珠海科技学院

毕业设计

基于 FPGA 的 JPEG 图像编解码系统设计

学　　院：电子信息工程学院

专　业　名　称：微电子科学与工程

学　生　姓　名：黄智为

学　　号：03210828

指导老师姓名、职称：孙永坚 讲师

完成日期：2025 年 3 月 7 日

摘　要

随着近几十年来多媒体技术、图像扫描技术、移动终端、通信技术的不断发展，数字图像和视频数据的广泛使用，图像数据的数量呈指数机增加。于此对图像数据的存储以及传输的需求日益严峻。一味地添加设备的存储容量和信道的带宽是不现实的，对此使用图像压缩技术来减少图像数据的数据量。而作为静态图像压缩国际标准格式的 JPEG（Joint Photographic Experts Group），因具有压缩率高、失真率小等特点。在国际上取得广泛的应用。

目前大多数的 JPGE 格式图像数据编解码系统都是基于软件编程从而运行在通用计算机上。这种方式存在计算效率低、实时性低、运行功耗高等诸多缺点。于此同时具有对硬件可编程、运行功耗低的 FPGA（Field Programmable Gate Array）芯片的规模不断增加，在 FPGA 芯片内实现复杂的数字信号处理系统已成为现实。因此本设计将 JPEG 压缩技术和 FPGA 相结合提升系统的性能，并从实际工程出发，设计出一套 JPEG 编解码系统。完成 JPEG 编解码在 FPGA 上的实现。

本论文的结构首先阐述了 JPEG 格式的编解码 FPGA 实现的研究背景与意义国内外研究现状，并以此提出了一种新型的基于脉动阵列实现 DCT 算法的流水线结构。接着描述了 JPEG 格式编码和解码算法的实现步骤。然后采用 SOC 的设计思想，给出整个硬件系统的内部结构、层次划分，对每个运算步骤进行了详细的描述。最后完成整体的验证。

本设计基于 Xilinx 的 Zynq 系列的 FPGA 的硬件平台，对 CMOS 图像传感器 OV5640 采集的图像数据进行采集和压缩，再通过使用低速串行通信协议传输压缩数据。在 EDA 工具 vivado2019.1 中完成综合仿真、布局布线以及码流生成。

关键词： FPGA；图像压缩；JPEG

ABSTRACT

this is adstract

keywords： FPGA,JPEG

目 录

1 绪 论

1.1 研究背景及研究意义

1.1.1 选题背景

图像数据作为信息的重要载体，在如今这个信息化的社会扮演着不可欠缺的角色相对与文本，图像具有强大的表达能力，通过像素、色彩及形状结构等元素传递出大量的信息。随着高清图像和视频的普及，图像数据的数量呈指数级增长。对高存储空间传输带宽的需求日益严峻。一味地添加设备的存储容量和信道的带宽是不现实的，因此图像压缩技术孕育而生。

目前图像压缩格式应用最广的是 JPGE 压缩格式。它运用图像数据在空间上的冗余性以及人眼对图像的辨别度有限等特性来减少图像的数据量。目前 JPEG 图像大多数使用软件进行编解码。但对于医疗器械、自动驾驶、视频监控等对视频画面逐帧处理要求高的场景，这种方式有着效率低、运行功耗高等诸多缺点。

1.1.2 研究意义

本设计采用 FPGA 高并行计算能力和低功耗的特性设计出一套高效运行的 JPGE 格式编解码系统。该系统能保证图像数据高实时传输地同时保持较低的功耗，适合运它运用在嵌入式系统或边缘计算设备。通过对算法进行硬件级优化，FPGA 能够对 JPGE 压缩的各个环节进行流水线和并行处理。相对于传统的软件开发方式极大地提升了系统的吞吐量和响应速度，同时保持较低的运行功耗。此外这种编解码系统作为软核结合 FPGA 使用使用具有较强的可扩展性，可与大多数视频图像处理系统相结合，助力更多地创新应用开发。

1.2 国内外研究现状

随着近几十年来信息技术地不断发展，图像和视频数据被广泛运用。因此如何降低数据的存储大小和传输系统的负担一直是研究的问题。现如今图像数据压缩编码主要分为两种，一种是针对静止图像在空间在空间的纬度上进行压缩编码的压缩方式，如 JPEG、JPEG-2000、JPEG-LS。另一种是针对多个数据帧在时间纬度上进行压缩的方式如 H.264、H.256 等。其中 JPEG（Joint Photographic Exprts Group，联合专家组）标准有 ISO 在 1991 年提出，之后有相继提出了 JPEG-L、JPEG-MOTION 、JPEG2000 三个图像标准，JPEG-LS 是一种接近无损压缩的压缩格式，其工作简单高效，但是输出的编码率随原图像的改变存在较大的波动。PEG-MONTION 是基于 JPEG 发展起来的，可用于动态图像的压缩，但是压缩率比较低。JPEG2000 是用小波变换代替 JPEG 的离散余弦变换（DCT），在低比特率下，有更良好的图像压缩性能，但是算法的复杂度高，因此在一下追求低负责度的实时应用中，JPEG2000 无法替代 JPEG，在大多数的场合之下，使用 JPEG 就能满足需要了。

国内在 JPEG 的研究和应用方向上主要都是通过软件编程运行在 CPU 实现的。在应用方面，为了解决胶囊内窥镜有效空间和电池容量以及传输实时性要求高的场景，上海交通大学的赵恒阳、刘华 [1] 使用 fpga 芯片对图像进行 JPEG 格式的编码，以 2Mbps 的数据速率进行传输，可以达到 30fps 的视频刷新率。在算法改进方面杜英杰 [2] 针对传统 JPEG 使用固定量表，在压缩图像的视觉质量和压缩率的制衡不能灵活调节的问题。提出了一种基于感知量化

和统计量化的自适应量化算法，可以根据在不同频率信息中应用不同量化步长的方法，实现更高的压缩比和更优的图像压缩效果，并使用 FPGA 进行硬件实现。

国外在 JPEN 格式编解码 FPGA 实现的研究现状有：里斯本高级工程学院的学者 [10] 利用 FPGA 独有的硬件结构可编程的特点，使用了 FPGA 动态局部可重构技术，根据编码的流程对 FPGA 硬件逻辑资源进行分时复用，实现硬件资源利用最大化。采用该方案相对使用传统的静态解决方案在资源占用上节省了 60%，但代价是是运行速度上慢了 9 倍。Shan 等人 [11] 通过行列分解的方式实现 2D-DCT（二维离散余弦变换），并在设计中引入乒乓缓存器。在运行在 100MHZ 的时钟信号下，针对 1920*1080 的图像，最快解码率可达到 30fps 的视频刷新率。Teja 等人针对 2D-IDCT（二维离散余弦逆变换）设计了一种全流水的硬件结构，同时不使用乘法器，降低了 2D-IDCT 模块的资源利用。

1.3 使用芯片简介

本设计使用的芯片型号是 Xilinx 公司 Zynq-7000 系列的 XC7Z020-CLG484。Zynq-7000 系列是 Xilinx 公司推出的全可编程片上系统（All Programmable SOC），其中包含了 PS（Processing System，处理器系统）和 PL（Programmable Logic，可编程逻辑）两部分。

Zynq SoC 整合了 ARM 双核 cortex-A9 处理器和 FPGA 架构，实际上是一个片上系统（System on Chip, SoC），因此使得它不仅具有 FPGA 在能耗、性能、硬件可编程的优点，同时具有处理器软件可编程的优点，以提供强大的系统性能、灵活性与可扩展性。该芯片的可编程逻辑部分基于 Xilinx 28nm 工艺的 7 系列 FPGA。

1.4 开发环境

本设计使用的开发环境是 EDA 工具 Vivado2019.2。

1.4.1 Vivado 简介

Vivado 是 Xilinx 公司开发的集成开发环境，用于数字设计、验证和实现 FPGA 和 SoC 解决方案。Vivado 提供了一个全面的工具集，帮助设计者从硬件设计到软件开发的整个流程，包括：设计与综合、FPGA 的布局布线、仿真和验证、码流生成、ILA（Integrated Logic Anglzer）在线调试、PS 端的软件开发、IP 核集成。Vivado 旨在提示 FPGA 设计的效率，特别是在复杂的系统级集成和高性能应用中。它广泛运用在通信、自动化、医疗、汽车、芯片设计等领域。

2 JPEG 图像压缩相关理论

要对一张静态图像数据进行 JPEG 编解码从而做到压缩和解压，需要经历多个过程。如图 2-1 所示。

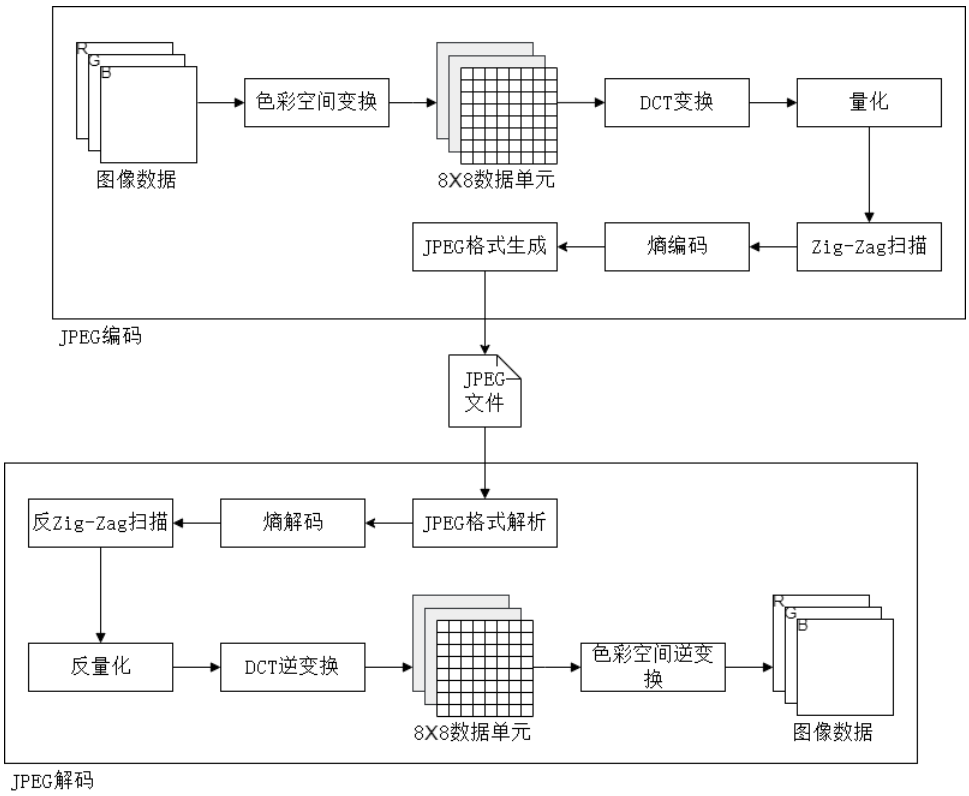

图 2-1 JPEG 的编码和解码流程

JPEG 格式的图像压缩流程为：首先将图像每个像素点的 R、G、B 颜色分量通过色彩空间变换转换成色度分量 Cr、Cb 以及亮度分量 Y。之后将图片划分为若干个 8*8 的数据单元每个单元包含 64 个像素点，再对每个数据单元进行二维离散余弦变换（DCT），将二维的空间域数据变成二维的频域数据。再根据量化表对对应的频域分量进行量化处理。再通过 Zig-Zag 扫描将二维的频域数据转换成一维的序列。最后，依次通过游程编码和霍夫曼编码去除掉冗余的数据从而压缩 JPEG 图像数据。

解压缩流程的流程与压缩的各个流程相反，除了量化和彩色空间变换这一过程会丢失一定的信息之外，其他的步骤都可以无损还原原数据，因此 JPEG 是一种有损压缩技术。

下面依次对各个步骤做详细的描述。

2.1 彩色空间变换及逆变换

绝大多数的颜色都可以使用 R、G、B 三种颜色分量的线性组合进行合成。因此大多图像数据每个像素点都是以 RGB 分量表示。特别是再计算机视频技术中，不管使用哪种形式的彩色空间表示，最后一定要转换为 RGB 彩色空间显示。

相关研究表明，人类的视觉系统有分别对红绿蓝三种颜色敏感层度的三种锥体细胞以及对明暗程度敏感的锥体细胞。其中对明暗程度的锥体细胞的数量大于对颜色敏感层度锥体细胞的数量。因此人类对是对色度辨识度大概是对明暗变换的辨识度的四分之一，因此可以利

用对颜色感知强度的不同，将 RGB 彩色空间变换到 YCrCb 彩色空间再根据感知能力做对应的处理。

ITU-R601 建议规定的 RGB 彩色空间到 YCbCr 彩色空间变换关系如式 (2.1)：

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ 0.500 & -0.419 & -0.081 \\ -0.169 & -0.331 & 0.500 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{2.1}$$

在 JPEG 编码中，RGB 颜色分量的取值范围通常是 0 到 255，因此 JPEG 数据使用的是 8 位无符号整数。而在 YCrCb 颜色空间中，色度的颜色分量 Cr、Cb 的范围为-128 到 127。为了将其范围转换为 0 到 255。故添加偏移量 128，以确保范围在 0 到 255 内。这有助于在 JPEG 编码和解码中正确处理色度信息。这个偏移量是 JPEG 编码标准的一部分，确保了色度信息在 JPEG 图像中正确显示。

彩色空间的逆变换如下式所示：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1.402 & 0. \\ 1 & -0.344 & -0.714 \\ 1 & 0 & 0.500 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} + \begin{bmatrix} 128 \\ 128 \\ 128 \end{bmatrix} \tag{2.2}$$

由于人类的视觉系统对色度变化的感知低于亮度，因此在对色度分量进行采样的时候可以有选择性地进行降采样，进而降低数据量，这也是最朴素的图像压缩技术之一。根据对色度分量的采样率不同分为一下几种采样格式：

YCbCr444：每个分量的采样率都为 1。这意味着对于每一个像素点都进行完整的采样，携带完整的原图片信息，没有信息丢失。因此 YCbCr44 也是最高质量的采样格式。

YCbCr422：在每两个水平相邻的像素中，只用一个 Cr 分量和一个 Cb 分量来表示该这两个像素点的色度分量，而每一个像素点都与之对应的 Y 分量。这种降采样的方式减少了颜色信息的存储和传输需求，在一定程度上牺牲了色度分辨率，但保留了较高的图像质量。

YCbCr420：在每两个水平相邻以及垂直相邻的像素中，只用一个 Cr 分量和一个 Cb 分量表示这 4 个像素点的色度分量，而每一个像素点都有一个与之对应的 Y 分量。这种降采样的方式减少了存储和传输的需求，同时牺牲了色度的分辨率，是广泛应用与图片和视频压缩的以及传输的一种采样格式。本设计采用该采样格式。

再经过采样后需要将采样得到的 Y、Cb、Cr 三种格式分别根据在空间上的分布整合成若干个 8 乘 8 的数据单元。该数据单元是之后 JPEG 编解码各个流程之中的处理单位，称为 MCU（Minimun Coded Unit，最小编码单元）。当使用 YCbCr422 采样格式时单个 MCU 有 2 个 8 乘 8 的 Y 分量单元以及 Cr 和 Cb 分量 8 乘 8 单元各一个。同理，当使用 YCbCr 采样格式时单个 MUC 有 4 个 8 乘 8 的 Y 分量单元以及 Cr 和 Cb 分量 8 乘 8 单元各一个。

2.2 离散余弦变换及逆变换

由于人类的视觉系统在对，图像上亮度以及色度在空间上变化频率高的细节部分的注意力并不高，因此可以将图像上的高频率的信息适当的过滤掉，进而对数据进行近一步的压缩。DCT（Discrete Cosine Transform 离散余弦变换）的作用是将图像从空间域转换到频域。在 JPEG 中，通过对每个 MCU 进行 2D-DCT 从而得到在空间上各个频率代表的正交基分量。2D-DCT

的变换公式如下：

$$
\begin{aligned}
Y(u,v) = \alpha(u)\alpha(v)\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} X(i,j)\cos\left[\frac{\pi}{N}\left(i+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{M}\left(j+\frac{1}{2}\right)v\right] \\
u = 0,1,2,\cdots,N-1 \\
v = 0,1,2,\cdots,M-1
\end{aligned}
\tag{2.3}
$$

其中

$$
\alpha(u) = \begin{cases} \sqrt{\frac{2}{N}} & ,u>0 \\ \frac{1}{\sqrt{N}} & ,u=0 \end{cases} \quad \alpha(v) = \begin{cases} \sqrt{\frac{2}{M}} & ,v>0 \\ \frac{1}{\sqrt{M}} & ,v=0 \end{cases} \tag{2.4}
$$

在 JPEG 中对 8×8 的像素块进行 2D-DCT 变换，所以有 $N=8, M=8$ 。带入 (2.3) 有：

$$
\begin{aligned}
Y(u,v) = \frac{1}{4}C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7} X(i,j)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)u\pi}{16} \\
C(v), C(u) = \begin{cases} 1 & ,u,v>0 \\ \frac{1}{\sqrt{2}} & ,u,v=0 \end{cases}
\end{aligned}
\tag{2.5}
$$

在公式中，$f(i,j)$ 表示在位置 (i,j) 的像素值。其中 $F(0,0)$ 实际上就是对 64 个像素点做加权平均，相当于 8×8 单元的平均亮度，成为 DC（Direct coefficient，直流）系数。其余的 63 个频率值的点称为 AC（Alternation coefficient，交流）系数。在交流系数中距离直流系数点越大代表该点的频率越高。

将频域转换成空间域的变换称为 IDCT（Inverse Discrete Cosine Transform，离散余弦逆变换），2D-IDCT 的表达式如下：

$$
\begin{aligned}
X(i,j) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{u=0}^{7}\sum_{v=0}^{7} Y(u,v)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16} \\
\alpha(v), \alpha(u) = \begin{cases} 1 & ,u,v>0 \\ \frac{1}{\sqrt{2}} & ,u,v=0 \end{cases}
\end{aligned}
\tag{2.6}
$$

2.3 量化及反量化

量化是对 DCT 系数进行压缩的最关键一步，它按照给定的量化系数对每个 DCT 进行除法，再通过四舍五入取整数的方式得到量化后的系数，这个过程是一个一对多的映射过程。也就代表量化后的数据将无法完整地还原回来，因此存在数据丢失。这也是导致 JPEG 有损压缩的原因之一，量化的公式如下：

$$
C(u,v) = round\left[\frac{F(u,v)}{Q(u,v)}\right] \tag{2.7}
$$

其中 $F(u,v)$ 是 2D-DCT 系数，$Q(u,v)$ 是步长值，$C(u,v)$ 是量化后的值。而反量化自然就是将量化后的值乘回步长值，反量化的公式如下

$$F(u,v) = C(u,v)Q(u,v) \tag{2.8}$$

量化是为了将大部分的高频分量都转换为 0，进而减少高频分量的信息，同时也是为了下一步编码作出准备。通过不同的量化表从而控制图像的压缩程度。JPEG 针对色度以及亮度有不同的量化表，如表 2-1和表 2-2所示

表 2-1 亮度量化表

亮度量化表							
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

表 2-2 色度量化表

色度量化表							
17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

对比两个表可以明显地看出，对于亮度步长的划分会更细一点，同时高频分量的值是普遍大于低频部分的。这是由于人眼对色度和高频部分图像信息的辨识能力低于亮度和高频部分。可以把量化当成一个在空间上的二维低通滤波器。

2.4 ZigZag 扫描及反扫描

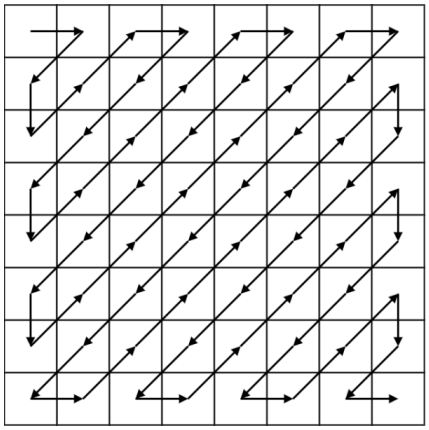

图 2-2 ZigZag 扫描

ZigZag 扫描的过程如图 2-2所示，它对 8×8 单元进行一维上的重排序。这一步骤也可以在量化前执行。对于大部分的 8×8 单元而言，在经过量化后，右下角的高频分量存在大量的零值。为了最大限度地将这些零值相邻为后面压缩做准备，通过 Z 字形扫描对 8×8 单元进行重排序。同理，反扫描即为将 1 维序列排回 8×8 单元。

2.5 熵编码及熵解码

图像数据的信息冗余量主要体现来两个层面：一是图像数据中各个相邻数据之间存在着一定的关联性。在一点体现在空间域上就是相邻的像素点之间的差距一般并不是很大，体现在频域上则是在经过量化和 ZigZag 扫描之后的数据中存在大量连续的零值；二是图像中不同的像素值的概率分布通常是不均匀的，某些像素值的出现的概率较高。如果对出现概率较高的数据采用跟少的位数进行编码，则能在一定程度减少一定的数据量。熵编码的主要目的为通过信息熵理论来减少这些冗余的信息，从而降低图像数据在传输和存储中所占用的时间和空间。同时熵编码是可逆的属于无失真压缩。

在量化后。对于 DC 系数和 AC 系数两者在统计性质有很大的不同，因此采用两种不同的编码方式，对于 DC 系数采用差分脉冲编码（differential Pluse Code Modulaiton，DPCM），对于 AC 系数则使用游程编码（Run-Length Encodeing，RLE）。这两种编码方式通过在空间和频率上各个相邻数据的相似性的特点进行编码。在霍夫曼编码之前，能有效地减少图像数据的冗余性，以实现更高效的图像压缩。

2.5.1 DC 系数差分脉冲编码

DC 系数的值通常会比 AC 的值要大，而 DC 系数可以认为是每个 8×8 单元的平均值。而在空间上相邻的 8×8 单元平均值的差异通常不大。为了充分利用这一特点，JPEG 采用了差分脉冲编码，通过对当前的 8×8 单元的 DC 系数与前一个 8×8 单元的 DC 系数的差值进行编码。设 DC_{diff} 当前的 8×8 单元的 DC 系数 DC_i 减去前一个 DC 系数 DC_{i-1},DC_0 表示第一个 8×8

单元的 DC 值。公式如下：

$$DC_{diff} = \begin{cases} DC_i - DC_{i-1} & ,i > 0 \\ DC_0 & ,i = 0 \end{cases} \tag{2.9}$$

DC_{diff} 的编码格式为（Size，Amplitude）。其中 Size 为 Amplitude 的位宽值，而 Amplitude 为 DC_{diff} 的幅值，当幅值为正是为原码。反正则为补码。因此 Amplitude 没有符号位。根据 Size 和 Amplitude 的第一位来判断 DC_{diff} 的正负。

2.5.2 AC 系数游程编码

在经过 ZigZag 扫描之后，AC 系数通常会出现大量连续的零值。也就可以通过连续 0 的个数来进行编码，这种方式称为游程编码。游程编码可以有效地表示续出现的相同的值，用该数值本事加上该数值的重复次数来替代。进而减少数据量来做到数据压缩。游程编码的编码格式为（Run，Size，Amplifier）。其中 Run（游程）表示非零值前面零值的个数，Size 表示非零值的尺寸，Amplifier 表示非零值的幅值。同理，也可以根据这三个值还原回原码，从而做到解码。

有两种特殊的情况需要注意：当出现从某个非零值直到最后第 64 个值都为零时的情况，使用 0/0（EOB）进行编码。当连续零值的数量超过 16 个时，使用 F0（ZRL）进行编码。

2.5.3 霍夫曼编码

霍夫曼编码是一种可变长度编码，这种编码方法由霍夫曼（Huffman）在 1952 年提出。它根据字符的出现的概率来构建编码映射。以实现码字的平均长度最短。该编码方式的压缩率接近香农所定义的极限压缩率，因此这种方法也称为最佳编码。通过查找 AC 和 DC 系数对应颜色分量的 Huffman 表进行编码。Huffman 码表是由 JP EG 标准通过大量的图像数据统计进而规定的，详细内容如下表所示。

表 2-3 亮度 DC_{diff}Huffman 表

尺寸	Huffman 码长度	Huffman 码字
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
A	8	11111110
B	9	111111110

表 2-4 色度 DC_{diff}Huffman 表

尺寸	Huffman 码长度	Huffman 码字
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
A	8	11111110
B	9	111111110

表 2-5 亮度 AC 系数 Huffman 表

游程/尺寸	Huffman 码长度	Huffman 码字
0/0(EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
...	...	...
F/A	16	1111111111111110

表 2-6 色度 AC 系数 Huffman 表

游程/尺寸	Huffman 码长度	Huffman 码字
0/0(EOB)	2	00
0/1	2	01
0/2	3	100
0/3	4	1010
0/4	5	11000
0/5	5	11001
0/6	6	111000
0/7	7	1111000
0/8	9	111110100
0/9	10	1111110110
0/A	12	111111110100
1/1	4	1011
...	...	...
F/A	16	1111111111111110

在解码时，当多个 Huffman 串行码流排列输出在一起时，可以使用二叉树解索引从而获取对应原码从而做到解码得到尺寸，再通过尺寸得到幅值所占的的位宽得到幅值。通过这种方式从而区分出码流中的各个像素点的数据。

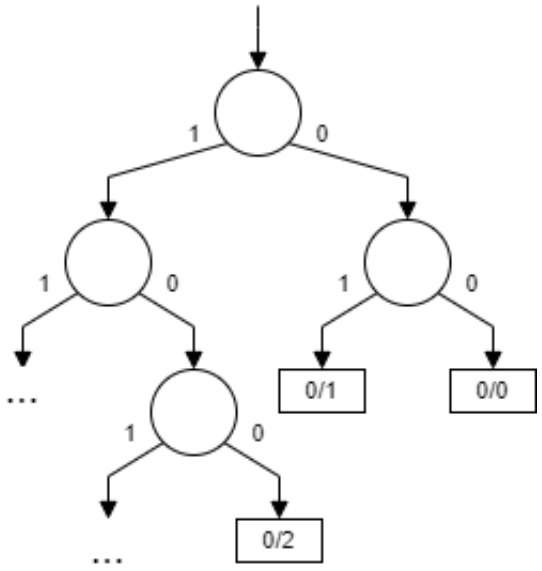

图 2-3 通过二叉树解码

2.6 JPEG 文件格式

通过上述过程得到了压缩之后的图像数据，除此之外，要对数据进行解码。还需要知道图像的相关属性等信息。JPEG 委员会在指定 JPEG 标准时，定义了许多用来区分图像数据及其相关信息的文件格式。目前，使用比较广泛的是 1992 年 9 月由 E ric Hamilton 提出的 JPEG 文件交换格式 JFIF（JPEG File Interchange Format）1.02 版本。大多数的设备都支持 JFIF 文件交换格式。JPEG 编码的最后一个步骤就是把各种标记代码和编码后的图像数据组成一帧一帧的位码流，这样就方便传输、存储和解码器译码。

JFIF 文件格式可以分为两个部分：标识和压缩数据。每个标识的前一个字节是固定值 0xFF。每个标记之前还可以添加数量不限的 0xFF。下表列举几种常见的标记码以及它们各自的数据结构。

表 2-7 几种常见的标记

标记种类	含义
SOI	图像的开始
APP0	JFIF 引用的数据块
SOF0	帧开始
DHT	Huffman 表
SOS	扫描线开始
EOI	图像结束

表 2-8 APP0 标识

标识结构	字节数	含义
0xFF	1	APP0 标识
0xE0	1	

表 2-9 OF0 标识

标识结构	字节数	含义
0xFF	1	APP0 标识
0xC0	1	
L_f	2	长度字段，表示图像帧信息长度
P	1	指每个像素点的颜色信息的宽度，通常是 8 位或 12 位
Y	2	图像的高度
X	2	图像的宽度
N_f	1	图像颜色通道的数量。通常为 1 或 3，分别表示灰度图和 RGB 彩色图
N_{NT}	1	颜色通道，0 表示 Y 通道，1 表示 Cb 通道，2 表示 Cr 通道
$H_{TN}Y_{TN}$	1	水平方向和垂直方向的采样率
T_{QNT}	1	表示使用的 Huffman 编码表的编号

表 2-10 DHT 标识

标识结构	字节数	含义
0xFF	1	APP0 标识
0xC4	1	
L_b	2	长度字段，表示 Huffman 码字段的长度
T_c	0.5	当为 1 时，表示使用该表处理 AC 系数，为 0 时，表示该表处理 DC 系数
T_b	0.5	表示 Huffman 表的编号
L_i	1	Huffman 表的长度统计，用于表示不同码字长度的符号数目，i 从 1 到 16
V_{ij}	1	代表每一个 Huffman 码表所代表的值

表 2-11 SOS 标识

标识结构	字节数	含义
0xFF	1	APP0 标识
0xDA	1	
L_s	2	长度字段，表示数据内容的长度
N_s	1	表示扫描所涉及到颜色通道的数量
C_sN_s	0.5	表示 Scan 中成分的编号
T_dN_s,T_aN_s	1	T_aN_s 表示数据的高 4 位 T_aN_s 表示数据的低 4 位
S_s	1	一般为 0
S_s	1	一般为 63
A_b,A_l	1	一般为 0

表 2-12 EOI 标识

标识结构	字节数	含义
0xFF	1	EOI 标识
0xD9	1	

2.7 本章总结

本章阐述了 JPEG 图像数据编解码的原理和相关理论, 以及各个执行的步骤。

3 JPEG 编码系统硬件结构设计

上一章节讲解了 JPEG 编码的步骤和过程。这一章讲解本设计如何使用硬件实现 JPEG 的编码以及解码。内容包含数字电路设计中的一些设计思想，编码系统的结构划分，以及各个模块的结构和运行原理。

3.1 设计思想

在数字系统设计中，一个经常围绕的问题就是速度和面积的权衡。使用多个处理单元对数据进行处理是一种朴素的提高系统计算速度和吞吐量的方式。与之带来的副作用就是电路面积的增大。面积增大所带来的副作用不仅仅是成本增加的问题，它也伴随着系统功耗和发热量的增加，这些负面影响都会降低系统的稳定性。而有些运算过程数据依赖性高、并行操作无法有效地提高性能。甚至可能因为额外的开销导致性能的下降。因此，一个好的数字系统设计往往能够权衡速度和面积。

通过前人的大量经验实践总结出了大量的设计思想。如果能在合适的情况下使用这些设计思想，就能得到一个好的设计。下面描述变设计所使用的几种设计思想。

3.1.1 流水线

如果一个运行周期长的操作的运行过程能个分解成一各个子步骤。并将这些子步骤封装成一个个并行的模块。让每个操作的不同步骤在不同时刻错开运行，每个模块同时执行不同操作所对应的步骤。最终每经过一个子步骤的时间，就可以得到一个操作的运行结果，通过这种方式系统的吞吐量能得到大幅度提高，在运行过程中每个步骤的结果数据一步步的从输入流动到输出，因此该操作得名流水线（Pilpeline）。这种方式在如今的工厂车间运用广泛。

下面通过对比一个分 3 个步骤的执行的例子，来分析流水线所带来的性能的提升。假设一个系统的执行需要通过步骤一、步骤二、步骤三三个步骤的到结果，每个步骤所花费的时间为一个时钟周期。如果依次按照顺序完成 N 次运算，每隔 3 个时钟周期得到一个运行结果，完成所有运算所花费的时间就是 3N。如果设置三个执行不同步骤的模块同时并行执行，且每个模块执行不同操作的步骤，如第一个时刻执行步骤一的模块执行操作一的步骤一，如第二个时刻执行步骤一的模块执行操作二的步骤一，执行步骤二的模块执行操作一的步骤二；第三个时刻执行步骤一的模块执行操作三的步骤一，执行步骤二的模块执行操作二的步骤二，执行步骤三的模块执行操作一的步骤三；之后 N 个时刻以此类推在第 4 个时刻得到操作一的结果，下一时刻得到操作二的结果。之后每经过一个时刻得到一个操作结果。得到 N 操作的结构则需要花费 3+N 个时钟周期的时间。通过这种方式显著地提升数据的吞吐量。

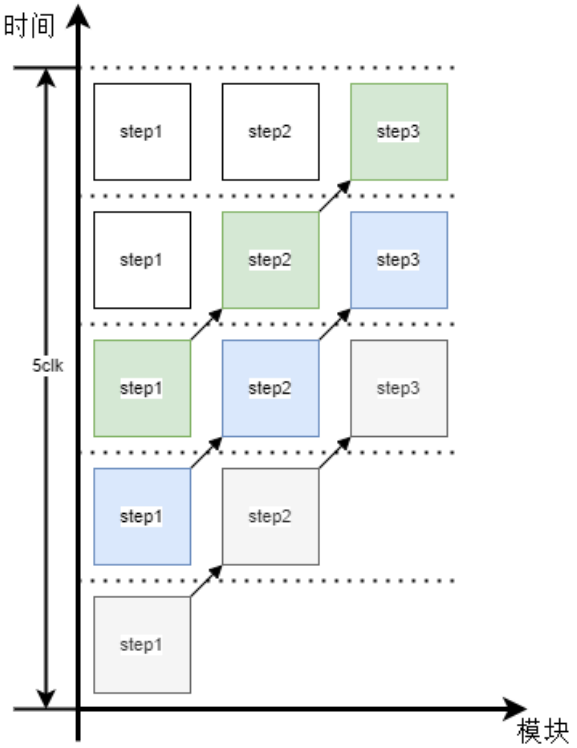

图 3-4 流水线操作

除此之外，流水线通常用于提高数字电路的最大运行时钟频率。在时序电路中时钟周期至少要大于保持时间、建立时间、组合逻辑延迟时间三者之和。而组合逻辑延迟时间取决于组合逻辑中最长的电路路径。如果在组合逻辑之中插入寄存器，从而切分组合逻辑的关键路径，使组合逻辑在每个时钟周期安装流水线的方式运行。通过这种方式提高时序电路的最大运行时钟频率。从而提高整个电路的运算速度。

3.1.2 脉动阵列

脉动阵列（Systolic Array）是一种由众多简单的运算元件（Processing Element，PE）按照一定的规则排列的硬件架构。它最早由 H.T. Kung 在 1982 年提出。一个脉动阵列具备一下特征：

(1) 由单一或多种构造的 PE 按照规则排列；

(2) 只有相邻的 PE 互相连接，数据只能通过在局部范围内移动；

(3)PE 只重复进行简单的数据处理和必要的数据收发；

(4) 所有 PE 由统一的时钟同步工作；

每个 PE 都和相领的 PE 同步进行数据收发和运算。数据从外部流入，PE 阵列一边搬运数据，一边采用流水线或并行的方式对其进行处理。各个 PE 的运算和数据的收发动作和心脏规律地收缩促使血液流动的过程非常相似，因此得称脉动阵列。

由于脉动阵列的数据移动只在相邻的 PE 中进行，这种方式有利于芯片或 FPGA 的布局步线，脉动阵列根据排列和连接方式，主要可分为串行的一维脉动阵列，网格连接的二维脉动阵列。其中一维的脉动阵列能实现 FIR 滤波器，向量乘法，数据排序。二维脉动阵列能完成矩阵运算，如图 3-6。

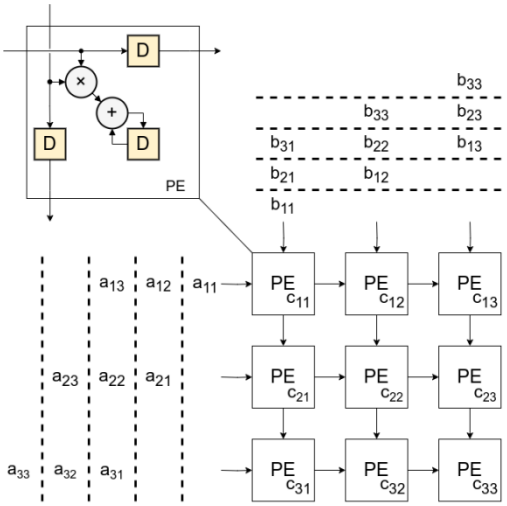

图 3-5 二维脉动技术矩阵乘法

3.1.3 乒乓操作

在两个功能模块进行数据传输时，如果两个模块的吞吐量不同，对数据接收的顺序存在差异等原因。导致两个模块不能够同时工作。这个情况下就可以在两个模块之间添加乒乓缓存（Ping-pong buffer）来解决这个问题，这种解决方法被称为乒乓操作。

乒乓操作的处理流程，在两个模块之间添加 2 个 buffer，来进行缓存数据。这个 buffer 通常是 FIFO（First in first out，先入先出队列）或者 RAM。初始状态下，数据发送模块先向一个 buff er 写入发送的数据，当 buffer 的容量满后，数据接收模块开始向这个 buff er 读出数据。同时数据模块继续向另外一个 buffer 进行写入。之后依次类推，两个 buffer 交错读写。这样，只有在初始时数据发送模块在往第一个 buffer 写入数据时，接收模块是空闲状态，在其余时间内，两个模块都是同时工作的。

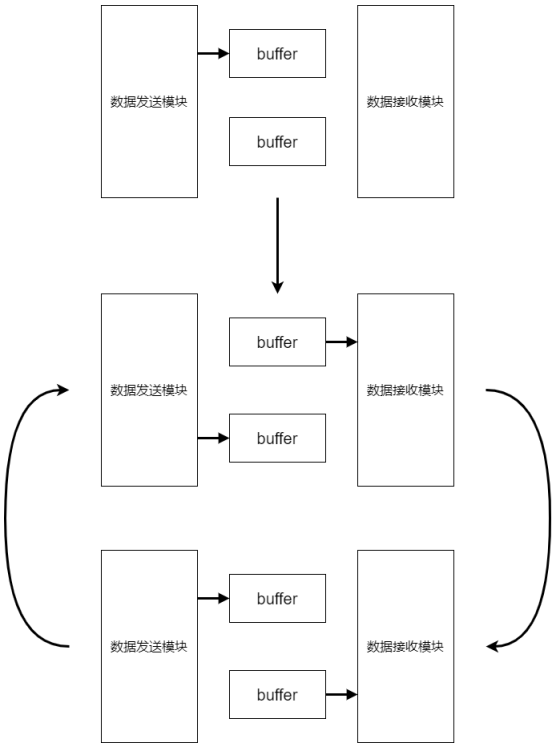

图 3-6 乒乓操作

3.2 编码模块划分

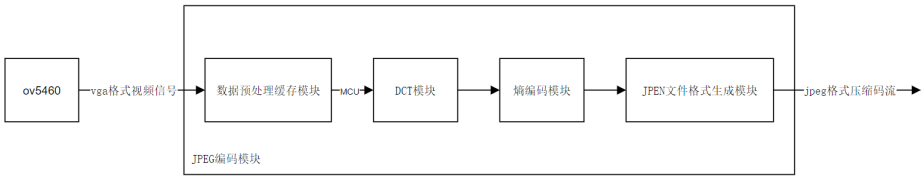

图 3-7 编码模块

JPEG 编码模块的内部模块划分如图 3-8，它接收 ov5460CMOS 图像传感器的 vga 格式的视频信号。输出 jpeg 格式的压缩码流。视频信号依次经过数据预处理缓存模块、DCT 模块、熵编码模块、JPE G 文件格式生成模块。下面的小节详细介绍这 4 个模块。

3.3 数据预处理模块

3.4 DCT 模块

DCT 模块接收数据预处理模块输出的，Y、Cr、Cb 三个颜色通道的 8* 8 像素单元。并对每个颜色通道实现进行 DCT、量化、Zigzag 扫描这三个步骤。

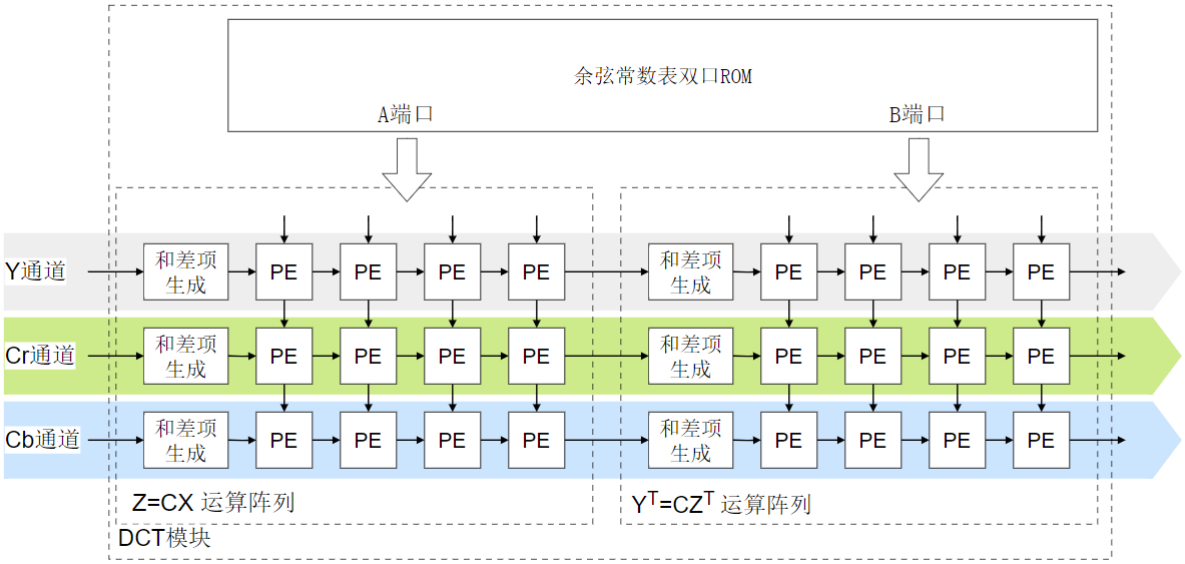

图 3-8 DCT 模块内部结构

如果采用 YCrCb444 以外的采样格式，则 Cr、Cb 可以复用一个通道，减少资源的消耗。

根据 2D- DCT 原公式 (2.5)，对单个 8×8 单元进行运算，需要进行 4096 次乘法和 40 96 次加法加法运算。需要使用较多的大量的乘法器资源，而乘法器资源在 fpga 中比较稀缺。为了降低计算的复杂度，本设计使用了 DA 算法，通过行列分解 2D-DCT 运算。

3.4.1 DA 算法

根据式 (2.5) 的特征可以根据行和列分解成两次一维的 DCT 变换，计算过程如下：

$$Y(u,v) = \frac{1}{4}C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7}X(i,j)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}$$

$$Y(u,v) = \frac{1}{2}C(u)\sum_{i=0}^{7}\cos\frac{(2i+1)u\pi}{16}\frac{1}{2}C(v)\sum_{j=0}^{7}X(i,j)\cos\frac{(2j+1)v\pi}{16}$$

令

$$Z(i,v) = \frac{1}{2}C(v)\sum_{j=0}^{7}X(i,j)\cos\frac{(2j+1)v\pi}{16} \tag{3.1}$$

有

$$Y(u,v) = \frac{1}{2}C(u)\sum_{i=0}^{7}Z(i,v)\cos\frac{(2i+1)u\pi}{16} \tag{3.2}$$

下面再将式 (3.1) 和式 (3.2) 整理成矩阵运算，令 C 为带常数项的系数矩阵，则有

$$C = \begin{bmatrix} a & a & a & a & a & a & a & a \\ b & d & e & g & -g & -e & -d & -b \\ c & f & -f & -c & -c & -f & f & c \\ d & -g & -b & -e & e & b & g & -d \\ a & -a & -a & a & a & -a & -a & a \\ e & -b & g & d & -d & -g & b & -e \\ f & -c & c & -f & -f & c & -c & f \\ g & -e & d & -b & b & -d & e & -g \end{bmatrix}, \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \cos(4\pi/16) \\ \cos(\pi/16) \\ \cos(2\pi/16) \\ \cos(3\pi/16) \\ \cos(4\pi/16) \\ \cos(5\pi/16) \\ \cos(6\pi/16) \end{bmatrix} \tag{3.3}$$

则式 (3.1) 和式 (3.2) 的矩阵形式分别为

$$Z = CX \tag{3.4}$$

$$Y = (CZ^T)^T \tag{3.5}$$

整理得

$$Y^T = C(CX)^T \tag{3.6}$$

通过上述步骤，将 8×8 的 2D-DCT 分解成两次对 C 的左乘运算。对于 Z 的每个列向量有

$$\begin{bmatrix} z_{i0} \\ z_{i1} \\ z_{i22} \\ z_{i3} \\ z_{i4} \\ z_{i5} \\ z_{i6} \\ z_{i7} \end{bmatrix} = \begin{bmatrix} a & a & a & a & a & a & a & a \\ b & d & e & g & -g & -e & -d & -b \\ c & f & -f & -c & -c & -f & f & c \\ d & -g & -b & -e & e & b & g & -d \\ a & -a & -a & a & a & -a & -a & a \\ e & -b & g & d & -d & -g & b & -e \\ f & -c & c & -f & -f & c & -c & f \\ g & -e & d & -b & b & -d & e & -g \end{bmatrix} \begin{bmatrix} x_{i0} \\ x_{i1} \\ x_{i22} \\ x_{i3} \\ x_{i4} \\ x_{i5} \\ x_{i6} \\ x_{i7} \end{bmatrix} \tag{3.7}$$

观察到 C 具有左右对称的性质，可将 (3.7) 分解如下

$$\begin{bmatrix} z_{i0} \\ z_{i2} \\ z_{i4} \\ z_{i6} \end{bmatrix} = \begin{bmatrix} a & a & a & a \\ c & f & -f & -c \\ a & -a & -a & a \\ f & -c & c & -f \end{bmatrix} \begin{bmatrix} x_{i0} + x_{i7} \\ x_{i1} + x_{i6} \\ x_{i2} + x_{i5} \\ x_{i3} + x_{i4} \end{bmatrix}, \begin{bmatrix} z_{i1} \\ z_{i3} \\ z_{i5} \\ z_{i7} \end{bmatrix} = \begin{bmatrix} b & d & e & g \\ d & -g & -b & -e \\ e & -b & g & d \\ g & -e & d & -b \end{bmatrix} \begin{bmatrix} x_{i0} - x_{i7} \\ x_{i1} - x_{i6} \\ x_{i2} - x_{i5} \\ x_{i3} - x_{i4} \end{bmatrix} \tag{3.8}$$

Z 每个列向量的偶项等于 $x_n + x_{7-n}$（下面统称为和项）的线性组合，奇项等于 $x_n - x_{7-n}$ 的线性组合。对于 $Y^T = CZ^T$ 也是同理。

通过上述步骤将 8 乘 8 的矩阵运算转换为两次 4 乘 4 的矩阵运算。减少了计算的复杂度减少了一倍。为了在硬件中使用定点数运算，将中各个系数都扩大 2^{10} 倍, 相当于将系数左移 10 位，再将输出的乘积右移进行截位得到结果。

3.4.2 DA 算法的硬件实现

对于 DA 算法中 2 次 4 乘以 4 的矩阵运算，本文提出了一种基于脉动阵列的硬件实现。如图 3-8所示。对于单个颜色通道，可以认为一维脉动阵列如图 3-8为单个颜色通道计算 $Z = CX$ 的结构。图 3-8 中输入以 $x_{00}, x_{07}, x_{01}, x_{06}, x_{02}, x_{05}, x_{03}, x_{04}$ 的顺序依次输入到和差项生成模块生成 $x_{00} + x_{07}, x_{00} - x_{07}$ 到 $x_{03} - x_{04}, x_{03} + x_{04}$, 的和差项。再将和差项送进一行四列的脉动阵列。得到 z_{00} 到 z_{07}。

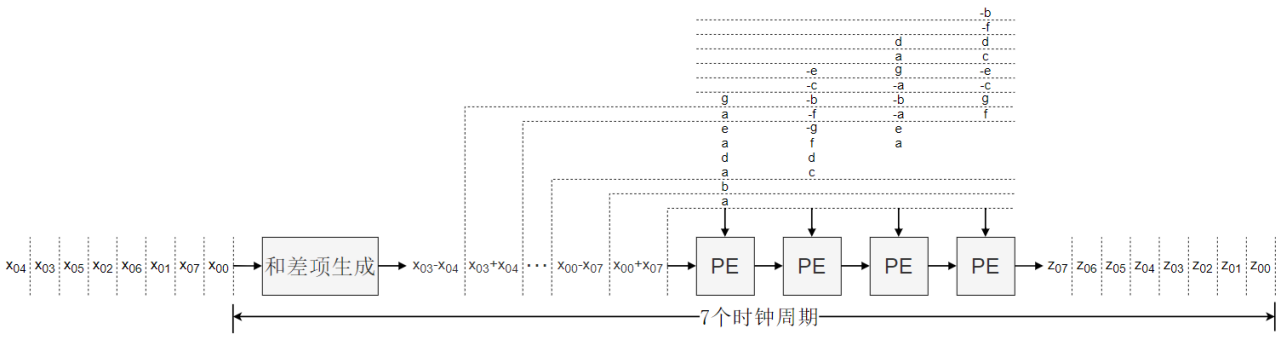

图 3-9 计算 $Z = CX$ 的一维脉动阵列实现

计算 $Z = CX$ 和 $Y^T = CZ^T$，脉动阵列的原理一致，区别在于内部移位寄存器延迟的时

钟周期数不同，两个矩阵运算的 PE 内部构成如图 3-10和图 3-11 所示

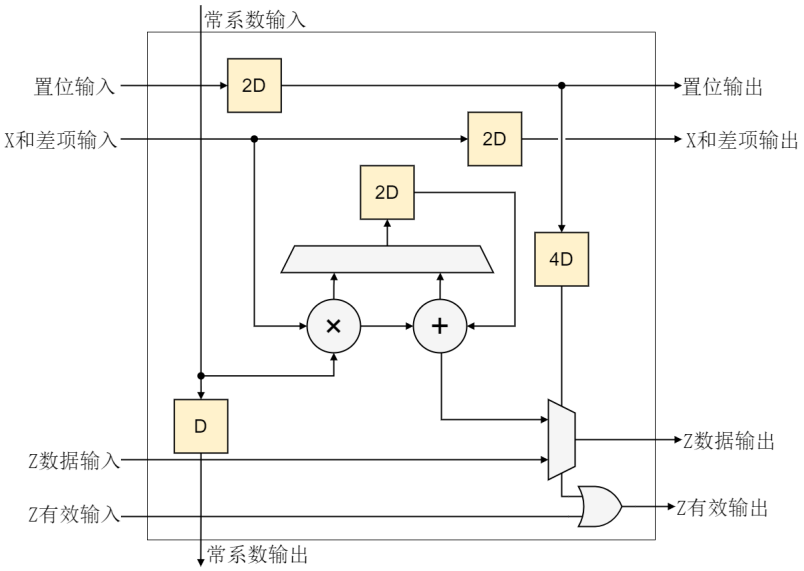

图 3-10 $Z = CX$ 的 PE

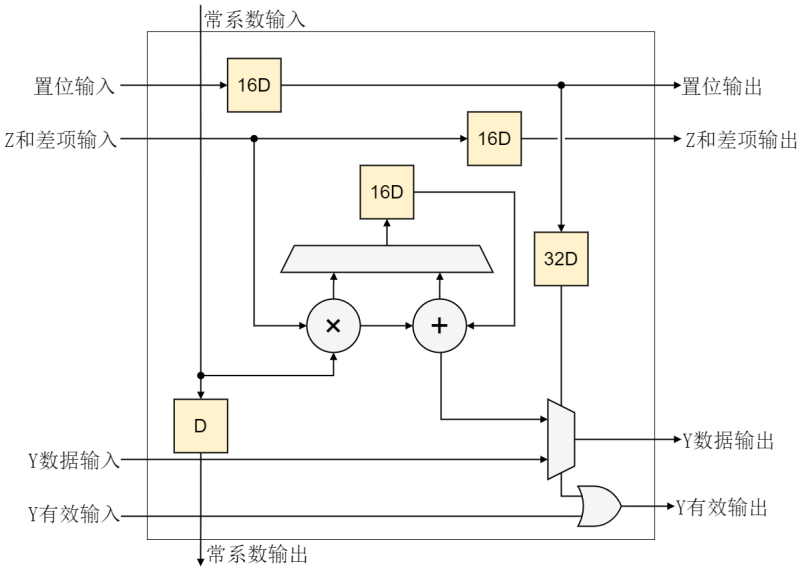

图 3-11 $Y^T = CZ^T$ 的 PE

以 $Z = CX$ 中列向量元素构成的和差项 $x_{00} + x_{07}, x_{00} - x_{07}$ 到 $x_{03} - x_{04}, x_{03} + x_{04}$ 得到 z_{01}, z_{02} 的过程为例，描述单个 PE 的运行原理。当 $x_{00} + x_{07}, x_{00} - x_{07}$ 输入时，置位输入有效。将其与余弦常数项输入相乘得到乘积项，再将其送进延迟 2 拍的移位寄存器进行暂存。当后续输入 $x_{01} + x_{06}, x_{01} - x_{06}$ 到 $x_{03} + x_{04}, x_{03} - x_{04}$ 输入时。置位输入无效移位寄存器输入为乘积和延迟 2 拍的输出进行累加。当 $x_{03} + x_{04}, x_{03} - x_{04}$ 累加后得到 $z_{00}, z01$。从 $x_{00} + x_{07}$ 到 z_{00} 的生成经过 6 个时钟周期，因此将置位信号延迟 6 拍得到 z 数据输出 MUX 的选择信号。具体的运行时序如图 3-12所示。

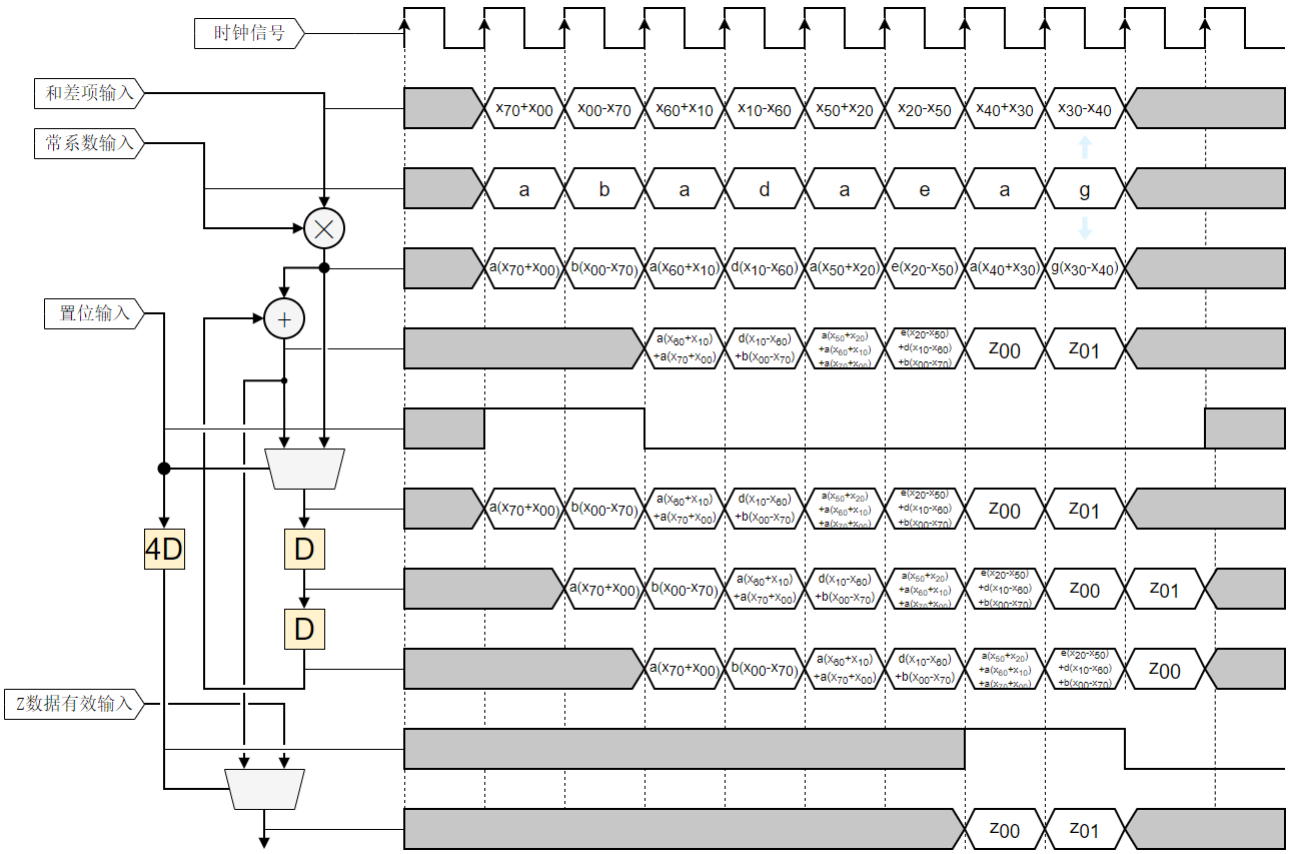

图 3-12 $Z = CX$ 阵列 PE 运行时序

除第一列以外，每个 PE 的输入来自水平相邻 PE 的输出。除了第一行的颜色通道之外，每个 PE 的常系数输入来自垂直相邻 PE 的输出。第一行的常系数来自 ROM。第 n 列的 PE 运算 $z_{0,2(n-1)}, z_{0,2(n-1)+1}$。为了输出 z_{00} 到 z_{07} 的串行流。如图 3-10 所示，水平方向 x 数据延迟 2 拍，使运算结果 z 在时间上不重叠。再通过每个 PE 中 z 数据级联 MUX 和有效信号级联或门的方式使最后一列 PE 的 Z 数据输出 z_{00} 到 z_{07} 的串行流对应时序如图 3-13所示。

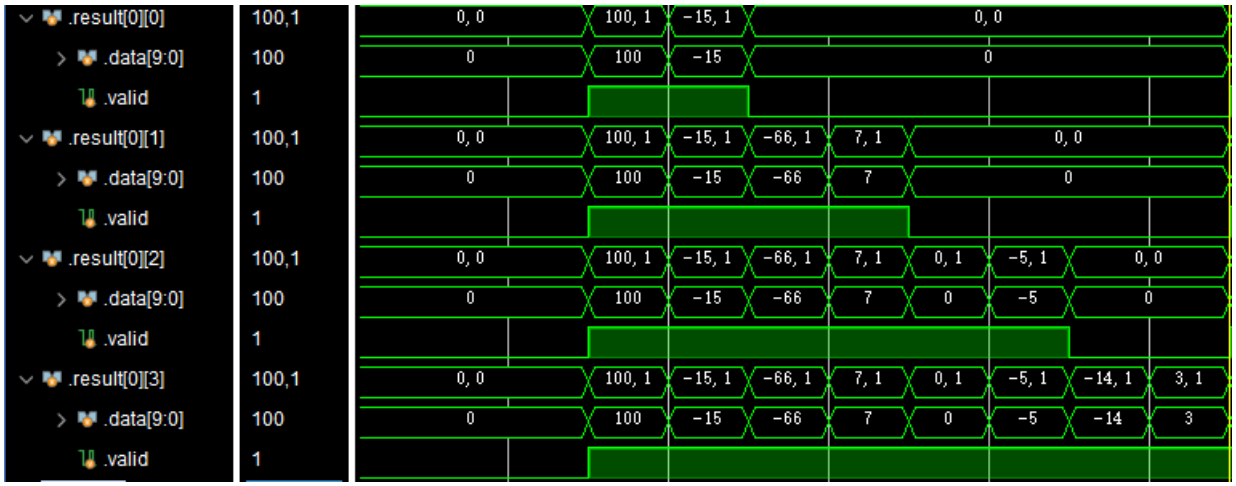

图 3-13 z 输出时序

通过上述过程可以得到 Z, 还需要再计算 $Y^T = CZ^T$,$Z = CX$ 阵列依次输出 Z 的列向量 $Z_0, Z_7, Z_1, Z_6, Z_2, Z_5, Z_3, Z_4$ 所以需要通过 Z_n, Z_{7-n} 输入到和差项生成模块得到 Z^T 列向量的和差项, 下面通过此过程描述和差项的运行原理。

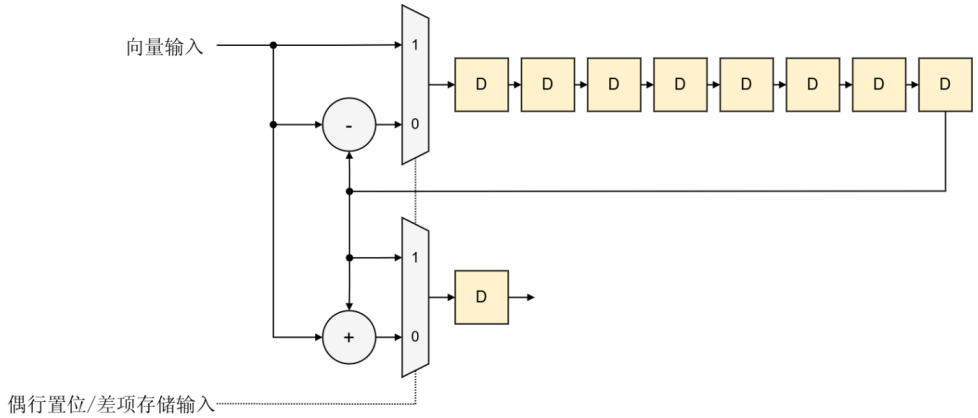

图 3-14 $Y^T = CZ^T$ 和差项生成模块

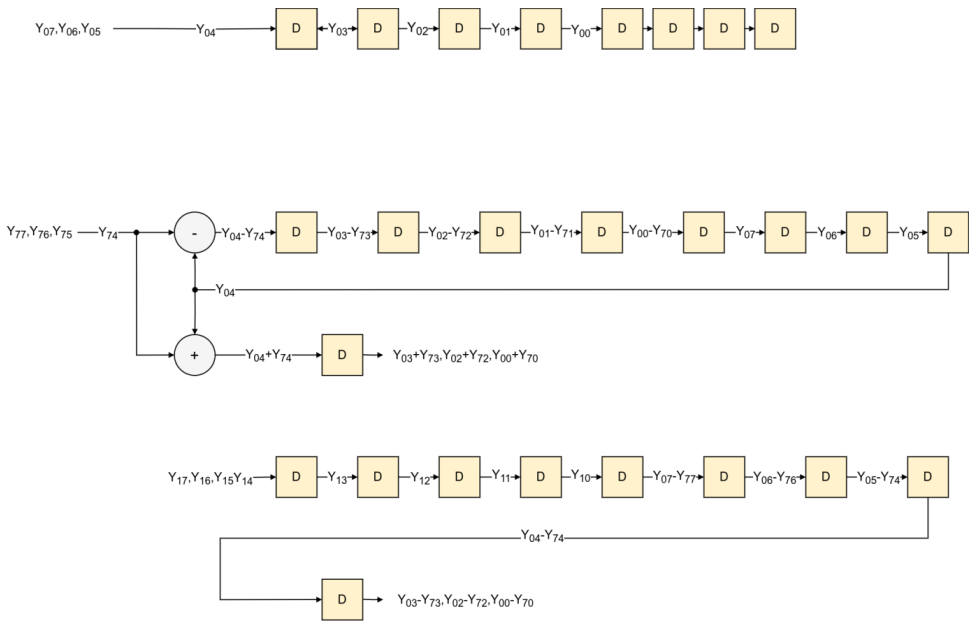

图 3-15 Z^T 和差项生成步骤

该结构的运行步骤如图所示，当的第 1 行（Z_{00} 到 Z_{07}）输入时，偶行置位为 1。将该行输入到 8 拍的移位寄存器上；当的第 8 行（Z_{70} 到 Z_{77}）输入时，偶行置位为 0。将移位寄存器输出的 1 行和 8 行输入到减法器并送回到移位寄存器，同时输入到加法器输出和项；当 Y 的第 2 行（Z_{10} 到 Z_{17}）输入时，偶行置位为 1 将该行输入到 8 拍的移位寄存器上，同时移位寄存器输出上一步骤计算的差项。其他行之间的和差项运算同理。

3.4.3 DCT 模块功能仿真结果

本设计使用 systemverilog 进行 RTL 级描述。在 EDA 软件 vivado 进行综合。DCT 模块中运算 DCT 的模块 RTL 视图如图 3-16所示。其中 in[0]，out[0]、in[1]，out[1]、in[2]，out[2] 表示三个颜色通道的输出输出。

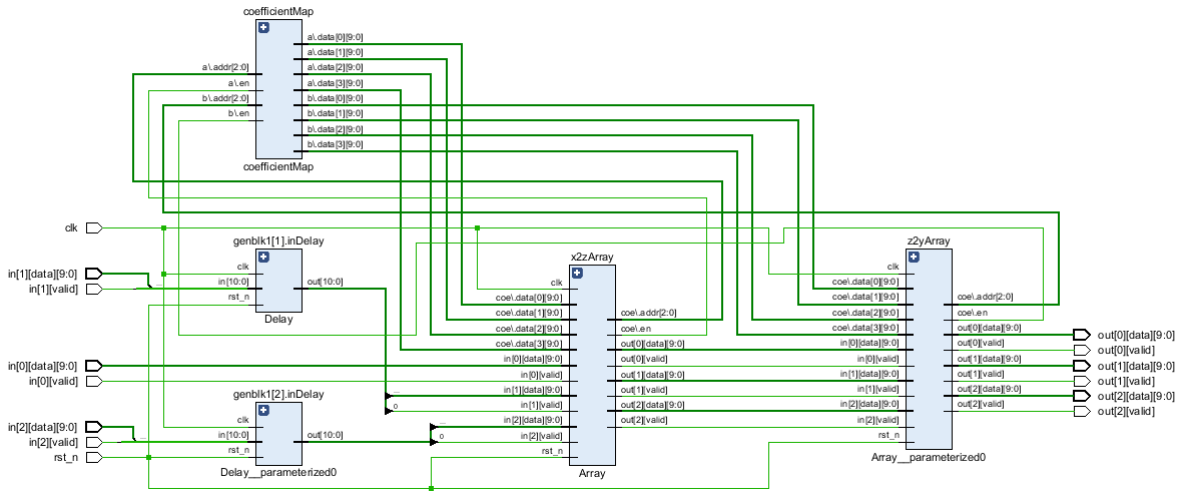

图 3-16 DCT 运算部分 RTL 视图

下面以一个 8×8 单元输入为激励进行仿真，仿真后的波形如图 3-16 其中灰色波形为 Y 通道，绿色波形为 Cr 通道，蓝色波形为 Cb 通道。由图可知从 X 到 Z 的时滞为 7 个时钟周期，从 Z 到 Y 的时滞为 56 个时钟周期，所以整模块的时滞为 63。由公式（2.5）可知输出 $Y(u,v)$ 的每个元素都需要 $X(i,j)$64 个元素进行计算，所以当数据串行输入时，理论极限最小时延为 63 个时钟周期，本设计时沿达到理论极限。

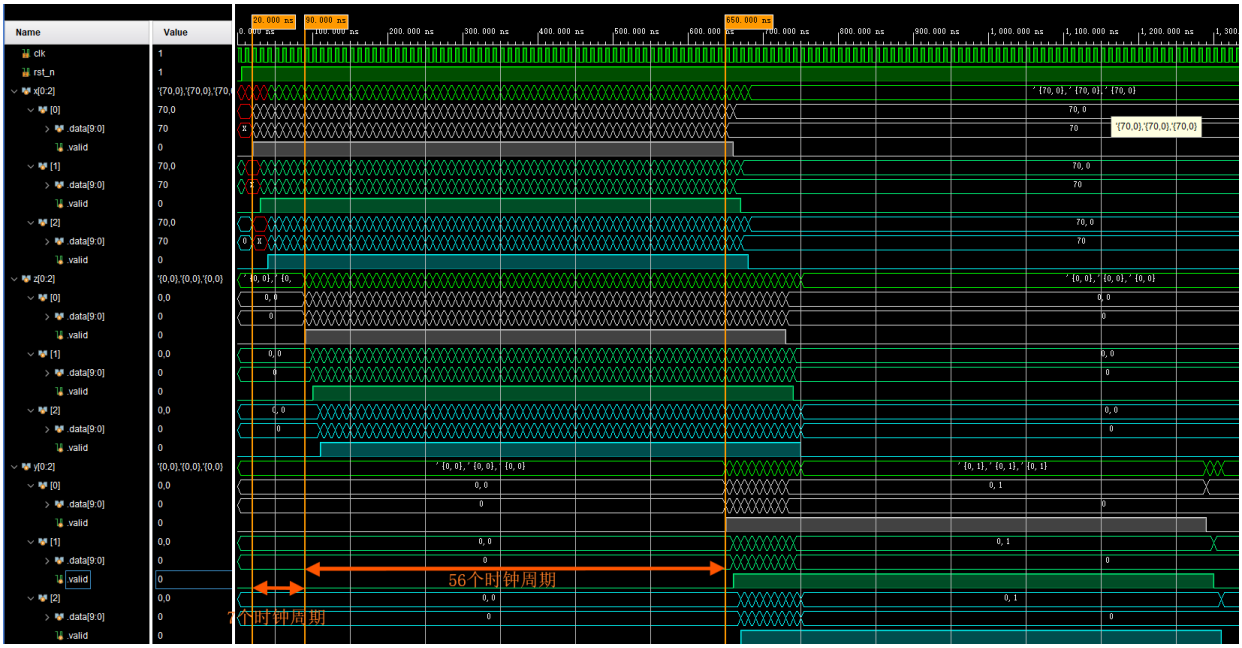

图 3-17 DCT 模块仿真波形

下面验证计算结果的准确性，以如式（3.9）矩阵作为输入。与使用 python 的 numpy 库编

写的 DCT 计算结果进行对比。

$$\begin{bmatrix} 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\ 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\ 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\ 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\ 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\ 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\ 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\ 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \end{bmatrix} \tag{3.9}$$

```
C((CX^T)^T)
[[ 280.   -191.3    -0.    -3.64    0.    -28.64    0.    -9.29]
 [  -0.      0.     -0.     0.      -0.    -0.       0.     0.  ]
 [   0.      0.      0.    -0.      -0.     0.       0.    -0.  ]
 [  -0.      0.      0.    -0.       0.     0.       0.     0.  ]
 [   0.     -0.      0.     0.      -0.    -0.       0.    -0.  ]
 [   0.     -0.     -0.    -0.      -0.     0.       0.     0.  ]
 [  -0.      0.     -0.     0.       0.     0.       0.     0.  ]
 [   0.      0.     -0.    -0.       0.    -0.       0.     0.  ]]
```

图 3-18 python 脚本运行结果

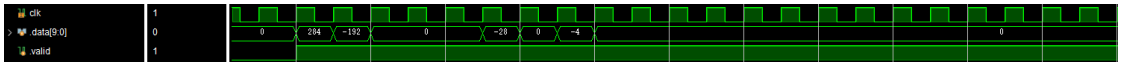

图 3-19 DCT 仿真结果波形

图 3-18为 python 脚本运行结果，图 3-19为 DCT 模块计算结果的波形，可以粗略地估计模块的运算结果和实际的运算结果存在 ± 5 的误差。这是由于对定点数运算结果进行截位导致的。本模块使用 10 位数据进行运算因此误差的

3.5 Zigzag 扫描模块

由于 DCT 模块输出的 y_{ij} 的顺序为从左到右，从上到下。所以要进行 zigzag 排序，需要把数据先写入到 RAM 进行暂存，经过一段延时实际后，再使用 zigzag 模块进行读出。为了最大限度提高数据的实时性，使用 python 脚本对写入和读出进行建模仿真。计算出写入和读出的延时时间最少为 27 个时钟周期。

Zigzag 扫描模块的内部结构划分如图 3-20所示。当输入数据有效时，计数器开启计数，计数值作为 RAM 的写地址。地址高低 3 位分别为 y，x 坐标。由此实现水平扫描写入。当计数值大于等于延迟时间时，启动 zigzag 扫描器读出数据。

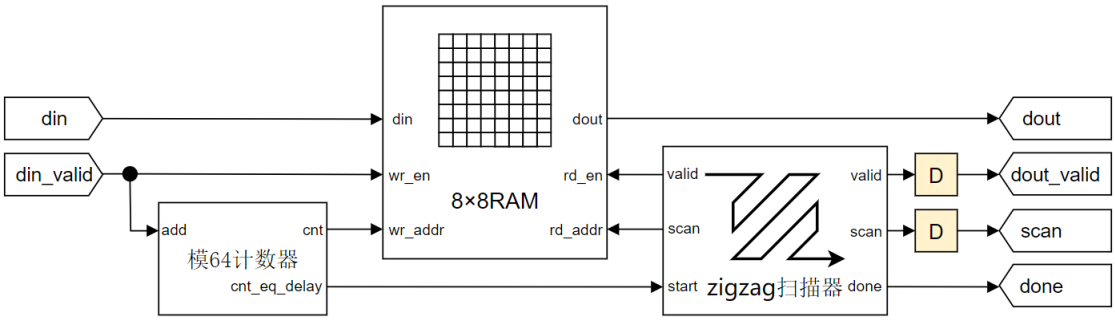

图 3-20 Zigzag 扫描模块内部划分

3.5.1 zigzao 扫描器

由图 3-21所示。该模块使用一个状态来控制两个代表坐标值的计数器当状态机 right、left 信号有效时，计数器分别加一和减一。down、up 信号同理。当 zero 信号有效时两个计数器清零。

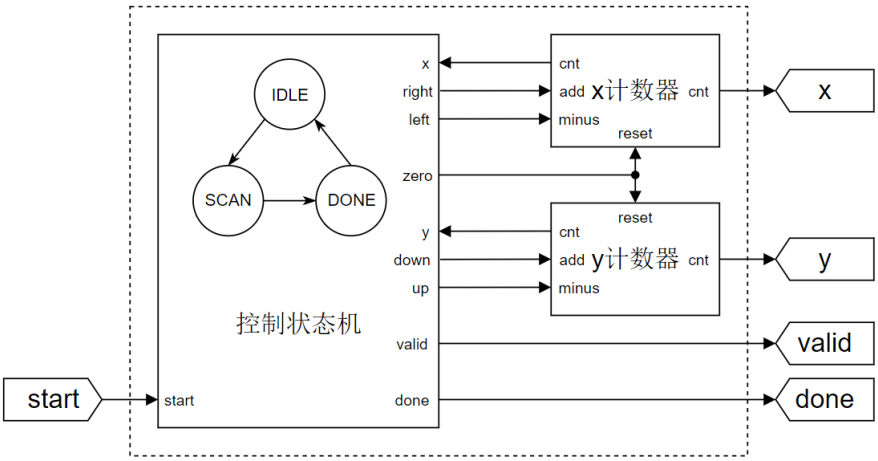

图 3-21 Zigzag 扫描器

表 3-13 Zigzag 模块输入输出信号

信号名称	方向	功能
start	输入	当高电平时启动 zigzag 扫描
x	输出	zigzag 扫描的水平坐标
y	输出	zigzag 扫描的垂直坐标
valid	输出	表示输出坐标值有效
done	输出	当高电平时表示 zigzag 扫描结束

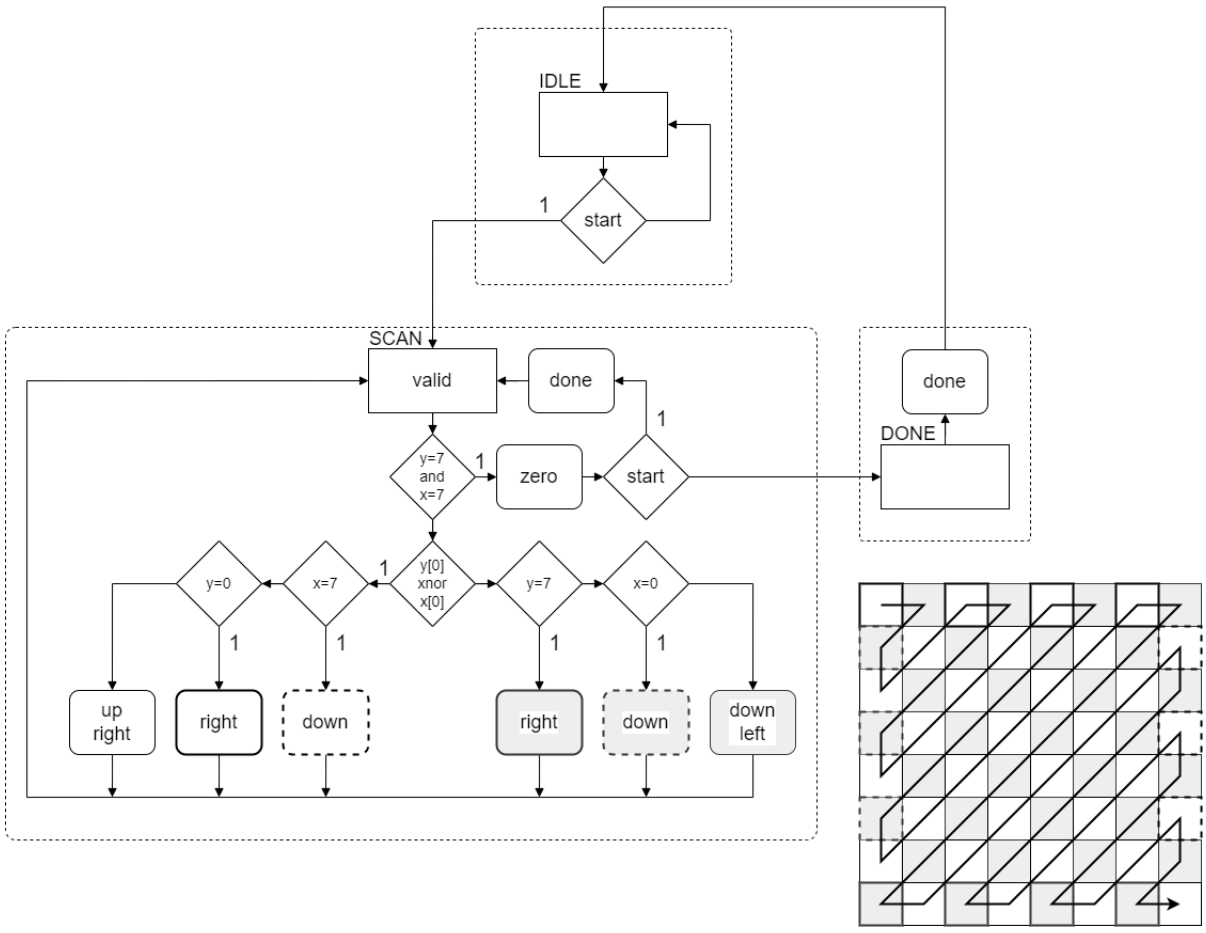

图 3-22 Zigzag ASM 图

该状态机的 ASM 图如图 3-22所示，在 ASM 图中每个虚线框代表一个状态每个状态中都带有一个矩形框，矩形框的左上角标注状态名称，内部存放穆尔型输出，同时表示该信号此时有效。与流程图类似，棱形框代表状态输入信号的判断。圆角矩形内部存放米利型输出。下面分别用三个状态描述 zigzag 扫描的运行步骤。

初态为 IDLE 状态，表示该模块空闲，当接收到 start 信号有效时，跳转到 SCAN 状态开始工作；SCAN 状态表示该模块正进行扫描，valid 信号有效。如图 3-22右下角所示, 在此状态下根据当前所在的坐标控制下个时钟的移动方向。移动的方向根据坐标分为 6 种（灰色矩形左下，灰色粗线矩形右，灰色虚线矩形下，白色矩形右上，白色粗线矩形右，白色虚线下矩形，）。当坐标值为 (7, 7) 时，输出 zero 信号有效对坐标值进行清零，再根据 start 判断是否进行下一次扫描，为 0 时跳转到 DONE 状态结束扫描，为 1 时输出 done 信号表示本次扫描结束；DONE 状态表示扫描结束输出 done 信号有效，并无条件跳转到 IDLE 状态。

该模块的 valid 输出最终接入 RAM 模块的读使能。如果在清理之后直接跳转到 DONE 状态，会导致 RAM 的读数据有 2 个时钟周期的中断。如果 RAM 一直保持写入，就会导致 RAM 的吞吐率不一致，进而导致数据丢失。所以增加清零后 SCAN 状态的跳转路径。zigzag 扫描器的仿真波形如图 3-23所示。

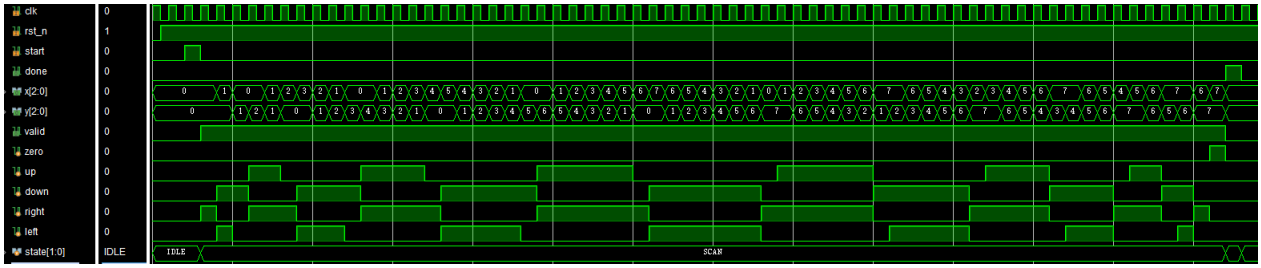

图 3-23 Zigzag 仿真波形图

3.6 量化模块

由于 DCT 模块的有些输出为组合逻辑输出，如第 64 个像素点。此时信号的建立时间会比较长，为了不继续增加组合逻辑关键路径，就先把数据存在 Zigzag 扫描模块中进行缓存，再通过量化模块进行量化。量化最直接实现就是使用除法器进行实现。由于除法器在 fpag 中资源比较稀缺，同时工作时钟频率不高。本设计不采用除法器进行实现。考虑到量化中的除数为常数，同时如果被除数是 2 的次幂，除法运算就相当于右移运算。如式（3.10）可以将除法分解为一次乘法再进行一次 2 次幂的除法（移位）, 当 n 越大时，精度就越大。这样就可以使用乘法器实现除数为常数的除法。本设计采用 2^{16} 作为其中的被除数。

$$\frac{F(u,v)}{Q(u,v)} = F(u,v) * \frac{\left(\frac{2^n}{Q(u,v)}\right)}{2^n} \tag{3.10}$$

量化器的 RTL 视图如图 3-24。将对应的乘数 $\frac{2^{16}}{Q(u,v)}$ 存进 ROM 中，把 Zigzag 模块的扫描输出作为 ROM 的读地址输入。进而获取对应的乘数，再将乘积进行右移截位得到最终的量化值。

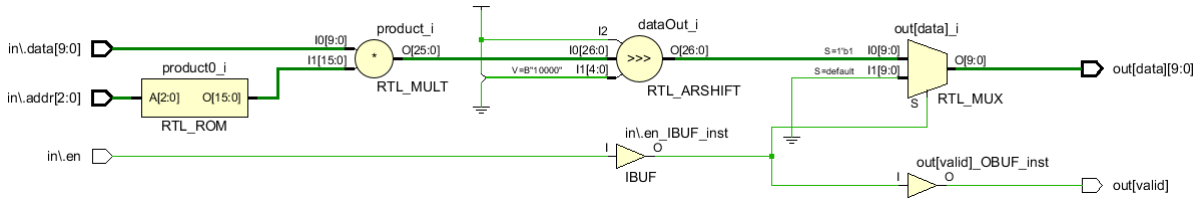

图 3-24 量化器结构图

量化模块的输出波形如图**??**所示，可以看出经过量化后出现大量连续的 0 值。

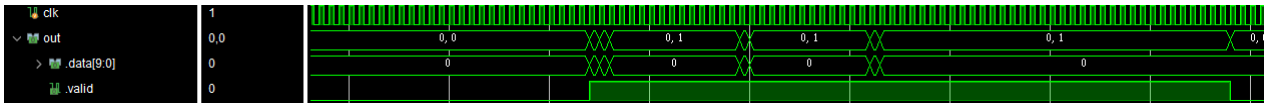

图 3-25 量化器输出波形

3.7 熵编码模块

3.8 JPEG 文件格式生成模块

4 JPEG 编解码系统 FPGA 实现以及验证

4.1 验证系统模块划分

参考文献

[1] 王成昊. 基于 FPGA 的低复杂度 JPEG 解码系统设计及实现 [D]. 四川师范大学,2024.DOI:10.27347/d.cnki.gssdu.2024.000798.

[2] 杜英杰. 基于 FPGA 的 JPEG 图像压缩的自适应量化编码 [D]. 西华大学,2023.DOI:10.27411/d.cnki.gscgc.2023.000416.

[3] 赵恒阳, 刘华.Baseline JPEG 压缩器在 Xilinx FPGA 上的设计与实现 [J]. 电子设计工程,2014,22(20):153-156.DOI:10.14022/j.cnki.dzsjgc.2014.20.044

[4] 季永国. 基于 FPGA 硬件 JPEG 压缩器的无线胶囊内窥镜系统 [D]. 上海交通大学,2015.

[5] 涂友钢. 单兵夜视眼镜视频信号处理技术研究 [D]. 南京理工大学,2019.DOI:10.27241/d.cnki.gnjgu.2019.001490.

[6] 董鑫怡. 不同应用场景下图像压缩方法的 FPGA 实现 [D]. 南京理工大学,2023.DOI:10.27241/d.cnki.gnjgu.2023.002037.

[7] 郝亚喆. 基于高性能 JPEG2000 编码器的图像处理系统设计与实现 [D]. 天津大学,2022.DOI:10.27356/d.cnki.gtjdu.2022.000903

[8] 任静. 图像压缩编解码的 FPGA 设计与实现 [D]. 南京林业大学,2020.DOI:10.27242/d.cnki.gnjlu.2020.000238.

[9] 刘多强. 高速图像压缩存储系统关键技术研究与实现 [D]. 西南科技大学,2020.DOI:10.27415/d.cnki.gxngc.2020.001074.

[10] Rodrigues T, Vestias M. Using dyanamic revonfiguration to reduce the area of a JPEG decoder on FPGA[C]/2015 Euromicro Conference on Digital System Design. IEEE, 2015: 65-71

[11] Shan J, Wang D, Yang E. High performance JPEG decoder based on FPGA[C]/2011 Asia Pacific conference on Postgraduate Research in Microelectronics & Electonice. IEEE, 2011:57-60