

16 点复数 FFT 算法函数说明书

文档类型	技术说明
编制	杜伟韬
版本	1
创建日期	06 年 1 月 4 日

注：本文作为示范文档用于数字化工程中心学生培训

C 接口函数及数据类型

函数 `cfft16_init`

描述

初始化 16 点 FFT 运算函数的实例 (instance)

函数原型

```
int cfft16_init(cfft16plan_t *cfft16planHandle);
```

参数

- `cfft16planHandle`

16 点 FFT 函数的句柄 (handle)，指向一个 16 点 FFT 函数使用的实例 (instance)，该实例用于保存 FFT 运算函数运行的状态信息和内部数据的数据结构

函数 `cfft16_proc`

描述

16 点 FFT 运算函数，支持同址 FFT (即 in 可以等于 out)，输入数据格式为复数格式

函数原型

```
int cfft16_proc(
    cfft16plan_t *cfft16planHandle,
    cfft_t *in,
    cfft_t *out
```

);

参数

- `cfft16planHandle`

经过初始化的 16 点 FFT 函数的句柄, 该句柄指向一个 16 点 FFT 函数使用的实例 (instance), 该实例用于保存 FFT 运算函数运行的状态信息和内部数据的数据结构

- `in`

输入的时域数据存储区域的首地址

- `out`

输入的时域数据存储区域的首地址

接口数据类型

- `cfft_t`

描述

复数格式数据类型, 定义于文件 `cfft16.h` 中

定义

```
typedef struct {  
    float r;  
    float i;  
} cfft_t;
```

其中, `r` 表示实部, `i` 表示虚部

函数 `cfft16_close`

描述

FFT 实例的析构函数

函数原型

```
int cfft16_close(cfft16plan_t *cfft16planHandle);
```

参数

- `cfft16planHandle`

经过初始化的 16 点 FFT 函数的句柄, 该句柄指向一个 16 点 FFT 函数使用的实例 (instance), 该实例用于保存 FFT 运算函数运行的状态信息和内部数据的数据结构

函数 ComplexPrint

描述

以 MATLAB 列向量定义的格式向标准终端 (stdout) 打印复数数组, 例如:

```
a = [  
    1+2j  
    2+3j  
]
```

函数原型

```
void ComplexPrint(  
    char *name,  
    cfft_t *data,  
    int N  
);
```

参数

- name

字符串指针, 指向打印输出中数组名称的字符串

- data

复数数组的起始地址

- N

复数数组中复数元素的个数

算法原理及存储方案设计

16 点 FFT 算法

采用行列分解的旋转因子法计算 16 点 DFT, 将 16 点 DFT 分解为 4×4 点 DFT
设

$$W_N = e^{-j2\pi/N}$$

$$\begin{aligned} i &= 4i_1 + i_2 & k &= k_1 + 4k_2 \\ i_1 &= 0, 1, 2, 3 & k_1 &= 0, 1, 2, 3 \\ i_2 &= 0, 1, 2, 3 & k_2 &= 0, 1, 2, 3 \end{aligned}$$

则 16 点 DFT 可以表示成为

$$V_{(k_1, k_2)} = \sum_{i_2=0}^3 W_4^{i_2 k_2} [W_{16}^{i_2 k_1} \sum_{i_1=0}^3 W_4^{i_1 k_1} v_{(i_1, i_2)}]$$

即，输入的 16 点数据序列首先**按照行**排成 4×4 的矩阵，然后逐列作 4 点 DFT，乘以旋转因子，再逐行作 4 点的 DFT，最后**按照列**输出。

16 点 FFT 的旋转因子

对于进行变换的 4×4 的矩阵，第 1 行的旋转因子是 1，第 2 行的旋转因子是 $W_{16}^0, W_{16}^1, W_{16}^2, W_{16}^3$ ，第 3 行的旋转因子是 $W_{16}^0, W_{16}^2, W_{16}^4, W_{16}^6$ ，第 4 行的旋转因子是 $W_{16}^0, W_{16}^3, W_{16}^6, W_{16}^9$ ，这些旋转因子要事先脱机计算并保存。

矩阵转置算法

行列分解的 FFT 需要映射在一维数组中的 2 维矩阵进行转置运算，因同址转置算法设计难度很大，程序中使用不支持同址转置的算法，这会带来额外的存储器开销。

4 点 FFT 算法

4 点 FFT 使用下列公式计算

$$\begin{aligned} V_0 &= v_0 + v_1 + v_2 + v_3 \\ V_1 &= v_0 - jv_1 - v_2 + jv_3 \\ V_2 &= v_0 - v_1 + v_2 - v_3 \\ V_3 &= v_0 + jv_1 - v_2 - jv_3 \end{aligned}$$

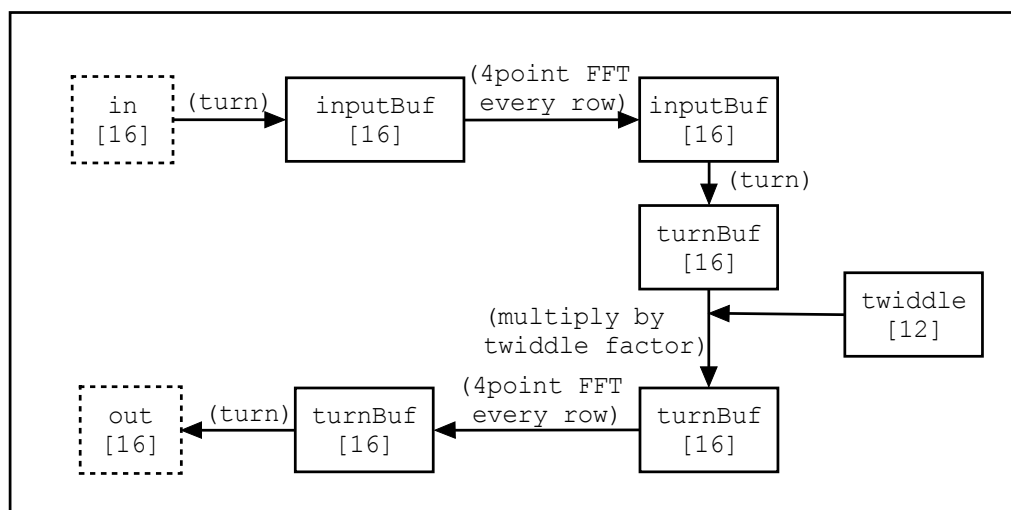
存储方案设计

使用一维数组映射二维矩阵

旋转因子 FFT 算法是使用二维 FFT 来处理输入的一维数据，这是一种概念上的方法，在实际编程实现时所有的数据序列都是存储于一维数组中的，然后通过数组下标运算来映射成为概念上的二维行列分解的 FFT，另外由于 FFT 函数只能处理地址连续的输入序列（即输入序列要保存在一个数组中），所以对于映射在一维数组中的二维矩阵中的所有列进行 FFT 运算时，必须先把该概念上的二维矩阵进行转置，然后在对相应的所有行进行 FFT，然后在把 FFT 的结果进行转置，最终等效为对概念上二维矩阵中的所有列进行了 FFT。

16 点 FFT 存储方案设计

下图中实线方框表示的存储区域需要在 16 点 FFT 运算函数的实例中实现，其中方框中的字母表示缓冲区的名称，[] 中的数字表示数组元素的个数，() 中的文字表示运算操作。由于 4×4 FFT 中，第一行的旋转因子均是 1，所以不必保存，只需要保存其余 3 行共 12 个旋转因子即可。



4 点 FFT 存储方案设计

