第34卷 第12期 计 算 机 工 程 2008年6月
Vol.34 No.12 Computer Engineering June 2008
·开发研究与设计技术· 文章编号：1000—3428(2008)12—0239—03 文献标识码：A 中图分类号：TP303

基于FPGA的H.264去块滤波系统的优化设计

欧阳剑[1,2]，杜学亮[3]
(1. 中国科学技术大学软件学院，苏州 215021；2. 中国科学技术大学苏州研究院，苏州 215021；
3. 中国科学技术大学物理系，合肥 230026)

摘 要：提出一种H.264去块滤波系统的优化设计方法。通过合理设计流水线级数提高并行性，适当增加内部SRAM来提高系统速度和总线利用率，使用一种层次化的有限状态机设计方法，实现对数据流的精确控制并且有效降低硬件实现复杂度。基于FPGA的验证结果显示在最坏情况下滤波每个宏块平均只需220个时钟，比原有方案快10个时钟以上。

关键词：H.264标准；去块滤波；有限状态机；现场可编程逻辑器件(FPGA)

Optimized Design of H.264 Deblocking Filter System Based on FPGA

OUYANG Jian[1,2], DU Xue-liang[3]
(1. School of Software Engineering, University of Science and Technology of China, Suzhou 215021;
2. Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215021;
3. College of Physics, University of Science and Technology of China, Hefei 230026)

【Abstract】This paper presents an optimized deblocking filter method. The design improves the speed of system and efficiency of bus via a pipeline with suitable stages, enhancing the parallelism, and using some local SRAM. The paper presents a hierarchy Finite State Machine(FSM) design method which can control the data path precisely and reduce the complexity of hardware. The Field Programmable Gate Array(FPGA) simulation displays that filtering one Macro Block(MB) only costs 220 cycles in the worst case.

【Key words】 H.264; deblocking filter; Finite State Machine(FSM); Field Programmable Gate Array(FPGA)

在H.264/AVC中，基于块的帧内和帧间预测残差的DCT变换时，由于变换系数的量化过程相对粗糙而带来的反量化时的误差，以及在运动补偿预测中的匹配误差都会引起图像出现方块效应[1]，因此需要引入环路去块滤波器，改善编解码性能。去块滤波器的高度复杂性首先表现在它的运算量非常大，一般占到解码部分 30%的计算量[2]。在滤波过程中同一个4×4小块内的像素要执行滤波操作4次，数据吞吐量非常大。其次去块滤波器是高度自适应的滤波器，该滤波器既要滤掉方块效应造成的假边界又要保留图像原有的真实边界。

1 去块滤波算法

每次滤波计算需要8个像素数据、对应的滤波强度变量(*BS*)以及阈值(*Threshold*)变量 α 和 β(用来保护图像的真实边界，同时也用来判断输入的像素是否要滤波)。因此在滤波前，要先计算相应的 *BS*, α 和 β。

1.1 滤波顺序及滤波用到的邻接宏块中的数据

H.264 的官方文档规定先对 4×4 块的垂直边缘水平滤波，然后对其水平边缘垂直滤波。在对宏块左边缘水平滤波时需要用到左邻宏块的数据，对宏块上边缘滤波时需要用到上邻宏块的数据。

1.2 求解滤波强度

BS 控制滤波强度(Boundary Strength, BS)，它的值在0~4之间，与边界的性质有关。滤波强度与 *BS* 的值成正比，*BS*＝4 代表滤波强度最强，*BS*＝0 表示不用对该边缘进行滤波。如果4×4块处在SI或SP Slice里面或采用帧内编码的方式，当滤波边缘为宏块的边界时，则 *BS*＝4，否则 *BS*＝3。如果2个4×4块都不是处在SI或SP Slice且都不是采用帧内编码，若有一个块含有非零变换残差，则 *BS*＝2。如果上面的条件均不满足，则它们运动补偿的参考帧不一样时，*BS*＝1；参考帧一样但运动矢量差的绝对值大于4时，*BS*＝1；否则 *BS*＝0。*U* 和 *V* 分量的 *BS* 不用单独计算，它们和 *Y* 分量使用相同的 *BS*。

1.3 滤波计算流程

滤波操作就是平滑输入的数据，在滤波之前，还要进行判断，只有当如下条件成立，才进行滤波操作：

$$BS!=0\&\&Abs(p0-q0)<\alpha\&\&Abs(p1-p0)<\beta\&\&Abs(q1-q0)<\beta$$

2 算法实现

2.1 系统框架

系统构架如图1所示。

TopSram：24×32 bit，暂存当前滤波宏块的上邻接宏块中4×4小块的数据，共要24×32 bit。

LeftSram：2×24×32 bit，暂存当前要滤波的宏块左邻接的 4×4 小块的数据。在正常情况下当前宏块最右侧的一列4×4 小块滤波后要暂存在这里，下个宏块滤波时就可以从LeftSram读入左邻宏块数据，减少对总线的读写[3]。考虑到帧场自适应图像以宏块对为单位，因此要暂存当前宏块对的最右侧4×4小块，所以LeftSram大小为2×24×32 bit。

LocalSram：96×32 bit，暂存当前宏块水平滤波后的数据，垂直滤波时数据可以从LocalSram读出，减少对总线的访问[3]。该方案可以充分利用FPGA的片内SRAM。

作者简介：欧阳剑(1981－)，男，硕士研究生，主研方向：数字集成电路，嵌入式系统；杜学亮，博士研究生

收稿日期：2007-06-28 **E-mail：**oyjian@mail.ustc.edu.cn

Treg：数据转置寄存器。数据水平写入垂直读出或垂直写入水平取出可以实现数据的转置。水平滤波后的数据经过转置放到LocalSram中方波垂直滤波时取数据，垂直滤波后经转置将数据还原送到总线[3]。

BS&Threshold Generator：计算宏块滤波要用到的参数 *BS* 和 *Threshold*，当前宏块的滤波计算和下一宏块求 *BS* 与 *Threshold* 的计算是同时进行的，以提高并行性，降低单个宏块的处理时间。

FSM：该有限状态机控制整个滤波过程，是设计的关键，为了提高滤波效率同时简化硬件设计，把状态机划分成 3 级，包括帧图像级、宏块级和像素级，实验表明这样能极大简化控制流程，降低硬件实现难度。

Filter：一维FIR滤波器，在该设计中为了提高速度，同时去掉数据暂存寄存器[3]，降低电路的复杂度，采用了 4 级流水线设计，如图 2 所示。

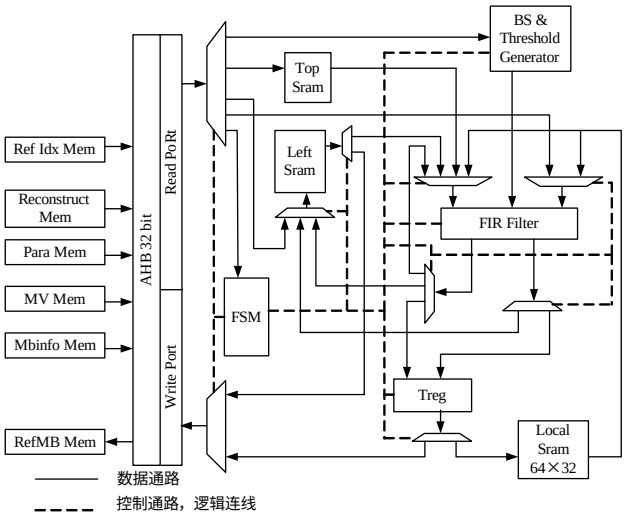


图 1　系统构架

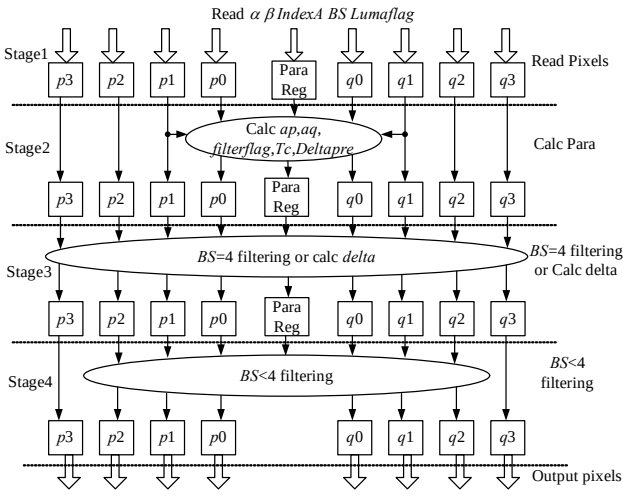
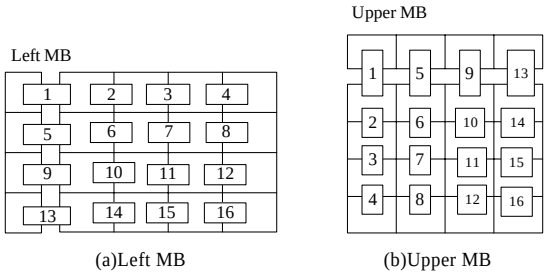


图 2　滤波计算流水线

2.2　滤波计算流水线规划

第 1 级流水线为读入并暂存 8 个像素和对应的参数*BS*, *α*, *β*, *Lumaflag*和*IndexA*。第 2 级流水线用来判断输入的数据是否满足滤波条件(*filterflag*)，计算式(1)，以及*BS*=4 和*BS*<4 时滤波过程中需要用到的中间变量：*ap*, *aq*, *Tc*及*Deltapre*。第 3 级流水线计算*BS*=4 时的滤波，同时计算*BS*<4 滤波要用到的中间变量*delta*。第 4 级流水线计算*BS*<4 时的滤波，并且寄存器暂存滤波后的数据。为了不打断流水线的操作，需要重新安排滤波顺序。滤波顺序如图 3[3]所示，这与ITU协议规定的滤波顺序并不矛盾。



(a)Left MB　　(b)Upper MB

图 3　重新规划的滤波顺序

按照这样的顺序，如图 4 所示，4×4 *Block*1 和*Block*2 滤波后，*Block*2 的数据可以紧接着和*Block*3 滤波，由于滤波计算用了 4 级流水线，因此数据从输入滤波器到输出要延时 4 个Cycle，假如在*Clock*0 时刻滤波器读入 4×4 *Block*0 和 4×4 *Block*1 的第 0 行数据*R*(0,0,1)，*Clock*1 读入第 1 行*R*(1,0,1)，*Clock*2 读入第 2 行*R*(2,0,1)，*Clock*3 读入第 3 行*R*(3,0,1)，同时*Row*(0,*B*0)，*Row*(0,*B*1)已经滤波完毕，处在写出阶段，在*Clock4*，需要读入已经滤波一次的*Row*(0,*B*1)和还没滤波的*Row*(0,*B*2)，此时*Row*(0,*B*0)可以从总线读入，而*Row*(0,*B*1)直接从滤波器的输出端读入即可。这样不需要额外设置暂存寄存器[3]保存*Block*1 的中间结果，节省了硬件资源的同时降低了控制的复杂度。

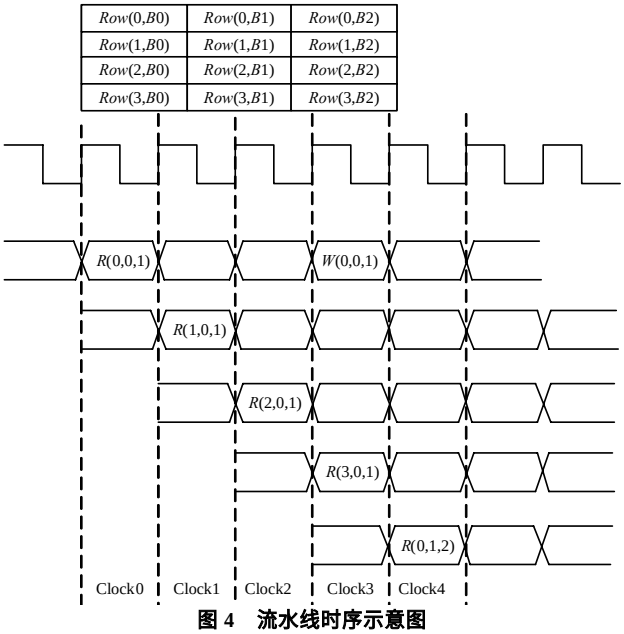


图 4　流水线时序示意图

2.3　层次化 FSM 设计

为了尽量提高滤波效率，除提高并行性之外，还可以增加对宏块的判断，根据宏块是否需要滤波而从外部 SDRAM 读入数据，这样可以从整体上降低单个宏块的平均处理时间，还可以降低对总线的读写频率。文献[3]将滤波划分成 8 种模式，其中在有些测试序列中完全不要滤波的情况占到 80%以上，为了简化硬件设计，可以把宏块的滤波分为 2 种模式：完全不用滤波(skip)和要滤波(normal)。如果宏块的所有 *BS* 值均为 0 或 *disable_deblocking_filter_idc*＝1，那么该宏块属于完全不用滤波的模式，否则归到要滤波的模式。

不同的滤波模式对应不同的处理方式，即使是同一种滤

波模式，对帧边缘宏块和宏块的边缘 4×4 小块都要作特殊的处理，因此为了能够精确控制数据的流向，且硬件实现方案要简单，需要合理设计控制 FSM。在该设计中，可以把 FSM 分成 3 个不同的抽象层设计，分别为 Frame 层，MB 层，Pixel 层。Frame 层具有最高抽象层次，MB 层次之，Pixel 最低，可以把低一级的 FSM 看做是高一级的子状态机。

2.3.1 Frame 层

对于一帧图像，根据滤波操作特性，可以把其内部的宏块分成 6 种：滤波时不需要读入左邻宏块和上邻宏块的数据(StartMB)；滤波时不需要读入上邻宏块的数据(TopMB)；滤波时不需要读入上邻宏块的数据，同时垂直滤波后该宏块最右侧的一列 4×4 小块不需要暂存在 LeftSram 里面，而是直接写回到总线(TopRightMB)；垂直滤波后该宏块最右侧的一列 4×4 小块不需要暂存在 LeftSram 里面，而是直接写回到总线(RightMB)；滤波时不需要读入左邻宏块的数据(LeftMB)；正常滤波，要读入左邻宏块和上邻宏块的数据，且滤波后最右侧一列小块的数据滤波后要送到 LeftSram (IntraMB)。

Frame 层的状态图如图 5 所示。*MB_end* 为宏块滤波结束的标志信号，*MBXcnt* 为在状态 *Frame_top* 和 *Frame_intra* 中统计已滤波宏块的计数器，*FrameWidth* 为帧的宽度(以宏块为单位)，*MBYcnt* 为在状态 *Frame_rignt* 中统计已滤波宏块的计数器，*FrameHeight* 为帧的高度(以宏块为单位)。

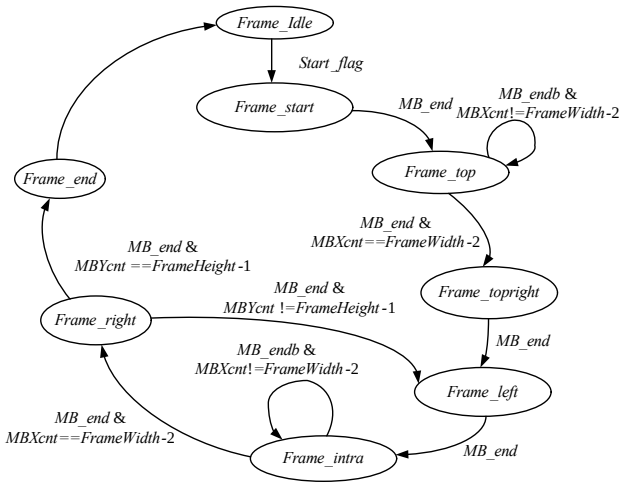


图 5 Frame 层状态图

2.3.2 MB 层 FSM

宏块的滤波分为 Skip 和 Normal 2 种情况，MB 层的 FSM 主要负责处理这 2 种情况的调度并决定宏块级的数据流向。每个宏块滤波时都和前一宏块有着极强的相关性，因为当前宏块滤波需要左邻宏块的数据，同时当前宏块的滤波输出会作为中间变量和下一宏块进行滤波，所以 MB 层的 FSM 要综合考虑当前宏块和前一宏块的相互关系，如表 1 所示。

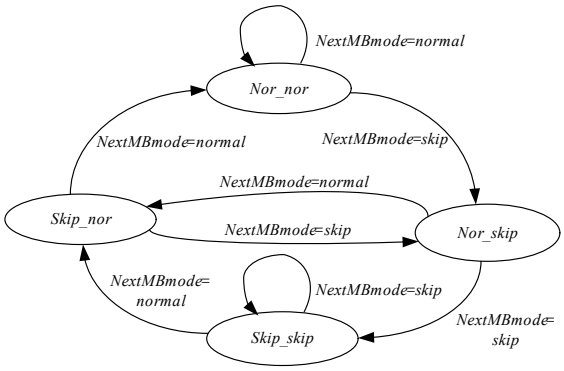
表 1 宏块间相互关系

前一宏块模式	当前宏块模式	状态	操作说明
nor	*nor*	*Nor_nor*	滤波前先读入上邻宏块的数据 24×32 bit
nor	*skip*	*Nor_skip*	向总线写回前一宏块中右侧小块的数据 24×32 bit
skip	*nor*	*Skip_nor*	滤波前先读入上邻和左邻宏块的数据，共 2×24×32 bit
skip	*skip*	*Skip_skip*	求下一宏块 *BS* 并判断下一宏块滤波模式

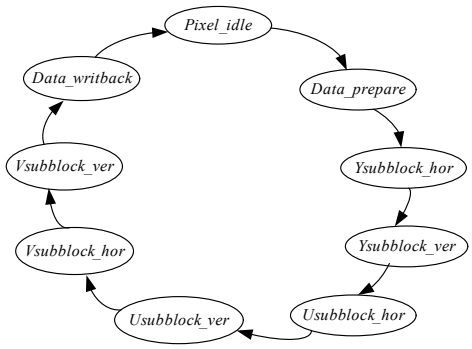
MB 层的状态图如图 6(a)所示。无论在什么状态，都要读入下一宏块的滤波信息，如 *disable_deblocking_filter_idc*、宏块类型、运动矢量等来计算 *BS*，以判断下一宏块是否要滤波，然后决定状态跳转。

2.3.3 Pixel 层 FSM

该层的 FSM 精确控制数据的流向和滤波器以及本地 SRAM 的动作。该层的 FSM 比较复杂，为了尽量简化设计，把每个 4×4 小块的滤波动作看作一个状态，每个 4×4 小块可以看作对应 2 个滤波动作：左边缘的水平滤波和上边缘的垂直滤波。因此每个小块对于 2 个 Pixel level 状态，例如对于亮度分量的第 0 个 SubBlock 可以设置如下 2 个状态：*Sub0Y_Hor* 和 *Sub0Y_Ver*，分别对应左边缘的水平滤波和上边缘的垂直滤波。其他 SubBlock 的状态设置相同。把滤波前邻接宏块的数据读入和滤波后的数据写回也归于该层状态，以实现数据流的精确控制。Pixel 层的状态图如图 6(b)所示。



(a)MB 层



(b)Pixel 层

图 6 MB 层与 Pixel 层的状态图

在实际应用中，各个状态还要细分，例如 *Data_prepare* 状态可以细分为 *LoadTopData* 和 *LoadLeftData*，对应取上邻和左邻宏块的数据。*Ysubblock_hor* 状态可以细分为 *Sub0Y_Hor*, *Sub1Y_Hor*, …, *Sub*15*Y_Hor*(*Y* 分量有 16 个 4×4 小块，按照地址递增顺序分配一编号，*UV* 分量同理)。这样有利与数据流的精确控制，便于硬件实现。

Frame 层、MB 层、Pixel 层的状态是紧密联系的，以共同精确控制数据的流向。

3 实验结果

用 Verilog HDL 进行 RTL 级描述，在 Synplify 平台上选用 Altera 的 Stratix II FPGA 器件进行静态时序分析，最大频率 110 MHz，关键路径出现在 filter 流水线的第 4 级。实验结果表明，MB 级当前状态为 *Nor_nor* 时，处理一个宏块需要 220 个 cycle；当前状态为 *Skip_nor* 时，如果 Frame level 状态

(下转第 244 页)