 DIF-FFT 算法运算流图

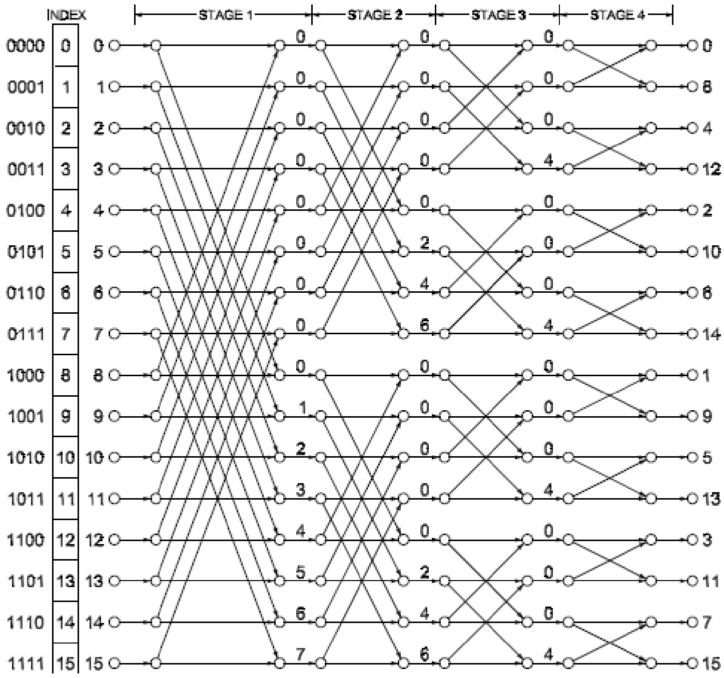

图 4.4　16 点 DIF-FFT 算法运算流图

5、参考文献

[1] Garrido M , Huang S J , Chen S G . Feedforward FFT Hardware Architectures Based on Rotator Allocation[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2018, 65(2):581-592.