

H.264 视频解码 IP 核的设计与实现

作者：梁盼 陶宝泉 指导教师：付永庆

哈尔滨工程大学 信息与通信工程学院 150001

摘要：H.264 以其优异的性能在实时网络视频通信、数字广播电视及高清视频存储播放等方面获得广泛应用，因此研究 H.264 算法的硬件实现意义重大。本文设计了一种基于 FPGA 高效并行结构的 H.264 视频解码 IP 核，在设计中提出了优化遍历查表的 CAVLC 熵解码设计方案，并详细介绍了全流水线并行运算结构的反量化反 DCT 变换模块和帧内预测模块的硬件实现。整个设计通过 Altera 公司 Stratix II 系列的 EP2S60F672C5ES 平台验证，在最高时钟频率 82MHz 下能以 50frame/s 的速度解码分辨率为 320*240 的灰度图像，在速度，功耗，成本，可移植性等方面都具有独特的优势和良好的发展空间。

关键词：H.264, SOPC, 帧内预测, CAVLC, DCT

Design and Application of H.264 IP Core

Author: Liang Pan Tao Baoquan Teacher: Fu Yongqing

(College of Information and Communication Engineering, Harbin Engineering University, Harbin 150001, China)

Abstract: H.264 has a widespread application in real-time internet video communication, digital television broadcast, HDTV and other various aspects because of its excellent performance. So it's necessary to study on the hardware application of H.264. This paper describes the design of a highly parallel structure H.264 IP Core based on FPGA. At the design stage, we propose an optimized method of CAVLC table's looking up. The hardware application of entire pipeline parallel arithmetic inverse quantization, inverse DCT module and intra prediction module is also introduced in detail. The H.264 coding system is tested on the EP2S60F672C5ES, at the highest clock frequency of 82MHz, it can process 50 gray images with the resolution of 320*240 in a second, which has unique advantage and good space for development in speed, power consumption, cost, portability and so on.

Keyword: H.264, SOPC, intra prediction, CAVLC, DCT

1、引言

随着网络技术与多媒体技术的不断发展，为了能够在较低带宽上提供高质量的图像传输，H.264 视频编码技术脱颖而出，以其高压缩比，强大的抗误码效应和良好的网络亲和性得到各方面的青睐，其代表了现今通信行业融合的趋势，满足了包括广电、网络多媒体以及 3G 移动网络等多方面的需求，然而其高性能是以算法复杂度的增加为代价的。如此大的计算量用 DSP 等纯软件方案，因局限于 C 语言的串行执行思想实现比较吃力，且对于高清视频很难达到理想的效果；如果使用纯硬件的 ASIC 方案，则在应用的灵活性上受到限制；而拥有高处

理速度及高灵活性的 FPGA+SOPC 平台，则能很好的满足这一要求，基于此，本文提出了利用 Altera 公司提供的基于 Avalon 总线的 SOPC 结构实现 H.264 解码算法的思想。

2、H.264 解码算法简介

H.264 是由联合视频组 (JVT) 提出的最新数字视频编解码器标准。其解码过程如图 2.1 所示，视频编码比特流首先经过解析，分离成参数信息和图像残差数据，其中参数信息指明宏块类型和解码方案等，图像残差数据经熵解码，逆 Zig_Zag 重排序，反量化和反变换得到时域残差，与此同时，利用已经重建的邻近块作为参考对当前块进行帧内预测或帧间预测，预测的结果加上时域残差即得到当前块的重建值。其中熵解码包括基于上下文的自适应变长编码 CAVLC 和基于上下文的自适应算术二进制编码 CABAC，这二者应用于 H.264 的不同档次级别中，本文针对基本档次中的 CAVLC 展开讨论，另外考虑到帧间预测算法复杂且需要保存前后多帧数据用于参考，而开发板无法提供如此庞大的资源开销，所以本文的设计过程中暂没有考虑帧间预测的实现。

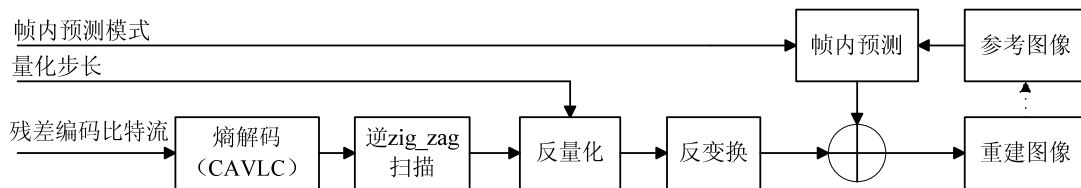


图 2.1 解码总体流程图

3、IP 核结构设计

3.1 预处理模块

1、编码比特流数据结构分析

一段完整的 H.264 视频编码比特流是由若干 NAL 单元组成的，如图 3.1 所示，其中第一个 NAL 单元是序列参数集 SPS，第二个 NAL 单元是图像参数集 PPS，这二者携带了该段视频的一些编码特征信息，第三个及以后的 NAL 单元则携带图像数据，一般一个 NAL 单元携带一幅图像数据，由若干片组成，每个片又是由若干大小为 16*16 的宏块组成，每一个宏块包括 4 个 8*8 的亮度块，一个 8*8 的 Cr 色度块和一个 8*8 的 Cb 色度块，其中每一块又分成 4 个 4*4 的子块。宏块层以上的句法元素解析主要是一些读码流、判断、存储参数的操作，基本没有算术运算，用 CPU 软件处理会更加方便，因此用 NIOSII 来实现，解析所得到的参数信息和剩余的图像残差数据都存入 SRAM 中供后续解码

所用，而解码 IP 核则按照光栅扫描顺序依次解码各宏块。在解码宏块的过程中首先解析宏块头信息，然后利用宏块头提供的特征信息依次解码各块中的子块，即对这些 4*4 的子块进行熵解码，反量化反变换，帧内预测等处理得到重建图像。

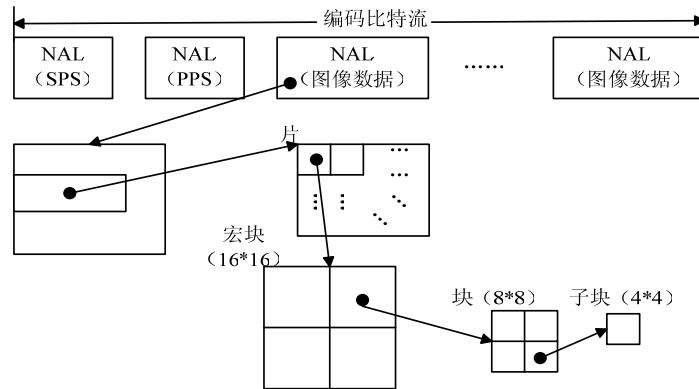


图 3.1 编码比特流数据格式

2、宏块头解析

宏块头解析模块就是一个状态机，依次解码各句法元素，主要包括指明宏块类型的句法元素 `mb_type`，决定量化步长的句法元素 `mb_qp_delta` 和指明子块帧内预测模式的两个句法元素 `prev_intra4*4_pred_mode_flag` 和 `rem_intra4*4_pred_mode` 等，这些句法元素都采用哥伦布编码，哥伦布解码的原理已相当成熟，可参见文献^[5]，这里不再详述。

3.2、帧内预测模块

3.2.1 帧内预测原理与算法分析

视频压缩中的帧内预测是通过当前编码的上方块和左方块来计算当前块的预测值，即利用相邻块之间的相似性消除空间冗余。H.264 采用了比其他标准更为精确和复杂的帧内预测方式，包括有：4*4 亮度块的帧内预测，16*16 亮度块的帧内预测和 8*8 色度块的帧内预测，本文只讨论 4*4 亮度块的帧内预测，4*4 亮度子块有 9 种可选预测模式，这 9 种预测模式的预测方向如图 3.2 (a) 所示，图 3.2 (b) 给出了参考像素和待预测像素的标号，其中 4*4 亮度块的上方和左方像素 A~M 为已编码并重构的像素，用作编解码器中的预测参考像素，a~p 为待预测像素。

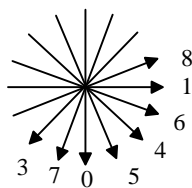


图 3.2(a) 4*4 亮度块帧内预测方向图
表 3.1 给出了 9 种模式的数学描述。

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

图 3.2(b) 参考像素和待预测像素的标号

表 3.1 帧内预测模式的数学描述

	垂直	水平	均值	下左对角	下右对角	垂直向右	水平向下	垂直向左	水平向上
a	A	I	Mean	$(A+2B+C+2) \gg 2$	$(A+2M+I+2) \gg 2$	$(M+A+1) \gg 1$	$(M+I+1) \gg 1$	$(A+B+1) \gg 1$	$(I+J+1) \gg 2$
b	B	I	Mean	$(B+2C+D+2) \gg 2$	$(M+2A+B+2) \gg 2$	$(A+B+1) \gg 1$	$(I+2M+A+2) \gg 2$	$(C+B+1) \gg 1$	$(I+2J+K+2) \gg 2$
c	C	I	Mean	$(C+2D+E+2) \gg 2$	$(A+2B+C+2) \gg 2$	$(B+C+1) \gg 1$	$(M+2A+B+2) \gg 2$	$(C+D+1) \gg 1$	$(J+K+1) \gg 1$
d	D	I	Mean	$(D+2E+F+2) \gg 2$	$(B+2C+D+2) \gg 2$	$(C+D+1) \gg 1$	$(A+2B+C+2) \gg 2$	$(E+D+1) \gg 1$	$(J+2K+L+2) \gg 2$
e	A	J	Mean	$(B+2C+D+2) \gg 2$	$(M+2I+J+2) \gg 2$	$(I+2M+A+2) \gg 2$	$(I+J+1) \gg 1$	$(A+2B+C+2) \gg 2$	$(J+K+1) \gg 1$
f	B	J	Mean	$(C+2D+E+2) \gg 2$	$(A+2M+I+2) \gg 2$	$(M+2A+B+2) \gg 2$	$(M+2I+J+2) \gg 2$	$(B+2C+D+2) \gg 2$	$(J+2K+L+2) \gg 2$
g	C	J	Mean	$(D+2E+F+2) \gg 2$	$(M+2A+B+2) \gg 2$	$(A+2B+C+2) \gg 2$	$(M+I+1) \gg 1$	$(C+2D+E+2) \gg 2$	$(K+L+1) \gg 1$
h	D	J	Mean	$(E+2F+G+2) \gg 2$	$(A+2B+C+2) \gg 2$	$(B+2C+D+2) \gg 2$	$(I+2M+A+2) \gg 2$	$(D+2E+F+2) \gg 2$	$(K+3L+2) \gg 2$
i	A	K	Mean	$(C+2D+E+2) \gg 2$	$(I+2J+K+2) \gg 2$	$(M+2I+J+2) \gg 2$	$(K+J+1) \gg 1$	$(C+B+1) \gg 1$	$(K+L+1) \gg 1$
j	B	K	Mean	$(D+2E+F+2) \gg 2$	$(M+2I+J+2) \gg 2$	$(M+A+1) \gg 1$	$(I+2J+K+2) \gg 2$	$(C+D+1) \gg 1$	$(K+3L+2) \gg 2$
k	C	K	Mean	$(E+2F+G+2) \gg 2$	$(A+2M+I+2) \gg 2$	$(A+B+1) \gg 1$	$(I+J+1) \gg 1$	$(E+D+1) \gg 1$	L
l	D	K	Mean	$(F+2G+H+2) \gg 2$	$(M+2A+B+2) \gg 2$	$(B+C+1) \gg 1$	$(M+2I+J+2) \gg 2$	$(E+F+1) \gg 1$	L
m	A	L	Mean	$(D+2E+F+2) \gg 2$	$(J+2K+L+2) \gg 2$	$(I+2J+K+2) \gg 2$	$(K+L+1) \gg 1$	$(B+2C+D+2) \gg 2$	L
n	B	L	Mean	$(E+2F+G+2) \gg 2$	$(I+2J+K+2) \gg 2$	$(I+2M+A+2) \gg 2$	$(J+2K+L+2) \gg 2$	$(C+2D+E+2) \gg 2$	L
o	C	L	Mean	$(F+2G+H+2) \gg 2$	$(M+2I+J+2) \gg 2$	$(M+2A+B+2) \gg 2$	$(K+J+1) \gg 1$	$(D+2E+F+2) \gg 2$	L
p	D	L	Mean	$(G+3H+2) \gg 2$	$(A+2M+I+2) \gg 2$	$(A+2B+C+2) \gg 2$	$(I+2J+K+2) \gg 2$	$(E+2F+G+2) \gg 2$	L

注：表中 Mean= $(A+B+C+D+E+F+G+H+4) \gg 3$

表 3.1 是在 A~M 都存在的情况下成立的，对于邻近块不可获得的情况需特殊考虑，如对于垂直预测，如果上方子块不存在，则 a=-1；对于水平预测，如果左方子块不存在，则 a=-1；对于 DC 预测，如果上方子块不存在，则 Mean= $(I+J+K+L+2) \gg 2$ ；如果左方子块不存在，则 Mean= $(A+B+C+D+2) \gg 2$ ；如果上方和左方都不存在，则 Mean=128 等。

3.2.2 帧内预测控制模块

在进行预测之前，需要检查各个参考点是否存在，以决定预测公式，另外还要从相应内存空间中获取参考像素 A~M，控制模块主要就是完成这两大功能。

1、邻近子块可用性判断

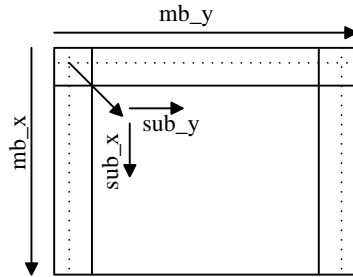


图 3.3 邻近子块可用性示意图

对一帧 320*240 的图像而言，大部分子块的相邻子块都是可获得的，但有一些特殊位置子块的相邻块的可获得性比较特殊，需要单独讨论，如图 3.3 所示，主要包括以下四种情况，其中 mb_x 和 mb_y 表示当前宏块在整帧图像中的纵横坐标， sub_x ， sub_y 表示当前子块在当前宏块中的纵横坐标。

- 1) 第一行的每个宏块中的第一行子块（即 $mb_y=0$ and $sub_x=0\sim3$ and $sub_y=0$ ）的 $up_available$ ， $up_left_available$ 和 $up_right_available$ 都不存在；
- 2) 第一列的每个宏块中的第一列子块（即 $mb_x=0$ and $sub_y=0\sim3$ and $sub_x=0$ ）的 $left_available$ ， $up_left_available$ 不存在；
- 3) 最后一列的每个宏块中的最后一列子块（即 $mb_x=19$ and $sub_y=0\sim3$ and $sub_x=3$ ）的 $up_right_available$ 不存在。

图 3.4 给出了前三个宏块中各子块的邻近块的可用性判断的仿真结果。

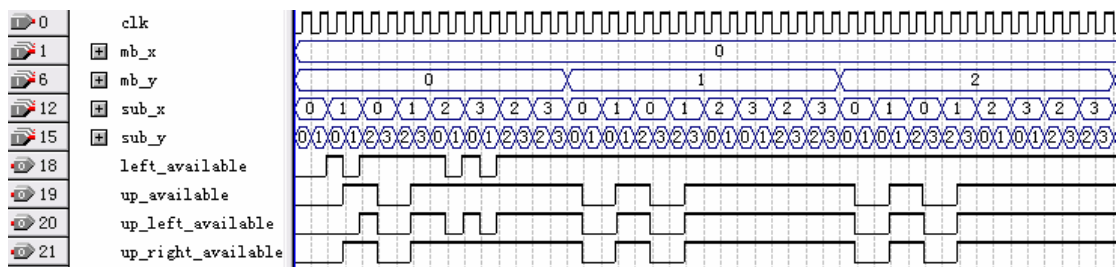


图 3.4 邻近块可获得性仿真波形图

2、参考像素 A~M 的获取

待解码的编码比特流和解码后重建的图像分别存储在 NIOSII 开发板上的 SRAM 和 SDRAM 中。在进行帧内预测之前，应该根据当前处理的子块位置从重建图像内存空间读取预测参考值 A~M，设计的关键就在于计算地址偏移，重建图像是按光栅扫描的宏块顺序依次存储在 SRAM 中的，内存宽度为 32 位，即一个内存区间能存储 4 个像素的亮度值，计算地址偏移的公式如下：

1) 求左边的 I, J, K, L

如果 $\text{left_available}=0$, $E=F=G=H=128$;

否则, $\text{address}(I)=16*\text{sub_x}+4*\text{sub_y}+64*\text{mb_Num}-4$,

$\text{address}(J)=\text{address}(I)+1$; $\text{address}(K)=\text{address}(J)+1$; $\text{address}(L)=\text{address}(L)+1$; I, J, K, L 分别取这 4 个地址所对应的内存区间中的 32 位数据的低 8 位。

2) 求上边的 A,B,C,D

如果 $\text{up_available}=0$, $A=B=C=D=128$;

否则 $\text{address}=16*\text{sub_x}-1+4*\text{sub_y}+1+64*\text{mb_Num}+3$; A, B, C, D 为该地址所对应的内存区间中的 32 位数据。

3) 求 M

如果 $\text{up_left_available}=0$, 则 $M=128$;

否则 $\text{address}=16*(\text{sub_x}-1)+4*(\text{sub_y}-1)+1+64*\text{mb_Num}+3$, M 取该地址所对应的内存区间中的 32 位数据的低 8 位。

4) 求右上的 E,F,G,H:

如果 $\text{up_right_available}=0$, 则 $E=F=G=H=D$;

否则, $\text{address}=16*(\text{sub_x}-1)+4*(\text{sub_y}-1)+64*\text{mb_Num}+3$, E, F, G, H 为该地址所对应的内存区间中的 32 位数据。

以上各式中的 address 是对应访问 SRAM 的地址偏移, mb_num 是当前处理宏块在图像中的位置, $\text{mb_num}=\text{mb_x}*\text{PicWidthinMB}+\text{mb_y}$, 其中 PicWidthinMB 表示图像的宏块宽度。

3.2.3 帧内预测算法设计

除了表 3.1 中所描述的 $4*4$ 亮度块的 9 种预测模式以外, H.264 的帧内预测还包括 $16*16$ 亮度块和 $8*8$ 色度块各 4 种预测模式, 为了实现实时解码, 最直观简单的方法就是对应这 17 种预测模式设计 17 个预测器, 但很显然这样设计增加了系统复杂度和对硬件资源的占用, 经过对表 3.1 所有预测模式的仔细分析和研究可以看出, 虽然不同预测模式对应着不同的预测值求解方式, 但是涉及到的运算框架都是基本一样的, 即先进行求和然后移位的运算模式, 不同的只是具体模式下求和的参数和移位的位数不同, 因此可以一个通用的运算模型来完成所有运算, 这样既使设计变得简单, 又节省了硬件资源, 由此设计的运算模型如图 3.4 所示:

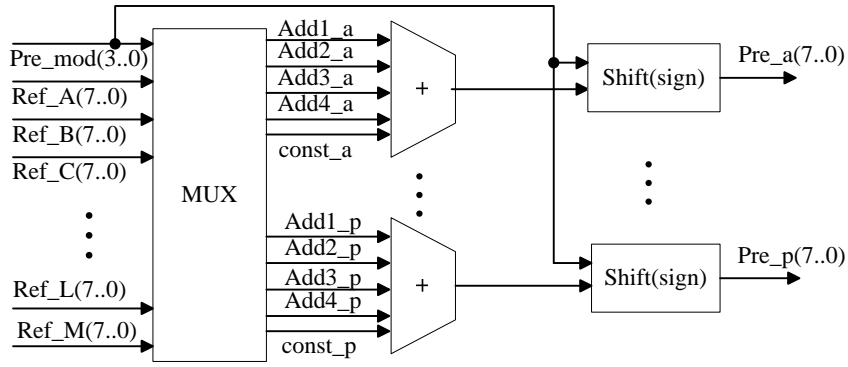


图 3.4 帧内预测运算模型

图 3.4 中，Pre_mod 是从宏块头中获取的当前子块的预测模式，Ref_A 到 Ref_M 是从 SRAM 相应的内存区间获得的相邻块的重建像素值作为预测参考像素，结合表 3.1，设计了一个包含 5 个加数的加法器，其中 Add1_x~Add4_x 是 4 个变加数，const_x 是常量加数，若对应的求和公式中不足 5 个加数，则补零处理，Shift 模块是有符号移位截断模块，移位的位数由 Pre_mod 和像素位置确定。针对一幅实际图像的一个子块的数据，图 3.5 给出了 4*4 亮度块 9 种预测模式的预测结果，图中红色矩形框中即是运算结果，由图可以看出，在给出预测模式和参考像素值以后，3 个时钟周期后就可以输出稳定的预测结果。

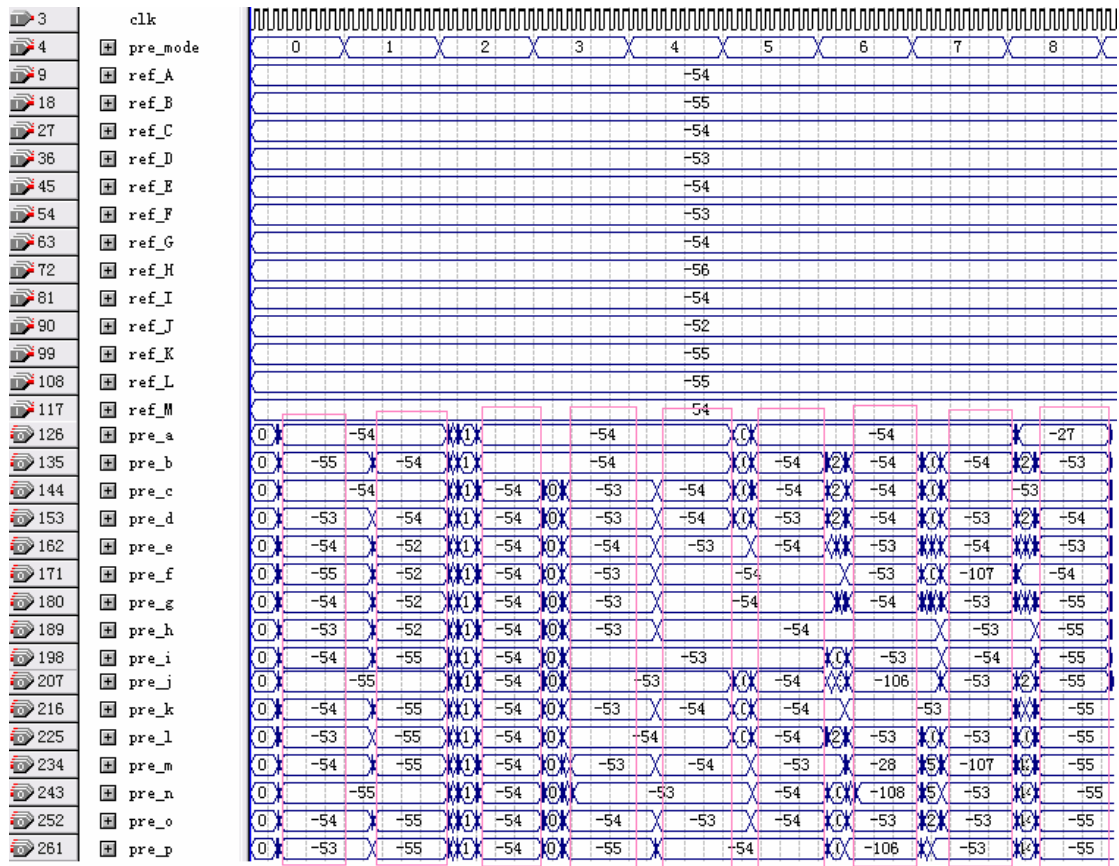


图 3.5 帧内预测波形仿真图

3.3 CAVLC 熵解码

3.3.1 CAVLC 熵解码流程

子块残差数据经 CAVLC 熵编码后的码流包括 5 个句法元素: `coeff_token`, `trailing_ones_sign_flag`, `level`, `TotalZeros` 和 `RunBefore`。CAVLC 解码就是依次解码这 5 个句法元素, 下面分别详细讲述这 5 步的解码原理及过程, 为了更直观的展示解码流程, 以一个 4*4 数据块为例给出仿真波形: { 0, 3, -1, 0; 0, -1, 1, 0; 1, 0, 0, 0; 0, 0, 0, 0 } , 5 个句法元素的编码比特流依次为 0000100, 011, 1, 0010, 111。

1、解码非零系数数目 TotalCoeff 和拖尾系数的数目 TrailingOnes

解码二者需根据变量 `nc` 的值和编码比特流查询 `coeff_token` 码表(关于 `nc` 的具体含义和求取过程将在后面将进一步详细介绍, 此处假设 `nc=1`), 图 3.6 给出了查表的仿真波形图, 由图可以看出, 编码比特流 0000100 对应的 `TotalCoeff=5`, `TrailingOnes=3`。

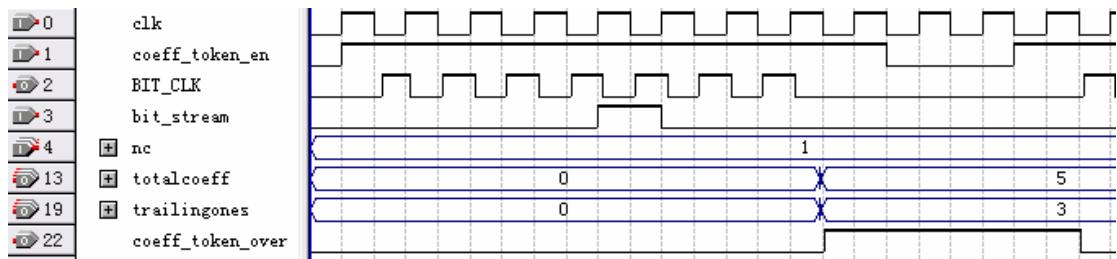


图 3.6 `coeff_token` 解码仿真波形图

2、解码拖尾系数的符号

由第一步解码得到 `TrailingOnes=3`, 即数据块中至少存在 3 个 ± 1 , 紧接着就按照反向扫描的顺序对每个拖尾系数的符号进行解码, 其中 0 表示+, 1 表示-, 因此编码比特流 011 解码出三个拖尾系数依次是+1, -1, -1;

3、解码除了拖尾系数之外的非零系数的幅值

`level` 包括前缀 `level_prefix` 和后缀 `level_suffix`, 其解码的简明步骤为:

1) 设定循环次数 $i = \text{TotalCoeff} - 1 - \text{TrailingOnes}$, 确定后缀长度 `suffixlength` 的初始值, 若 $\text{TotalCoeff} > 10 \ \&\& \ \text{TrailingOnes} < 3$ 的话, $\text{suffixlength} = 1$, 否则 $\text{suffixlength} = 0$;

2) 根据编码码流查码表得到 `level_prefix`;

3) 接着从码流中读取 `suffixlength` 位的数据作为后缀 `level_suffix`, 而 $\text{LevelCode} = (\text{level_prefix} \ll \text{suffixlength}) + \text{level_suffix}$;

4) 若 `Levelcode` 为偶数, 则 $\text{level} = (\text{level} + 2) / 2$, 若 `Levelcode` 为奇数, 则 $\text{level} = (-\text{level} - 1) / 2$;

5) 如果 $level[i]$ 的绝对值大于设定的阈值, 则 $Suffixlength=Suffixlength+1$, 循环次数 i 减 1, 回到步骤 2 继续执行, 直到 $i=0$, 除拖尾系数外的非零系数解析完毕。

对于本数据块, i 的初值为 1, $Suffixlength=0$, 查表 '1' 对应的前缀 $levelprefix=0$, 所以 $levelcode=0$, 计算得到 $level=1$, $i--$; $i=0$, $suffixlength=1$, 查表 "0010" 对应的前缀 $levelprefix=2$, 计算得到 $levelcode=4$, $level=3$, 所以两个非零系数分别为 1 和 3, 图 3.7 给出了仿真波形。

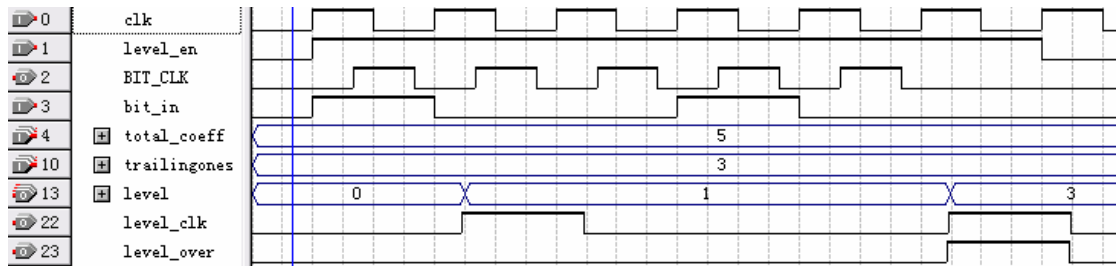


图 3.7 level 解码仿真波形图

4、解码最后一个非零系数前零的总数 TotalZeros

根据 $TotalCoeffs=5$ 和输入码流 111 查文献^[5]得到 $TotalZeros=3$, 图 3.8 给出了仿真波形。

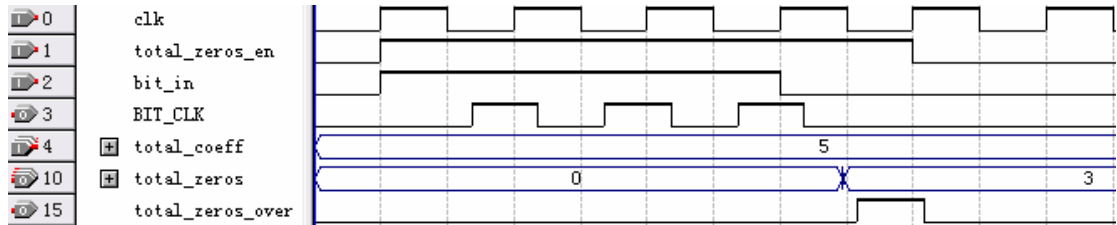


图 3.8 total_zeros 仿真波形图

5、解码每个非零系数前零的个数 run_before

按照反向解码的顺序查文献^[5], 表格的入口参数为 $ZerosLeft$ (表示当前非零系数左边的所有零的总数目), $ZerosLeft$ 的初始值等于 $TotalZeros$, 每解完一个 $RunBefore$ 后, $ZerosLeft$ 即时更新, 直到 $ZerosLeft$ 等于 0 或者已经解码到最后一个非零系数时结束。对于本数据块, 初始 $i=TotalCoeffs-1=4$, $zeroleft=TotalZeros=3$, 5 个非零系数前零的数目解析过程如下:

- 1) $i=4$, $zeroleft=3$, 根据码流 "10" 查表得到 $runbefore[4]=1$;
- 2) $i=i-1=3$, $zeroleft=zeroleft-runbefore[4]=2$, 根据码流 "1" 查表 $runbefore[3]=0$;
- 3) $i=i-1=2$, $zeroleft=zeroleft-runbefore[3]=2$, 根据码流 "1" 查表 $runbefore[2]=0$;
- 4) $i=i-1=1$, $zeroleft=zeroleft-runbefore[2]=2$, 根据码流 "01" 查表 $runbefore[1]=1$;

5) $i=i-1=0$, $zeroleft= zeroleft-runbefore[1]=1$ 。

图 3.9 给出了仿真波形图

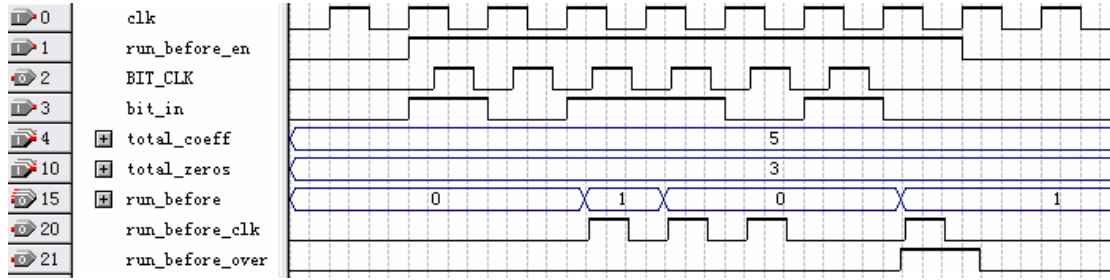


图 3.9 run_before 仿真波形图

波形仿真图 3.6 到 3.9 中都分别有一个 en 信号和一个 over 信号，其中 en 信号控制相应模块启动，over 信号标明本模块运算完毕，为了保证第一步到第五步的顺序执行，用上一模块的 over 信号控制启动下一模块的使能 en，BIT_CLK 信号是读取内存的时钟，其仅在 en 使能时才存在，这样才能保证编码比特流不会串位，另外由于一个数据块中可能有多个 run_before 和 level，又都是变长变码，每个查表的周期不同，所以另设 run_before_clk 和 level_clk 两个信号标志每个 run_before 和 level 句法元素是否解码完毕。

将第一步到第五步解码的结果送入最后的重排序模块，其中有 5 个非零系数，其中包括 3, 1 和 3 个拖尾系数 -1, -1, 1，这 5 个非零系数前零的数目分别是 1, 0, 0, 1, 1，重排序后得出解码结果为 0, 3, -1, 0; 0, -1, 1, 0; 1, 0, 0, 0; 0, 0, 0, 0; 与原数据一致。

3.3.2 NC 值的求取

3.3.1 节中提到在解码非零系数数目和拖尾系数数目时要用到变量 nc，在邻近子块存在的情况下， $nc=(NA+NB)/2$ ，其中 NA, NB 分别是当前块左边/上边 4*4 块的非零系数数目，其电路结构图如图 3.10 所示，图中设计两个存储器存储 NA 和 NB，其中 lpm_ram_dq0 深度为 4，存储当前块左边相邻 4 个子块的非零系数数目，lpm_ram_dq1 存储当前块上面一行子块的非零系数数目，control_nc 模块从这两个内存中获得 NA 和 NB 后，计算得到 nc，送入 search_blank 模块，同时启动该模块，search_blank 根据码流和 nc 进行查表得到 trailingones 和 totalcoeff，totalcoeff 更新到 lpm_ram_dp0 和 lpm_ram_dp1 的相应位置中。

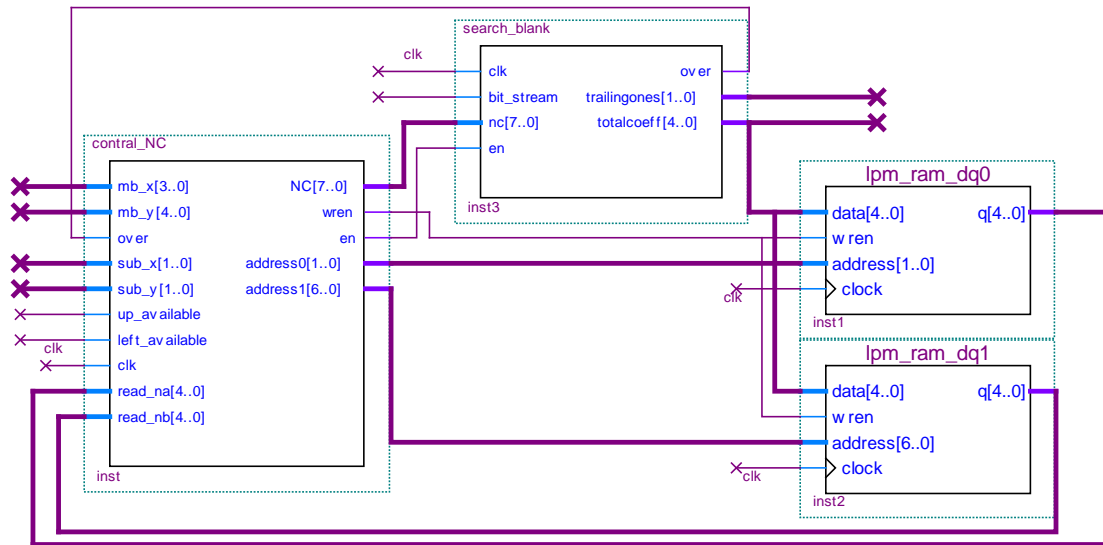


图 3.10 求取 NC 的硬件电路图

3.3.3 查表算法的优化

由 3.3.1 节中所讲述的 CAVLC 的解码原理可以看出，几乎在解码每个句法元素的过程中，都需要查询相关码表，且所要查询的码表都是变长码表，而硬件设计中每个周期只能从码流中读入 1bit 的数据，若每读进 1bit，就进行遍历查表的话，对速度和资源而言都是巨大的开销，例如 coeff_token 码表的每个码表最多就有 62 种取值，码长为 0-16 位不等，因此需要寻求更为高效的查表方法。仔细分析各码表发现所有句法元素的码字都是一系列 0 和 1 的组合，并且绝大部分的码字都是以若干 0 开始，然后是 1 引领的若干后缀，这种组合不难让我们联想到哥伦布码流结构，表 3.2 给出了 coeff_token 码表中 nc 取值范围在 0~2 之间的一系列码表的统计分析情况。

表 3.2 coeff_token 码表统计情况

前缀零的个数	后缀的情况 (0 < nc < 2)
0, 1, 2, 14	无后缀
3, 4	01, 00, 1
5, 6, 7, 8, 13	00, 01, 10, 11
9, 10, 11, 12	000, 001, 010, 011, 100, 101, 110, 111

由表 3.2 可以看出在获取前缀 0 的个数后，后缀最多只有 8 种情况，这就大大缩小了遍历的范围。具体优化的查表过程如图 3.11 所示，其它表格的查询方法依此类推。

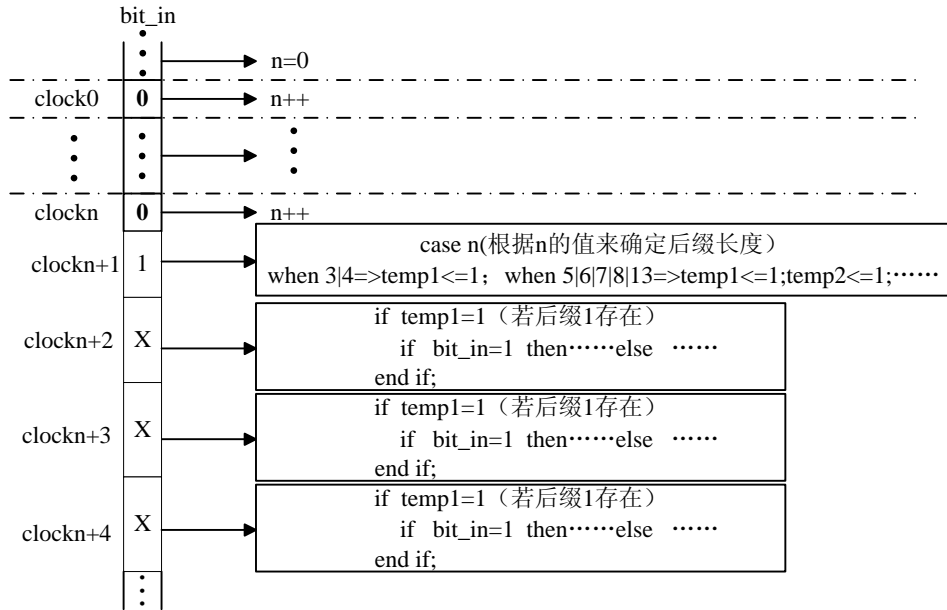


图 3.11 查表优化算法

3.4 反量化反变换模块设计

H.264 采用的是 4×4 块的整数 DCT 变换^[3], 整个变换过程仅用整数加减和移位操作就可以完成。这样既降低了设计复杂度, 又避免了编解码误匹配等问题, 还能够得到与离散 DCT 变换类似的编码效果, 由此带来的编码性能的减少微乎其微。按照整数 DCT 变换的原理得到逆 DCT 变换公式为:

$$\mathbf{X} = \mathbf{C}_i^T (\mathbf{W} \otimes \mathbf{E}_i) \mathbf{C}_i$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \left\{ \mathbf{W} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right\} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

式中, $a=1/2$, $b=\sqrt{2/5}$, \mathbf{W} 是逆量化后的结果, H.264 将“ $\mathbf{W} \otimes \mathbf{E}_i$ ”结合到逆量化过程中去, 得到逆量化公式为 $\mathbf{W}_{r(i,j)} = (\mathbf{W}_{Q(i,j)} \mathbf{V}_{ij}) \ll (QP/6)$, 式中 \mathbf{W}_Q 是待处理的残差矩阵, QP 是量化参数, 从宏块头中获得, 取值范围为 0~51, 决定每个宏块的量化步长, \mathbf{V}_{ij} 是融合在反量化过程中的逆 DCT 中的比例变换系数, 它的值根据 QP 查表获得, 反量化硬件设计结构如图 3.12 所示, 由图 3.12 可以看出, 数据按列依次输入, 根据 QP 的值计算地址偏移, 查询相应的比例系数相乘然后移位, 一个周期完成一列 (4 个数据) 的反量化, 循环 4 次, 完成一个子块的逆量化, 逆量化的结果作为逆 DCT 变换的输入。

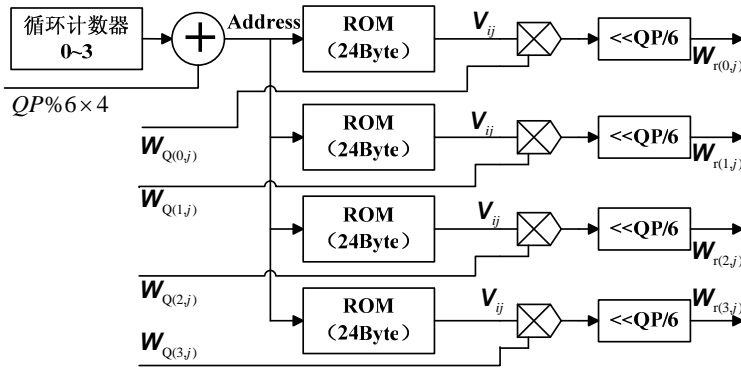


图 3.12 反量化硬件设计结构框图

图 3.13 可以完成逆 DCT 变换和逆 Hadamard 变换（用于对直流系数的特殊处理），由 IS_Hadamard 引脚进行切换，输入的一列数据先进行一维逆 DCT 变换，需一个时钟周期输出一列变换结果 row0—row3，此列变换的结果暂存起来，四个时钟周期后完成一个 4*4 矩阵的列变换，此时再依次按行输出一行值 col0~col3 进行行变换，同样需要 4 个周期完成行变换，其中 ROW_TO_COL 模块是行列转换模块，采用了“乒乓缓存”机制，这样仅在处理第一个子块时有几个周期的延时，后面每一个周期都能输出一行变换结果，变换的最终结果与帧内预测的结果相加得到最终重建图像值，存入 SRAM 的相应内存区间中，反量化和反变换的仿真结果如图 3.14 所示。

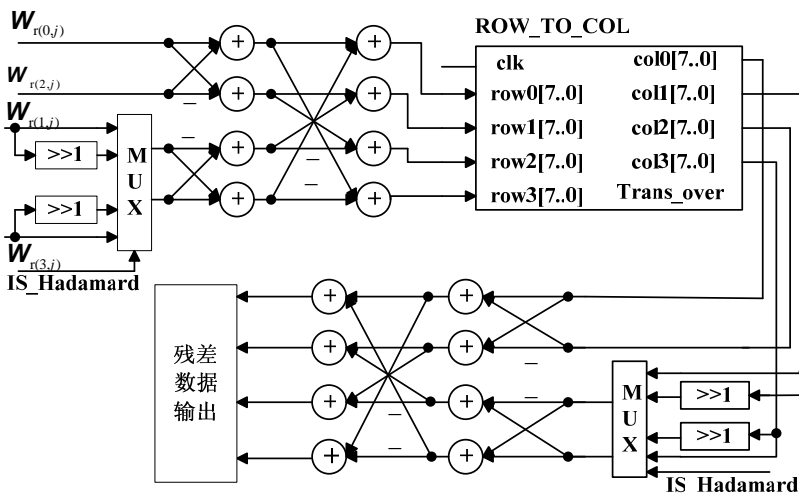


图 3.13 反 DCT 变换硬件设计结构框图

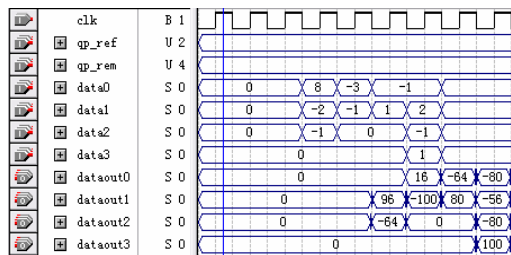


图 3.14 (a) 反量化仿真波形图

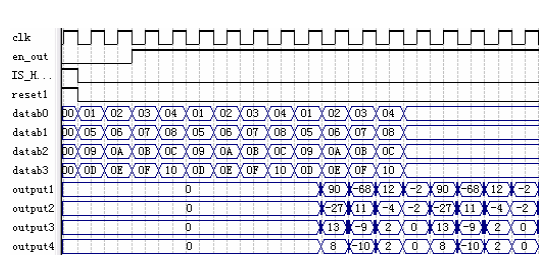


图 3.14 (b) 反 DCT 变换仿真波形图

4、基于 SOPC 的实际验证系统

H.264 解码器验证系统的构建基于 NIOS II 开发板，其核心芯片是 StratixII 系列的 FPGA EP2S60C5ES，结构如图 4.1 所示，H.264_decode IP Core 输出端是一个 Avalon 总线上的具有流控制属性从端口。乒乓切换模块用于实现 Avalon 总线与 H.264_decode IP Core 共享 SRAM。首先由网络接收经过 PC 机编码好的 H.264 编码图象数据，在 NIOSII 的控制下进行帧头解析，并将解析后的参数信息和图像残差数据存放在开发板上的 SRAM 内。这时通过乒乓切换将 SRAM 使用权分配给 H.264_decode IP Core 进行解码，解码完成后通过 DMA 将解码图像数据读出，存储在开发板上的 SDRAM 中。整个过程无需 NIOSII 处理器干预，只须等 DMA 控制器写满 SDRAM 后以中断的方式通知 NIOSII 处理器以使其挂起 H.264_decode IP Core 核，防止存入 SDRAM 中数据被覆盖，最后在 NIOSII 的控制下将 SDRAM 中的数据通过网口上传到 PC 机中进行后续处理。

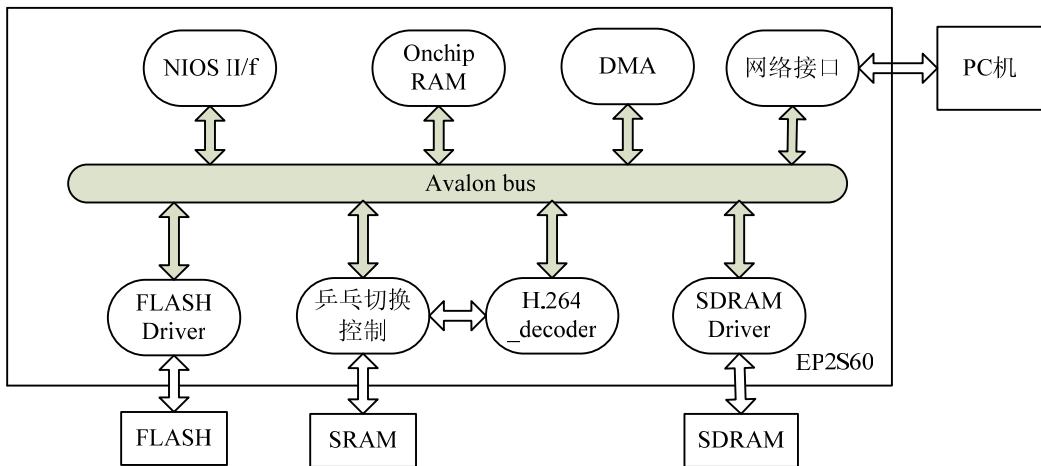


图 4.1 H.264 解码器验证系统结构

Use	Connections	Module Name	Description	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		<input type="checkbox"/> cpu	Nios II Processor				
		instruction_master	Avalon Master	sys_clk			
		data_master	Avalon Master				
		jtag_debug_module	Avalon Slave				
<input checked="" type="checkbox"/>		<input type="checkbox"/> ext_ram_bus	Avalon-MM Tristate Ebridge	sys_clk			
<input checked="" type="checkbox"/>		<input type="checkbox"/> ext_flash	Flash Memory (CFI)	sys_clk	0x02000000	0x02ffffff	
<input checked="" type="checkbox"/>		<input type="checkbox"/> ext_ram	IDT71V416 SRAM	sys_clk	0x04100000	0x041ffffff	
<input checked="" type="checkbox"/>		<input type="checkbox"/> lan91c111	LAN91C111 Interface				
		s1	Avalon Tristate Slave	sys_clk	0x04210000	0x0421ffff	0
<input checked="" type="checkbox"/>		<input type="checkbox"/> sys_clk_timer	Interval Timer	sys_clk	0x0422d060	0x0422d07f	1
<input checked="" type="checkbox"/>		<input type="checkbox"/> sysid	System ID Peripheral	sys_clk	0x0422d130	0x0422d137	
<input checked="" type="checkbox"/>		<input type="checkbox"/> sdram	SDRAM Controller	sys_clk	0x03000000	0x03ffffff	
<input checked="" type="checkbox"/>		<input type="checkbox"/> onchipRAM	On-Chip Memory (RAM or ROM)	sys_clk	0x04229000	0x04229fff	
<input checked="" type="checkbox"/>		<input type="checkbox"/> H_264_DECODER_0	H.264_DECODER	sys_clk	0x0422c800	0x0422cfff	
		avalon_slave_0	Avalon Slave				
<input checked="" type="checkbox"/>		<input type="checkbox"/> dma	DMA Controller				
		control_port_slave	Avalon Slave	sys_clk	0x0422d020	0x0422d03f	2
		read_master	Avalon Master				
		write_master	Avalon Master				
<input checked="" type="checkbox"/>		<input type="checkbox"/> pll	PLL	clk	0x0422d000	0x0422d01f	

图 4.2 H.264_decode IP Core 在 SOPC Builder 中的定制图

在 Quartus 7.2 环境下，分别对帧内预测模块、CAVLC 熵解码模块、反量化和反变换模块和整个解码器进行了综合，结果如表 4.1 所示。

表 4.1 各模块资源利用情况

	ALUTS	Registers	Memory bits
帧内预测	1870	1069	0
CAVLC 熵解码	1497	276	420
反量化反变换模块	761	465	640

对 H.264 Decoder IP 核进行静态时序分析，得到最高运行频率为 82MHz。

Timing Analyzer Summary				
	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	14.258 ns
2	Worst-case tco	N/A	None	9.982 ns
3	Worst-case tpd	N/A	None	12.968 ns
4	Worst-case th	N/A	None	-3.388 ns
5	Clock Setup: 'clk'	N/A	None	82.05 MHz (period = 12.188 ns)

图 4.3 H.264 Decoder IP 核静态时序分析结果

FPGA 解码模块在 50MHz 的时钟下，其解码运行时间与 PC 机对比如表 4.2 所示：

表 4.2 各模块运行时间

	帧内预测	CAVLC 熵解码	反量化反变换
PC 机解码	6.3ms	27ms	16ms
硬件解码	0.35ms	15ms	1.5ms

图 4.4 给出了一幅大小为 320*240 的灰度图像原始图像和先由 H.264 编码，后用 H.264 decoder IP 核解码后的图像。



图 4.4 (a) 原始图像



图 4.4 (b) 解码后的图像

图 4.5 给出了系统实物图。



图 4.5 系统实物图

5、结束语

本文详细介绍了基于 SOPC 的 H.264 解码 IP 核的设计和实现。重点讨论了帧内预测、CAVLC 熵解码和反量化反变换模块的设计，其中帧内预测结合 4*4 亮度块的 9 种预测模式的运算特点，用一个统一模型完成所有模式的计算，保证运算速度的同时节省了硬件资源；而针对 CAVLC 熵解码中频繁用到的遍历查表运算，根据码表特性提出了前缀和后缀分开查找的优化方案，大大降低了运算量和查表时间；在二维反 DCT 变换中引入“乒乓”缓存机制，使得一维行变换和二维列变换之间的延迟达到最小，提高了运算效率。

参考文献

- [1] Altera. Quartus II Version 6.0 Handbook[DB/OL]. 2005
- [2] Altera. Nios II Software Developer's Handbook [EB/OL].<http://www.altera.com>,2005
- [3] 毕厚杰.新一代视频压缩编码标准[M]. 人民邮电出版社, 2005.5
- [4] ITU-T. H.264 / MPEG-4 Part 10 White Paper[EB/OL]. <http://www.vcodex.com>,2007
- [5] 国际电信联盟.ITU-T H.264 建议书。 <http://www.itu.int>, 2005
- [6] Atul Puri , Xuemin Chen , Ajay Luthra. Video coding using the H.264/MPEG-4 AVC compression standard [J]. Signal Processing: Image Communication,2004,9: 793–849.

原创性声明

郑重声明：此篇题为《H.264 视频解码 IP 核的设计与实现》的论文，是作者在导师指导下，于哈尔滨工程大学攻读硕士学位期间，进行研究工作的成果，除了文中标注的文献引用部分，论文中不包含其他人已经发表或撰写的研究成果。

电子设计题目： H. 264 视频解码 IP 核的设计与实现

电子设计作者签名： 梁盼 陶宝泉

日期： 二 00 九年八月二十八日

作者： 梁盼： 女， 1985， 湖北省随州市， 哈尔滨工程大学在读研究生， 研究方向： H.264 的视频编解码的研究与实现

作者： 陶宝泉： 男， 1983， 黑龙江省哈尔滨市， 哈尔滨工程大学在读研究生， 研究方向： 基于 SOPC 的视频处理技术

联系方式： 地址： 哈尔滨工程大学 1 公寓 304 室

邮编： 150001

Email: moqibijun@163.com 13101588183

zhongyuanmingshi@163.com 13766835740