• 桂林理工大学 信息科学与工程学院

SOPC 技术基础课程

结 课 报 告

设 计 题 目：Nios 图像卷积指令

专 业：电子信息工程

班 级：电子信息工程

学 生：k2o

学 号：xmjtf@outlook.com

课程教师：

2019 年 4 月 17 日

摘要

设计了一种基于 ALTERA 的 FPGA 的自定义四点灰度卷积指令，使用 NIOSⅡ软核添自定义指令，使用 VHDL 硬件描述语言设计四像素点卷积的硬件，并挂载到 ALU 上，通过硬件方法处理四个灰度像素点和对应卷积核的卷积运算，实现 SOPC 对图像处理的卷积部分的加速。并在软件中配合该指令进行 C 语言程序的调用。该模块可以灵活的运用到其他 NIOSⅡ的系统上。使用时间戳记录时间，测试该软硬件结合的自定义指令处理方法针对不同大小的卷积核，相较于 C 语言软件处理卷积运算可以节省 30%以上的运行时间。

关键词：自定义指令；SOPC；图像卷积；硬件加速

目录

1. 图像卷积原理和特点

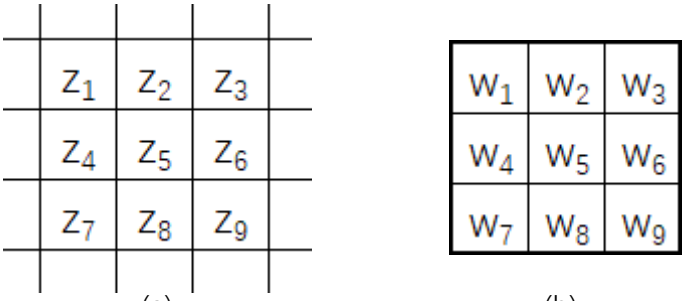
1.1 图像卷积特点

图像的卷积运算在图像处理中十分重要，卷积运算可以有效的提取图像的特征，也可以在图像初步处理的时候滤除噪声和还原退化的图片。现在机器视觉和机器学习的过程中往往需要对图像进行多次多层的卷积，这就需要消耗大量的运算资源。对于一幅 N×N 大小的灰度图像，利用 M×M 的卷积核进行卷积运算（M<N）需要进行的乘加运算的次数位为 M^2N^2 次[1]，低主频的单片机往往难以承当该计算，这一部分的计算可以交给 FPGA 利用硬件进行快速解决，在 ALTERA 公司的 Nios 软核基础上添加自定义卷积指令，用硬件实现大部分的乘法和加法操作，而 C 语言软件程序只要完成相应的赋值语句和少量的加法运算，这可以很好的提高整个系统的运算速度和处理的实时性。

1.2 图像卷积的原理

图像卷积是用原始图像的各个像素及其领域的灰度值图（1-1 a）与卷积核的对应元素核图（1-1 b）进行逐项相乘后相加的过程,运用式（1-1）并把结果 Y 重新赋值给 z_5。对卷积核的设计应该考虑空间占有数组来表达图像，对不同数组单元取不同值达到运算目的[2]

$$Y = \sum_{i=1}^{9} w_i z_i \qquad \text{(式 1-1)}$$



(a) (b)

图 1-1 卷积运算示例

如图（1-2 a）是输入的原始图像,图（1-2 b）是用于边缘检测的卷积核。利用上述公式对原始图像中的非边缘点和其邻域进行卷积运算，让卷积核遍历完了整张图象，会得到能展示出原图像边缘特征分布的矩阵如图（1-2 c）。

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	1	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1

(a)

-1	-1	-1
-1	8	-1
-1	-1	-1

(b)

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	-1	-1	-1	0	0
0	-1	-3	5	-3	-1	0
-1	-3	3	2	3	-3	-1
-3	3	1	0	1	3	-3
3	1	0	0	0	1	3

(c)

图 1-2 边缘检测

卷积核在与原图像中的对应特征卷积时会产生最大的结果，在灰度表现上为最亮的点，也就对应着在原图像中这一块区域是最有可能是该特征的部分。如图（1-2 c）中的值为 3 或 5 的点，对应了原图像中图形边缘的点。

2. SOPC 自定义指令构建

2.1 SOPC 的总体框架

SOPC 是可编程自定义的片上系统，使用非常的灵活，用户可以仅仅调用自己需要的外设，并加入自定义的指令或外设，用软硬件结合的方法有针对性的完成所需要的运算。对于该自定义卷积指令系统的 SOPC 的构建用到了以下的模块，Nios Ⅱ软核；JTAG_UART 用于和计算机的相互通信；PIO 用于表示系统正常工作的指示灯；因为图像本身的储存比较耗费空间，使用 sdram_controller 来操作片外的 32MB 的 SDRAM；timer 用于提供系统时钟和程序运行时间的计算；sysid 用于规定软硬件的统一。同时在 Nios Ⅱ之外还加入了锁相环 PLL 用于提供准确的系统时钟，和与系统时钟有-60 度相位差的 SDRAM 的时钟。如图（2-1）。

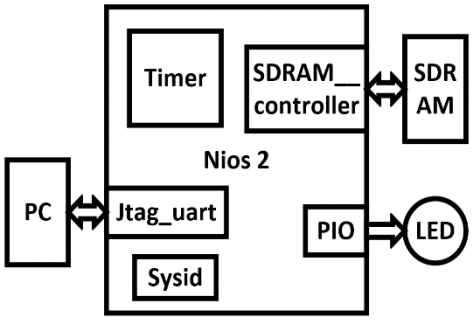


图 2-1 SOPC 系统结构

2.2 自定义指令部分的实现。

自定义指令可以让开发者定制 Nios II 处理器来满足特定应用程序的需求。在 ALU 上挂载自定义硬件逻辑块，可以加快软件的执行速度，并且可以方便的在软件上调用[3]。因为 NiosⅡ给出的组合逻辑的自定义指令的传入接口是两个 32bit 的输入 dataa 和 datab。而原始灰度图像的数据是 8 位的无符号整形的变量，卷积核为有符号的 8 位的变量，所以该系统在设计时把输入数据分割成为 4 个变量，其中 dataa 分为四个图像像素点的灰度值，datab 分为 4 个卷积核的权重值，如图（2-2）。

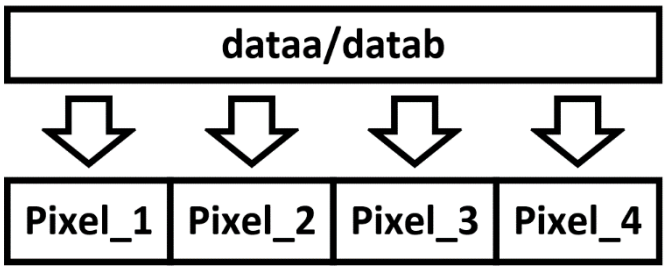


图 2-2 VHDL 实现数据分割

并把这四个灰度值和权重进行对应项相乘之后相加，交给输出接口 32 位的 result 输出，并且在输出结果前设计符号位扩展操作。这样就可以使用所提供的接口完成了四个像素点的卷积，可以一次性完成 2x2 像素点的灰度值卷积。通过重复调用这个指令并把结果相加，同样可以完成 3x3，4x4 等的图像卷积运算。

2.3 自定义指令软件调用。

运用 NoisⅡ内核所提供的自定义指令的调用宏 ALT_CI_CONVOLUTION_0(A,B)，可以调用 VHDL 设计的卷积指令。但是因为该宏命令的传入参数只有两个 32 位长的整形变量，所以应当依照上面 VHDL 硬件描述语言中的数据划分方法，在 C 语言软件上需要把四个传入的 8 位灰度像素点进行整合如图（2-3），对于用位移相或的方法，把原始图像的灰度移动到 32 位的变量中再传入自定义指令的宏中。

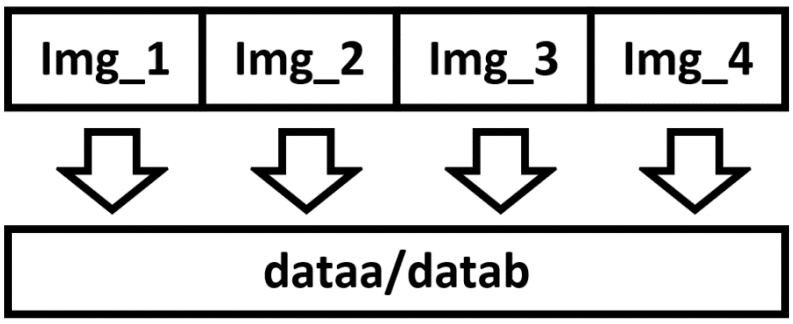


图 2-3 软件位移合并图像数据

3. 成果及运用

3.1 功能实现

利用该自定义指令系统进行图像卷积，采用 Laplacian 算子（图 4-1）的卷积核进行边缘提取[4]。在机器视觉中边缘提取往往是最基础的特征，具有很强的代表性。

-1	-1	-1
-1	8	-1
-1	-1	-1

图 4-1：Laplacian 算子

先分别对文字和智能车实验赛道进行退化处理，统一缩小到 80×80 像素，这样图像的边缘会发生模糊（图 3-2 a，c），之后再使用 Laplacian 算子的卷积核对该图像卷积，对应得到如下的结果图像（图 3-2 b，d）。可以看到该自定义指令系统可以完成对应的图像卷积特征提取的过程。

(a) (b) (c) (d)

图 3-2：边缘提取原图及输出图像

3.2 效率提升分析

在能完成功能的基础上，我们利用 nios 内核提供的时间戳（timestamp）功能，分别对完全 C 语言软件实现和 C 语言自定义指令实现的过程进行计时，并且输出两者完成所用的时钟周期。该 Nios Ⅱ 系统采用 50MHz 的时钟。

关于该指令的利用效率，因为该自定义指令需要一次执行 4 个乘法和加法运算，如果卷积核非 4 的倍数则需要加 0 填充，这会造成运算的浪费，故定义卷积指令利用率 S_c（式 3-1）来表示对于不同大小的卷积核，自定义卷积指令的利用效率，其中 T 为总的乘法加法次数，U 为实际有用的乘法加法次数。

$$S_c = \frac{U}{T} \times 100\% \qquad \text{(式 3-1)}$$

对于 3×3 的卷积核需要调用三次自定义四点卷积指令，共计 12 次乘加运算，实际有效乘加次数为 9,S_c=75%。经过测试，把通过计时多次得到结果取平均值得到完全软件实现所用时钟周期为 108397813，而通过 VHDL 构建的自定义指令需要的时钟周期为 62150273。为纯软件实现的 57.34%，速度提升了 74.4%。

4. 结语

该自定义指令系统，运用 VHDL 构建了一个四个灰度像素点的卷积器，相比于以往的完全靠软件实现图像卷积，加快了卷积的过程。用单个指令一次性完成四点的卷积，再理解图像和图像的滤波上可以使速度加快 30%以上，特别的现在对于图像的卷积层数越来越深，用这种软硬件结合的方式对于图像的实时处理有积极的意义。因为使用的是组合逻辑构建的自定义指令系统，可以忽略不同系统的系统时钟来进行移植，具有良好的移植性，也容易被不同的项目所运用。

5. 参考文献

[1] 宋鹤鸣. 一种 CNN 网络卷积并行计算方法[J]. 工业控制计算机.2019
[2] 章毓晋.图像处理和分析技术（第三版）[M].高等教育出版社.2014
[3] NiosⅡ Custom Instruction User Guide[EB].intel 2017.12
[4] 柏春岚.Matlab 在图像边缘提取中的应用[J]. 科技信息.2009

附录：

VHDL：

```
library ieee;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_1164.all;


entity self_convolution is
port(dataa,datab:in         std_logic_vector(31 downto 0);
             result    :out       std_logic_vector(31 downto 0));
end entity;

architecture behave of self_convolution is
signal temp1,temp2,temp3,temp4:signed(16 downto 0);
signal temp_t:std_logic_vector(16 downto 0);
begin
      process(dataa,datab)
      begin
            temp1 <= (unsigned(dataa(7 downto 0))*signed(datab(7 downto 0)));
            temp2 <= (unsigned(dataa(15 downto 8))*signed(datab(15 downto 8)));
            temp3 <= (unsigned(dataa(23 downto 16))*signed(datab(23 downto 16)));
            temp4 <= (unsigned(dataa(31 downto 24))*signed(datab(31 downto 24)));
            result(16 downto 0) <= temp1+temp2+temp3+temp4;
            temp_t(16 downto 0) <= temp1+temp2+temp3+temp4;
            if temp_t > 65535 then
                  result(31 downto 17) <= "111111111111111";
            else
                  result(31 downto 17) <= "000000000000000";
            end if;
      end process;
end behave;
```

C 语言代码：

```
char kernel_r[3][3] = {{-1,-1,-1},
                                    {-1, 8,-1},
                                    {-1,-1,-1}};
unsigned char img_out[78][78] = {0};
int main()
{
      alt_u32 time1;
      alt_u32 time2;
      alt_u32 time3;
```

```
unsigned char i = 0,j = 0;
int Y1,Y2,Y3,Y;
int A1,A2,A3,B1,B2,B3;
printf("pokemon_iibui!\n");
IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,0x5);
if (alt_timestamp_start() < 0)
{
	printf("Can't Start Timestamp...\n");
}
//while(1)
{
	time1 = alt_timestamp();
	for(i=1;i<79;i++)
	{
		for(j=1;j<79;j++)
		{
			Y = (img[i-1][j-1]*kernel_r[0][0]+img[i-1][j]*kernel_r[0][1]+
				img[i-1][j+1]*kernel_r[0][2]+img[i][j-1]*kernel_r[1][0]+
				img[i][j]*kernel_r[1][1]+img[i][j+1]*kernel_r[1][2]+
				img[i+1][j-1]*kernel_r[2][0]+img[i+1][j]*kernel_r[2][1]+
				img[i+1][j+1]*kernel_r[2][2]);

			if(Y < 0)
				Y = 0;
			else if(Y > 255)
				Y = 255;
			img_out[i-1][j-1] = Y;
		}
	}
	time2 = alt_timestamp();
	B1 = (*(int*)(*(kernel_r)));
	B2 = (*(int*)(*(kernel_r)+3));
	B3 = (*(int*)(*(kernel_r)+6));

	for(i=0;i<78;i++)
			{
				for(j=0;j<78;j++)
				{
					A1=0;A2=0;A3=0;
					A1 = (img[i][j]|(img[i][j+1]<<8)|(img[i][j+2]<<16));
					A2 = (img[i+1][j]|(img[i+1][j+1]<<8)|(img[i+1][j+2]<<16));
					A3 = (img[i+2][j]|(img[i+2][j+1]<<8)|(img[i+2][j+2]<<16));
					Y1 = (int)ALT_CI_CONVOLUTION_0(A1,B1);
					Y2 = (int)ALT_CI_CONVOLUTION_0(A2,B2);
```

```
                        Y3 = (int)ALT_CI_CONVOLUTION_0(A3,B3);
                        Y = Y1+Y2+Y3;

                        if(Y < 0)
                            Y = 0;
                        else if(Y > 255)
                            Y = 255;
                        img_out[i][j] = Y;
                    }
                }
            time3 = alt_timestamp();
            printf("func1 needs %u\n",(unsigned int)(time2-time1));
            printf("func2 needs %u\n",(unsigned int)(time3-time2));
            usleep(10000000);
        }
    return 0;
}
```

顶层图设计：