FFT 的 FPGA 实现

1.1 前言

FFT 一般有多种计算方式，常见的为基 2-DIT-FFT 与基 2-DIF-FFT。两种不同计算方式的最大区别是数据输入和输出的位序，在 DIT-FFT 中，输入序列为“码位倒读”顺序，输出序列为自然顺序，在 DIF-FFT 中序列顺序则相反。

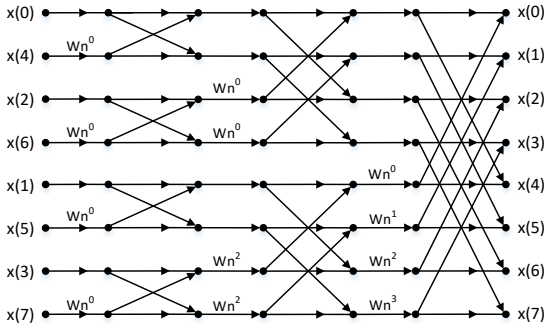

图 1.1 DIT-FFT 流程图

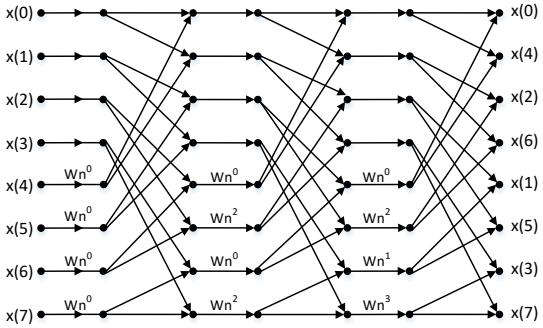

图 1.2 DIF-FFT 流程图

当然，通过改变运算顺序和增加中间存储器，也可以实现输入和输出序列均为自然顺序，但是这样会额外增加 FPGA 资源与时钟周期。在基 2DIT-FFT 方法中，蝶形算子的两个输入与输出的地址相同，即同址运算。因此使用一个深度为 N、位宽为 2M 的 RAM 作为数据存储空间即可，N 为计算 FFT 的点数，也是信号的长度；M 为数据实部与虚部的位宽大小；高 M 位存储数据实部，低 M 位存储数据虚部。

虽然使用同址运算会使得程序更加复杂，但为尽可能减少 FFT 完成的时间与消耗资源，以给后续的网口 TCP 通信实现提供足够的资源与时钟盈余，这里采用 DIT-FFT 实现计算。

1.2 蝶形运算模块 Butterfly

在图 1.1 的 DIT-FFT 运算中，最小的运算单元为一次蝶形运算。蝶形运算结构如图 1.3 所示：

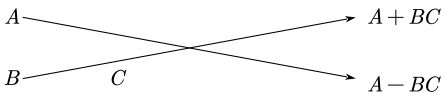

图 1.3 蝶形算子结构

图 1.3 中的 A、B 均为复数形式的输入，C 为本次蝶形运算旋转因子$W_N^k = e^{-j\frac{2\pi}{N}k}$.设$A = ar + j \cdot ai, B = br + j \cdot bi, C = cr + j \cdot ci.$

由于在 DIT-FFT 中，蝶形运算为同址运算，因此数据 A、B 计算得到的数据 A+BC、A—BC 仍然是保存在原始 A、B 存储的地址。

设$OUTA = A + BC, OUTB = A — BC$。在一次蝶形运算中，需要完成四次乘法，即 B、C 的实、虚部之间互相相乘；待乘法结束后，数据需要进行两次加减法运算。蝶形算子计算大致过程如图 1.4 所示：

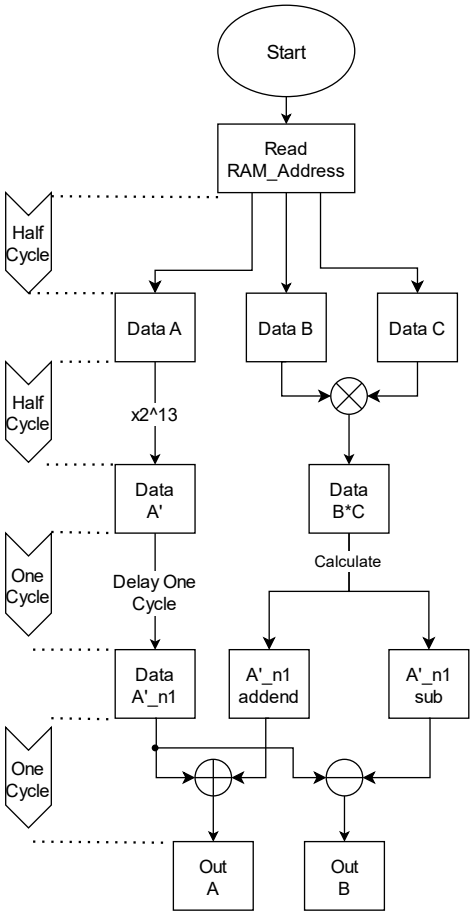

图 1.4 蝶形算子完成一次运算过程

在计算时，数据 Data A、B 的位宽均为 24 位，旋转因子 Data C 位宽为 16 位，由于$W_N^k = e^{-j\frac{2\pi}{N}k}$。在每一次蝶形运算中，旋转因子均扩大了$2^{13}$倍，并提前计算好，保存在 ROM 中，供每次蝶形运算进行选择。相应地，Data A 同样需要扩大2^{13}倍。

由图 1.4 可知，从 RAM 读到地址输出数据到最后蝶形运算计算完成，需要 3 个 Cycle.具体如下：

1）控制模块 RAM_Ctrl 输出的地址位与时钟上升沿保持同步输入至 RAM 模块中；RAM 在时钟下降沿进行读取数据，送至 Butterfly 中。

2）送至 Butterfly 中的数据与时钟下降沿保持同步。在 Butterfly 中，在时钟上升沿对 Data A 完成移位操作，同时并发事件有 Data B、C 完成相乘操作。

3）在下一个时钟上升沿来临时，对移位完成的 Data A 进行寄存一拍，同时并发事件为对 Data B、C 相乘得到的数据进行加减操作。

4）计算输出。

上述过程并保持了流水线结构，最大限度地节省了时钟周期。流水线时序示意图如图 1.5 所示。

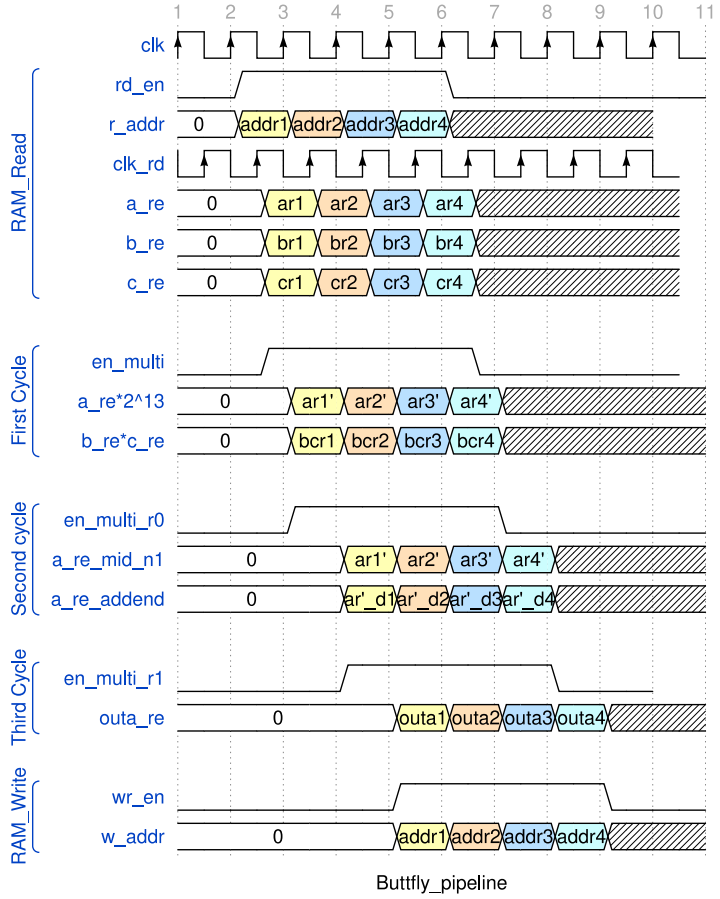

图 1.5 蝶形运算流水线

由图 1.5 可知，从读取地址到输出最后结果，共需要 3 个 Cycle。因此计算 n 次蝶形运算共需要消耗 n+3 个 Cycle。

1.3 控制模块 RAM_Ctrl

由图 1.1 可知，蝶形运算的两个输入节点与蝶形算子的选择存在一定数学规律。假设 N 为 2 的整数次幂，如果要完成 N 点 FFT，则需要完成log_2N级计算。此时 FFT 按照 L=1,2…,log_2N的顺序完成各级运算。

在各级完成运算时，每级运算对应有 N/2 个蝶形运算，并且存在2^{L-1}种不同的旋转因子。假设在 L 级中，采用相同旋转因子完成计算的索引为第 J 层。如在第 2 级、第 1 层的所有蝶形运算使用的旋转因子相同。

在第 L 级运算时，对应有2^{L-1}种旋转因子。因此在 L 级中按照 J=0,1,…,$2^{L-1}-1$的次序完成运算。在第 L 级、第 J 层中，蝶形运算节点 A 的取值满足 J:2^L:N。

上述过程由 RAM_Ctrl 控制模块实现，并且伪代码如下所示：

```
L=1:1:L_MAX
        J=0:1:2^(L-1)-1
           factor_re_r <= rom[(2^(L_max-L))*J][31:16]
           factor_im_r <= rom[(2^(L_max-L))*J][15:0]
              k=J:2^L:N
                 rd_add1 = k;
                 rd_add2 = k+2^(L-1);
```

其中$L_MAX = log_2 N$；rd_add1、rd_add2 为取出数据 Data A、B 的地址位；factor_re_r、factor_im_r 分别为旋转因子的实部和虚部。程序流图如图 1.6 所示。

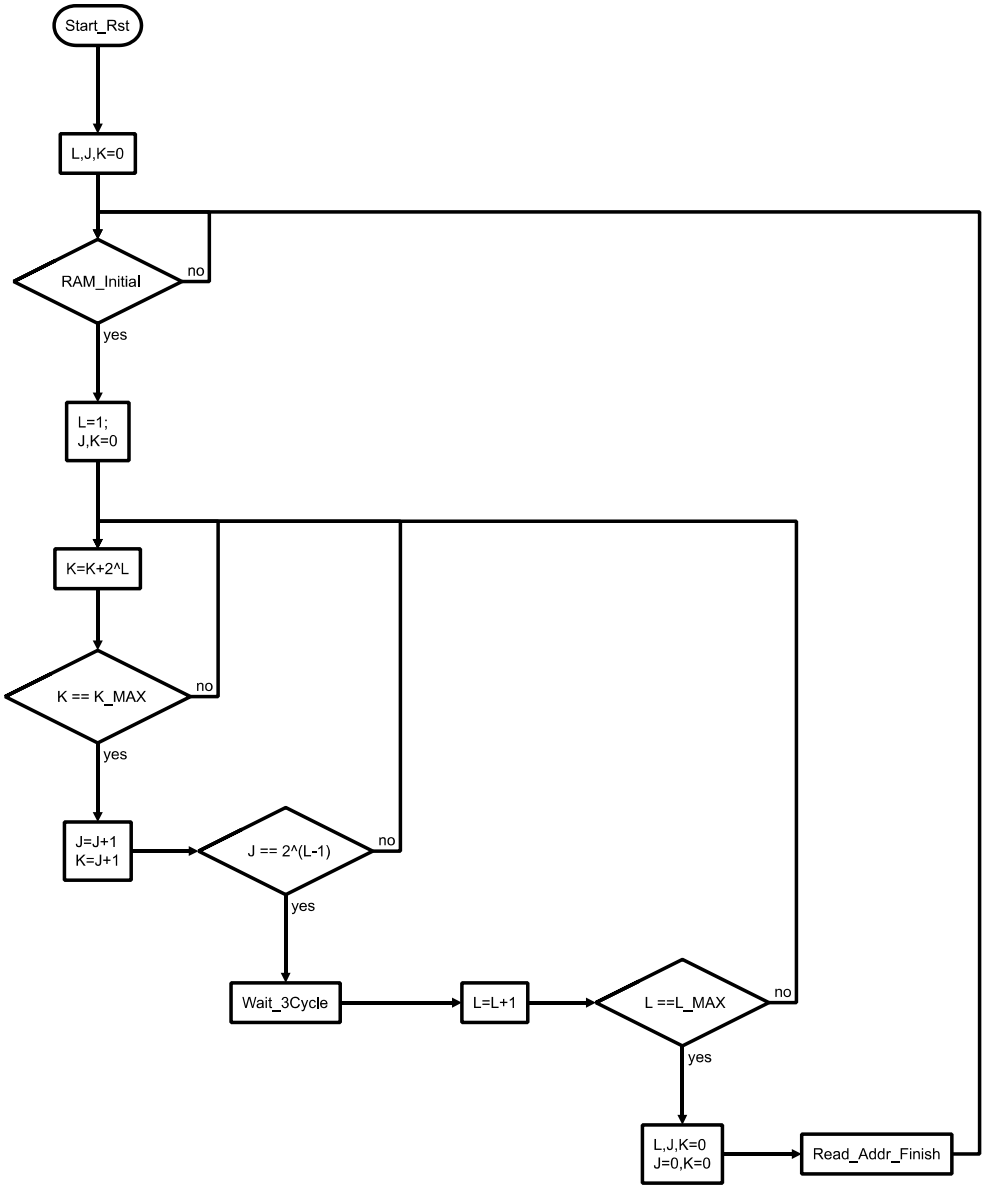

图 1.6 RAM_Ctrl 程序流图

由图 1.6 可知，在每次 FFT 开始前，要等待 RAM 中数据初试化完毕，也就是对初始数据按照“码位倒读”顺序进行存储完成后，才可以开始第一级的运算；否则需要一直等待初试化结束。

并且在计算完每一级运算之后，需要等待 3 个 Cycle 作为数据写入缓冲期，以避免在 RAM 中同时对一个地址进行读写操作。如图 1.7 所示。

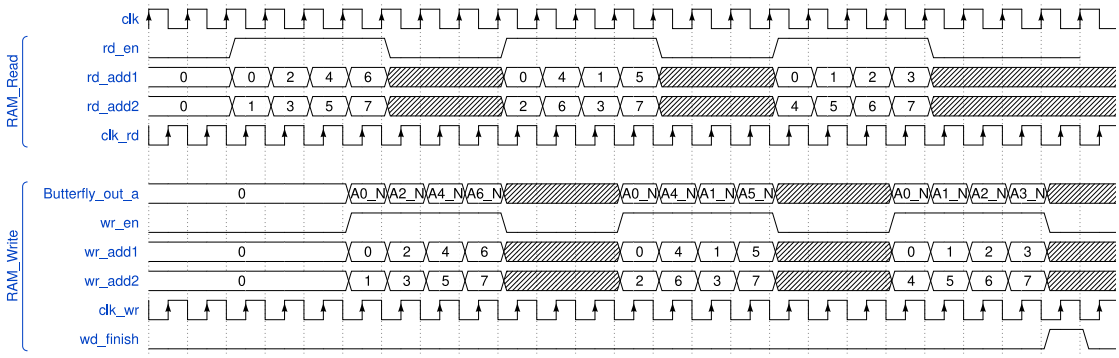

图 1.7 RAM_Ctrl 控制读写操作

待所有操作完成之后，输出 wd_finish 信号，代表本次数据 FFT 结束，此时进入等待期，等待下一次 FFT 操作开始。

1.4 N 点 FFT 的 FPGA 结构及分析

根据 1.2、1.3 的分析，本次 FPGA 实现的整体框图如图 1.8 所示。

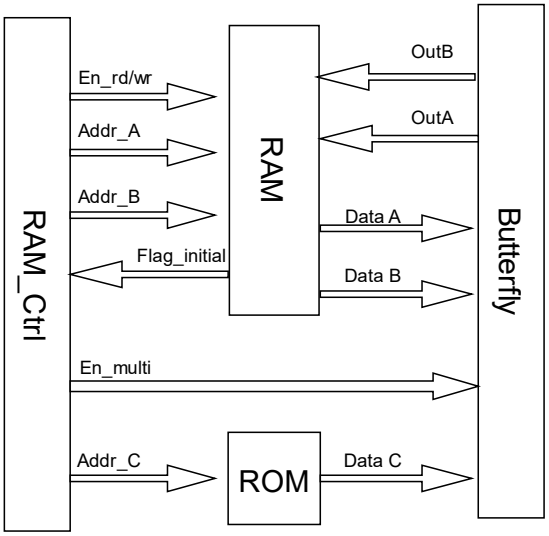

图 1.6 系统结构

据图 1.6 可知，本系统使用了一个 RAM 作为存储空间，相对于传统的 FFT 使用“乒乓结构”的两个 RAM，可以节省 50%的资源。并且每一级计算结束后，RAM 仅需要 3 个 Cycle 作为写入缓冲期，并没有因为少使用一个 RAM 而大大增加时钟消耗。

ROM 中存储的是旋转因子。第 L 级旋转因子有2^{L-1}种，每级旋转因子有 N/2 个。按照传统运算，是将每级所有的旋转因子存储下来，再寻址。

本结构利用数学规律，将每级运算又分为 J 层，J= 2^{L-1}。每层运算的旋转因子相同，因此在同一层运算时，取旋转因子的地址位不变即可。此时我们仅需存储$N-1$个不同的旋转因子，相对于传统需要存储$Nlog_2N/2$个旋转因子，可以节省较多的 ROM 资源。