代　号	10701	学　号	1077190430
分类号	TP334	密　级	公开
U D C		编　号	

题（中、英文）目　基于 FPGA 的二值图像 JBIG 压缩算法研究与实现

Research and Implementation of JBIG Compression for Binary Image Based on FPGA

作者姓名	陈聪	学校指导教师姓名职称	王泉 教授
工程领域	计算机技术	企业指导教师姓名职称	尤玉虎 高工
论文类型	应用基础技术	提交论文日期	二〇一三年一月

西安电子科技大学

学位论文创新性声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师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，论文中不包含其它人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它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做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：_____ 日期_____

西安电子科技大学

关于论文使用授权的说明

本人完全了解西安电子科技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西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布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论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名单位为西安电子科技大学。

（保密的论文在解密后遵守此规定）

本学位论文属于机密，在___年解密后适用本授权书。

本人签名：_____ 日期_____

导师签名：_____ 日期_____

摘要

图像压缩技术是打印系统中关键技术之一，直接影响数据存储空间大小以及数据传输速度，在打印机中占有重要地位。传统的图像压缩是在 PC 机上依靠高速处理器以及大容量存储设备，以满足打印系统时性要求。但是面对打印机独立处理数据需求的日益增加，能够快速、高效的处理和存储数据，将图像压缩过程移植到打印系统内部就成为了新的课题。

本文主要是对在 FPGA 上二值图像 JBIG 压缩算法及硬件实现进行了研究。论文首先阐述了一个包含压缩过程的打印机系统架构；然后着重介绍了 JBIG 压缩算法的特点以及适用于 FPGA 实现的算法的并行处理方法和流水线优化方法；最后论述了实现方法中的数据结构、时序控制、读写控制等关键技术的实现，并通过实验数据验证了压缩与解压缩过程。达到打印系统独立压缩与解压缩设计的目的。

关键字：二值图像压缩　JBIG　FPGA　打印系统

Abstract

Image compression technology is one of the key technologies in a printing system, which has a direct impact on the size of the data storage space and the speed of data transfer. It occupies an important position in a printer. The conventional image compression is to meet the real-time requirements of a printing system by the use of the high-speed processors and large-capacity storage devices on PC. With the increasing demands of independent data processing, the transplantation of image compression process into a printing system, which is fast and efficient in data processing and storage, is becoming a new topic.

This paper is a research on a JBIG compression algorithm using Verilog HDL implemented on FPGA for the binary image. The thesis describes a printer system architecture including a compression process, and then highlights the characteristics of the JBIG compression algorithm, the algorithm idea, and the optimized algorithm using parallel processing and pipeline. It discusses the design of data structure and implementation of timing control, reading and writing control in detail. The compression and decompression process is verified by a set of experimental data. The paper achieves the design purpose of independent compression and decompression within a print system in the end.

Keyword: Binary Image Compression JBIG FPGA Pinting System

目录

第一章 绪论

1.1 课题背景及意义

信息时代的到来，导致大量数据快速增长，随之带来的是数据传输以及存储介质的革新。但是面对如今数据快速增长的现状，人们对图像数据质量和分辨率的要求越来越高，这就导致了更高的数据率和更加复杂的数据类型，使得用来描述图像的数据爆炸性的增长[1]。由于难度以及成本等多方面因素制约，单纯依靠提高运算能力和不断增加存储介质的容量已经不能满足发展的要求。所以高效的数据压缩技术得到了人们的重视，并且发展迅速。数据压缩通俗地说，就是用最少的数码来表示信号。其作用是：较快地传输各种信号，压缩数据的存储容量以及降低发信机功率[2]。数据压缩的总目标是要减少容纳给定消息集合的信号空间。所谓信号空间，是指物理空间、时间区间与电磁频谱区域，也就是为存储、传输给定信号集合所占用的空域、时域与频域空间。存储空间的减小也意味着传输效率的提高和占用频带的节省[3]。

数据压缩技术广泛的应用于打印机设备中。打印的幅面、分辨率以及速度都是衡量一个打印机设备优良的标准，幅面的增加以及分辨率的提高都会使图像存储数据量呈指数增长，而数据量的多少又直接影响传输速度。所以数据压缩对于优质的打印机十分必要。

一个打印机系统的打印流程如图所示 1.1 所示。包括图像半色调过程，数据压缩，数据传输、缓存，数据解压以及最后的图像打印。

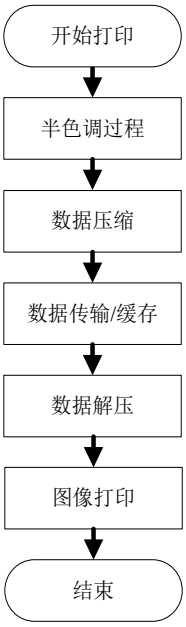

图 1.1 打印流程图

计算机传输的是连续色调图像，需要通过半色调技术将其转化为打印机可直接使用的数据类型。数字半色调技术是基于人眼的视觉特性和图像的成色特性，利用数学、计算机等工具，在二值（或多色二值）呈色设备上实现图像的最优再现的技术，是将连续色调图像经过处理后再输出以实现图像阶调再现的基础性研究[4]。利用半色调技术可以将一幅高分辨率的灰度级图像转换为低分辨率的二值图像（0 或 1 值），由于人类视觉的低通滤波特性，在一定的距离外观察转换后的二值图像，仍然觉得是一幅连续色调的图像[5]。

数据压缩技术应用于图像处理可分为静态图像压缩以及动态图像压缩。打印机处理的均为静态图像。静态图像常见国际数据标准及特点见表 1.1。

表 1.1 国际数据压缩标准

标准	简称	通用信源	典型应用
ITU-T T.82 ISO/IEC 11544	JBIG	二值图像 图形	G4 传真机 计算机图形
ITU-T SG8 ISO/IEC 14492	JBIG-2	二值图像 图形	G4 传真机 计算机图形
ITU-T T.81 ISO/IEC 10918	JPEG	连续色调 静止图像	图像库 传真 彩色印刷 数码相机
ITU-T T.87 ISO/IEC 14495	JPEG-LS	连续色调 静止图像	医学 遥感图像资料的无损、近似压缩
ISO/IEC 15444	JPEG 2000	连续色调 静止图像	各种图形 图像

从实时性的的角度来看，为了提高整个图像压缩处理系统的运行速度，通常可以有两类方法：其一，对图像压缩编解码算法的改进和简化，但事实上，很多算法理论已经相当成熟，降低算法实现复杂度的余地并不大；其二，对实现图像压缩处理手段的选择，根据算法的特性及对速度和质量等的需求，有针对性地选择适合的实现手段[6]。由于各类压缩算法计算量大，占用存储空间多，压缩算法方案常使用软件方式实现，以解决复杂算法和动态分配存储空间问题。其缺点是过多的占用 CPU 的宝贵资源，使得打印机只能依赖 PC 机进行预处理，而不能脱离 PC 机独立运行。为了拓展打印机独立使用空间、加快处理速度，有必要利用硬件方式实现压缩过程。现代 FPGA（Field Programmabel Gate Array 现场可编程阵列）的发展使得采用硬件方式实现压缩、解压缩过程成为可能[7]。

JBIG 压缩算法适用于二值图像的压缩，有较高的压缩比，其核心压缩编码方式基于算数编码，仅有加、减、移位操作，不涉及乘、除操作，有利于硬件方式实现。对于连续多幅面图像的处理，由于打印头喷墨过程是机械运动，其打印速

度受物理因素限制，其处理时间会大于压缩过程以及数据传输过程，这就要求压缩后的数据需要 FPGA 以外的存储设备进行数据缓存。

一个打印机系统总体框架如图 1.2 所示。其包含图像处理模块、打印机导引头引擎模块以及数据缓存模块三部分。系统工作采用基本流程是：首先图像处理模块将原图像数据读入，经过数据缓存、半色调以及压缩处理后输出。输出的压缩图像数据存储在数据缓存模块中。在需要打印数据时，将图像压缩数据读入到打印机导引头引擎模块中，将解压缩后数据缓存在打印头 FIFO 中，通过打印头引擎控制打印机开始打印操作。整个系统独立于 PC 机可以独立运行，自行控制数据传输与操作。

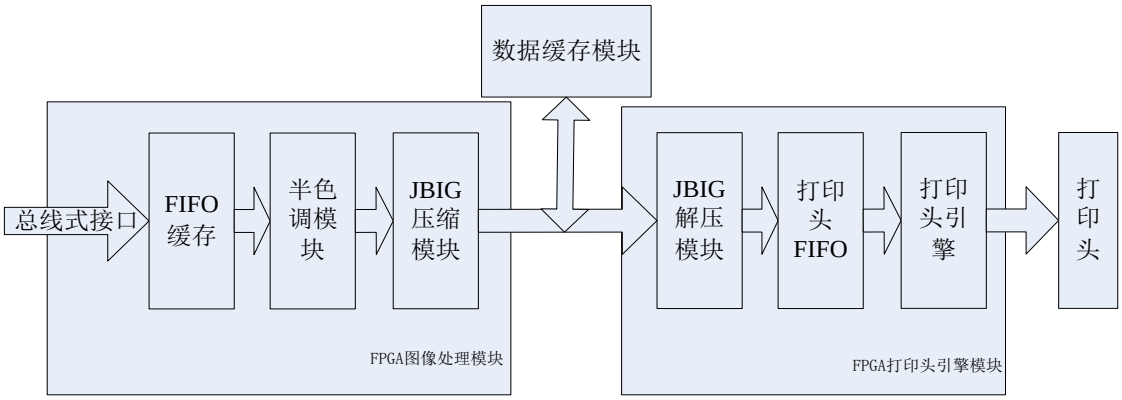

图 1.2 打印机系统总体框架图

综上所述如何高速实现 JBIG 压缩算法并构建完整可靠的压缩与解压系统，使其独立完成压缩与解压过程，有效减少传输时间，降低存储容量，满足打印机快速打印功能是本文的主要研究内容。

1.2 论文主要工作

本文主要研究了 JBIG 压缩算法并提出硬件实现方案。采用 Verilog 语言实现基于 FPGA 的 JBIG 压缩算法设计，通过对算法在关键问题上的探讨，完成并行处理、流水线结构等方面的改进，达到加快处理速度以及减少存储空间的目的。论文的研究工作主要包括：

(1) 介绍了图像压缩编码基础，压缩算法的分类。并着重分析 JBIG 压缩算法思路。

(2) 对于 JBIG 压缩算法关键问题讨论，提出并行处理、流水线结构以及各方面改进方法。

(3) 采用 Verilog 语言在 FPGA 上分别实现 JBIG 压缩过程以及解压缩过程，进行仿真验证。

(4) 通过两类实验验证，对实现的正确性进行验证，并且探讨压缩性能。

(5) 对整体方案进行深入分析，提出现有实现方案的不足以及进一步改进方法。

1.3 章节安排

第一章，绪论。介绍本文的课题背景意义，简述本文探讨的内容以及概括了所研究的工作，介绍论文的整体组织结构。

第二章，介绍了图像压缩编码基础，并着重介绍 JBIG 压缩算法思想。

第三章，介绍了基于 FPGA 硬件平台利用 verilog 语言实现具体方案。研究了 JBIG 压缩算法各模块间并行处理设计，探讨编码模块处理速度的瓶颈问题，并提出了优化方案。对压缩处理涉及的各个模块进行分析和仿真验证。最后对仿真验证进行分析。

第四章，研究 JBIG 压缩算法的解压模块的并行处理设计，介绍具体实现方案，对各模块进行分析和验证。

第五章，对基于 FPGA 的 JBIG 压缩算法整体进行功能和性能测试，给出测试结果，对整个压缩系统作评价。分析当前方案的不足，提出进一步改进的方法，并对将来工作进行展望。

第二章 数据压缩及 JBIG 压缩算法

2.1 图像编码基础

编码是对消息符号进行编码处理的过程。编码包括信源编码、保密编码、信道编码三大类，其中，信源编码是对信源输出的消息进行适当的变换和处理，以尽可能提高信息传输的效率，而信道编码是为了提高信息传输的可靠性而对信息进行的变换和处理[8]。图像压缩处理就是对图像进行编码的过程，压缩数据的重要方法是消除冗余数据，从数学角度来说是将原始图像转化为从统计角度看尽可能不相关的数据集。这个转换要在对图像进行存储、处理和传输等之间进行，而在这之后需要将压缩了的图像解压缩以重建原始图像或其近似图像[9]。

2.1.1 数据冗余

数据是用来表示信息的。可以使用不同的方法表示给定量的信息，假设用 n_1 和 n_2 分别表示相同信息的两个数据集合中的信息载体单位的个数，那么相对冗余 R_D 定义为：

$$R_D = 1 - \frac{1}{C_R} \tag{2-1}$$

其中 C_R 称为压缩率：

$$C_R = \frac{n_1}{n_2} \tag{2-2}$$

当 $n_1=n_2$ 时，$C_R=1$，$R_D=0$ 表示信息的第一种表达不包含任何冗余。当 $n_2<<n_1$，$C_R \to \infty$，$R_D \to 1$ 表示有显著的压缩和大量的冗余数据。当 $n_2>>n_1$，$C_R \to 1$，$R_D \to -\infty$ 表示第二个集合中包含有的数据大大超过原表达方式的数据量，这种数据扩展是不希望出现的情况。在数字图像压缩中，常用到的有三种基本的数据冗余：编码冗余、像素间冗余、心理视觉冗余。

2.1.1.1 编码冗余

对于一副由有规则的、在某种程度上具有可以预测形状和反射对象组成的图像，正常情况下其各个灰度级出现的概率是不相同的（即图像的直方图不是均匀分布的），然而它们的灰度级的自然二进制编码对有最大和最小可能出现的值分配

相同的比特数，从而产生的冗余就称为编码冗余。

2.1.1.2 像素间冗余

图像中任何给定的像素值若可以根据这个图像的相邻像素进行适当的预测，那么这个给定的像素就可以视为像素间的冗余。JBIG 压缩算法中有多个模块用于检测和消除像素间冗余，依靠的就是图像中像素间相关联的这种冗余关系。

2.1.1.3 心理视觉冗余

人眼感觉到的图像区域亮度不仅仅与区域的反射光有关，也包含其它一些因素。其原因是眼睛并不是对所有视觉信息有相同的敏感度。有些信息通常的视感觉过程中相对于其它信息并不重要，这些信息就是心理视觉冗余。去除心理视觉冗余的数据会导致信息的损失[10]。

2.1.2 压缩编码分类

2.1.2.1 无损压缩编码

无损压缩是利用数据的统计冗余进行压缩，可以完全恢复原始数据而不引起任何失真的压缩编码方式。无损压缩技术通常由两种彼此独立的操作组成：⑴为减少像素间冗余，建立一种可替代的图像表达方式；⑵对这种表达方式进行编码以便消除编码冗余。这种编码方式广泛应用于医学图像、卫星成像、商业文档等领域。无损压缩编码的优点是 100%保存原始信息，没有信号丢失，不容易受到信号源影响，容易转换到其它图像格式。但是缺点是压缩比不高，存储空间大。

常见的无损压缩编码有变长编码、LZW 编码、位平面编码和无损预测编码等。JBIG 压缩算法基于算数编码方式，其属于变长编码方式的一种。

2.1.2.2 有损压缩编码

有损压缩是利用人们对图像某些频率成分不敏感的特性，允许压缩过程中损失一定信息。图像解压后虽然不能恢复原始数据，但是所损失的部分对理解原始图像的影响小，而增加了更高的压缩比，可以减少大量存储空间。由于人眼对于光线比较敏感，所以对于图像在屏幕上显示对人的感官影响不大。但是对于高分

辨率打印机打印来说，图像质量就会有明显的受损痕迹。

常见的有损压缩编码包括有损预测编码、变换编码、小波编码等。

2.2 JBIG 压缩算法

2.2.1 JBIG 压缩算法概述

JBIG 是联合二值图像专家组（Joint Bi-Level Image Experts Group）的简称。JBIG 是一种最初为传真传输而提出的二值图像无损压缩算法，其算法编码过程思想来源于算术编码，是一种基于复杂数学概念以及图像像素点上下文关联的压缩算法[11]。JBIG 现在应用于压缩二值图像和灰度图像，对文档图像有较高的压缩率。在打印机半色调过程保证二值化图像质量的前提下，JBIG 也可适用于原彩色和灰度级图像的压缩。由于是无损压缩，数据解压缩后的图像与压缩前的图像完全相同，对于边沿、线条、文字敏感的图像，不会出现模糊和缺失。JBIG 标准对于半色调图像压缩比可以达到 8:1，对于打印字符的扫描图像压缩比可提高 1.1~1.5 倍，对于计算机生成的打印字符图像，压缩比可提高约 5 倍，对于用抖动或半色调表示的图像压缩比可提高 2~30 倍[12]。

使渐进式编码与顺序编码相兼容，是 JBIG 标准一个十分有用的特征，也是那些为不同分辨率显示设备服务的数据库应用的主要兴趣所在。渐进式编码用于分辨率可变的显示很有效，但对于硬复制输出就不需要那些低分辨率的中间层，而是希望解码分辨率“一步到位”[13]。渐进式编码方式如图 2.1。其中 I_D 表示图像分辨率层，$C_{s,d}$ 表示压缩后的数据。一个原始图像通过逐层递减的方式降低图像的分辨率，每一个降低后的分辨率层称为“差异层”，降低后的最后一个分辨率层称为“最低分辨率层”。压缩过程对每一分辨率层分别进行压缩、存储。而通过不同分辨率要求的操作可以选择相应层级的压缩数据进行存储或传输，而舍弃更高层级的值，从而达到进一步减少存储空间以及缩短传输时间的目的。

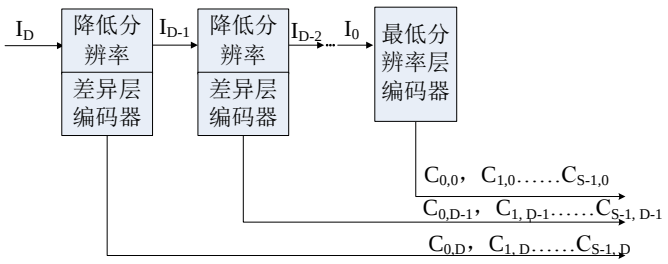

图 2.1 渐进式编码方式图

类似的，解压缩过程从最低分辨率层开始解压缩，通过逐层递增方式恢复数据产生相应差异层，最终恢复出原始图像，如图 2.2。特别对于可显示的输出设备

来说，当某一图像解压缩过程所恢复的图像分辨率已经满足使用需求时，即可立即停止解压缩过程，不再进行接收压缩值以及恢复图像的操作，可以缩短实际解压缩过程的时间。

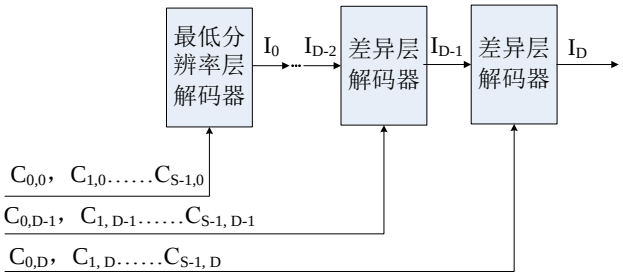

图 2.2 渐进式解码方式图

顺序式编码方式编码时不需要低分辨率的中间层，而是直接对单一分辨率层像素进行压缩处理，解压时也只进行一层图像的恢复。其相对于渐进式编码方式，应用于分辨率需要完全恢复图像、压缩解压过程速度快、压缩率相对次要的压缩情况下。两种编码方式的兼容性体现在渐进式编码方式对应的解码器可以用于读取顺序式编码方式所产生的压缩图像，而顺序式编码方式对应的解码器只能读取渐进式编码方式所产生的最低分辨率层压缩图像。

2.2.2 压缩数据结构及存储数据结构

JBIG 压缩算法的渐进式编码方式对于两个相邻的分辨率层虽然相互独立编码，但是各层的预测模块却同时需要使用这两层数据。在编码分辨率相对较低的层级时仍然要保留高一分辨率层级的所有数据，这就需要一个存储空间庞大的缓存。为了避免这样的存储空间耗费，JBIG 建议规范中引入了一条规定：将整幅图像细化分为若干条带（Stripe），每一条带又由若干像素行组成。在处理时按照条带为单位进行处理，各条带分别被编码、传输。存储每一个条带所包含的数据量远小于整幅图像的数据量，从而可以减少大量存储空间的使用。一个具有三层分辨率层每层具有三条条带的二值图像数据结构示意图如图2.3所示。其中 dpi（Dots Per Inch）是分辨率的单位，表示每英寸所打印的墨点数。d 表示分辨率层数，s 表示条带数，条带中的数字代表条带编号。

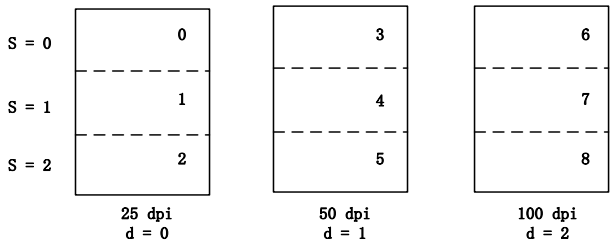

图 2.3 二值图像数据结构示意图

对于灰度图像，因为灰度图像使用一个小区域内打印点不同个数表示每一个像素点，每一个像素点所包含的数据不再是 1bit 而与其可达到的灰度级有关。对一幅用多个比特表示其灰度值的图象来说，其中的每个比特可看作表示了 1 个二值的平面，如图 2.4 所示。其中 p（plane）表示位平面[14]。一张 8bit 图像可以视为有 8 个 1bit 的图像的叠加。JBIG 压缩算法对这些位平面分别进行处理，这种位平面编码技术能减少编码和像素间的冗余。

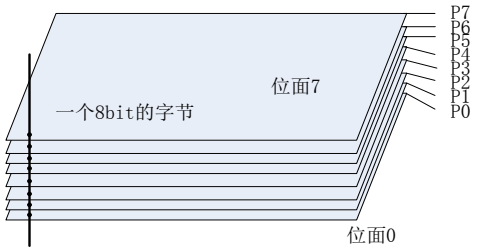

图 2.4 图像的位平面表示和位面图

一个具有三层分辨率层每层具有三条条带的灰度级为二的图像数据结构示意图如图 2.5 所示。该图像被分为 18 条条带，例如 09 条带位于分辨率为 50dpi，第 0 层位平面的第 0 条带的差异层上。

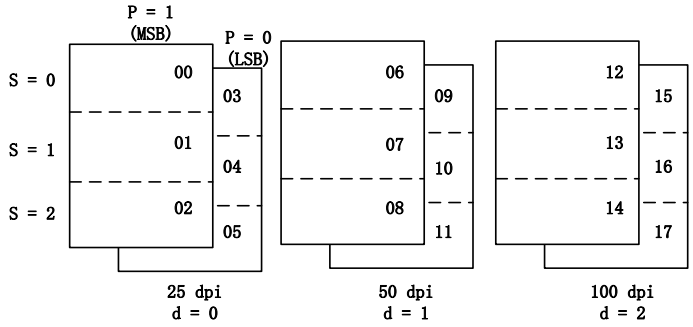

图 2.5 灰度图像数据结构示意图

由于JBIG压缩算法引入了条带概念,压缩过程再不直接对整幅图像进行压缩，而是以条带为单位进行分别处理。压缩后的数据形式也不再是连续的整幅图像的压缩值，JBIG 建议规范也引入了 JBIG 压缩后的数据格式。表 2.1 是 BIE (Bi-level Image Entity)二值图像压缩数据实体，其可分解为 BIH(Bi-level Image Header)二值图像数据包头和 BID（Bi-level Image Data）二值图像数据。

表 2.1 二值图像压缩数据实体（BIE）

BIE	
BIH	BID
字节值为变值	字节值为变值

BIH 中包含了像素点行数、列数、位平面个数、条带规模等图像基本信息，包括了自适应模块偏移量、条带存储顺序以及各预测模块选择等控制信息，还有

使用的个人确定性预测查找表。BIH 的作用就是在解码时，设置解码器的相关参数，实现正确的解码过程，如表 2.2。

表 2.2 二值图像压缩数据包头（BIH）

	BIH										
标志	Dl	D	P	-	Xd	Yd	L0	Mx	My	Order	Options	DPTABLE
字节	1	1	1	1	4	4	4	1	1	1	1	0 or 1728

BID 中包括多个浮动标志段以及多个 SDE（Strip Data Entity）条带数据实体，如表 2.3。浮动标志段对应多个不同控制字节，用于自适应像素 AT 的设定、新长度的修改以及加入个人注释等方面的操作。SDE 中包含了条带压缩后的数据 PSCD(受保护条带数据实体)和压缩此条带时的状态信息。在解压时通过各种控制信息和 PSCD 数据重建原始图像。

表 2.3 二值图像压缩数据（BID）

BID						
浮动标志	SDE	浮动标志	SDE	…	浮动标志	SDE
变长	变长	变长	变长	…	变长	变长

2.2.3 压缩、解压缩过程

差异层分辨率降低以及编码压缩过程如图 2.6。压缩过程包括分辨率降低模块(RR)、典型预测模块(TP)、确定性预测模块(DP)、自适应模块(AT)、上下文关联生成模块(CX)以及自适应算数编码模块(ENCODER)。压缩过程首先将接收的数据降低一层分辨率并连同高分辨率的数据一并存储。各个模块分别生成典型预测值（TPVALUE）、确定性预测值(DPVALUE)以及虚拟像素（LNTP）等传输给编码器。最后编码器通过判断并进行相应的处理。

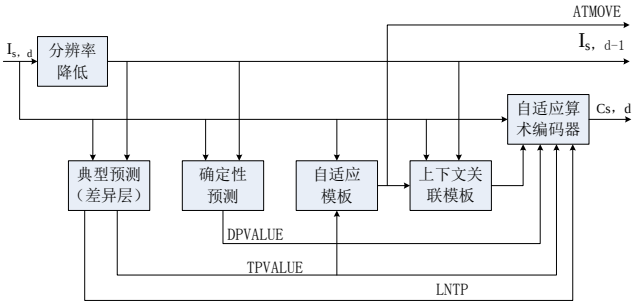

图 2.6 差异层压缩过程

最低分辨率层压缩过程如图 2.7。压缩过程包括典型预测模块、自适应模块、上下文关联生成模块以及自适应算数编码模块。输入值为上一差异层分辨率降低后的像素。其实现过程与差异层压缩相同，但对应的模块所使用的模板不同。典

型预测模块生成虚拟像素 SLNTP，上下文关联模块使用的模板有两行或三行模板之分。

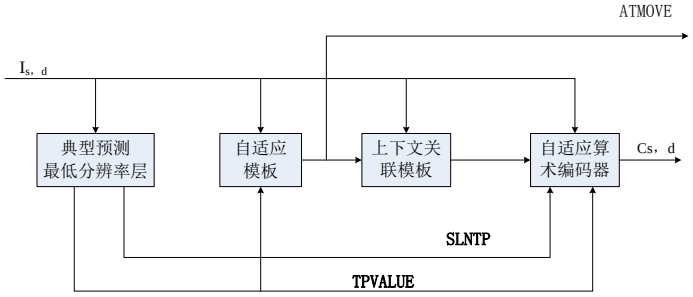

图 2.7 最低分辨率层压缩过程

解压缩过程可视为压缩过程的逆过程。由于差异层在压缩过程中，只对不可预测的值进行了压缩，而可预测的值在解压缩过程中可以通过各预测模块直接恢复，所以在解压前需要预先生成这些预测值。解压过程需要从最低分辨率层开始，逐层递增的顺序解压。

最低分辨率层解压过程如图 2.8。解压缩过程包括自适应算数解码器模块、典型预测模块和上下文关联生成模块。自适应算数解码模块首先将压缩值中恢复出虚拟像素 SLNTP 值，然后生成 TPVALUE 值与目标像素的上下文参考，最后再通过解码器恢复出第一个解压值。生成的每一个解压值都需要反馈到上下文生成模块以及典型预测模块中。

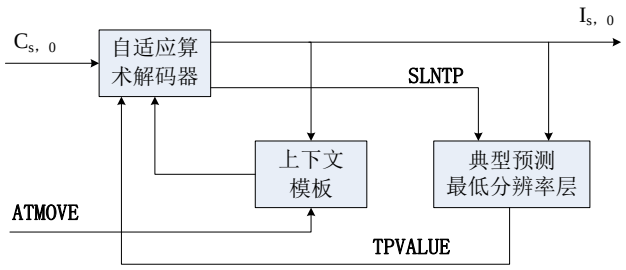

图 2.8 最低分辨率层解压缩过程

差异层解压缩过程如图 2.9。包括上下文生成模块、典型预测模块和确定性预测模块。其解压缩过程与最低分辨率层相同。

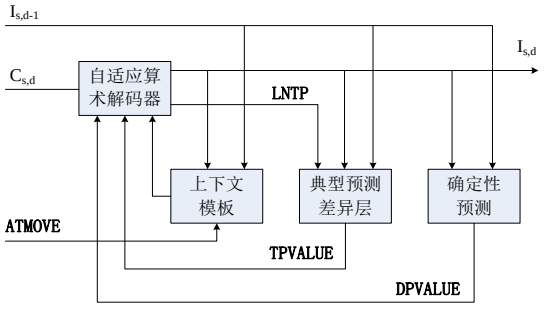

图 2.9 差异层解压缩过程

2.2.4 分辨率降低(RR)

分辨率降低算法有许多种，JBIG 压缩规范中建议使用的一种方法是通过模板线性求和计算出一个值，将此值与阈值比较从而决定降低后分辨率像素的颜色。通过一个查找表存储所有比较结果，模板如图 2.10。JBIG 的分辨率降低方法经过精心设计和广泛测试，对于文本、绘图、半调的灰度以及其它类型的图像都产生了极好的结果[15]。

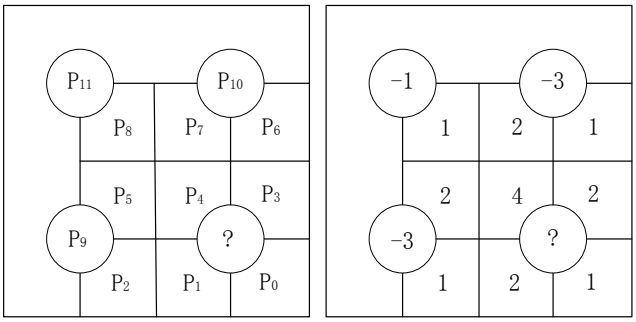

图 2.10 相关像素权值示意图

图中所示“？”为待确定的目标像素，它由周围 9 个高分辨率像素以及 3 个低分辨率像素共同决定。其中数字表示该像素对目标像素影响的权值。可由公式：

$$4P_4+2(P_7+P_5+P_3+P_1)+(P_8+P_6+P_2+P_0)-3(P_{10}+P_9)-P_{11} \qquad (2\text{-}3)$$

来决定目标像素的颜色。假设前景色（1）与背景色（0）是等概率的，且各个像素相互独立，那么通过公式计算出的值大于阈值 4.5 时，目标像素视为前景色赋值为 1，否则视为背景色赋值为 0。

分辨率降低过程使用 12 个点共同决定目标像素的颜色值，使用图 2.11 中所示分辨率降低层模板可以产生 4096 种可能并存储在查找表中。在分辨率降低处理时，将原图像与模板相比较，产生一个读地址，并通过查表的方式直接读取目标像素降低后的值。

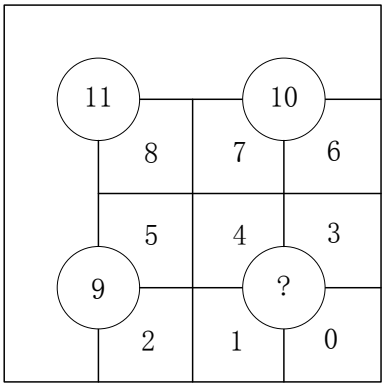

图 2.11 分辨率降低模板

2.2.5 典型预测(TP)

典型预测设计的目的是为了加快整个压缩过程，但在应用中这种预测确实能够提供更大的编码增益。典型预测利用低分辨率固定颜色区域对目标像素进行预测，当这一相同颜色区域内所包含的高分辨率颜色与之相同时，目标像素即可预测，那么就不再需要进行确定性预测、上下文关联值得生成以及自适应算数编码。一方面减少了压缩时间，另一方面又加大压缩率。对于文本格式以及线条图像，典型预测一般可以减少 95%的像素编码，而对于二值图像处理后所存储的压缩值也会明显减少。

对于差异层典型预测，需要使用目标像素的 8 个八临域内的低分辨率像素以及 4 个四临域内的高分辨率像素，如图 2.12。若一行数据中有一个或多个目标像素的八临域值相同且四临域值不同，则称目标像素对应的两行分辨率层行为非典型行，虚拟像素 LNTP 为 1，表示不可进行典型预测，所有输出值 TPVALUE 均为 2。在 LNTP 为 0 的情况时还需要进一步比较，目标像素与八临域内的像素值相同则 TPVALUE 直接赋值此目标像素的值。若颜色不完全相同，则 TPVALUE 赋值为 2，表示不可预测。

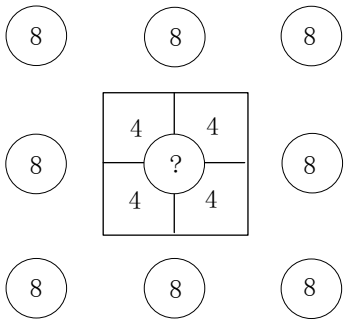

图 2.12 差异层典型预测模板

对最低分辨率层典型预测，需要使用目标像素所在行以及其上一行两行的所有像素。若当前行的所有像素与上一行位置对应的像素颜色都相同时，对预测结果 $LNTP_y$ 赋值为 0，否则赋值为 1 表示当前行不可预测。通过公式：

$$SLNTP_y = !(LNTP_y \oplus LNTP_{y-1}) \tag{2-4}$$

计算出最低分辨率层虚拟像素 $SLNTP_y$ 的值，$LNTP_{y-1}$ 为上一行的预测结果。解压时需要使用公式：

$$LNTP_y = !(SLNTP_y \oplus LNTP_{y-1}) \tag{2-5}$$

恢复出每一行 $LNTP_y$ 值。

生成的虚拟像素需要添加到原始图像行的首像素前，位置见图 2.13。LNTP 与

SLNTP 值在算数编码时，和像素一样处理，并将信息保存在压缩数据中，待解压相应行像素时使用。

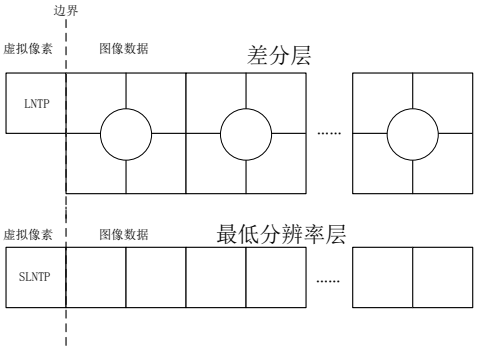

图 2.13 虚拟像素添加位置

2.2.6 确定性预测(DP)

确定性预测模块可以增大压缩率。对于文档格式的图像一般均可增加 7%的压缩增益。对于当前正在压缩的高分辨率层的像素，可能会出现此像素对于编码器与解码器来说都是可推断的情况，此时就可利用与之相关联的像素对其预测。确定性预测使用查找表的方式生成预测值，以减少算数计算。

确定性预测模块引入了相位概念如图 2.14。由于每两行高分辨率层像素可以产生一行的低分辨率层像素，所以对应每一个低分辨层像素，其四临域内的高分辨率像素可以分为 0、1、2、3 四个相位。

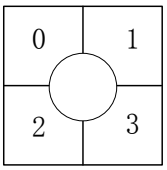

图 2.14 相位图

将图像像素与确定性预测模板相比较见图 2.15，产生一个读地址，然后对各相位表进行读操作，读取的值 2 则表示目标值不可预测，DPVALUE 赋值为 2。若为 0 或 1 则表示可预测，DPVALUE 赋值为相应值。

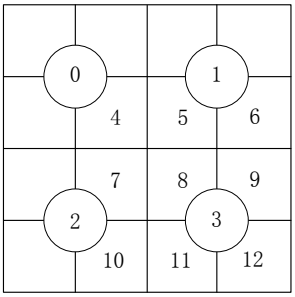

图 2.15 确定性预测模板

四个相位使用相同的预测模板，但是所参考的像素各不相同，所以每一相位都使用一个独立的查找表。相位与参考像素关系见表 2. 4。

表 2. 4 相位参考像素表

相位	目标像素	参考像素	默认表可预测像素数
0	8	0,1,2,3,4,5,6,7	20
1	9	0,1,2,3,4,5,6,7,8	108
2	11	0,1,2,3,4,5,6,7,8,9,10	526
3	12	0,1,2,3,4,5,6,7,8,9,10,11	1044

2.2.7 上下文关联值(CX)

JBIG 码采用的是一种基于对条件概率连续性预测的算术编码方法。为了确定条件概率，建议 T.82 描述了基于有限个像素点的上下文参考模板。通过图像与模板对齐、比较产生的上下文值将用于算术编码。算数编码器在对目标像素编码时，需要使用像素的条件概率估计。二值图像仅有 0、1 值，仅对目标像素进行条件估计只能得到极少的像素间关联信息。所以算数编码器对目标像素的上下文关联值保持一个给定的条件概率估计，从而代替对目标像素的条件概率估计，能够更好的反应目标像素与周围关联值得关系，从而达到加大压缩率的目的。

差异层上下文关联模块四相位模板如图 2.16。图中 A 表示自适应像素点，其位置是自适应像素初始化位置。X 表示像素的颜色，X 表示低分辨率层的同一个像素点，其四个相位分别为 0~3 相位的目标像素，四个相位需要参考的关联像素不完全相同。模板中的像素值排列顺序在 JBIG 规范中没有规定，在实现算法时可以使用任意顺序，但压缩与解压缩过程中各相位模板使用的顺序应保持一致。

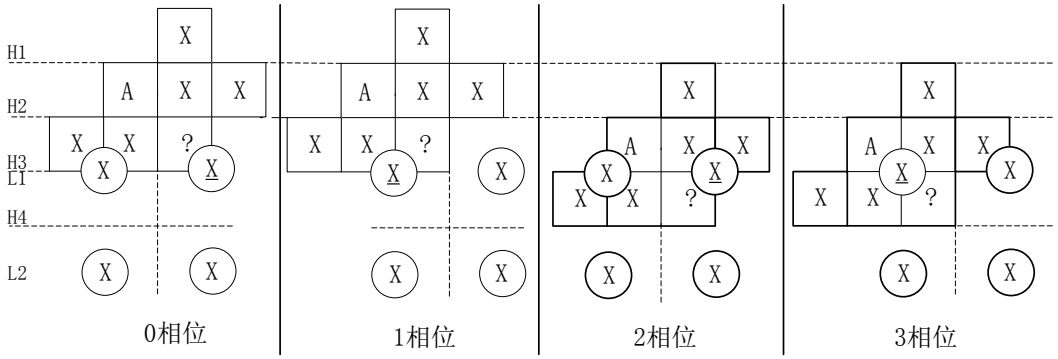

图 2.16 差异层上下文关联模板

最低分辨率层上下文关联模板如图 2.17。最低分辨率层目标像素只参考本层的像素点，其有两行模板、三行模板两种模式。

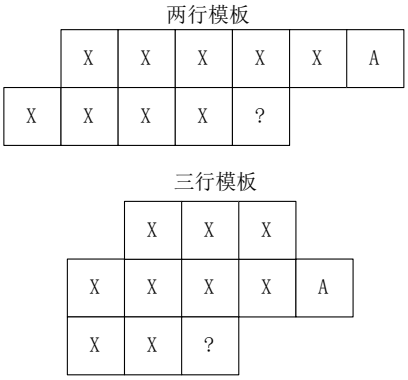

图 2.17 最低分辨率层上下文关联模板

2.2.8 自适应像素(AT)

自适应模块也可以增加编码增益，对半色调后的灰度图像有良好的效果。使用自适应像素的目的是为了寻找图像的周期性规律，并利用这种规律通过在图 2.16 与 2.17 中 A 的移动来修改模板。

图 2.18 指示了自适应像素 AT 可移动的范围。其中数字代表自适应模板像素与目标像素的水平和垂直偏移量τ_X，τ_Y。M_X，M_Y 代表矩形模板水平和垂直的规模。如果编码器要改变自适应模板像素位置，会通过 ATMOVE 标志段通知解码器，其中包括偏移量的信息，解码器接到 ATMOVE 段后会对解码器中的自适应模板做出相应的修改。对于差异层自适应模板像素的移动对四个相位是同时有效，但对于不同分辨率层之间相互独立。

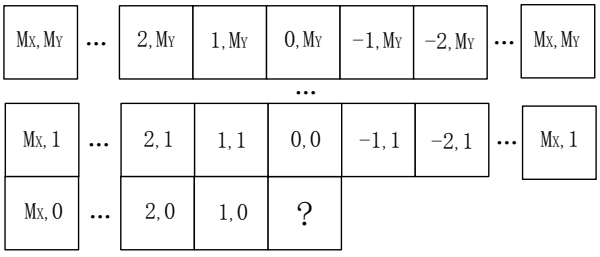

图 2.18 自适应像素可移动范围

2.2.9 算术编解码（ENCODER）

算数编码模块是一个四输入一输出的模块见图 2.19 所示。其中 PIX 为待压缩的像素以及 LNTP 或 SLNTP 值。其它三输入是当前像素的上下文关联值、典型预测值和确定性预测值。在算数编码前，编码器首先要判断目标像素是否可预测，若有 TPVALUE 或 DPVALUE 任意一个不为 2 时，则表示可预测，不需要对目标像素进行算术编码，直接开始下一个像素的处理。若反之则需通过 CX 值对 PIX

进行编码操作。输出值为压缩后的条带数据实体。

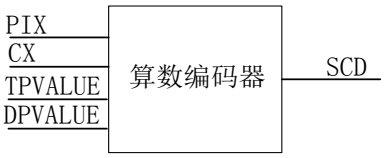

图 2.19 算数编码器模块

算数编码是一种从整个符号序列出发，采用递推形式连续编码的方法。其基本思想是把信源序列累积概率映射到[0，1)区间，确定每一序列对应区间内的一个点，即是一个二进制小数。这些点把[0，1)区间分割成了许许多多的小区间段，每区间段的长度与某一序列的概率相匹配，编码时的输出就是所选取的区间段内的一个二进制小数，这样就使某一序列与相应的一个二进制小数建立起一一对应关系；且该码是唯一可译的。由此使平均码长接近于信源熵，从而实现图像压缩编码[16]。

图 2.20 是一个{0，1，0，0}序列编码划分[0，1）区间的示例。图中 MPS（More Probable Symbol）表示大概率符号，LPS（Less Probable Symbol）表示小概率符号。约定将大概率符号划分区间始终处于小概率符号划分区间的下方。需要指出的是，对于有先验知识的编码方式，MPS 或 LPS 可能固定对应 0 或者 1。如示例序列，0 为大概率符号，1 为小概率符号。但是对于自适应的算术编码方式，MPS 与 LPS 的值可能在 0、1 值出现概率情况改变的情况下而互换。

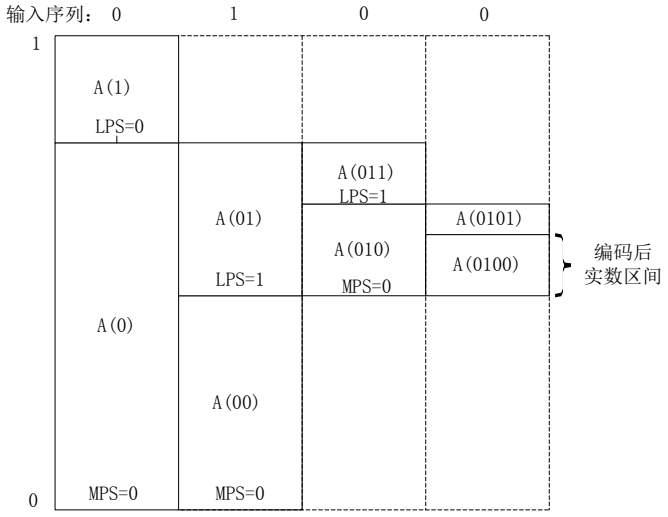

图 2.20 算数编码示例

依照符号序列输入顺序，编码器依次根据概率大小重新划分新的实数区间。区间划分根据公式：

$$A_{n+1} = p_x(x_n) \times A_n \tag{2-6}$$

$$C_{n+1} = C_{n+1} + P_x(x_n) \times A_n \tag{2-7}$$

$$P_x=\begin{cases}0 & ,MPS\text{编码}\\ 1-p_x(x_n), & LPS\text{编码}\end{cases} \tag{2-8}$$

其中是 A_n 是区间长度，p(x)为输入符号的概率，C_n 是概率区间的下限，$Px(x_n)$是累积分布。A_0 初始化为 1，C_0 初始化为 0。在编码序列第一个符号 0 时，假设 p1(0)为 0.8，由于编码 MPS，所以新区间长度为 A_1=0.8，新区间下限不变 C_1=C_0=0。第一次编码后的用区间[0，0.8）表示压缩值。对于某划分区域[C_n, C_n+A_n)来说，连续输入的信号源符号序列对应的概率区间可以进行嵌套，且每个不同的信源符号编码输出的概率区间集合并不连续，而他们的总集是[1，0)，对于每一个不同信源符号划分的子区间是唯一确定的。

在 JBIG 压缩算法规范中，使用一个 32 位宽的寄存器 A 用来存储当前压缩区间,初始化为十六进制 0x10000。一个 32 位宽的寄存器 C 用来存储区高位区间值，初始化为 0。其数据结构见表 2. 5。

表 2. 5 压缩寄存器数据结构

	最高有效位 最低有效位
寄存器 C	0000cbbb，bbbbbsss，xxxxxxxx，xxxxxxxx
寄存器 A	00000000，0000000a，aaaaaaaa， aaaaaaaa

其中 a 比特位为划分区间的小数部分存储空间，b 比特位为要代表要移除（输出）的位。s 比特位为间隔位。x 为待累加寄存器 A 的位存储空间。由于算数编码使用一个小数表示压缩值，但是当编码位数足够长时，表示区间的实数就要有足够的长的精度来保存压缩值。为了避免因编码精度而影响编码值，所以引入了归一化概念。寄存器 A 保存划分区间小数部分值，当它小于十六进制值 0x8000 时，寄存器 A 需要左移操作，即乘 2 操作。使其范围始终保持在区间[0x8000，0x10000)内。采用归一化后，寄存器 A 始终只存储小数部分末尾值，而相同的高位不再参与处理，解决了精度问题。由于寄存器 C 需要与寄存器 A 小数部分累加，所以当寄存器 A 左移操作时，寄存器 C 也要做同样处理，保证不出现错位累加。

在公式 3-5 和 3-6 中计算包含了乘法运算，在硬件实现中乘法运算的代价是很高昂的。所以 JBIG 压缩算法又引入了一个近似算法替代，公式：

$$p\times A\approx p\times\overline{A}=LSZ \tag{2-9}$$

其中 p 是当前 LPS 的概率。$\overline{A}$是平均值，用这个近似乘积项 LSZ 代替实际区间的划分。JBIG 压缩规范将 LSZ 所有可能生成一张查找表 PET(Probability Estimation Table)，这样利用查表的方法，就能快速而又节省硬件资源的实现子区间的划分。因为 A 的取值范围固定在[0x8000，0x10000)之间，所以利用其平均值代替不会引入过多的误差。在编码 LPS 时会出现 LPS 的子区间大于 MPS 子区间的情况，应

该将两个对应区间大小进行互换，始终保持 MPS 区间大于 LPS 区间。

算数解码器模块也是一个四输入一输出的模块见图 2.21 所示。其中 SDE 为压缩后的条带数据实体，其余三输入值和编码器相同。但与编码过程不同的是，这三个值并非预产生的值，而是通过解码器模块解压后逐一生成，再反馈到解码器的输入值。在解码前，解码器首先要判断目标像素是否可预测，若有 TPVALUE 或 DPVALUE 任意一个不为 2 时，则表示可预测，不需要对目标像素进行算术解码，直接将预测值赋值给 PIX 用于输出。若反之则需通过 CX 值对 SDE 进行解码操作，输出值为解压后的像素值[17]。

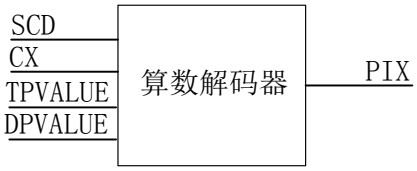

图 2.21 算数解码器模块

第三章 JBIG 压缩算法改进与 FPGA 设计

JBIG 压缩模块需要完成的工作是将输入的原始图像经过压缩处理，分别产生高低分辨两层的压缩数据进行独立输出，其实现如图 3.1 所示。整个压缩过程在同一个使能控制模块下并行处理，将半色调值通过分辨率降低模块生成新的分辨率层数据，然后对各层各目标像素利用确定性预测、典型预测和上下文关联模块预先生成所需的辅助值，在压缩时将目标像素以及相应的辅助值汇入，进行压缩处理并输出存储。

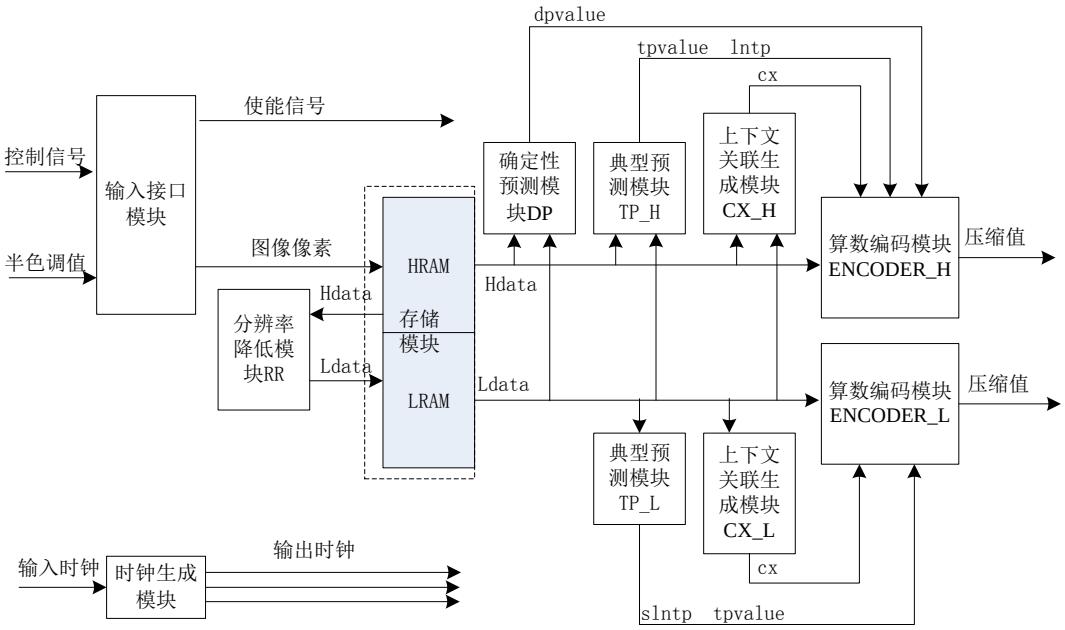

图 3.1 JBIG 压缩算法结构图

3.1 数据结构设计

JBIG 压缩算法需要将一副完整图像划分为分辨率层、位平面和条带的数据结构处理。对于本文实现的算法，需细化其存储和处理的数据结构。JBIG 压缩算法的输入值是上层半色调模块产生的二值图像数据，即一系列 0（背景色）、1（前景色）数据流，其位平面为一层。为了描述简单，本文只讨论分辨率降低一层的情况，即分为差异层和最低分辨率层两层，其中差异层也可称为高分辨率层。本文所述算法没有直接使用条带概念分割图像，而是借用了条带处理数据的思路。整体上看整幅图像就是一个条带，但是在存储方面，循环利用固定存储空间，仅保存 10 行像素见图 3.2。相对于条带分割方法，除了保留了原有节省存储空间的优势，其最大优点是有利于硬件实现并行处理。所不同的是分割条带处理后编码器保留了条带区域内的图像特征，而整幅图像编码保留了整体图像特征，对于一副

图像上下文差异较大的情况，条带分割方法会对像素出现概率有更准确的预测。实际上本文的处理方式与条带分割的方法并不冲突，如果功能需要可以很容易实现转换。

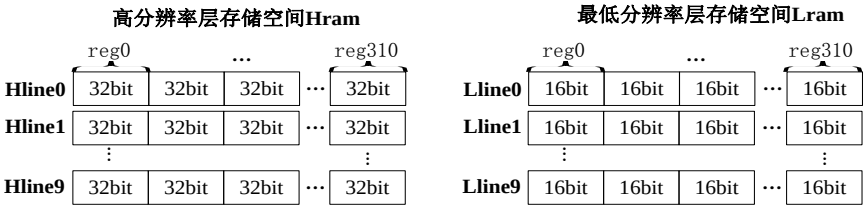

图 3.2 存储空间数据结构

根据上层半色调模块算法设计，除去半色调模块起始准备以及初始化时间以外，可以稳定每一个时钟产生 1bit（位）数据。JBIG 压缩算法的时间瓶颈在于高分辨率层的压缩过程，若使用相同的时钟对 1bit 像素值进行压缩，所使用的时间总是大于 1 个时钟。这就使 JBIG 压缩过程所使用的时间远远大于半色调过程，导致半色调过程与 JBIG 压缩过程之间需要一块很大的存储空间缓存图像数据。然而 FPGA 芯片的 Memory 资源有限，不可能提供一幅图像规模级的空间用于数据存储，这就要求上层半色调过程与 JBIG 压缩过程的时间同步。称此时钟为半色调过程与 JBIG 压缩过程同步时钟，简称同步时钟。

JBIG 压缩算法在压缩 1bit 像素前，需要对该目标像素预先计算相应的典型预测值、确定性预测值以及上下文关联值。统称以上预先计算的值均为待压缩像素的辅助值，其模块称为辅助值生成模块。所有的辅助值生成模块都需要读取同一块高分辨率层数据缓存和最低分辨率数据缓存。若以 1bit 为单位存储数据，各个模块同时读取同一数据缓存将造成竞争，而采用互斥等待读取数据，会增加额外时间从而降低压缩过程的速度，所以需要采用多比特位为单位进行存储。实现中选用 32bits 为单位存储数据。半色调过程与压缩过程内部各模块均可在 32 个同步时钟内完成 32bits 数据处理（压缩模块也满足，但所使用的时钟不同，频率要快于同步时钟），从而避免了大量数据缓存空间。采用 32bits 为单位的存储方式，一方面可以减少各个模块读取同一块分辨率层数据缓存的次数，另一方面各个模块分时读取各缓存不再影响整个压缩过程的速度，使得并行处理成为可能。

根据设计需求，打印机需要满足处理 300dpi×600dpi、600dpi×600dpi 、600dpi×2400dpi 和 1200dpi×600dpi 等不同分辨率的数据。对于标准 A4 纸张 210mm×297mm 的尺寸，一行中最多将含有 9922 个像素点。以 32bits、16bits 为单位存储的 HRAM、LRAM，都最多需要 311 个 reg（32bits 或 16bits）组成一行数据。公式：

$$Add = i + j \times 311 \tag{3-1}$$

$$i=\begin{cases} i+1, i \neq Reg \\ 0\ , i = Reg \end{cases} \tag{3-2}$$

$$i=\begin{cases} j+1, \quad j[3]=0 \\ 0\ , j[3]=1\&j[0]=1 \end{cases} \tag{3-3}$$

用于产生对存储空间的读地址，其中 i 为 Reg 的索引，Reg 为当前图像一行像素所需要的 reg 数目。j 为行号索引，“[ ]”为位操作符，其意义是当 j 累加到 9 时，对 j 进行赋 0 操作。这样操作是为了满足 10 行存储空间的循环利用，同时使用位操作符和条件判定语句实现了求余操作，避免引入除法对资源的耗费。

3.2 并行处理

广义地讲，并行性有两种含义：一是同时性，指两个以上事件在同一时刻发生；二是并行性，指两个以上事件在同一时间间隔内发生[18]。JBIG 压缩算法的并行处理体现在两方面，一方面 JBIG 压缩模块与上层半色调模块的并行处理，主要目的是为了减少存储空间。另一方面，压缩模块与各个辅助值生成模块的并行处理，目的是为了加快压缩速度。

3.2.1 半色调与压缩模块并行处理

由于半色调模块与压缩模块的速度不匹配，为了减少大量的缓存空间，需要对半色调过程与压缩过程进行并行处理，如图 3.3 所示。L 表示行（Line），y 表示高分辨率层的总行数，y/2 表示分辨率降低后的总行数。半色调过程开始产生第一个 32bits 数据后使能 JBIG 压缩模块，压缩模块开始预存储高分辨率层数据，在开始存储第三行数据的时，低分辨存储模块才开始并行存储处理后的数据。

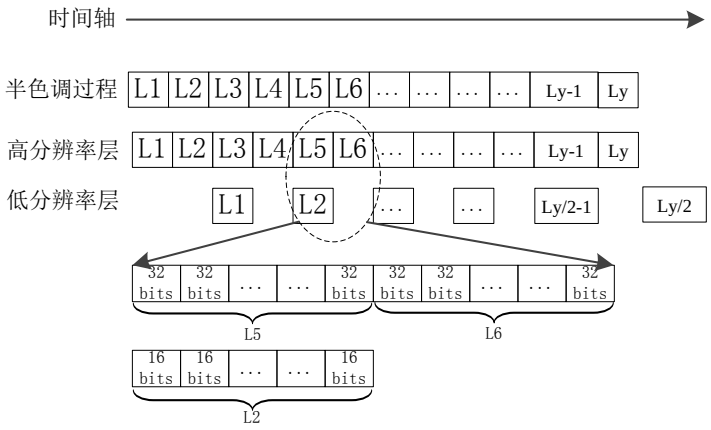

图 3.3 半色调模块与压缩模块数据存储并行处理过程

3.2.2 压缩过程并行处理

JBIG 压缩过程各模块间并行处理的设计，其目的是缩短压缩过程的时间，加快整个系统的运行速度，如图 3.4。其中大写字母表示的模块在该时间间隔内连续处理整行数据，小写字母表示的模块在该区间内间隔处理，每次间隔处理 32bits 数据。

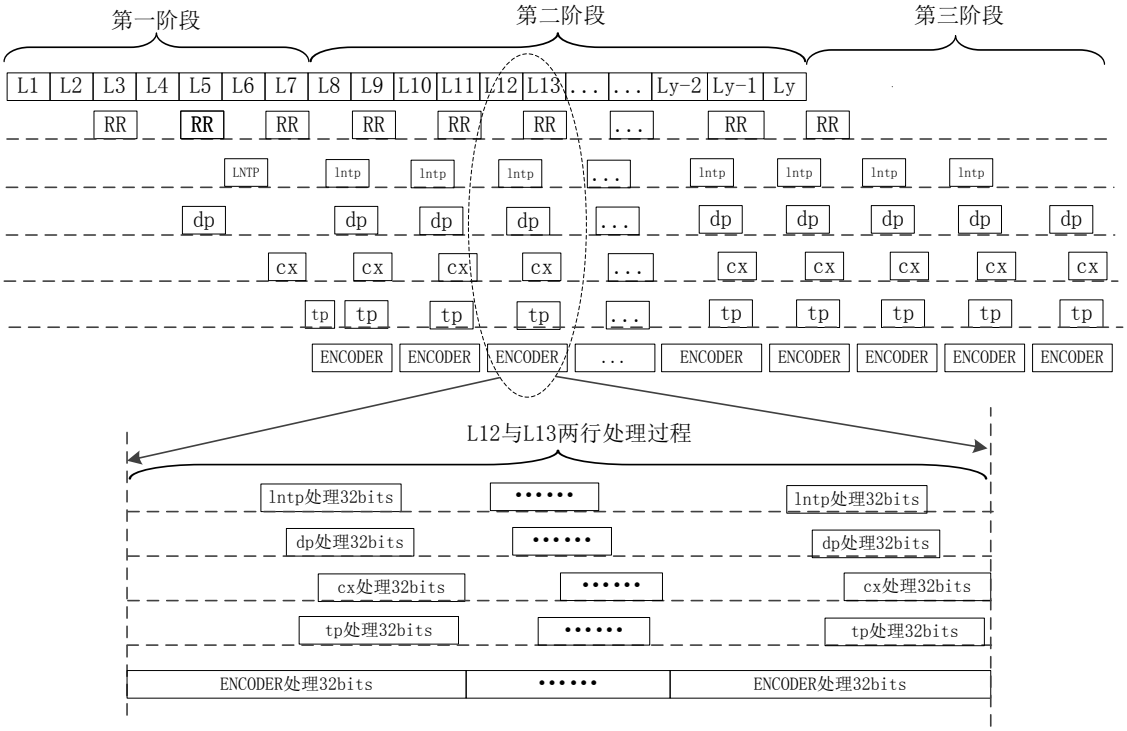

图 3.4 并行处理过程

JBIG 压缩过程总共可以分为三个阶段。第一阶段，高分辨率层预先缓存半色调模块输入的二值图像。当高分辨率层接收了 L1，L2 两行数据后才能满足分辨率降低的初始操作。在设计上当高分辨率层接收 L3 行数据时，将并行处理降低分辨率模块。降低分辨率模块每次使用两行新接收的高分辨率像素用于地址映射（模板使用三行，还包括上一行已经使用过得相关联行），并在 L3 行接收的时间区间内完成对 L1、L2 行的分辨率降低过程，所以 RR 模块间隔执行。类似的，其余模块均利用各自模板或算法特性，在存储的数据可满足其起始条件后开始并行执行，图中各模块的起始位置就是设计中各自起始时间。根据各模块可以启动时所需存储的高分辨率层数据行数，可以确定其存储空间的大小为 10 行。

由于 RR 模块在严格的时间周期下运行，如果 DP、CX、TP 模块以 RR 模块作为同步对象，虽然同步后时序控制简单，但由于各个模块对每一个高分辨率层像素点均分别产生一个 2bits 的确定性预测值、一个 10bits 的上下文关联值和一个 2bits 的典型预测值。存储两行高分辨率像素点的所有辅助值，将使用等同于 28 行

高分辨率层数据的存储空间，这是一个十分庞大的存储量。为了优化实现，将不采用 DP、CX、TP 模块与 RR 模块同步的方式。在第一阶段期间，DP、CX、TP 模块只进行一组 32bits 数据的处理，而暂缓处理其它已经满足处理条件像素的辅助值计算，以减少存储空间的损耗。

第二阶段，ENCODER 模块开始并行运行。同样采用 32bits 为单位处理数据，并在 L8、L9 时间内完成对 L1、L2 两行高分辨率层数据的压缩过程。进入第二阶段，各模块全部稳定并行处理，其中 RR 模块、ENCODER 模块面向高分辨率层数据存储过程同步，其余 LNTP、DP、CX、TP 模块面向 ENCODER 模块同步。如图 3.4 中扩展部分，ENCODER 模块执行时间最长，在压缩一组 32bits 像素后，即连续读取下一组数据进行处理。虽然 LNTP、DP、CX、TP 模块执行时间各不同，但均小于 ENCODER 模块的执行时间，在这些辅助值生成模块执行后暂停数据处理，等待压缩模块的同步信号后才重新开始并行操作。

各类辅助值生成模块面向 ENCODER 模块同步方式的设计，一方面实现了各模块间的并行处理，使整个 JBIG 压缩过程除去第一阶段初始化以及预处理所需的时间外，所有的时间均只用于第二、三阶段的数据压缩过程。并行处理将每一个待压缩像素的辅助值预先生成并存储，压缩时只需读取缓存即可，大量减少了串行生成辅助值所消耗时间的代价。另一方面这种同步方式还解决了预存储数据空间耗费问题。所有辅助值生成仅需存储 32bits 为单位的一组数据，例如 CX 模块产生的 2×32(CX 同时处理时需要两行高分辨率层像素值)个像素对应的上下文关联值需要 2×32×10bits 即 640bits 的存储空间，比面向 RR 模块同步方式所需 2×10 即 20 行高分辨率层存储数据的空间小了许多，极大的减少了 FPGA 的资源损耗。同时这种同步方式还解决了多个模块同时读取高、低分辨率数据缓存的冲突问题。由于各个模块处理相同数量的像素值所使用的时间均不相同，所以在各模块执行过程中如何保证在不同的时间读取相同的缓存是本文所述的并行处理方案的关键。所有辅助值生成模块都同步于 ENCODER 模块，并在 ENCODER 模块执行期间分时重新使能，这种周期性的执行方式，有利于硬件算法的实现，同时也使各模块间相对随机读取信号固定在特定时间区域内，保证了并行处理方案的可行。

需要注意的是第二阶段 LNTP 模块与第一阶段的执行方式不同，前者连续执行生成第一个虚拟像素 lntp 值，用于满足 TP 模块的启动条件。而第二阶段只需要在 TP 模块处理下一行前生成对应的 lntp 值即可，所以同 TP 模块一样，开始采用了面向 ENCODER 模块的同步方式。

第三阶段，上层半色调模块完成了对一副图像的处理。由于第一阶段高、低分辨率层数据预先缓存，各模块处理的像素总是前几行时间区域内缓存的值，所以此阶段只并行执行 ENCODER、LNTP、DP、CX、TP 模块，用于处理高、低分

辨率层中的剩余数据。从第三阶段可以看出，整个并行处理中，JBIG 压缩过程所用的时间比上层半色调模块所使用的时间多 7 行处理时间，达到了并行处理缩短压缩过程的目的。

为了简洁介绍过程，图 3.4 中只包含了高分辨率层数据压缩的并行处理过程，而没有最低分辨率层数据压缩过程，两者使用对应模块相同的使能信号，因此不再赘述。

3.3 流水线处理

3.3.1 流水线设计

为了减少算数编码时间损耗，对压缩过程需要实现流水处理。流水线设计是最常用的速度优化技术[19]。流水线设计是把一个逻辑操作分成几个较小的操作串联起来，并在多个高速时钟周期内完成，每个时钟周期利用寄存器锁存数据。吞吐率是衡量流水线性能指标，它是流水线单位时间内所能流出的任务数或结果数。若一个流水线中各个子过程经过的时间都是 Δt_0，满负荷后流水线每 Δt_0 个时钟都有一个结果生成，其最大吞吐率 T_{pmax} 为 $1/\Delta t_0$。但是在实际中，各个划分模块由于其各自功能不同，很难使其都在相同的时间内产生相应的结果，而不同的操作周期都会对吞吐率造成影响。根据公式：

$$T_{MAX} = \frac{1}{\Delta t_{MAX}} \tag{3-4}$$

最大吞吐率与子过程最大运行周期成反比，其性能受限于流水线子过程中最大运行周期。称这个子过程为瓶颈子过程，子过程处理时间越长，吞吐率越小，性能越差。所以如何合理的划分各个模块对流水线技术至关重要。假设原受限系统时钟周期为 T，使用流水线技术后时钟周期为 T'。显然，流水线技术可以保证 T'<T，因此系统的工作速度显然可以加快，吞吐量大。

3.3.2 流水线实现

根据 3.2 节中所述，JBIG 压缩算法使用了并行处理的方式，在进入稳定的压缩阶段，所有时间均被压缩模块占用，所以压缩模块是整个算法的瓶颈过程。在最通常情况下压缩一个像素并输出的时间为 $9\Delta t_0$，串行压缩过程如图 3.5 所示：

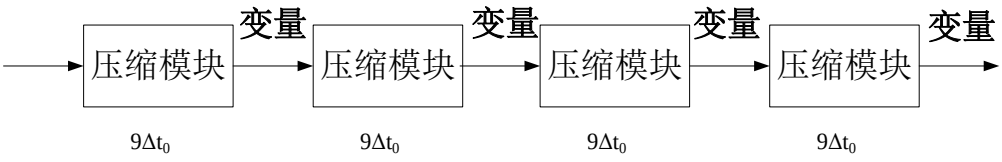

图 3.5 串行压缩过程

压缩七个像素并输出的时间为 $63\Delta t_0$，其吞吐率为 $1/9\Delta t_0$。对于这一瓶颈过程消除的最主要方法是对其进行再细分。对于压缩模块处理过程细化过程如图 3.6 所示：

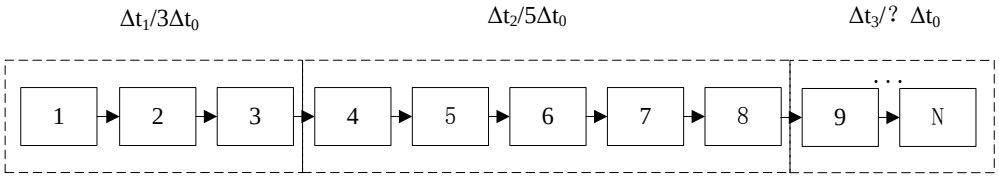

图 3.6 压缩模块细分

压缩过程可以细分为 Δt_1、Δt_2、Δt_3 三个子过程对应压缩过程中数据读取、压缩处理以及压缩值输出三个模块，因此整个 JBIG 算数编码过程可以划分为三级流水线。第一、二子过程因为根据算法内部数据的读取顺序等限制，所以设计上只能分别在 Δt_1 和 Δt_2 个固定时钟内完成。第三个子过程 Δt_3 为不确定时钟：一般情况下，压缩一个像素后并不直接输出压缩值，此时 Δt_3 为 0。在压缩值组成一个 8bits 后将以字节形式输出，此时 Δt_3 为 1。极少数情况下编码器会同时产生 2 个字节的输出值，此时 Δt_3 为 2。在有 n 个连续进位或 0xff 压缩值生成时，此时 Δt_3 为 n+1。一般情况下连续 8 个压缩过程才会产生输出一个压缩字节，所以可以认为 $8\Delta t_2$ 大于 Δt_3 的时间总成立，并且输出过程不影响压缩处理，所以将 Δt_2 过程视为新的瓶颈子过程。由于后一级压缩过程依赖上一级的输出，所以 Δt_2 过程不可再分。

流水线设计如图 3.7。流水线过程稳定后，压缩一个像素点所需要的时间相当于 Δt_2 个时钟。作为对比，流水线 1~7 层压缩这七个像素并输出的时间为 $38\Delta t_0$，其吞吐率为 $1/5\Delta t_0$，相比于串行执行显然流水线加快了压缩过程的速度。在每一层流水线第 2 段，前一级 Δt_2 过程输出值直接汇入当前级的 Δt_2 过程，保证压缩过程能够连续不断的流水进行。对于每一层的 Δt_3 过程，由于它的输出值并不影响下一个压缩像素的相关运算，所以设计上将它从压缩模块中独立出来。图中绘制了上文提到的各种 Δt_3 可能出现情况，需要特别说明的是第 6 层流水线可能出现的极端情况，当前流水线 Δt_2 过程完成准备输出时，此时上一层流水线的输出过程仍在进行，那么就要求输出模块在输出压缩值的同时具有缓存新值的能力。对于当前流水线来说，其输出可以视为被延时处理，由于其不影响压缩过程，所以流水线本身并不受影响。流水线瓶颈过程的确立、各子过程所使用的时间长度以及具体实现方案将在 3.4.8 节中详述。

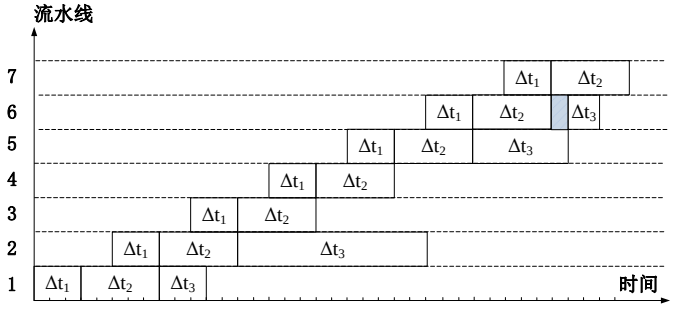

图 3.7 压缩模块流水线设计

3.4 JBIG 压缩算法实现

本文在 Quartus Ⅱ 软件平台利用 Verilog HDL 编写，并使用 ModelSim-Altera 软件进行联合仿真。Quartus Ⅱ 软件是 Altera 公司的综合开发工具，它集成了 Altera 的 FPGA 开发流程中所涉及的所有工具和第三方软件接口。ModelSim-Altera 是第三方联合仿真软件，其主要特点是仿真速度快、仿真精度高。支持包括 Verilog HDL 等多硬件语言的混合编程的仿真[20]。Verilog HDL 是一种硬件描述语言，为了实现数字电路而用来描述 ASIC 和 FPGA 的设计之用。用于算法级、门级到开关级的多种抽象设计层次的数字系统建模。一般的高级编程语言没有时序的概念，硬件语言不仅可以描述硬件电路的功能，还可以描述电路的时序[21]。

JBIG 压缩过程涉及模块较多，为了描述简单，本文所有模块仿真过程均基于以下输入数据编写测试激励，且仅给出重要模块及信号的仿真波形。此组数据具有设计特殊性，可以满足各功能模块的仿真过程。输入值是一个 8 行 96 列共 768 个像素的二值图像：

Xd=96，Yd=8；

第一行数据： 96'hf0fe_ffff_ffff_ffff_ffff_ffff；

第二行数据： 96'hfef0_ffff_ffff_ffff_ffff_ffff；

第三至八行数据： 96'hffff_ffff_ffff_ffff_0000_0000；

按照处理数据结构分析，每一行 96 个像素点，将分为 3 个 reg 进行处理。

3.4.1 时钟生成模块

根据上文流水线方式算数编码分析，每 5 个时钟可以完成一个像素的压缩，那么最快 163（5×32＋3）个时钟即可完成 32bits 像素的处理。为了与上层半色调模块同步，需要对输入时钟进行六分频。分频器是一种应用十分广泛的电路，其功能就是对较高频率的信号进行分频，其输出是根据分频常数对输出信号的高、

低电平进行控制[22]。时钟生成模块采用六分频用于生成同步时钟（clk2）。分频后 32 个同步时钟对应 192（32×6）个输入时钟，同时可以产生 32bits 数据。这一设计确保接收数据与处理数据同步，实现循环使用 10 行存储空间。

时钟生成模块由锁相环模块和六分频模块组成。随着系统时钟频率逐步提升，I/O 性能要求也越来越高。在内部逻辑实现时，往往需要多个频率和相位的时钟，于是在 FPGA 内部出现了一些时钟管理软件，最具代表性的就是锁相环[23]。锁相环模块采用 Altera 提供的 LPM（参数化）库提供的 ALTPLL 实现，其功能是产生一个与输入时钟 clk 相差 180 度相位的时钟。六分频器模块功能是产生两个相位相差 180 度且周期为输入时钟六倍的时钟。clk 时钟用于压缩模块以及面向压缩模块同步的所有模块，clk1 用于前者应用的存储空间。clk2 用于与半色调过程同步以及 RR 模块，clk3 用于 RR 查找表，如图 3.8 所示。

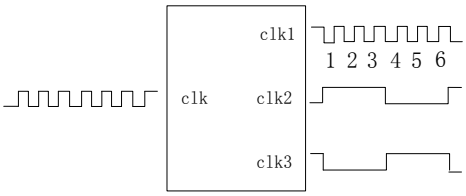

图 3.8 时钟生成模块

3.4.2 输入接口模块

为了能够并行处理，需要有统一的控制来进行各模块的使能，输入接口模块就是为了这一目而设计，如图 3.9。上层半色调模块控制将图像像素写入 FIFO 存储空间，存储空间采用 Altera 提供的 LPM（参数化）库提供的 FIFO 实现。由于采用同步处理设计，存储空间仅占 32bits×8 字节。控制模块一方面将图像行、列值和图像基本信息锁存，并在整个压缩处理过程保持稳定输出。另一方面，按照 10 行循环存储空间的设计，将输入的图像数据写入到高分辨层存储空间用于各个模块的读取。按照同步处理设计，该模块还产生各模块的使能信号，保证整个压缩过程正常运行。

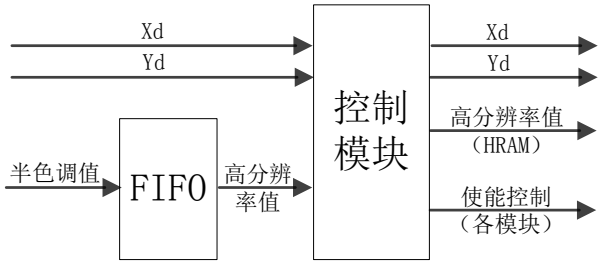

图 3.9 总控制模块

使能控制仿真波形图如图 3.10 所示。使能控制所有模块的起始操作时间，并

在运行期间保持为高有效，其中需要间隔处理的模块呈现出周期性使能的规律。按照并行处理设计，对输入像素共进行四次分辨降低，以及四次压缩处理。需要特别说明的是分辨率降低使能信号，其被使能模块前 4 个周期用于分辨率降低和 LNTP 模块的使能控制，后 2 个周期仅延时进行使能控制。

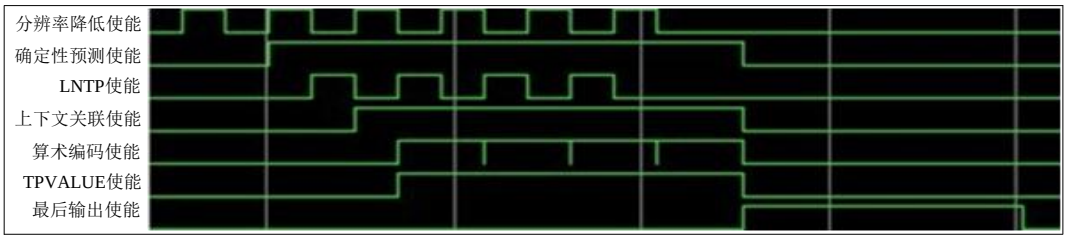

图 3.10 使能控制仿真图

3.4.3 高、最低分辨率层存储模块

高、最低分辨率层存储模块用于存储两层各 10 行分辨率数据，10 行存储空间依照 0 至 9 行顺序循环使用，满足并行处理设计的同时降低存储空间的损耗。如图 3.11 所示。其中 HRAM 和 LRAM 分别用于存储高和最低分辨率层数据，均采用 Altera 提供的 LPM 库中的 dpram 实现。HRAM 和 LRAM 分别以 32bits 和 16bits 为 1 字节存储数据，各需 4096 个字节。

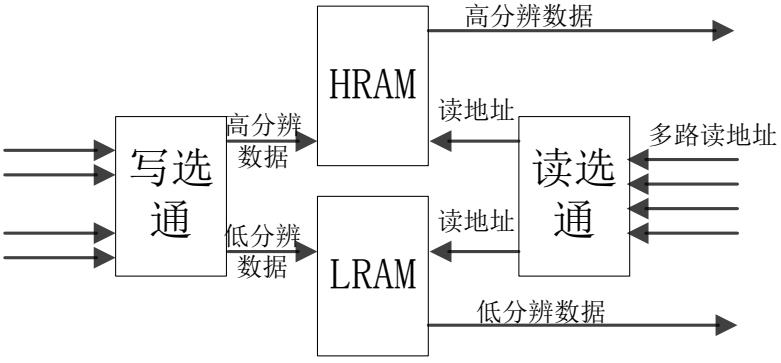

图 3.11 高、最低分辨率层存储模块

由于高、最低分辨率层存储模块 HRAM 和 LRAM 需要有多个模块分别读取和写入，但是在硬件电路设计中，每一个输入端口同一个时刻只允许有一个输入，多路同时输入一个端口会造成数据混乱。所以各模块对于 HRAM 和 LRAM 的读写地址以及读写输入信号需要分时有效，因此需要具有优先级的写路径选通和读路径选通功能模块。由于分辨率降低模块使用同步时钟，其读写周期最长，为了避免长时间占用读写地址和数据总线，需要将其选通优先级设置为最低。其它模块优先级排列仅根据其读写顺序先后排列，表 3.1 中显示了需要读写 HRAM 和 LRAM 的模块以及对应的优先级顺序，括号内文字代表模块操作的分辨率层。

图 3.1 读、写路径选通优先级表

优先级	读 HRAM	读 LRAM	写 HRAM	写 LRAM
高	算数编码（高）	上下文（高）	总控制	分辨率降低
↓	上下文（高）	上下文（低）	—	总控制
↓	确定性预测（高）	TPVALUE（高）	—	—
↓	LNTP（高）	确定性预测（高）	—	—
↓	分辨率降低	LNTP（高，低）	—	—
↓	—	算数编码（低）	—	—
低	—	分辨率降低	—	—

整个压缩设计中有两个模块需要写 LRAM 空间。分辨率降低模块写入处理后的像素值，总控制模块的写操作用于重新初始化 LRAM 空间的第 8 和 9 行。因为在每一幅图像开始处理时，首先要参考第 8、9 行中的像素值，所以这一设计目的是为了满足多幅图像的连续处理。上下文关联模块也有类似重新初始化的操作，其写路径选通设计思路相同，不再赘述。

3.4.4 分辨率降低模块

由于 JBIG 压缩处理对于各个不同的分辨率像素进行分别处理，所以对于原始图像来说，首先需要将其进行分辨率降低过程。这一处理可以视为将相邻四相位 4bits 高分辨率层像素降低为 1bit 低分辨像素的过程。将接收到的数据与降低分辨率模板比对，产生读查找表地址，从而读取相应分辨率降低后的像素。分辨率降低模块如图 3.12 所示。RR_LUT 采用 Altera 提供的 LPM（参数化）库提供的 LPM_ROM 实现，预先通过 mif 文件初始化整个存储空间，大小为 1bit×4096 字节，用于存储 12 个参考像素组合与阈值比较后的分辨率降低值。

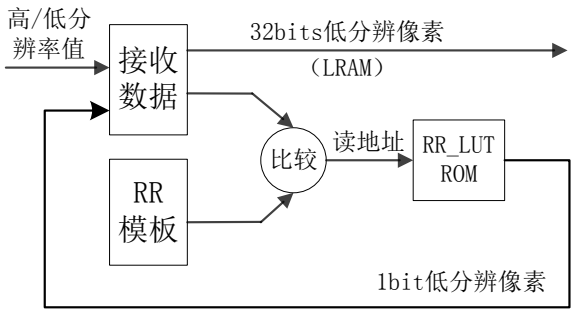

图 3.12 分辨率降低模块

分辨率降低过程首先进入空闲等待阶段，在时钟上升沿到来时采集使能信号。若使能有效则开始读取高、低分辨率层像素值，然后进行查表操作，分辨率降低

过程见状态转移图 3.13。按照数据结构设计，该模块每处理 32bits 高分辨层数据相应产生 16bits 低分辨率层数据，并且需要输出存储一次，而后再次进入空闲状态，等待使能信号重新开始下一组数据的操作。

这一硬件实现过程使用 FSM(有限状态机)进行控制。状态机使用了触发器，对于 n 个触发器而言，将会有 2^n 种可能的组合输出结果。状态机有两种构建类型，即 Moore 型和 Mealy 型。在 Moore 型中，电路输出仅是当前状态的函数。这样，当状态机在某状态中时期输出有效，而在状态转换过程中，输出无效。在 Mealy 型中，电路输出是当前状态和当前输入信号的函数[24]。实用的状态机一般都设计为同步时序电路，它在时钟信号的触发下，完成各个状态之间的转移[25]。

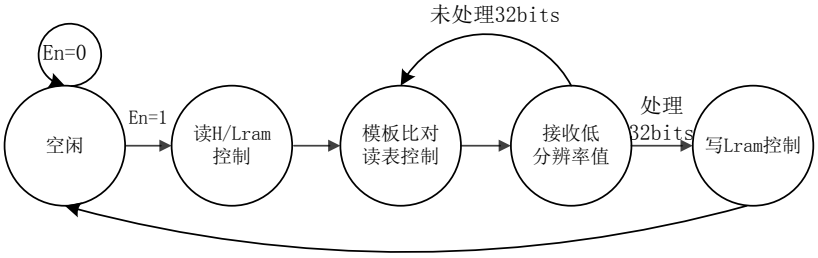

图 3.13 分辨率降低模块状态转移图

分辨率降低过程仿真波形图如图 3.14 所示。图中显示了在时钟控制信号下，读取相关像素以及生成寻址地址查找分辨率降低值的过程。输出生成值引脚输出的是低分辨值。放大图着重显示了第一组分辨率降低处理后的数据 0xeeff 的产生过程。输入激励共产生 4 行 12 组输出数据：0xeeff，0xffff，0xffff；0xffff，0xffff，0x0000；0xffff，0xffff，0x0000；0xffff，0xffff，0x0000。

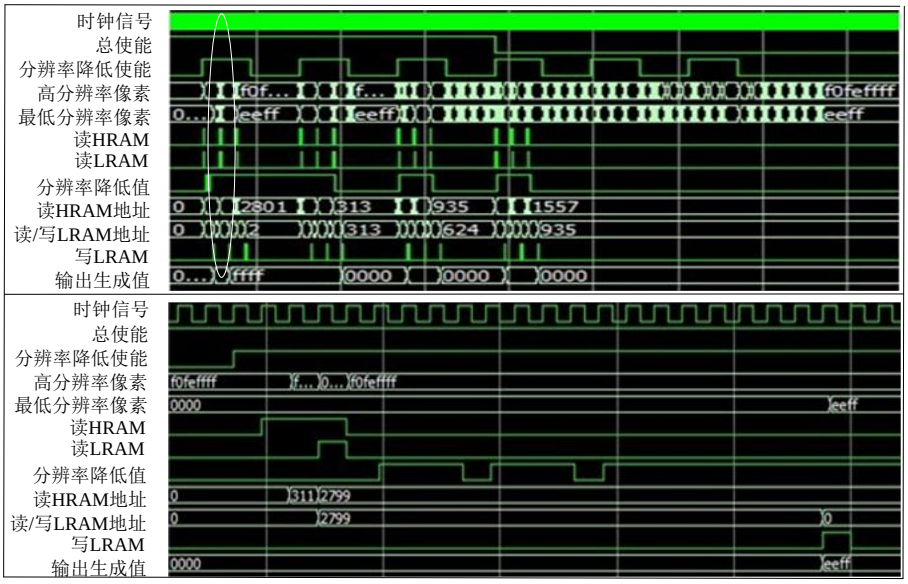

图 3.14 分辨率降低模块仿真图

3.4.5 确定性预测模块

确定性预测模块设计是通过 4 相位目标像素的 4 个不同模板产生查表读地址，从而获得 DPVALUE（确定性预测）值的过程，如图 3.15。生成值需要分别存储在 0 至 3 相位以 2bits 为单位的存储空间 DP_FIFO 中，各占 32 字节。确定性查找表也以相位区分，将预测结果存储在以 2bits 为单位各 256、512、2048 和 4096 个字节的存储空间中。

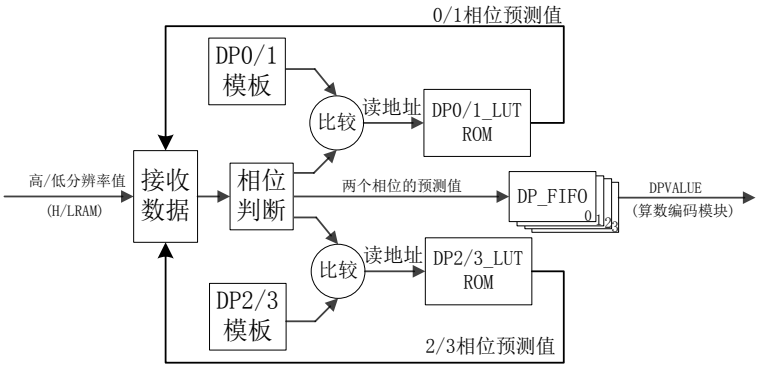

图 3.15 确定性预测模块图

确定性预测模块实现如图 3.16 所示。在使能信号到来时开始读取高低分辨像素值，然后根据 4 相位模板生成读地址，通过确定性预测查找表得到目标相位的预测值，并输出到存储模块中。每处理 32bits 数据后，该模块进入等待状态，直至时钟上升沿时检测到 restart 信号，才进入下一组数据的操作。restart 信号为面向压缩模块的同步信号，类似的设计还出现在所有面向压缩模块同步的模块中。

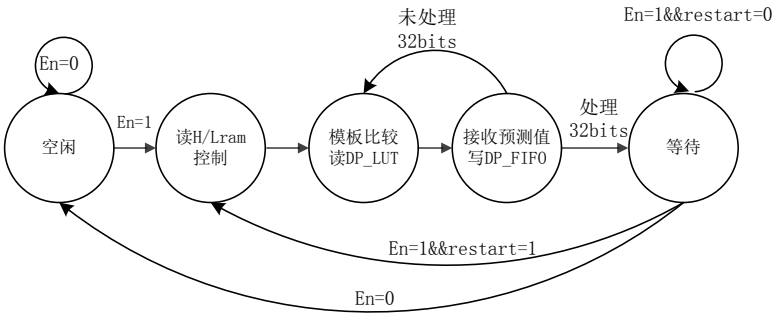

图 3.16 确定性预测状态转移图

由于计算 DPVALUE 值的算法特性，每一次操作可以同时产生 0 至 3 相位四个确定性预测值。但是压缩过程不是按照相位的排列顺序处理，而是以行的顺序进行操作，所以每次只能使用 0、1 相位或 2、3 相位所在行产生的预测值。如果同时存储 4 相位的生成值，在压缩过程处理 0、1 相位所在行时，同时还要存储 2、3 相位所在行的所有预测值，将导致耗费大量存储空间。所以本文采用每次操作仅

存储两个相位的预测值，而多增加一轮比较运算，来补偿 2、3 相位未存储的情况。类似的存储方式还应用于所有区分相位操作的模块中。

确定性预测仿真波形与放大图见图 3.17 所示。在使能与重新使能两个控制信号下，确定性预测呈现周期性处理特性。通过对 4 相位确定性预测表生成查找地址，生成确定性预测输出信号，其值由 8 位组成，分别表示 0 至 3 相位各两位的预测值输出。由图中写 0、1 相位与写 2、3 相位信号所示，预测过程交替进行，增加了一轮处理过程，用于减少存储空间。其余信号为读写各存储空间的控制与数据信号。

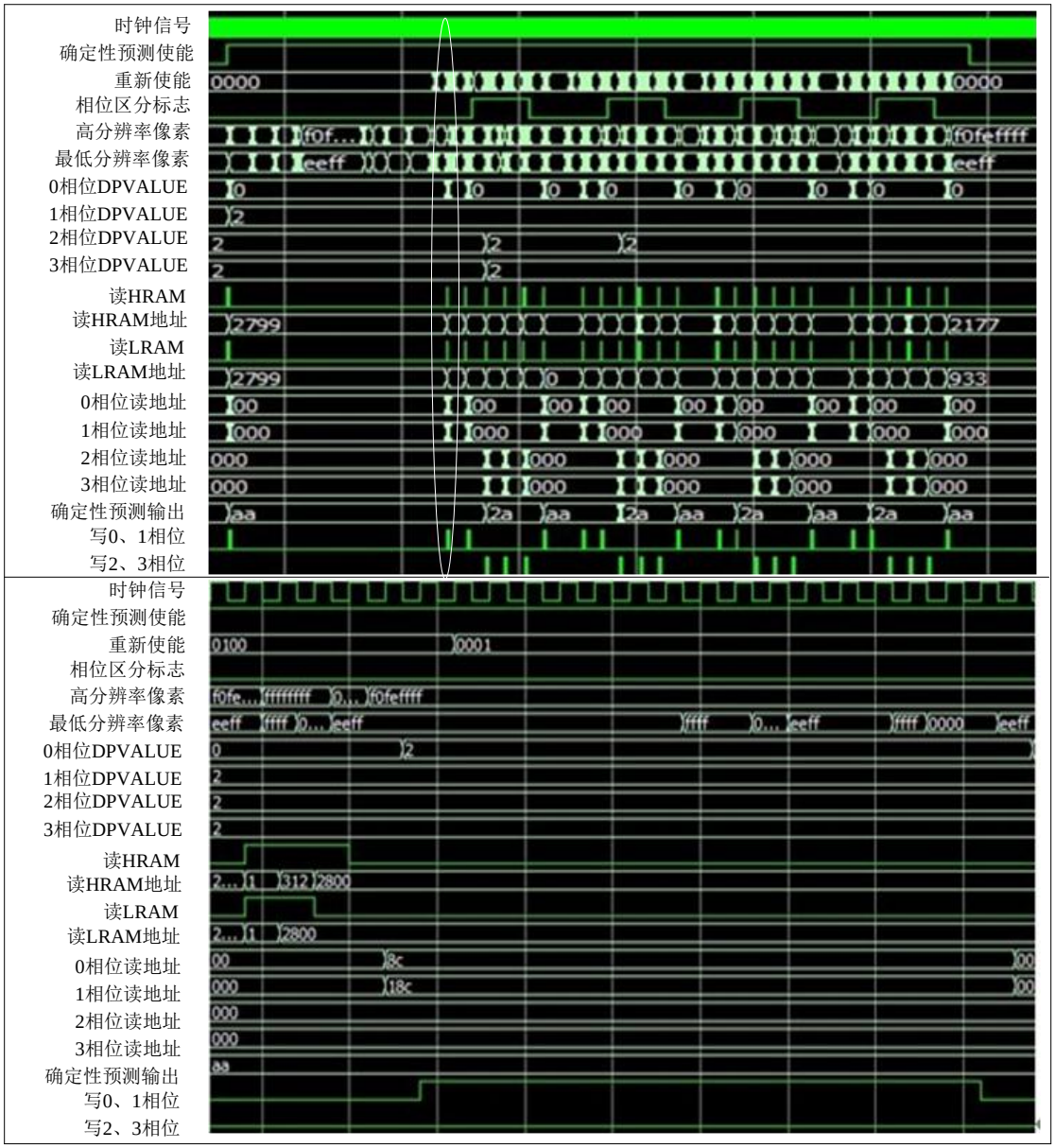

图 3.17 确定性预测模块仿真图

3.4.6 典型预测模块

3.4.6.1 高分辨率层典型预测模块

为了减少相关联像素的压缩处理，在压缩目标像素前需要进行典型预测。高分辨率层典型预测模块用于生成典型预测值，首先需要对目标像素所在行进行判断，从而得到行的预测特征 LNTP(虚拟像素)值，然后通过行的特性再对每一个目标像素进行典型值的预测，如图 3.18。所以该模块需要由两个模块 LNTP 和 TPVALUE 组成，两模块间需要有以 1bit 为单位共 4 字节的 LNTP_FIFO 存储空间用于数据缓存。由于虚拟像素值分别用于像素的典型预测和算数编码，所以需要同时使用两个相同的空间存储相同的数据。对于 4 相位目标像素的典型预测值需要使用以 2bits 为单位各 32 字节的 4 个存储空间。

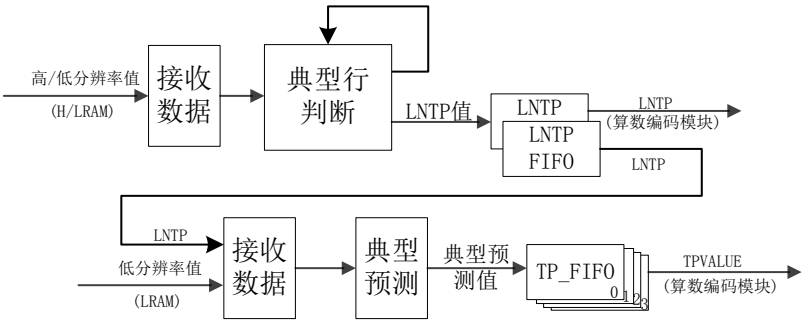

图 3.18 高分辨率层典型预测模块

对 LNTP 模块处理过程中，首先在使能控制下，读取相关高低分辨层数据，然后将低分辨层中目标像素与八临域值和对应的高分辨层率层四邻域值进行比较。若判断高分辨层两行为非典型行时，将 LNTP 值赋 1，并在这两行的处理中不再修改。若为典型行则始终保持 LNTP 为 0 不变。直至整行数据均比较完成后，将 LNTP 写到 FIFO 中存储。对于每一对高分辨层行的判断操作前，需将 LNTP 初始化为 0，默认每行均是典型的。典型预测过程实现见图 3.19 状态转移图。

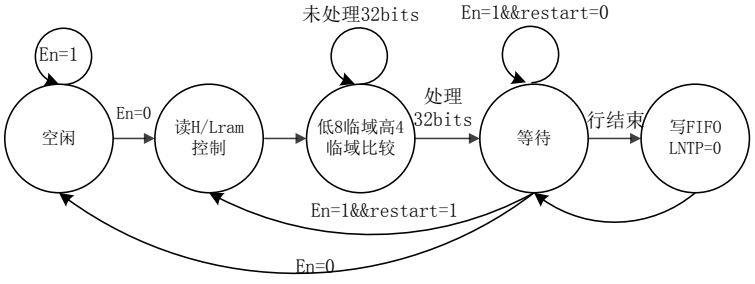

图 3.19 LNTP 模块状态转移图

对目标像素进行典型预测时，需要先将其所在行可否预测信息 LNTP 值读取出来。若目标行为非典型行，那么无需进行进一步比较操作，直接将目标像素对应的高分辨率层像素典型预测值赋值为 2，表示不可预测。若非典型行，还需将目标像素与其八临域值进行比较。处理并存储 32bits 数据后，状态机进入等待状态，等待下一组数据的处理，如图 3.20。

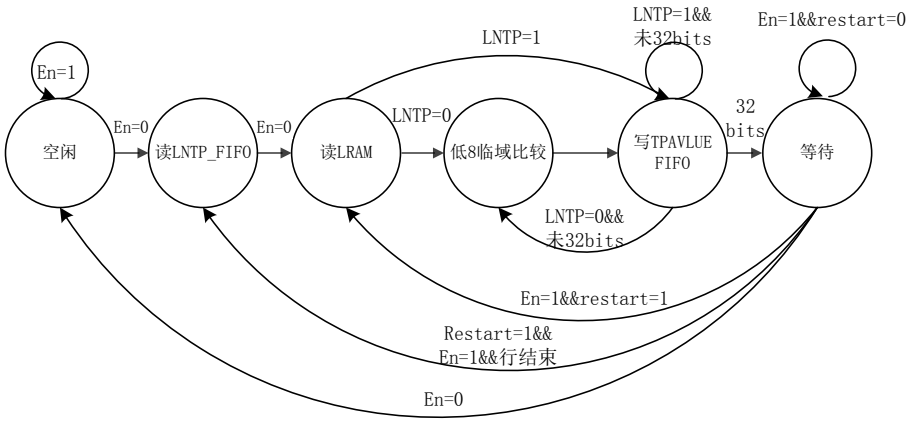

图 3.20 TPVALUE 模块状态转移图

LNTP 模块仿真波形图如图 3.21 所示。按照并行处理设计，周期性读取各分辨率层数据用于行特性的判断写 LNTP 信号向存储空间输出了 4 组数据，由 LNTP 值信号可以看出分别为 1、0、0、0，即第一、二行为非典型行，三至八行为典型行。

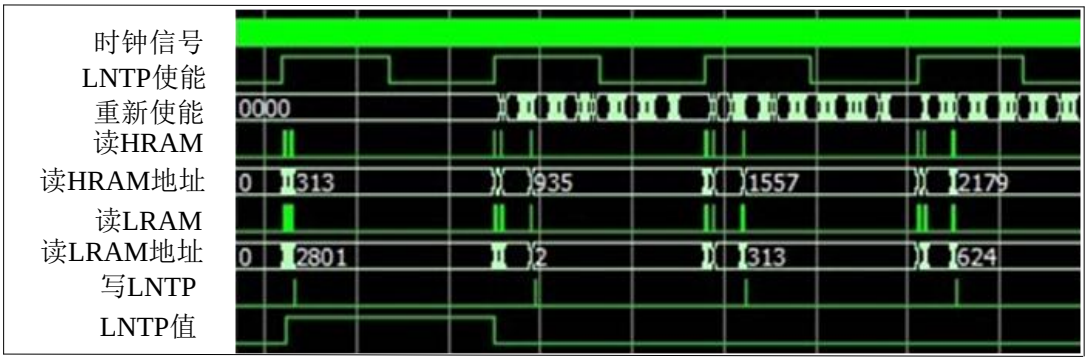

图 3.21 LNTP 模块仿真图

TPVALUE 模块仿真波形图如图 3.22 所示。写 TPVALUE0/1 和 TPVALUE2/3 引脚交替对 0、1 和 2、3 相位存储模块输出写信号。TPVALUE 输出值信号由 0 至 3 相位共 8 位组成，用于输出预测后结果，由于一、二行为非典型行，其输出均为 2。

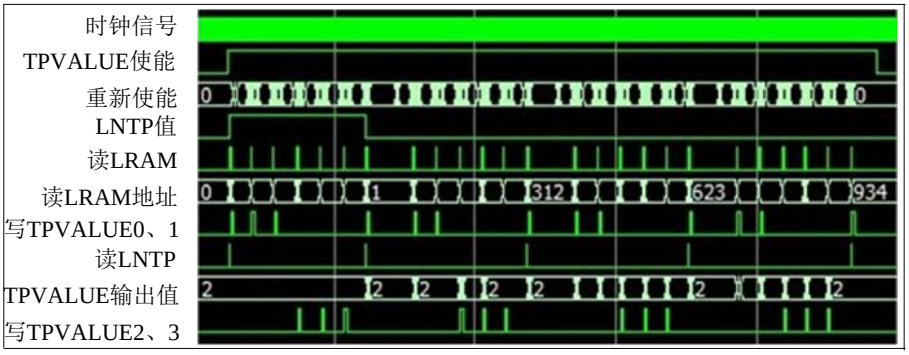

图 3.22 TPVALUE 模块仿真图

3.4.6.2 最低分辨率层典型预测模块

最低分辨率层典型预测值仅取决于目标行与上一行对应像素是否相同，若所有像素都相同，那么本行数据就可由上一行像素值完全描述，仅在压缩过程中存储一个虚拟像素 SLNTP 即可。若有任何一个或以上像素不同，则本行数据都视为不可预测，需要将所有数据进行压缩处理，如图 3.23 所示。其中 SLNTP_FIFO 用于存储最低分辨层虚拟像素值，LNTPy_FIFO 用于存储 LNTPy 值，用于指示目标像素可否预测。两块存储空间均占 1bit×4 字节空间。

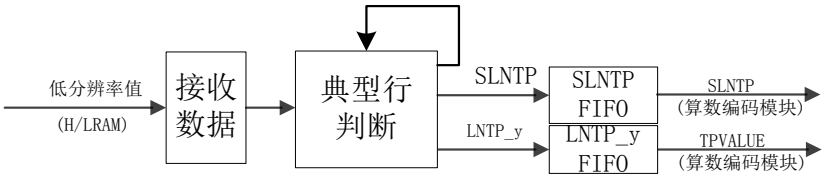

图 3.23 SLNTP 模块

最低分辨虚拟像素 SLNTP 用于输出到算数编码模块进行压缩，以保存当前行可否预测的信息。LNTPy 值在压缩工程中作为典型预测值进行判断，而无需进一步确定具体的预测值。

SLNTP 模块仿真波形图如图 3.24 所示。该模块不产生读最低分辨率层地址，而是通过并行处理设计，直接接收来自高分辨率层典型预测模块读取相应像素行的数据。这一设计可以大量减少读取 LRAM 模块的次数。仿真图中写信号同时对两个存储空间产生四组写有效信号，分别存储了 SLNTPy 值 1、1、0、1 和 LNTPy 值 1、1、0、0。前者用于算数编码过程中虚拟像素的压缩，后者用于判断目标像素所在行的像素可否预测。对于其值为 1 的行，该行的所有目标像素均不可预测，需要压缩处理，这样通过 LNTPy 信号代替典型预测值 TPVALUE 的计算。

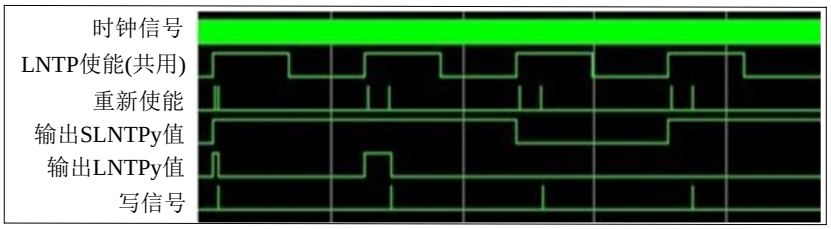

图 3.24 SLNTP 模块仿真图

3.4.7 上下文关联生成模块

3.4.7.1 高分辨率层上下文关联生成模块

最高分辨层上下文生成模块用于生成目标像素的一个与周围像素的关系值，当所有预测均无效时，这个关联值一方面保存目标像素信息，另一方面还包含了一小区域内的图像特征，将其作为输入序列用于确定新的概率估计值，如图 3.25。该模块处理也需要按照 4 相位目标像素分别处理，所以需要使用四个 10bits×32 字节存储空间。

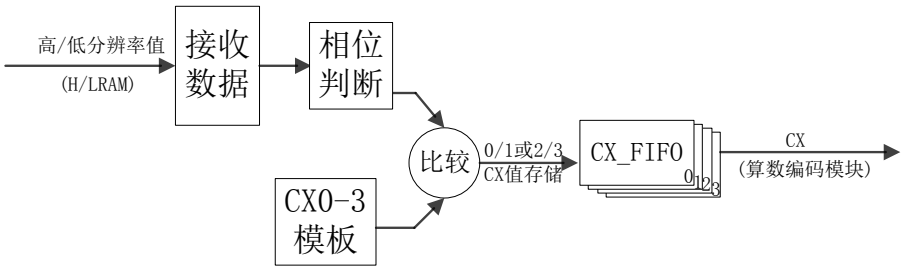

图 3.25 高分辨率层上下文关联生成模块

上下文关联模块在时钟上升沿采集到使能信号，首先读取相关行的像素值，然后利用 4 个不同相位模板，产生对应的上下文关联值并存储。和其它涉及相位存储方式相同，每一轮操作仅存储 0、1 或 2、3 相位对应的生成值，从而减少存储空间，状态转移见图 3.26。

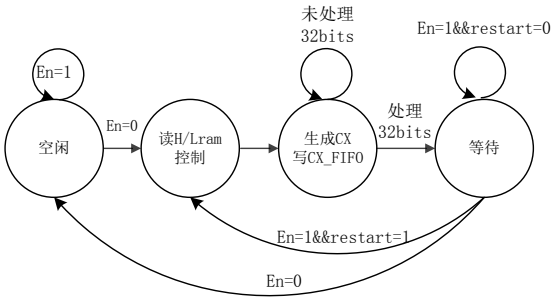

图 3.26 上下文关联生成模块状态转移图

上下文关联模块仿真波形图与放大图如图 3.27 所示。类似上文其它模块读取过程，将接收数据按照模板生成的上下文关联值分别通过写 0、1 相位和写 2、3 相位引脚交替对 0、1 和 2、3 相位存储模块输出写信号，并将对应数据进行分别存储，相位区分标志信号实现了相位交替处理的设计。放大图显示了一组关联值生成的全过程，写 0、1 相位信号共持续了 16 个时钟，模块分别对 0、1 相位存储空间写上下文关联值，其中 0 相位存储 15 个二进制值 1111000011 和 1 个 1011000011 值，1 相位存储 16 个 1111000011。生成共计 32 个目标像素的上下文关联值，达到了在固定时间内完成数据处理的设计目的。

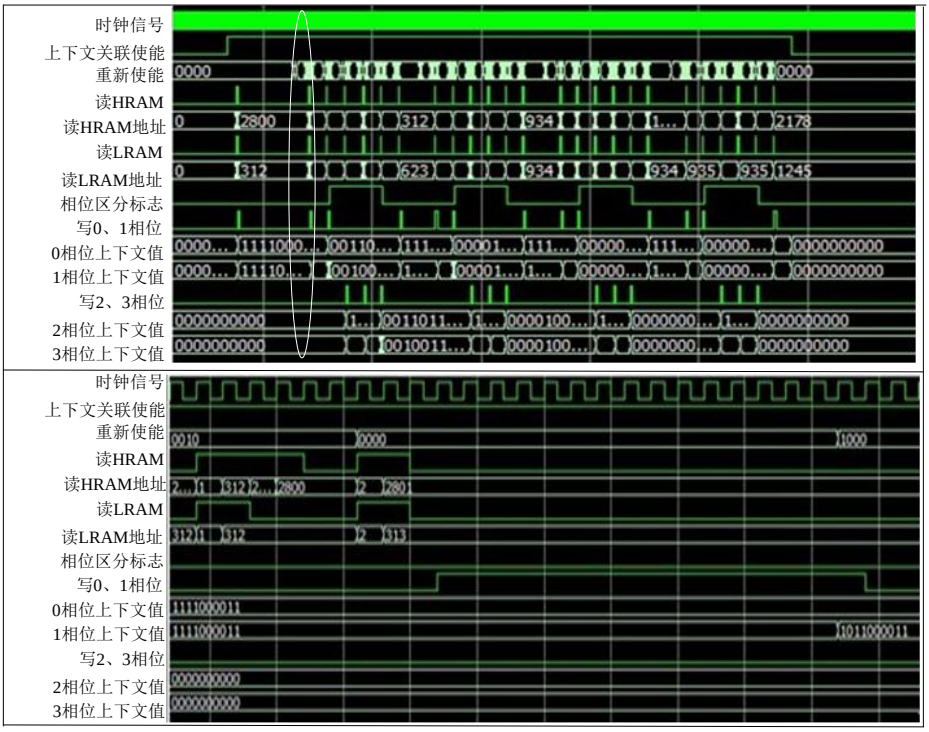

图 3.27 高分辨率层上下文关联生成模块仿真图

3.4.7.1 最低分辨率层上下文关联生成模块

最低分辨层上下文生成模块实现的功能和过程与高分辨率层上下文关联模块相同。所不同的是最低分辨上下文不再区分相位处理，而是使用一个 10bit×16 字节的存储空间。处理过程中使用的模板也不相同，本文实现采用了两行模板作为目标像素的上下文参考，如图 3.28。

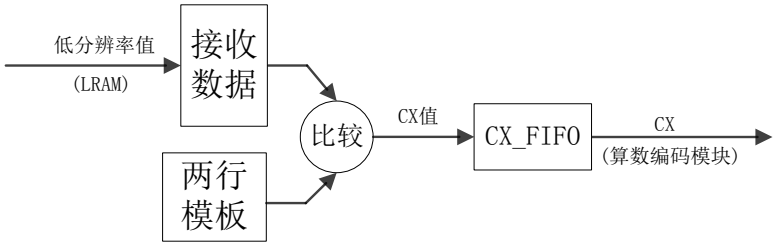

图 3.28 最低分辨率层上下文关联生成模块

最低分辨率层上下文关联模块仿真波形图如图 3.29 所示。写信号不再区分相位，直接对当前生成上下文关联值进行输出。放大图显示了写第 2、3 组数据的全过程，写入上下文关联值分别为 16 个二进制 0000001111 值，符合接收到数据与两行模板的对应关系。

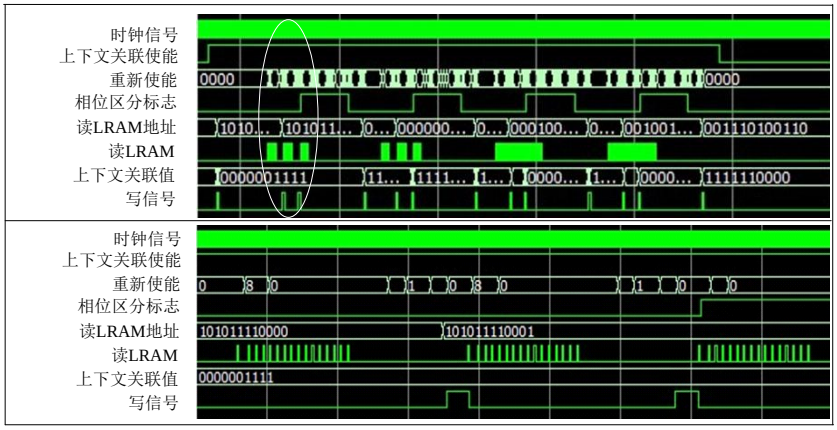

图 3.29 最低分辨率层上下文关联生成模块仿真图

3.4.8 算数编码模块

3.4.10.1 高分辨率层算数编码模块

高分辨率层算数编码模块是本文设计的核心。将高分辨层像素作为处理对象，并将该像素预先产生的辅助值输入作为参考，然后进行压缩处理。压缩后的图像数据通过一个 FIFO 模块作为 JBIG 模块压缩输出接口，用于输出到外部存储设备中。一个完整的高分辨率层算数编码模块如图 3.30 所示。按照流水线设计，将其划分为 3 个子过程。第一子过程主要进行数据读取。对于不相同的上下文关联值，由于其出现的概率并不相同，所以需要对其每一种可能值保持不同的概率估计，当出现相同的上下文关联值参与压缩过程时，可以通过存储信息查找到划分新子空间的条件概率值。其中 CXT 模块就是用于存储每一种可能的上下文关联值的概率估计查找表地址，以及此时大概率符号 MPS 值，占用 8bits×1024 字节存储空

间。PET_T 用于存储概率估计值，占用 31bits×128 字节。第二子过程进行压缩处理，第三子过程用于压缩值的输出。由于流水线设计将输出过程完全独立于压缩过程，所以在第三子过程中需要有一个 8bits×8 字节的 TEMP_FIFO 空间对生成的压缩值进行缓存，以保证新生成的压缩值不会因为上一个输出过程未结束而造成数据丢失。输出过程又可细分为两类，一种用于压缩过程中的输出，一种用于压缩过程结束后的输出，以保证图像压缩值的完整。

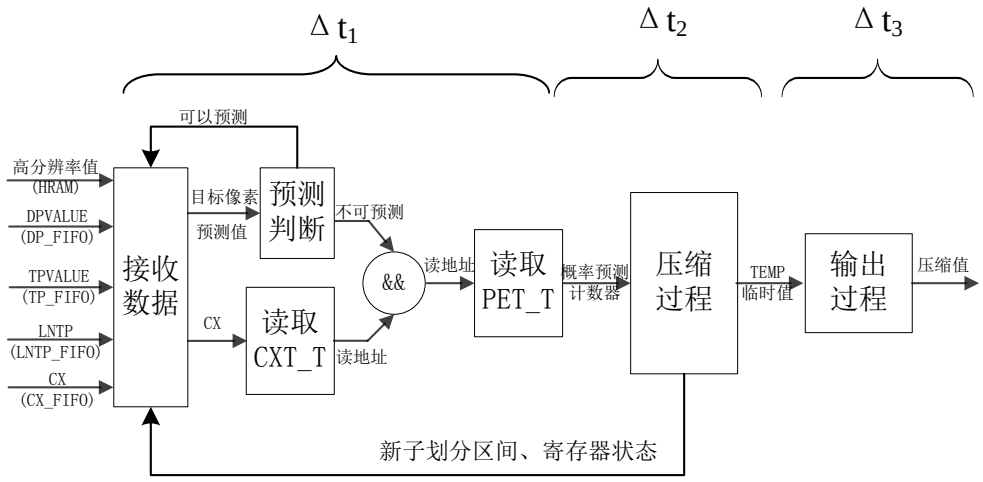

图 3.30 高分辨率层算数编码模块

第一子过程实现如图 3.31 所示。在模块检测到使能信号后，每处理 32bits 数据读取一次高分辨率层像素值，并且每次处理均需要读取目标像素的各类辅助值。下一个时钟上升沿到来时，根据接收到各类数据判断是否可以进行预测，若目标像素不可预测，则后续时钟继续查表操作读取压缩所需的信息，由于流水线第二过程是瓶颈过程且需要 5 个时钟才能完成一个像素的处理，所以第一子过程需要有延时操作。第一子过程通过寄存器 state 来控制第二过程的状态转移。赋值为 1 后，在下一个时钟上升沿到来时第二子过程将进入下一个流水线过程。若目标像素可以预测，则第一子过程直接进入延时操作，且不改变 state 的状态，那么就不会对当前像素进行压缩处理。

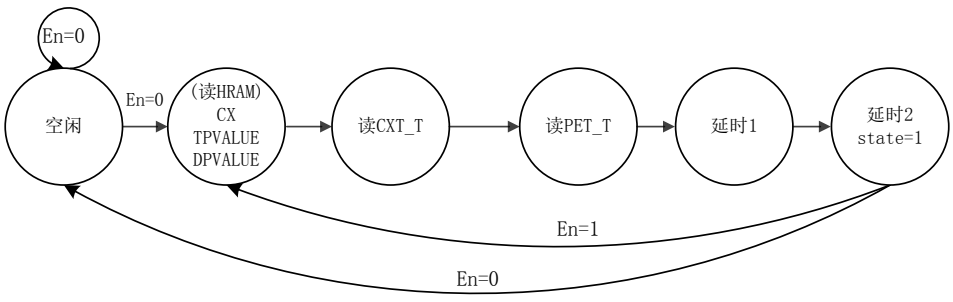

图 3.31 高分辨率层算数编码模块状态转移图 1

第二子过程按照编码方式划分的流程如图 3.32 所示。在状态转移寄存器 state 变化下进入压缩操作，第一个时钟直接接收第一模块读取的 PET_T 表的数据，并且通过公式：

$$A = A - LSZ \tag{3-5}$$

计算新子空间的大小。在第一时钟内还需要判断目标像素是否是大概率符号（判断是否等于 MPS 的值），从而确定下一步采用的编码方式。JBIG 压缩算法共有大概率和小概率两种编码方式。若第二个时钟采用小概率编码方式，需要首先判断读 PET_T 表中的 SWTCH 值是否为 1，若为 1 表示 LPS 所保存的概率区间大于 MPS 所保存的概率区间，需要将寄存器 MPS 的值取反，并将下一个小概率编码概率预测值的查找表地址赋给 ST 寄存器，两个寄存器的值组成一个 8 位数据作为输出值，用当前像素的上下文关联值作为输出地址，写到 CXT 模块中，这一操作的目的是用于变换当前上下文关联值所对应的概率估计。同样在第二至第五时钟内还要并行进行归一化的处理，根据新概率子空间 A 与当前上下文关联值的概率估计值的关系共划分有三种不同的归一化模式。

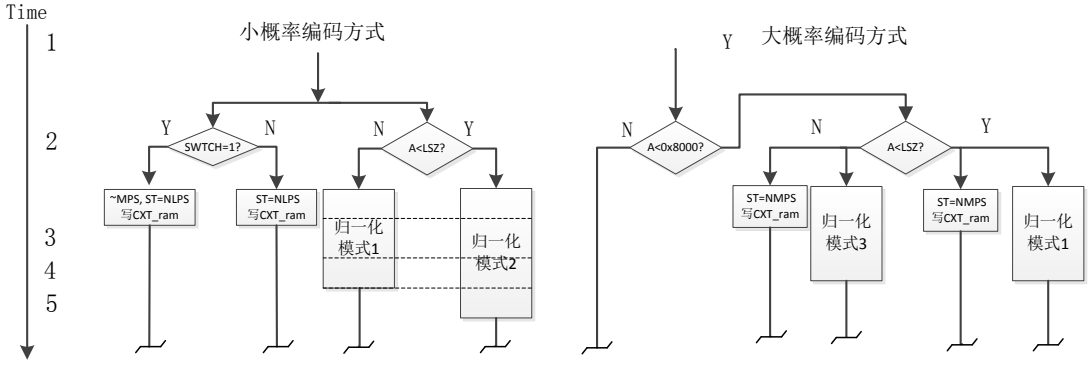

图 3.32 高分辨率层编码方式

当 A 寄存器保存的概率子空间小于 0x8000 值时，超出了概率子区间的下限，需要通过左移操作（乘 2 操作）使其值重新恢复到规定区间内，这时就需要归一化过程。归一化过程需要使用到 CT 寄存器，A 寄存器保存左移的次数，每左移 8 次操作，表示产生一字节压缩值，这一寄存器采用自减的方式操作。三种归一化模式如图 3.33。图中所示按照功能顺序，而不是按照时序（时钟）顺序介绍整个流程。

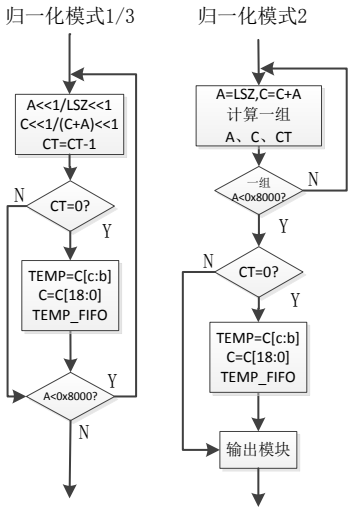

图 3.33 归一化过程流程图

归一化模式 1，首先将 A、C 寄存器值左移一位，并对 CT 自减 1，然后判断是否需要输出操作以及进一步的左移。通过对查找表 PET 中所有可能出现的概率估计值 LSZ 以及压缩算法的分析， A<LSZ 成立的条件只可能是 LSZ 大于 0x4000 的情况，在极端情况下，LSZ 取得的最大值为 0x5b12，A 同时取最小值 0x8000，那么经过 A-LSZ 后操作，新区间 A 为 0x24ee，对 A 经过两次左移操作即可完成归一化，使 A 空间变化为 0x93b8，大于下限 0x8000。所以对应于硬件实现来说，归一化过程可以控制在 3 个时钟内完成，保证了流水线设计的正确。在 A 变化时寄存器 C 与 CT 也进行相应变化，当 CT 为 0 时需将 C 寄存器中 c 与 b 共 9 位有效值进行输出(即 C 寄存器的 27 至 19 位)，并重新赋值 C 寄存器为其低 19 位的值，这一操作去除了已输出位的数据而保留低位的数据，并初始化 CT 寄存器为 8。

在 A<LSZ 判定条件不成立时，需要采用第二种归一化模式。在极端情况下，取值 LSZ 最小值 0x0001，将其赋值给 A 作为新的子区间，在归一化时，需要连续进行 15 次左移操作。所以归一化需要 15 个时钟才能完成，不能满足流水线的要求。改进方法是在第一个时钟内对 A 同时进行 1 至 8 次的左移操作，并用 8 个寄存器存储，在第二个时钟内将 8 个寄存器中取值最小且大于 0x8000 的值赋给 A 作为归一化后的结果。若无满足条件的值生成，则在这一时钟内对 A 进行 9 至 15 次的左移操作，根据最坏情况，下一时钟也一定会得到归一化的结果。对于 C 寄存器，首先要将其赋值为 A+C，将新概率空间的特征反映到 C 寄存器中，然后再根据 A 的左移次数进行相同的操作。在第一组 A 左移 0 至 8 次后均不满足的条件下，寄存器 CT 必然会经历一次归零的过程，此时输出第一个字节的压缩值，并对 CT 进行一次模 8 操作。在第二组 A 左移后，CT 相应的操作可能会连续生成第二个压缩值并输出。这一设计可以将整个左移操作和输出操作控制在 4 个时钟内完成，保证了流水线设计的正确。

由于寻找满足条件归一化结果的操作，会引入条件判定语句的嵌套，对于每一层嵌套，在硬件实现上都会引入一定延时。对于顺序查询方式，最坏情况下需要判定 8 次才能得到结果。所以对左移操作次数概率统计的进一步分析，判定的优先级进行了如图 3.34 的改进，其中数字代表左移操作次数。由于采用最优二叉树设计也需要同样嵌套 4 层，所以综合考虑仍采用如图所示判定设计。在概率统计上左移 1 次的情况最多，但是为了减少判定嵌套层数，选择了左移 2 作为首次判定的条件。由图中可知，最大嵌套层数为 4，比顺序判定减少了一半的判定次数。

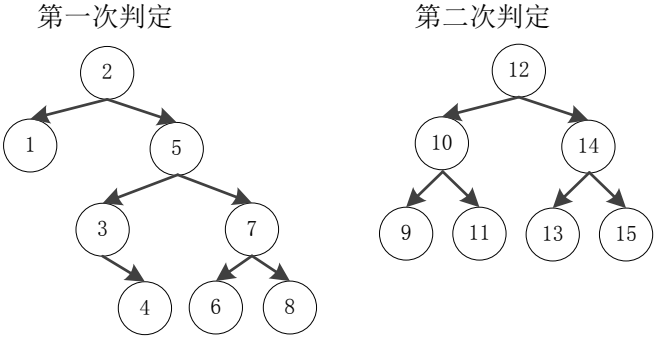

图 3.34 判定优先级示意图

归一化模式 3 和模式 1 类似，所不同的是在第一次赋值且左移的过程中，先将 LSZ 左移一位赋值给 A 寄存器，将(C+A)的值左移一位赋值给 C 寄存器，而后的左移与输出判定的操作完全相同。

按照归一化方式划分的第二子过程状态转移图如图 3.35。按照流水线设计，5 个时钟内可以完成各种编码以及各种归一化方式下的压缩过程。整个状态转移由 state 寄存器值控制。在压缩一个值结束时，若 state 值由第一子过程重新赋值为 1 时，表明下一个目标像素不可预测，需要进行压缩处理，当前子过程状态转移到 1 过程开始相应的操作。若 state 值保持不变，表明有 1 个或多个连续像素可以被预测，无需进行压缩过程。

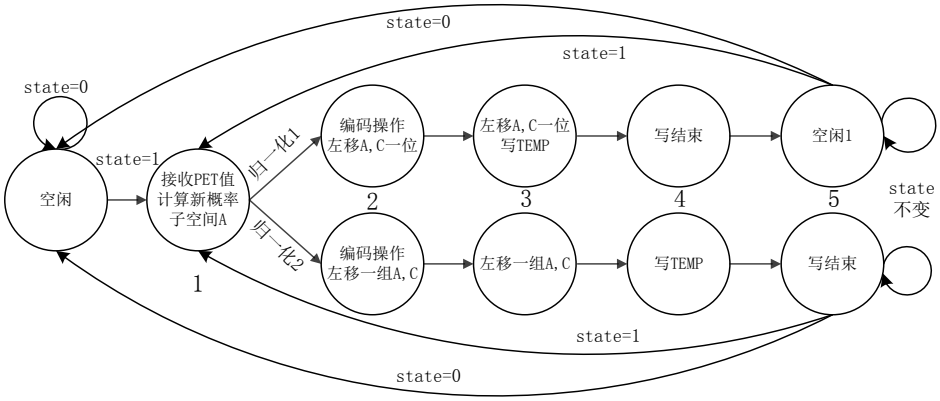

图 3.35 高分辨率层算数编码模块状态转移图 2

第三子过程用于将压缩值的输出，其状态转移图如图 3.36。当 TEMP_FIFO 不

为空，就需要读取一个 TEMP 值。接收到 TEMP 值并不直接进行输出，而是需要判断其最高位是否为 1（是否有进位），若无进位还需要判断 TEMP 值是否为 0xff 值，若是需要通过寄存器 SC 自增 1。SC 保存的是连续 0xff 值的个数，在后面字节产生进位时，对于连续的 0xff 值会产生连续进位，并将 SC 个 0xff 变化为连续的 0x00。所以在接收新的 TEMP 值且有进位时，将输出上一个接收到的 TEMP 值加 1 后的值，然后连续输出 SC 个 0x00 值（SC 也可能为 0）。若无进位，则直接输出上一个 TEMP 值以及连续 SC 个 0xff 值。这种连续输出的情况就是导致输出过程时间不固定的原因。

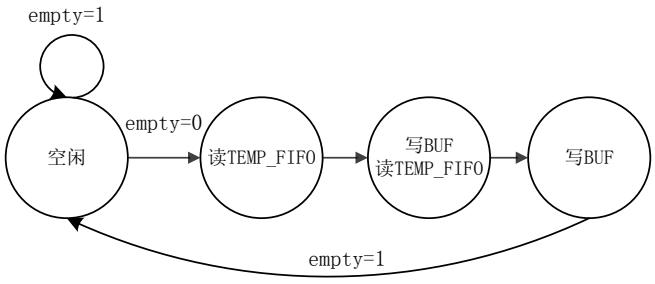

图 3.36 高分辨率层算数编码模块状态转移图 3

对于所有目标像素压缩结束后，输出模块中还存有一个尚未输出的 TEMP 值，且 A、C 寄存器中分别保有压缩后的像素特征，所以需要对所有数据进行最后的输出控制，按照功能顺序的流程图如图 3.37 所示。

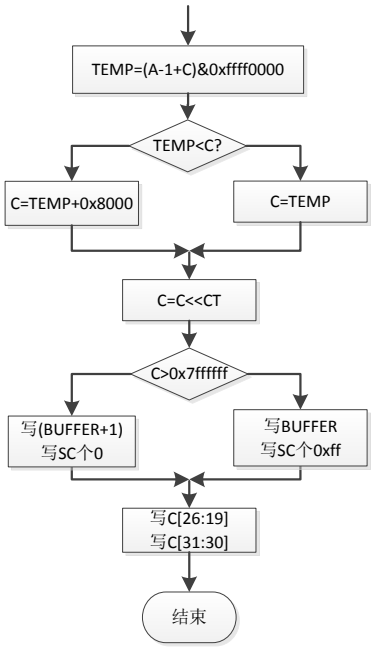

图 3.37 写最后一写模块流程图

根据并行处理的设计，算数编码模块需要对各辅助值生成模块产生多个重新使能信号。但由于分辨率降低模块与算数编码模块使用不同时钟，且两模块间并

无同步机制，对于 HRAM 或者 LRAM 的读取仍会产生冲突。所以压缩模块使用 clk 和 clk2 两个时钟并根据它们的特性做如下改进，如图 3.38 所示。首先控制算数编码模块和辅助值生成模块读有效信号总在同步时钟 clk 上升沿时才有效（辅助值生成模块是利用重新使能信号控制，从而间接控制读有效信号能够在时钟上升沿有效），其次为了保证各模块与分辨降低模块不冲突，只在时钟 clk 上升沿后的对应 1~4 个时钟内进行读操作，而 5、6 时钟留给分辨率降低模块，使其在 clk 下一个上升沿时采集正确的输入值。对于各个模块在 4 个时钟内未完成全部值的读取，则在第二个 clk 时钟上升沿后继续同样操作。实际上所有模块中最多只有连续读 6 个值的情况，所以只使用了第一个 clk2 时钟对应的 1~4 区间，和第二个的 1、2 区间。

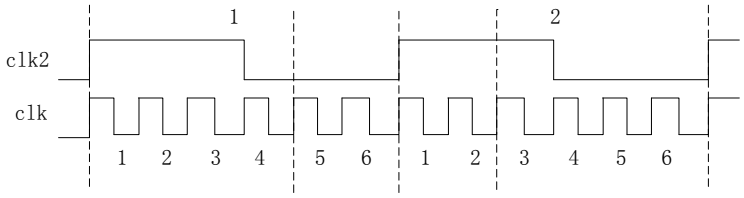

图 3.38 时钟特性

高分辨层压缩模块读写子过程仿真波形图如图 3.39 所示。图中显示了压缩模块读 HRAM 控制以及读各个辅助模块缓存的全部过程，仿真图只按照并行处理设计显示了读取过程，而没有将接收到的结果信号也罗列出来，其值是以上个模块仿真图中所示的生成值。通过对整幅图像压缩处理读写子过程的仿真，可以看出压缩模块周期性处理的特点。

图 3.39 高分辨率层压缩模块仿真图 1

压缩处理与输出处理子过程仿真波形图与放大图如图 3.40 所示。图中显示了

压缩过程中重要寄存器变化状态。压缩值输出信号将处理后的结果输出，按照输入激励设计，其输出为十六进制压缩值：fc，8b，8f，98，f1，e1，7c，74，15，56，1d，9a，00 共 13 个字节。放大图中显示了生成并输出压缩值 0xfc 的过程，其中几个寄存器值变化是调试的关键，A 寄存器的变化在压缩和解压缩过程中变化是一致的。

图 3.40 高分辨率层压缩模块仿真图 2

3.4.10.2 最低分辨率层算数编码模块

最低分辨率层算数编码模块类似于高分辨率层算数编码模块，实现过程基本相同。区别在于编码器读取的各类缓存是最低分辨率层的辅助值，且没有确定性预测的判断。在压缩时低分辨率层总是快于高分辨率层，所以在最低分辨层算数编码模块应有一个输入引脚 finish，用于与高分辨率层压缩模块同步。最低分辨率层与高分辨层压缩数据分别存储。

最低分辨层压缩模块仿真波形图如图 3.41 所示。图中显示了压缩模块读 LRAM 控制以及读各个辅助模块缓存的全部过程。压缩值输出信号将处理后的结果输出，为十六进制压缩值：f9，da，e8，2e，e4，74，00 共 7 个字节。

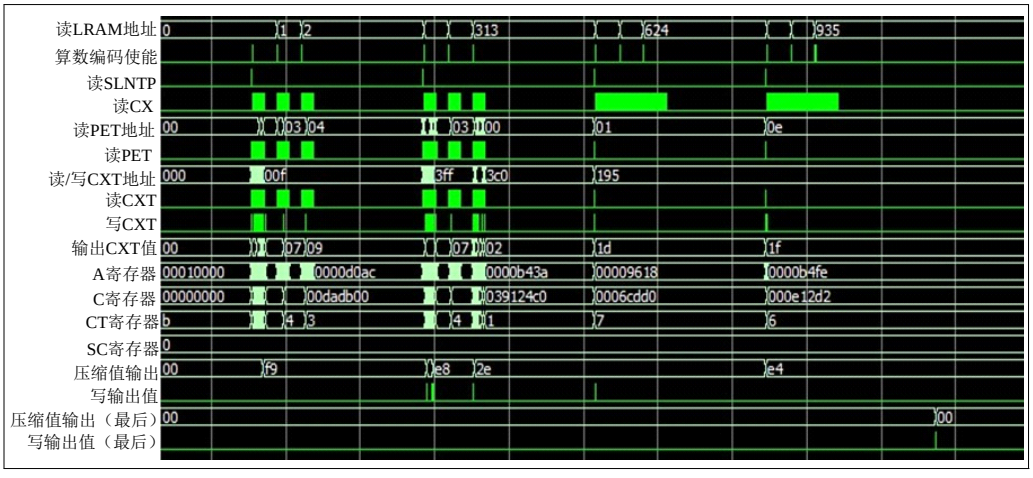

图 3.41 最低分辨率层压缩模块仿真图

3.5 JBIG 压缩算法实现分析

通过 3.3、 3.4 节 JBIG 压缩模块设计实现与仿真可以得到以下结论：

1) JBIG 压缩过程压缩后的数据最低分辨层压缩值为 f9，da，e8，2e，e4，74，00；高分辨率层压缩值为 fc，8b，8f，98，f1，e1，7c，74，15，56，1d，9a，00 共 20 字节十六进制值。相比于原始图像 96 字节，压缩比为 20.8％。
2) 输入图像数据具有特殊性。一方面满足了各个模块仿真需求，使各个辅助值生成模块可能出现的情况都进行了验证。另一方面有利于观察和描述，但是缺点是图像的压缩率不具有普遍性。
3) JBIG 压缩算法实现的正确性需要利用解压缩过程辅助。经过测试，将上述压缩值输入后，重建的图像与原图像相同，证明了压缩过程的正确性。

第四章 JBIG 解压缩算法改进与 FPGA 设计

JBIG 解压缩过程是其压缩过程的逆过程。在操作过程中会用到编码过程中相同的辅助值生成模块。这样的好处是显然的，将大量减少编码与调试的成本。但是对于压缩过程，所有辅助值生成模块使用的各类模板中像素都是已知的，所以独立于压缩模块可以提前进行预计算和存储，从而达到与压缩过程并行处理的目的。然而 JBIG 解压过程所有的像素需要一位一位的解压还原，对目标像素进行处理时，并不能并行的对下一个像素提前产生其辅助值。这种算法上的限制会导致压缩过程设计中并行处理方式不再适用，且解压缩的时间会远大于压缩过程。如果再利用压缩过程中辅助值生成模块，在不能并行处理的情况下，又加入大量各模块的读 HRAM 和 LRAM 时钟，会进一步降低解压缩过程的速度。所以一种实现方法见图 4.1

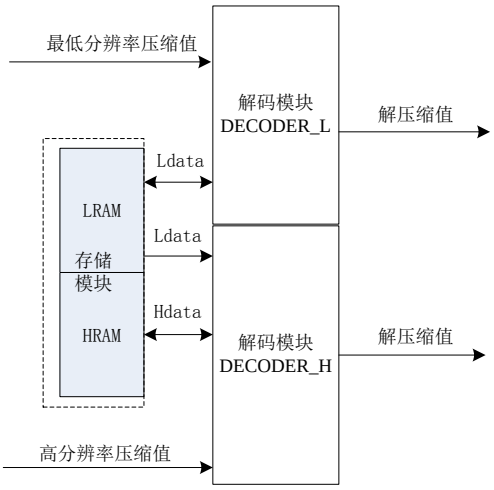

图 4.1 JBIG 解压缩过程结构图

4.1 并行处理

JBIG 解压缩过程将辅助值生成模块融入到解压缩模块 ENCODER 模块中。并行处理就是解决最低分辨率层与高分辨层同时运算的设计。如图 4.2 是两层解压缩过程在稳定后并行处理分析。图中小写字母表示最低分辨率层解压缩过程，大写字母表示高分辨率层解压缩过程。

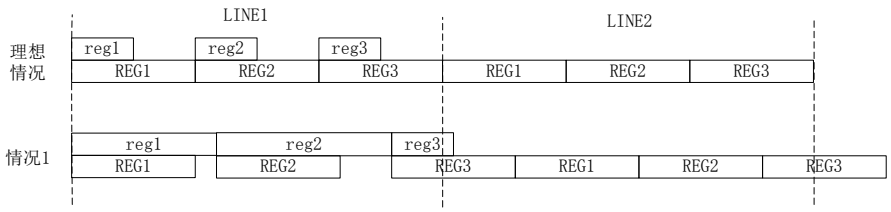

图 4.2 JBIG 解压缩过程分析

在理想情况下，最低分辨率层解压缩过程要快于高分辨率层。但是由于前者要先于后者提前生成一行数据后，高分辨层才能开始执行。这种错位一行并行处理的方式会使同时处理的两个分辨率层行的图像特征不完全关联，这种不关联会导致情况 1 的出现。只要有一个分辨率层没有结束就要停止操作等待，显然这种面向每一个单位处理的方式会大大的降低并行处理过程。所以一种改进方案如图 4.3。

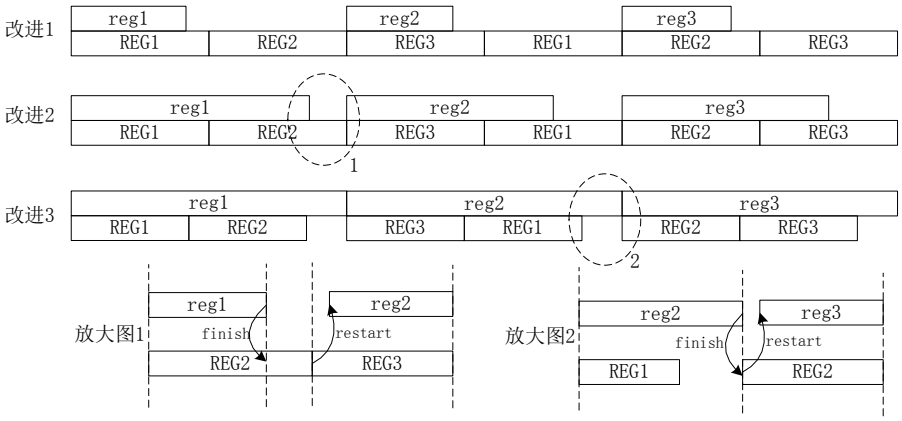

图 4.3 JBIG 解压缩过程并行处理改进

根据每处理两行高分辩层数据同时处理一行低分辨率层的特性，将每处理 2 个单位 REG 的时间与处理 1 个 reg 的时间同步，见图中改进 1。当出现改进 2、3 的情况下相对于上文所述会更多的节省解压缩的时间。图中放大图 1 所示，最低分辨率层完成 1 单位 reg 处理后对高分辨层发出 finish 信号，这种情况下并不增加整个解压缩过程时间。图中放大图 2 所示，最低分辨率层完成 1 单位处理时间大于高分辨层同时处理 2 个 REG 的时间，需要停止高分辨率层解压缩过程等待，直到接收到 finish 信号后重新开始，并向低分辨层发送重新开始信号 restart。需要并行处理一方面可以减少处理时间，更重要的是解决对 LRAM 模块的读取冲突问题。

4.2 JBIG 解压缩算法实现

4.2.1 最低分辨率层解压缩模块

最低分辨率层解压缩模块用于重建最低分辨率层的图像数据，与外部模块需要有一个 8bits×8 字节的 FIFO 作为缓存空间，如图 4.4。接收数据后首先需要还原行的虚拟像素值 SLNTP，从而判断该行像素可否预测。若不可预测则进一步通过压缩算法的逆运算重建图像的每一个像素值，组成 16bits 的解压缩值，一方面要输出到打印机用于打印操作，另一方面要存储到 LRAM 中，作为其它像素辅助值的参考值。

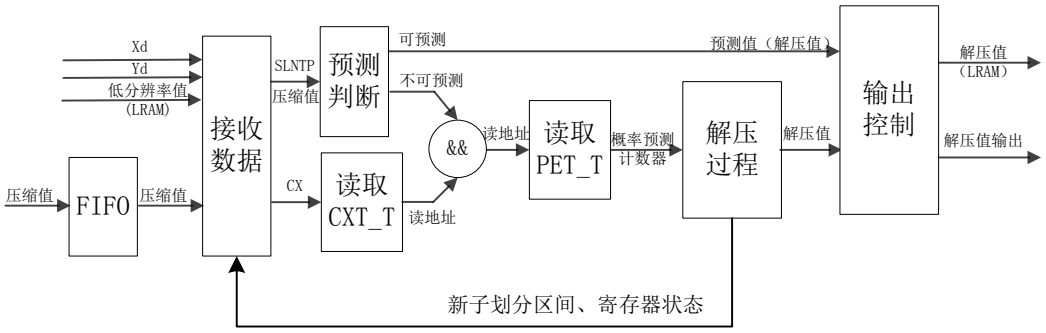

图 4.4 最低分辨率层解压缩模块

解压缩过程也需要通过有限状态机实现，其状态转移图可以借鉴 3.8 节中压缩过程的前 2 个子过程。由于流水线设计不再适用于解压缩过程，所以整个过程只可串行执行。解压缩过程分为小概率、大概率和直接解码三种方式，将替换状态转移图中压缩过程中的两种编码方式，如图 4.5 所示。第一时钟内判断采用何种解码方式，第二时钟产生解压缩像素 PIX，并且变换当前上下文关联值对应概率估计查找表地址以及可能出现的大概率符号。第三个时钟开始进行归一化操作，此处归一化过程均采用前文所述归一化模式 1，所不同的是当寄存器 CT 归零时，将重新读取一个新的压缩值存储到 C 寄存器的 15 至 8 位，用于进一步的解压缩操作。

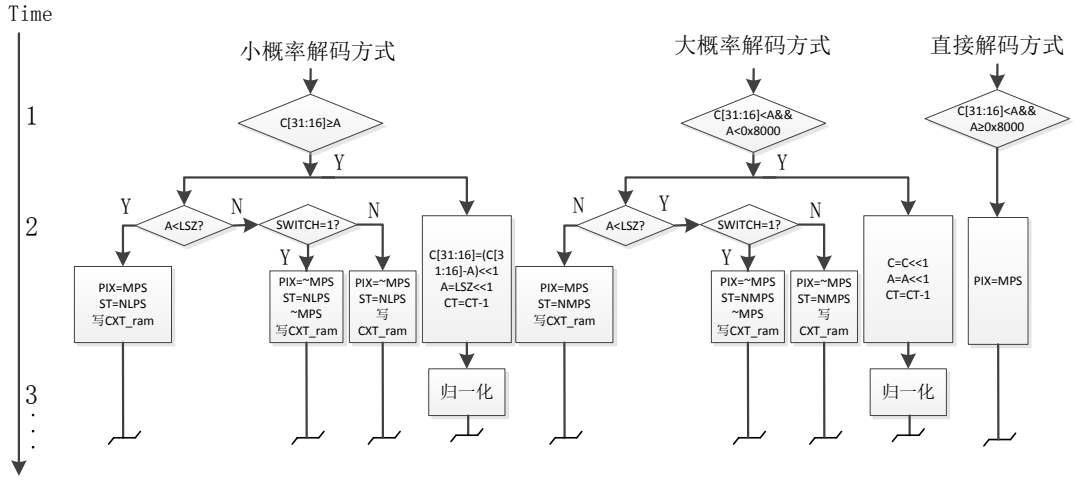

图 4.5 解压缩过程分解

最低分辨率层解压缩读写控制部分仿真如图 4.6 所示。图中显示了读入压缩值以及对各个模块的读写控制，由于解压缩过程不能采用并行生成辅助值方式，所以读写信号不再具有周期性特点。图中将接收到的压缩值输入进行解压处理，还原出 SLNTP 值后计算出当前行可否预测判定值 $LNTP_y$，然后做进一步的处理。图中在 $LNTP_y$ 为 1 时，需要进行解压缩处理，对应的读写有效的控制信号密集，而值为 0 时，直接将接收到的 LRAM 值作为解压缩值输出，不进行解压缩处理过程中对各个查找表的读写控制。

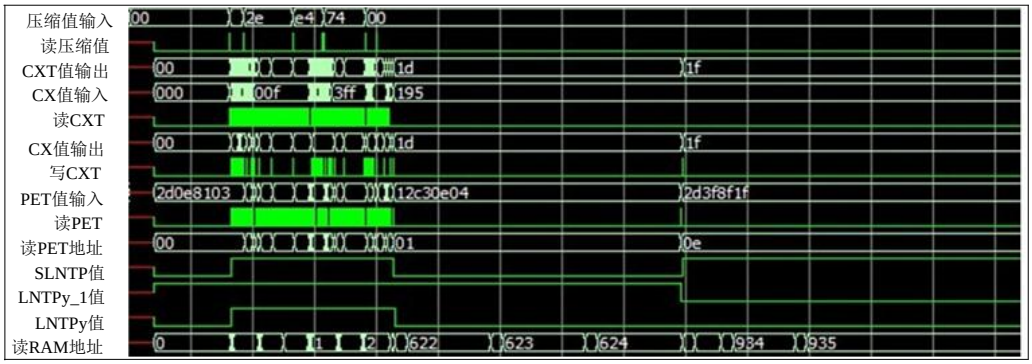

图 4.6 最低分辨率层解压缩仿真图 1

解压部分仿真与放大图如图 4.7 所示。图中显示了解压缩值输出以及寄存器变化信号等。写压缩值信号共有效 12 次，表明还原了 12 组各 16bits 数据，即还原了最低分辨率的图像 4 行像素。放大图显示的是第一组数据还原时的状态，还原像素值为 0xeeff，重建像素正确。

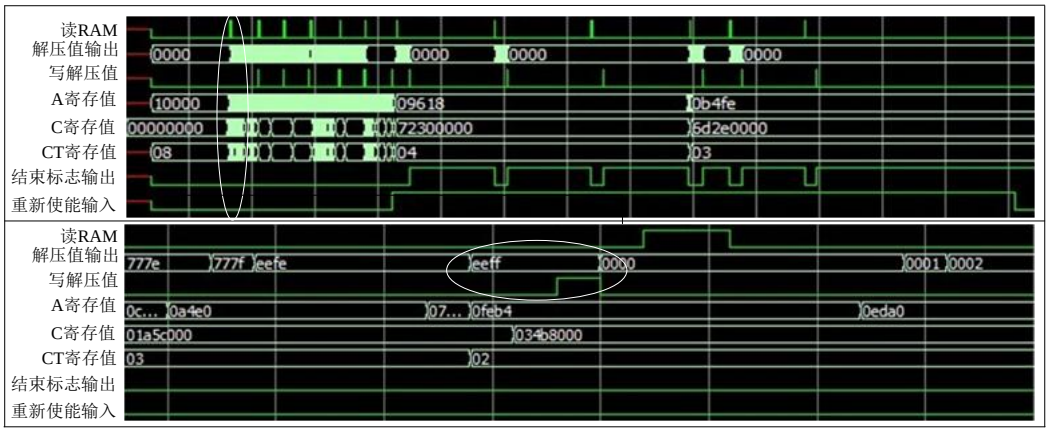

图 4.7 最低分辨率层解压缩仿真图 2

4.2.2 高分辨率层解压缩模块

高分辨率层解压缩模块与最低分辨率层解压缩模块相较，仅多了 4 个 0 至 3 相位的确定性预测查找表和确定性预测过程，如图 4.8。在操作过程中，需要首先恢复 LNTP 值，进而计算 TPVALUE，并通过模板产生查找表地址计算 DPVALUE，以判断目标像素可否预测。若可以预测则将预测值直接作为解压值，若不可预测则进入解压缩过程。解压缩过程与最低分辨层解压缩过程完全相同。高分辨层解压值输出以 32bits 为单位。

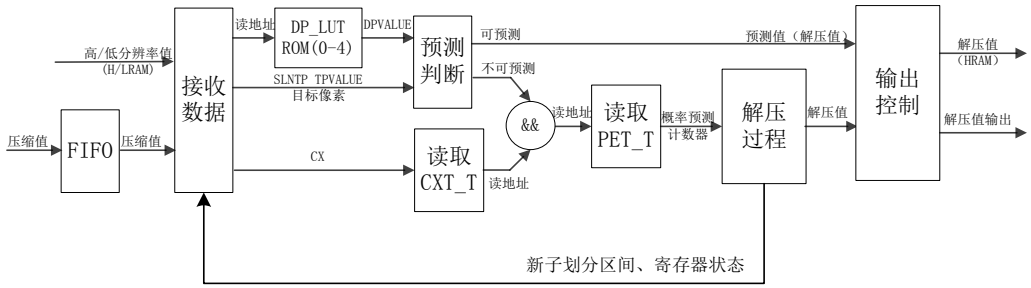

图 4.8 高分辨率层解压缩模块

高分辨率层解压缩读取控制部分仿真如图 4.9 所示。图中显示与最低分辨率层解压缩过程类似，增加了对确定性预测值生成过程的输出显示。从读 LRAM 信号可以看出来，由于压缩处理的输入激励前几行数据差异较大，对应的解压缩时间也较长，而最后几行数据相同，LRAM 信号有效间隔变短，表明解压缩时间较短。解压缩过程所需要的时间和原图像的特征有密切的关系。

图 4.9 高分辨率层解压缩仿真图 1

解压缩过程仿真与放大图如图 4.10 所示。图中写 HRAM 信号显示了共还原了 24 组 32bits 数据，即重建了 8 行高分辨层。放大图显示了第一组数据还原时的状态，还原像素值为 0xf0feffff，重建像素正确。

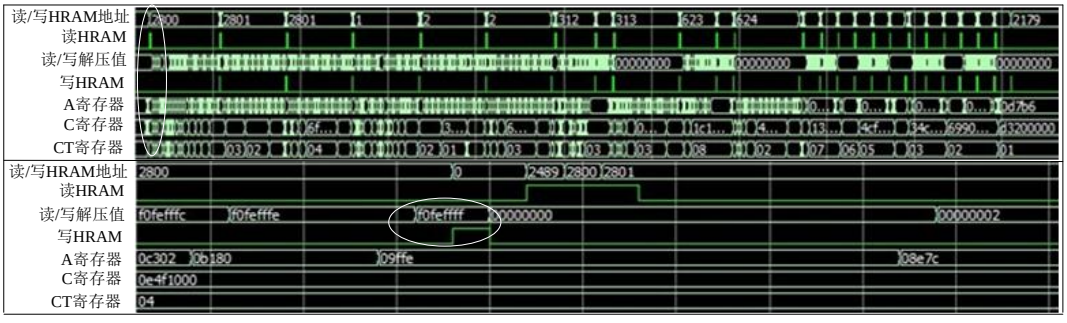

图 4.10 高分辨率层解压缩仿真图 2

4.2.3 高、最低分辨率层存储模块

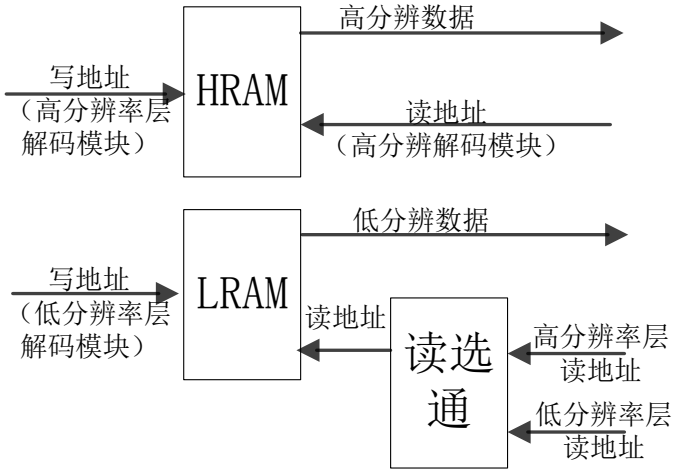

图 4.11 存储模块

解压缩的存储结构与压缩过程 3.4.3 节中所述设计思路相同。实现上不同仅体现在读写控制的模块有所区别。特别在存储空间方面，并行处理的设计，满足了 10 行存储空间循环利用的目的。

4.3 JBIG 解压缩算法实现分析

1) 将 3.4.8 节中压缩值作为 JBIG 解压缩过程的输入，通过以上仿真图可知，解压缩处理后的数据和前文输入测试激励相同，验证了解压缩算法实现的正确性。
2) 上文压缩过程将图像处理后，达到了压缩数据和减少存储空间的目的。在解压处理后，重建了原始输入值，从而也验证了压缩算法实现的正确性。

第五章 结束语

5.1 实验测试结果

本系统的主要目的是实现在 FPGA 上的 JBIG 压缩算法。为验证系统功能，设计两种实验模式。

5.1.1 功能验证

功能实验设计是为了验证 JBIG 压缩算法实现的正确性与可靠性。共采用四组实验数据，分别进行压缩与解压缩操作，见表 5.1 所示。表项中高、低分辨率层所填写的是压缩后的输出数据。四组实验数据在设计上能够满足验证各个模的功能的要求，其压缩率分别为 9.1%、13.2%、20.8%和 69.8%。其中前三组输入值行与行之间对应像素值差异小，但列与列之间像素变化逐渐增多。最后一组测试值无特定规律，像素点间关联不密切。经过对压缩值进行解压缩处理，四组数据分别重建了原始图像像素值。达到了硬件语言实现 JBIG 压缩算法的目的。

表 5.1 测试数据表

第一组	输入值 96×10	1至10行：ffff_ffff ffff_ffff 0000_0000（10行数据相同）
	低分辨层	共5字节：fd 9f 87 70 00
	高分辨层	共6字节：14 63 f8 be 40 00
第二组	输入值 96×12	1至10行：ffff_ffff ffff_ffff 0000_0000　第11行：fffe_ffff ffff_feff ffff_feff 第12行：ffff_ffff ffff_ffff ffff_ffff
	低分辨层	共7字节： fd 9f 87 74 dd 88 00
	高分辨层	共11字节：14 63 f8 be 46 2e da cf 21 7c 00
第三组	输入值 96×8	第1行： f0fe_ffff ffff_ffff ffff_ffff　第2行：fef0_ffff ffff_ffff ffff_ffff 3至6行：ffff_ffff ffff_ffff 0000_0000
	低分辨层	共7字节： f9 da e8 2e e4 74 00
	高分辨层	共13字节： fc 8b 8f 98 f1 e1 7c 74 15 56 1d 9a 00
第四组	输入值 96×8	第1行： f0fe_ffff faff_fcbf 1f3f_f87f　第2行：fef0_ffff ff2f_f4f6 f5ff_2f4f 第3行： f3ff_ff8f ff7f_ff4f 0000_0000　第4行：ffff_ffff ffff_ffff 0000_0000 第5行： ffff_ffff ffff_ffff 0000_0000　第6行：4fff_f7ff fff0_ffff 0000_0000 第7行： ffff_ffff ffff_ffff 0000_0000　第8行：f1f1_f783 f2f1_f4f6 79fe_ef3f
	低分辨层	共21字节：fd dc 73 d2 f3 44 8b 25 e2 13 d5 da 69 00 4e 06 0e 87 fe c8 00
	低分辨层	共46字节：fc 8b 94 31 d3 2d af f5 f1 fb da bb e4 a3 92 eb fa c1 92 ec 88 28 c9 18 98 81 b7 cd b1 80 a5 f1 d1 2d 80 ca 06 a7 41 bb 23 e8 74 4c ba 00

5.1.2 性能验证

性能实验是为了验证 JBIG 压缩算法实际的压缩率而设计。共采用两组实验数

据。第一组数据采用 800×1124bits 大小的二值文本图像作为输入数据，如图 5.1 所示。压缩后高分辨率值与最低分辨率值分别为 7327 和 5874bytes(字节)，其压缩率为 11.74%。第二组数据采用 256×256bits 半色调后的 LENNA 图如图 5.2 所示，压缩后高分辨值与最低分辨率值分别为 4769 和 1659bytes，其压缩率为 78.47%。从压缩率的性能看，JBIG 压缩算法对于纯文本格式有较好的压缩率，对于半色调后的图像也能够达到压缩的目的。


With the urgent need for protecting intellectual property in published or printed media in recent years, print-scan resilient image watermarking algorithm is becoming a hot issue for researchers. In the study of this issue, the work of image printing and scanning is very tedious and repetitive. And due to quality of ink, mechanical failures, or incorrect operations, some distortions like pixel distortion, geometric distortion, and other noises may be brought in when an image is printed and scanned. These distortions will affect performance of watermark algorithm, and make the research a tough work. If we can find a good model to simulate the print-scan procedure, the PS image can be obtained from the model which takes the original image as the input, without undergoing the real print-scan operation. Through this way, it will be able to eliminate image distortions caused by

图 5.1 测试的输入文本图像示例（800×1124）

图 5.2 测试的输入 LENNA 图像示例（256×256）

5.2 总结

随着打印机系统打印分辨率不断提高，以及人们对于图像质量要求的不断增加，图像数据的规模也越来越大，所以实现良好的压缩算法减少存储空间显得至关重要。本文在 JBIG 压缩算法和利用 Verilog 语言硬件实现方面做出了有意义的尝试。

论文对 FPGA 中实现 JBIG 压缩算法进行了一定优化，设计并实现了压缩以及解压缩相关的所有模块，并在此基础上展示了解、压缩过程和仿真结果。

本文所述压缩算法实现有以下优点：

(1) 采用硬件并行方式实现压缩以及解压缩过程，速度快，可满足实时性要求。

(2) 压缩与解压缩模块的前后端都采用了 FIFO 作为接口，并且使能控制简单，易于移植到其它图像处理硬件系统中。

(3) 压缩和解压缩分别可以视为单独模块独立运行，承担压缩与解压缩过程。

(4) 各个模块接口设计合理，有利于后期对这一实现方法的改进。

(5) 对于文本格式以及二值图像压缩比高。

综上所述，可以看出本文设计的基于 FPGA 的 JBIG 压缩算法基本实现了设计的目标，提高了压缩与解压缩过程的执行速度，对图像压缩有较好的压缩比，可以满足打印机系统对于节省存储空间的要求。

5.3 展望

由于时间和条件的限制，本论文实现的 JBIG 压缩和解压缩算法还有很多地方需要改进和完善，主要体现在以下几个方面：

(1) 数据包头的编码并没有实现，而是采用了固定的参数配置。需要改进的地方是允许用户选择压缩控制，配置相应数据包头，支持各个预测模块开启或者关闭状态，从而对应选择压缩比最优还是压缩速度最优等实现。

(2) 半色调模块和压缩模块在电路设计中加入外部存储设备，用于解决速度不匹配问题，从而降低两模块间的关联性以及时序控制难度，减少 FPGA 内存空间耗费。在现有实现基础上增加一个初始化操作控制就可以完全实现条带划分和编码方式。

(3) 进一步优化时序控制，一方面允许图像可以自适应降低多次分辨率层和编码。另一方面继续减少压缩和解压缩的时间，在各模块同步和重新使能控制方面仍有进一步提升的可能。

(4) 有待实现自适应模块 AT，进一步提高半色调图像的压缩比。

致谢

本论文的研究工作是在导师王泉教授的悉心指导下完成，从论文选题到算法的实现，直至最终成文，王老师都给予我极大的帮助。王老师严谨的治学态度，辛勤的工作作风，高瞻远瞩的学术眼界以及充分信任和鼓励都对我产生了深刻的影响。在研究生学习生活期间，王老师给我提供了许多宝贵的机会，使我参与了多个科研与工程项目的工作，利用这一平台我可以将多年所学用于项目实践，使我对自己所研究学习的领域有了更深层次的理解，也对我将来进一步发展给予了方向。值此论文完成之际，谨向王老师表示最由衷的谢意。

感谢杨鹏飞、卢飞、黄伟、张吉阳、丁晓东、柳明等所有师兄弟们，感谢他们在研究、学业以及生活方面的帮助与支持。

感谢我的父母，感谢他们多年对我的爱护和付出，为我提供一个良好的生活环境，使我能够顺利完成学业，追求理想和实现自我价值。感谢他们成为我奋斗的理由以及坚强的后盾。

感谢所有曾经帮助过我、激励我和支持我的师长、同学和朋友们。

参考文献

[1] 邹伟玉．基于分组的变长码解码算法及硬件实现结构研究．上海大学硕士论文．2006, 2．

[2] 胡启明．浅谈数据压缩技术．数据通信．1999, 第 2 期．

[3] 戴善荣编著.《数据压缩》. 西安：西安电子科技大学出版社．2005.

[4] 周正林. 基于噪声特性的数字半调研究. 西安电子科技大学硕士论文. 2005, 1.

[5] 张文琪．误差扩散算法 IP 核设计及基于 SOPC 的半色调系统实现．西安电子科技大学硕士论文．2011, 1．

[6] 张艳．基于 FPGA 的 JPEG 解码算法的研究与实现．南京理工大学硕士论文．2009, 6．

[7] 赵双龙，郝永生. LZW 改进压缩算法的 FPGA 实现. 现代电子动技术. 2011, 1.

[8] 唐朝京，雷菁编著.《信息论与编码基础》. 北京：人民邮电出版社．2010.

[9] 章毓晋编著.《图像工程（上册）图像处理》. 第二版. 北京：清华大学出版社. 2006.

[10] Rafael C. Gonzalez, Richard E. Woods 编著．阮秋琦，阮宇智等译.《数字图像处理》. 第二版．北京：电子工业出版社．2010.

[11] 吴振宇．二值图像压缩 JBIG 标准．电子技术．1998, 第 8 期．

[12] 常环．静态图像压缩标准的发展回顾．中国现代教育装备．2007, 第 4 期．

[13] 童晓峰．二值图像压缩新方法研究．华中科技大学硕士论文．2002, 5．

[14] http://baike.baidu.com/view/2108017.htm．位平面编码．

[15] David Salomon 著．吴乐南等译.《数据压缩原理与应用》. 第二版．北京：电子工业出版社．2003.

[16] 刘伟华，刘立柱，张沛．JBIG 编码方式研究．信息工程大学学报．2008, 第 9 卷, 第 1 期．

[17] ISO / IEC 11544．ITU-T Recommendation T. 82．1993．JBIG 规范

[18] 白中英著.《计算机组成原理》. 第三版．北京：科学出版社．2001.

[19] 何永泰，董刚，黄文卿．流水线技术在 FPGA 设计中的实现．天津工业大学学报．2006, 第 25 卷, 第 4 期．

[20] 王诚，吴继华，范丽珍等编著.《Altera FPGA/CPLD 设计（基础篇）》. 第一版．北京：人民邮电出版社．2005.

[21] 徐洋，黄智宇，李彦等编著.《基于 Verliog HDL 的 FPGA 设计与工程应用》. 北京：人民邮电出版社．2009.

[22] 周润景，苏良碧编著.《基于 QuartusⅡ的数字系统 Verilog HDL 设计实例详解》. 北京：电子工业出版社．2010.

[23] 吴继华，蔡海宁，王诚等编著.《Altera FPGA/CPLD 设计（高级篇）》. 第二

版．北京：人民邮电出版社．2011．

[24] Ian Grout 著．黄以华等译．《基于 FPGA 和 CPLD 的数字系统设计》．北京：电子工业出版社．2009．

[25] 张洪润，张亚凡等编著．《FPGA/CPLD 应用设计 200 例（下册）》．北京：北京航空航天大学出版社．2009．