



# 《数字信号处理 VLSI 设计》课程设计

设计题目： PJ-03 64点FFT处理器设计

成 员： 刘迅 学 号： 19212020063

成 员： 梁策 学 号： 19212020090

成 员： 施淳信 学 号： 19212020081

成 员： 苟傲睿 学 号： 19110860008

院 系： 微电子学院

指导老师： 曾晓洋

完成日期： 2020年 01月12日

## 设计要求与目标

### 设计目标

设计一个64点的FFT处理器

### 基本要求

1) 使用按频率抽选(DIF)的库利－图基(Cooley－Tukey)算法。的Cooley－Tukey快速算法，五步骤

A. 将改写成,利用



B. 做r2个r1点的DFT，得到。



C. 把N个乘以相应的旋转因子，组成。



D. 做r1个r2点DFT，得到。



E. 进行整序，得到，其中。



可任选基2，基4，基8算法实现：

64＝2×2×2×2×2×2

64＝4×4×4

64＝8×8

……

2) 数据吞吐率(throughput)为8MSample/s。其中，一个Sample表示一个数据点，即实部虚部各10bit。

3) 最高时钟工作频率需要达到30MHz。

4) 输入输出端口说明：

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **输入/输出** | **位宽(bit)** | **描述** |
| clk | 输入 | 1 | 系统时钟 |
| rst\_n | 输入 | 1 | 异步复位信号，低电平有效 |
| din\_re | 输入 | 10 | 实部输入（二进制补码） |
| din\_im | 输入 | 10 | 虚部输入（二进制补码） |
| din\_valid | 输入 | 1 | 输入有效信号，高有效 |
| dout\_re | 输出 | 10 | 实部输出（二进制补码） |
| dout\_im | 输出 | 10 | 虚部输出（二进制补码） |
| dout\_valid | 输出 | 1 | 输出有效信号，高有效 |

### 设计内容

1. 用Verilog语言编写64点的FFT处理器。
2. 要求进行仿真，给出仿真结果。
3. 需要综合到门级（可以采用FPGA流程或ASIC流程）。

### 递交结果

1. 设计报告，报告内容包含：

A. 设计要求

B. 设计方法和流程和设计用EDA软件

C. 验证方法和结果分析

D. 总结

E. 附录：要求以表格形式说明文件名，文件内容说明，文件路径

1. 源程序以及testbench
2. 综合后的门级网表

## 电路设计与算法

### FFT算法

快速傅里叶变换（Fast Fourier Transform, FFT）是一种快速有效地计算离散傅里叶变换（Discrete Fourier Transform, DFT）的方法。

对于DFT运算，有：

，，

这一项完成了时域n到频域k的变换。

尽管，DFT是信号分析与处理中的一种重要变换，但直接计算DFT的计算量与变换区间长度N的平方成正比，其复杂度为当N较大时，计算量太大，直接用DFT算法进行谱分析和信号的实时处理是不切实际的。

在上世纪60年代中期，Cooley和Tukey提出了一种离散傅里叶变换的快速算法，它减少了复数乘法的运算量，其复杂度变为。将N点的DFT变换拆解成r1\*r2\*r3\*…rn更低维的简单运算，体现了降维的思想，由基本FFT操作单元的不同可分为基2，基4，基8等。

FFT算法实质上就是DFT的重排：以8点FFT（DIT）为例，其FFT架构如图

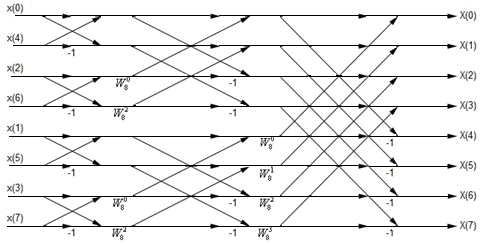


图2.1 8点FFT（DIT）

通过对输入数据的奇偶分治，最大限度减少了DFT中的数据依赖。

### 64点FFT基2电路设计

基于FFT原理，在电路设计中要实现的就是重排和2点的蝶形运算。

由于系统的输入输出限制在10bit，即1个数据点，而FFT是存在不同时刻输入数据依赖的。所以在进行实际的FFT运算前，必定需要经过一个串并/并串转换模块。这也就说明，在进行两次64点FFT之间有64clk的时间余量，所以选择迭代次数最多的基2FFT。

通过观察图2.1，其中的每一级FFT都是4个2点FFT，只不过每一级数据都进行了重排。基于此设计64点FFT的状态机如图所示：需要32个2点FFT单元

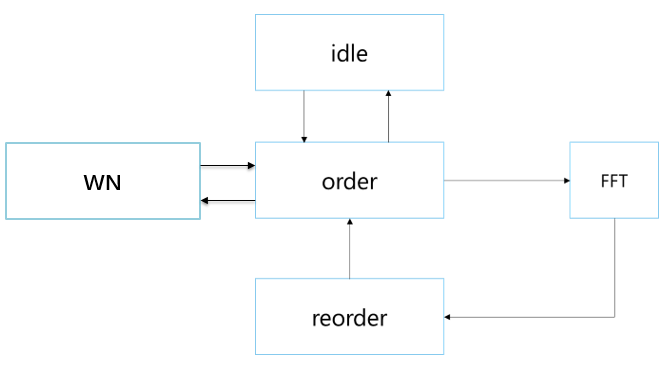


图2.2 64点FFT状态机

以输入重排后的数据为基准0~63，第一级输出后4个为一组重排，再输入回堆叠的32个2点FFT模块，得到输出再映射回到初始的0~63；再在第二级以8个为一组进行重排，依此类推。实际上reorder这个步骤是可以省略的，直接由第一级的重排结果重排得到第二级所需的重排顺序，但这会随着层数的加深使得排序越来越复杂。综合考虑下来，为了逻辑的清晰，选择保留reorder模块。在ord处层数++，并获取下一层的旋转因子。

顶层模块化设计如图所示：

data\_out

data\_in

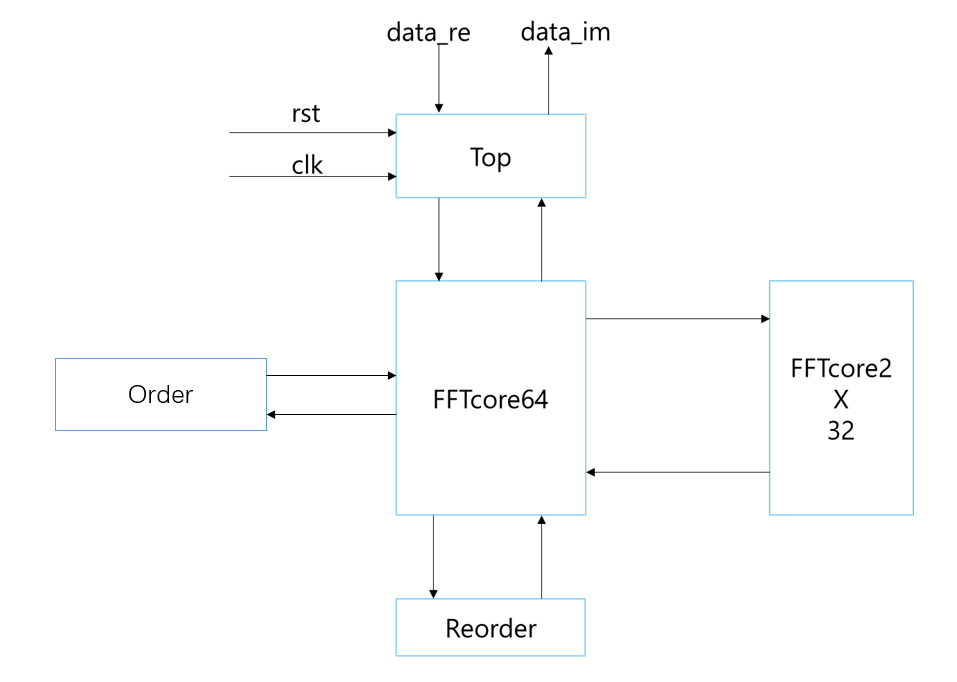


图2.3 顶层模块图

顶层调用core64模块，在core64模块中构建状态机，并且调用core2x32/order/reord模块。

### FFT core2

运算中统一位10bit，并且旋转因子必定存在，故选择将输入和旋转因子均左移8bit，允许的输入为(-2, 2)。在进行FFT过程中不进行数据的扩位，因此在乘法结束后需要截尾，去掉多余的8bit精度。本次PJ中尝试了两种截尾方式：截尾和四舍五入。

### FFT order/reord

1. 输入级：奇偶分治

有两种排序逻辑，① 输入反序，如表所示，以8点FFT为例；② 矩阵操作，如图所示，以16点FFT为例。

|  |  |  |
| --- | --- | --- |
| Input index | Reverse index | Output index |
| 0/000 | 000 | 0 |
| 1/001 | 100 | 4 |
| 2/010 | 010 | 2 |
| 3/011 | 110 | 6 |
| 4/100 | 001 | 1 |
| 5/101 | 101 | 5 |
| 6/110 | 011 | 3 |
| 7/111 | 111 | 7 |

表2.1 输入反序重排

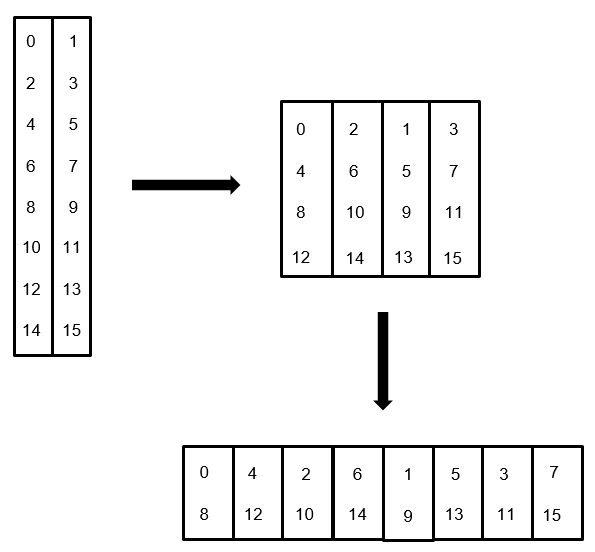


图2.4 矩阵重排（奇偶分治）

1. 迭代中的各级：

以输入级整序后为基准，8点FFT为例，参考图2.1：① 第一级，02/13/46/57，前4个为一组，间隔为2；② 第二级04/15/26/37，8点为一组，间隔为4。对于16、32、64点FFT可依此类推。

## 功能验证

### 16点FFT

输入为线性数据，是过零点的奇函数，直流分量为零；core2截尾。可以看到理论输出和RTL输出有较好的一致性。

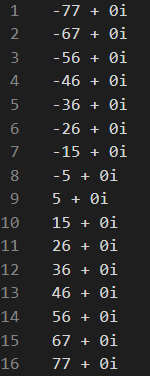
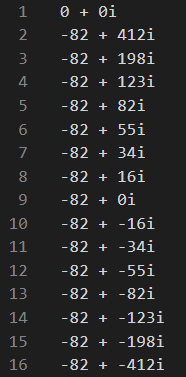
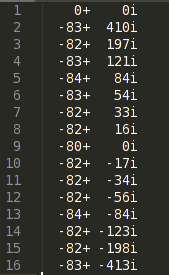
  

图3.1 16点FFT输入/理论输出/RTL输出

### 32点FFT

输入为线性数据；core2截尾。可以看到理论输出和RTL输出有较好的一致性。

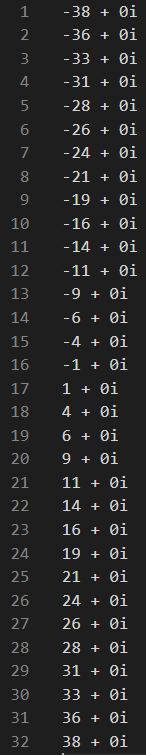
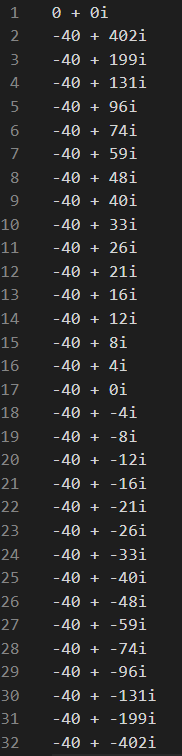
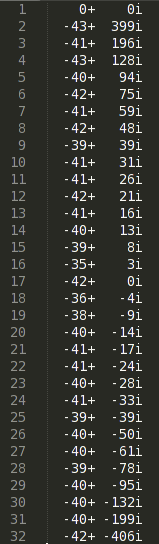
  

图3.2 32点FFT输入/理论输出/RTL输出

### 64点FFT

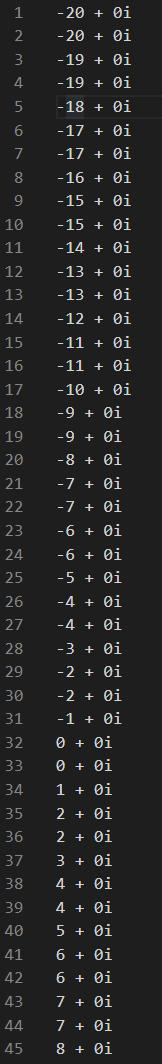
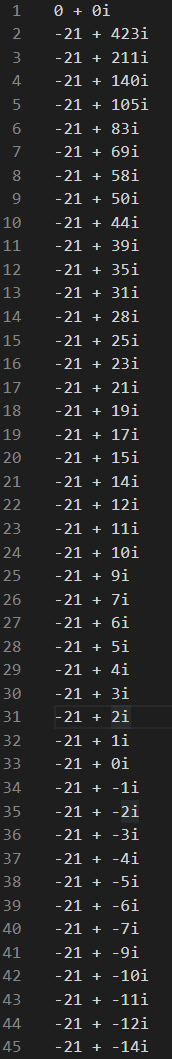
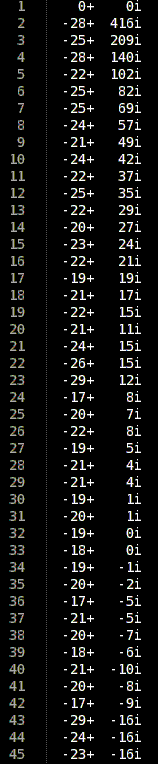
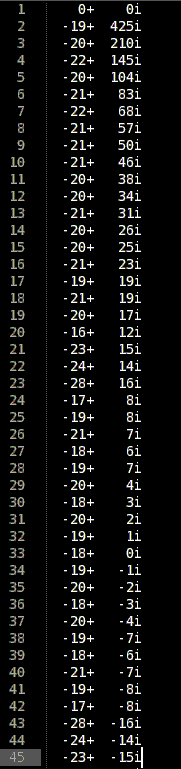
此处讨论截尾方式造成的影响：输入为线性数据，core2分别采用截尾/四舍五入。   

图3.3 64点FFT输入/理论输出/RTL输出(截尾)/RTL输出(四舍五入)

可以看到理论输出和RTL输出有较好的一致性。随着迭代深度的增加，每次迭代造成的误差会逐渐累积，但总体上四舍五入的截尾方式比直接截尾造成的误差要小。

### 输入输出波形

## DC综合