高斯差分图像处理模块设计

摘要

本文通过FPGA实现一个高斯差分（Difference of Gaussian,DOG）图像处理模块，两个高斯滤波器掩膜尺寸分别为3×3、7×7；输入图像为256×256，8位灰度图。该模块由Verilog HDL语言编写，工作频率100MHz，并基于主流FPGA开发平台Vivado进行仿真分析，同使用Matlab对仿真结果进行验证。

引言

在计算机视觉中，**高斯差分函数**，即**高斯差**（英语：Difference of Gaussians，简称“DOG”）是一种将一个原始灰度图像的模糊图像从另一幅灰度图像进行增强的算法，通过DOG以降低模糊图像的模糊度。这个模糊图像是通过将原始灰度图像经过带有不同标准差的高斯核进行卷积得到的。用高斯核进行高斯模糊只能压制高频信息。从一幅图像中减去另一幅可以保持在两幅图像中所保持的频带中含有的空间信息。这样的话，DOG就相当于一个能够去除除了那些在原始图像中被保留下来的频率之外的所有其他频率信息的带通滤波器 [1] 。

高斯滤波是一种线性平滑滤波，适用于消除高斯噪声，广泛应用于图像处理的减噪过程。通俗的讲，高斯滤波就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到。高斯滤波的具体操作是：用一个模板（或称卷积、掩模）扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。

在图像处理中，高斯滤波一般有两种实现方式，一是用离散化窗口滑窗卷积，另一种通过傅里叶变换。最常见的就是第一种滑窗实现，只有当离散化的窗口非常大，用滑窗计算量非常大（即使用可分离滤波器的实现）的情况下，可能会考虑基于傅里叶变化的实现方法。本文采用第一种实现方式。

算法原理

高斯函数具有五个重要的性质，这些性质使得它在早期图像处理中特别有用．这些性质表明，高斯平滑滤波器无论在空间域还是在频率域都是十分有效的低通滤波器，且在实际图像处理中得到了工程人员的有效使用．高斯函数具有五个十分重要的性质，它们是：

（1）二维高斯函数具有旋转对称性，即滤波器在各个方向上的平滑程度是相同的．一般来说，一幅图像的边缘方向是事先不知道的，因此，在滤波前是无法确定一个方向上比另一方向上需要更多的平滑．旋转对称性意味着高斯平滑滤波器在后续边缘检测中不会偏向任一方向．

（2）高斯函数是单值函数．这表明，高斯滤波器用像素邻域的加权均值来代替该点的像素值，而每一邻域像素点权值是随该点与中心点的距离单调增减的．这一性质是很重要的，因为边缘是一种图像局部特征，如果平滑运算对离算子中心很远的像素点仍然有很大作用，则平滑运算会使图像失真．

（3）高斯函数的傅立叶变换频谱是单瓣的．正如下面所示，这一性质是高斯函数傅里叶变换等于高斯函数本身这一事实的直接推论．图像常被不希望的高频信号所污染(噪声和细纹理)．而所希望的图像特征（如边缘），既含有低频分量，又含有高频分量．高斯函数付立叶变换的单瓣意味着平滑图像不会被不需要的高频信号所污染，同时保留了大部分所需信号．

（4）高斯滤波器宽度(决定着平滑程度)是由参数σ表征的，而且σ和平滑程度的关系是非常简单的．σ越大，高斯滤波器的频带就越宽，平滑程度就越好．通过调节平滑程度参数σ，可在图像特征过分模糊(过平滑)与平滑图像中由于噪声和细纹理所引起的过多的不希望突变量(欠平滑)之间取得折衷．

（5）由于高斯函数的可分离性，较大尺寸的高斯滤波器可以得以有效地实现．二维高斯函数卷积可以分两步来进行，首先将图像与一维高斯函数进行卷积，然后将卷积结果与方向垂直的相同一维高斯函数卷积．因此，二维高斯滤波的计算量随滤波模板宽度成线性增长而不是成平方增长．

基本理论

首先，高斯函数表示定义为

$$G_{\sigma_1}(x,y)=\frac{1}{\sqrt{2\pi\sigma_1^2}}exp\left(-\frac{x^2+y^2}{2\sigma_1^2}\right)$$

其次，两幅图像的高斯滤波表示为：

$$g_1(x,y)=G_{\sigma_1}(x,y)*f(x,y)$$

$$g_2(x,y)=G_{\sigma_2}(x,y)*f(x,y)$$

最后，将上面滤波得到的两幅图像g1和g2相减得到：

$$g_1(x,y)-g_2(x,y)=G_{\sigma_1}*f(x,y)-G_{\sigma_2}*f(x,y)=(G_{\sigma_1}-G_{\sigma_2})*f(x,y)=DoG*f(x,y)$$

即：可以DOG表示为：

$$DoG\triangleq G_{\sigma_1}-G_{\sigma_2}=\frac{1}{\sqrt{2\pi}}\left(\frac{1}{\sigma_1}e^{-(x^2+y^2)/2\sigma_1^2}-\frac{1}{\sigma_2}e^{-(x^2+y^2)/2\sigma_2^2}\right)$$

MATLAB的算法实现

```matlab
clear all
clc
imfinfo('gaosi8.jpg')%图像信息
image=imread('gaosi8.jpg');
imshow(image),title('原图像');

image4=double(image);
image64=double(image);
image_diff=zeros(256);
image_diff1=zeros(256);

conv_k3=[1,2,1];%3阶高斯卷积核
conv_k7=[1,6,15,20,15,6,1];%7阶高斯卷积核
%3阶和7阶高斯滤波实现
%行高斯卷积运算
for i=1:1:256
    for j=1:1:256
        if j==1
            image4(i,j)=dot([image4(i,j+1),image4(i,j),image4(i,j+1)],conv_k3)/4;
            image64(i,j)=dot([image64(i,j+3),image64(i,j+2),image64(i,j+1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        elseif j==2
            image64(i,j)=dot([image64(i,j+3),image64(i,j+2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        elseif j==3
```

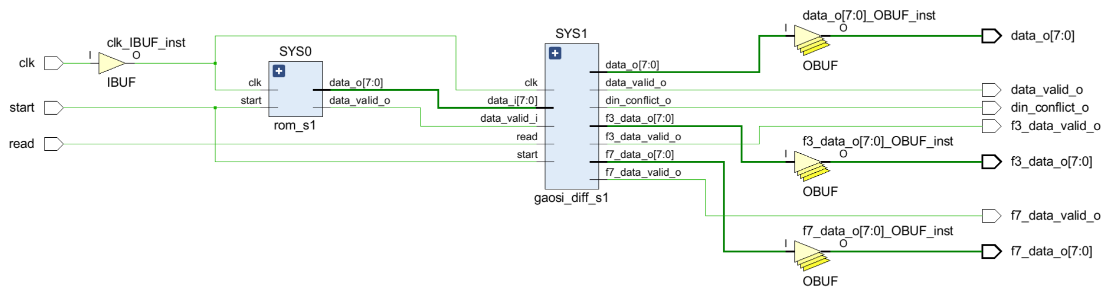
```
            image64(i,j)=dot([image64(i,j+3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        elseif j==254
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j-3)],conv_k7)/64;
        elseif j==255
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j-2),image64(i,j-3)],conv_k7)/64;
        elseif j==256
            image4(i,j) =dot([image4(i,j-1),image4(i,j),image4(i,j-1)],conv_k3)/4;
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j-1),image64(i,j-2),image64(i,j-3)],conv_k7)/64;
        else
            image4(i,j) =dot([image4(i,j-1),image4(i,j),image4(i,j+1)],conv_k3)/4;
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        end
    end
end
image4=image4';%矩阵的转置
image64=image64';%矩阵的转置
%列高斯卷积运算
for i=1:1:256
    for j=1:1:256
        if j==1
            image4(i,j) =dot([image4(i,j+1),image4(i,j),image4(i,j+1)],conv_k3)/4;
            image64(i,j)=dot([image64(i,j+3),image64(i,j+2),image64(i,j+1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        elseif j==2
            image64(i,j)=dot([image64(i,j+3),image64(i,j+2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        elseif j==3
            image64(i,j)=dot([image64(i,j+3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        elseif j==254
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j-3)],conv_k7)/64;
        elseif j==255
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j-2),image64(i,j-3)],conv_k7)/64;
        elseif j==256
            image4(i,j) =dot([image4(i,j-1),image4(i,j),image4(i,j-1)],conv_k3)/4;
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j-1),image64(i,j-2),image64(i,j-3)],conv_k7)/64;
        else
            image4(i,j) =dot([image4(i,j-1),image4(i,j),image4(i,j+1)],conv_k3)/4;
            image64(i,j)=dot([image64(i,j-3),image64(i,j-2),image64(i,j-1),image64(i,j),image64(i,j+1),image64(i,j+2),image64(i,j+3)],conv_k7)/64;
        end
    end
end
image4=image4';
```

```
image64=image64';
%高斯差分实现
for i=1:1:256
    for j=1:1:256
        %image_diff(i,j)=abs(image64(i,j)-image4(i,j));
        image_diff1(i,j)=image4(i,j)-image64(i,j);
    end
end
```

原图像

基于FPGA的Verilog设计

测试系统总体框架



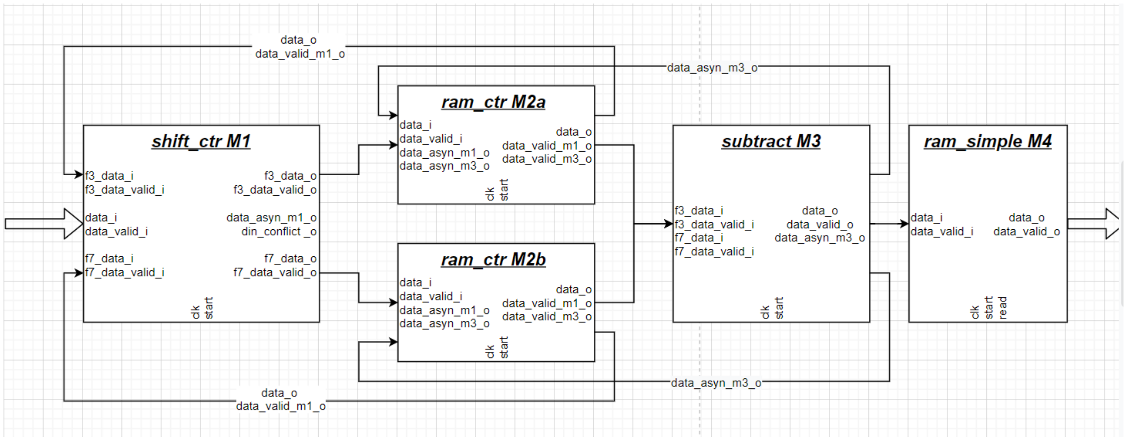
测试系统由存储原图像数据的ROM和高斯差分处理单元构成。其中，f3_data_o,f7_data_o及其数据有效标志信号均用于后续的testbech输出3阶和7阶高斯滤波后的图像数据，用于matlab仿真验证。ROM模块中载入的.coe文件由matlab生成，具体代码如下所示：

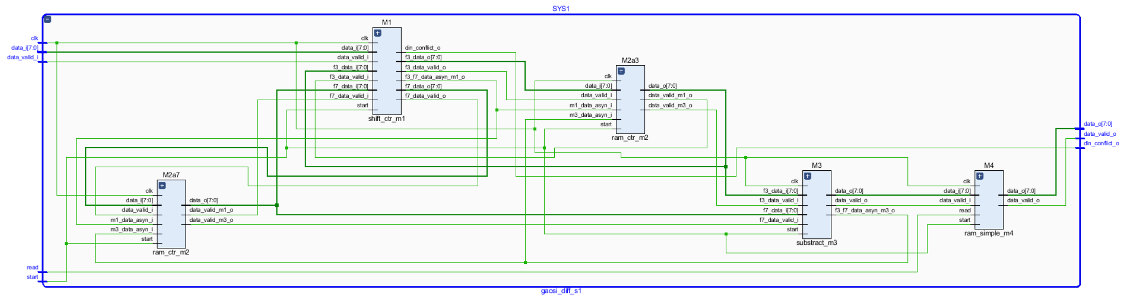
```
clear all
clc
imfinfo('gaosi8.jpg')%图像信息
image=imread('gaosi8.jpg');
width=8;%rom中数据的宽度
depth=256;%rom的深度
fid=fopen('D:\Engineering_Project\Matlab\gaosi\image.coe','w');%写入文件路径%打开一个.coe文件
%存放ROM中的.coe文件第一行字符串，16表示16进制，可以改成其他进制
```

```
fprintf(fid,'memory_initialization_radix=16;\n');
%存放在ROM中的.coe文件第二行字符串
fprintf(fid,'memory_initialization_vector=\n');
for i=1:1:256
   for j=1:1:256
      if (j==256)&&(i==256)
        fprintf(fid,'%x;\n',image(i,j));
     else
       fprintf(fid,'%x,\t',image(i,j));
      end
   end
end
fclose(fid);%关闭文件指针
```

高斯差分模块总体框架

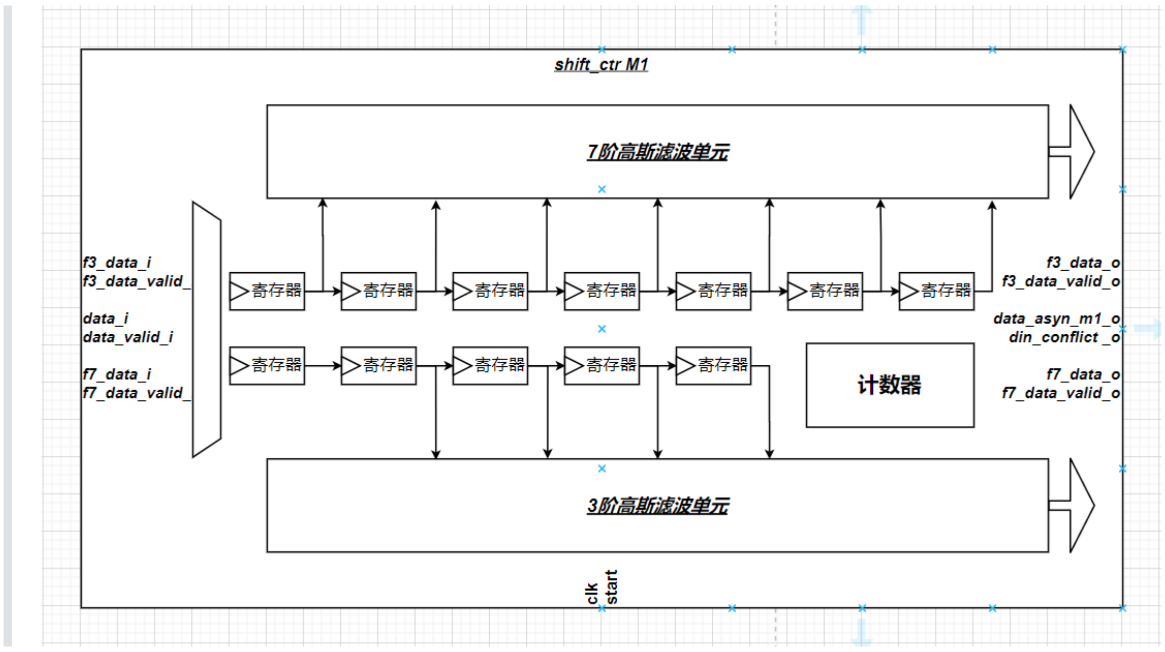
高斯差分模块由shift_ctr_m1，ram_ctr_m2a，ram_ctr_m2b，subtract_m3，ram_simple_m4五个子模块构成，可以划分为四个层次。如下图所示：





shift_ctr_M1子模块

该模块主要并行实现7阶、3阶高斯滤波运算。其结构简图如下所示：



ram_ctr_M2子模块

该模块主要实现双端口RAM的数据写入读出的功能，为了充分利用高斯函数的可分离特性，将二维卷积操作分解成分别对行和列卷积，ram_ctr_M2模块实现两种可选择的读写地址生成方式，以实现对矩阵的转置操作。具体机制可参考如下代码：

```
//写端口==========================================================================
reg     [16:0] m2_wr_pt; //写指针
    wire m2_wr_style;// 0：按行写写入数据，1：按列写入数据
    always @(posedge clk) begin
        if(start) m2_wr_pt <= 17'd0;
        else if(data_valid_i)   m2_wr_pt <= m2_wr_pt + 1'b1;
    end
    assign m2_wr_style = m2_wr_pt[16];
    wire [15:0] m2_wr_addra;//写端口地址
    assign m2_wr_addra = m2_wr_style ? {m2_wr_pt[7:0],m2_wr_pt[15:8]} : m2_wr_pt[15:0];
//==============================================================================
//读端口=========================================================================
reg     [16:0] m2_rd_pt;//读指针
    wire m2_rd_style;// 0：按行写写入数据，1：按列写入数据
    always @(posedge clk) begin
        if(m2_rd_start)  m2_rd_pt <= {m2_wr_pt[16],16'b0000_0000_0000_0000};
        else if(m2_rd_enb) m2_rd_pt <= m2_rd_pt + 1'b1;

    end
    assign m2_rd_finish = &m2_rd_pt[15:0];
    wire [15:0] m2_rd_addrb;//读端口地址
    assign m2_rd_addrb = m2_rd_style ? {m2_rd_pt[7:0],m2_rd_pt[15:8]} : m2_rd_pt[15:0];
    assign m2_rd_style = m2_rd_pt[16];
//==============================================================================
```

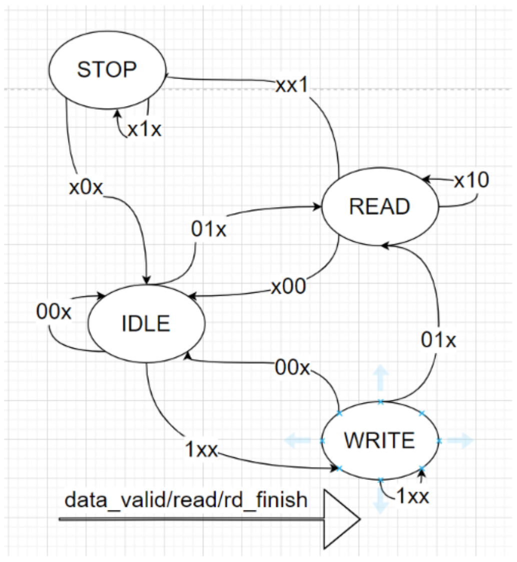
subtract_M3子模块

该模块主要实现高斯差分操作，根据mtalab的软件运算结果，为了更好地显示高斯差分后的图像，将3阶高斯滤波与7阶高斯滤波的差值平移至8位数据的中值进行观察，即差值结果加128，以保证图像边沿检测的极大值和极小值都能在数据结果中体现。

```
//实现减法运算：f3_data-f7_data+128
//将负数变成补码形式
    wire    [8:0]   m3_f7_data_copl;
    assign m3_f7_data_copl = {1'b1,(~f7_data_i)} + 1'b1;
    wire    [8:0]   m3_data_diff_copl;
    assign m3_data_diff_copl = {1'b0,m3_f3_data_i} + m3_f7_data_copl;
//根据软件仿真结果预分析，为了更加明显的观测到图像边缘，将计算结果+128;该操作在操作数
f3_data_i上提前进行
```

ram_simple_M4子模块

该模块实现了一个简单单端口RAM的读写操作，以存储运算结果。操作模式变换以一个有限状态机实现。如下图所示。下图给出了状态转移图和实现部分状态信号的生成的任务。



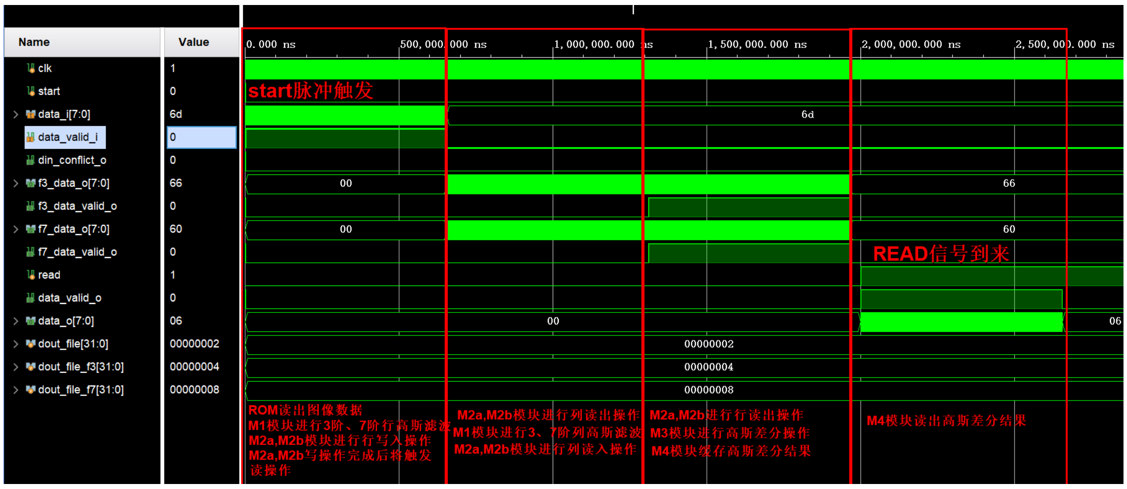
```
reg m4_ram_ena,m4_ram_wren;
    task IDLE_out;
        begin
            m4_ram_ena = 1'b0;
            m4_ram_wren= 1'b0;
        end
    endtask
    task WR_out;
        begin
            m4_ram_ena = 1'b1;
            m4_ram_wren= 1'b1;
        end
    endtask
    task RD_out;
        begin
            m4_ram_ena = 1'b1;
            m4_ram_wren= 1'b0;
        end
    endtask
```

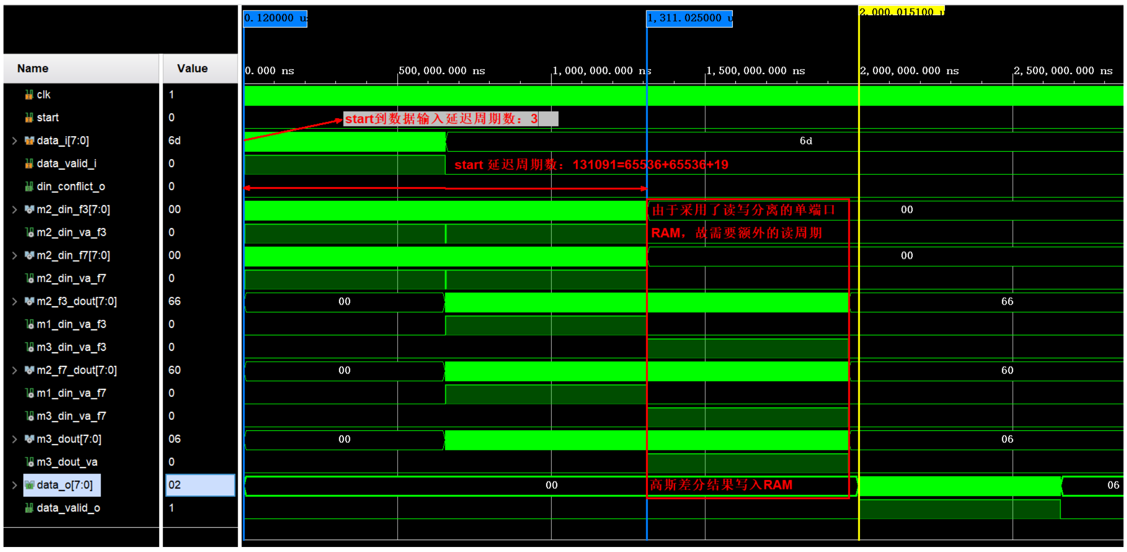
行为级仿真

在Vivado集成开发环境中添加仿真文件testbench_sys.v，并进行行为级仿真，同时将数据结果导出，用于matlab进行验证。

```
integer dout_file;
initial begin
dout_file=$fopen("C:/Users/Tingf/Desktop/Vivado_Project/Gaosi_diff_filter/image.txt");
     if(dout_file == 0)begin
        $display ("can not open the file!");    //创建文件失败，显示can not open the file!
        $stop;
     end
end
always @(posedge clk)
     if(data_valid_o)   //使能
       $fdisplay(dout_file,"%d",data_o);    //使能信号有效，每来一个时钟，写入到所创建的文件中
```

下图显示了高斯差分模块各个子模块在系统运行下的各个阶段的功能状态。



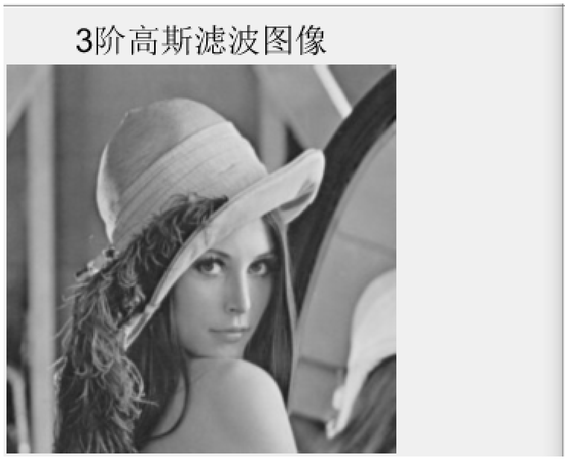


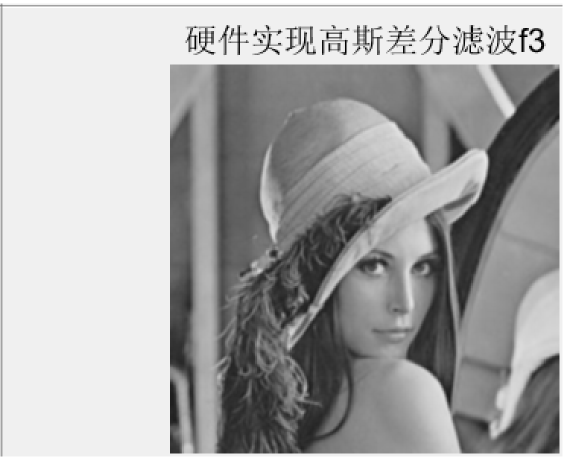
Matlab验证

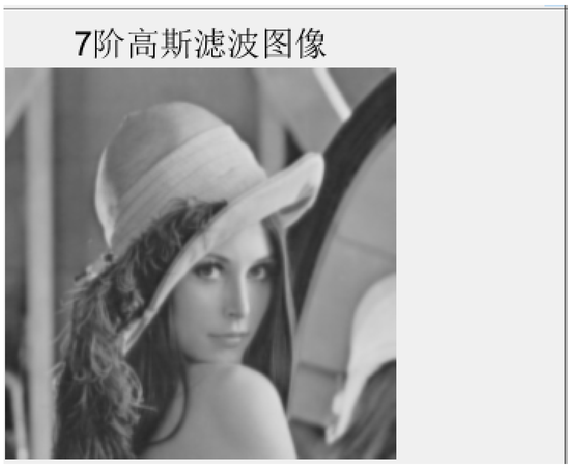
将仿真的数据结果导入Matlab，进行必要的数据格式处理，以图像的方式显示并与matlab运算结果对照。

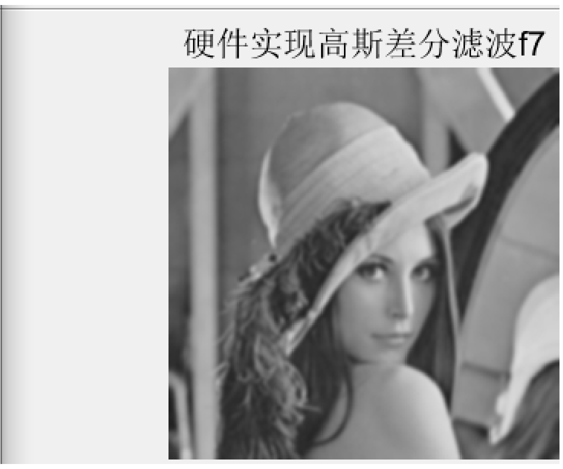
```
%matlab实现结果的图像
image4_o=uint8(image4);
figure,imshow(image4_o),title('3阶高斯滤波图像');
image64_o=uint8(image64);
figure,imshow(image64_o),title('7阶高斯滤波图像');
```

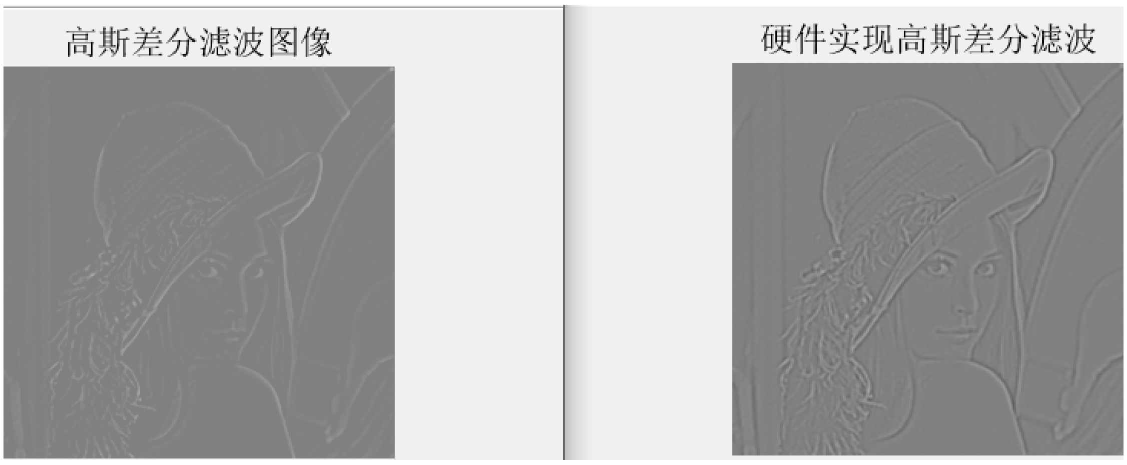
```matlab

image_diff1_o=uint8(image_diff1)+uint8(ones(256)*128);
figure,imshow(image_diff1_o),title('高斯差分滤波图像');
%硬件实现结果
imageH=load('C:\Users\Tingf\Desktop\Vivado_Project\Gaosi_diff_filter\image.txt','%s');
imageH8=uint8(imageH);
imageH8=reshape(imageH8,256,256);
imageH8=imageH8';
figure,imshow(imageH8),title('硬件实现高斯差分滤波');

imageH_f3=load('C:\Users\Tingf\Desktop\Vivado_Project\Gaosi_diff_filter\image_f3.txt','%s');
imageH8_f3=uint8(imageH_f3);
imageH8_f3=reshape(imageH8_f3,256,256);
imageH8_f3=imageH8_f3';
figure,imshow(imageH8_f3),title('硬件实现高斯差分滤波f3');

imageH_f7=load('C:\Users\Tingf\Desktop\Vivado_Project\Gaosi_diff_filter\image_f7.txt','%s');
imageH8_f7=uint8(imageH_f7);
imageH8_f7=reshape(imageH8_f7,256,256);
imageH8_f7=imageH8_f7';
figure,imshow(imageH8_f7),title('硬件实现高斯差分滤波f7');
```









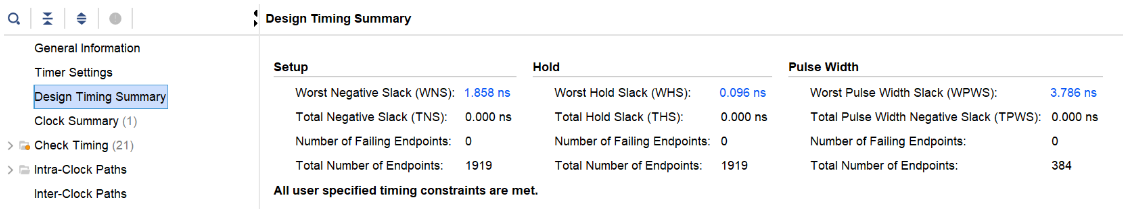


图像结果对照显示：本文的高斯差分图像处理模块的verilog HDL实现比较好地实现了该算法，误差在允许范围内。

综合实现

在Vivado集成开发环境中添加时序约束文件，并实现综合。

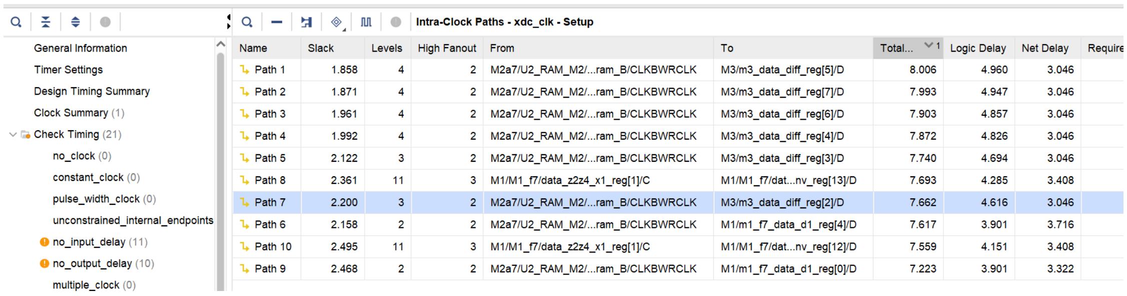
```
create_clock -period 10.000 -name xdc_clk -waveform {0.000 5.000} [get_ports clk]
```



Design Timing Summary

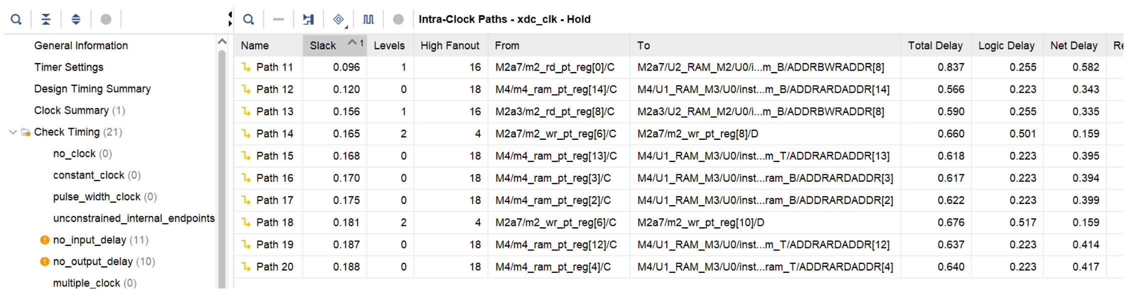
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.858 ns	Worst Hold Slack (WHS):	0.096 ns	Worst Pulse Width Slack (WPWS):	3.786 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1919	Total Number of Endpoints:	1919	Total Number of Endpoints:	384

All user specified timing constraints are met.



Intra-Clock Paths - xdc_clk - Setup

Name	Slack	Levels	High Fanout	From	To	Total...	Logic Delay	Net Delay	Require
Path 1	1.858	4	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M3/m3_data_diff_reg[5]/D	8.006	4.960	3.046	
Path 2	1.871	4	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M3/m3_data_diff_reg[7]/D	7.993	4.947	3.046	
Path 3	1.961	4	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M3/m3_data_diff_reg[6]/D	7.903	4.857	3.046	
Path 4	1.992	4	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M3/m3_data_diff_reg[4]/D	7.872	4.826	3.046	
Path 5	2.122	3	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M3/m3_data_diff_reg[3]/D	7.740	4.694	3.046	
Path 8	2.361	11	3	M1/M1_f7/data_z2z4_x1_reg[1]/C	M1/M1_f7/dat...nv_reg[13]/D	7.693	4.285	3.408	
Path 7	2.200	3	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M3/m3_data_diff_reg[2]/D	7.662	4.616	3.046	
Path 6	2.158	2	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M1/m1_f7_data_d1_reg[4]/D	7.617	3.901	3.716	
Path 10	2.495	11	3	M1/M1_f7/data_z2z4_x1_reg[1]/C	M1/M1_f7/dat...nv_reg[12]/D	7.559	4.151	3.408	
Path 9	2.468	2	2	M2a7/U2_RAM_M2/...ram_B/CLKBWRCLK	M1/m1_f7_data_d1_reg[0]/D	7.223	3.901	3.322	

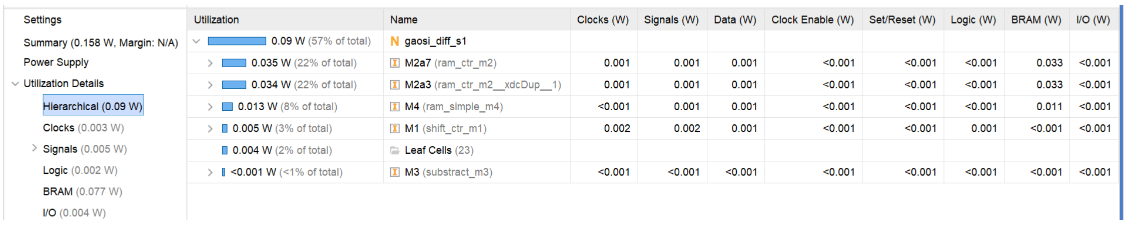


Intra-Clock Paths - xdc_clk - Hold

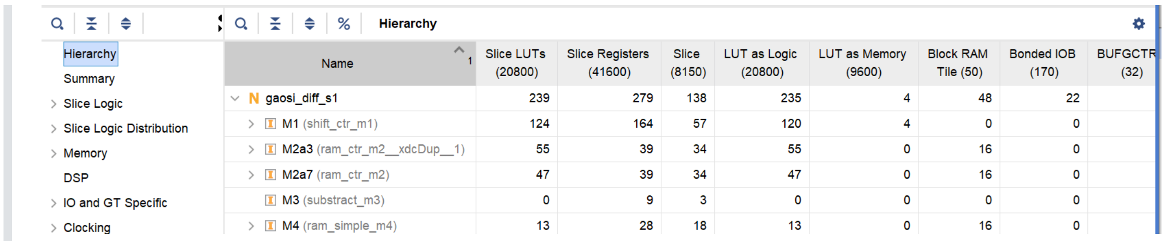
Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Re
Path 11	0.096	1	16	M2a7/m2_rd_pt_reg[0]/C	M2a7/U2_RAM_M2/U0/i...m_B/ADDRBWRADDR[8]	0.837	0.255	0.582	
Path 12	0.120	0	18	M4/m4_ram_pt_reg[14]/C	M4/U1_RAM_M3/U0/inst...m_B/ADDRARDADDR[14]	0.566	0.223	0.343	
Path 13	0.156	1	16	M2a3/m2_rd_pt_reg[8]/C	M2a3/U2_RAM_M2/U0/i...m_B/ADDRBWRADDR[8]	0.590	0.255	0.335	
Path 14	0.165	2	4	M2a7/m2_wr_pt_reg[6]/C	M2a7/m2_wr_pt_reg[8]/D	0.660	0.501	0.159	
Path 15	0.168	0	18	M4/m4_ram_pt_reg[13]/C	M4/U1_RAM_M3/U0/inst...m_T/ADDRARDADDR[13]	0.618	0.223	0.395	
Path 16	0.170	0	18	M4/m4_ram_pt_reg[3]/C	M4/U1_RAM_M3/U0/inst...ram_B/ADDRARDADDR[3]	0.617	0.223	0.394	
Path 17	0.175	0	18	M4/m4_ram_pt_reg[2]/C	M4/U1_RAM_M3/U0/inst...ram_B/ADDRARDADDR[2]	0.622	0.223	0.399	
Path 18	0.181	2	4	M2a7/m2_wr_pt_reg[6]/C	M2a7/m2_wr_pt_reg[10]/D	0.676	0.517	0.159	
Path 19	0.187	0	18	M4/m4_ram_pt_reg[12]/C	M4/U1_RAM_M3/U0/inst...m_T/ADDRARDADDR[12]	0.637	0.223	0.414	
Path 20	0.188	0	18	M4/m4_ram_pt_reg[4]/C	M4/U1_RAM_M3/U0/inst...ram_T/ADDRARDADDR[4]	0.640	0.223	0.417	

由上图可知，由于大量使用Block RAM 路径延迟较大，有优化空间。设计的建立时间和保持时间都能满足，且有一定冗余，可以保证设计一定的工作可靠性。

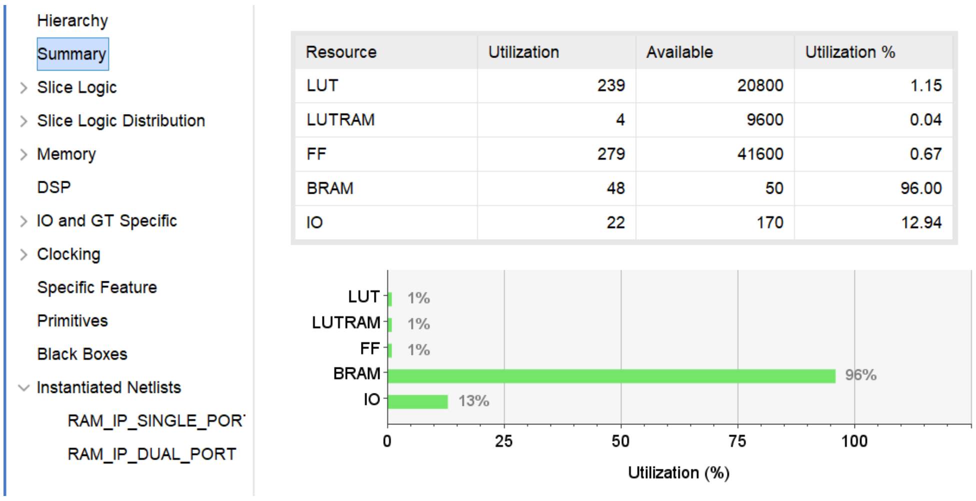
能耗及资源占用情况



Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	I/O (W)
0.09 W (57% of total)	gaosi_diff_s1								
0.035 W (22% of total)	M2a7 (ram_ctr_m2)	0.001	0.001	0.001	<0.001	<0.001	<0.001	0.033	<0.001
0.034 W (22% of total)	M2a3 (ram_ctr_m2__xdcDup__1)	0.001	0.001	0.001	<0.001	<0.001	<0.001	0.033	<0.001
0.013 W (8% of total)	M4 (ram_simple_m4)	<0.001	0.001	0.001	<0.001	<0.001	<0.001	0.011	<0.001
0.005 W (3% of total)	M1 (shift_ctr_m1)	0.002	0.002	0.001	<0.001	<0.001	0.001	<0.001	<0.001
0.004 W (2% of total)	Leaf Cells (23)								
<0.001 W (<1% of total)	M3 (substract_m3)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001



Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IOB (170)	BUFGCTR (32)
gaosi_diff_s1	239	279	138	235	4	48	22	
M1 (shift_ctr_m1)	124	164	57	120	4	0	0	
M2a3 (ram_ctr_m2__xdcDup__1)	55	39	34	55	0	16	0	
M2a7 (ram_ctr_m2)	47	39	34	47	0	16	0	
M3 (substract_m3)	0	9	3	0	0	0	0	
M4 (ram_simple_m4)	13	28	18	13	0	16	0	



Resource	Utilization	Available	Utilization %
LUT	239	20800	1.15
LUTRAM	4	9600	0.04
FF	279	41600	0.67
BRAM	48	50	96.00
IO	22	170	12.94

故最高工作频率为100MHz，数据吞吐率为800bps。

附录

源代码

```
//===========================================================================
=======
module filter3_unit(
    input [7:0] data_z0_i,
    input [7:0] data_z1_i,
    input [7:0] data_z2_i,
    input [0:2] valid_i,

    output [7:0] data_conv_o,
    output       data_valid_o,
    input clk
);
    //数据有效选择信号
    wire [7:0] data_z0;
    wire [7:0] data_z2;
    wire data_valid;
    assign data_z0 = valid_i[0] ? data_z0_i : data_z2_i;
    assign data_z2 = valid_i[2] ? data_z2_i : data_z0_i;
    assign data_valid = valid_i[1];

    reg [8:0] data_z0z2;
    reg [8:0] data_z1z1;
    always @(posedge clk) begin
        if(!data_valid)  begin
            data_z0z2 <= 9'b0;
            data_z1z1 <= 9'b0;
        end
        else begin
            data_z0z2 <= data_z0 + data_z2;
            data_z1z1 <= {data_z1_i,1'b0};
```

```verilog
        end
    end

    reg data_valid_d1,data_valid_d2;
    reg [9:0] data_conv;
    always @(posedge clk) begin
        data_conv       <= data_z0z2 + data_z1z1;
        data_valid_d1   <= data_valid;
        data_valid_d2   <= data_valid_d1;
    end
    assign data_conv_o  = data_conv[9:2];
    assign data_valid_o = data_valid_d2;

endmodule
//==========================================================================
========
//==========================================================================
=======
module filter7_unit(
    input [7:0] data_z0_i,
    input [7:0] data_z1_i,
    input [7:0] data_z2_i,
    input [7:0] data_z3_i,
    input [7:0] data_z4_i,
    input [7:0] data_z5_i,
    input [7:0] data_z6_i,
    input [0:6] valid_i,

    output [7:0] data_conv_o,
    output       data_valid_o,
    input clk
);
    //数据有效选择信号
    wire [7:0] data_z0;
    wire [7:0] data_z1;
    wire [7:0] data_z2;
    wire [7:0] data_z4;
    wire [7:0] data_z5;
    wire [7:0] data_z6;
    wire data_valid;
    assign data_z0 = valid_i[0] ? data_z0_i : data_z6_i;
    assign data_z1 = valid_i[1] ? data_z1_i : data_z5_i;
    assign data_z2 = valid_i[2] ? data_z2_i : data_z4_i;
    assign data_z4 = valid_i[4] ? data_z4_i : data_z2_i;
    assign data_z5 = valid_i[5] ? data_z5_i : data_z1_i;
    assign data_z6 = valid_i[6] ? data_z6_i : data_z0_i;
    assign data_valid = valid_i[3];

    reg [8:0] data_z0z6_x1;
    reg [8:0] data_z1z5_x2;
    reg [8:0] data_z1z5_x4;
    reg [8:0] data_z2z4_x1;
    reg [8:0] data_z2z4_x2;
    reg [8:0] data_z2z4_x4;
    reg [8:0] data_z2z4_x8;
    reg [10:0] data_z3z3_x4;
    always @(posedge clk) begin
        if(!data_valid)  begin
```

```
            data_z0z6_x1  <= 9'b0;
            data_z1z5_x2  <= 9'b0;
            data_z1z5_x4  <= 9'b0;
            data_z2z4_x1  <= 9'b0;
            data_z2z4_x2  <= 9'b0;
            data_z2z4_x4  <= 9'b0;
            data_z2z4_x8  <= 9'b0;
            data_z3z3_x4  <= 11'b0;
        end
        else    begin
            data_z0z6_x1  <= data_z0 + data_z6;
            data_z1z5_x2  <= data_z1 + data_z5;
            data_z1z5_x4  <= data_z1 + data_z5;
            data_z2z4_x1  <= data_z2 + data_z4;
            data_z2z4_x2  <= data_z2 + data_z4;
            data_z2z4_x4  <= data_z2 + data_z4;
            data_z2z4_x8  <= data_z2 + data_z4;
            data_z3z3_x4  <= {data_z3_i,2'b0} + data_z3_i ;
        end
    end

    wire [13:0] data_z0z6_z1z5_z3z3;
    assign data_z0z6_z1z5_z3z3 = ({data_z1z5_x4,2'b0} + {data_z1z5_x2,1'b0})  +  ({data_z3z3_x4,2'b0} + data_z0z6_x1);
    wire [12:0] data_z2z4;
    assign data_z2z4 = ({data_z2z4_x2,1'b0} + data_z2z4_x1) + ({data_z2z4_x8,3'b0} + {data_z2z4_x4,2'b0});

    reg data_valid_d1,data_valid_d2;
    reg [13:0] data_conv;
    always @(posedge clk) begin
        data_conv       <= data_z0z6_z1z5_z3z3 + data_z2z4;
        data_valid_d1   <= data_valid;
        data_valid_d2   <= data_valid_d1;
    end
    assign data_conv_o  = data_conv[13:6];
    assign data_valid_o = data_valid_d2;

endmodule
//==============================================================================
//===============================================================================
module gaosi_diff_s1(
    input clk,
    input start,
    input read,
    input   [7:0] data_i,
    input         data_valid_i,
    output  [7:0] data_o,
    output        data_valid_o,
    /*
    output  [7:0] f3_data_o,
    output        f3_data_valid_o,
    output  [7:0] f7_data_o,
    output        f7_data_valid_o,
    */
    output        din_conflict_o
```

```
);

    wire [7:0]  m2_din_f3,m2_din_f7;
    wire        m2_din_va_f3,m2_din_va_f7;
    wire [7:0]  m2_f3_dout,m2_f7_dout;
    wire        m1_din_va_f3,m1_din_va_f7;
    wire        m1_data_asyn,m3_data_asyn;
shift_ctr_m1 M1(
    .clk(clk),
    .start(start),
    .data_i(data_i),
    .data_valid_i(data_valid_i),
    .f3_data_i(m2_f3_dout),
    .f3_data_valid_i(m1_din_va_f3),
    .f7_data_i(m2_f7_dout),
    .f7_data_valid_i(m1_din_va_f7),

    .f3_data_o(m2_din_f3),
    .f3_data_valid_o(m2_din_va_f3),
    .f7_data_o(m2_din_f7),
    .f7_data_valid_o(m2_din_va_f7),

    .f3_f7_data_asyn_m1_o(m1_data_asyn),
    .din_conflict_o(din_conflict_o)
);
    wire m3_din_va_f3,m3_din_va_f7;
ram_ctr_m2 M2a3(
    .clk(clk),
    .start(start),
    .data_i(m2_din_f3),
    .data_valid_i(m2_din_va_f3),
    .m1_data_asyn_i(m1_data_asyn),
    .m3_data_asyn_i(m3_data_asyn),

    .data_o(m2_f3_dout),
    .data_valid_m1_o(m1_din_va_f3),
    .data_valid_m3_o(m3_din_va_f3)
);

ram_ctr_m2 M2a7(
    .clk(clk),
    .start(start),
    .data_i(m2_din_f7),
    .data_valid_i(m2_din_va_f7),
    .m1_data_asyn_i(m1_data_asyn),
    .m3_data_asyn_i(m3_data_asyn),

    .data_o(m2_f7_dout),
    .data_valid_m1_o(m1_din_va_f7),
    .data_valid_m3_o(m3_din_va_f7)
);

    wire [7:0]  m3_dout;
    wire        m3_dout_va;

substract_m3 M3(
    .clk(clk),
    .start(start),
    .f3_data_i(m2_f3_dout),
    .f3_data_valid_i(m3_din_va_f3),
    .f7_data_i(m2_f7_dout),
    .f7_data_valid_i(m3_din_va_f7),

    .data_o(m3_dout),
    .data_valid_o(m3_dout_va),
    .f3_f7_data_asyn_m3_o(m3_data_asyn)
);

ram_simple_m4 M4(
    .clk(clk),
    .start(start),
    .read(read),
    .data_i(m3_dout),
    .data_valid_i(m3_dout_va),
    .data_o(data_o),
    .data_valid_o(data_valid_o)
);
/*
    assign f3_data_o = m2_f3_dout;
    assign f3_data_valid_o = m3_din_va_f3;
    assign f7_data_o = m2_f7_dout;
    assign f7_data_valid_o = m3_din_va_f7;
*/
endmodule
//===============================================================================
//=============================================================================
module ram_ctr_m2(
    input clk,
    input start,
    input   [7:0]   data_i,
    input           data_valid_i,
    input           m1_data_asyn_i,
    input           m3_data_asyn_i,

    output  [7:0]   data_o,
    output          data_valid_m1_o,
    output          data_valid_m3_o
);
//======================写端口控制=====================================
    reg     [16:0] m2_wr_pt; //写指针
    wire m2_wr_style;// 0：按行写写入数据，1：按列写入数据
    always @(posedge clk) begin
        if(start) m2_wr_pt <= 17'd0;
        else if(data_valid_i)   m2_wr_pt <= m2_wr_pt + 1'b1;
    end
    assign m2_wr_style = m2_wr_pt[16];
    wire [15:0] m2_wr_addra;//写端口地址
    assign m2_wr_addra = m2_wr_style ? {m2_wr_pt[7:0],m2_wr_pt[15:8]} : m2_wr_pt[15:0];
    wire m2_wea,m2_ena;
    assign m2_wea = data_valid_i;
    assign m2_ena = data_valid_i;

    reg m2_rd_start;//读端口开始脉冲信号;读端口完成或者检测到两个RAM输出不同步启动
    always @(posedge clk) begin
        m2_rd_start <= (&m2_wr_pt[15:0]) || (m1_data_asyn_i |
m3_data_asyn_i);
    end

//==========================================================================

//=====================读端口控制========================================
    reg m2_rd_enb;//读端口使能
    wire m2_rd_finish;
    always @(posedge clk) begin
        if(start)    m2_rd_enb <= 1'b0;
        else if(m2_rd_start) m2_rd_enb <= 1'b1;
        else if(m2_rd_finish) m2_rd_enb <= 1'b0;
    end
    reg     [16:0] m2_rd_pt;//读指针
    wire m2_rd_style;// 0: 按行写写入数据, 1: 按列写入数据
    always @(posedge clk) begin
        if(m2_rd_start)  m2_rd_pt <=
{m2_wr_pt[16],16'b0000_0000_0000_0000};
        else if(m2_rd_enb) m2_rd_pt <= m2_rd_pt + 1'b1;

    end
    assign m2_rd_finish = &m2_rd_pt[15:0];
    wire [15:0] m2_rd_addrb;//读端口地址
    assign m2_rd_addrb = m2_rd_style ? {m2_rd_pt[7:0],m2_rd_pt[15:8]} :
m2_rd_pt[15:0];
    assign m2_rd_style = m2_rd_pt[16];

    wire m2_datapath_ctr;//数据传输方向控制信号: 0: 无效;1:有效
    assign m2_datapath_ctr = m2_rd_pt[16];
    reg m2_rd_valid_m1;//读出数据有效标志信号
    reg m2_rd_valid_m3;//读出数据有效标志信号
    always @(posedge clk) begin
        if(!m2_rd_enb)    begin
            m2_rd_valid_m1 <= 1'b0;
            m2_rd_valid_m3 <= 1'b0;
        end
        else begin
            m2_rd_valid_m1 <= m2_datapath_ctr;
            m2_rd_valid_m3 <= ~m2_datapath_ctr;
        end
    end
    assign data_valid_m1_o = m2_rd_valid_m1;
    assign data_valid_m3_o = m2_rd_valid_m3;


//============================================================================
==

//简单双端口RAM例化,位宽8bit,深度65536,前后均无额外寄存器,读写阶延迟一时钟周期
RAM_IP_DUAL_PORT U2_RAM_M2(
    .clka(clk),
    .ena(m2_ena),
    .wea(m2_wea),
    .addra(m2_wr_addra),
    .dina (data_i),
    .clkb (clk),
    .enb(m2_rd_enb),
    .addrb(m2_rd_addrb),
    .doutb(data_o)
);

endmodule
//==========================================================================
=========
//==========================================================================
======
module ram_simple_m4(
    input clk,
    input start,
    input read,
    input   [7:0]   data_i,
    input           data_valid_i,
    output  [7:0]   data_o,
    output          data_valid_o
);

    localparam IDLE_STATE   = 2'b00;
    localparam WR_STATE     = 2'b01;
    localparam RD_STATE     = 2'b10;
    localparam RD_STOP      = 2'b11;

    reg [1:0] CS,NS;
    always @(posedge clk) begin
        if(start)   CS    <= IDLE_STATE;
        else        CS    <= NS;
    end
    reg [15:0] m4_ram_pt;//ram指针
    always @(posedge clk) begin
        if(~^CS) m4_ram_pt <= 16'd0;
        else     m4_ram_pt <= m4_ram_pt +1'b1;
    end
    wire m4_pt_count_finish;
    assign m4_pt_count_finish = &m4_ram_pt;

    always @(*) begin
        case(CS)
            IDLE_STATE: begin
                IDLE_out;
                if(data_valid_i)    NS = WR_STATE;
                else if( read)      NS = RD_STATE;
                else                NS = IDLE_STATE;
            end
            WR_STATE:   begin
                WR_out;
                if(data_valid_i)    NS = WR_STATE;
                else if(read)       NS = RD_STATE;
                else                NS = IDLE_STATE;
            end
            RD_STATE:   begin
                RD_out;
                if(m4_pt_count_finish) NS = RD_STOP;
                else if(read)          NS = RD_STATE;
                else  NS = IDLE_STATE;
            end
```

```verilog
            RD_STOP :   begin
                IDLE_out;
                if(read)            NS = RD_STOP;
                else                NS = IDLE_STATE;
            end
        endcase
    end
    reg m4_ram_ena,m4_ram_wren;
    task IDLE_out;
        begin
          m4_ram_ena = 1'b0;
          m4_ram_wren= 1'b0;
        end
    endtask
    task WR_out;
        begin
          m4_ram_ena = 1'b1;
          m4_ram_wren= 1'b1;
        end
    endtask
    task RD_out;
        begin
          m4_ram_ena = 1'b1;
          m4_ram_wren= 1'b0;
        end
    endtask
    //输入数据锁存一时钟周期
    reg [7:0] m4_data;
    always @(posedge clk) begin
        m4_data <= data_i;
    end
    //输出有效信号
    reg m4_data_valid;
    always @(posedge clk) begin
        if(CS==RD_STATE) m4_data_valid <= 1'b1;
        else m4_data_valid <= 1'b0;
    end
    assign data_valid_o = m4_data_valid;
RAM_IP_SINGLE_PORT U1_RAM_M3(
    .clka(clk),
    .ena(m4_ram_ena),
    .wea(m4_ram_wren),
    .addra(m4_ram_pt),
    .dina(m4_data),
    .douta(data_o)
);

endmodule
//============================================================================
========
//============================================================================
======
module shift_ctr_m1(
    input clk,
    input start,
    input   [7:0]   data_i,
    input           data_valid_i,
    input   [7:0]   f3_data_i,
    input           f3_data_valid_i,
    input   [7:0]   f7_data_i,
    input           f7_data_valid_i,

    output  [7:0]   f3_data_o,
    output          f3_data_valid_o,
    output  [7:0]   f7_data_o,
    output          f7_data_valid_o,

    output          f3_f7_data_asyn_m1_o,
    output          din_conflict_o
);

//输入数据有效
    assign f3_f7_data_asyn_m1_o = f3_data_valid_i ^ f7_data_valid_i;
    wire m1_data_valid;
    assign m1_data_valid = (f3_data_valid_i && f7_data_valid_i) || data_valid_i;
    assign din_conflict_o = data_valid_i && (f3_data_valid_i || f7_data_valid_i);
//输入数据选择
    wire [7:0]  m1_f3_data;
    assign  m1_f3_data = f3_data_valid_i ? f3_data_i : (data_valid_i ? data_i : 8'd0);
    wire [7:0]  m1_f7_data;
    assign  m1_f7_data = f7_data_valid_i ? f7_data_i : (data_valid_i ? data_i : 8'd0);

//三阶移位控制
    reg [7:0] m1_f3_data_z1,m1_f3_data_z0,m1_f3_data_d1,m1_f3_data_d2,m1_f3_data_d3;
    always @(posedge clk) begin
        m1_f3_data_z1 <= m1_f3_data;
        m1_f3_data_z0 <= m1_f3_data_z1;
        m1_f3_data_d1 <= m1_f3_data_z0;
        m1_f3_data_d2 <= m1_f3_data_d1;
        m1_f3_data_d3 <= m1_f3_data_d2;
    end
//七阶移位
    reg [7:0] m1_f7_data_d1,m1_f7_data_d2,m1_f7_data_d3,m1_f7_data_d4,m1_f7_data_d5,m1_f7_data_d6,m1_f7_data_d7;
    always @(posedge clk) begin
        m1_f7_data_d1 <= m1_f7_data;
        m1_f7_data_d2 <= m1_f7_data_d1;
        m1_f7_data_d3 <= m1_f7_data_d2;
        m1_f7_data_d4 <= m1_f7_data_d3;
        m1_f7_data_d5 <= m1_f7_data_d4;
        m1_f7_data_d6 <= m1_f7_data_d5;
        m1_f7_data_d7 <= m1_f7_data_d6;
    end
//输出数据有效信号及计数标志信号
    wire m1_slice_start;
    reg m1_data_valid_d1,m1_data_valid_d2,m1_data_valid_d3,m1_data_valid_d4;
    always @(posedge clk) begin
        if(start) m1_data_valid_d1 <= 1'b0;
        else if(m1_data_valid) m1_data_valid_d1 <= 1'b1;
        else m1_data_valid_d1 <= 1'b0;
    end
    always @(posedge clk) begin
        if(start) begin
          m1_data_valid_d2 <= 1'b0;
          m1_data_valid_d3 <= 1'b0;
          m1_data_valid_d4 <= 1'b0;
        end
        else begin
          m1_data_valid_d2 <= m1_data_valid_d1;
          m1_data_valid_d3 <= m1_data_valid_d2;
          m1_data_valid_d4 <= m1_data_valid_d3;
        end
    end

    assign m1_slice_start = m1_data_valid_d4;

    reg [7:0] m1_slice_pt;//行或列指针：用于计算单元的数据处理
    always @(posedge clk) begin
        if(!m1_slice_start)   m1_slice_pt <= 8'd0;
        else  m1_slice_pt <= m1_slice_pt +1'b1;
    end
//3阶和7阶高斯滤波输入数据有效
    wire [0:2] m1_f3_data_valid;
    wire [0:6] m1_f7_data_valid;
    assign m1_f3_data_valid[0] = ~(&m1_slice_pt);
    assign m1_f3_data_valid[1] = m1_data_valid_d4;
    assign m1_f3_data_valid[2] = |m1_slice_pt;

    assign m1_f7_data_valid[0] = ~((&m1_slice_pt[7:2]) && 
m1_slice_pt[1:0]);
    assign m1_f7_data_valid[1] = ~(&m1_slice_pt[7:1]);
    assign m1_f7_data_valid[2] = ~(&m1_slice_pt);
    assign m1_f7_data_valid[3] = m1_data_valid_d4;

    assign m1_f7_data_valid[4] = |m1_slice_pt;
    assign m1_f7_data_valid[5] = |m1_slice_pt[7:1];
    assign m1_f7_data_valid[6] = (|m1_slice_pt[7:2]) || 
(&m1_slice_pt[1:0]);

//计算单元例化
filter3_unit M1_f3(
    .data_z0_i(m1_f3_data_d1),
    .data_z1_i(m1_f3_data_d2),
    .data_z2_i(m1_f3_data_d3),
    .valid_i(m1_f3_data_valid),

    .data_conv_o(f3_data_o),
    .data_valid_o(f3_data_valid_o),
    .clk(clk)
);

filter7_unit M1_f7(
    .data_z0_i(m1_f7_data_d1),
    .data_z1_i(m1_f7_data_d2),
    .data_z2_i(m1_f7_data_d3),
    .data_z3_i(m1_f7_data_d4),
    .data_z4_i(m1_f7_data_d5),
    .data_z5_i(m1_f7_data_d6),
    .data_z6_i(m1_f7_data_d7),
    .valid_i(m1_f7_data_valid),

    .data_conv_o(f7_data_o),
    .data_valid_o(f7_data_valid_o),
    .clk(clk)
);

endmodule
//============================================================================
========
//============================================================================
========
module substract_m3(
    input clk,
    input start,
    input   [7:0]   f3_data_i,
    input           f3_data_valid_i,
    input   [7:0]   f7_data_i,
    input           f7_data_valid_i,

    output  [7:0]   data_o,
    output          data_valid_o,
    output          f3_f7_data_asyn_m3_o
);
    assign f3_f7_data_asyn_m3_o = f3_data_valid_i ^ f7_data_valid_i;
    reg m3_data_valid;
    always @(posedge clk) begin
        if(start)   m3_data_valid <= 1'b0;
        else    m3_data_valid <= f3_data_valid_i && f7_data_valid_i;
    end
    assign data_valid_o = m3_data_valid;


//实现减法运算：f3_data-f7_data+128
//将负数变成补码形式
    wire    [8:0]   m3_f3_data_plus;
    assign m3_f3_data_plus[8:7] = f3_data_i[7] + 1'b1;
    assign m3_f3_data_plus[6:0] = f3_data_i[6:0];
    wire    [8:0]   m3_f7_data_copl;
    assign m3_f7_data_copl = {1'b1,(~f7_data_i)} + 1'b1;
    wire    [8:0]   m3_data_diff_copl;
    assign m3_data_diff_copl = m3_f3_data_plus + m3_f7_data_copl;
    reg     [7:0]   m3_data_diff;
    always @(posedge clk) begin
        m3_data_diff <= m3_data_diff_copl[7:0];
    end
//根据软件仿真结果预分析，为了更加明显的观测到图像边缘，将计算结果+128;该操作在操作数
f3_data_i上提前进行
    assign data_o = m3_data_diff;

endmodule
//============================================================================
========
//============================================================================
=======
module test_SYS(
    input clk,
    input start,
    input read,
    output [7:0] data_o,
    output data_valid_o,
    /*
    output  [7:0] f3_data_o,
    output        f3_data_valid_o,
    output  [7:0] f7_data_o,
    output        f7_data_valid_o,
    */
    output din_conflict_o
);

    wire [7:0] sys0_data;
    wire sys0_data_valid;
gaosi_diff_s1 SYS1(
    .clk(clk),
    .start(start),
    .read(read),
    .data_i(sys0_data),
    .data_valid_i(sys0_data_valid),
    .data_o(data_o),
    .data_valid_o(data_valid_o),
    /*
    .f3_data_o(f3_data_o),
    .f3_data_valid_o(f3_data_valid_o),
    .f7_data_o(f7_data_o),
    .f7_data_valid_o(f7_data_valid_o),
    */
    .din_conflict_o(din_conflict_o)
);

rom_s1 SYS0(
    .clk(clk),
    .start(start),
    .data_o(sys0_data),
    .data_valid_o(sys0_data_valid)
);
endmodule
//============================================================================
========
//============================================================================
========
`timescale 1ns/1ps
module testbench_sys(
    output [7:0] data_o,
    output data_valid_o,
    output din_conflict_o,
    output [7:0] f3_data_o,
    output f3_data_valid_o,
    output [7:0] f7_data_o,
    output f7_data_valid_o
);

    reg clk;
    initial begin
        clk = 1'b0;
        forever begin
            #5 clk = ~clk;//生成100MHz的激励时钟信�???
        end
    end

    reg start;
    initial begin
        start = 1'b0;
        #100;
        start = 1'b1;
        #20;
        start = 1'b0;
    end

    reg read;
    initial begin
        read=1'b0;
        #2000000;
        read=1'b1;
    end

    initial begin
        #3000000;
        $finish;
    end

test_SYS SYS_test(
    .clk(clk),
    .start(start),
    .read(read),
    .data_o(data_o),
    .data_valid_o(data_valid_o),
    .din_conflict_o(din_conflict_o),
    .f3_data_o(f3_data_o),
    .f3_data_valid_o(f3_data_valid_o),
    .f7_data_o(f7_data_o),
    .f7_data_valid_o(f7_data_valid_o)
);

    integer dout_file;
    integer dout_file_f3;
    integer dout_file_f7;
    initial begin
 dout_file=$fopen("C:/Users/Tingf/Desktop/Vivado_Project/Gaosi_diff_filter/image.txt");
 dout_file_f3=$fopen("C:/Users/Tingf/Desktop/Vivado_Project/Gaosi_diff_filter/image_f3.txt");
 dout_file_f7=$fopen("C:/Users/Tingf/Desktop/Vivado_Project/Gaosi_diff_filter/image_f7.txt");
      if(dout_file == 0)begin
                $display ("can not open the file!");    //创建文件失败，显示can not open the file!
                $stop;
       end
       if(dout_file_f3 == 0)begin
```

```
                $display ("can not open the file!");    //创建文件失败，显示can not open the file!
                $stop;
       end
       if(dout_file_f7 == 0)begin
                $display ("can not open the file!");    //创建文件失败，显示can not open the file!
                $stop;
       end
    end

    always @(posedge clk)
     if(data_valid_o)   //使能
       $fdisplay(dout_file,"%d",data_o);    //使能信号有效，每来一个时钟，写入到所创建的文件中
    always @(posedge clk)
     if(f3_data_valid_o)   //使能
       $fdisplay(dout_file_f3,"%d",f3_data_o);    //使能信号有效，每来一个时钟，写入到所创建的文件中

    always @(posedge clk)
     if(f7_data_valid_o)   //使能
       $fdisplay(dout_file_f7,"%d",f7_data_o);   //使能信号有效，每来一个时钟，写入到所创建的文件中
endmodule
//============================================================================
```

参考文献

1、D. Marr; E. Hildreth (29 February 1980). "Theory of Edge Detection". Proceedings of the Royal Society of London. Series B, Biological Sciences. The Royal Society. 207 (1167): 215–217. Bibcode:1980RSPSB.207..187M. doi:10.1098/rspb.1980.0020. JSTOR 35407.