



OpenVX 高效能并行可重构运算通路的设计与实现

王 宇,李 涛,邢立冬,冯臻夫

(西安邮电大学 电子工程学院,西安 710121)

摘 要:针对专用硬件在处理图形图像时无法同时兼顾灵活性、可扩展性和时效性的问题,设计一种支持 OpenVX 1.3 标准的专用处理器。通过对 OpenVX 1.3 标准中的核函数进行数据通路映射,分析实现函数高效处理所需的运算单元数目,确定适用于该标准的数据通路运算器的结构。通过编写指令对数据通路进行重构,适应 OpenVX 标准的演进和扩展。应用 65 nm CMOS 工艺库对整体电路进行综合验证,实现的 OpenVX 可重构数据通路运算器面积为 21 076.21 μm^2 、功耗为 778.63 mW、系统主频为 500 MHz、吞吐量为 1.86 GB/s。实验结果表明,该数据通路运算器具有较强的可编程性和可扩展性,能够有效满足实时和高速的通用图像处理要求。

关键词:图像处理;计算机视觉;OpenVX 标准;并行处理;可重构

开放科学(资源服务)标志码(OSID):



中文引用格式:王宇,李涛,邢立冬,等.OpenVX 高效能并行可重构运算通路的设计与实现[J].计算机工程,2021,47(12):236-248.

英文引用格式:WANG Y, LI T, XING L D, et al.Design and implementation of OpenVX highly efficient parallel reconfigurable computing pathway[J].Computer Engineering, 2021, 47(12):236-248.

Design and Implementation of OpenVX Highly Efficient Parallel Reconfigurable Computing Pathway

WANG Yu, LI Tao, XING Lidong, FENG Zhenfu

(School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, China)

[Abstract] When used for graphics and image processing, application-specific hardware usually fail to balance flexibility, scalability and speed. To address the problem, an application-specific instruction set processor supporting the OpenVX 1.3 standard is designed. Through the data path mapping of the kernel function in OpenVX 1.3 and the analysis of the number of operation units needed to realize the efficient processing of the function, the suitable computing path structure for the standard is determined. The data path is reconstructed by writing instructions to adapt to the evolution and expansion of openvx standard. The overall circuit is verified by using 65nm CMOS process Library. The area of the OpenVX reconfigurable computing pathway is 21 076.2 μm^2 , the power consumption is 778.63 mW, the main frequency of the system is 500 MHz, and the system throughput is 1.86 GB/s. The experimental results show that the data path operator has strong programmability and scalability, and can effectively meet the requirements of real-time and high-speed general image processing.

[Key words] image processing; computer vision; OpenVX standard; parallel processing; reconfigurability

DOI: 10. 19678/j. issn. 1000-3428. 0060764

0 概述

近年来,随着电子计算机及半导体技术的快速发展,图像处理与计算机视觉(Computer Vision, CV)作为计算机应用领域中的重要分支,在军事、医学、地质勘探、多媒体等领域应用广泛^[1]。由于人们

对流畅视觉画面及视觉感受的要求越来越高,图形图像处理器设计及基于加快图像数据处理速度的计算机视觉算法优化不断面临新的挑战^[2]。

OpenVX 标准^[3]能为跨平台加速计算机视觉处理提供参照,从而实现计算机视觉处理性能和功耗的优化。其作为图像处理、图计算、深度学习和图形

基金项目:陕西省科技统筹项目(2015KTCQ013);陕西省教育厅协同创新中心项目(17JF032);陕西省教育厅科研计划项目(20JY058)。

作者简介:王 宇(1995—),男,硕士研究生,主研方向为集成电路系统设计、计算机图形学;李 涛,教授、博士;邢立冬,高级工程师、博士;冯臻夫,讲师、博士。

收稿日期:2021-02-01

修回日期:2021-03-15

E-mail:xc9833279500@163.com

预处理或辅助处理的标准,被诸多芯片企业(如NVIDIA、AMD、Intel、TI、Apple等)采用,具有广泛的应用前景。通过对OpenVX硬件设计进行调研,发现国内目前专门为OpenVX设计的硬件芯片较少。YAN等^[4]采用多态阵列架构(PAAG)处理器实现了OpenVX核函数中的像素级图像处理。HUANG等^[5]针对人脸识别项目中的预处理操作,提出基于OpenVX的并行化处理方法。TAGLIAVINI等^[6]介绍一个快速设计和优化OpenVX应用程序的框架,SAJJAD等^[7]提出将视觉通道的OpenVX图形级规范(graph-level specification)合成优化的FPGA框架,ABEYSINGHE^[8]提出一种基于性能模型的方法以优化OpenVX图形。以上方法均没有给出底层核函数的硬件加速设计方案,缺少实现OpenVX核函数的具体数据通路映射及分析。

本文设计一种支持OpenVX 1.3标准的并行可重构数据通路运算器,通过配置指令重构数据通路完成图像处理任务。此外,通过研究OpenVX中大量kernel函数算法,并采用相应的映射方案对不同类别的函数进行数据通路映射,从而设计出适合不同类别函数的数据通路运算器。

1 OpenVX介绍

OpenVX中的图概念如图1所示,将OpenVX中每一次对图像的基本操作看成整个流程中的一个节点(node),该节点通过处理前后的图像和其他node相连,形成图(graph)。OpenVX提供了一种自定义节点机制,用户可根据需要编写节点,并最终融合成图。OpenCV是一套完整的计算视觉软件系统,提供了图像处理的底层操作,虽然其有底层硬件加速函数(HAL),但OpenVX提供了一套更全面且结合了图计算(Graph Computing)方式的标准。

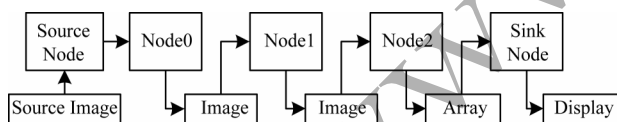


图1 OpenVX中的图概念

Fig.1 Graph concept in OpenVX

OpenVX计算视觉标准支持最基本的图像处理和计算视觉函数。OpenVX中多数kernel函数是针对图像的像素级处理,这些kernel函数构成了一个适用于硬件加速的函数子集^[9]。这些像素级处理包括点处理、局部处理、全局处理、特征提取4大类。OpenVX 1.3支持的kernel函数中包含的数据类型如表1所示。

表1 OpenVX 1.3支持的数据类型

Table 1 Data types supported by OpenVX 1.3

数据类型	定义	数据类型	定义
vx_int8	8位有符号值	vx_unit16	16位无符号值
vx_int16	16位有符号值	vx_unit32	32位无符号值
vx_int32	32位有符号值	vx_unit64	64位无符号值
vx_int64	64位有符号值	vx_float32	32位浮点值
vx_unit8	8位无符号值	vx_float64	64位浮点值

2 OpenVX函数的数据通路映射及分析

通过对不同函数进行相应数据通路的映射,使各个函数均具备高效的处理性能,得出不同函数实现所需运算器的种类及数目后,才能进行整体可重构数据通路运算器的设计。OpenVX 1.3标准中共包含58个kernel函数,本文映射函数的数据通路中包含点处理类27个,局部处理类10个,全局处理类7个,特征提取类7个。由于相同种类函数的数据通路相似,本文着重介绍点处理中的基本运算类、图像色系变换、仿射变换、透视变换、图像局部处理中的Sobel 3×3、图像全局处理中的均值及图像特征提取中的Canny边缘检测,并根据相应函数的映射方案和时序图对整体数据通路运算器所需的运算单元进行分析。

2.1 数据通路映射方案

数据通路映射方案包含流水线数据通路、并行数据通路、并行结构结合流水线数据通路。

1) 流水线数据通路

流水线处理电路^[10]采用面积换取速度的思想,可以大幅提高电路的工作频率,尤其对于图像处理任务中的二维卷积运算、图像滤波器、色系变换等。采用流水线设计可以保证一个时钟输出一个像素。由于对大部分图像处理任务而言,处理过程均采用串行的处理思路,因此流水线是较好的设计方式^[11]。如图2所示为典型的流水线结构,每个步骤独立为一个单独的处理单元,与其他处理单元同时运行,提高速度的同时也降低了设计的复杂度。

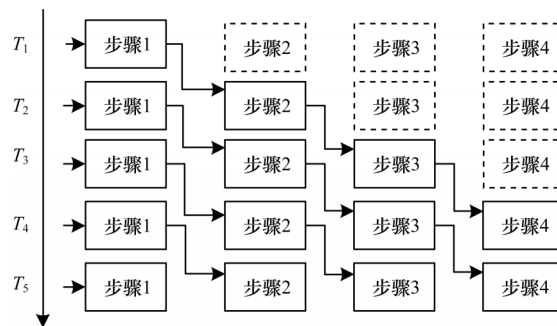


图2 流水线处理结构

Fig.2 Pipeline processing structure

2) 并行数据通路

在并行处理电路^[12]中,多组并行排列的子电路同时接收整体数据的多个部分进行并行计算。并行处理电路的结构如图3所示,对每个处理数据支路均生成对应的处理电路,这样虽然提高了整体电路的处理速度,但是却造成了更大的资源消耗,即用面积换取速度。

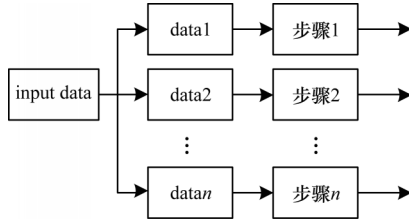


图3 并行处理电路结构

Fig.3 Parallel processing circuit structure

3) 并行结构结合流水线数据通路

并行处理电路中的子电路可以是简单的组合电路,也可以是复杂的时序电路,例如上面提到的流水线数据通路。如果受逻辑资源限制,无法同时处理全部数据,也可以依次处理部分数据直到完成全部数据的处理^[13]。

2.2 函数映射

2.2.1 点处理函数映射

点处理函数映射包含基本运算类映射、图像色系变换和仿射变换。

1) 基本运算类映射

基本运算类包括算术类运算和逻辑类运算。算术类运算包括绝对差、算术加法和算术减法,数据类型为 vx_unit8 和 vx_int16。算术类运算可以通过配置指令将加法器配置为8位或16位定点加法或减法器进行并行计算,并输出计算结果。逻辑运算类包括按位与、按位或、按位异或、按位非和逻辑移位。本文采取数据通路的合并,即逻辑类运算操作和加法操作进行数据通路的合并,以减小整体电路的面积。对于一个32位的基本运算器,通过指令配置,可以在每个时钟周期并行完成2个16位或4个8位的算术类运算或逻辑类运算。

2) 图像色系变换

该函数可以实现颜色的转换,将指定格式的图像转换成另一种格式。以RGB线性转换为例,转换公式如式(1)所示:

$$R_2 = (R_1 \times 900 + G_1 \times 88 + B_1 \times 10) \gg 10 \quad (1)$$

其中: R_1 、 G_1 、 B_1 代表原来的颜色通道; R_2 表示新的R通道。

对其进行数据通路映射如图4所示,分为3级流水线:第1级并行计算式(1)中的3个乘积项;第2级把前2个乘积项结果送入加法器 a_0 ,并把第3个乘积

项进行寄存;第3级把 a_0 和寄存器值相加,并移位输出。由图4可知,色系变换的流水线数据通路需要3个定点乘法器及2个定点加法器。

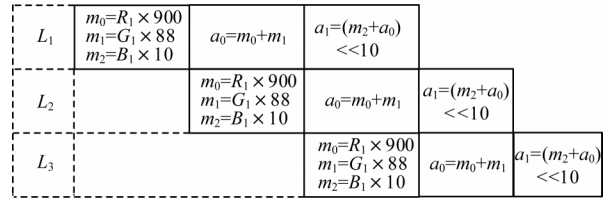


图4 色系变换流水线

Fig.4 Color convert pipeline

3) 仿射变换

仿射变换对图像进行仿射运算,支持的数据类型为 vx_unit8 和 vx_float32。该函数使用 2×3 的仿射矩阵 M 对输入像素进行仿射变换,具体计算如式(2)~式(4)所示:

$$x_0 = M_{1,1} \times x + M_{1,2} \times y + M_{1,3} \quad (2)$$

$$y_0 = M_{2,1} \times x + M_{2,2} \times y + M_{2,3} \quad (3)$$

$$O_{\text{output}}(x, y) = I_{\text{input}}(x_0, y_0) \quad (4)$$

仿射变换数据通路的映射如图5所示,第1级使用4个浮点乘法器并行计算式(2)和式(3)中的乘积项;第2级将4个输出结果两两相加;第3级将 $M_{1,3}$ 、 $M_{2,3}$ 分别和 a_0 、 a_1 相加并输出最终计算结果。由图5可知,仿射变换的流水线数据通路需要4个浮点乘法器及4个浮点加法器。

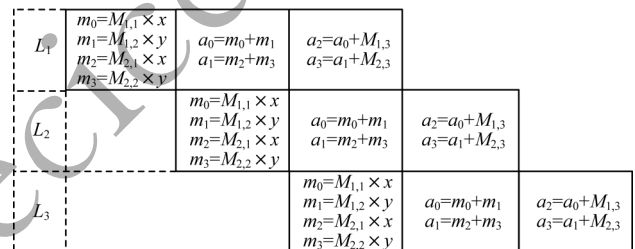


图5 仿射变换流水线

Fig.5 Affine transformation pipeline

透视变换对输入图像进行透视变换运算,支持的数据类型为 vx_unit8 和 vx_float32。该函数使用 3×3 的透视矩阵 M 对像素进行透视变换,具体计算如式(5)~式(8)所示:

$$x_0 = M_{1,1} \times x + M_{1,2} \times y + M_{1,3} \quad (5)$$

$$y_0 = M_{2,1} \times x + M_{2,2} \times y + M_{2,3} \quad (6)$$

$$z_0 = M_{3,1} \times x + M_{3,2} \times y + M_{3,3} \quad (7)$$

$$O_{\text{output}}(x, y) = I_{\text{input}}\left(\frac{x_0}{z_0}, \frac{y_0}{z_0}\right) \quad (8)$$

对其进行数据通路的映射如图6所示,分为4级流水线:第1级并行计算式(5)~式(7)中的乘积项;第2级并行计算式(5)~式(7)中的第1个加法;

第3级并行计算式(5)~式(7)中的第2个加法;第4级并行计算 $x_0/z_0, y_0/z_0$ 。由图6可知,透视变换的流

水线数据通路映射需要6个浮点乘法器、6个浮点加法器、2个浮点除法器。

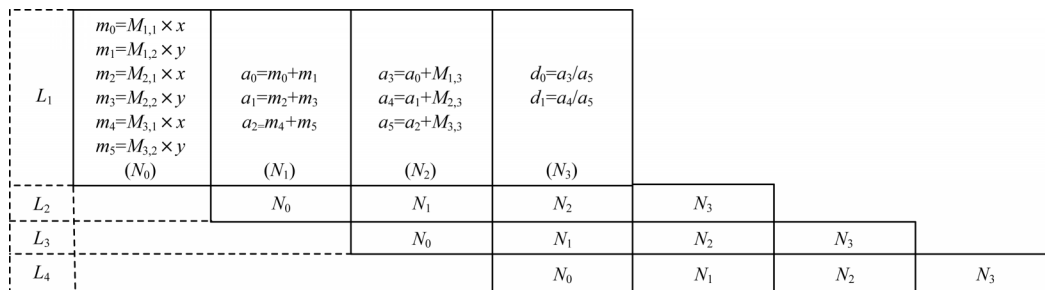


图6 透视变换流水线

Fig.6 Perspective transformation pipeline

2.2.2 局部处理函数映射

Sobel图像滤波支持的数据类型为vx_unit8,当滤波模板大小为 3×3 时,需要将9个像素 $p_0 \sim p_8$ 进行计算,其流水线数据通路如图7所示:第1级并行计

算8个像素的加法,第2级将上一级的结果两两相加,第3级将上一级的结果相加,第4级对最后一个像素进行加法计算并得出最终结果。由图7可知,Sobel 3×3 滤波流水线电路需要8个定点加法器。

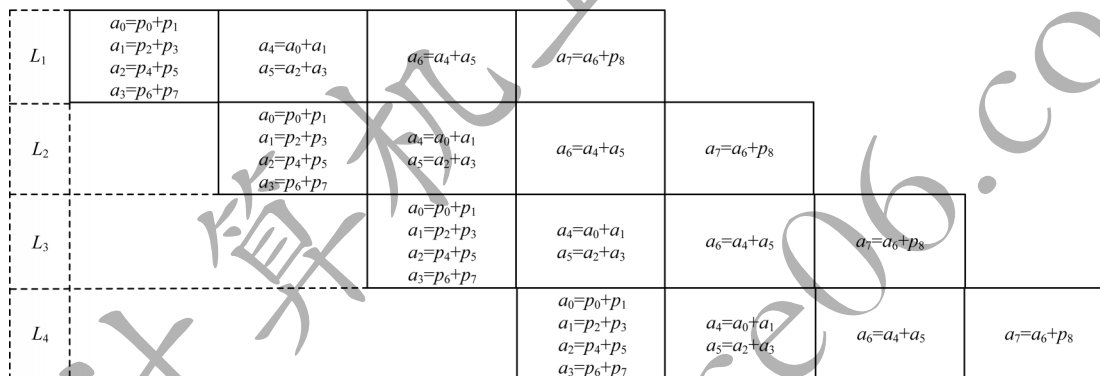


图7 Sobel滤波流水线

Fig.7 Sobel filter pipelined

本文通过电路的扩展和配置指令,可将中间结果写回寄存器堆或输出,实现滤波模板大小的可配置,其扩展性和兼容性更好。当滤波模板大小为 5×5 时,使用上述的流水线通路一次可计算9个像素值,而 5×5 模板需要计算邻域内25个像素值,需要循环此数据通路3次。在前2次循环中,需要将计算的中间结果写回寄存器堆2次,读出寄存器堆2次,第3次循环结束时完成计算。对于 7×7 的模板,需要写回寄存器堆5次,读寄存器堆5次,第6次循环结束后完成计算。

2.2.3 全局处理函数映射

在进行全局参数计算时,由于运算器的资源有限,有时需要将部分计算或比较结果存入寄存器堆中。以计算输入图像的均值为例,输入数据的类型为vx_unit8,输出数据的类型为vx_float32。均值的计算如式(9)所示:

$$\mu = \frac{\sum_{y=0}^H \sum_{x=0}^W S_{src}(x, y)}{W \times H} \quad (9)$$

由于实现Sobel 3×3 滤波流水线电路需要8个定

点加法器,因此进行平均值计算最多需要8个定点加法器约束,平均值计算采用并行结构结合流水线的的数据通路映射方案,如图8所示。映射过程为:第1级对4行像素并行累加计算,每1行计算结束后进行换行操作,一直迭代到最后一行像素;第 $n+1$ 级用 a_4, a_5 对4个累加结果两两计算;第 $n+2$ 级计算 a_4, a_5 的和以及 $W \times H$;第 $n+3$ 级计算浮点除法。完成平均值并行计算需要7个定点加法器、1个定点乘法器及1个浮点除法器。

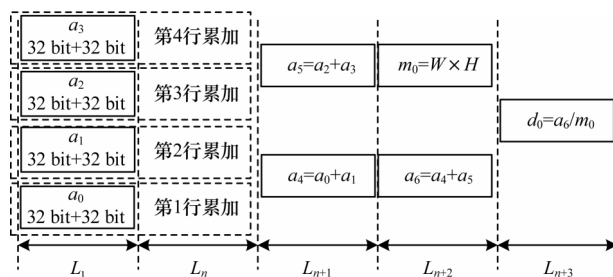


图8 并行计算均值

Fig.8 Parallel computing mean

2.2.4 特征提取类函数映射

Canny 边缘检测计算过程相对复杂,需要将中间结果写回存储中,处理过程主要分为3步:梯度幅值和方向计算,非极大抑制及边缘追踪。

1) 梯度幅值和方向计算

将输入图像与指定大小的垂直和水平方向的 Sobel 算子进行卷积,使用两个定向梯度图像 G_x 和 G_y 计算梯度大小和梯度方向。当计算梯度的类型为 VX_NORM_L1 时,梯度幅值为 $|G_x|+|G_y|$ 。由于 $\arctan(x)$ 计算复杂,且在非极大值抑制中只需知道

像素的梯度方向在哪块区域即可,不要求出实际的角度。因此,本文根据 G_x 、 G_y 的倍数关系及符号判断梯度方向,将梯度方向划分为4个区域 $path_1$ 、 $path_2$ 、 $path_3$ 和 $path_4$ 。计算梯度幅值和方向具体数据通路的映射如图9所示。其中:第1级利用 Sobel 算子计算水平、垂直方向的梯度幅值 G_x 和 G_y 以及符号类型;第2级计算梯度幅值 G ,判断 $path_1$ 及 $path_3$ 的类型;第3级判断 $path_2$ 及 $path_4$ 的类型;第4级把梯度幅值及梯度类型写回存储中。

L_1	$G_x=Sobel_x$ $G_y=Sobel_y$ $sign_x=G_x[0]$ $sign_y=G_y[0]$ $type=Sign_x \wedge Sign_y$ (N_0)	$G= G_x + G_y $ $path_1=G_x>(G_y \times 2.5)$ $path_3=G_y>(G_x \times 2.5)$ $start=path_1 path_3$ (N_1)	$path_2=type(\sim start)$ $path_4=\sim type(\sim start)$ $K_0=G>TH$ $K_1=G>TL$ (N_2)	$write_back$ (N_3)			
L_2		N_0	N_1	N_2	N_3		
L_3			N_0	N_1	N_2	N_3	
L_4				N_0	N_1	N_2	N_3

图9 梯度幅值和方向计算

Fig.9 Calculation of gradient amplitude and direction

2) 非极大抑制

非极大抑制是仅当检测像素的梯度大小在垂直于其边缘方向上大于或等于像素时,才将检测像素保留为潜在边缘像素。例如,当像素的梯度方向为 0° 时,则其梯度幅值大于像素为 90° 和 270° 时的梯度幅值,才保留像素作为边缘。非极大抑制的映射过程为:第1级从缓存中读出当前像素梯度及邻域内8个像素的梯度值及梯度方向类型 $\{path_4, path_3, path_2, path_1\}$;第2级根据 $\{path_4, path_3, path_2, path_1\}$ 的值分别在4个方向进行像素梯度幅值的比较,并输出比较结果;第3级将比较结果写回存储中。

3) 边缘追踪

输出图像的最终边缘通过双阈值法进行识别。所有梯度幅度大于高阈值的像素均标记为已知边缘像素(非0),小于等于低阈值的像素赋0。对于高阈值和低阈值间的像素使用8连通区域确定,只有与高阈值像素连接时才被视为边缘点。边缘追踪的映射过程为:第1级从存储中读出非极大抑制后像素比较的结果;第2级对大于高阈值的像素值直接赋255;大于低阈值的像素值进行8个邻域像素值与高阈值进行比较,若大于高阈值,则输出255,否则输出0。

2.3 所需运算单元分析

所需运算单元的分析如下:

1) 所需运算单元种类的分析

由表1可知,OpenVX 1.3 支持定点和浮点数据类型,所以设计的数据通路运算器中需要包含定点计算单元和浮点计算单元。根据上述函数的映射方案可知,色系变换中需要用到定点加法和定点乘法运算,仿射变换需用到浮点加法和浮点乘法运算,透视变换需用到浮点加法、浮点乘法和浮点除法计算,均值计算需用到定点加法、定点乘法、定点除法和浮点除法计算,Sobel 需用到定点加法计算。综上,设计的数据通路运算器需包含定点加法器、定点乘法器、定点除法器、浮点加法器、浮点乘法器和浮点除法器。

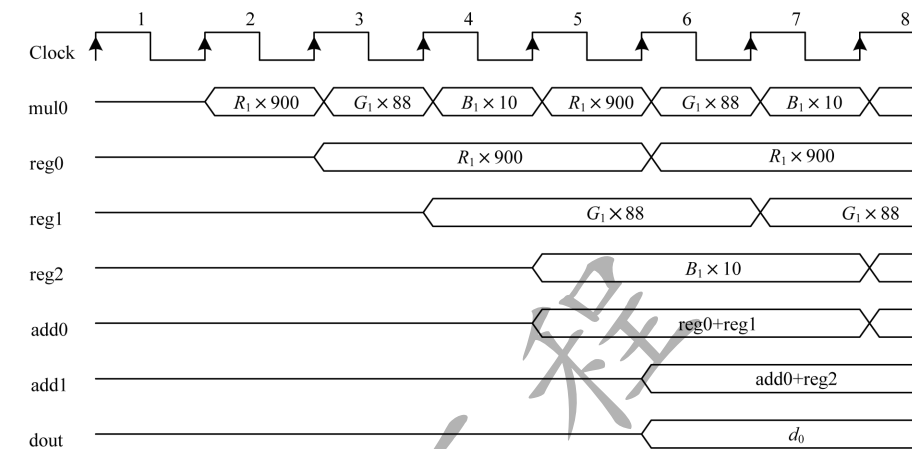
2) 所需运算单元数目的分析

由于采用不同数目运算器构建的流水线映射方案不同,处理不同函数所需的时间也不同,因此需要用时序图排序来分析各种 OpenVX 函数是否达到或者接近最佳性能。当采用不同个数的运算器时,分别对色系变换函数、仿射变换函数、均值计算函数、Sobel 滤波函数进行时序图的排序分析。

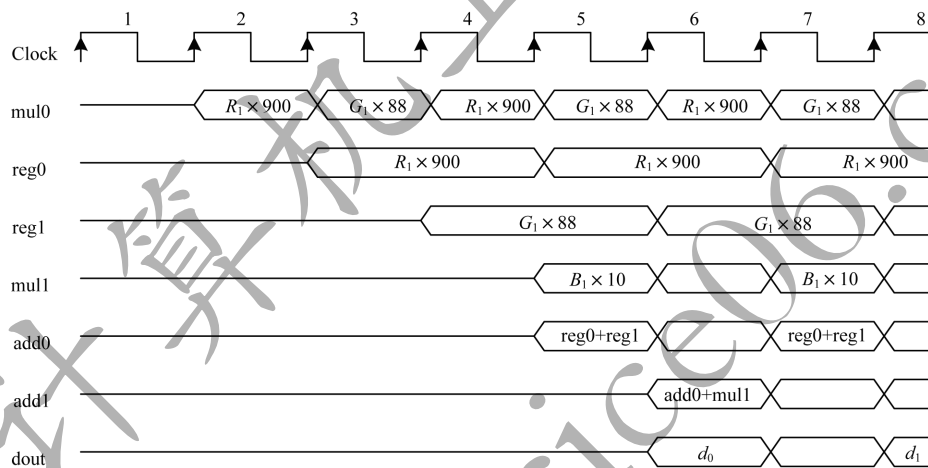
色系变换时序图如图10所示。由图10(a)可知,当采用1个定点乘法器和2个定点加法器时,色系变换函数在第5个时钟周期时处理完第1个像素,流水线输出时每隔2个时钟周期输出1个像素。由图10(b)可知,当采用2个定点乘法器和2个定点加

法器时,色系变换函数第5个时钟周期处理完第一个像素,流水线输出时每隔1个时钟周期输出一个像素,处理性能较好。由图10(c)可知,当采用3个

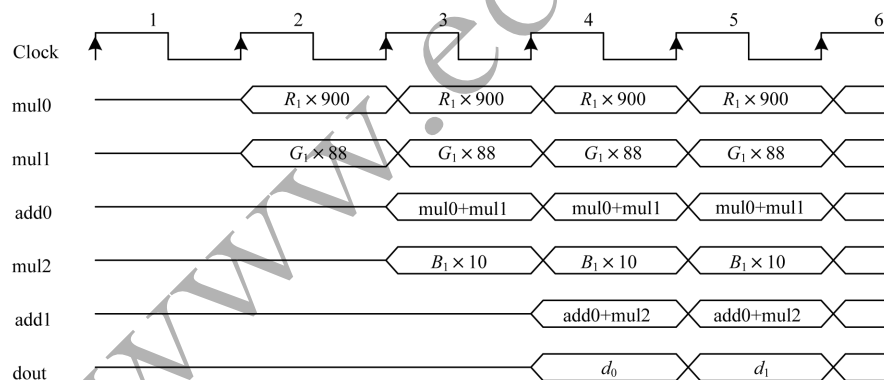
定点乘法器和2个定点加法器时,色系变换第3个时钟周期处理完第1个像素,流水线输出时每个时钟周期产生1个像素,处理性能最好。



(a)色系变换方案1



(b)色系变换方案2



(c)色系变换方案3

图10 色系变换时序图

Fig.10 Sequence diagram of color convert

仿射变换时序图如图11所示。由图11(a)可知,当采用1个浮点乘法器和4个浮点加法器时,仿射变换处理在第5个时钟周期时处理完第1个像素,流水线输出时每隔4个时钟周期输出1个像素。由图11(b)可知,当采用2个浮点乘法器,4个浮点加法器时,仿射变换处理

在第4个时钟周期时处理完第1个像素,流水线输出时每个时钟周期产生1个像素,处理性能较好。由图11(c)可知,当采用4个浮点乘法器、4个浮点加法器时,仿射变换处理第3个时钟周期处理完第一个像素,流水线输出时每个时钟周期产生一个像素,处理性能较好。

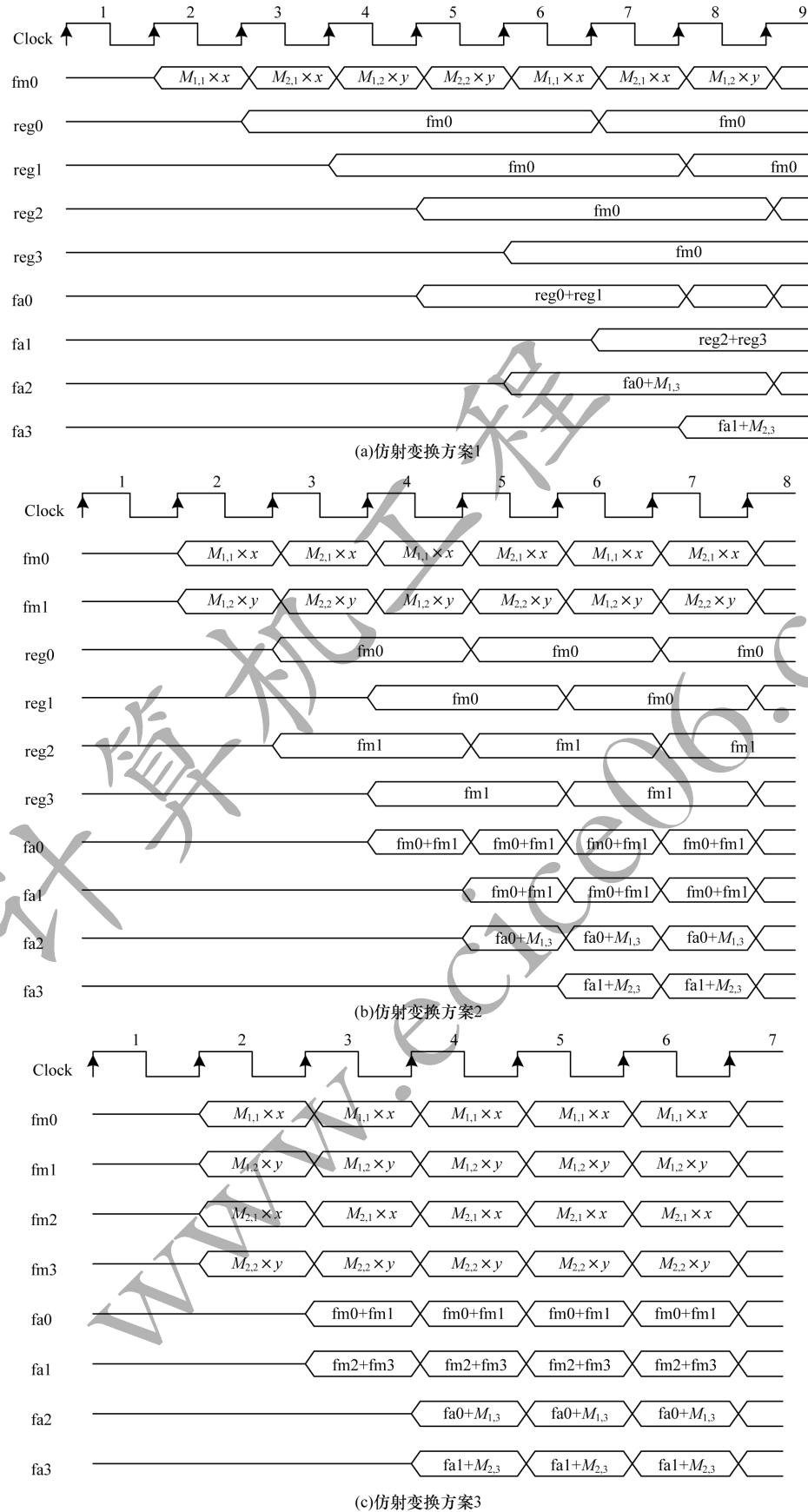
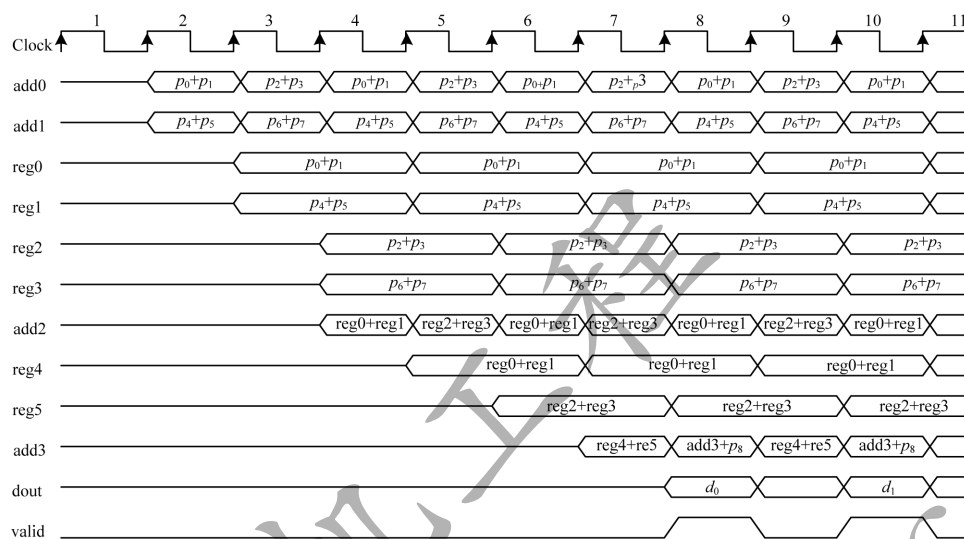


图 11 仿射变换时序图

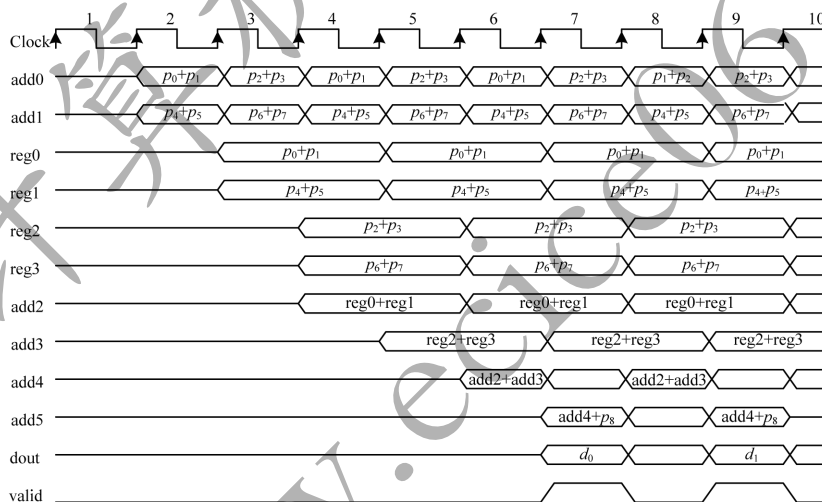
Fig.11 Affine transformation sequence diagram

Sobel滤波时序图如图12所示。由图12(a)可知,当采用4个定点加法器时,Sobel 3×3滤波在第7个时钟周期时处理完第一个像素,流水线输出时每隔1个时钟周期产生1个像素。由图12(b)可知,当采用6个定点加法器时,Sobel 3×3滤波在第6个时钟周期时处理完

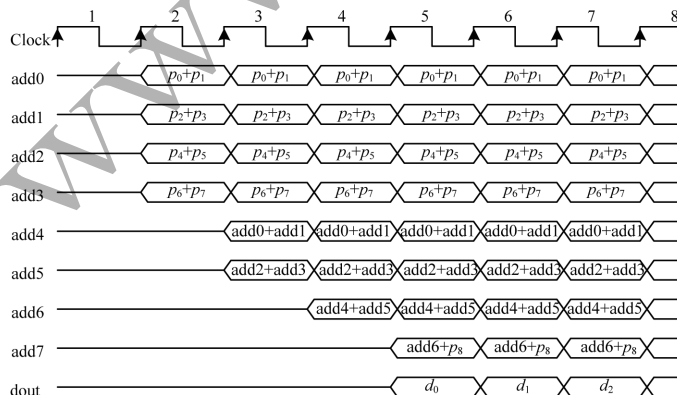
第1个像素,流水线输出时每隔1个时钟周期产生1个像素,处理性能较好。由图12(c)可知,当采用8个定点加法器时,Sobel 3×3滤波在第4个时钟周期时处理完第1个像素,流水线输出时每个时钟周期输出1个像素,处理性能最好。



(a)Sobel滤波方案1



(b)Sobel滤波方案2



(c)Sobel滤波方案3

图12 Sobel滤波时序图

Fig.12 Sobel filtering sequence diagram

均值计算时序图如图13所示。由图13(a)可知,当采用1个定点加法器、1个定点乘法器、1个浮点乘法器时,均值计算函数在第11个时钟周期时处理完第1个像素,流水线输出时每隔9个时钟周期完成1次均值计算。由图13(b)可知,当采用3个定点加法器、1个定点乘法器、1个浮点乘法器时,均值计算

函数第6个时钟周期处理完第1个像素,流水线输出时每隔4个时钟周期输出1个像素,处理性能较好。由图13(c)可知,当采用7个定点加法器、1个定点乘法器、1个浮点乘法器时,均值计算函数第4个时钟周期处理完第1个像素,流水线输出时每隔2个时钟周期输出1个像素,处理性能最好。

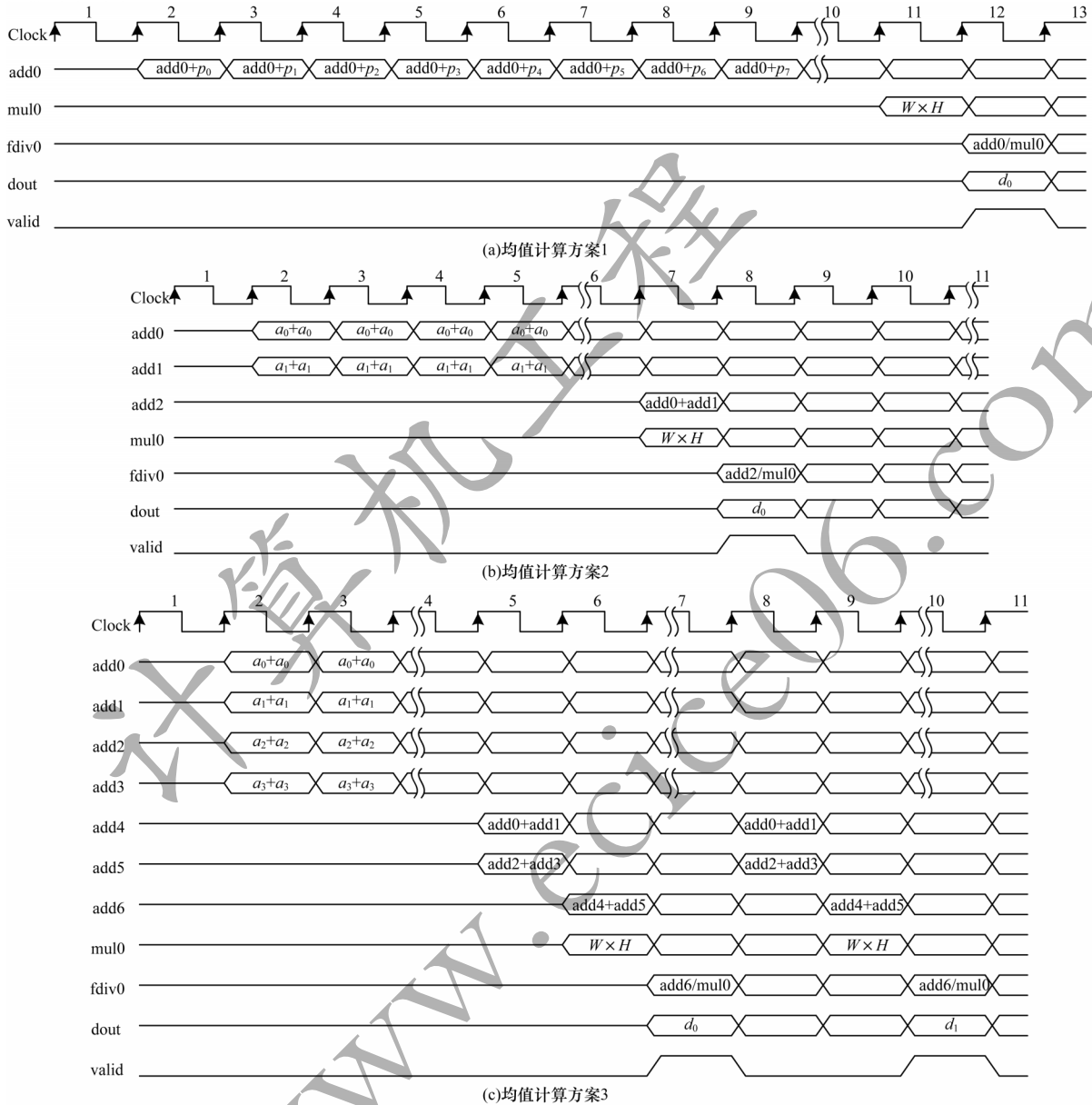


图13 均值计算时序图

Fig.13 Mean calculation sequence diagram

从函数处理的性能和电路面积以及电路的可编程性、灵活性考虑,确定定点加法器数目为8个,定点乘法器数目为4个,定点除法器数目为2个,浮点加法器数目为4个,浮点乘法器数目为2个,浮点除法器数目为2个,从而满足上述函数的高效通用处理要求。

3 数据通路运算器的设计

3.1 数据通路运算器的整体结构

根据OpenVX 1.3中各类函数的实现方法,将所需

计算单元进行分析后,设计出的OpenVX并行可重构数据通路运算器^[14-15]如图14所示,包括基本运算单元(arithmetic unit)、数据交叉互联模块(cross-bar interconnect)、指令配置模块(ins_config)、指令存储(ins_cache)、数据寄存器堆(register file)和读/写控制模块(rd/wr_ctl)。基本运算单元的设计是根据具体函数的映射要求,将加法器和乘法器设计为并行可拆分的结构,以适应多种数据格式的要求。数据通路运算器的执行过程如下:指令配置单元从指令存储中取出

相应的指令,发送给数据交叉互联模块、rd/wr_ctl单元和基本运算单元。rd/wr_ctl模块根据指令从寄存器堆中取出相应的运算数据并进行译码及截位处理,并输出到交叉互联电路。交叉互联电路根据相应的配置信息将数据送入运算单元。运算单元根据相应指令配置进行计算,并将计算结果写回到交叉互联电路或输出。

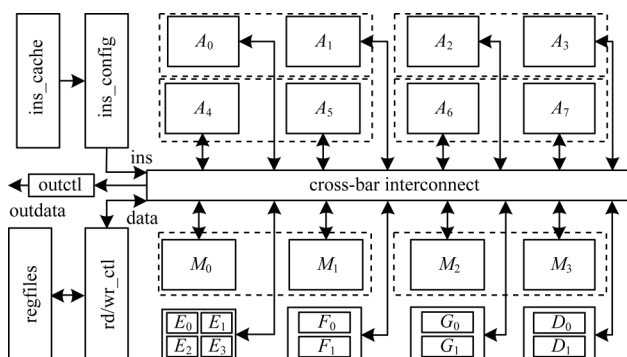


图14 数据通路运算器的总体结构

Fig.14 Overall structure of the data path arithmetic unit

数据通路运算器支持的计算类型和计算速度直接影响整体处理器的性能参数^[16]。本文设计的数据通路运算器共有22个基本计算单元,包括:8个定点加法器($A_0 \sim A_7$),4个定点乘法器($M_0 \sim M_3$),2个定点除法器(D_0, D_1),4个浮点加法器($E_0 \sim E_3$),2个浮点乘法器(F_0, F_1)及2个浮点除法器(G_0, G_1)。各运算单元之间采用数据通路合并、功能单元共享、电路资源复用等方法,以减少整

体面积占用^[17]。各运算单元支持多种数据类型,定点计算单元支持8位有/无符号数、16位有/无符号数、32位有/无符号数的运算,浮点单元支持32位浮点数的运算。

3.2 子模块内部结构

3.2.1 定点单元设计

定点单元支持的计算类型包括定点算术类和定点逻辑类计算,其中定点算术类计算包括定点加、减、乘、除、绝对值和比较。定点逻辑类计算包括逻辑与、逻辑或、逻辑非、逻辑异或和逻辑移位。

1) 定点加法器

定点加法器的设计依照数据通路的映射方案及数据类型,根据指令配置可并行计算2个32位,或4个16位,或8个8位的有符号、无符号数定点加法。加法运算的关键路径在进位链上^[18],为缩短这一关键路径,加法器的设计采用进位选择加法器(CSA),并设计基本单元为8位的进位选择加法器。

2) 定点乘法器

定点乘法器的设计采用混合乘加结构(Fused Multiply Add, FMA),依照数据通路的映射方案及数据类型,混合乘加运算器可根据控制信号完成32位的混合乘加运算,支持2组32位、4组16位、8组8位定点数的加法或乘法运算,提高数据并行计算的能力。采用混合乘加运算数据通路,可以减小逻辑电路大小。混合乘加器的电路设计如图15所示,功能复用上述的进位选择加法器,从而减少整个电路的面积。混合乘加器可根据控制信号对加法进位链进行截断,并输出相应的加法计算结果或乘法计算结果。

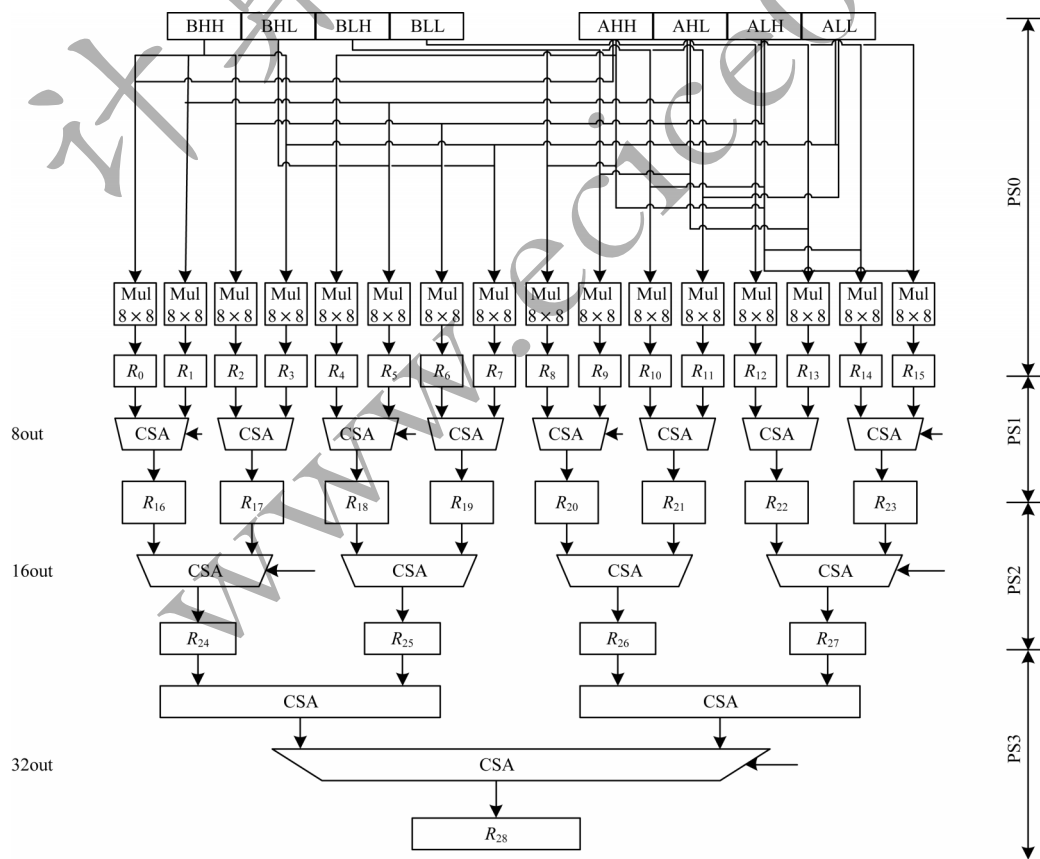


图15 混合乘加器结构

Fig.15 Fused multiply adder structure

果,图20(f)表示经过Canny边缘检测后的结果。由图20可知,使用不同算法在测试程序上进行功能仿真,均可达到预期处理目的,且和软件处理结果相同。

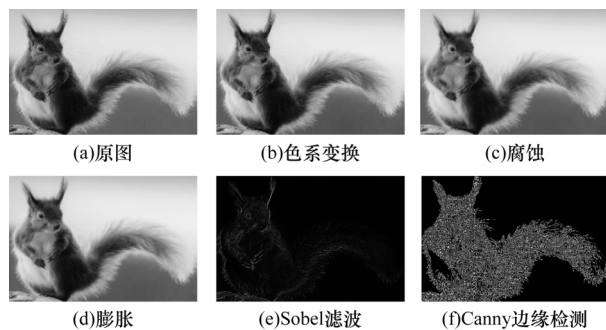


图20 不同函数的功能仿真结果

Fig.20 Functional simulation results of different function

将上述函数的算法在数据通路运算器上进行映射后,对每秒处理后的图像帧数进行统计如表2所示。由表2可知,当屏幕刷新率为640像素×480像素@60 Hz,800像素×600像素@60 Hz,1 024像素×768像素@60 Hz及以下标准时,色系变换、Sobel、腐蚀和膨胀均可达到快速刷新并显示的要求。

表2 不同分辨率下每秒处理的图像帧数

Table 2 Image frames per second processed under different resolutions

图像分辨率/像素	色系变换 /(frame·s ⁻¹)	Sobel /(frame·s ⁻¹)	腐蚀 /(frame·s ⁻¹)	膨胀 /(frame·s ⁻¹)	Canny /(frame·s ⁻¹)
640×480	258	332	498	335	131
800×600	157	211	299	214	75
1 024×768	101	128	187	130	48
1 600×1 200	41	53	79	51	20
2 048×1 536	25	32	48	31	12

当处理图片的分辨率在1 600像素×1 200像素、2 048像素×1 536像素及以上时,单个数据通路运算器无法满足实时性及高速率的通用图像处理要求,此时需要将多个数据通路运算器采用互联拓扑结构组合构成OpenVX并行处理器,在并行处理器上获得处理性能的提升,达到实时处理的目的。

4.2 性能分析

采用Synopsys公司综合工具Design Compiler,链接SMIC 65nm CMOS工艺库对整体电路及各运算器进行综合实现,并得到时序报告、面积报告和功耗报告。对于32位的除法运算,本文改进的牛顿迭代除法器最大主频可达520 MHz,相同实验条件下,对比传统的牛顿迭代除法器,其速率提升了22%。根据综合结果对整体电路进行性能分析,结果如表3所示,电路的时延为2 ns,且1/2 ns=500 MHz,因此系统最高时钟频率达500 MHz,系统的吞吐量为1.86 GB/s。

表3 整体电路性能分析

Table 3 Overall circuit performance analysis

指标	指标值
延迟/ns	2.00
面积/ μm^2	21 076.21
功耗/mW	778.63
吞吐量/(GB·s ⁻¹)	1.86

5 结束语

本文对OpenVX 1.3标准中kernel函数算法进行分析与映射,根据映射后电路的时序图分析运算器数目,并基于分析结果对可重构数据通路运算器构建整体结构及内部子模块。此外,对可重构数据通路运算器进行灵活编程,设计基于OpenVX 1.3标准的kernel函数算法,完成通用的图像处理。实验结果表明,基于OpenVX的并行可重构数据通路运算器能满足实时及高速率的通用图像处理要求。下一步将优化各运算单元性能,通过设计处理性能更好的运算器及编写相应的微指令,实现上层复杂特征提取函数的数据通路映射。

参考文献

- [1] 李涛,孙建,王鹏博. 基于PAAG的OpenVX核心库函数并行化实现[J]. 西安邮电大学学报,2015,20(2):7-10. LI T, SUN J, WANG P B. Parallel implementation of kernels of OpenVX based on PAAG[J]. Journal of Xi'an University of Posts and Telecommunications, 2015, 20(2): 7-10. (in Chinese)
- [2] RADHAKRISHNA G. The khronos OpenVX working group[EB/OL]. [2021-01-01]. https://www.khronos.org/registry/OpenVX/specs/1.3/html/OpenVX_Specification_1.3.html
- [3] BRILL F, ERUKHIMOV V, GIDUTHURI R, et al. Deploying an OpenVX graph to a target platform[M]. OpenVX Programming Guide, 2020: 65-84.
- [4] 延酉玫,李涛,王鹏博. OpenVX与三维渲染在多态GPU上的并行实现[J]. 计算机应用,2015,35(1):53-57. YAN Y M, LI T, WANG P B, et al. Parallel implementation of OpenVX and 3D rendering on polymorphic graphics processing unit[J]. Journal of Computer Applications, 2015, 35(1): 53-57. (in Chinese)
- [5] 黄灿. 基于OpenVX的图像预处理算法的并行化研究[J]. 现代计算机,2020(34):36-39. HUANG C. Parallel research of image pre-processing algorithm based on OpenVX[J]. Modern Computer, 2020(34): 36-39. (in Chinese)
- [6] TAGLIAVINI G, HAOUGOU G, MARONGIU A, et al. ADRENALINE: an OpenVX environment to optimize embedded vision applications on many-core accelerators[C]//Proceedings of the 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip. Washington D. C., USA: IEEE Press, 2015: 289-296.
- [7] SAJJAD T, PAYMAN B, ELI B, et al. AFFIX: automatic acceleration framework for FPGA implementation of OpenVX vision algorithms[C]//Proceedings of 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. New York, USA: ACM Press, 2019: 252-261.

- [8] ABEYSINGHE M, VILLARREAL J, WEAVER L, et al. OpenVX graph optimization for visual processor units[C]//Proceedings of the 30th International Conference on Application-Specific Systems, Architectures and Processors. Washington D. C., USA: IEEE Press, 2019: 123-130.
- [9] TAHERI S, HEO J, BEHNAM P, et al. Acceleration framework for FPGA implementation of OpenVX graph pipelines[C]//Proceedings of the 26th Annual International Symposium on Field-Programmable Custom Computing Machines. Washington D. C., USA: IEEE Press, 2018: 227-227.
- [10] 李岑, 贺光辉. 用于实时目标检测的FPGA神经网络加速器设计[J]. 微电子学与计算机, 2020, 37(7): 6-11.
LI C, HE G H. Design of FPGA-based neural network accelerator for real-time objective detection[J]. Microelectronics and Computer, 2020, 37(7): 6-11. (in Chinese)
- [11] 邵杰, 万书芹, 任凤霞. 基于ASIC的并行流水线级联半带滤波器设计[J]. 固体电子学研究与进展, 2020, 40(1): 60-65.
SHAO J, WAN S Q, REN F X. An ASIC based parallel pipelined cascade half band filter design[J]. Research and Progress of SSE, 2020, 40(1): 60-65. (in Chinese)
- [12] 吴皓月, 邓军勇, 山蕊, 等. 可重构阵列处理器Harris算法并行化实现[J]. 微电子学与计算机, 2019, 36(4): 67-71.
WU H Y, DENG J Y, SHAN R, et al. Reconfigurable array processor harris algorithm is implemented in parallel[J]. Microelectronics and Computer, 2019, 36(4): 67-71. (in Chinese)
- [13] 邓文齐, 郑启龙, 盛鑫, 等. 分簇架构处理器上卷积并行计算算法的研究[J]. 小型微型计算机系统, 2018, 39(3): 520-524.
DENG W Q, ZHENG Q L, SHENG X, et al. Study of convolution parallel algorithm on multi-cluster processor[J]. Journal of Chinese Computer Systems, 2018, 39(3): 520-524. (in Chinese)
- [14] 章子凯, 武继刚, 姜文超, 等. 容错处理器阵列的多逻辑列并行重构算法[J]. 计算机工程与科学, 2018, 40(1): 24-33.
ZHANG Z K, WU J G, JIANG W C, et al. A parallel multiple logical columns reconfiguration algorithm on fault-tolerant processor arrays[J]. Computer Engineering and Science, 2018, 40(1): 24-33. (in Chinese)
- [15] 李涛, 杨婷, 易学渊, 等. 萤火虫2: 一种多态并行机的硬件体系结构[J]. 计算机工程与科学, 2014, 36(2): 191-200.
LI T, YANG T, YI X Y, et al. Firefly 2: a hardware architecture for polymorphic parallel computers[J]. Computer Engineering and Science, 2014, 36(2): 191-200. (in Chinese)
- [16] 高向强, 冯春阳, 闫鑫, 等. 一种面向64位DSP处理器的可重构ALU研究及设计[J]. 微电子学与计算机, 2015, 32(10): 1-6.
GAO X Q, FENG C Y, YAN X, et al. Research and design of a reconfigurable ALU for 64 bit digital signal processor[J]. Microelectronics and Computer, 2015, 32(10): 1-6. (in Chinese)
- [17] 张嘉琛, 蒋剑飞, 毛志刚. 基于功能复用的高性能ALU设计[J]. 信息技术, 2010, 34(3): 58-60, 63.
ZHANG J C, JIANG J F, MAO Z G. Design of an efficient ALU based on reused logic structure[J]. Information Technology, 2010, 34(3): 58-60, 63. (in Chinese)
- [18] 刘容, 赵洪深, 李晓今. 基于改进型选择进位加法器的32位浮点乘法器设计[J]. 现代电子技术, 2013, 36(16): 133-136.
LIU R, ZHAO H S, LI X J. Design of 32-bit floating-point multiplier based on improved carry-select adder[J]. Modern Electronics Technique, 2013, 36(16): 133-136. (in Chinese)
- [19] 宋子豪, 李涛, 杜晓鸽, 等. 光栅化中多格式除法器的设计与实现[J]. 电子世界, 2020(5): 114-115.
SONG Z H, LI T, DU X G. Design and implementation of multi format divider in rasterization[J]. Electronics World, 2020(5): 114-115. (in Chinese)
- [20] 何军, 黄永勤, 朱英. 一种高性能四倍精度浮点乘加器的设计与实现[J]. 计算机工程, 2014, 40(2): 294-299.
HE J, HUANG Y Q, ZHU Y. Design and implementation of a high performance quadruple precision floating-point multiplier accumulator[J]. Computer Engineering, 2014, 40(2): 294-299. (in Chinese)
- [21] 车文博, 刘衡竹, 田甜. M-DSP中高性能浮点乘加器的设计与实现[J]. 计算机应用, 2016, 36(8): 2213-2218.
CHE W B, LIU H Z, TIAN T. Design and implementation of high performance floating-point multiply accumulate for M-DSP[J]. Journal of Computer Applications, 2016, 36(8): 2213-2218. (in Chinese)

编辑 赖玉玲