多级嵌套维纳滤波器中协方差矩阵计算的 FPGA 实现

邓路宽[1,2],李双勋[1],程 翥[1],皇甫堪[1]
(1.国防科学技术大学 电子与科学工程学院,湖南 长沙 410073;
2.武警 8732 部队,湖南 耒阳 421800)

摘 要:通过分析观察信号协方差矩阵构成的特点,提出了一种适合 FPGA 实现的协方差矩阵计算方法,为多级嵌套维纳滤波器提供了更易于工程实现的方法。

关键词:MSNWF 协方差矩阵 FPGA

估计理论中的一般性问题是从观测信号 $x_0(n)$ 恢复未知信号 $d_0(n)$。维纳滤波器(WF)仅仅利用二阶统计特性在最小均方误差(MMSE)意义下解决了这个问题。WF 易于实现,因此应用广泛。然而,产生的滤波器需要计算观测信号的协方差矩阵的逆。这就意味着如果观测信号 $x_0(n)$ 的维数很高,将使工程实现十分复杂。多级嵌套维纳滤波器 MSNWF(Multi-Stage Nested Wiener Filter)可以在观测信号协方差矩阵和观测信号、期望信号互相关矢量的 Krylov 子空间中求解出 Wiener-Hopf 方程的解。无论最早提出的主成分法 PC (Principal Component)[1]、互谱密度法 CS(Cross-Spectral)[2],还是近年提出的基于 Lanczos 算法[3]和共轭梯度法 CG(Conjugate Gradient)算法的 MSNWF[4],都是围绕降秩提出的,通过降秩简化运算,使算法更容易实现。但最终实现 MSNWF 都无一例外地需要计算观测信号的协方差矩阵和观测信号、期望信号的互相关矢量。因此计算协方差矩阵是实现 MSNWF 的关键部分,也是整个系统中计算量最大、工程实现最复杂的部分。本文通过分析观察信号协方差矩阵构成的特点,充分利用 FPGA 集成的内部器件,如存储器和数字信号处理(DSP)单元,提出了一种适合 FPGA 实现的协方差矩阵计算方法。此方法设计的协方差计算模块占用器件资源少、可扩展性强、计算速度快,使器件能保留足够多的资源实现算法中权值和均方误差的更新。作者应用该方法在某空时自适应系统中实现了文献[4]中提出的 CG MSNWF 算法。

1 设计方案分析

1.1 协方差矩阵构成特点

本文以一个阵元数 $M=3$、时间延时单元数 $P=3$ 的空时自适应 MSNWF 为例,研究实现该算法中协方差矩阵计算的方法。

设输入数据组成的矩阵为:

$$X=[X_1(1)\quad X_1(2)\quad X_1(3)\quad X_2(1)\quad L\quad X_3(3)]^{\mathrm{T}} \tag{1}$$

其中:$X_i(j)=[x_i(j)x_i(j+1)\ L\ x_i(j+35)]$,$i,j\in\{1,2,3\}$,表示阵元 i 在时间上的第 j、第 $j+1$、第 $j+2$ 个延时矢量。$(\ )^{\mathrm{T}}$ 为转置。每个 $X_i(j)$ 为一个 36 维的行矢量(参考文献[5]中关于 MSNWF 降秩自适应处理收敛所需数据点数的准则 $L=4MP$)。

输入矢量 x 的协方差矩阵为:

$$R_{XX}=[R_{X_i(j)X_m(n)}]=[X_i(j)gX_m^{\mathrm{H}}(n)] \tag{2}$$

$R_{X_i(j)X_m(n)}$ 表示协方差矩阵中的一个元素。其中,$i,j,m,n\in\{1,2,3\}$,$(\ )^{\mathrm{H}}$ 为共轭转置。

将式(2)右边展开为如下矩阵表示形式:

$$\left[\begin{array}{ccccc}
R_{X_1(1)X_1(1)} & R_{X_1(1)X_1(2)} & R_{X_1(1)X_1(3)} & R_{X_1(1)X_2(1)} & R_{X_1(1)X_3(3)} \\
R_{X_1(2)X_1(1)} & R_{X_1(2)X_1(2)} & R_{X_1(2)X_1(3)} & R_{X_1(2)X_2(1)} & R_{X_1(2)X_3(3)} \\
R_{X_1(3)X_1(1)} & R_{X_1(3)X_1(2)} & R_{X_1(3)X_1(3)} & R_{X_1(3)X_2(1)} & R_{X_1(3)X_3(3)} \\
\hline
R_{X_2(1)X_1(1)} & R_{X_2(1)X_1(2)} & R_{X_2(1)X_1(3)} & R_{X_2(1)X_2(1)} & R_{X_2(1)X_3(3)} \\
R_{X_2(2)X_1(1)} & R_{X_2(2)X_1(2)} & R_{X_2(2)X_1(3)} & R_{X_2(2)X_2(1)} & R_{X_2(2)X_3(1)} \\
R_{X_2(3)X_1(1)} & R_{X_2(3)X_1(2)} & R_{X_2(3)X_1(3)} & R_{X_2(3)X_2(1)} & R_{X_2(3)X_3(1)} \\
\hline
R_{X_3(1)X_1(1)} & R_{X_3(1)X_1(2)} & R_{X_3(1)X_1(3)} & R_{X_3(1)X_2(1)} & R_{X_3(1)X_3(1)} \\
R_{X_3(2)X_1(1)} & R_{X_3(2)X_1(2)} & R_{X_3(2)X_1(3)} & R_{X_3(2)X_2(1)} & R_{X_3(2)X_3(1)} \\
R_{X_3(3)X_1(1)} & R_{X_3(3)X_1(2)} & R_{X_3(3)X_1(3)} & R_{X_3(3)X_2(1)} & R_{X_3(3)X_3(3)}
\end{array}\right]$$

从展开式可以看出,矩阵可以看成 3 个独立部分构成,每个部分分别为 1 个阵元的时间延时数据与 3 个阵元的所有时间延时数据进行卷积运算。若令:

$$R_{X_1}=\left[\begin{array}{ccccc}
R_{X_1(1)X_1(1)} & R_{X_1(1)X_1(2)} & R_{X_1(1)X_1(3)} & R_{X_1(1)X_2(1)} & \mathrm{L} \quad R_{X_1(1)X_3(3)} \\
R_{X_1(2)X_1(1)} & R_{X_1(2)X_1(2)} & R_{X_1(2)X_1(3)} & R_{X_1(2)X_2(1)} & \mathrm{L} \quad R_{X_1(2)X_3(3)} \\
R_{X_1(3)X_1(1)} & R_{X_1(3)X_1(2)} & R_{X_1(3)X_1(3)} & R_{X_1(3)X_2(1)} & \mathrm{L} \quad R_{X_1(3)X_3(3)}
\end{array}\right] \tag{3}$$

则式(2)可以表示为:

$$R_{XX}=[R_{X_1}\quad R_{X_2}\quad R_{X_3}]^{\mathrm{T}} \tag{4}$$

从式(3)可以看出,R_{X_1} 是输入矩阵中阵元一的 3 个时间延时矢量分别与共轭转置矩阵中每个矢量相乘的结果,若使用存储器将输入矩阵预先存入 RAM,计算时通过地址选择不同的时间、空间延时矢量,依次通过乘累加器件进行累加运算,即复用 1 个复数乘累加器件即

可得到 R_{X_1} 的各个结果。采用 3 组与实现 R_{X_1} 同样的计算模块就可完成协方差矩阵的硬件实现。这种设计方法既缩短了设计周期，也充分考虑了设计系统的计算速度和占用器件资源等问题，使系统能在使用较少乘法器资源的情况下获得较快的计算时钟频率，而且这种设计方案的扩展性和重用性、移植性都要优于直接实现。

1.2 设计思路

从上述分析可知，如果采用固定的高速卷积运算单元进行乘累加运算，通过存储器控制数据总线上数据变化，可以将矩阵计算转化为简单的存储器读写控制设计，只需控制存储器的读写时钟和地址就可以在数据总线上得到不同的数据，参与乘法运算，从而得到不同输入矢量的乘积和。

以计算式(3)所示 R_{X_1} 为例，假设计算按顺序执行，根据时间先后可将矩阵计算分为 3 个矢量与矩阵相乘的运算，即：

$$\begin{array}{c} \boldsymbol{T} \\ j=1 \\ j=2 \\ j=3 \end{array} \begin{array}{c} \begin{array}{ccccc} m=1,n=1 & m=1,n=2 & m=1,n=3 & m=2,n=1 & m=3,n=3 \end{array} \\ \begin{bmatrix} R_{X_1(1)X_1(1)} & R_{X_1(1)X_1(2)} & R_{X_1(1)X_1(3)} & R_{X_1(1)X_2(1)} & R_{X_1(1)X_3(3)} \\ R_{X_1(2)X_1(1)} & R_{X_1(2)X_1(2)} & R_{X_1(2)X_1(3)} & R_{X_1(2)X_2(1)} & R_{X_1(2)X_3(3)} \\ R_{X_1(3)X_1(1)} & R_{X_1(3)X_1(2)} & R_{X_1(3)X_1(3)} & R_{X_1(3)X_2(1)} & R_{X_1(3)X_3(3)} \end{bmatrix} \end{array}$$

其中 $X_1(j)$ 表示阵元 1 的 3 个时间延时矢量，$X_m(n)$ 表示输入共轭转置矩阵中阵元 M 的第 n 个时间延时矢量，$T_{j,m,n}$ 用来表示得到矩阵 R_{X_1} 中 $X_1(j)$ 与 $X_m(n)$ 的乘累加结果的时间，其中 $j,m,n \in \{1,2,3\}$。

若预先将参与计算的阵元的所有时间延时数据分别存入各自的存储器，令 j、n 表示相应的时间延时单元数，m 表示共轭转置中阵元数据存储的存储器标号，$T_{j,m,n}$ 表示在当前时刻，将阵元一存储器中的第 j 个时间延时矢量和存储器 m 中的第 n 个延时时间矢量读出，送往乘累加运算单元得到的矩阵结果。依次改变 j、m、n 的值，即可完成矩阵 R_{X_1} 的计算。

图 1 所示为矩阵 R_{X_1} 计算的简单示意框图，将 $X_1(j)$，$j \in (1,2,3)$ 存入第 1 个 RAM 中，读出的数据直接送往乘累加器 MAC 进行计算；$X_i(j)^{\mathrm{H}}$，$i,j \in (1,2,3)$ 分别存储在其他 3 个 RAM 中，通过 MUX 选择器选择一路数据送往乘累加器 MAC 进行计算，结果存入 R_{X_1} 存储器中。

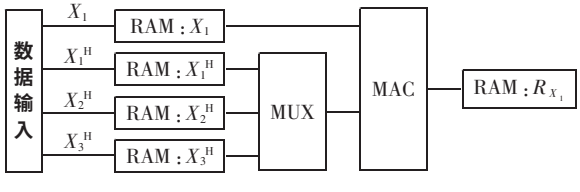


图 1 R_{X_1} 计算原理示意图

2 FPGA 实现

2.1 片内存储器的使用

存储器的使用和控制是实现矩阵计算的关键，正确写入和读取数据直接决定着整个计算的完成。一般 FPGA 器件提供 3 种内部 RAM 结构供用户选择，即单口 RAM、双口 RAM、real 双口 RAM。本文采用双口 RAM 完成设计。其元件图如图 2 所示。

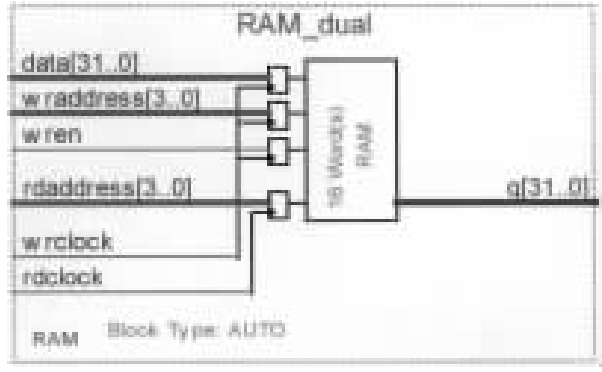


图 2 双口 RAM 元件图

图 2 所示的双口 RAM 元件有两组时钟和地址信号，分别为读、写时钟和地址信号，可以通过不同的读写时钟、地址信号来对 RAM 进行写入和读取操作，得到不同的输出数据。所以存储器的正确使用就转化为存储器读写时钟和地址信号的正确控制，设计能产生正确的读写控制信息的控制模块成为完成整个设计的关键。

2.2 Rxx 计算控制模块设计

Rxx 计算控制模块是实现协方差矩阵计算的微控制器，设计中其他所有模块的工作完全由控制模块控制，由惟一的计数时钟 sys_clk 计数产生所需的控制信号。这样既有利于整体设计的无缝连接，又将状态机的设计融入控制器设计中，保证设计能达到最高的工作频率，得到最佳的系统性能。

Rxx 的计算控制输出分 3 部分：

(1)读写时钟和地址信号。这部分控制的完成是实现整个矩阵计算的关键。结合式(3)和图 1 可以看出，首先利用写时钟信号 w_clk、写地址信号 w_addr 将参与计算的输入信号写入 RAM 中；然后同时发送两组读时钟和地址信号从相应的两组 RAM 中读取一个 36 维矢量送往 MAC 进行乘累加运算。以计算矩阵 R_{X_1} 的第一个元素 $R_{X_1(1)X_1(1)}$ 为例，只需产生读时钟信号 r_clk、读地址信号 r_addr，r_clk_1、读地址信号 r_addr_1，分别从 $X_1(j)$，$j \in (1,2,3)$ 存储器(RAM1)、$X_1(j)^{\mathrm{T}}$，$j \in (1,2,3)$ 存储器(RAM2)中读出 $X_1(1)$ 和 $X_1(1)^{\mathrm{T}}$ 进行乘累加计算，其他存储器的时钟和地址信号置零。计算 $R_{X_1(1)X_1(2)}$ 时保持 RAM1 的时钟和地址信号、RAM2 的时钟信号，将读取 RAM2 的地址信号加 1，得到延时 1 个时间单元的延时矢量 $X_1(2)^{\mathrm{T}}$，即可完成第二个元素的计算。通过同样的步骤可以完成其他元素的运算，得到不同延时时间单元的乘累加结果，从而得到协方差矩阵的计算结果。

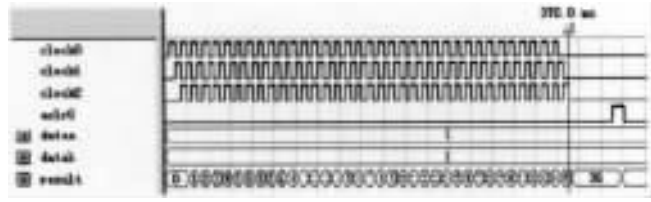
(2)RAM 读取数据选择信号 judge_1～judge_3、矩阵计算结果数据保存信号 memory_star、乘累加器清零信号 r_en_1～r_en_3。judge 信号为图 1 所示 MUX 选择器的选

择判断信号，当控制信号从 $X_i(j)^T, i, j \in (1,2,3)$ 存储器中读取数据时，将judge_i 置为高，选择器根据 judge 信号选择对应的 i 存储器的数据送往乘累加器进行运算。memory_star 为结果数据保存信号，具体应用见 2.4 节结果数据保存设计。r_en_i 信号为乘累加器的清零信号，3 个信号通过“或门”连接乘累加器的清零端，在一次 36 个数据读取完成后，发送一个 r_en 信号，对乘累加器进行清零，同时，r_en 信号还作为一个边沿触发使能信号，将累加结果清零的同时将结果送往输出管脚。

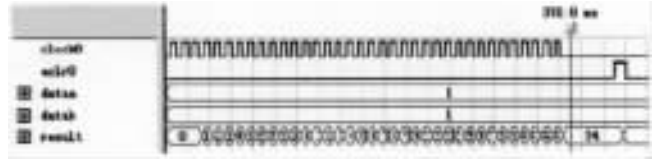
(3)乘累加器计算时钟信号。设计采用 FPGA 集成的 DSP 模块进行乘累加运算，可以大幅提高整体设计的性能。设计采用控制模块产生计算时钟，是为了便于实现精确的流水控制，在最短时间内得到计算结果，取得最高的系统工作时钟频率。

2.3 乘累加器的使用

图 3 为乘累加器件的示意图，其计算时钟由控制器产生，分别控制乘法器、乘法结果、累加器三级时钟信号流水工作，清零信号 clr 对乘累加器内所有寄存器清零。图 4 所示为使用流水线时钟和采用单一时钟进行计算的器件工作时序图。从图 4 中可以看出，同样370ns 时间内，使用图 4(a)的流水线时钟，可以进行 36 次乘累加操作，而使用图 4(b)的单一时钟，则只能进行 34 次操作。



(a)流水时钟控制的乘累加时序



(b)单一时钟控制的乘累加时序

图 4

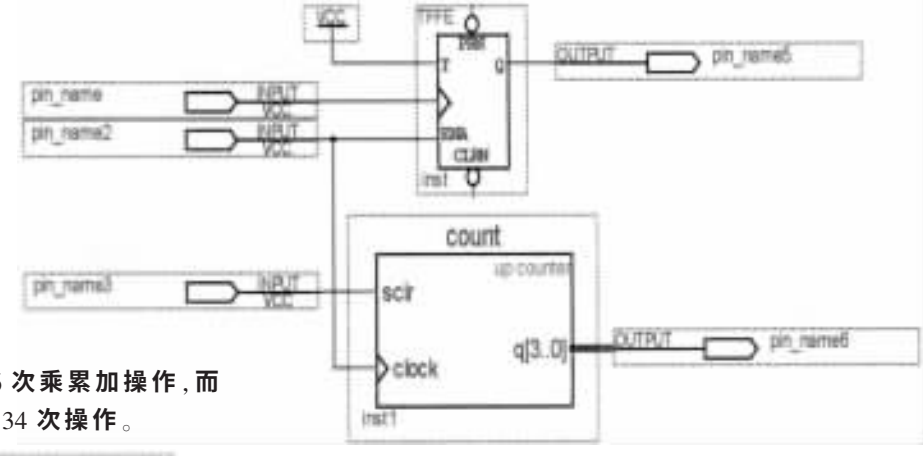


图 5 Rxx 计算结果存储控制原理图

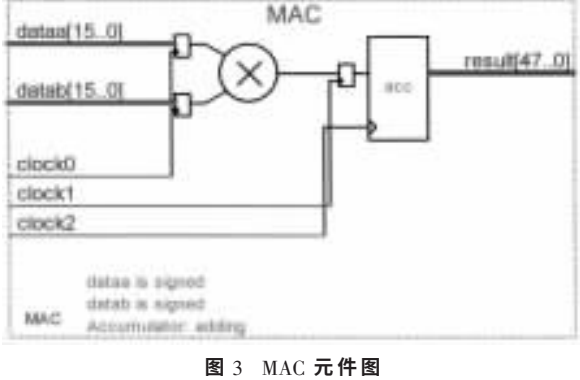


图 3 MAC 元件图

2.4 结果数据保存设计

结果数据通过清零信号 r_en 触发送往输出管脚，将数据存入 RAM 供下级计算使用协方差矩阵结果。需要对数据存入进行控制，控制模块提供片内存储器的控制信号 memory_star 和 memory_end，图 5 所示为使用这两个信号对存储器进行写操作的控制原理图。

如图 5 所示，memory_star 为 T 触发器的翻转使能信号和计数器的计数时钟信号。当 memory_star 为高时，T 触发器翻转 clk 时钟，产生写存储器的写时钟信号，同时计数器进行一次计数，产生写地址信号送往 RAM。memory_end 为矩阵计算完毕标志，同时对计数器清零，将地址置为初值零，等待下次写操作。

3 实现结果

设计选用 Altera 公司的 Stratix Ⅱ EP2S60 器件完成一个 9×36 维复数输入矩阵的协方差矩阵计算，并行运行 3 组与计算 R_{X_1} 模块相同的模块，系统存储器、乘法器以 65MHz 频率工作，完成整个协方差矩阵计算共需时 16.380μs，每个乘法器需完成 972 次计算，乘法器仅有 11%的时间处于空闲，其中包括数据写入时间。设计占用器件资源如下：

Total ALUTs：1247(2%)，
Total combinational functions：815，
Estimated Total ALMs：519，
Total registers：681，
DSP block 9-bit elements：48(16%)，
Total memory bits：49152(1%)。

本文提出了一种可以在 FPGA 上实现协方差矩阵计算方法，按照矩阵的构成特点将矩阵分块，使用 FPGA 内部集成的计算模块和存储器资源，通过控制存储器的读写时钟和地址信号，达到转换数据的目的。使高速的乘法部件变成透明，设计者只需要调整存储器的时钟和地址信号，就可以适应不同输入矩阵，无需关心乘法部件的工作状况，可移植性、可扩展性非常强。

参考文献

[1] HOTELLING H.Analysis of a complex of statistical variables into principal components.Journal of educational psychology, 1993, 24(9/10): 417-441.

[2] GOLDSTEIN J S, REED I S.Subspace selection for partially adaptive sensor array.IEEE Transactions on aerospace and electronic systems, 1997, 33(2): 539.

[3] JOHAM M, ZOLTOWSKI M D.Interpretation of the multistage nested wiener filter in the krylov subspace framework. Tech.Rep.TUM-LNS-TR-00-6, Munich university of technology, November 2000, Also: Technical report TR-ECE-00-51, Purdue University.

[4] DIETL G, ZOLTOWSKI M D, JOHAM M.Recursive Reduceed-rank adaptive equalization for wireless communications. SPIE digital wireless communication Ⅲ, vol.4395, pp.16-27, August 2001.

[5] 孙晓昶.GPS 接收机联合空时抗干扰技术研究.国防科学技术大学博士学位论文, 2003.10: (40).

(收稿日期: 2006-09-25)