

基于高速 FFT 结构的频域抗干扰算法的 FPGA 实现

周景龙

(电子科技大学 电子科学技术研究院, 四川 成都 611731)

摘 要: 提出了一种基于高速 FFT 结构的算法硬件设计与实现, FFT 采用基 4 算法, 旋转因子采用 CORDIC 算法生成, 节省了存储资源, 最后在硬件平台上测试, 取得了很好的抗干扰效果。

关键词: 频域抗干扰算法; FFT; 基 4; CORDIC

中图分类号: TN402

文献标识码: A

文章编号: 1000-7180(2014)05-0032-04

A FPGA Implementation of Frequency-Domain Anti-jamming Algorithm Based on a Structure of High-Speed FFT

ZHOU Jing-long

(Research Institute Electronic Science and Technology,
University of Electronic Science Technology of China, Chengdu 611731, China)

Abstract: This paper focuses on the algorithm hardware design and implementation base on a structure of high-speed FFT. The FFT uses radix-4, and the twiddle-factor generated by CORDIC. This way can save a lot of storage sources. Finally, by testing on the hardware platform, good anti-jamming effect has achieved.

Key words: frequency domain anti-jamming algorithm; FFT; radix-4; CORDIC

1 引言

GNSS 卫星位于离地面 2 万多公里的高空, 导航信号传送到地面非常微弱, 一般比高斯白噪声还要低 30 多个 dB, 所以导航信号非常容易受到环境的干扰. 频域抗干扰算法是目前卫星导航领域广泛采用的一种抗干扰技术, 它具有较好的抗窄带干扰效果. 该算法的依据是, 相对于扩频信号(导航信号), 窄带干扰的能量集中在较窄的频带, 在频域上, 扩频信号的频谱比较平坦, 而窄带干扰的频谱却呈现出一个较窄的尖峰. 将混有窄带干扰的导航信号通过 FFT 变换到频域后, 可以很容易检测到干扰谱线对应的位置, 将这些谱线置零, 然后再做 IFFT 变换回时域, 就会获得抑制了窄带干扰的导航信号^[1].

在实际的应用中, 由于需要对信号每次选取一段进行处理, 势必会发生频谱泄露现象, 故要对信号进行加窗, 同时为了避免加窗导致的信噪比降低, 一

般采用如下的方法进行处理^[2]. 算法原理图如图 1 所示.

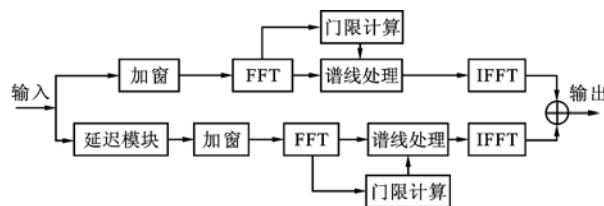


图 1 频域抗干扰算法原理图

2 基 4 算法原理

在高速 FFT 设计中, 采用的是 Radix4 算法, 下面对 Radix4 原理作一下简要介绍. 对于序列 $x(n)$, $n = 0, 1, \dots, N-1$ 的离散傅里叶变换(DFT) 为

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk},$$

其中,

$$W_N^{nk} = e^{-j\frac{2\pi}{N}nk}, k = 0, 1, \dots, N-1 \quad (1)$$

基4DIT(Decimation-In-Time)在时域上将 $x(n)$ 进行抽取,即将 n 分解为 $4m$ 、 $4m+1$ 、 $4m+2$ 、 $4m+3$,因此式(1)可以写成^[3]

$$X(k) = \sum_{n=0}^{N/4-1} x(4m) W_N^{4nk} + \sum_{n=0}^{N/4-1} x(4m+1) W_N^{(4m+1)k} + \sum_{n=0}^{N/4-1} x(4m+2) W_N^{(4m+2)k} + \sum_{n=0}^{N/4-1} x(4m+3) W_N^{(4m+3)k} \quad (2)$$

又因为 $W_N^{4nk} = W_{N/4}^{nk}$,再令 $A = \sum_{n=0}^{N/4-1} x(4m) W_{N/4}^{nk}$, $B = \sum_{n=0}^{N/4-1} x(4m+1) W_{N/4}^{nk}$, $C = \sum_{n=0}^{N/4-1} x(4m+2) W_{N/4}^{nk}$, $D = \sum_{n=0}^{N/4-1} x(4m+3) W_{N/4}^{nk}$,根据旋转因子 W_N^k 的性质,结合式(2)得到

$$\begin{cases} X(k) = A + BW_N^k + CW_N^{2k} + DW_N^{3k} \\ X(k+N/4) = A - jBW_N^k - CW_N^{2k} + jDW_N^{3k} \\ X(k+N/2) = A - BW_N^k + CW_N^{2k} - DW_N^{3k} \\ X(k+3N/4) = A + jBW_N^k - CW_N^{2k} - jDW_N^{3k} \end{cases} \quad (3)$$

通过观察式(3),右侧算式可以进一步重组以易于蝶形运算单元的硬件实现,有

$$\begin{cases} X(k) = (A + CW_N^{2k}) + (BW_N^k + DW_N^{3k}) \\ X(k+N/4) = (A - CW_N^{2k}) - j(BW_N^k - DW_N^{3k}) \\ X(k+N/2) = (A + CW_N^{2k}) - (BW_N^k + DW_N^{3k}) \\ X(k+3N/4) = (A - CW_N^{2k}) + j(BW_N^k - DW_N^{3k}) \end{cases} \quad (4)$$

图2是根据式(4)画出的基4蝶形运算单元。

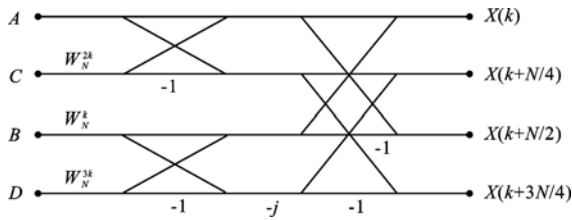


图2 基4蝶形运算单元

3 频域抗干扰算法设计方案

3.1 算法实现的基本结构

算法硬件结构如图3所示。

在频域抗干扰算法中,要对输入的一段数据作加窗处理,在这里选取1024点数据作为一段进行处理。我们将预先把1024点的Hamming窗函数值存储在一块ROM中,AD采集来的数据先存储到一块1024点的SRAM中,然后与ROM中数据进行相乘,也就是对输入数据加窗。先是逐一按顺序与ROM中的窗函数值进行相乘,然后送到后面的双

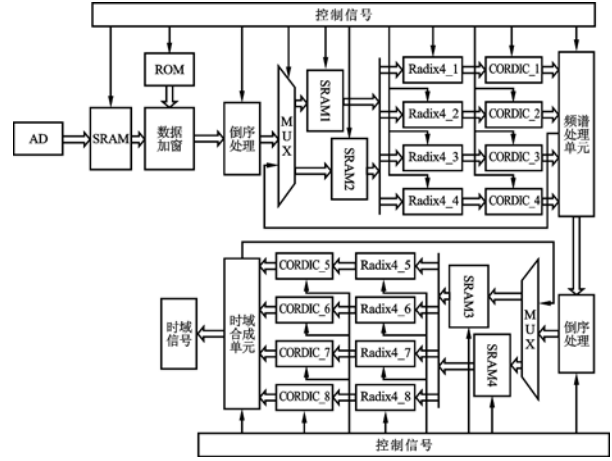


图3 频域抗干扰算法硬件实现基本结构

口SRAM1中,并开始后续FFT处理,同时将该段数据延迟512点再加窗,即该段数据的前512点数据与ROM中后512点逐一按顺序相乘,而该段数据的后512点与ROM中的前512点逐一按顺序进行相乘,然后送到后面的SRAM2中,于此同时,开始新一段数据的输入,也就是所谓的“乒乓”操作,这样可以提高计算的速度和效率。

在写入SRAM1和SRAM2之前,要先对加窗后的数据进行倒序处理。运算部分是有4个蝶形单元组成,可以同时计算16个数据,计算出的结果送到后面的CORDIC运算单元中,与CORDIC提供的旋转因子相乘,然后将运算结果原位返回到SRAM中,也即“原位运算”。直到所有数据都完成了这样的运算,则此时完成了第一级计算。在第二级的计算中,同样先取16个数据,将第1、5、9、13个数据送到第一个蝶形单元,第2、6、10、14个数据送到第二个蝶形单元,后面的以此类推。后面的三、四、五级计算实现类似。数据读取顺序由控制单元和地址产生单元联合实现。当SRAM1中的数据经FFT计算结束后,SRAM2中数据应写入完毕,并开始进行FFT计算,与上述过程一致。同时开始新一段数据输入,并写入到SRAM1中。

当SRAM中的数据在进行最后一级(第五级)FFT计算时,计算出的结果要同时送入到SRAM1中存储和后面的频谱处理单元进行累加求和(计算频谱门限值),当FFT计算结束后,有三个操作:(1)将SRAM中的数据输出到频谱比较单元与计算出的门限值进行比较,去除干扰;(2)向SRAM中写入新的数据;(3)SRAM中的数据开始FFT计算。这样做的目的主要就是节省硬件资源和提高处理效率。

频谱处理单元输出的数据送入到后面的IFFT

单元中,将新得到的频域数据(滤除窄带干扰的数据)转换到时域. IFFT 单元的硬件实现与 FFT 实现结构是一样的,比较 DFT 的综合和分析表达式,可以看到两式仅有两处不同,一是 IDFT 比 DFT 表达式多了一项比例常数 $\frac{1}{N}$,这个在硬件实现中

$$\begin{cases} X(k) = \text{DFT}[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ x(n) = \text{IDFT}[X(k)] = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \end{cases} \quad (5)$$

很容易做到;二是在表达式中 IDFT 旋转因子是 W_N^{-nk} ,这个变化影响的只是 CORDIC 单元中正弦和余弦的符号,也很容易实现. 所以 IFFT 单元的硬件结构与 FFT 几乎是完全一致的. 当加延迟和未加延迟两路信号都进行 IFFT 后,将两路时域信号合成输出,即去除窄带干扰后的时域信号.

3.2 CORDIC 运算单元的实现

高速 FFT 中的旋转因子采用 CORDIC 运算单元生成,这样可以节省大量的存储资源,CORDIC 运算单元采用流水线方式实现,计算速度快^[3],如图 4 所示.

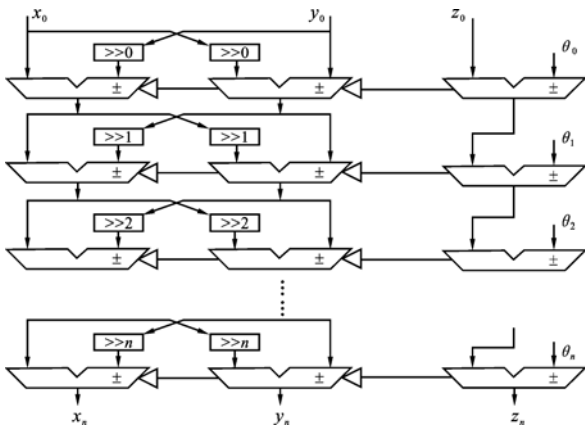


图 4 CORDIC 运算单元的流水线实现

计算时,只需将数据的实部和虚部分别输入到上图中的 x_0 和 y_0 ,将旋转因子中的 $-\frac{2nk\pi}{N}$ (IFFT 为 $\frac{2nk\pi}{N}$) 部分输入到 z_0 ,通过 n 次迭代实现数据与旋转因子的相乘结果^[4]. 由于旋转因子角度的正负,只是影响正弦函数的符号,也即迭代算式中的符号,若都是以角度 $\frac{2nk\pi}{N}$ 的形式输入到 z_0 ,则有^[5]:

FFT:

$$\begin{cases} x_{n+1} = x_n + \delta_n y_n 2^{-n} \\ x_{n+1} = x_n - \delta_n y_n 2^{-n} \\ z_{n+1} = z_n - \delta_n \arctan 2^{-n} \end{cases};$$

IFFT:

$$\begin{cases} x_{n+1} = x_n - \delta_n y_n 2^{-n} \\ x_{n+1} = x_n + \delta_n y_n 2^{-n} \\ z_{n+1} = z_n - \delta_n \arctan 2^{-n} \end{cases}.$$

在 CORDIC 运算单元中计算旋转因子时需要注意的是,其所能旋转角度总和是具有一定的范围 $(0, z_{\max})$ 的,其中 $z_{\max} = \lim_{n \rightarrow \infty} \sum_{i=1}^n \arctan(2^{-i}) = 99.911^\circ$. 当旋转的角度超过 $z_{\max}$ 时,若直接用 CORDIC 运算单元进行旋转,则结果将不收敛,此时需要进行预处理. 根据三角函数的性质,可以很方便的得到预处理结果^[6],如表 1 所示.

表 1 预处理操作

旋转角度范围	预旋转角度	x_0	y_0
$[0, \frac{\pi}{2}]$	0	x_{in}	y_{in}
$[\frac{\pi}{2}, \pi]$	$\frac{\pi}{2}$	y_{in}	$-x_{in}$
$[\pi, \frac{3\pi}{2}]$	π	$-x_{in}$	$-y_{in}$
$[\frac{3\pi}{2}, 2\pi]$	$\frac{3\pi}{2}$	$-y_{in}$	x_{in}

3.3 旋转因子的生成

旋转因子生成的传统方法就是将旋转角度的正、余弦值先存储到一块 ROM 中,然后通过控制电路将需要的值送入到蝶形单元中. 这里采用一种方便的方法,不需要事先存储,减少了硬件资源占用,这对大点数的 FFT 来说是有意义的. 对于 FFT 计算的每一级,其旋转因子都有一定的规律. 第一级蝶形运算不需要旋转因子,故可以将 SRAM 中的数据直接输入到蝶形单元中计算. 第二级计算中,数据要乘的旋转因子是 W_{16}^{nk} ,其中 $n=0,1,2,3, k=0,1,2,3$,以 SRAM 中 16 个数据作为一组. 第三级计算中,数据要乘的旋转因子是 W_{64}^{nk} ,其中 $n=0,1,2,3, k=0,1,2,3, \dots, 15$,以 SRAM 中 64 个数据作为一组. 第四、五级旋转因子形式则是 W_{256}^{nk} ($n=0,1,2,3, k=0,1,2,3, \dots, 63$) 和 W_{1024}^{nk} ($n=0,1,2,3, k=0,1,2,3, \dots, 255$),可见计算的级数确定了旋转因子的形式,即分母的大小. 由 CORDIC 算法可以知道,就其实质而言, z_n 和 θ_n 只需要比较其大小就可以确定下一次的旋转方向,旋转的方向最终影响的是所要求的旋转因子的值, z_n 和 θ_n 各自具体是什么值对旋转方向没有影响,而是它们比较后的结果确定下一次旋转的方向,故可以将 z_n 和 θ_n 提取出因子 π/M ,这样只需要通过 n 和 k 的值就可以求得旋转因子的值. n 和 k 与 SRAM 中数据存储的地址有关,由于采用的是原位计算,这种旋转因子生成方法方便实用,

对于 1024 点 FFT, N 个数据在 SRAM 中存储地址以二进制形式表示, 其中各级旋转因子的分母, 即 M , 其减 1 对应的 bit 位数的高两位反序为 n 值, 其他低位表示的就是 k 值。

4 FPGA 实现与功能验证

4.1 硬件测试平台

选择的硬件平台是一个小型的 GNSS 导航接收机, 抗干扰模块主要是在基带部分进行, 在进行实际验证时主要使用的部分器件如下:

(1) 两片 ADI 公司的 AD9251: AD 输出两路信号, 其量化位数为 14 位, 工作时钟频率为 61.38 MHz, 该频率可以用板上 10.23 MHz 的晶振在内部经 PLL6 倍频得到。

(2) 基带板的 FPGA 采用 Altera 公司的 Stratix III 系列 EP3SL340F1517C3N。

(3) 一个频率为 10.23 MHz 的恒温晶振, 为基带处理板提供统一时钟源。

(4) 此外, 还包括开关电源、存储、串口通信以及电平转换芯片等。

4.2 功能验证

本设计采用 Verilog 语言实现, SRAM 存储器采用 FPGA 内置的 RAM 资源配置实现, 测试中, 在导航信号中加入频率为 1 268.52 MHz 的窄带干扰, 其中, 导航信号由华力创通公司导航信号源 HWA-RNSS-7600 产生, 干扰信号由安捷伦公司信号发生器 E4438C 提供, 经 Quartus II 仿真综合, 仿真结果如图 5 所示。通过 Quartus II 的在线逻辑分析软件 SignalTap 分别采集未经过抗干扰处理和经过抗干扰处理后的输出信号, 导入 Matlab 中分析对比各自的频谱, 从实验结果可以看出经过抗干扰处理后, 很好的去除了混在信号中的干扰信号。

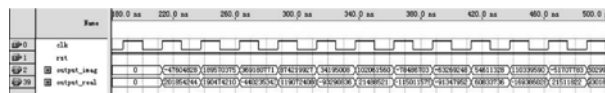


图 5 Quartus II 仿真结果

5 结束语

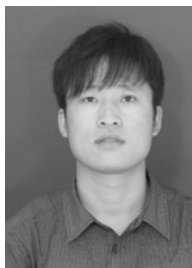
本文研究了一种基于高速 FFT 结构的抗干扰算法的 FPGA 实现, 可以实现对 1024 点信号采集

的实时处理, 对于处理窄带干扰有显著的效果, 可以应用在接收机的实现中。输入端是经 AD 采集后的实部和虚部均为 14 bit 的数据, 内部 FFT 架构采用块浮点的数据格式, 其以定点运算的速度达到了浮点运算的精度, 且耗费资源与定点运算相当。旋转因子采用 CORDIC 算法生成, 节省了大量存储资源。从 Matlab 对处理后的数据进行分析可以知道, 混有窄带干扰的导航信号经过该抗干扰硬件模块处理后, 达到了预期效果。

参考文献:

- [1] Young J A, Lehnert J S. Analysis of DFT-based on frequency excision algorithms for direct sequence spread-spectrum communications[J]. IEEE Trans on communications, 1998, 46 (8):1076-1087.
- [2] Paul T Capozza. A single-chip narrow-band frequency domain excisor for a global positioning system (GPS) receiver[J]. IEEE, Solid-State Circuits, 2000, 35(3): 401-411.
- [3] 李成诗, 初建朋, 李新兵. 基于 CORDIC 的一种高速实时定点 FFT 的 FPGA 实现[J]. 微电子学与计算机, 2004, 21(4): 88-96.
- [4] 董惠, 卫铭斐, 江丽, 等. 基于 FPGA 的 FFT 处理器的设计与仿真[J]. 微电子学与计算机, 2008, 25(11): 117-120.
- [5] Garrido M, Grajal J. Efficient memoryless CORDIC for FFT computation [C]// International Conference on Acoustics, Speech and signal Procession (ICASSP2007). Honolulu, IEEE, 2007: II/113-II/116.
- [6] Liu G H, Feng Q Y. ASIC design of low-power reconfigurable FFT processor[C] // ASICON'07. China: Shanghai, IEEE, 2007: 44-47.

作者简介:



周景龙 男, (1987-), 硕士研究生. 研究方向为卫星导航中的抗干扰算法。