

引用格式: 许博闻, 王敏, 李琛, 等. 基于图像去噪的导向滤波算法的硬件实现[J]. 微电子学与计算机, 2019, 36(7): 22-26. [XU B W, WANG M, LI C, et al. Hardware design for image denoising using guided filter[J]. Microelectronics & Computer, 2019, 36(7): 22-26.]

基于图像去噪的导向滤波算法的硬件实现

许博闻^{1,2}, 王 敏¹, 李 琛², 王鹏飞², 王国兴¹

(1 上海交通大学 电子信息与电气学院, 上海 200240; 2 上海集成电路研发中心有限公司, 上海 201210)

摘 要: 在图像去噪算法领域, 导向滤波算法在去噪的同时能够保持更多的图像细节, 起到很好的边缘保持效应. 相比于其他边缘保持算法, 导向滤波算法运算速度更快, 在 CMOS 图像传感器领域, 能更快更有效的对图像进行去噪处理. 本文在导向滤波原有的算法公式上, 提出一种改进的硬件实现方法, 并应用于图像传感器芯片中, 像素分 8 通道进入模块进行处理, 采用 RGB 色彩模式, 算法模块最高时钟频率为 100 MHz, 每行最大处理 4 800 个像素, 一共占用 46. 875 kB 片上存储, 以及 XCRU040 的 13% 的 LUTS 资源.

关键词: 导向滤波; 边缘保持算法; CMOS 图像传感器; 硬件实现

中图分类号: TN402

文献标识码: A

文章编号: 1000-7180(2019)07-0022-05

Hardware design for image denoising using guided filter

XU Bo-wen^{1,2}, WANG Min¹, LI Chen², WANG Peng-fei², WANG Guo-xing¹

(1 School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China; 2 Shanghai IC R&D Center, Shanghai 201210, China)

Abstract: In the field of image denoising, the guided filter can reserve more image details and has a good edge-preserving effect. Comparing with other edge-preserving algorithm, the guided filter presents a higher efficiency. As a result, in the field of CMOS Image Process(CIS), the hardware design of the guided filter is able to process image denoising quickly and effectively. In this paper, based on the original algorithm of guided filtering, a hardware implementation method is proposed and used in the CIS. The pixels are divided into 8 channels to enter module for processing. Color mode is RGB. The highest clock frequency is 120 MHz and the pixels number processed per line by the module can reach 4 800. The total amount of the on-chip memory occupied by this module is 46. 875 kB. Total LUTS occupy 13% resource of XCKU040.

Key words: guided filter; edge-preserving algorithm; CMOS image process; hardware implementation

1 引言

图像噪声是指存在于图像数据中的不必要的或多余的干扰信息. 图像在生成和传输过程中, 会因为这些噪声的而降低图像质量, 为后续的图像处理和图像视觉效应产生不利的影响. 因此, 为了抑制噪声, 改善图像质量, 必须对图像进行噪声去除处理. 在图像去噪领域, 基本的滤波算法例如中值滤波, 高

斯低通滤波^[1]等属于各向同性滤波, 在处理噪声的同时, 会导致边缘被磨平, 影响图像质量. 导向滤波算法是何恺明提出的一个保边滤波的算法, 作为各向异性滤波^[2], 在去噪的同时更好的保存原图像的细节, 特别是边缘纹理部分. 在此基础上, 出于图像传感器项目的需求, 本文将导向滤波算法通过硬件实现, 充分利用硬件并行化, 流水线的特点, 提高运算速度, 为整个图像传感器项目不可或缺的一个组

成部分.

2 导向滤波算法

这里先简单介绍一下导向滤波的模型^[3]. 首先, 导向滤波模型认为, 某函数上的一点与邻近部分的点成线性关系, 因此可以用多个局部线性函数表达特定的函数, 当求特定函数上某一点的值时, 对所有线性函数的值做平均即可. 在图像方面, 可以认为图像是一个二维函数, 假设输出和输入在一个二维窗口内满足线性关系:

$$q_i = a_k I_i + b_k, \forall i \in w_k \quad (1)$$

式中, q_i 是输出图像的值; I 是输入图像的值; (a_k, b_k) 为窗口位于 k 时的系数; I 既可以是待滤波图像, 也可以是引导图像, 这就是被称为引导滤波的原因.

为了最小化输出图像 q 和输入图像 p 的差值, 下面公式的值最小:

$$E(a_k, b_k) = \sum_{i \in w_k} ((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2) \quad (2)$$

式中, 输入图像 p 只能是待滤波图像; ϵ 为正则化参数, 用于防止 a_k 过大. 根据公式(2), 得到系数 a_k, b_k 的表达式.

$$a_k = \frac{\frac{1}{|w|} \sum_{i \in w_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon} \quad (3)$$

$$b_k = \bar{p}_k - a_k \mu_k \quad (4)$$

式中, μ_k 和 σ_k^2 为输入图像 I 在窗口位于 k 时的均值和方差; $|w|$ 为窗口位于 k 时的像素数量; $\bar{p}_k$ 为输入图像 p 的均值.

上述公式是针对单个窗口的, 当我们求某一个像素的值时候, 该像素由多个窗口包围, 因此采取的策略是对所有包含该像素的窗口求平均值, 如公式(5)所示.

$$q_i = \frac{1}{|w|} \sum_{k: i \in w_k} (a_k I_i + b_k) = \bar{a}_i I_i + \bar{b}_i \quad (5)$$

式中, $\bar{a}_i = \frac{1}{|w|} \sum_{k \in w_i} a_k, \bar{b}_i = \frac{1}{|w|} \sum_{k \in w_i} b_k$, 为所有包含像素 i 的系数的均值.

当该模型应用于图像去噪时, 引导图像 I 为输入图像 p 本身, 因此我们对公式(3), (4) 进行简化:

$$a_k = \frac{\sigma_k^2}{\sigma_k^2 + \epsilon} \quad (6)$$

$$b_k = (1 - a_k) \mu_k \quad (7)$$

由公式(6)和(7)可以看出, 当 $\epsilon = 0$ 时, $a = 1, b = 0$, 输出图像为 $q = I$; 当 $\epsilon > 0$ 时, 在像素变化很小的区域 (σ_k^2 值很小), a 近似为 0, b 近似为 μ_k , 即对

该像素做了一个加权均值滤波处理; 在像素变化大的区域 (σ_k^2 值很大), a 近似为 1, b 近似为 0, 对图像的滤波效果很弱, 输出图像近似于输入图像, 有助于保持边缘. 这里可以看出 ϵ 的作用, 用于界定像素的变化范围, ϵ 越大, 滤波效果越明显.

3 硬件设计

3.1 数据处理流程

算法处理单元为 5×5 的单色通道数据, 处理的图像数据模式为 RGB 的正常模式, 如图 1 所示, 用 9×9 的数据模板从左到右, 从上到下移动, 并从 9×9 数据的数据中选择其中标写数据具体坐标的 5×5 数据作为后续算法处理的单元.

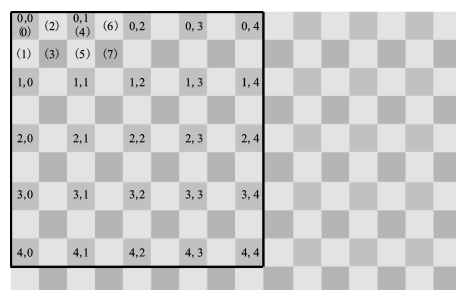


图 1 数据模式 (RGB normal 模式)

具体硬件实现为, 数据分 8 通道进入算法处理模块, 通道 0~7 对应图 1 中括号中的 0~7, 方向同样是先从左到右, 从上到下, 两个通道对应单像素的一行, Ch0 和 Ch4 对应一行中的一类像素, 同样的, (Ch1, Ch5), (Ch2, Ch6), (Ch3, Ch7) 分别对应一行中的一类像素, 这样每行的像素, 都是通过两个通道并行进入的, 这大大提高了数据的传输速度.

由于数据顺序进入, 通过控制模块, 将两个通道的同类数据进行合并, 在实际处理中, 单通道数据位宽为 10 比特, 合并完的数据为 20 比特, 从图 2 可以看出, 8 通道的数据进入算法模块后, 组合成 4 通道 20 比特的数据进行处理.

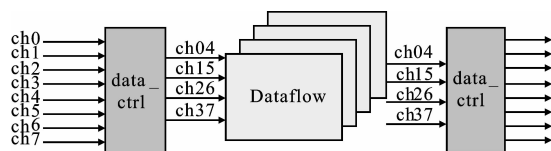


图 2 8 通道数据流

每一个单色数据对应 4 个 SRAM (300×80), 数据一共需要缓存 4 行, 通过控制片选信号 (cs0, cs1, cs2, cs3) 将数据分别存储在 4 块 SRAM 中, 如图 3 所示. 同时, 第五行数据进入模块后, 将会有两

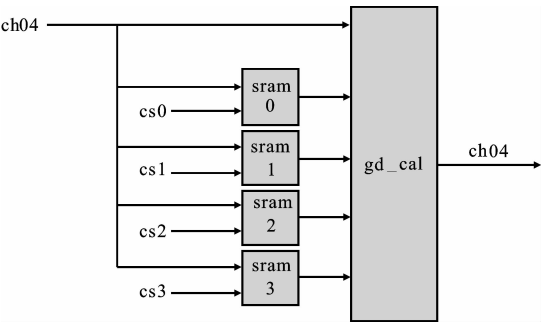


图 3 同类型数据

个功能,一个是与之前存入的四行,构成 8×5 大小的基本模块,送入到后续运算模块中;二是存入到用于存储第一行数据的 SRAM 中,替换老的数据,最终第五行的数据会完全覆盖第一行数据.

对于四行 SRAM 的数据刷新采取行卷绕的方式,并引入 mask 标志,用来标志行的顺序,例如,开始时,SRAM0-3 存储的行数为 1,2,3,4,当第五行的数据完全覆盖第一行时,SRAM0-3 存储的行数为 4,1,2,3,同理,当第二行刷新完时,变为 3,4,1,2,依次类推. 8×5 的运算模块一直采用 1,2,3,4 的四行再加入新的五个像素来计算. mask 的刷新是在每行结束时刷新,关于 mask[2:0]定义:0:第一行;1:第二行;2:第三行;3:第四行.

对于单个 SRAM,一次写 8 个像素,即一次在 SRAM 一个地址写入 80 比特数据. 一次读 8 个像素,即在对应的 SRAM 地址读出 80 比特数据如表 1 所示. 如图 4 所示,将 0 和 4 通道的数据合并后,缓存三个时钟周期,最终将四组共 80 比特的数据一并写入 SRAM,因此在物理上,四个是时钟周期发起一次读写操作,这样的情况下,就不需要每时每刻对 SRAM 进行访问,减小了一半的访存带宽.

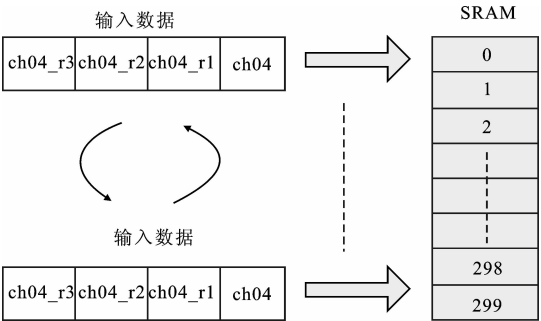


图 4 数据存储

表 1 单个 SRAM 参数

工作频率	数据位宽	访存次数	访存带宽
75 M	80 bit	37.5 M	3 Gb

由于 SRAM 一次能读出 8 个像素,因此,每次处理的数据模块大小为 8×5 ,分两个数据运算单元(DCU)进行处理,如图 5 所示,对于行开头 8 个列像素,0-4 列,1-5 列像素分别进入上下两个 DCU,进行运算,在下一个时钟周期,2-6 列,3-7 列进入. 处理完毕后,下一组 8×5 数据进入,同时缓存之前一组 8×5 中的后 4 列,如图 6 显示,其中灰色部分为数据缓存部分,此时,针对新的数据,两个 DCU 需要四个时钟周期将处理数据依次处理,具体为列 (4-8、5-9), (6-10、7-11), (8-12、9-13), (10-14、11-13). 以此类推,后面来的数据都按四个时钟周期处理,缓存上一组数据的后四列,与新来的数据合起来,在四个时钟周期内依次处理完,并缓存新数据的后四列,等待下一组数据.

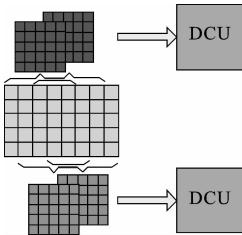


图 5 行开头的数据处理

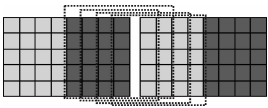


图 6 中间数据处理

由于前 8 列数据处理,只需要两个时钟周期,为保持数据连贯性,前 8 列数据延迟两个时钟周期进入到 DCU 模块. 这样数据的处理(4 个时钟周期处理完)与数据的读出(4 个时钟周期发起一次读操作)便保持了一致性,这也充分体现出硬件处理流水线的特点.

3.2 数据运算单元

导向滤波模块中的基本运算单元由加法器、减法器、乘法器、除法器构成,针对 5×5 单元模块,每一个数据的输出,一共需要经过 258 次运算,如图 7 所示,其中 mean 模块为均值模块,用与求矩阵的均值.

$p_{5 \times 5}$ 为 5×5 单色通道数据,分别求每个点的平方,并以中心 3×3 每个点的周围 3×3 个点的数值来计算平方值的均值,同时求 5×5 中心 3×3 区域每个点的均值以及均值的平方,通过相减得到方差. 根据求得的方差,得出系数 a_k 和 b_k 的值,并根据输入

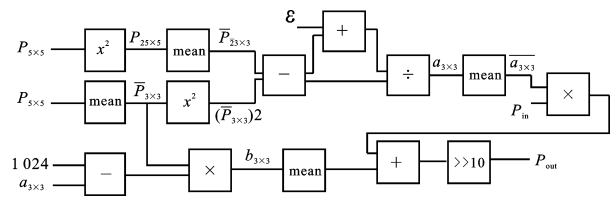


图 7 DCU 的运算设计

像素得到最终的输出像素. 运算单元的所需的计算次数统计如表 2.

表 2 单元运算次数统计

运算类型	均值的平方	平方的均值	系数 a_k 和 b_k	输出	总次数
运算次数	97	97	61	3	258

3.3 运算精度的提升

3.3.1 a_k 和 b_k 小数处理

为了提升算法中间数据精度, $a_{3\times3}$ 和 $b_{3\times3}$ 的元素为(0,1)之间的小数,经过多张图像多组 $\in$ 参数测试, $a_{3\times3}$ 和 $b_{3\times3}$ 放大1 024倍之后,对最后结果的影响是,像素灰度值最大差距小于 2 个灰度 LEVEL,对于 10 bit 的图像数据,是可以接受的,因此将数据放大1 024倍,如公式(8)和(9)所示:

$$a_{3\times3} = (1\,024 * \text{block3} \times 3_{\text{var}}) / (\text{block3} \times 3_{\text{var}} + \epsilon)$$

(8)

$$b_{3\times3} = (1\,024 - a_{3\times3}) * p_{3\times3}$$

(9)

相应的,最终结果如公式(10)所示,需要缩小1 024倍,在硬件实现中,我们将它右移 10 位即可.

$$p_{\text{out}} = (a_{3\times3} * p_{\text{in}} + b_{3\times3}) / 1\,024$$

(10)

3.3.2 均值模块的精度提升

一个数据运算单元一共使用了 20 次均值子模块,为其最主要的运算,作为整个设计而言,8 通道数据的同时刻输出,一共使用了 160 次 mean 模块,因此对于 mean 的均值子模块,为了减少运算数据位宽,采用图 8 所示的硬件设计,将九个数据分 4 层相加,这样相比输入数据,第四层的数据位宽提升 4 位,减少了硬件资源的利用,并采用乘法和移位的方式代替除法器,缩短运算周期.

由于去噪模块会对每一个像素都进行多次运算,因此对精度要求很高,四舍五入的处理不同,导致误差传递,放大,最终导致数据比对失败. $a/9$ 采用了小数点后 30 位的精度.具体的推导过程如下:

$$a/9 = a \times 2^{30} \div 9 \div 2^{30}$$

(11)

$$2^{30} \div 9 = 27'h71c71c7$$

(12)

$$a/9 = a \times 27'h71c71c7 \gg 30$$

(13)

选取小数后 30 位,主要是根据 C 代码与 RTL

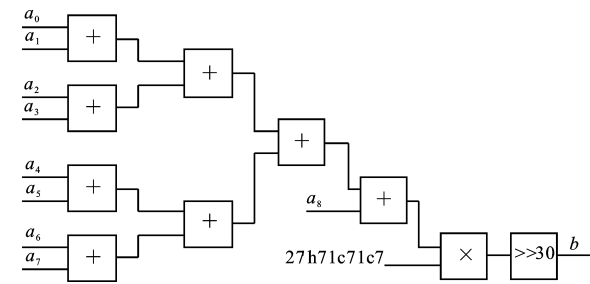


图 8 均值模块的设计

代码结果比对的结果一步一步修改的,验证平台在第四节介绍.从表 3 可以看出,当精度从 26 位调整到 30 位,未出现差值大于等于 1 的情况.

表 3 RTL 结果与 C 代码结果比对

测试像素:1 752×2 336(单通道)		
差值	数量	
	精度 26 位	精度 30 位
1	261	0
2	0	0
3	0	0
≥4	0	0

在本节的最后申明一下,本模块所有涉及到移位或者除法的部分都是采用四舍五入的方式,而非截位的方式.移位的做法为,若是向左移 10 位,则结果在加上第 10 位的值即可,关于除法的做法,将余数和除数的一半进行对比,若大于的话,结果加 1,否则结果不变.

4 模块实现与结果对比

对于算法模块的设计,仿真验证采用如图 9 所示,stimulus 为激励端,将激励信号传递给 DUT 算法设计模块,产生输出信号给 monitor 模块,然后将 monitor 的数据与 C 代码产生的数据进行对比,通过 scoreboard 展示出来.

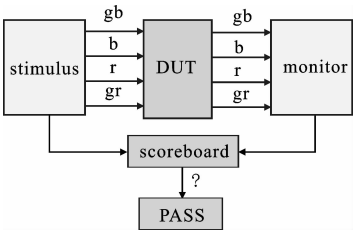
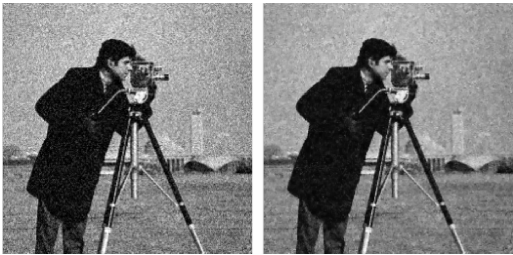


图 9 验证平台

对算法进行验证时,采用基于 FPGA 的验证测试,选取型号为 XCKU040^[4]的 Xilinx 的 FPGA 板卡,采用 Synplify 工具综合后,最高时序为 100

MHz,其中 LUT 占用率为 13%,DSP 利用率为 31%. 存储采用单端 SRAM,单块宽度 80 bit,深度为 300,一共使用了 18 块 SRAM,总的存储器大小为 46.875 kB.

图 10 为含有噪声的图像经过导向滤波处理后的图像,可以明显的看出高斯噪声得到了去除. 同样的,我们也针对 C 代码图像和 RTL 代码产生的图像进行了对比,C 代码与 RTL 处理效果几乎一样,看不出任何区别.



(a)原图像 (b)经过导向滤波算法
出后的图像

图 10 图像经过导向滤波处理后的效果图

表 4 总结了之前一些提出的关于导向滤波实现的成果,同时就处理的图像尺寸、类型和处理效率进行对比.

表 4 不同导向滤波算法实现的比较

作者	He ^[2]	Rhemann ^[5]	Kao ^[6]	本论文
硬件实现	CPU Core2 DUO 2.0GHz	GPU Geforce GTX480	ASIC TSMC 90nm	FPGA XC7V040
图像尺寸	1.0 M	1.0 M	2.07 M	16.3 M
图像类型	Gray-scale/RGB	RGB	Gray-scale	RGB/ Gray-scale
处理效率 (pixel/sec)	12.5 M/3.33 M	200 M	62 M	49 M/196 M

相比之前文献中的有关 CPU,GPU 和 ASIC 的设计,我们提出的设计的输出,每秒处理的图像要提升很多,模块处理的像素大小为 3 504×4 672,比起 1M 的像素大小,提升 16 倍,同等像素下,每秒处理速度同样提升 16 倍,而跟 2.07M 的像素相比,速度提升约为 8 倍. 由此可见,本设计对于像素的处理效率要比之前的设计高很多.

5 结束语

本论文提出基于图像去噪的导向滤波算法的硬件设计,处理的最大像素为 3 505×4 672,处理图像

格式为 RGB 模式,基于 XCKU040 的 FPGA 板卡,最高时序为 100 MHz,占用 LUT6%,并占用了总大小为 46.875 kB 片上存储. 相比之前的设计而言,占用存储率更低,处理效率更高. 本算法作为 16M 像素 CMOS 图像传感器项目的一部分,在 FPGA 上验证完成后,会参与后续的整体项目流片,后续的工作是在流片回来后验证算法的功能和处理效果.

参考文献:

[1] 冈萨雷斯. 数字图像处理[M]. 3 版. 北京:电子工业出版社,2010.

[2] LEE J E, CHUNG M K, ALEX A L. Evaluation of anisotropic filters for diffusion tensor imaging[C]// 3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2006. Arlington, VA, USA,2006.

[3] HE K, SUN J, TANG X. Guided image filtering[C]// European Conference on Computer Vision. Berlin, Springer-Verlag, 2010:1-14.

[4] KCU105 Board User Guide[EB/OL]. https://www.xilinx.com/support/documentation/boards_and_kits/kcu105/ug917-kcu105-eval-bd.pdf.

[5] HOSNI A, RHEMANN C, BLEYER M, et al. Fast cost-volume filtering for visual correspondence and beyond. [J]. IEEE Trans Pattern Anal Mach Intell. 2013, 35(2):504-511.

[6] CHIEH-CHI KAO, JUI-HSIN LAI, SHAO-YI CHIEN. VLSI architecture design of guided filter for 30 frames/s full-HD Video. [J]. IEEE Transactions on Circuits and Systems for Video Technology, 2014, 24(3):513-524.

作者简介:

许博闻 男,(1993-),硕士,硕士研究生. 研究方向为图像处理算法. E-mail:369920495@qq.com.

王敏 男,(1993-),硕士,助理工程师. 研究方向为生物系统电路集成.

李琛 男,(1982-),博士,高级工程师. 研究方向为人工智能算法与应用.

王鹏飞 男,(1985-),硕士,高级工程师. 研究方向为算法研究与软件工程.

王国兴 男,(1975-),博士,副教授. 研究方向为集成电路设计.