串并结合的维特比译码算法的FPGA实现

摘 要

卷积码是一种应用广泛的信道纠错码，维特比算法是卷积码近最优的译码算法。随着FPGA芯片处理能力和设计的灵活性发展，在FPGA芯片中完成维特比译码算法成为了通信系统设计的一个有效手段。

本文在分析研究维特比算法的基础上，设计并实现了了一个软判决维特比译码器。译码器引入了串并结合的设计结构，和全并行的设计相比，在满足译码速度的同时，节约了芯片资源；提出了一种路径度量值存储器的组织方式，简化了控制模块的逻辑电路，优化了系统的时序；在幸存路径的选择输出上采用了回溯译码方法，减少了寄存器的使用，降低了功耗和设计的复杂度。本论文设计的译码器能够同时对两路可变速率的数据进行译码运算，达到了资源占用和数据吞吐量之间的平衡，其译码运算的核心模块具有较强的可移植性，能够应用于其他的通信系统之中。

本论文使用Verilog语言在Xilinx ISE开发环境下完成了译码器的FPGA实现，在实现过程中采用了流水线等FPGA设计方法，提高了算法的运行效率。为了验证设计的正确性，在本文中还设计了配套的仿真平台，采用了业界流行的仿真、调试工具对译码器的设计和实现进行了验证。

关键词： 卷积编码，维特比译码，FPGA，Verilog HDL

THE IMPLEMENTATION OF SERIAL-PARALLEL COMBINED VITERBI DECODER IN FPGA

ABSTRACT

Convolutional code is widely used in wireless communication systems while the Viterbi algorithm is the near-optimal decoding scheme for it. With the development of FPGA chips, the processing ability and the flexibility in the design are greatly improved, it is an excellent way to implementing the Viterbi decoder of a communication system in FPGA chips.

According to the analyzing result of Viterbi decode algorithm, this paper designs and implements a soft-decision Viterbi decoder. The decoder has introduced a Serial-Parallel combined architecture; compare with the parallel way, this design can use less chip resource to achieve the decoding rate. In order to simplify the logic circuit of the control module and optimization the system, this paper proposes a new architecture to store the path metric. This paper introduces the track back processing to get the decode results, it saves register resources and has fewer power consumption. The decoder designed in this paper can finish the decode operation to two streams of data which rate is variable. It balances the consumption of resource with the throughput capacity, the decode core can be used in other communication systems.

This paper implements the Viterbi decoder with Verilog HDL in the Xilinx ISE integrated development environment. During the implement, we use pipeline to increase the operation efficiency, and verify it by simulating. The Viterbi decoder also achieved the design requirements in the performance test.

KEY WORDS: Convolutional code, Viterbi decode, FPGA, Verilog HDL

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师的指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：杨阳 日期：2009.3.11

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：杨阳 日期：2009.3.11

导师签名：[illegible] 日期：2009.3.11

第一章　绪论

1.1　研究背景和意义

1.1.1　研究背景

与有线通信通过光纤，同轴电缆等有形的介质传输信息不同，无线通信的信息传输介质是无形的无线信道。在这里信道是对无线通信中发送端和接收端之间的通路的一种形象比喻，对于无线电波而言，它从发送端传送到接收端，其间并没有一个有形的连接，它的传播路径也有可能不只一条，但是我们为了形象地描述发送端与接收端之间的工作，我们想象两者之间有一个看不见的道路衔接，把这条衔接通路称为信道。信道具有一定的频率带宽，正如公路有一定的宽度一样。

无线信道中电波的传播不是单一路径，而是许多路径来的众多反射波的合成。由于电波通过各个路径的距离不同，因而各个路径来的反射波到达时间不同，也就是各信号的时延不同。当发送端发送一个极窄的脉冲信号时，移动台接收的信号由许多不同时延的脉冲组成，我们称为时延扩展。

同时由于各个路径来的反射波到达时间不同，相位也就不同。不同相位的多个信号在接收端迭加，有时迭加而加强（方向相同），有时迭加而减弱（方向相反）。这样，接收信号的幅度将急剧变化，即产生了快衰落。这种衰落是由多种路径引起的，所以称为多径衰落。

此外，接收信号除瞬时值出现快衰落之外，场强中值（平均值）也会出现缓慢变化。主要是由地区位置的改变以及气象条件变化造成的，以致电波的折射传播随时间变化而变化，多径传播到达固定接收点的信号的时延随之变化。这种由阴影效应和气象原因引起的信号变化，称为慢衰落。

而且，由于移动通信中移动台的移动性，如前所说那样，无线信道中还会有多普勒效应。在移动通信中，当移动台移向基站时，频率变高，远离基站时，频率变低。我们在移动通信中要充分考虑“多普勒效应”。虽然，由于日常生活中，我们移动速度的局限，不可能会带来十分大的频率偏移，但是这不可否认地会给移动通信带来影响，为了避免这种影响造成我们通信中的问题，我们不得不在技术上加以各种考虑。也加大了移动通信的复杂性。

综上所述，无线信道包括了电波的多径传播，时延扩展，衰落特性以及多普

勒效应，这些因素使得无线信道的传输环境比有线信道要恶劣许多。因此，在移动通信中，我们要充分考虑这些特性以及解决的方案。

提高信息传输的可靠性和有效性是通信系统一直追求的目标，随着现代通信的发展，人们对传输的可靠性和有效性的要求也越来越高。信道纠错码是提高信息传输可靠性的一种重要手段。卷积码作为一种有效的信道编码方式已经得到广泛的应用，它与维特比（Viterbi）译码算法共同实现了前向纠错，从而改进了在多噪声及衰落信道下译码的准确性，增强了数字通信系统的性能。

现代信息和编码理论的奠基人香农（Shannon）在 1948 年提出了著名的有噪信道编码定理，在定理中香农给出了在数字通信系统中实现可靠通信的方法以及在特定信道上实现可靠通信的信息传输速率上限。同时，该定理还给出了有效差错控制编码的存在性证明，从而促进了信道编码领域研究的快速发展。

卷积码是 Elias 等人在 1955 年提出的是一种非常有前途的编码方法，尤其是在其最大似然译码算法——维特比译码算法提出之后，卷积码在通信系统中得到了极为广泛的应用。其中约束长度 *K=7*，码率为 *1/2* 和 *1/3* 的 Odenwalder 卷积码已经成为商业卫星通信系统中的标准编码方法。在“航海家”以及“先驱者”等太空探测器上也都采用了卷积码作为其差错控制编码方法。在移动通信领域，GSM 采用约束长度 *K=5*，码率为 *1/2* 的卷积码;在 IS95-CDMA 中，上行链路中采用的是约束长度 *K=9*，码率为 *1/3* 的卷积码，在下行链路中采用的是约束长度 *K=9*，码率为 *1/2* 的卷积码。在第三代移动通信标准中也是以卷积码以及与卷码相关的编码方法作为差错控制编码方案，如 TDS-CDMA 和 WCDMA 标准都采用了 *K=9*，码率为 *1/2* 和 *1/3* 的卷积码。

Viterbi 算法是 1967 年由 Viterbi 提出的概率译码方法，它实质上就是最大似然译码，处理过程是对接受到的信息序列进行一系列的“相加——比较——选择”（Add Compare Select ACS）的操作，得到一条可能性最大的状态之间的转移路径，进而得到译码的结果。在译码算法的实现过程中，运算复杂度最高的部分就在于状态转移的 ACS 操作，而且这个复杂度会随着卷积码编码器的约束长度 K 的增加以指数增加，例如约束长度 *K=8* 的（2，1，9）卷积码，有 2 的 8 次方共 256 个状态，每次状态转换都要进行 256 次 ACS 操作。所以，在数据速率较高的情况下，需要通过并行执行 ACS 操作来实现 Viterbi 译码器的设计和实现。

1.1.2 研究意义

本论文实现的 Viterbi 译码器是一个 CDMA 通信系统的数字处理部分的一个模块。该系统的调制解调部分采用全数字化处理，所需的各种频率采用 DDS 产生。调制解调、同步、信道编译码、信息接口等部分的算法采用 FPGA 或 DSP

器件实现。在实现过程中，对各部分功能都先使用计算机上进行在线仿真、调试和修改，然后加载到芯片上进行实际测试。由于采用全数字化处理方式，设备生产和调测方便，设备性能稳定可靠。

基于系统的整体设计，本论文实现了一个适用于多种数据速率，能同时处理多路数据的Viterbi译码器，为了能够在FPGA芯片中同时实现系统的其他部分，对Viterbi译码器的资源消耗也维持在一定的限度之内。该Viterbi译码器的设计和实现达到了数据处理效率和资源消耗的平衡，可以应用于其他有较大数据吞吐量，支持多种速率和多路数据传输的通信系统。

1.2 论文的主要工作

本文的主要工作是在Xilinx的Virtex-4-SX35的FPGA芯片上实现了串并结合的（2，1，9）卷积码译码器，通过了仿真，综合，布局布线和测试，性能达到设计要求。

本文具体工作如下：

1) 分析Viterbi译码算法原理，研究了决定算法复杂度和译码性能的关键因素，结合项目要求确定Viterbi译码器的设计参数。

2) 深入研究了Viterbi译码器实现的关键技术，包括架构设计、路径度量值的归一化处理、ACS 运算单元的设计、路径度量值存储单元的实现、幸存路径选择输出等，在此基础上，综合考虑译码速度、译码延时、功耗、资源占用等各项要素给出了一个符合项目要求的设计。

3) 采用 Verilog HDL语言实现各个模块的RTL级代码，编写testbench对设计进行了功能仿真。

4) 将译码模块结合到整个中频调制解调单元中去，完成了译码器的片上测试，测试结果验证了设计的正确性。

本文中设计的Viterbi译码器采用乒乓操作和流水线技术等EDA设计方法，对部分模块进行了时序优化，通过整个设计流程，使作者掌握了现代化的EDA设计方法，为下一步的研究工作打下基础，本文取得的研究成果对于无线多媒体传输系统的基带处理芯片的研究和开发积累了重要的知识和经验，对以后的研究和开发具有重要的意义。

1.3 论文的章节安排

本文主要围绕串并结合的Viterbi译码器的设计和FPGA实现展开讨论，主

要内容和章节安排如下：

第一章，介绍了该项目的背景，研究目的和意义，以及本人在项目中主要承担的工作。

第二章，介绍了卷积码的编码原理，重点讨论了 Viterbi 译码算法原理和决定算法复杂度和译码性能的几个关键因素。

第三章，根据第二章的分析结果，确定了译码器的相关参数，设计了译码器电路的整体结构。针对设计中出现的难点提出了解决方法。

第四章，将第三章中的设计用 FPGA 芯片实现的流程。介绍了 FPGA 的设计流程技巧，接下来对各个模块的具体实现进行了详述，并给出了用 Modelsim 的仿真波形。

第五章，译码器的功能仿真和硬件测试过程，以及仿真和调试的一些技巧。并结合整个中频调制解调单元对译码器性能进行了分析评价。

第六章，对论文的主要工作进行了总结，并展望了下一步的工作内容。

第二章　卷积码和维特比译码算法

2.1　信道编码

信道编码是为了保证通信系统的传输可靠性，克服信道中的噪声和干扰而专门设计的一类抗干扰技术和方法。它根据一定的(监督)规律在待发送的信息码元中加入一些必要的（监督）码元，在接收端利用这些监督码元与信息码元之间的（监督）规律，发现和纠正差错，以提高信息码元传输的可靠性。称待发送的码元为信息码元，人为加入的多余码元称为监督码元（或校验码元）。信道编码的目的就是试图以最少的监督码元代价，以换取最大程度的可靠性的提高。

在数字通信中，对整个通信系统进行差错控制的方式主要有三种：前向纠错（FEC）、检错重发（ARQ）和混合纠错（HEC）。前向纠错也称自动纠错，发送端发送具有纠错性能的码，如果在传输过程中产生的错误介于该纠错码能纠正的类型，则此译码器不仅能检错，而且能够自动纠错；检错重发又称自动请求重发，接收端译码后，如发现传输有错误，则通知发送端重发接收端认为错误的消息，直到接收端认可为止;混合纠错是上述两种方式的混合。接收端对少量的接收差错自动纠正，而超出纠正能力的差错则通过自动请求重发的方法加以纠正[1]。

2.1.1　卷积码编码原理

按照对信息元处理的方法不同，纠错编码可以分为分组码和卷积码两大类。分组码是把信源输出的信息序列，以每 k 个码元分组，通过编码器把每组的 k 个信息元按一定规律产生 r 个多余码元(称为校验元或监督元)，输出长度为 $n=k+r$ 的一个码字（组）。因此每一码组的 r 个校验元仅与本组的信息元有关，而与别组无关。分组码用（n，k）表示，n 表示码长，k 表示信息位。卷积码是把信息源输出的信息序列，以每 k 个码元分为一段，通过编码器输出长为 n（$\geq k$）的一段码段。但是该码段的 n-k 个校验元不仅与本段的信息元有关，而且也与其前 m 段的信息元有关，称 m 为编码存贮，也称约束长度，因此卷积码用（n，k，m）表示[2]。

由于在卷积码的编码过程中，充分利用了各码段之间的相关性，在与分组码同样的码率和设备复杂性条件下，无论从理论上还是实际上都已经证明卷积码的性能要优于分组码，且实现最佳译码和准最佳译码也较分组码容易。所以在当今的信息学领域内，卷积码成为研究的热点，并在已经实现的通信系统中得到了广

泛的应用。

2.1.2 卷积码编码方法

卷积码的典型结构可看作由一个有 k 个输入端，n 个输出端，具有 m-1 节寄存器构成的一个有限状态机，或有记忆系统，也可以看作一个有记忆的时序网络。

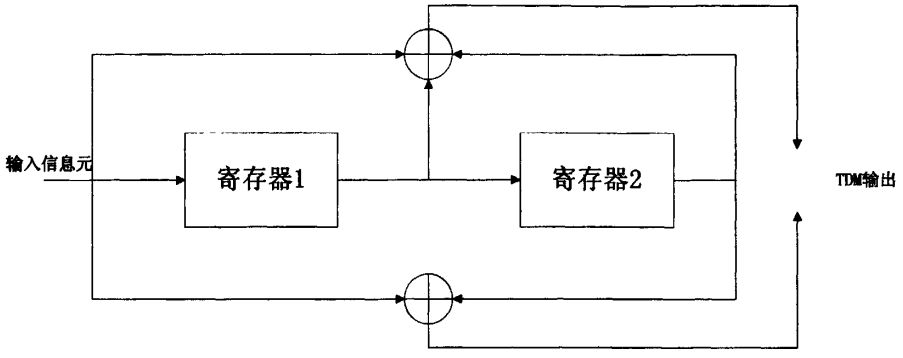


图 2-1 （2，1，3）卷积码编码原理图

上图是（2，1，3）卷积码的编码原理图，输入的信息元将经过两级移位寄存器，而两编码结果将是其中某些寄存器内容和当前输入信息元的和。两路编码数据时分复用输出，所以该（2，1，3）卷积码的码率为 1/2。

2.1.3 卷积码表示方法

卷积码的描述可以分为两大类型：解析法，它可以用数学公式直接表达，包括离散卷积法、生成矩阵法、码生成多项式法；图形法，包括状态图（最基本的图形表达形式）、树图及格图（或称为篱笆图）。

在解析法中，较为常用的主要是码生成多项式法。这种表示方法为编码器指定 n 个连接矢量集，每个矢量对应一个模 2 加法器，每个矢量都是 m-1 维，表示该模 2 加法器和编码移位寄存器之间的连接。矢量中第 i 位上的 1 表示移位寄存器相应级和模 2 加法器的连接，若是 0，则表示相应级与模 2 加法器之间无连接。图 2-1 中编码器，用连接矢量g^1代表上方连接，g^2代表下方的连接，如下以（2，1，3）卷积码为例，输入数据序列及其对应的多项式为：

$$U = (10111) \leftrightarrow U(x) = 1 + x^2 + x^3 + x^4 \tag{2-1}$$

$$g^1 = (111) \leftrightarrow g^1(x) = 1 + x + x^2 \tag{2-2}$$

$$g^2 = (101) \leftrightarrow g^2(x) = 1 + x^2 \tag{2-3}$$

输出的码组多项式为：

$$C^1(x) = U(x) \times g^1(x) = 1 + x + x^4 + x^6 \quad (2\text{-}4)$$

$$C^2(x) = U(x) \times g^2(x) = 1 + x^3 + x^5 + x^6 \quad (2\text{-}5)$$

采用连接矢量来表示卷积编码器相对于连接图表示方法比较的简洁，目前大多的标准规范都采用连接矢量来表示[1]。

在图形法中，较常用的是状态图法，这种表示法比较适合于描述译码，也是3种图形表示方法的基础。图2-2所示卷积编码器的输出由输入信息和编码器的状态（寄存器中的内容）决定，可以将它看成是一个有限状态机（Finite-State-Machine）的电路结构，因此可以用状态图来表示卷积码编码器。

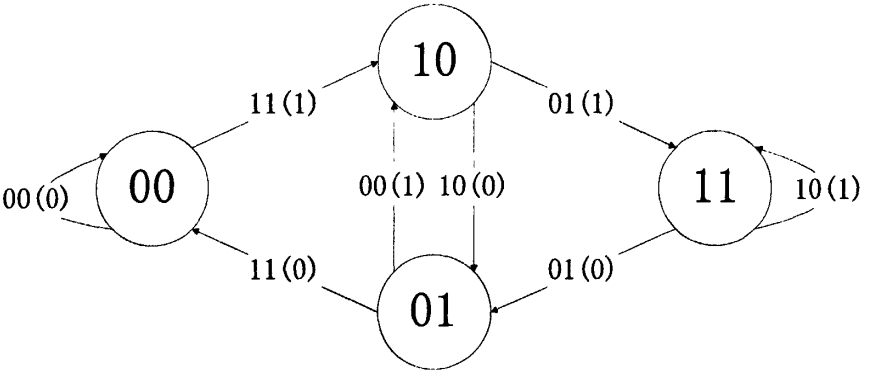


图2-2 （2，1，3）卷积码的状态转移图

图2-2中圆圈内码字表示编码器的状态，为编码寄存器最右端 *m-1* 级的内容，共有4种：00，10，01，11；状态间的路径表示由此状态转移时的输出分支。对应于两种可能的输入比特，从每一状态出发只有两种转移，状态转移时的输出分支字标注在相应的转移路径旁。括号中的0表示输入比特0时的路径，1表示输入比特1时的路径。编码器只在有限的几个状态之间转换，状态图表示方法明确的表示了当输入信息不同时状态的改变和与之对应的输出[1]。

2.1.4 卷积码的自由距离

卷积码的自由距离，是用来衡量所有可能码字序列对之间的距离的。其定义为：整个编码码树上，所有半无限长序列之间的最小汉明距离。由于卷积码的自由距离直接决定了它的纠错能力，所以寻找具有最大自由距离的卷积码是一项非常重要的工作。不过对于卷积码的构造，目前除了计算机搜索外还没有其它更好的方法。下表是由奥登沃尔德 （Odenwalder，1970 年）和拉森（Larsen，1973年）采用计算机搜索方法得到的固定码率和约束长度时具有最大自由距离的卷积码[3]。

表 2-1 卷积码的自由距离与编码增益

约束长度 K	生成多项式（八进制）	自由距离	编码增益 db
3	（5，7）	5	4.26
4	（15，17）	6	5.23
5	（23，35）	7	6.02
6	（53，75）	8	6.37
7	（133，171）	10	6.99
8	（247，371）	10	7.72
9	（561，753）	12	7.78

本文中实现的维特比译码算法针对的就是（2，1，9）卷积码。

2.2 维特比译码原理

卷积码有三种比较好的译码方法：

1) 1963 年由梅西（Massey）提出的门限译码，这是一种利用码代数结构的代数译码，类似于分组码中的大数逻辑译码；

2) 1961 年由沃曾克拉夫特（Wozeneraft）提出，1963 年由费诺（Fano）改进的序列译码，这是基于码树图结构上的一种准最佳的概率译码；

3) 1967 年由维特比（Viterbi）提出的 Viterbi 算法，这是基于码的网格图（trellis）基础上的一种最大似然译码算法，是一种最佳的概率译码算法。

在码的约束度较小时，Viterbi算法比序列译码算法效率更高、速度更快，译码器也较简单。因而从 Viterbi 算法提出以来，无论在理论上还是在实践上都得到了极其迅速的发展，并广泛应用于各种数字传输系统，特别是卫星通信系统中。

2.2.1 维特比译码准则

在数字与数据通信中，通信的可靠性度量一般是采用平均误码率P_e，由概率论可知，最小平均误码率等效于最大后验概率，即

$$min\ P_e = min \sum_Y P(Y)P(e/Y) = min \sum_Y P(Y)P(\hat{C} \neq C/Y) \qquad (2\text{-}6)$$

式中，$P(Y)$为接收信号序列的概率，它与具体译码方法无关；e 为差错序列；$\hat{C}$为接收端恢复的码组（字）；C 为发送的码组（字）。由贝叶斯公式，在信源等先验概率的条件下，最大后验概率准则与最大似然准则是等效的。即

$$P(e/Y) = \frac{P(C)P(Y/e)}{P(Y)} \qquad (2\text{-}7)$$

当$P(C)$为等概率分布时，有

$$max\ P(\hat{C} = C/Y) = max\ P(Y/\hat{C} = C) \qquad (2\text{-}8)$$

对于无记忆的二进制对称信道 BSC，最大似然准则又可以等效于最小汉明距离准则，即

$$max\ \ lgP(Y/\hat{C}=C)=min\ d(Y,C)=min\sum_{l=0}^{L-1}d(y_l,c_l) \tag{2-9}$$

在维特比译码中，硬判决中常采用最小汉明距离准则，而软判决中常采用最大似然准则[4]。

2.2.2 维特比算法

由卷积码的编码过程可以看出，码序列的个数是很大的。例如当码序列长度 $L=50$，$n=3$，$k=2$ 时，则共有$2^{kL}=2^{100}>10^{30}$个码序列。若 $m=5$，则 $L+m=55$。如果在一秒中内送出这 $kL=100$ 个信息元，则信息传输速率只有 100bps。即使在如此低的信息速率下，也要求译码器在一秒中计算、比较10^{30}个似然函数（或汉明距离、欧氏距离），这相当于要求译码器计算每一似然函数的时间小于10^{-30}秒，这根本无法实现。更何况通常情况下 L 是成百上千的。因此，有必要寻找新的最大似然译码算法。Viterbi 译码算法正是为了解决以上困难所引入的一种最大似然译码算法。由于并不是在网格图上一次比较所有可能的2^{kL}条路径，而是接收一段，比较一段，选一段最可能的译码分支，从而达到整个码序列是一个有最大似然函数的序列，其实现步骤的简单过程如下：

1) 从某一时间单位 $l=m$ 开始，计算进入每一状态的所有长为 l 段分支的部分路径度量，为每一状态，从转移到其中的所有可能路径中挑选并存储一条有最大度量的部分路径及其部分度量值，称此部分路径为相应状态下的可能路径。

2) l 增 1，把此时刻进入每一状态的所有分支度量，和同这些分支相连的前一时刻幸存路径的度量相加，得到了此时刻进入每一状态的可能路径度量值，在其中选取一个具有最大路径度量值的路径，并删去其他路径，从而得到了新的幸存路径，因此幸存路径延长了一个分支。

3) 若 $l<L+m$，则重复以上各部，否则停止，译码器得到了有最大路径度量的路径。由时间单位 m 至 L，卷积编码2^{kL}个状态中每一个有一条幸存路径，共有2^{km}条。但是在 L 时间单位（节点）后，网格图上的状态数目减少，留选路径也相应减少。最后到第 $L+m$ 单位时间，网格图回归到全为 0 的状态，因此仅剩下一条幸存路径。这条路径就是要找的具有最大似然函数的路径，也就是译码器输出的估值码序列$\hat{C}$。

由此可知，在网格图上用 Viterbi 译码算法找到的路径一定是一条最大似然路，因而这种译码方法是最佳的[4]。

2.3 维特比算法复杂度和译码性能

一般来说译码算法越复杂，译码过程中，数据越精确，获得的编码增益就越高，译码性能就越好，然而算法越复杂，数据越精确，所需的硬件资源就越多，译码速度就越慢。所以，就一个 Viterbi 译码器的具体实现来讲，必须在算法复杂度和译码性能之间做一个很好的平衡，下面从四个方面对这一问题进行阐述。

2.3.1 软判决译码

关于两电平（硬）判决与多电平（软）判决，两电平是非此即彼即非 0 即 1 的判决，所以称它为硬判决；而多电平则不属于非 0 即 1 的简单硬判决。软、硬判决所允许的归一化噪声、干扰水平是不一样的。电平级数越多，允许噪声和干扰越大判决性能越好，但是电平级数越多实现就越复杂并且资源占用就更多，一般取 4 或 8 电平即可。二者的主要差异有：

1) 信道模型不一样。硬判决采用二进制对称信道 BSC 模型，软判决采用离散无记忆信道模型即 DMC 模型。

2) 度量值与度量标准不一样，硬判决的度量值是汉明距离，度量准则是最小汉明距离准则，软判决的度量值是似然值，度量标准是最大似然准则。

软判决与硬判决相比，增加了一些复杂度，但是在性能上却比硬判决好 *1.5~2dB*，所以在实际译码中常采用软判决。在本设计中采用了带符号位 5 比特的软判决方式[1]。

2.3.2 译码蝶形图

Viterbi 译码算法的核心部分是对每个编码状态的路径度量值进行的加比选（ACS）运算。ACS 运算可以用蝶形图来描述，图 2-3 列出了基-2（Base-2）和基－4（Base-4）的译码蝶形图[3]。

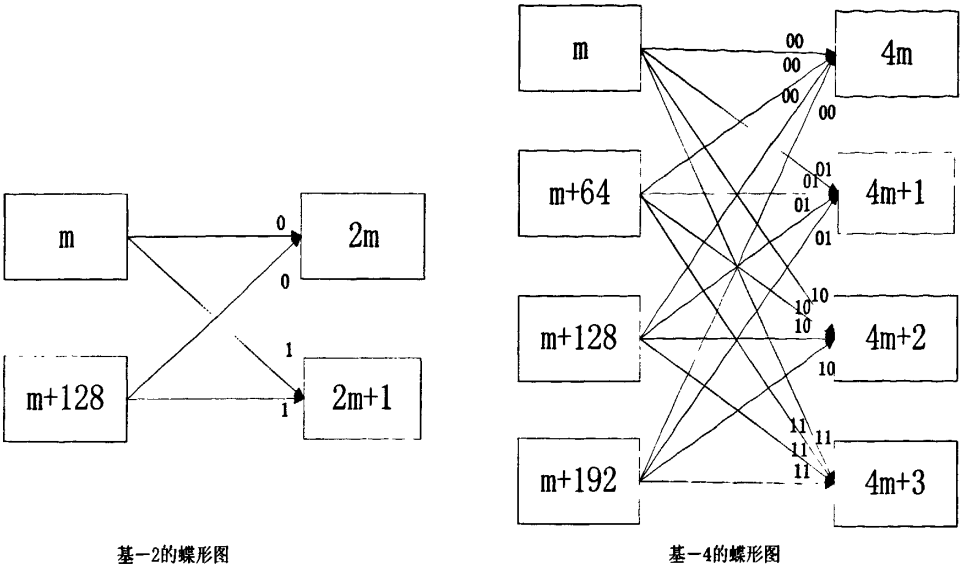


图 2-3 基-2 和基-4 的蝶形图

如图所示，在基-2 蝶形图中，每个当前状态都可能转变为 2 个新状态中的 1 个，ACS 运算就是通过对当前路径度量值，该状态所用可能的输出与读入的 2 个编码之后码字的差进行加比选操作，得出可能性最大的状态转移路径。

与基-2 的蝶形图不同，基-4 的蝶形图在每次的 ACS 运算中需要读入 4 个编码之后的码字，同时进行 4 个状态的加比选，获得 4 个新状态的路径度量值和 2 比特的回溯路径。

与基-2 的蝶形运算相比，尽管基-4 的蝶形运算需要更多的硬件资源，但是能够提供 2 倍的译码效率，有利于在有限的时钟频率下完成所需的运算量。为了在有限的时钟周期内完成（2，1，9）卷积码的 256 个编码状态的加比选运算，本文实现的译码器采用了 8 个并行的基-4 的译码蝶形图。

2.3.3 路径度量值位数的选择

在维特比译码算法中，路径度量值（即硬判决中的汉明距离，软判决中的似然值）是随着计算次数的增加而无界增长的。但是在硬件实现的过程中，所有的变量都需要被限定在一个有限的范围内。因此，在一般的维特比算法实现中，需要在每隔一定的时间进行一次归一化，即用将所有的路径度量值减去一个最小的路径度量值。这种操作需要额外的硬件和时序资源。实际上，有理论可以证明，幸存路径的选择只是和路径度量值的差值有关，而这个差值是有界的[5]。基于这个理论，只要采用 2 的补码表示软判决的数据，以此来计算路径度量值，就可以解决路径度量值的溢出。

2.3.4 截尾译码与译码深度的选择

理论上 Viterbi 译码算法要等到全部信息接收完毕后才能得到译码结果。若 L 很大，译码器的存储量太大而难以适用。在发送序列较长时，需要的存储量和译码延时都相当大，这显然不能满足实时通信的要求。所以从实用的角度来看，在 Viterbi 译码算法的具体实现中，通常会采用截尾译码的方式，即每个路径存储器不必存储长度为 L 的很长的码序列，在实际的译码算法实现中，考虑到译码器的存储量，一般在译码器开始处理第 $N+1$ 个码段时，就对所有路径信息缓冲区中的第一段信息元做出判决并输出，这种译码方法被称为 Viterbi 译码的截尾译码，其性能比非截尾的稍差，但是如果在 $N=(5\sim10)m$ 的情况下，则对译码错误概率影响很小[1]。

本译码模块的实现中，也采用了截尾译码，只是稍作了部分修改：在此 $N=2N_0$，N_0是上述意义上的译码输出的延迟码段，当译码器开始处理第 $N+1$ 个码段的同时，从 0 状态开始（从任何状态开始都可以），沿幸存路径回溯前 N 个码段，并且判决输出前N_0段信息元；再往后处理第$N+N_0+1$个码段的时候，同样沿最佳路径回溯前 N 个码段，并且判决输出前N_0段信息元。这种方法减少了复杂度，也比较节省回溯时间。

第三章　串并结合的维特比译码器设计

3.1　译码器总体设计

本论文中实现的 Viterbi 译码器能够同时处理 2 路并行的输入数据，其中第 1 路数据的速率可变，会根据需要以 64Kbps，128Kbps，256 Kbps ，512 Kbps ，1024 Kbps 和 2048Kbps 这 6 种数据速率中的一种来进行数据传输，在传输过程中也可以进行传输速率的切换。第 2 路数据速率是固定的 64Kbps。

输入 Viterbi 译码器的 2 路数据具有不同的帧结构，但是帧的长度都是 20ms，由同一个帧头信号来标示每帧的开始，在 64Kbps 的速率下，每帧只有 1 个子帧；128Kbps 速率下有 2 个子帧；256Kbps 速率下 1 帧分为 4 个子帧，以此类推，在 2048Kbps 的速率下 1 帧将被分为 32 个子帧。所以 Viterbi 译码器需要在 20ms 内完成对 2 帧数据的译码才能同时处理完 2 路数据。为了实现这个目标，本论文设计的 Viterbi 译码器将 2 路并行的数据分别存储相应的寄存器中，先完成第 1 路数据的译码，在第 1 路数据的最后一个子帧译码结束之后，再用该子帧剩余的时间完成第 2 路数据的译码。在第 1 路数据速率达到 2048Kbps 时达到译码器的最大吞吐量，即在 1 个 2048Kbps 的子帧之内需要完成第 1 路 1280 个比特加上第 2 路 1120 个比特数据的译码运算。

要使 Viterbi 译码器具有较高的吞吐量，可以使用较高的时钟频率去驱动的译码器 ACS 运算，或者并行更多的 ACS 处理单元，在有限的时钟频率下完成更多的 ACS 运算。前者需要对 Verilog HDL 的设计进行相关的优化，在较高的时钟频率下比较难以实现。后者需要较多的芯片资源，而且较高的资源消耗也会影响最后实现的时钟频率。

综合考虑以上的因素之后，本论文实现的 Viterbi 译码器采用了 65.536MHz 的时钟频率，并行执行 8 个 ACS 运算单元。在这个频率之下，输入给 ACS 单元的 4 个待译码字的持续时间为 32 个时钟周期，每个 ACS 单元在这 32 个时钟周期内比较 8 组，共 4x8＝32 个编码状态。因为并行了 8 个 ACS 单元，所以在 32 个时钟周期内就可以完成 8x32＝256 个状态的加比选。在最高的 2048Kbps 的数据速率下，每个编码之后的码字持续时间为 16 个时钟周期，每输入 4 个编码码字的时间为 4x16＝64 个时钟周期。所以，在最高的数据速率下，译码模块也只需要 1/2 个第 1 路数据的子帧时间完成该子帧的译码，剩下 1/2 的时间可以用作第 2 路数据的译码，从而在 1 帧（20ms）的时间内完成两路数据的译码运算。因为译码所需的时间是固定的，与数据速率无关，所以在第 1 路使用低数据速率

时，空闲的时间更多。

Viterbi 译码器的总体结构如下图所示：

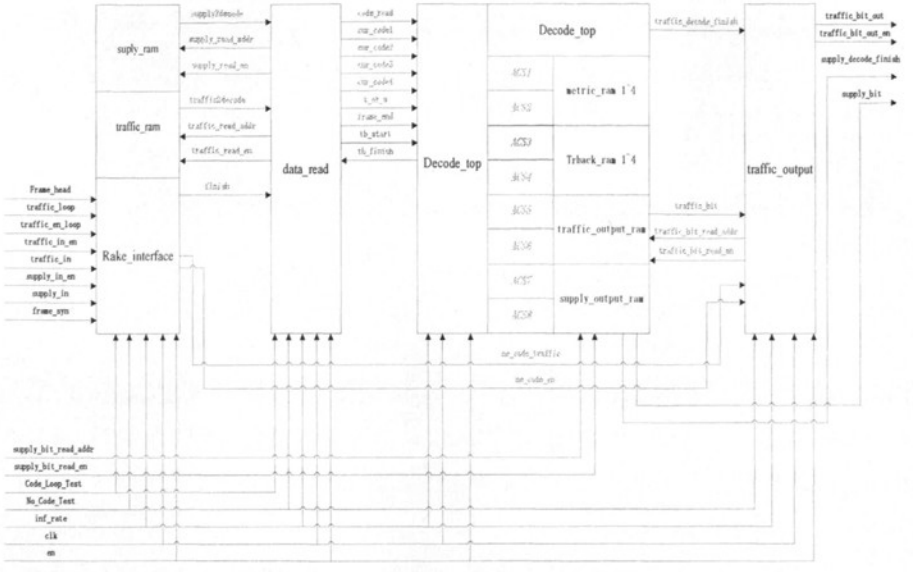


图 3-1　Viterbi 总体架构图

Viterbi 译码的总模块包括四个子模块：

1) 数据接口模块，外部待译码数据和 Viterbi 译码器之间的接口；

2) 数据读取模块，读取码字缓冲区内容的模块；

3) 译码处理模块，实现 Viterbi 译码算法的模块；

4) 数据输出模块，译码器输出译码比特流的接口。

图 3-1 描述了译码模块和其他模块的数据的交互，以及本模块内部各子模块之间的各数据和控制信号的产生交互。所有以黑色表示的信号都是译码模块与其他模块的交互信号。 译码处理模块是 Viterbi 译码模块的核心，实现真正的译码算法，它接收数据读取模块从数据接口模块中读取各种类型的码字，并行送入 8 个 ACS 单元，译出的信息比特放在各类型 DPRAM 中，数据输出模块读出译码结果，按不同速率送到各信号线上。

3.2　译码器子模块设计

3.2.1　数据接口模块

本模块将来两路待译码数据存入各自的码字缓冲区。为了实现寄存器的“乒乓操作”，子模块将分别存储两路数据的各 2（子）帧码字，第 1 路数据缓冲区

的首地址分别为 0 和 2560，第 2 路数据缓冲区的首地址分别为 0 和 2240。当第 1 路数据的 1 个子帧码字存储完成，就给读数据模块（数据读取）送缓冲区满 finish 信号，启动数据读取模块开始工作。

在存储码字时，数据缓冲区的写入地址按交织器的规律跳变，比如：对第 1 路数据，当缓冲区的写入地址大于 2495，就减去 2495；否则加上 64。在完成数据的存储之后，即可完成两路数据的解交织。

子模块收到帧头信号后，即转入工作状态。计数器开始计数，存储码字，1 帧存储完成以后，回到空闲状态，再检测帧头信号（标志着每一帧码字的开始）后，开始新 1 帧的存储。如果检测到失帧信号，子模块将被重置，等接收到新的帧头信号后重新开始存储。

为了采到稳定的码字，数据的存储并不是在信号使能的上升沿完成，而是 5~7 个时钟之后。

3.2.2 数据读取模块

本模块读取数据接口模块存储在码字缓冲区内的码字，按照不同的速率和数据类型进行插 0 或解重复，再送入译码模块进行译码。从收到第一个 finish 信号起，本子模块开始读取第 1 路数据缓冲区内的数据，每读出 160 个数据，就插入 1 个 0（5 比特 00000）。当读过 1 帧第 1 路数据的码字，就开始读第 2 路数据码字缓冲区内的数据，依照数据帧结构的不同，对码字做相应的相加求均值处理。读完 1 帧第 2 路数据码字之后，返回空闲状态，等待下一个 *finish* 信号。

因为本 Viterbi 译码模块中采用了基-4 的蝶形图，所以一次要向译码模块传送 4 个并行的 5 比特数据。这些数据还要经过一些相关的处理，不会同时就绪，在 4 路并行数据读取完成后，将产生 1 个 *code_read* 信号，送到译码处理模块，启动这 4 个数据的译码。为了给译码过程留有充足的时间，每次读出的 4 路并行码字将保持 32 个时钟周期。

为了在译码模块中启动回溯操作，在读取一定数目的数据之后，向译码模块送出 *tb_start* 信号，第 1 路为 160 个数据，第 2 路为 240 个。

3.2.3 译码处理模块

本模块并行 8 个 ACS 单元，完成各种速率的（2，1，9）卷积码译码。按照 Viterbi 译码算法，计算出最大似然的回溯路径，由此译出信息比特，存储到到各类型的信息缓冲区中。

译码状态之间的转换如下图：

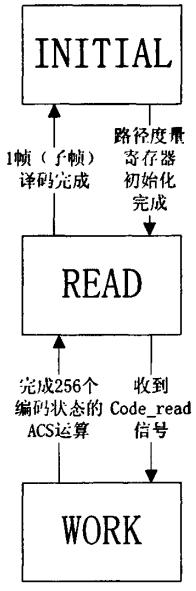


图 3-2 译码子模块状态机

1) INITIAL:

在 INITIAL 状态中，4 个路径度量值寄存器被赋初值，这个操作需要 8 个时钟周期，赋值结束后，即转入 READ 状态。

2) READ:

在 READ 状态中，程序等待数据读取模块的数据读取，当收到表示 4 路 5 比特数据读取完成的信号 *code_read* 后，即转入 WORK 状态进行译码操作。如果收到表示 1 帧（子帧）译码完成的信号 *frame_end*，即转入 INITIAL 状态，准备下 1 帧（子帧）的译码。

3) WORK:

在 WORK 状态中，程序将 4 路数据和相应的编码状态，路径度量值分别输入 8 个 ACS 单元，在 32 个时钟周期内，完成 256 个编码状态幸存路径和相应的路径度量值的计算，并将结果写入相应的寄存器。完成这一系列操做后，回到 READ 状态，等待下 4 路数据的到来。

在基-4 的蝶形图中，为了得到每个状态的幸存路径，需要进行 4 个候选度量值的比较。处理时将 4 个度量值分为两组，分别比较出每组中较小的值，再比较这 2 个值，等到最终结果。对两组候选度量值进行比较，选择两个较小值，假如两个度量值分别为 *m1*，*m2*，采用以下方法进行比较：

h1，*h2* 是 *m1* 和 *m2* 的符号位，*n1*，*n2* 是 *m1* 和 *m2* 除去符号位后的无符号数。

$$comp = ((n1 - n2) \geq 0)?0:1 \tag{3-1}$$

$$temp = comp \wedge h1 \wedge h2 \tag{3-2}$$

结果 *temp* 等于 1，则 *m1* 为较小值，否则 *m2* 为较小值，以此替代补码的减运算。

3.2.4 数据输出模块

本模块收到译码处理模块送来的 *traffic_decode_finish* 信号后，开始从存储译码结果的寄存器中读取译码比特，并且生成对应的数据使能信号，按照给定的数据速率输出。

3.3 路径度量值寄存器设计

对于路径度量值的位数，可以通过参考文献[5]中的公式来确定：

$$2^{c-1} - 1 \geq (m+2)B \tag{3-1}$$

其中，B 为分支度量绝对值得上界，m 等于卷积码的存储阶数，在本文实现的译码器中，使用 5 比特有符号数，因此 B=2x16=32，m=8，由此可以此估算度量值的位数 c 需选用 10 比特。

3.3.1 初始度量值的选择

对于（2，1，9）卷积码来说，共有 256 个状态，0 状态赋值 0，其它状态赋值初始值 inimetric，它的选择考虑到路径度量值模 2 运算中两个初始值的差最好在 1/2 的满量程之间，可以随意选，在本程序中选择了约 1/4 的差[14]。

3.3.2 路径度量值寄存器的结构

为了进行并行的 ACS 运算，需要同时读出和写入多个状态的路径度量值，程序将所有的 256 个状态每 8 个分成 1 组，将这 8 个 10 位的路径度量值并成一个 80 位的值，然后在 4 个位宽为 80，深度为 16 的路径度量值寄存器之间进行读写操作。寄存器的结构如下图：

表 3-1 路径度量值寄存器结构

	0	1	2	3	4	5	6	7
0	0	25	18	11	4	29	22	15
1	8	1	26	19	12	5	30	23
2	16	9	2	27	20	13	6	31
3	24	17	10	3	28	21	14	7

每 1 行表示 1 个 80 位寄存器，列是寄存器的深度，图中只列出前 0~7 列，

后 8 列和前 8 列一样，前后交替读写。图中 0~31 的数字表示编码的状态，状态组 0 表示状态（0，32，64，96，128，160，192，224），状态组 1 表示状态（1，33，65，97，129，161，193，225），以此类推。将 10 位的路径度量值组合称 80 位的数据加以存储，是为了方便一次性读取和写入多个数据，既提高了读写的效率，也减少了读写地址寄存器所需要的资源。

每次顺序读取 4 个寄存器的同一列，将读出的路径度量值分离提取，送给相应的 ACS 单元进行算，0~7 列（或 8~15 列）刚好使 8 个 ACS 单元完成 8 组加比选运算。ACS 单元得到幸存路径度量值后，再将其合并写回寄存器的 8~15 列（或 0~7 列）。

因为得到幸存路径的过程，也是状态转换的过程，得到已经不是读出状态的度量值，所以不能以读出的顺序写入，而是按照图中的颜色，同时写入不同寄存器。第 1 个写入状态组 0~3，第 2 个写入 4~7，第 3 个 8~11，第 4 个 12~15，第 5 个 16~19，第 6 个 20~23，第 7 个 24~27，第 8 个 28~31。

进行这样的操作，读取 8 个时钟周期，写入 8 个时钟周期（读写的操作的一部分时钟是重叠的），加上运算和操作的一些延时，在 32 个时钟周期内完全可以完成 256 个状态的 ACS 运算，时钟还有富裕。

3.4 回溯路径控制

本论文实现 Viterbi 译码器的回溯深度设为 64 个编码状态，输出前 32 个状态对应的译码结果。因为采用基-4 的蝶形图，每回溯一个状态会输出 2 比特的译码结果，所以对应基-2 的蝶形图，实际上是回溯 128 位，输出前 64 位。对应（2，1，9）卷积码，可以满足回溯深度的要求。

3.4.1 回溯路径寄存器结构

为了同时并行的写入 ACS 单元得到的幸存路径，程序构造了 4 个 16 位宽的回溯路径寄存器，将 8 个 ACS 单元每个时钟周期生成的 8x4=32 个 2 比特的回溯路径，8 个 1 组，合并成 4 个 16 位的值，分别写入 4 个寄存器。每个寄存器的地址是 10 位的，其中，高 7 位表示的是2^7=128 位的回溯路径存储深度，低 3 为代表存储的回溯路径的状态，结合 4 个回溯路径寄存器，即可以存储所有 256 个状态，深度为 128 的回溯路径。各组回溯路径在寄存器中的分布如下图：

表 3-2 回溯路径寄存器结构

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30

3	7	11	15	19	23	27	31

图中每1行代表1个不同的寄存器，第0组回溯路径表示状态（0，32，64，96，128，160，192，224）的组合，第1组表示（1，33，65，97，129，161，193，225）的组合，以此类推。每次写入不同寄存器的同一列。回溯读取的时候，根据回溯的深度和当前状态分别生成高7位和低3位的地址，合并成完整的地址，在不同的寄存器中读取相应状态的回溯路径。

3.4.2 回溯过程

Viterbi译码器的回溯处理的流程如下：

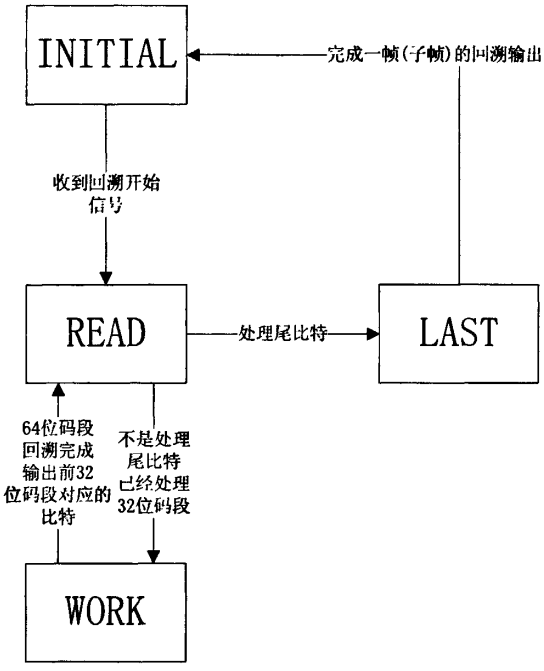


图 3-3 回溯过程的状态机

1) INITIAL:

回溯操作一开始处于INITIAL状态，收到*tb_start*信号后，转入READ状态。

2) READ:

在READ状态中，正常处理时，每当译码器处理完32位码段，即转入WORK状态进行回溯输出；如果收到*tb_lastbit*信号，则转入LAST状态做尾比特处理。

3) WORK:

在WORK状态中，从0状态开始回溯，以当前的回溯起点为高位地址，状态（0状态对应8'b00000000）的4~2位为低位地址（8位状态从高到低分别为76543210位），高低位地址合并得到回溯路径寄存器的读地址，再根据状态的低2位，选择0~3号寄存器中的1个，从相应的寄存器可以读出到达当前状态（最

开始是 0 状态）的幸存路径。

这是 1 个 2 比特的值，分别对应基-4 蝶形图的 4 条潜在路径中的 1 条，也即是当前状态的前一个状态的最高 2 位，将这 2 比特值作为 8 位新状态的最高 2 位（7~6 位），当前状态的 7~2 位作为新状态的 5~0 位，即可得到前一状态。而当前状态的最低 2 位也即是译码器的输出。

重复这个过程，即可以回溯 64 位的码段，并且将前 32 位的输出作为译码结果保存到相应的寄存器中。当完成 64 位的回溯后，操作将回到 READ 状态，等待下一次的回溯。

4) LAST:

在 LAST 状态中，处理尾比特的回溯。尾比特回溯也是从 0 状态开始，状态转换和结果输出都和 WORK 状态的处理相同。不同的是，在 LAST 状态中，因为可以确定起始的状态（也是 0 状态），所以不需要等到回溯到前 32 位再向结果寄存器输出，而应该从回溯开始即进行译码结果的输出。当完成 1 帧（子帧）的回溯之后，输出译码完成信号，转到 INITIAL 状态。

第四章　维特比译码器的FPGA实现

4.1　FPGA实现和硬件描述语言

4.1.1　FPGA实现

Viterbi译码器实现的两大主流是:基于DSP开发和基于ASIC技术开发。前者是在DSP上进行软件设计，具有算法设计灵活，升级方便，精度可编程控制等优点，但是存在处理速度慢，成本高，资源利用率低等缺点。基于ASIC技术开发的Viterbi译码器则可以很好地克服这些缺陷，过去由于硬件技术水平较低而很少采用，但随着微电子技术的发展，可编程器件性能的提高，ASIC实现方式越来越成为设计的主流，而且随着Viterbi译码算法的深入研究，其硬件复杂性问题得到越来越好地解决。在现代通信系统中，图象、语音、数据、视频的多种业务复用，数据的传输率越来越高，对系统的处理速度要求也越来越高，为了数据的实时传输，必须有高速处理信息的能力。采用DSP方式开发的Viterbi译码器越来越难以满足高速数据吞吐率的需求，而必须采用ASIC方式来实现[6]。

随着微电子技术的发展，系统设计师们更愿意自己设计专用集成电路（ASIC）芯片，而且希望ASIC的设计周期尽可能短，最好是在实验室里就能设计出合适的ASIC芯片，并且立即投入实际应用之中，因而出现了现场可编程逻辑器件（FPLD），其中应用最广泛的当属现场可编程门阵列（FPGA）和复杂可编程逻辑器件（CPLD）。

FPGA（现场可编程门阵列）是专用集成电路（ASIC）中集成度最高的一种，由可编程逻辑单元阵列、布线资源和可编程的I/O单元阵列构成，一个FPGA包含丰富的逻辑门、寄存器和I/O资源。一片FPGA芯片就可以实现数百片甚至更多个标准数字集成电路所实现的系统。FPGA的结构灵活，其逻辑单元、可编程内部连线和I/O单元都可以由用户编程，可以实现任何逻辑功能，满足各种设计需求。用户对FPGA的编程数据放在Flash芯片中，通过上电加载到FPGA中，对其进行初始化。也可在线对其编程，实现系统在线重构，其速度快，功耗低，通用性强，特别适用于复杂系统的设计。使用FPGA还可以实现动态配置、在线系统重构（可以在系统运行的不同时刻，按需要改变电路的功能，使系统具备多种空间相关或时间相关的任务）及硬件软化、软件硬化等功能[12]。

1985 年 Xilinx 公司推出第一片 FPGA 至今，FPGA 已经历了十几年的发展历史，占据了巨大的市场，逐渐取代了 ASIC，其原因在于 FPGA 不仅解决了电路系统小型化、低功耗、高可靠性等问题，而且其开发周期短、开发软件投入少、芯片价格不断降低，特别是对小批量、多品种的产品需求，使 FPGA 成为首选。电路设计人员使用 FPGA/CPLD 进行电路设计时，不需要具备专门的集成电路深层次的知识，随着现代 EDA 技术的发展，借助高性能 EDA 软件来辅助设计，可以使设计人员更能集中精力进行电路设计，快速将产品推向市场。随着半导体亚微米技术的发展，FPGA 的芯片密度已经达到了百万门级甚至千万门级，它的设计越来越接近于 ASIC 的设计，价格也越来越接近 ASIC，因此 FPGA 也被称为可编程的 ASIC。在某些应用领域已经出现了 FPGA 取代 ASIC 的趋势，它们之间的互相竞争也进一步推动了半导体技术的发展。本文中的 Viterbi 译码模块就是在 Xilinx 公司 Virtex-4-SX35 的芯片上实现的[20]。

4.1.2 硬件描述语言

几十年前，当时所做的复杂数字逻辑电路及系统设计规模比较小也比较简单，其中所用到的 FPGA 或 ASIC 设计工作往往只能采用厂家提供的专用电路图输入工具来进行。为了满足设计性能指标，往往需要花费好几天或更长的时间进行艰苦的手工布局布线。设计人员还得非常熟悉所选择器件的内部结构和外部引线特点，才能达到设计要求。这种低水平的设计方法大大延长了设计的周期。

现代的 EDA 技术普遍使用硬件描述语言来完成 FPGA 的设计。HDL（Hardware Description Language），顾名思义，就是指对硬件电路进行行为描述、寄存器传输描述或者结构化描述的一种新兴语言[6]。

主流的 HDL 分为 VHDL 和 Verilog HDL。VHDL 诞生于 1982 年。在 1987 年底，VHDL 被 IEEE 和美国国防部确认为标准硬件描述语言。自 IEEE 公布了 VHDL 的标准版本，IEEE- 1076（简称 87 版）之后，各 EDA 公司相继推出了自己的 VHDL 设计环境，或宣布自己的设计工具可以和 VHDL 接口。此后 VHDL 在电子设计领域得到了广泛的接受，并逐步取代了原有的非标准的硬件描述语言。

Verilog HDL 是由 GDA（Gateway Design Automation）公司的 PhilMoorby 在 1983 年末首创的，最初只设计了一个仿真与验证工具，之后又陆续开发了相关的故障模拟与时序分析工具。1985 年 Moorby 推出它的第三个商用仿真器 Verilog-XL,获得了巨大的成功，从而使得 Verilog HDL 迅速得到推广应用。1989 年 CADENCE 公司收购了 GDA 公司，使得 Verilog HDL 成为了该公司的独家专利。1990 年 CADENCE 公司公开发表了 Verilog HDL,并成立 LVI 组织以促进 Verilog HDL 成为 IEEE 标准，即 IEEE Standard 1364-1995。

由于GDA公司本就偏重于硬件，所以不可避免地Verilog HDL就偏重于硬件一些，故Verilog HDL的底层统合做得非常好。而VHDL的逻辑综合就较之Verilog HDL要出色一些。所以，Verilog HDL作重强调集成电路的综合，而VHDL强调于组合逻辑的综合

目前在我国广泛应用的硬件描述语言主要有：ABEL语言、AHDL语言、Verilog语言、和VHDL语言，其中Verilog语言和VHDL语言最为流行，本论文的算法实现使用的就是Verilog HDL。

采用Verilog输入设计法时，由于Verilog HDL的标准化，可以很容易的把完成的设计移植到不同厂家的不同芯片中去，并在不同规模应用时可以较容易的做修改。这不尽是因为用Verilog HDL所完成的设计，它的信号位数是很容易改变的，可以很容易的对它进行修改，来适应不同规模的应用；在仿真验证时，仿真测试矢量还可以用同一种描述语言来完成，而且还因为采用Verilog HDL综合器生成的数字逻辑是一种标准的电子设计互换格式（EDIF）文件，独立于所采用的实现工艺。有关工艺参数的描述可以通过Verilog HDL提供的属性包括进去，然后利用不同厂家的布局布线工具，在不同工艺的芯片上实现[19]。

采用Verilog输入法最大的优点是其与工艺的无关性。这使得工程师在功能设计、逻辑验证阶段，可以不必过多考虑门级及工艺实现的具体细节，只需要利用系统设计时对芯片的要求，施加不同的约束条件，即可以设计出实际电路。实际上这是利用了计算机的巨大能力并在EDA工具的帮助下，把逻辑验证与具体工艺库匹配、布线及时延计算分成不同的阶段来实现，从而减轻了人们的繁琐劳动。

4.2 FPGA设计流程

现代集成电路制造工艺技术的改进，使得在一个芯片上集成数十万乃至数千万个器件成为可能，但是很难想象仅由一个设计师独立设计如此大规模的电路而不出现错误。因此，在设计过程中，引入了自顶向下（Top-Fown）的设计概念。自顶向下的设计是从系统级开始，把系统划分为若干个基本单元，然后再把每个基本单元划分为下一层次的基本单元，一直这样做下去，直到可以直接用EDA元件库中的基本元件来实现为止[19]。

完成了层次模块的设计之后，FPGA设计流程大致分为文件设计、功能仿真、综合、布局布线、优化和布局布线，布线后仿真、FPGA配置下载成等一系列步骤。具体的FPGA设计流程如下图所示：

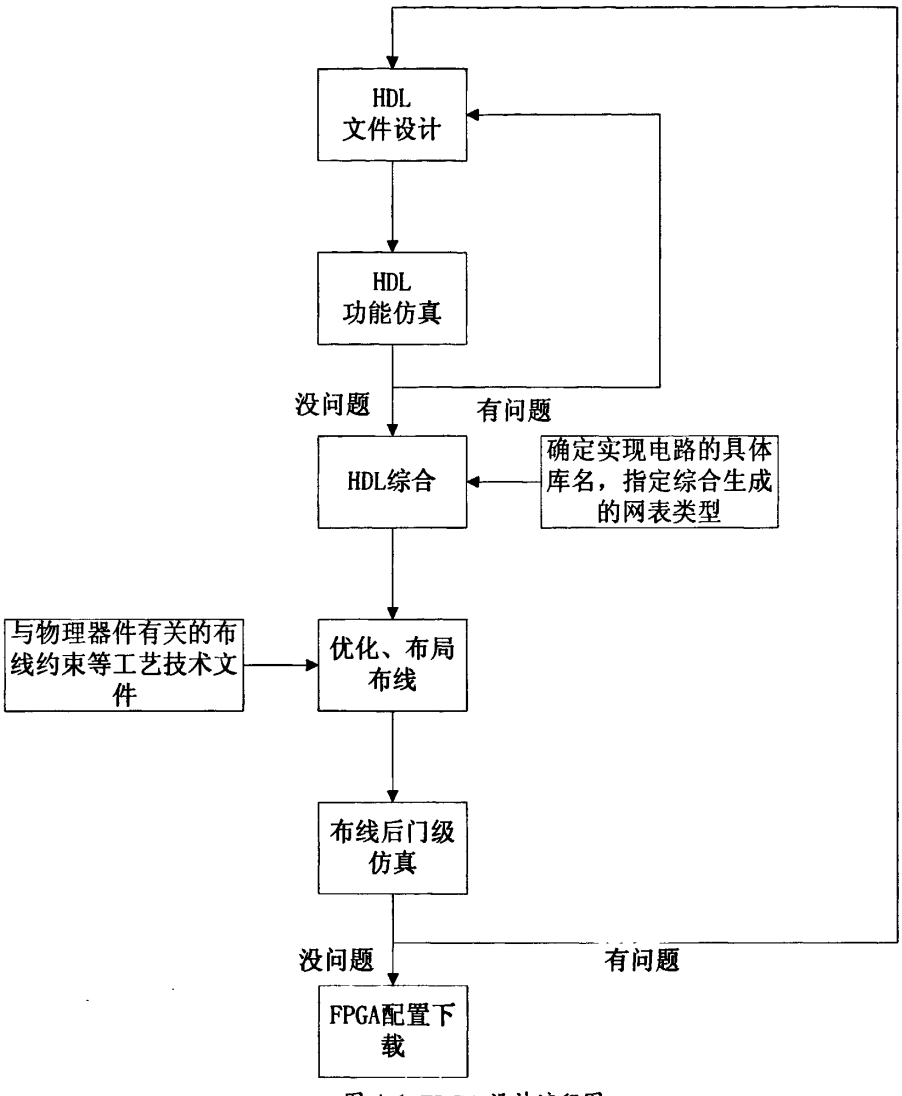


图 4-1 FPGA 设计流程图

1) HDL 文件设计：

文件设计是指将模块设计阶段定义好的模块借助于一定的设计输入手段转换为 EDA 工具接受的信息格式。目前主要的设计输入手段有：高级硬件描述语言 HDL(包括 VHDL 和 Verilog HDL)和原理图。HDL 语言支持不同层次的描述，不依赖于 FPGA 厂家的工艺器件，便于修改。它可以用任意的文本编辑器作为输入平台，在状态机、控制逻辑、总线功能方面较强。原理图输入法具有图形化强、直观等特点，但是不适合描述复杂的逻辑。在本设计中主要采用了 Verilog HDL 作为设计输入的主要工具语言。

2) HDL 功能仿真：

功能仿真就是在仿真器上通过模拟实际电路的工作环境来对设计进行验证。

功能仿真需要把测试激励数据送入 RTL 代码模型，把 RTL 模型的输出通标准的输出结果进行比较，从而验证 RTL 模型的正确性。在该设计中采用了 Mentor 公司的 Modelsim 仿真工具。

3) HDL 综合：

综合实际上是根据设计功能和实现该设计的约束条件（如面积，速度，功耗等），将涉及描述文件变换成满足要求的电路设计方案，该方案必须满足预期的功能和约束条件，对于综合来说，满足要求的方案可能有多个，综合器将产生一个最优的或接近最优的结果。因此，综合的过程也就是设计目标的优化过程，最终获得结果与综合器的性能有关。这个阶段产生的网表中包含了目标器件的逻辑原件和互联的信息，供后续的静态时序分析和布局布线使用。FPGA 的综合工具有很多例如 Synplicity 公司的 Synplify，Synopsys 公司的 Compiler IIFPGA 等，在本设计中采用了 Xilinx ISE 9.2 中自带的综合工具。

4) 布局布线：

这一步骤是要完成实现方案(网表)到实际目标期间的变换。根据设计中指定的约束条件、目标期间的结构资源和工艺特征，将电路方案中的逻辑元件分解布局，用作拓扑器件的连线资源，实现布线连接，在布局布线时序信息产生反标注文件（.sdf），给后续的时序仿真使用，同时还产生 FPGA 配置时所需要的位流文件。FPGA 设计中的布局布线工具主要由 FPGA 厂商提供，在本设计中采用了 Xilinx ISE 9.2 集成环境中自带的布局布线工具。

5) 布线后仿真：

布线后仿真是验证设计结果是否符合设计要求的重要流程，和功能仿真侧重于验证程序的逻辑正确性不同，它的主要目的是分析综合出来的电路本身是否违反设计规则。

6) 配置下载：

配置下载是在功能仿真与时序仿真正确的前提下，将布局布线后形成的位流文件通过下载工具下载到具体的 FPGA 芯片中，这个过程也叫 FPGA 编程。将位流文件下载到 FPGA 器件内部后，就可以将 FPGA 和其他的芯片构成的系统进行物理测试，当得到正确的测试结果后就证明了设计的正确性。在该设计中采用了 Xilinx ISE 9.2 集成环境中自带的配置下载工具。

FPGA 的设计流程也是一个类似软件开发的迭代过程，如果在仿真和验证中发现设计和实现上的问题，就需要回到产生问题的设计阶段去作修改，直到问题解决为止。

4.3 维特比译码器设计方法

4.3.1 芯片资源和速度的平衡

芯片资源指的是设计中消耗的 FPGA 的逻辑资源数量，例如触发器（FF），查找表（LUT），块存储器（BlockRam）等。速度是指设计在芯片上稳定运行所能达到的最高频率，这个频率由设计的时序状况决定。芯片资源和速度是设计质量评价的终极标准。同时，二者又是对立统一的矛盾体，要求一个设计同时具备占用资源最少，运行速度最高是不现实的。科学设计的目标是应该在满足时序要求的前提下占用最少的资源。因为占用资源越少在单芯片上实现的功能模块就越多，整个系统需要的芯片数量就越少，产品的成本才能降低。芯片资源和速度之间是可以转化的，一个设计如果时序余量较大，所能跑得频率远远高于设计要求，那么就能够通过功能模块复用减少占用的芯片资源；反之如果一个设计的时序要求很高，普通方法达不到设计频率，那么一般可以通过将数据流进行串并转换，复制到各个运算模块，在输出数据时在对数据进行“串并转换”，从而达到设计的目标。

4.3.2 寄存器的乒乓操作

为了在接收到输入数据之后，能够持续不断的进行译码操作，本论文实现的译码器中，所有的寄存器都使用了 DPRAM，以此来实现数据流的乒乓操作。乒乓操作是一个常常应用于数据流控制的处理技巧，典型的乒乓操作方法下图所示[6]：

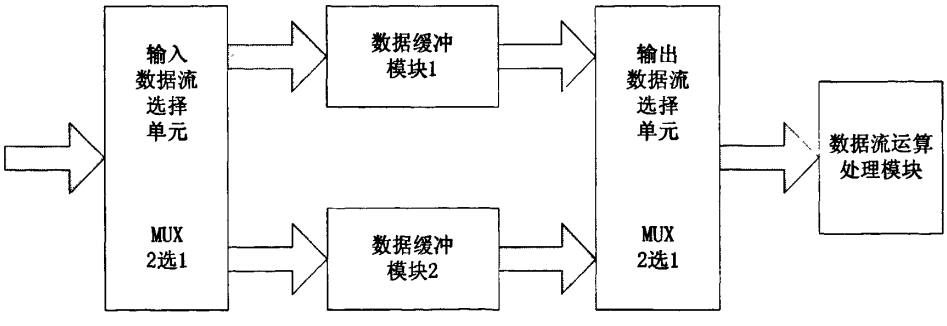


图 4-2 乒乓操作示意图

乒乓操作的处理流程为：输入数据流通过“输入数据选择单元”将数据流等时

分配到两个数据缓冲区，数据缓冲模块可以为任何存储模块，比较常用的存储单元为双口 RAM（DPRAM）、单口 RAM（SPRAM）、FIFO 等。在第一个缓冲周期，将输入的数据流缓存到“数据缓冲模块 1”；在第 2 个缓冲周期，通过“输入数据选择单元”的切换，将输入的数据流缓存到“数据缓冲模块 2”，同时将“数据缓冲模块 1”缓存的第 1 个周期数据通过“输出数据选择单元”的选择，送到“数据流运算处理模块”进行运算处理；在第 3 个缓冲周期通过“输入数据选择单元”的再次切换，将输入的数据流缓存到“数据缓冲模块 1”，同时将“数据缓冲模块 2”缓存的第 2 个周期的数据通过“输出数据选择单元”切换，送到“数据流运算处理模块”进行运算处理。如此循环。

乒乓操作的最大特点是通过“输入数据选择单元”和“输出数据选择单元”按节拍、相互配合的切换，将经过缓冲的数据流没有停顿地送到“数据流运算处理模块”进行运算与处理。把乒乓操作模块当作一个整体，站在这个模块的两端看数据，输入数据流和输出数据流都是连续不断的，没有任何停顿，因此非常适合对数据流进行流水线式处理。所以乒乓操作常常应用于流水线式算法，完成数据的无缝缓冲与处理。

乒乓操作的第二个优点是可以节约缓冲区空间。比如在 WCDMA 基带应用中，1 个帧是由 15 个时隙组成的，有时需要将 1 整帧的数据延时一个时隙后处理，比较直接的办法是将这帧数据缓存起来，然后延时 1 个时隙进行处理。这时缓冲区的长度是 1 整帧数据长，假设数据速率是 3.84Mbps，1 帧长 10ms，则此时需要缓冲区长度是 38400 位。如果采用乒乓操作，只需定义两个能缓冲 1 个时隙数据的 RAM（单口 RAM 即可）。当向一块 RAM 写数据的时候，从另一块 RAM 读数据，然后送到处理单元处理，此时每块 RAM 的容量仅需 2560 位即可，2 块 RAM 加起来也只有 5120 位的容量。

本论文实现的 Viterbi 译码器所采用的数据寄存器的深度为 2（子）帧待译码数据，当前 1（子）帧数据存储完成之后，即对该（子）帧数据进行译码，同时将之后的 1（子）帧数据写入寄存器的空余部分。这样，再完成前 1（子）帧数据的译码完成之后，就可以持续不断的进行译码运算。

4.3.3 ACS 单元的流水线处理

ACS 单元是真个 Viterbi 译码模块的核心运算部分，该部分的运算速度决定了整个译码模块的整体性能。在本设计中，译码器的 ACS（加比选）操作充分使用了流水线技术，将“加，比，选”三个步骤分开实现，这样的设计对系统的关键路径进行了时序优化，显著的提高了系统时钟的运行频率。

流水线处理是高速设计中的一个常用设计手段。如果某个设计的处理流程分

为若干步骤，而且整个数据处理是“单流向”的，即没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入，则可以考虑采用流水线设计方法来提高系统的工作频率。流水线技术和 CPU 中采用的流水线技术有所区别，CPU 中的流水线技术是指在一个时钟周期内同时处理多条指令的指令处理方法，而这里讲的流水线技术是数字电路设计中的时序优化技术，从本质上来说就是把在一个时钟周期内执行的操作分成几步较小的操作，并在多个较高速的时钟内完成[19]。

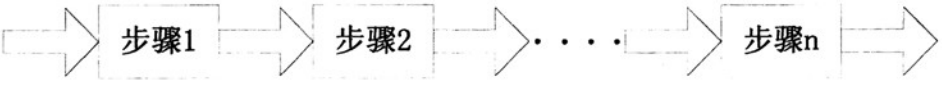


图 4-3 流水线结构示意图

流水线设计的结构示意图如图 4-3 所示。其基本结构为：将适当划分的 n 个操作步骤单流向串联起来。流水线操作的最大特点和要求是，数据流在各个步骤的处理从时间上看是连续的，如果将每个操作步骤简化假设为通过一个 D 触发器（就是用寄存器打一个节拍），那么流水线操作就类似一个移位寄存器组，数据流依次流经 D 触发器，完成每个步骤的操作。流水线设计时序如图 4-4 所示。

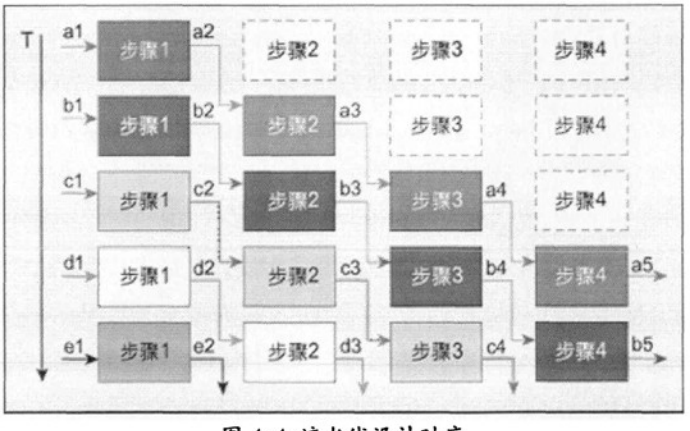


图 4-4 流水线设计时序

流水线设计的一个关键在于整个设计时序的合理安排，要求每个操作步骤的划分合理。如果前级操作时间恰好等于后级的操作时间，设计最为简单，前级的输出直接汇入后级的输入即可；如果前级操作时间大于后级的操作时间，则需要对前级的输出数据适当缓存才能汇入到后级输入端；如果前级操作时间恰好小于后级的操作时 间，则必须通过复制逻辑，将数据流分流，或者在前级对数据采用存储、后处理方式，否则会造成后级数据溢出。

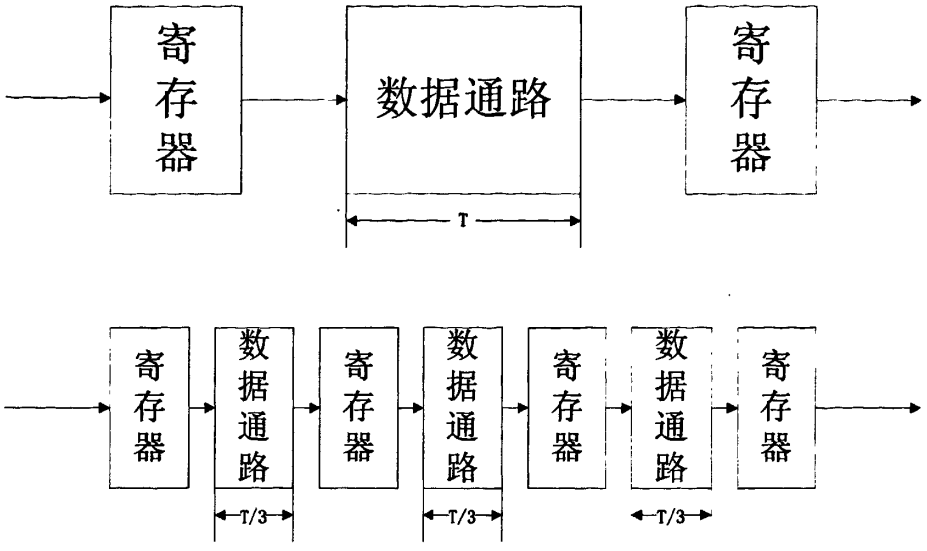


图 4-5 流水线的实现

对于一个组合时序逻辑，如果它的数据通路的延时为 *T*，则该电路不考虑寄存器的影响，从输入到输出的最高时钟频率就为 *1/T*。而在图 4-5 中，假设在理想情况下所分成的 3 级，每级的延时为 *3/T*，则该电路从输入到输出的最高频率可提高到原来的 3 倍，采用流水线技术有效地提高了系统的时钟频率，因而在多个时钟周期连续工作情况下，就提高了整个系统的数据处理量。在图 4-5 中，不使用流水线设计，数据通路的输出在一个时钟周期后被锁存在输出寄存器中。插入了流水线后，数据通路的输出需要三个时钟周期才能被输出寄存器锁存。所以采用流水线技术的缺点在于消耗了寄存器资源和增加了数据输出的延时。不过，这和它对整个系统时钟运行频率的增益相比较往往是微不足道的。

使用了流水线的处理方式之后，本来需要 3 个时钟周期的 ACS 运算只需要 1 个时钟周期就可以得到结果，大大提高了算法的处理速度，使得 Viterbi 译码器在有限的时钟频率下可以达到足够的数据吞吐量。

4.4 译码器子模块的 FPGA 实现

4.4.1 数据接口模块

数据接口模块接收 2 路 5 比特位宽的待译码数据，将之存到各自的寄存器中，并完成解交织操作。表 4-1 列出了模块的主要接口。

表 4-1 数据接口模块端口列表

端口名称	端口位宽	端口类型	端口描述
clk	1	input	65.536MHz 工作时钟
en	1	input	复位信号，高有效
inf_rate	3	input	3 位速率指示信号，3'b010 到 3'b111 分别表示 64kps 到 2048kps
frame_syn	1	input	帧头指示
traffic_in_en	1	input	第 1 路数据使能
supply_in_en	1	input	第 2 路数据使能
traffic_in	5	input	第 1 路数据
supply_in	5	input	第 2 路数据
traffic_read_addr	13	input	来自数据读取模块的第 1 路数据寄存器读地址
supply_read_addr	13	input	来自数据读取模块的第 2 路数据寄存器读地址
traffic_read_en	1	input	来自数据读取模块的第 1 路数据寄存器读使能
supply_read_en	1	input	来自数据读取模块的第 2 路数据寄存器读使能
traffic_out;	5	output	从第 1 路数据寄存器读出的数据，送往数据读取模块
supply_out	5	output	从第 2 路数据寄存器读出的数据，送往数据读取模块
finish	1	output	1（子）帧数据存储完成指示

数据接口模块写完 1 个子帧的数据后，即用 *finish* 信号通知数据读取模块进行读取。因为寄存器是使用了乒乓操作设计的 DPRAM，所以在读取的同时，还可以写入新的数据，这样就可以持续不断的进行数据的读写操作。通过寄存器写地址的转换，解交织在写入寄存器的同时完成，不需要额外的操作。数据写入过程的状态机如下图所示：

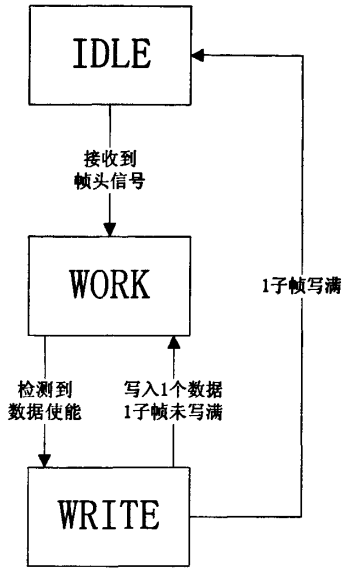


图 4-6 数据接口模块数据写入流程图

4.4.2 数据读取模块

数据读取模块读取数据接口模块存储在寄存器中的数据，做预处理之后 4 路并行数据译码处理模块。因为 2 路数据的帧长度不同，所以需要标示当前读出的是哪一路数据。表 4-2 列出了模块的主要接口。

表 4-2 数据读取模块端口列表

端口名称	端口位宽	端口类型	端口描述
clk	1	input	65.536MHz 工作时钟
en	1	input	复位信号，高有效
inf_rate	3	input	速率指示信号
frame_full	1	input	1（子）帧数据存储完成指示，来自数据接口模块
traffic_in	5	input	第 1 路数据，来自数据接口模块
supply_in	5	input	第 2 路数据，来自数据接口模块
tb_finish	1	input	1 子帧译码完成信号，来自译码处理模块
traffic_read_addr	13	output	送往数据接口模块的第 1 路数据寄存器读地址

supply_read_addr	13	output	送往数据接口模块的第 2 路数据寄存器读地址
traffic_read_en	1	output	送往数据接口模块的第 1 路数据寄存器读使能
supply_read_en	1	output	送往数据接口模块的第 2 路数据寄存器读使能
cur_code1	5	output	并行输出的待译码数据，送往译码处理模块
cur_code2	5	output	并行输出的待译码数据，送往译码处理模块
cur_code3	5	output	并行输出的待译码数据，送往译码处理模块
cur_code4	5	output	并行输出的待译码数据，送往译码处理模块
code_read	1	output	4 路并行数据读取完成指示，送往译码处理模块
frame_end	1	output	读完 1 子帧指示信号，送往译码处理模块
tb_start	1	output	回溯开始指示信号，送往译码处理模块
t_or_s	1	output	数据类型指示，‘0’表示送出的是第 1 路数据，‘1’表示送出的是第 2 路

数据读取模块在收到来自数据接口模块传来的*finish* 信号后，即开始读取数据寄存器中存储的数据。首先读取的是业务寄存器的数据，每读完 1 个子帧，则子帧计数器 SF_counter 加 1；当 1 帧业务数据中的所有数据子帧全部读完，则将 SF_counter 置 0，等检测到译码完成信号 *tb_finish*，开始读取补充数据寄存器中的数据，并同时将数据类型指示 *t_or_s* 置为 1；当补充数据帧读完后，等检测到译码完成信号 *tb_finish*，再从新读新的业务数据帧。数据读出的流程图如图 4-7

所示。

在读取数据的过程中，对业务和补充数据要做相应的处理。具体的，对于业务数据，用计数器 traffic_inter_counter 来记录读取数据的个数，每读取 160 个数据，即在第 161 个数据处插入‘0’，实现方法是读第 160 个数据的地址 2 次，并在第 2 次读取时用‘0’给相应的输出寄存器赋值。对于补充数据，需要把 1 帧中重复的数据相加并求均值。第 1 个数据先赋值给相应的寄存器，之后读到的重复数据都加到这个寄存器中，读完所有重复数据后，再将寄存器的内容截位输出。

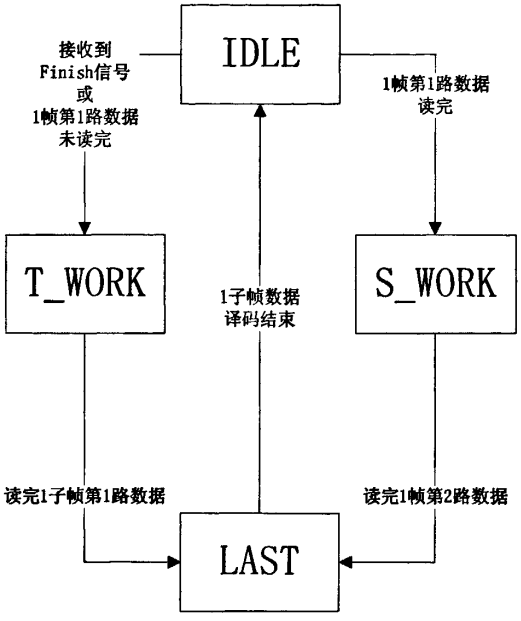


图 4-7 数据读取模块数据读出流程图

4.4.3 ACS 子模块

ACS 子模块是译码处理模块的子模块，也就是完成 ACS 运算的单元模块。译码处理模块中共有 8 个并行的 ACS 子模块，也就是并行了 8 个 ACS 单元。ACS 子模块根据输入的状态和待译码数据进行 ACS 运算，与译码之外的数据处理过程无关，可以灵活的适用于各种速率的维特比译码运算，可移植性很高。表 4-3 列出了模块的主要接口。

表 4-3 ACS 子模块端口列表

端口名称	端口位宽	端口类型	端口描述
clk	1	input	65.536MHz工作时钟
cur_state	8	input	ACS 的起始状态
cur_code1	5	input	输入的待译码数据

cur_code2	5	input	输入的待译码数据
cur_code3	5	input	输入的待译码数据
cur_code4	5	input	输入的待译码数据
metric1	10	input	输入的各状态的路径度量值
metric2	10	input	输入的各状态的路径度量值
metric3	10	input	输入的各状态的路径度量值
metric4	10	input	输入的各状态的路径度量值
survival_metric1	10	output	输出的各状态的幸存路径度量值
survival_metric2	10	output	输出的各状态的幸存路径度量值
survival_metric3	10	output	输出的各状态的幸存路径度量值
survival_metric4	10	output	输出的各状态的幸存路径度量值
tbdata1	2	output	输出的各状态的幸存路径度
tbdata2	2	output	输出的各状态的幸存路径度
tbdata3	2	output	输出的各状态的幸存路径度
tbdata4	2	output	输出的各状态的幸存路径度

ACS 子模块采取了流水线的设计方法。在 ACS 子模块进行运算的每个时钟周期中，首先根据输入的起始状态生成 4 个待运算的状态，分别是 cur_state、cur_state+64、cur_state+128、cur_state+192，这和译码器的基-4 的蝶形图相对应，分别计算这 4 各状态下所有可能的 16 中输出。

然后，对应这 4 个状态和它们各自的路径度量值 metric1、metric2、metric3、metric4 以及当前的待译码数据 cur_code1、cur_code2、cur_code3、cur_code4，分别计算不同的 4 中译码结果所对应的 16 个候选路径度量值。

最后，将对应着每个输出状态的 4 个候选路径度量值进行比较，保留最小的作为该状态的幸存路径度量，同时输出该状态的幸存路径。

8 个 ACS 子模块在译码处理模块中路径度量寄存器和回溯路径寄存器的配

合下，可以在 8 个时钟周期内完成（2，1，9）卷积码的 256 个状态的 ACS 操作。但是应为寄存器的写入和读出之间有时差，而且待译码数据的读取需要相应的时延，所以，对应每 4 个待译码数据的译码时间定为 32 个时钟周期。

4.4.4 译码处理模块

译码处理模块接收来自数据读取模块的 4 路并行的待译码数据，配合对路径度量值寄存器的操作，将待译码数据和相应的编码状态分别送入 8 个 ACS 子子模块，在 32 个时钟周期内完成 256 个编码状态的 ACS 运算，即每个 ACS 单元完成 8 次运算。表 4-4 列出了模块的主要接口。

表 4-4 译码处理模块端口列表

端口名称	端口位宽	端口类型	端口描述
clk	1	input	65.536MHz工作时钟
en	1	input	复位信号，高有效
inf_rate	3	input	速率指示信号
code_read	1	input	4 路并行数据读取完成指示，来自数据读取模块
tb_start	1	input	回溯开始指示信号，来自数据读取模块
cur_code1	5	input	输入的待译码数据
cur_code2	5	input	输入的待译码数据
cur_code3	5	input	输入的待译码数据
cur_code4	5	input	输入的待译码数据
traffic_bit_read_addr	12	input	第 1 路数据寄存器读地址
supply_bit_read_addr	11	input	第 2 路数据寄存器读地址
traffic_bit_read_en	1	input	第 1 路数据寄存器读使能
supply_bit_read_en	1	input	第 2 路数据寄存器读使能
frame_end	1	input	读完 1 子帧指示信

			号，来自数据读取模块
t_or_s	1	input	数据类型指示
tb_finish	1	output	1 子帧译码完成信号，送往数据读取模块
traffic_bit	1	output	译码之后的第 1 路数据
supply_bit	1	output	译码之后的第 2 路数据
traffic_decode_finish	1	output	第 1 路数据译码完成指示
supply_decode_finish	1	output	第 2 路数据译码完成指示

经由 ACS 单元运算得到的各状态的幸存路径度量值和回溯路径分别存储到路径度量值寄存器和回溯路径寄存器，不同状态下路径度量值的读取和存储操作在第三章的 3.3 有介绍。在实现过程中通过 Verilog 语言中的 case 语句，根据不同的运算次数分离和组合各个状态的路径度量值，并且生成各个寄存器的相应地址加以读取和存储。下图是第 1 次 ACS 运算的输入状态和输出状态，接下去的 7 次 ACS 运算中，初始状态依次加 1。

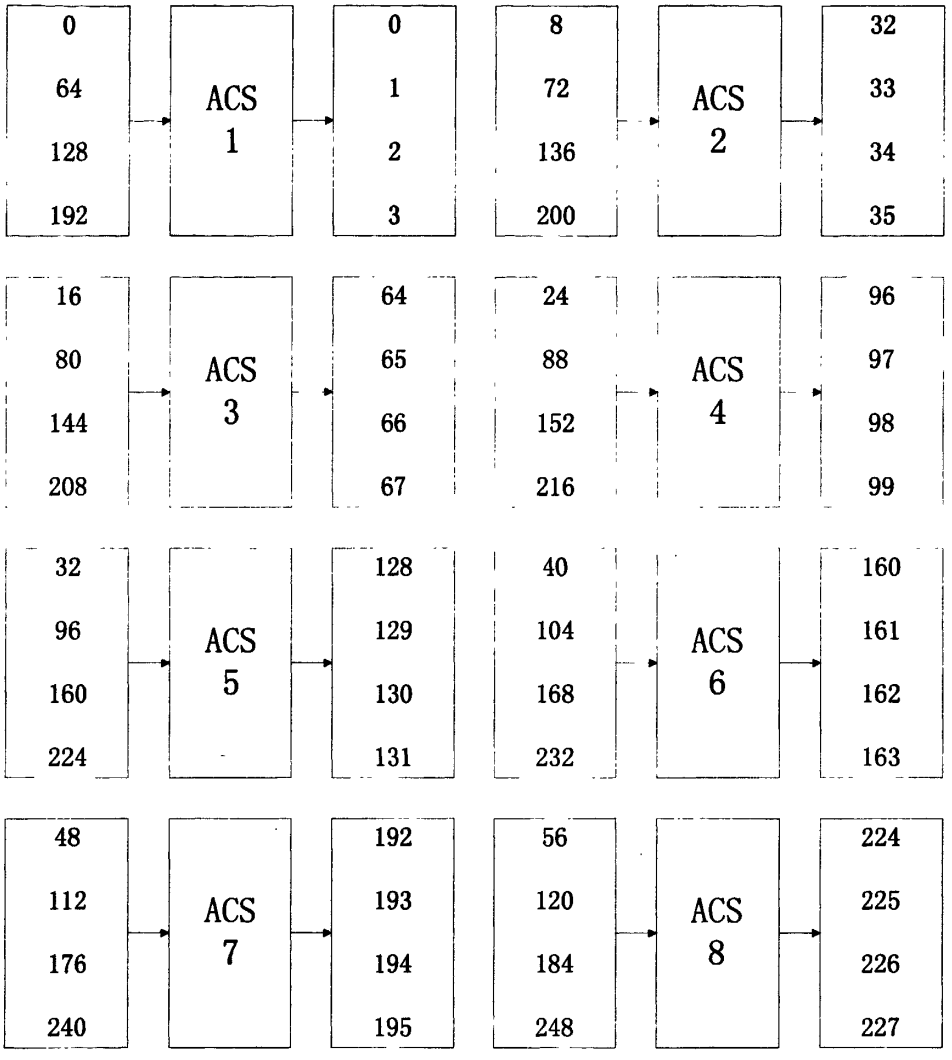


图 4-8 第 1 次运算的输入和输出状态

ACS 运算不但生成各个状态的幸存路径度量值，还生成各个状态的幸存路径。幸存路径由 1 个 2 比特的数字表示，即用 0，1，2，3 这 4 个数字表示输出状态是由 4 个输入状态中的哪一个转换来的。这些幸存路径值被存储到回溯路径寄存器，回溯路径寄存器的结构在第三章的 3.4 节有介绍。

当检查到 *tb_start* 信号后，即开始回溯并输出译码结果。回溯路径寄存器的地址由当前状态，回溯深度来得到，选择 0~3 号寄存器中的哪一个，由当前状态决定。

从寄存器中读出的 2 比特的值，对应基-4 蝶形图的 4 条潜在路径中的 1 条，也即是当前状态的前一个状态的最高 2 位，通过这个值即可得到前一状态。当前状态的最低 2 位也即是译码器的输出。

译码结果将进行 1 次串并变换，从 2 比特数据变成串行的 1 比特数据，并被

存储到译码结果寄存器，等待 1 子帧码字译码结束之后进行输出。

4.4.5 数据输出模块

数据输出模块读取译码的结果，按照接口之时的速率进行输出。表 4-5 列出了模块的主要接口。

表 4-5 数据输出模块端口列表

端口名称	端口位宽	端口类型	端口描述
clk	1	input	65.536MHz工作时钟
en	1	input	复位信号，高有效
inf_rate	3	input	速率指示信号
traffic_decode_finish	1	input	数据译码完成指示
traffic_bit	1	input	译码之后的数据
traffic_bit_read_addr	12	output	比特寄存器读地址
traffic_bit_read_en	1	output	比特寄存器读使能
traffic_bit_out	1	output	译码结果
traffic_bit_out_en	1	output	译码结果使能

模块在接收到译码处理模块送来的 *traffic_decode_finish* 信号之后，即开始读取并输出译码结果寄存器内的业务数据译码结果。模块的实现中，根据不同的业务数据速率，设置了不同的数据输出 counter_max 值，使得译码结果的读取和输出可以按照要求的速率进行。

数据输出模块对译码结果的读取和输出是按照实际的数据速率进行的，因此其速率远远低于译码的速率，所以可以做到持续不断的输出译码结果。

4.4.6 译码器顶层模块

译码器顶层模块是所有上述模块的顶层模块，将上述模块联结起来。该模块接收来自系统其他模块的信号，分别将这些信号送入相应的子模块进行处理，并将子模块的输出导入下一级的处理。最终的输出结果也是通过译码器顶层模块送出。表 4-6 列出了模块的主要接口。

表 4-5 译码器顶层模块端口列表

端口名称	端口位宽	端口类型	端口描述
clk	1	input	65.536MHz工作时钟
rst	1	input	复位信号，高有效

inf_rate	3	input	速率指示信号
frame_syn	1	input	帧同步信号
frame_lost	1	input	帧同步失败指示信号
traffic_in_en	1	input	第 1 路数据使能
supply_in_en	1	input	第 2 路数据使能
traffic_in	5	input	第 1 路数据数据
supply_in	5	input	第 2 路数据数据
traffic_bit_out	1	output	第 1 路数据译码结果
traffic_bit_out_en	1	output	第 1 路数据译码结果使能
supply_bit	1	output	第 2 路数据译码结果
supply_decode_finish	1	output	第 2 路数据译码结束信号

译码器顶层模块的另一个作用是处理来自帧同步模块的*frame_lost*信号。每当检测到*frame_lost*信号，则说明帧同步处理失败，需要等待下一次的帧同步成功才能有效的进行译码操作。于是，译码器顶层模块将*frame_lost*信号也作为一种复位信号。

第五章　维特比译码器的仿真和测试

5.1　译码器仿真

用 Verilog HDL 语言完成了译码器的设计之后，根据 FPGA 的设计流程，需要进行对应的功能仿真（又称前仿真）和布线后门级仿真（又称后仿真）。

5.1.1　功能仿真

功能仿真的主要目的是测试 Verilog HDL 程序设计的正确性，仿真过程忽略电路的时延等因素，主要使用的仿真工具是 Modelsim6.2。为了对译码器进行仿真验证，设计了如下图的仿真测试平台。

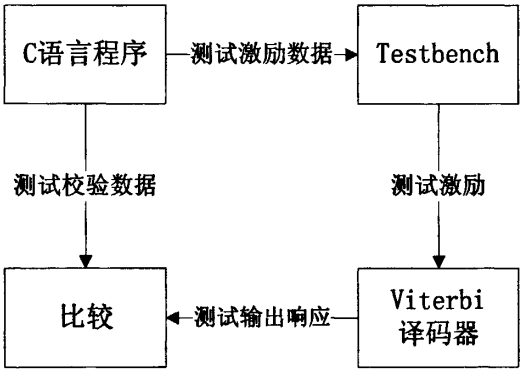
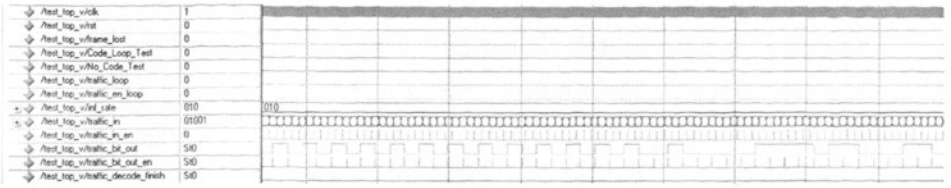


图 5-1 功能仿真测试平台示意图

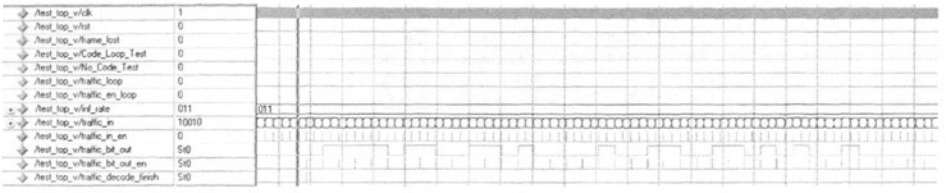
测试平台由两部分组成，第一部分是 C 语言的仿真程序，这一部分程序在译码器设计的前期用来对译码算法进行仿真，分析算法性能。在译码器的 RTL 级代码编写完成以后，这一部分的程序为功能仿真提供测试激励数据和测试校验数据。第二部分是 Verilog HDL 编写的 Testbench，Testbench 读取 C 语言程序产生的激励数据，并转换成相应的测试激励提供给待测的译码器，译码器产生的响应和 C 语言程序产生的译码输出比较，如果比较结果一致则证明了设计的正确性。

仿真工具 Modelsim6.2 读入 Testbench 和译码器的 Verilog HDL 设计文件，链接，编译后进行仿真，在仿真过程中可以自由的设置仿真时间，显示或隐藏响应的信号，很大程度上方便了程序的调试过程。以下是各个不同速率下业务数据的仿真结果：

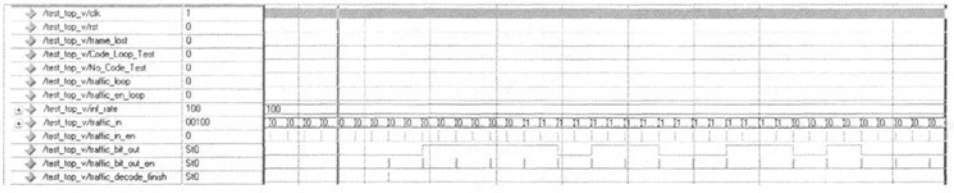
1) 第 1 路数据 64Kbps 仿真结果：



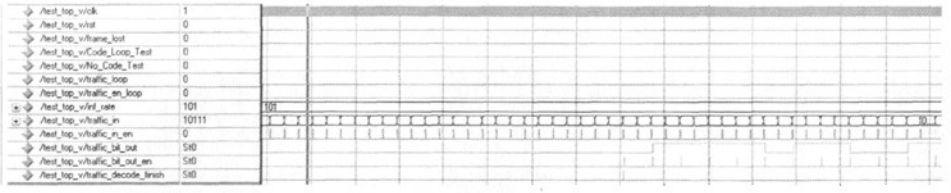
2) 第 1 路数据 128Kbps 仿真结果：



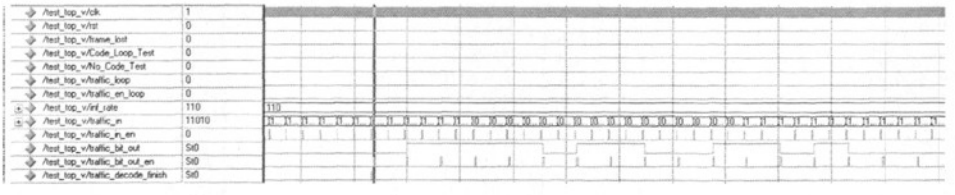
3) 第 1 数据 256Kbps 仿真结果：



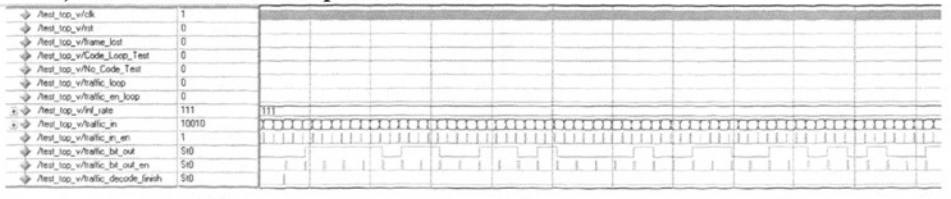
4) 第 1 路数据 512Kbps 仿真结果：



5) 第 1 路数据 1024Kbps 仿真结果：



6) 第 1 路数据 2048Kbps 仿真结果：



在仿真过程中发现的问题需要回到程序中做进一步的修改，将修改后的程序再次仿真测试，直到结果正确，即可进入下一步的调试。

5.1.2 布线后门级仿真

完成功能仿真，确定 Verilog HDL 程序的正确之后，下一步就是使用 Xilinx ISE 自带的工具对程序进行综合和布局布线。综合过程即是把 Verilog HDL 程序转化成门级的电路，布局布线既是把综合所生成的门级电路转化成为可以写入 FPGA 的逻辑元件布局。在这个过程中还需要输入模块的时序，管脚等约束条件，使得生成的结果适用与实际设计要求。

表 5-1 列出了译码器在综合和布局之后的资源使用情况：

表 5-1 Viterbi 译码器资源使用状况

芯片资源	使用状况	百分比
BUFGs	1 out of 32	3%
ILOGICs	1 out of 448	1%
External IOBs	26 out of 448	5%
LOCed IOBs	0 out of 26	0%
RAMB16s	23 out of 192	11%
Slices	5369 out of 15360	34%
SLICEMs	0 out of 7680	0%

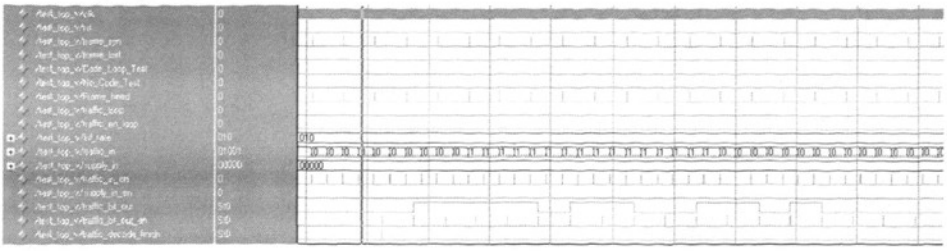
由上表可见，本论文中实现的 Viterbi 译码器消耗主要的芯片资源是 FPGA 的 RAM 和 Slices。前者用于生成译码器中的各个数据寄存器，后者构成了译码器中的各种控制和运算逻辑。由于译码器采用了串并结合的运算结构，在一定程度上节省了芯片资源。布线之后译码器所支持的最高时钟频率为 88.739MHz，超过了设计所需要的 65.536MHz，能够实现数据吞吐量的要求。

布线后门级仿真就是对 Xilinx ISE 所生成的布局布线的结果进行仿真，在这一步的仿真过程中，不止是要验证布局布线后逻辑的正确与否，还引入了电路之间的时延等实际存在，并且会对最终的测试产生影响的因素。所以布线后门级仿真是一种更加接近实际的，更加全面的仿真过程。

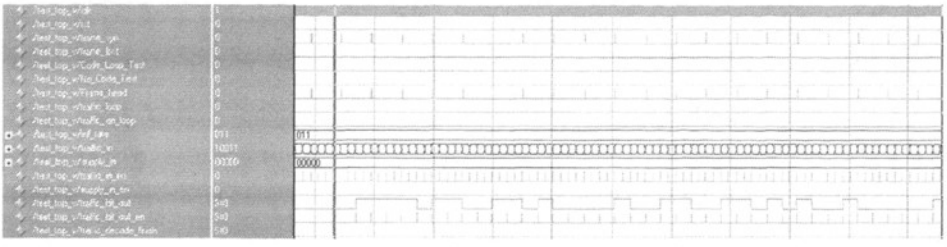
因为程序已经经过了综合，布局布线等处理，其中使用的一些变量和寄存器都变成了实际的电路。因此，无法像功能仿真时那样随意的查看所有的中间处理过程，而是需要将某些信号引到顶层才能查看，所以布局布线后仿真的操作能加繁琐一些。

布线后门级仿真所使用的工具仍然是 Modelsim6.2，软件读入 Xilinx ISE 生成的布局布线结果，读取测试激励，并生成仿真结果。

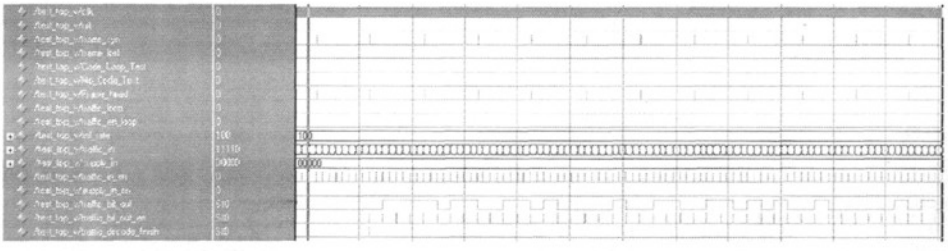
1) 第 1 路数据 64Kbps 仿真结果：



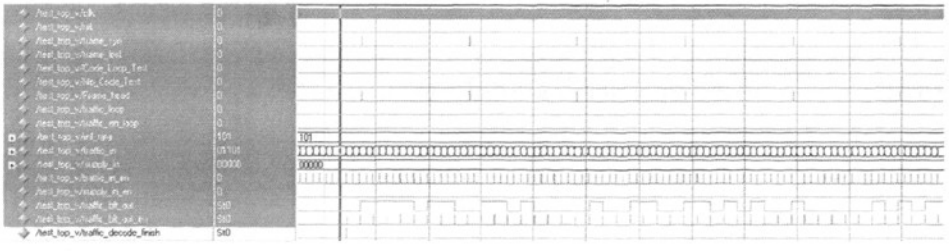
2) 第 1 路数据 128Kbps 仿真结果：



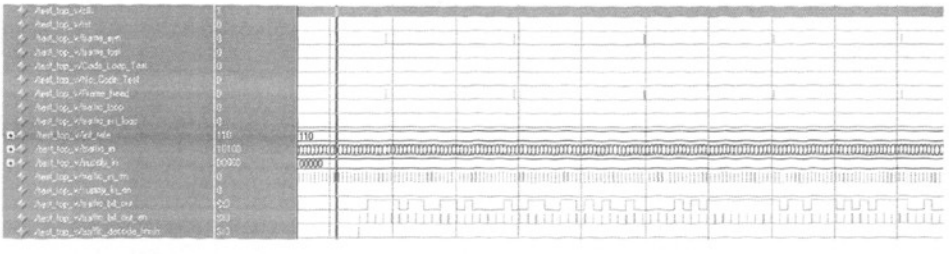
3) 第 1 路数据 256Kbps 仿真结果：



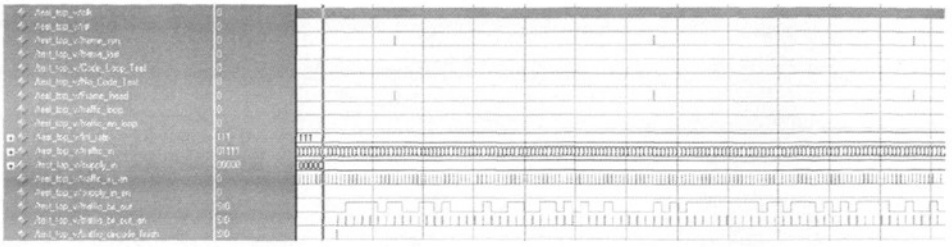
4) 第 1 路数据 512Kbps 仿真结果：



5) 第 1 路数据 1024Kbps 仿真结果：



6) 第 1 路数据 2048Kbps 仿真结果：



5.2 译码器测试

完成译码器的仿真，综合，布局布线之后，下一步工作就是把 Xilinx ISE 在布局布线阶段生成的，用于 FPGA 芯片下载的 BIT 文件加载到相应的 FPGA 芯片中去，在芯片中测试设计是否能正确运行。

在测试中，需要观测并记录保存模块工作过程中产生的信号，主要使用的工具有逻辑分析仪和 ChipScope 软件。

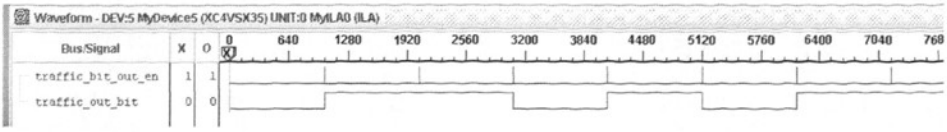
逻辑分析仪需要在布局布线，生成 BIT 文件的过程中，将要观测，保存的信号引到开发板的预留的外接管脚上，然后通过探头将信号导入逻辑分析仪，再设置一些触发条件，当信号满足这些条件时，就会显示并保存这些信号所对应的数据。逻辑分析仪支持较大的存储深度和多种灵活的触发条件，但是开发板上的预留管脚比较有限，一次无法引出太多的信号，测试信号的修改也比较麻烦。

ChipScope 软件通过在芯片中插入测试用的 IPcore 来完成信号的显示和保存。在完成 Verilog HDL 程序的综合之后，即可调用 ChipScope 生成对应的测试模块，这个 ChipScope 模块可以和待测模块中特定信号相连接，还可以设置一定的触发条件和保存深度。将加载了 IPcore 的模块完成布局布线并生成 BIT 文件之后，将之下载到 FPGA 芯片中，就可以通过 ChipScope 软件启动这个测试模块，当信号达到触发条件后，ChipScope 测试模块就会把信号保存到芯片中的 Block RAM 中，当数据存满之后，再从 RAM 中读出信号并显示出来。用 ChipScope 进行测试不需要引出信号线，不受预留管脚的限制，修改也比较灵活，缺点是需要占用芯片的资源，而且存储深度比不上逻辑分析仪。因为译码器测试所需的数据深度有限，而且需要引出的中间变量比较多，所以本设计主要采用 ChipScope 作为测试工具[12]。

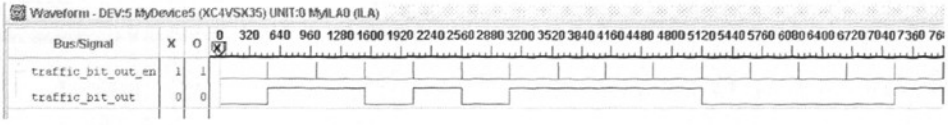
在测试过程中，将 C 语言仿真程序生成的 1 帧的业务和补充数据写入芯片的 RAM 中，并依照不同的速率读取出来，配合芯片上的时钟，作为测试激励输入译码器模块，用 ChipScope 软件记录译码输出，并和 C 语言仿真程序的译码结

果做比较。以下是各种速率下最终的测试结果。

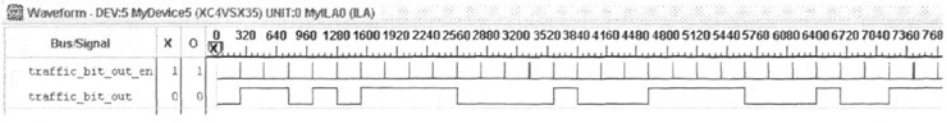
1) 第 1 路数据 64Kbps 测试结果：



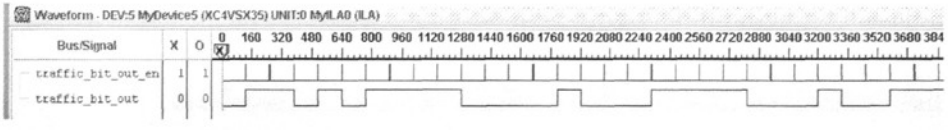
2) 第 1 路数据 128Kbps 测试结果：



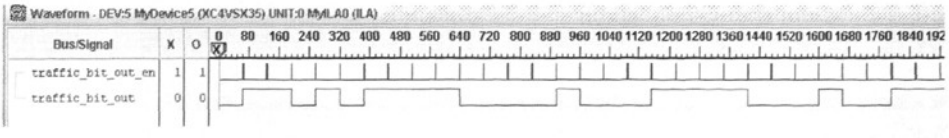
3) 第 1 路数据 256Kbps 测试结果：



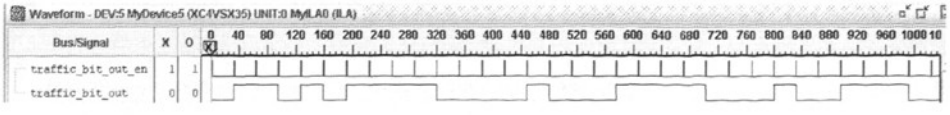
4) 第 1 路数据 512Kbps 测试结果：



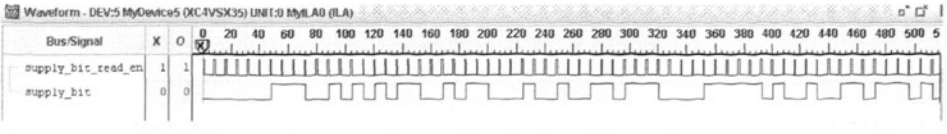
5) 第 1 路数据 1024Kbps 测试结果：



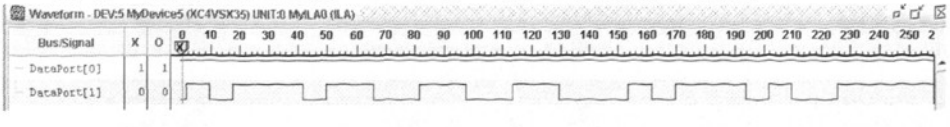
6) 第 1 路数据 2048Kbps 测试结果：



7) 第 1 路数据 64Kbps 时第 2 路数据测试结果：



8) 第 1 路数据 128Kbps～2048Kbps 时第 2 路数据测试结果



本论文实现的 Viterbi 译码器经过了功能仿真，门级仿真和芯片测试，均能够得到正确译码结果。可见本论文中实现的维特比译码器是正确的。

第六章　结束语

6.1　主要工作总结

从 1967 年维特比提出卷积码的译码算法以来，维特比译码算法本身已经有了很大的发展。各种改进的，可应用于不同领域的维特比译码算法被陆续提出，在信道译码上的应用尤为突出，已经成为了主流的第三代通信协议中处理卷积译码的方法。

因为维特比译码算法的广泛应用，研究其在实际的工程中的应用也十分重要，当前通信算法的主要使用可编程的芯片和 DSP 实现。随着大规模集成电路技术和数字通信的发展，人们对于数字通信的通信质量和通讯速度要求日益提高，更快、功耗更低并且可以使用于多领域的处理芯片将会成为今后研究的重点。

本文在分析了维特比算法的基础上，对算法复杂度和译码器性能之间的关系做了详细的阐述，总结了维特比译码器设计的关键参数，并用 C 语言程序对算法进行了仿真，检验维特比算法有效性的同时为后面的设计提供了测试数据。

本文在分析研究现有的维特比算法实现技术的基础上，设计了一个可应用于多路可变速率的软判决维特比译码器。设计中采用了串并结合方式和基-4 的 ACS 运算单元，同时，设计了与之相对应的路径度量值和回溯路径寄存器的组织结构，采用这种方式不会打断 ACS 的流水线操作，同时还具有内部连接关系简单，地址和控制信号产生非常容易，易于对控制模块的逻辑进行时序优化的特点，综合结果也表明采用这种设计方式大大减少了逻辑单元的资源占用。

为了验证设计的正确性，在本文中设计了功能仿真平台，采用了业界流行的仿真、调试工具对设计进行了验证。

最后，本文中设计的译码器在 Xilinx Virtex4-SX35 FPGA 芯片上进行了片上测试，文中简单介绍了测试平台的搭建方法和调试技术。

本文中设计的（2，1，9）软判决维特比译码器通过了片上测试，运行稳定可靠，综合结果也表明该译码器的性能和资源占用情况是相称的，有较高的工程实用价值，同时也为以后进一步的研究进行了技术积累和储备。

6.2 下一步工作展望

本文中设计的维特比译码器符合设计要求，具有较高的工程应用价值，但是仍有一些需要进一步研究和改进的地方：

1) 本文设计的译码器为了节省资源，在读取数据寄存器时没有使用缓存，而是顺序依次读取和送出数据，这种操作造成译码的 ACS 单元需要等待 4 个数据全部送到才能开始运算。因此，本来只需要 12 个时钟周期的 ACS 运算操作被分配了 32 个时钟周期，其中大部分时间都在等待数据的送达。最后综合的结果证明，这样的设计其实并不能节省太多的资源，反而浪费了很多的时钟。所以，在下一步的优化中，可以根据实际需要，或者减少分配给 ACS 单元的时钟周期，以提高译码器的处理速率，或者保持时钟周期不变，减少 ACS 单元的个数，以减少译码器所需的资源。

2) 本文设计的译码器是固定参数的，而在无线多媒体传输系统中，不同的信道质量和业务对编码增益的要求不一样。信道质量好的情况下需要高的译码速率，低延迟，信道质量差的情况下需要高的编码增益。语音、视频信息流需要高的译码速率来保证实时性，数据业务需要低的误码率保证准确性。所以，设计编码效率，约束长度等参数可配置的译码器是下一步的主要工作内容。

3) 本文设计的译码器综合后的最大运行时钟频率为 65.536MHz，距离 Xilinx 公司的 Virtex4-SX35 FPGA 芯片的最大运行时钟频率还有相当的差距。所以说该设计还有很大的优化空间。认真分析设计的关键路径，优化系统时序使之工作在更高的时钟频率也是下一步需要重点做的工作。

维特比译码器设计的方法和技术还有很多，本文中仅仅介绍了其中的一部分，局限于本人知识水平和项目开发时间，文中的不足之处还很多，需要进一步的学习和提高。

参考文献

[1] 吴伟陵，牛凯,《移动通信原理》，电子工业出版社，2005，pp.183-201

[2] John G.Proakis 《数字通信》（第四版） 电子工业出版社 2003.1.

[3] Viterbi, A. J. C, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm, IEEE Trans. Inform. Theory, vol. IT-13, pp. 260-269, April.

[4] Chameleon Logics "Viterbi decoding in field programmable gate arrays(FPGA)" Soild-State Circuits IEEE Journal of Volume:25, Issue:4, Aug.1990

[5] ANDRIES P. HEKSTRA "An Alternative to Metric Rescaling in Viterbi Decoders" IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. 31, NO. 11, NOVEMBER 1989

[6] 夏宇闻 《Verilog数字系统设计教程》 北京航空航天大学出版社 2003.7

[7] 周炯磐，庞沁华等,《通信原理(下)》，北京邮电大学出版社，2002，pp.148-150

[8] 查光明，熊贤祚 《扩频通信》， 西安电子科技大学出版社, 2002.

[9] Theodore S. Rappaport 著，蔡涛等译. "Wireless Communications Principles and Practice"，电子工业出版社，1999

[10] Meyr, H., Moenclaey, M., and Fechtel, S. A. "Digital Commun. Receivers", Wiley, New York. 1998.

[11] H.Meyr，MaerMoenelaey，StenafA.Feehtel, "DigitalConununieationReeeivers: SynehroniZation, ChnnaelEstimation and Singal Poreessing" JohnWiley&Sons，Inc.1998

[12] 田耘，徐文波，张延伟 《无线通信FPGA设计》， 电子工业出版社，2008

[13] S. G. Wilson， "Digital communication and Coding"（影印版）， 电子工业出版社 1998，p604

[14] Rodger E.Ziemer, Roger L.Peterson, "Introduction to Digital Communication", Prentice-Hall, Inc. 2001, pp.360-365

[15] M. K. Simon, M. Alouini "Digital communication over fading channels-A unified approach to performance analysis", John Wiley & Sons Inc. 2000

[16] 任晓东，文博 《CPLD/FPGA高级应用开发指南》，电子工业出版社，2003.

附录 1. 缩略语及符号说明

FEC	Forward Error Correction
ARQ	Automatic Repeat-reQuest
HEC	Hybrid Error Correction
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate-Array
CPLD	Complex Programmable Logic Device
DSP	Digital Signal Processing
HDL	Hardware Description Language
CDMA	Code Division Multiple Access
BSC	Binary Symmetric Channel
RTL	Register Transfer Logic
EDA	Electronic Design Automation
DPRAM	Dual Port Ram

致 谢

在这里，我首先要衷心的感谢我的导师吴伟陵教授。硕士研究生阶段吴伟陵老师在学术和生活上对我进行了悉心的指导和热情关怀，引导我了解和熟悉了无线通信领域的前沿技术以及宽广的发展前景。

同时感谢林家儒教授，在撰写论文这一段时间里，林家儒老师对我的研究方向进行了详尽的分析，并对我在技术上遇到的一些问题进行了指导，帮助我完成了本论文的研究工作。林家儒老师也十分关心我的生活和学习，并尽其所能的为我们的研发创造良好的条件。从林家儒老师身上，我汲取了到很多受益终身的东西。

在这里还需要感谢我实验室的吴文礼老师、别志松老师、林雪红老师和吴晓非老师，在我的研究生学习生涯中，他们一直给予我最真诚的帮助和关心，给了我很多学习上的指导和精神上的鼓励。

感谢项目组的于光炜、李志、金朝晖、卢宏刚、迟大鹏、胡彬、田耘、谷涛、云峻岭、王怀、季晓鹏、陈俊杰等同学，这些同学在我困难时都曾给予我无私的帮助，在我的论文项目研究中提供了十分有益的指导和帮助。

感谢项目里合作厂方的石志涛工程师和闫挺工程师，他们在项目的设计、调试上给予了我很大的帮助。

最后，再次感谢那些在我的研究工作中帮助过我的老师和同学们，同时也感谢所有在我成长过程中帮助过我的人们。

攻读硕士学位期间发表论文

[1] 杨刚，《有冲突的无线 Mesh 网络中随机网络编码对文件共享的增益》，中国科技论文在线，2008.7.18