

基于 FPGA 的图像处理

ISP 算法篇

目录

第 1 章 DPC.....	3
1.1 DPC 简介.....	3
1.2 算法模型.....	5
1.3 FPGA 架构.....	7
1.4 FPGA 模块设计.....	8
1.4.1 DPC 顶层模块设计.....	8
1.4.2 dpc_delta_5x5 模块设计.....	10
1.4.3 LineBuffer 设计.....	13
1.4.4 delta_3x3 设计.....	16
1.4.5 mid_3x3 设计.....	21
1.4.6 dpc_dat_judge 设计.....	25
1.5 代码性能.....	27
1.5.1 utilization 概览.....	27
1.5.2 时序收敛.....	28
1.5.3 性能概述.....	32
1.6 仿真.....	33
1.6.1 生成待测图片.....	33
1.6.2 待测图片生成 txt 文件.....	34
1.6.3 生成 VESA 标准视频同步信号.....	35
1.6.4 用 txt 生成视频流.....	38
1.6.5 视频流生成 txt.....	40
1.6.6 txt 生成图片.....	41
1.6.7 testbench 全貌.....	42
1.6.8 仿真波形图.....	45
1.7 后记.....	47
附录 1： 版本说明.....	48
附录 2： 参与讨论.....	49

第 1 章 DPC

1.1 DPC 简介

在数字图像处理领域，特别是与图像信号处理器（ISP）相关的环节中，Defective Pixel Correction（DPC）是一个关键技术。DPC 应对的主要问题是传感器像素的缺陷，这些缺陷可能源于制造缺陷、长期磨损或外部因素如灰尘侵入，现象如图 1.1.1。这些缺陷像素在成像时无法准确记录光线信息，表现为静态的亮点、暗点或彩点，从而破坏图像的整体质量。

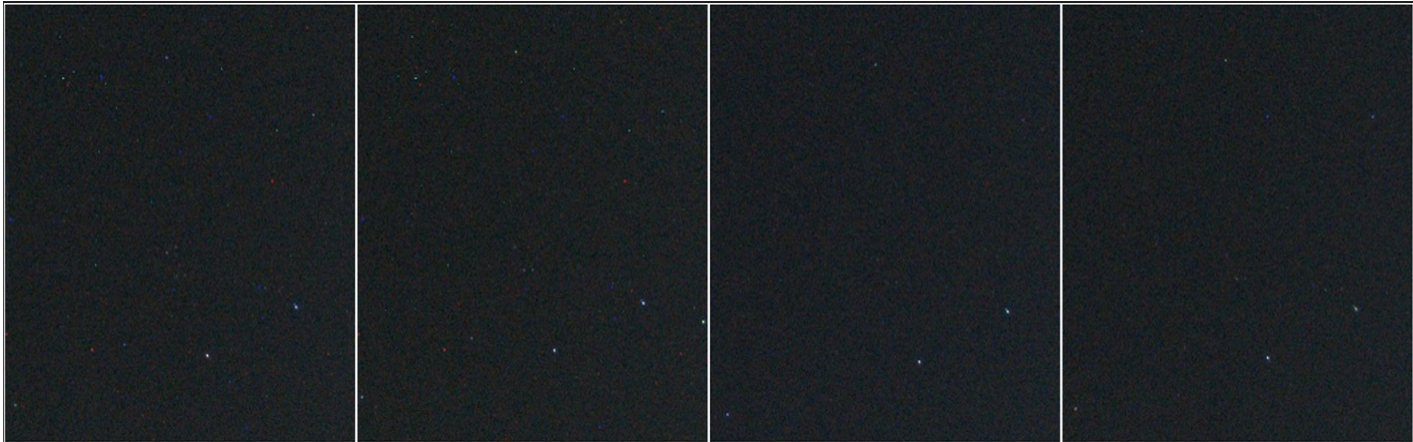


图 1.1.1 坏点现象

坏点校正技术的实施通常涉及两个阶段：检测和校正。在检测阶段，算法分析传感器输出的原始图像数据，识别出异常像素，如图 1.1.2 所示。然后，在校正阶段，利用周围正常像素的值采用插值等方法对这些异常像素进行修复。这一过程的关键在于恰当平衡，以去除缺陷的同时尽可能保留图像的真实细节。当 DPC 算法得当，其结果是让最终图像在视觉上免受这些缺陷像素的影响，保证了图像质量的准确性和一致性。

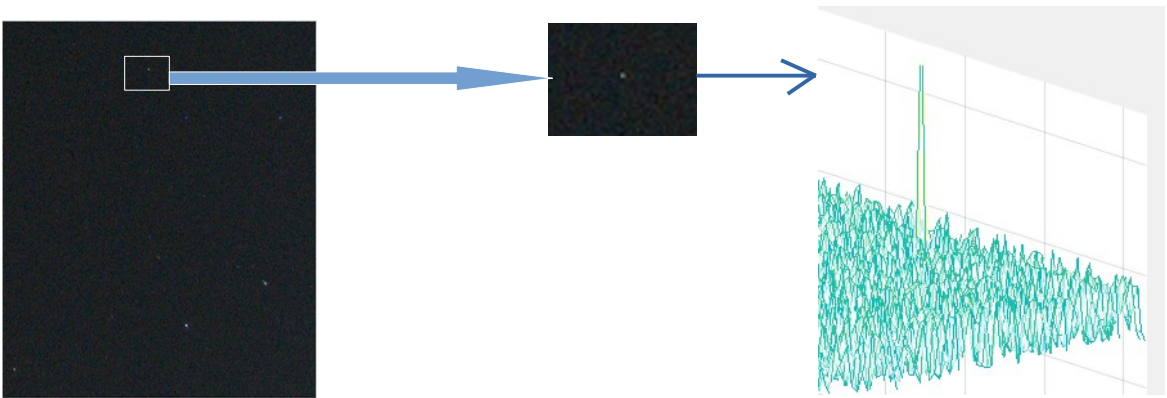


图 1.1.2 识别异常像素

运算的一般方法为

- 1, 在邻域的 3×3 区域内, 对比中心像素与周围像素之间的差值, 如此得到 8 个差值
- 2, 如果第一步中得到的 8 个差值同为正数, 或者同为负数, 并且所有值的绝对值大于设定的阈值, 那么进行第三步。否则直接输出原有的中心像素值。
- 3, 在 3×3 的区域内, 找到中值。
- 4, 用中值替换掉原来的中心像素值。

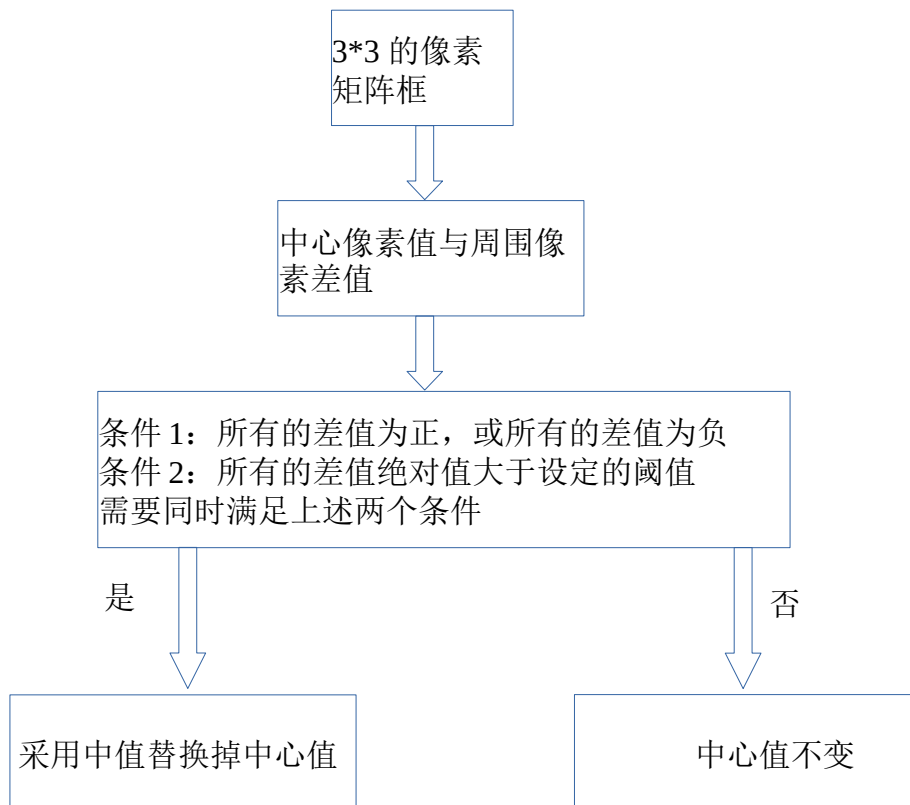


图 1.1.3 DPC 运算流程图

不过这里需要注意的是, 我们是在 Bayer 域下, 这里的 3×3 邻域在 Bayer 域下就需要变成 5×5 的矩阵框了, 没接触过 bayer 数据的读者一时还不明白也没关系, 后面还会具体讲解到。

1.2 算法模型

1.2.1 原始算法代码

将上述的计算思想转换成 matlab 代码，原始的 matlab 代码如下：

```
1 % Date : 2023
2 % author : qingshuangyimeng
3 % function : DPC
4 % Description:
5 %
6 % -----
7 % -----
8 clc;clear;close all;
9 tic;
10 % -----
11 Thrd_value = 30;
12 % -----raw parameters-----
13 raw_dat = imread('dpc_patten.png');
14 bayerFormat = 'BGGR'; %
15 % -----
16 [height, width] = size(raw_dat);
17 bayerPad = zeros(height+4,width+4);
18 bayerPad(3:height+2,3:width+2) = raw_dat;
19
20 % B G B G B G
21 % G R G R G R
22 % B G B G B G
23 % G R G R G R
24 % B G B G B G
25 % G R G R G R
26 for i = 3 : 1 : height
27     for j = 3 : 1 : width
28         if mod(i,2) == mod(j,2) % R / B
29             dpc_pad = [bayerPad(i-2, j-2) bayerPad(i-2, j) bayerPad(i-2, j+2) ...
30                       bayerPad(i, j-2) bayerPad(i, j) bayerPad(i, j+2) ...
31                       bayerPad(i+2, j-2) bayerPad(i+2, j) bayerPad(i+2, j+2)] ;
32         else % G
33             dpc_pad = [ bayerPad(i-2, j) ...
34                       bayerPad(i-1, j-1) bayerPad(i-1, j+1) ...
35                       bayerPad(i, j-2) bayerPad(i, j) bayerPad(i, j+2) ...
36                       bayerPad(i+1, j-1) bayerPad(i+1, j) bayerPad(i+1, j+1) ...
37                       bayerPad(i+2, j)] ;
38         end
39         dpc_delta = dpc_pad - ones(1, 8) * bayerPad(i, j);
40         dpc_med = median(dpc_delta);
41         if ( (nnz(dpc_delta > 0) == 8) || (nnz(dpc_delta < 0) == 8)) ...
42             & nnz((abs(dpc_delta)) > Thrd_value) == 8
43             bayerPad(i, j) = dpc_med ;
44         end
45     end
46 end
47 dpc_dat = uint8(bayerPad(3:height+2, 3:width+2));
48
49 figure();
50 %subplot (1,2,1);
51 imshow(raw_dat); title('org patten ');
52 %subplot (1,2,2);
53 figure();
54 imshow(dpc_dat); title('dpc patten');
55
56 imwrite(dpc_dat,'dpc_dat.png');
57
```

代码的 17, 18 两行的作用, 是将图像上下左右扩充 2 行 2 列, 用以保证图像输出时大小和输入时候大小一致。

代码的 29, 33 两行得出了两个不同的矩阵框, 分别代表着中心颜色不同的时候, 取数的方式不同。一共分四种情况, 图解方式如图 1.2.1。为方便理解, 代码中 `dpc_pad` 的取数也采用了矩阵和菱形的不同排列方式。

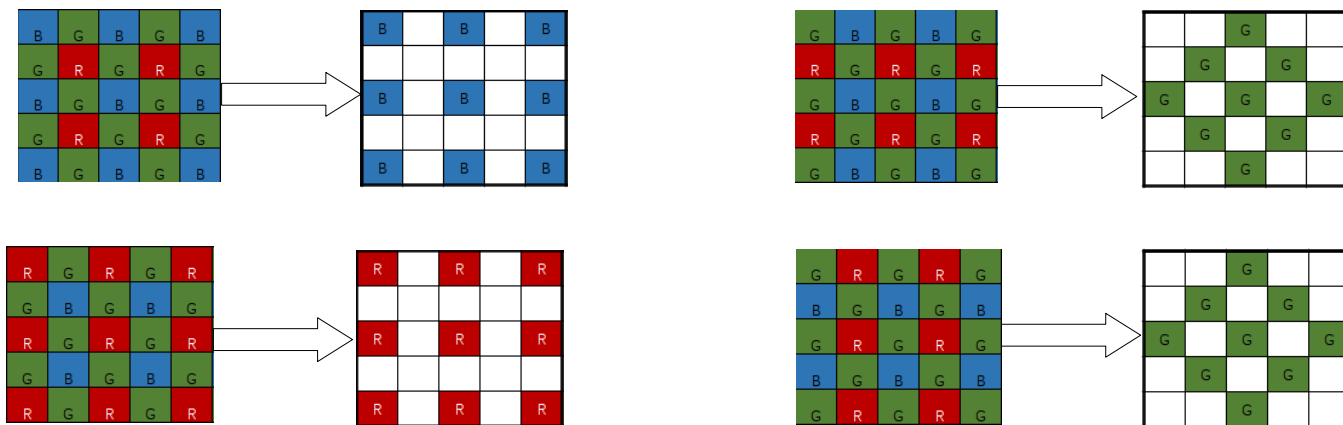


图 1.2.1 分别表示 BGGR, GBRG, RGGB,GRBG 四种领域取值方式

请记住这种领域取值方式, 因为在 Bayer 域下, 这种间隔取值方式随处可见。在 1.1 中, 最后提到为什么 3*3 的矩阵框到了 Bayer 域下就变成了 5*5, 就是这个原因。

1.3 FPGA 架构

1.3.1 架构介绍

DPC 的算法模型代码和 FPGA 代码的对应性还是很强的。加上这个模块的结构也很简单，推导硬件架构还是比较容易的。

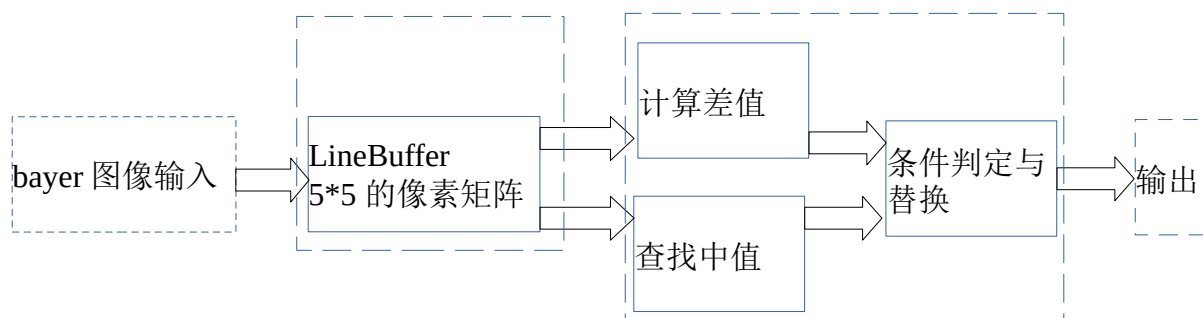


图 1.3.1 DPC 的 FPGA 架构图

首先需要有一个 LineBuffer，并且要求的是 5*5 的模式，然后在这 25 个点中取出需要的 9 个点。运用 FPGA 并行计算优势，让差值运算和中值查找并行进行。最后依据判定条件得到最终的值进行输出。

1.4 FPGA 模块设计

对于任何一个模块来说，设计之前首先需要想清楚的是，输入是什么，输出是什么，然后考虑的是如何实现这个功能，并且用最出色的方式实现这个功能。依据 1.3 中所说的模块划分方式分别进行了模块设计。

1.4.1 DPC 顶层模块设计

如同 1.3 中所构思的，DPC 完整目录结构如下：

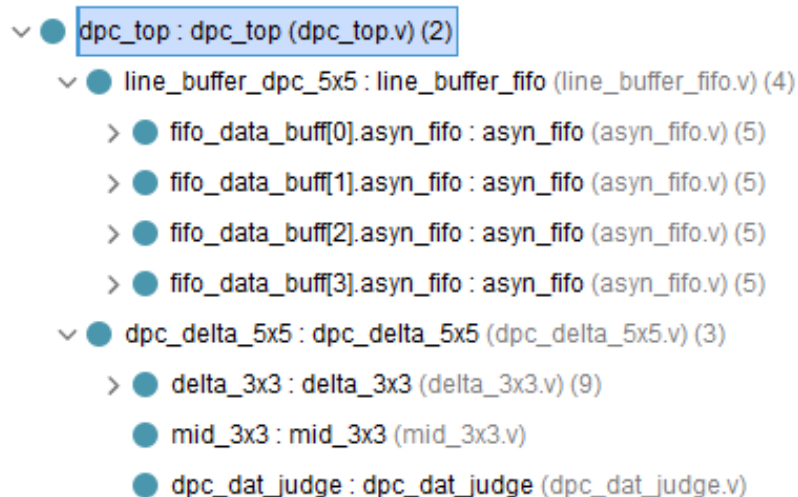


图 1.4.1.1 DPC 目录结构图

结构解释：

1，首先经过一个 LineBuffer 让一行一行输入的图像转成 5 行同时输出的图像，如此可以获取 5*5 的像素矩阵。

2，运算模块 dpc_delta_5x5 模块中，采用了并行运算模式，差值与中值一起计算，然后在 dpc_dat_judge 模块中进行判定是否需要替换原像素值。

介绍完了代码结构，再来看具体的代码实现：


```

33
34
35 module dpc_top #(
36     parameter VIDEO_DATA_WIDTH          = 8
37 )
38
39 input          I_video_clk              ,
40 input          I_video_reset           ,
41 input [7:0]    I_dpc_thrd_value,
42 input [0:0]    I_bayer_format          ,
43 input [2:0]    I_video_syn             ,
44 input [VIDEO_DATA_WIDTH-1:0] I_video_dat ,
45
46 //output [2:0]          O_line_syn_5_5 ,
47 //output [5*VIDEO_DATA_WIDTH-1:0] O_line_dat_5_5 ,
48 output [2:0]          O_video_syn      ,
49 output [VIDEO_DATA_WIDTH-1:0] O_video_dat
50 );
51 // -----
52 wire [2:0]          line_syn_5_5 ;
53 wire [VIDEO_DATA_WIDTH-1:0] line_5_1, line_5_2, line_5_3, line_5_4, line_5_5;
54 // -----
55
56 module line_buffer_fifo #(
57     .VIDEO_DATA_WIDTH (VIDEO_DATA_WIDTH ), //输入数据位宽
58     .FIFO_ADDR_WIDTH  (11                ), // 2**12= 4096>3840
59     .LINE_BUFFER_VECTOR (5                )
60 ) line_buffer_dpc_5x5 (
61     .I_video_clk      (I_video_clk      ),
62     .I_reset          (I_video_reset    ),
63     .I_line_vaild     (1'd0             ),
64     .I_video_syn      (I_video_syn      ),
65     .I_video_dat      (I_video_dat      ),
66     .O_video_line     ({line_5_5,line_5_4,line_5_3,line_5_2,line_5_1}),
67     .O_video_syn      (line_syn_5_5    )
68 );
69
70 module dpc_delta_5x5 #(
71     .VIDEO_DATA_WIDTH (VIDEO_DATA_WIDTH )
72 ) dpc_delta_5x5 (
73     .I_video_clk      (I_video_clk      ),
74     .I_dpc_thrd_value (I_dpc_thrd_value ),
75     .I_bayer_format   (I_bayer_format[0] ),
76     .I_video_syn      (line_syn_5_5    ),
77     .I_video_dat1     (line_5_1        ),
78     .I_video_dat2     (line_5_2        ),
79     .I_video_dat3     (line_5_3        ),
80     .I_video_dat4     (line_5_4        ),
81     .I_video_dat5     (line_5_5        ),
82     .O_video_syn      (O_video_syn      ),
83     .O_video_dat      (O_video_dat      )
84 );
85
86 //assign O_line_syn_5_5 = line_syn_5_5 ;
87 //assign O_line_dat_5_5 = {line_5_5,line_5_4,line_5_3,line_5_2,line_5_1};
88 endmodule
89

```

代码解释：

1, top 层, 输入为 Bayer 形式的图像数据和同步信号, 处理之后输出也是 Bayer 形式数据和图像, 格式不变。

2, 输入阈值设定 I_dpc_thrd_value 记得要预留到外面, 后续需要调试。

3, 需要外部设定一个 I_bayer_fomat 这个参数。从 matlab 代码中也可以看到, Bayer 的数据格式, B/R 在前与 G 在前的时候取数的方式是不一样的, 前者是矩阵模式, 后者是菱形模式。具体的理解方式, 可以看图 1.2.1。原本这个参数是多 bit 的, 巧妙的规定了一下格式, 让这里只需要一个 bit。

4, LineBuffer 的作用, 是将原本 1 行 1 行输入的方式生成多行一起并行输出的方式。从 matlab 源码中可以看到, 此模块中需要的 5 行一起并行输出, 形成一个 5*5 的矩阵, 所以参数 LINE_NUMBER_VECTOR 设定为 5. 明明是 3*3 的矩阵, 为何此处为 5, 在 1.2.1 里面也做了解释, 这里就不赘述。

5, dpc_delta_5x5 模块, 输入 5 行并行数据, 此模块的内部会经过差值, 查中值, 判定, 最后输出最终结果。

1.4.2 dpc_delta_5x5 模块设计

我们再来看看 dpc_delta_5x5 这个模块是怎么设计的。它在输入的 5x5 矩阵中选取所需要的 9 个点, 送入差值运算模块 delta_3x3 中进行差值运算, 同时在 mid_3x3 进行查找中值。最后在 dpc_dat_judge 中依据判定条件计算最后的像素值。此时就需要根据 I_bayer_format 来确定当下有效的 9 个点的位置, 于是就有了行计数, 列计数。

```

34
35 module dpc_delta_5x5 #(
36     parameter VIDEO_DATA_WIDTH          = 8
37 )
38
39 input          I_video_clk              ,
40 input [7:0]    I_dpc_thrd_value        ,
41 input [0:0]    I_bayer_format          , // 0: B/R G B/R , 1: G B/R G
42 input [2:0]    I_video_syn             ,
43 input [VIDEO_DATA_WIDTH-1:0] I_video_dat1 ,
44 input [VIDEO_DATA_WIDTH-1:0] I_video_dat2 ,
45 input [VIDEO_DATA_WIDTH-1:0] I_video_dat3 ,
46 input [VIDEO_DATA_WIDTH-1:0] I_video_dat4 ,
47 input [VIDEO_DATA_WIDTH-1:0] I_video_dat5 ,
48 output [2:0]    O_video_syn            ,
49 output [VIDEO_DATA_WIDTH-1:0] O_video_dat
50 );
51 // -----
52 reg [VIDEO_DATA_WIDTH-1:0] r1_1, r1_2, r1_3, r1_4, r1_5;
53 reg [VIDEO_DATA_WIDTH-1:0] r2_1, r2_2, r2_3, r2_4, r2_5;
54 reg [VIDEO_DATA_WIDTH-1:0] r3_1, r3_2, r3_3, r3_4, r3_5;
55 reg [VIDEO_DATA_WIDTH-1:0] r4_1, r4_2, r4_3, r4_4, r4_5;
56 reg [VIDEO_DATA_WIDTH-1:0] r5_1, r5_2, r5_3, r5_4, r5_5;
57 reg [12:0] syn_v, syn_h, syn_d;
58 wire synd_pos, synv_neg;
59 always @(posedge I_video_clk) {r1_1,r1_2,r1_3,r1_4,r1_5} <= {r1_2,r1_3,r1_4,r1_5,I_video_dat1};
60 always @(posedge I_video_clk) {r2_1,r2_2,r2_3,r2_4,r2_5} <= {r2_2,r2_3,r2_4,r2_5,I_video_dat2};
61 always @(posedge I_video_clk) {r3_1,r3_2,r3_3,r3_4,r3_5} <= {r3_2,r3_3,r3_4,r3_5,I_video_dat3};
62 always @(posedge I_video_clk) {r4_1,r4_2,r4_3,r4_4,r4_5} <= {r4_2,r4_3,r4_4,r4_5,I_video_dat4};
63 always @(posedge I_video_clk) {r5_1,r5_2,r5_3,r5_4,r5_5} <= {r5_2,r5_3,r5_4,r5_5,I_video_dat5};
64 always @(posedge I_video_clk) syn_v <= {syn_v[11:0],I_video_syn[2]};
65 always @(posedge I_video_clk) syn_h <= {syn_h[11:0],I_video_syn[1]};
66 always @(posedge I_video_clk) syn_d <= {syn_d[11:0],I_video_syn[0]};
67 assign synd_pos = (syn_d[4:3]==2'b01)? 1'd1:1'd0 ;
68 assign synv_neg = (syn_v[4:3]==2'b10)? 1'd1:1'd0 ;
69 // -----
70 reg pixel_odd, line_odd ;
71 reg [VIDEO_DATA_WIDTH-1:0] r1_1r0,r1_3r0,r1_5r0;
72 reg [VIDEO_DATA_WIDTH-1:0] r3_1r0,r3_3r0,r3_5r0;
73 reg [VIDEO_DATA_WIDTH-1:0] r5_1r0,r5_3r0,r5_5r0;
74 always @(posedge I_video_clk)
75     if(synd_pos) pixel_odd <= ~I_bayer_format ;
76     else if(synd_d[4]) pixel_odd <= ~pixel_odd ;
77 always @(posedge I_video_clk)
78     if(synv_neg) line_odd <= 1'd0 ;
79     else if(synd_pos) line_odd <= ~line_odd ;
80 always @(posedge I_video_clk)
81     if(pixel_odd == line_odd) begin // B / R
82         {r1_1r0,r1_3r0,r1_5r0}<={r1_1,r1_3,r1_5};
83         {r3_1r0,r3_3r0,r3_5r0}<={r3_1,r3_3,r3_5};
84         {r5_1r0,r5_3r0,r5_5r0}<={r5_1,r5_3,r5_5};
85     end

```

```

86   else begin      // G
87       {r1_3r0}          <={r1_3};
88       {r1_1r0,r1_5r0}    <={r2_2,r2_4};
89       {r3_1r0,r3_3r0,r3_5r0} <={r3_1,r3_3,r3_5};
90       {r5_1r0,r5_5r0}    <={r4_2,r4_4};
91       {r5_3r0}          <={r5_3};
92   end
93   // -----
94   wire [VIDEO_DATA_WIDTH:0] dlt1,dlt2,dlt3,dlt4,dlt5,dlt6,dlt7,dlt8,dlt9 ;
95   wire [VIDEO_DATA_WIDTH-1:0] mid_dat ;
96   delta_3x3 #(
97       .DATA_WIDTH          (VIDEO_DATA_WIDTH          )
98   ) delta_3x3 (
99       .I_clk                (I_video_clk                ),
100      .I_line3_1             ({r1_1r0,r1_3r0,r1_5r0} ),
101      .I_line3_2             ({r3_1r0,r3_3r0,r3_5r0} ),
102      .I_line3_3             ({r5_1r0,r5_3r0,r5_5r0} ),
103      .I_line_vaild          ( syn_d[5]                ),
104      .O_line3_1             ({dlt1,dlt2,dlt3}          ),
105      .O_line3_2             ({dlt4,dlt5,dlt6}          ),
106      .O_line3_3             ({dlt7,dlt8,dlt9}          ),
107      .O_line_vaild          (                          )
108   );
109
110   mid_3x3 #(
111       .DATA_WIDTH          (VIDEO_DATA_WIDTH          )           //输入数据
112   ) mid_3x3 (
113       .I_clk                (I_video_clk                ),
114       .I_line3_1             ({r1_1r0,r1_3r0,r1_5r0} ),
115       .I_line3_2             ({r3_1r0,r3_3r0,r3_5r0} ),
116       .I_line3_3             ({r5_1r0,r5_3r0,r5_5r0} ),
117       .I_line_vaild          ( syn_d[5]                ),
118       .O_mid_dat             (mid_dat                    ),
119       .O_mid_vaild           (                          )
120   );
121   // -----
122   dpc_dat_judge #(
123       .DATA_WIDTH          (VIDEO_DATA_WIDTH          )
124   ) dpc_dat_judge (
125       .clk                  (I_video_clk                ),
126       .I_dpc_thrd_value     (I_dpc_thrd_value          ),
127       .I_line3_1            ({dlt1,dlt2,dlt3}          ),
128       .I_line3_2            ({dlt4,dlt5,dlt6}          ),
129       .I_line3_3            ({dlt7,dlt8,dlt9}          ),
130       .I_dat_vaild          ( syn_d[9]                  ),
131       .I_mid_dat            ( mid_dat                    ),
132       .O_judge_dat          (O_video_dat                ),
133       .O_judge_vaild        (                          )
134   );
135   assign O_video_syn = {syn_v[12],syn_h[12],syn_d[12]} ;
136   endmodule

```

代码解释:

1, 59-63 行, 依据 5 行并行输入的特点, 在时序的加持下, 制造了一个 5*5 的像素矩阵

2, 74-79 行, 计算矩形框运行到了哪个像素点上。类似 1.2.1 中第 28 行 if 的功能。判定是在矩形框上取数据, 还是菱形框上取数据。对这几行有疑问的就回过去看 1.2.1 的算法代码。

3, 110 行, delta_3x3 就开始计算中心点与周围 8 个像素点的差值, 而 mid_3x3 就开始计算这 9 个点的中值。

4, 122 行, dpc_dat_judge 就在输入的差值上判定要不要用中值替换掉原来的值。

5, 从输入到输出, 总共耗费了 13 个 clock Latency 。

1.4.3 LineBuffer 设计

LineBuffer 是图像处理中经常用到的模块, 相信从事 FPGA 图像处理工作的人几乎都设计过这个模块 (xilinx 官方没有这个 IP), 那么能不能设计出一款类似的 IP 只需要配置一下, 就可以任意条件下调用呢? 答案当然是可以的, 没有的 IP 就自己设计。

这是我设计的一款支持任意行并行输出的 LineBuffer, 只需要配置一下参数, 有兴趣的朋友可以自行打包, 可以在任意平台下调用。LineBuffer 的代码如下:

```
32
33 module line_buffer_fifo #(
34     parameter VIDEO_DATA_WIDTH      = 8 ,
35     parameter FIFO_ADDR_WIDTH       = 12 ,
36     parameter LINE_NUMBER_VECTOR   = 5
37 ) (
38     input                                I_video_clk      ,
39     input                                I_reset          ,
40     input                                I_de_vaild       ,
41     input [2:0]                          I_video_syn     ,
42     input [VIDEO_DATA_WIDTH-1:0]         I_video_dat     ,
43     output [LINE_NUMBER_VECTOR*VIDEO_DATA_WIDTH-1:0] O_video_line , // first in low , 1
44     output [2:0]                          O_video_syn    ,
45     output [LINE_NUMBER_VECTOR-2:0]      O_lb_debug
46 );
47 // *****
48 reg [3:0] syn_v, syn_h, syn_d ;
49 reg [VIDEO_DATA_WIDTH-1:0] dat_r0, dat_r1, dat_r2;
50 wire syn_de_pos ;
51 always @(posedge I_video_clk) syn_d <= {syn_d[2:0],I_video_syn[0]};
52 always @(posedge I_video_clk) syn_h <= {syn_h[2:0],I_video_syn[1]};
53 always @(posedge I_video_clk) syn_v <= {syn_v[2:0],I_video_syn[2]};
54 always @(posedge I_video_clk) {dat_r2,dat_r1,dat_r0} <= {dat_r1,dat_r0,I_video_dat} ;
55 assign syn_de_pos = ({syn_d[0],I_video_syn[0]}==2'b01)? 1'd1:1'd0 ;
56 // -----
57 reg [LINE_NUMBER_VECTOR-1:0] fifo_disable ;
58 wire [VIDEO_DATA_WIDTH-1:0] fifo_data [0:LINE_NUMBER_VECTOR-1] ;
59 wire [LINE_NUMBER_VECTOR-2:0] wfull , empty ;
```

```

60 always @(posedge I_video_clk)
61     if(syn_v[0]|I_reset)        fifo_disable    <= {(LINE_NUMBER_VECTOR){1'd1}};
62     else if(syn_de_pos)          fifo_disable    <= fifo_disable<<1 ;
63 assign fifo_data[0]             = dat_r2 ;
64 genvar fifo_num ;
65 generate
66     for (fifo_num = 0;fifo_num< LINE_NUMBER_VECTOR-1; fifo_num = fifo_num + 1) begin:dat_bf
67 xilinx_fifo_in_lb    xilinx_fifo_in_lb(
68         .clk           (I_video_clk
69         .srst           (syn_v[0]
70         .wr_en          ((!fifo_disable[fifo_num])&syn_d[2]
71         .din            (fifo_data[fifo_num]
72         .rd_en          ((!fifo_disable[fifo_num+1])&syn_d[1]
73         .dout           (fifo_data[fifo_num+1]
74         .full           (wfull[fifo_num]
75         .empty          (rempty[fifo_num]
76     ); /*
77     asyn_fifo #(
78         .DSIZE          (VIDEO_DATA_WIDTH
79         .ASIZE          (FIFO_ADDR_WIDTH
80     )asyn_fifo(
81         .wclk           (I_video_clk
82         .wrst_n          (! (syn_v[0] |I_reset)
83         .winc           ((!fifo_disable[fifo_num])&syn_d[2]
84         .wdata          (fifo_data[fifo_num]
85         .rclk           (I_video_clk
86         .rrst_n          (! (syn_v[0] |I_reset)
87         .rinc           ((!fifo_disable[fifo_num+1])&syn_d[1]
88         .rdata          (fifo_data[fifo_num+1]
89         .wfull          (wfull[fifo_num]
90         .rempty         (rempty[fifo_num]
91     );*/
92 end
93 endgenerate

94 // -----
95 wire [LINE_NUMBER_VECTOR*VIDEO_DATA_WIDTH-1:0] lb_dat ;
96 genvar dat_num ;
97 generate
98     for (dat_num=0; dat_num < LINE_NUMBER_VECTOR;dat_num=dat_num+1) begin: reshape_fifo_dat
99         assign lb_dat[(dat_num+1)*VIDEO_DATA_WIDTH-1:dat_num*VIDEO_DATA_WIDTH]
100             = fifo_data[LINE_NUMBER_VECTOR-dat_num-1];
101     end
102 endgenerate
103 // -----
104 reg [LINE_NUMBER_VECTOR*VIDEO_DATA_WIDTH-1:0] lb_dat_r ;
105 wire de_vaild ;
106 always @(posedge I_video_clk) lb_dat_r <= lb_dat ;
107 assign de_vaild = (fifo_disable==(LINE_NUMBER_VECTOR){1'd0}))?1'd1:1'd0;
108 assign O_video_line = lb_dat_r ;
109 assign O_video_syn[0] = syn_d[3] & (de_vaild | (!I_de_vaild));
110 assign O_video_syn[1] = syn_h[3];
111 assign O_video_syn[2] = syn_v[3];
112 assign O_lb_debug     = wfull;
113 endmodule

```

代码解释:

1, 此 LineBuffer 采用 FIFO 设计, 集成到 generate 中。因此只需要修改 LINE_BUFFER_VECTOR 参数就可以实现任意 line 的并行输出。VIDEO_DATA_WIDTH 表示输入数据位宽。

FIFO_ADDR_WIDTH 表示开辟的 FIFO 的位宽大小，当然需要满足输入一行的数据总量 $Num \leq 2^{FIFO_ADDR_WIDTH}$ 。

2, 58 行，控制着 FIFO 的读写使能，因为在刚开始输入的时候，并不是所有 FIFO 都是开启的，只有上面的 FIFO 存满了一行，下面的 fifo 才会开启，如此反复，慢慢的数据流入到下一层的 FIFO 里面去。 fifo_disable 这个变量就在达到这个目的。

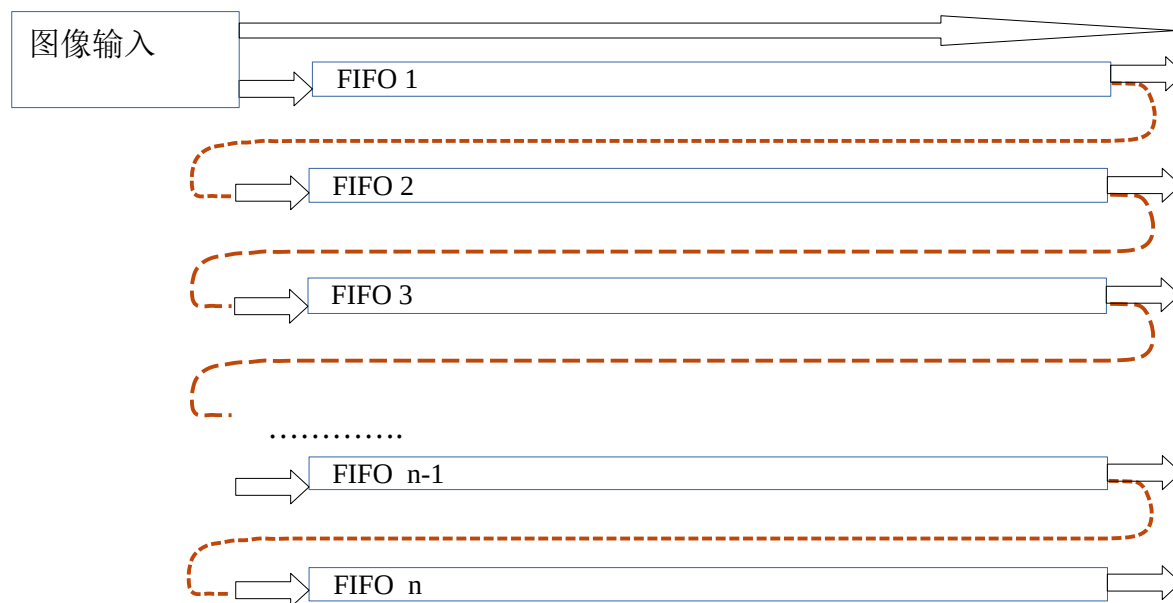


图 1.4.2.1 linebuffer 数据流动示意图

一共采用 n 个 fifo 就可以同时输出 $n+1$ 行并行数据。

3, 代码中引用 asyn_fifo 这个 IP，当然也可以使用同步 fifo，也可以用 xilinx 官方 IP，不过要注意在引用 xilinx 官方 IP 的时候位宽和深度信息需要与 VIDEO_DATA_WIDTH, FIFO_ADDR_WIDTH 这两个参数一致。asyn_fifo 我采用的是一个开源的异步 fifo 代码。

4, 103 行的 generate，是用来展开 fifo_data 这个二维矩阵，当然如果你采用 SystemVerilog 的 interface 也可以直接矩阵输出。

5, 111 行，将 fifo 的数据再打一拍输出。

6, 如果需要 M 行并行输出，那么此 LineBuffer 总延迟 = $(M-1 \text{ 行}) + (4 \text{ 个(也可以修改成 3 个, 或者 2 个)clock 的 Latency})$ 。

7, 如果需要 M 行并行输出，那么需要缓存的行数为 $M-1$ 行，代码中只会申请 $M-1$ 行的行缓存空间。

1.4.4 delta_3x3 设计

此模块的功能就是让输入的 8 个值与中间的值做一个减法。有了高度定制化的代码库，一般采用了调用的方法，如此会让这个模块看起来非常简洁，并且在 LUT 运算和 DSP 运算上自由切换。基本上实现了 ctrl+c 加 ctrl+v 就完成了此模块的设计。输入 3*3 的矩阵，输出也是 3*3 的矩阵。

```
34
35 module delta_3x3 #(
36     parameter DATA_WIDTH          = 8
37 ) (
38     input                                I_clk
39     input [3*DATA_WIDTH-1:0]          I_line3_1
40     input [3*DATA_WIDTH-1:0]          I_line3_2
41     input [3*DATA_WIDTH-1:0]          I_line3_3
42     input                                I_line_vaild
43     output [3*DATA_WIDTH+2:0]          O_line3_1
44     output [3*DATA_WIDTH+2:0]          O_line3_2
45     output [3*DATA_WIDTH+2:0]          O_line3_3
46     output                                O_line_vaild
47 );
48 // -----
49 reg [DATA_WIDTH-1:0] reg1_1, reg1_2, reg1_3 ;
50 reg [DATA_WIDTH-1:0] reg2_1, reg2_2, reg2_3 ;
51 reg [DATA_WIDTH-1:0] reg3_1, reg3_2, reg3_3 ;
52 reg [DATA_WIDTH-1:0] reg2_2_1, reg2_2_2, reg2_2_3 ;
53 reg data_vaild ;
54 always @(posedge I_clk) data_vaild <= I_line_vaild ;
55 always @(posedge I_clk) begin
56     {reg1_1, reg1_2, reg1_3} = {I_line3_1} ;
57     {reg2_1, reg2_2, reg2_3} = {I_line3_2} ;
58     {reg3_1, reg3_2, reg3_3} = {I_line3_3} ;
59 end
60 always @(posedge I_clk)
61     {reg2_2_1, reg2_2_2, reg2_2_3} = {3{I_line3_2[2*DATA_WIDTH-1:DATA_WIDTH]}} ;
62 // -----
63 wire [DATA_WIDTH:0] O_dat1, O_dat2, O_dat3, O_dat4, O_dat6, O_dat7, O_dat8, O_dat9;
64
65 subtr_signed #(
66     .SUB_CALCULATE_TYPE ("DSP" ), // "DSP "
67     .INPUT_DATA_WIDTH1   (DATA_WIDTH+1 ),
68     .INPUT_DATA_WIDTH2   (DATA_WIDTH+1 )
69 ) subtr_lty3_8_1 (
70     .I_clk                (I_clk ),
71     .I_dat_en              (data_vaild ),
72     .I_dat1                ({1'd0, reg2_2_1} ),
73     .I_dat2                ({1'd0, reg1_1} ),
74     .O_dat                 (O_dat1 ),
75     .O_dat_vaild           ()
76 );
77
78 subtr_signed #(
79     .SUB_CALCULATE_TYPE ("LUT" ),
80     .INPUT_DATA_WIDTH1   (DATA_WIDTH+1 ),
81     .INPUT_DATA_WIDTH2   (DATA_WIDTH+1 )
82 ) subtr_lty3_8_2 (
83     .I_clk                (I_clk ),
84     .I_dat_en              (data_vaild ),
85     .I_dat1                ({1'd0, reg2_2_2} ),
86     .I_dat2                ({1'd0, reg1_2} ),
```



```

87         .O_dat                (O_dat2          ),
88         .O_dat_vaild          ()
89     );
90
91     subtr_signed #(
92         .SUB_CALCUC_TYPE       ("LUT"           ),
93         .INPUT_DATA_WIDTH1     (DATA_WIDTH+1    ),
94         .INPUT_DATA_WIDTH2     (DATA_WIDTH+1    )
95     ) subtr_lty3_8_3(
96         .I_clk                 (I_clk           ),
97         .I_dat_en              (data_vaild      ),
98         .I_dat1                ({1'd0,reg2_2_1}),
99         .I_dat2                ({1'd0,reg1_3}   ),
100        .O_dat                 (O_dat3          ),
101        .O_dat_vaild           ()
102    );
103
104    subtr_signed #(
105        .SUB_CALCUC_TYPE       ("LUT"           ),
106        .INPUT_DATA_WIDTH1     (DATA_WIDTH+1    ),
107        .INPUT_DATA_WIDTH2     (DATA_WIDTH+1    )
108    ) subtr_lty3_8_4(
109        .I_clk                 (I_clk           ),
110        .I_dat_en              (data_vaild      ),
111        .I_dat1                ({1'd0,reg2_2_1}),
112        .I_dat2                ({1'd0,reg2_1}   ),
113        .O_dat                 (O_dat4          ),
114        .O_dat_vaild           ()
115    );
116
117    subtr_signed #(
118        .SUB_CALCUC_TYPE       ("LUT"           ),
119        .INPUT_DATA_WIDTH1     (DATA_WIDTH+1    ),
120        .INPUT_DATA_WIDTH2     (DATA_WIDTH+1    )
121    ) subtr_lty3_8_5(
122        .I_clk                 (I_clk           ),
123        .I_dat_en              (data_vaild      ),
124        .I_dat1                ({1'd0,reg2_2_2}),
125        .I_dat2                ({1'd0,reg2_3}   ),
126        .O_dat                 (O_dat6          ),
127        .O_dat_vaild           ()
128    );
129
130    subtr_signed #(
131        .SUB_CALCUC_TYPE       ("LUT"           ),
132        .INPUT_DATA_WIDTH1     (DATA_WIDTH+1    ),
133        .INPUT_DATA_WIDTH2     (DATA_WIDTH+1    )
134    ) subtr_lty3_8_6(
135        .I_clk                 (I_clk           ),
136        .I_dat_en              (data_vaild      ),

```

```

137     .I_dat1      ({1'd0,reg2_2_2}),
138     .I_dat2      ({1'd0,reg3_1} ),
139     .O_dat       (O_dat7      ),
140     .O_dat_vaild  ()
141 );
142
143 subtr_signed #(
144     .SUB_CALCULATE_TYPE ("LUT"      ),
145     .INPUT_DATA_WIDTH1  (DATA_WIDTH+1 ),
146     .INPUT_DATA_WIDTH2  (DATA_WIDTH+1 )
147 ) subtr_lty3_8_7(
148     .I_clk           (I_clk      ),
149     .I_dat_en        (data_vaild ),
150     .I_dat1          ({1'd0,reg2_2_3}),
151     .I_dat2          ({1'd0,reg3_2} ),
152     .O_dat           (O_dat8     ),
153     .O_dat_vaild     ()
154 );
155
156 subtr_signed #(
157     .SUB_CALCULATE_TYPE ("LUT"      ),
158     .INPUT_DATA_WIDTH1  (DATA_WIDTH+1 ),
159     .INPUT_DATA_WIDTH2  (DATA_WIDTH+1 )
160 ) subtr_lty3_8_8(
161     .I_clk           (I_clk      ),
162     .I_dat_en        (data_vaild ),
163     .I_dat1          ({1'd0,reg2_2_3}),
164     .I_dat2          ({1'd0,reg3_3} ),
165     .O_dat           (O_dat9     ),
166     .O_dat_vaild     ()
167 );
168 //-----
169 reg      [3:0]      vaild_r ;
170 wire     [DATA_WIDTH-1:0] reg2_2r ;
171 always @(posedge I_clk) vaild_r <= {vaild_r[2:0],I_line_vaild};
172
173 latency_module #(
174     .DATA_WIDTH          (DATA_WIDTH      ), //输入类
175     .LATENCY_VECTOR      (3              )
176 ) latency_delta(
177     .I_clk              (I_clk      ),
178     .I_dat              (reg2_2      ),
179     .O_dat              (reg2_2r     )
180 );
181 // -----
182 assign O_line3_1 = {O_dat1, O_dat2, O_dat3};
183 assign O_line3_2 = {O_dat4, {1'd0,reg2_2r}, O_dat6};
184 assign O_line3_3 = {O_dat7, O_dat8, O_dat9};
185 assign O_line_vaild = vaild_r[3] ;
186
187
188 endmodule

```

代码解释:

1, 此模块的输入是 3*3 的矩阵, 输出依然是 3*3 的矩阵。输出时, 中间的数据用原本数据填充。因为后面的模块还要用到它呢。

2, 此模块中大量引用了 subtr_signed, 这是一个带符号的减法器, 自己写或引用官方 IP 都是可以的, 延迟设定在 3 Clock Latency。此模块设计的目的是便于自己在不同的 vivado 版本之间切换, 以及方便仿真 —— 我可以用其他仿真工具。设计方式如下:

```
30 module subtr_signed #(
31     parameter SUB_CALCUL_TYPE = "DSP" , // NULL
32     parameter INPUT_DATA_WIDTH1 = 8 ,
33     parameter INPUT_DATA_WIDTH2 = 8
34 )
35 (
36     input I_clk ,
37     input I_dat_en ,
38     input [INPUT_DATA_WIDTH1-1:0] I_dat1 ,
39     input [INPUT_DATA_WIDTH2-1:0] I_dat2 ,
40     output [INPUT_DATA_WIDTH1-1:0] O_dat ,
41     output O_dat_vaild
42 );
43 // -----
44 generate
45 // -----
46 if (SUB_CALCUL_TYPE == "DSP") begin :sub_us_dsp_module
47
48     subtr_s_dsp #(
49         .INPUT_DATA_WIDTH1 (INPUT_DATA_WIDTH1) ,
50         .INPUT_DATA_WIDTH2 (INPUT_DATA_WIDTH2)
51     ) subtr_s_dsp (
52         .I_clk (I_clk) ,
53         .I_dat_en (I_dat_en) ,
54         .I_dat1 (I_dat1) ,
55         .I_dat2 (I_dat2) ,
56         .O_dat (O_dat) ,
57         .O_dat_vaild ( O_dat_vaild )
58     );
59
60 end
61 else begin :sub_us_lut_module
62 // -----
63 reg [2:0] en_r ;
64 always @(posedge I_clk) en_r <= {en_r[1:0], I_dat_en};
65
```

```

66 reg signed [INPUT_DATA_WIDTH1-1:0] reg1 , reg3, reg4;
67 reg signed [INPUT_DATA_WIDTH2-1:0] reg2 ;
68 always @(posedge I_clk)
69     if(I_dat_en) {reg1, reg2} <= {I_dat1,I_dat2} ;
70 always @(posedge I_clk) reg3 <= $signed(reg1) - $signed(reg2) ;
71 always @(posedge I_clk) reg4 <= reg3 ;
72
73 assign O_dat_vaild = en_r[2] ;
74 assign O_dat = reg4;
75
76 end
77 // -----
78 endgenerate
79
80
81 endmodule

```

（subtr_s_dsp 模块与这个模块代码部分是相同的，仅仅在 module 申明部分引入了综合指令（*use_dsp="yes"））

3，还有一个简单的延迟模块，整个模块更为简单。为了方便，也写成了一个独立的 module。只需要设定参数就可以任意时间延迟，方便信号同步。

```

32
33 module latency_module #(
34     parameter DATA_WIDTH      = 8 ,
35     parameter LATENCY_VECTOR   = 3
36 ) (
37     input          I_clk      ,
38     input [DATA_WIDTH-1:0] I_dat ,
39     output [DATA_WIDTH-1:0] O_dat
40 );
41 // *****
42 reg [LATENCY_VECTOR*DATA_WIDTH-1:0] dat_reg ;
43 always @(posedge I_clk)
44     dat_reg <= {I_dat,dat_reg[LATENCY_VECTOR*DATA_WIDTH-1:DATA_WIDTH]};
45 assign O_dat = dat_reg[DATA_WIDTH-1:0] ;
46 endmodule

```

1.4.5 mid_3x3 设计

它仅仅是一个中值滤波模块，但是此模块将是非常精彩。中值滤波本身是一个简单的功能，就是在 3*3 的模块中找出中间值，对此可以在网上找到很多版本的中值滤波，最为简约，时序最好，占用资源最少的将是我介绍的这种写法。

在 3x3 窗口中获取 9 个数据，对 9 这个数据值进行排序，排序步骤如下

- A) 窗内的每行数据找到 最大值、中间值和最小值；
- B) 把三列的最小值相比较，取其中的最大值；
- C) 把三列的最大值相比较，取其中的最小值；
- D) 把三列的中间值相比较，再取一次中间值；
- E) 再把 B,C,D 中得到的三个值再排序，获取中值。

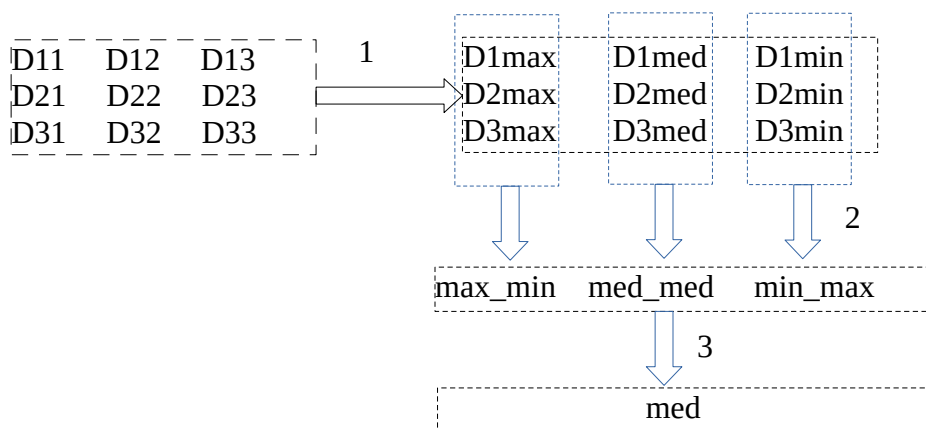


图 1.4.3.1 中值滤波计算方式示意图

三步找出中值，将问题由 9 个数里面找中值，变成了三步从三个数里面比较大小了。慢着，如果仅仅如此，此模块就称不上精彩了。下面要讨论如何在三个数里面找中值了。按照概率统计方法，三个数比较大小，应该需要比较 $C_3^2 = 3 \times 2 / 2 \times 1 = 3$ 次，这个没有丝毫疑问，比较三次之后会得到 3 个标志位：

```
comp[0] = a > b ? 1'd1: 1'd0 ;
comp[1] = a > c ? 1'd1: 1'd0 ;
comp[2] = b > c ? 1'd1: 1'd0 ;
```

好了，我现在有 3bit 用来表示 3 个数据之间的大小关系，此时脑海中就是这么一个代码段：

```
case (comp[2:0])
    3'b000 : {max, med, min} = ... ;
    3'b001 : {max, med, min} = ... ;
    ...
    3'b111 : {max, med, min} = ... ;
endcase
```

case 里面有 3 个 bit，一共 8 种可能，这是一个误区。当你把 8 种状态展开的时候会发现，有些状态是不存在的，其实只有 6 种可能的排序： $C_3^1 * C_2^1 * C_1^1 = 6$ 。

```

33 module mid_3x3 #(
34     parameter DATA_WIDTH = 8
35 ) (
36     input I_clk ,
37     input [3*DATA_WIDTH-1:0] I_line3_1 ,
38     input [3*DATA_WIDTH-1:0] I_line3_2 ,
39     input [3*DATA_WIDTH-1:0] I_line3_3 ,
40     input I_line_vaild ,
41     output [DATA_WIDTH-1:0] O_mid_dat ,
42     output O_mid_vaild
43 );
44 // *****
45 reg [3:0] line_vaild ;
46 always @(posedge clk) line_vaild <= {line_vaild[2:0] , I_line_vaild};
47 // -----
48 reg [DATA_WIDTH-1:0] dat1_1, dat1_2, dat1_3 ;
49 reg [DATA_WIDTH-1:0] dat2_1, dat2_2, dat2_3 ;
50 reg [DATA_WIDTH-1:0] dat3_1, dat3_2, dat3_3 ;
51 always @(posedge clk) {dat1_3, dat1_2, dat1_1} <= {I_line3_1} ;
52 always @(posedge clk) {dat2_3, dat2_2, dat2_1} <= {I_line3_2} ;
53 always @(posedge clk) {dat3_3, dat3_2, dat3_1} <= {I_line3_3} ;
54 //-----
55 wire a_cmp1_12, a_cmp1_13, a_cmp1_23 ;
56 wire a_cmp2_12, a_cmp2_13, a_cmp2_23 ;
57 wire a_cmp3_12, a_cmp3_13, a_cmp3_23 ;
58 assign a_cmp1_12 = (dat1_1 > dat1_2) ? 1'd0 : 1'd1 ;
59 assign a_cmp1_13 = (dat1_1 > dat1_3) ? 1'd0 : 1'd1 ;
60 assign a_cmp1_23 = (dat1_2 > dat1_3) ? 1'd0 : 1'd1 ;
61 assign a_cmp2_12 = (dat2_1 > dat2_2) ? 1'd0 : 1'd1 ;
62 assign a_cmp2_13 = (dat2_1 > dat2_3) ? 1'd0 : 1'd1 ;
63 assign a_cmp2_23 = (dat2_2 > dat2_3) ? 1'd0 : 1'd1 ;
64 assign a_cmp3_12 = (dat3_1 > dat3_2) ? 1'd0 : 1'd1 ;
65 assign a_cmp3_13 = (dat3_1 > dat3_3) ? 1'd0 : 1'd1 ;
66 assign a_cmp3_23 = (dat3_2 > dat3_3) ? 1'd0 : 1'd1 ;
67 //-----
68 reg [DATA_WIDTH-1:0] dat1_max, dat1_mid, dat1_min ;
69 reg [DATA_WIDTH-1:0] dat2_max, dat2_mid, dat2_min ;
70 reg [DATA_WIDTH-1:0] dat3_max, dat3_mid, dat3_min ;
71 always @(posedge I_clk)
72 case ({a_cmp1_12, a_cmp1_23, a_cmp1_13})
73     3'b000: {dat1_max, dat1_mid, dat1_min} <= {dat1_1, dat1_2, dat1_3};
74     3'b010: {dat1_max, dat1_mid, dat1_min} <= {dat1_1, dat1_3, dat1_2};
75     3'b011: {dat1_max, dat1_mid, dat1_min} <= {dat1_3, dat1_1, dat1_2};
76     3'b100: {dat1_max, dat1_mid, dat1_min} <= {dat1_2, dat1_1, dat1_3};
77     3'b101: {dat1_max, dat1_mid, dat1_min} <= {dat1_2, dat1_3, dat1_1};
78     3'b111: {dat1_max, dat1_mid, dat1_min} <= {dat1_3, dat1_2, dat1_1};
79     default : {dat1_max, dat1_mid, dat1_min} <= 0; // {3{DATA_WIDTH{1'd0}}};
80 endcase

```



```

81 | always @(posedge I_clk)
82 | case ({a_cmp2_12,a_cmp2_23,a_cmp2_13})
83 |     3'b000: {dat2_max, dat2_mid, dat2_min} <= {dat2_1,dat2_2,dat2_3};
84 |     3'b010: {dat2_max, dat2_mid, dat2_min} <= {dat2_1,dat2_3,dat2_2};
85 |     3'b011: {dat2_max, dat2_mid, dat2_min} <= {dat2_3,dat2_1,dat2_2};
86 |     3'b100: {dat2_max, dat2_mid, dat2_min} <= {dat2_2,dat2_1,dat2_3};
87 |     3'b101: {dat2_max, dat2_mid, dat2_min} <= {dat2_2,dat2_3,dat2_1};
88 |     3'b111: {dat2_max, dat2_mid, dat2_min} <= {dat2_3,dat2_2,dat2_1};
89 |     default : {dat2_max, dat2_mid, dat2_min} <= 0; // {3{DATA_WIDTH{1'd0}}} ;
90 | endcase
91 | always @(posedge I_clk)
92 | case ({a_cmp3_12,a_cmp3_23,a_cmp3_13})
93 |     3'b000: {dat3_max, dat3_mid, dat3_min} <= {dat3_1,dat3_2,dat3_3};
94 |     3'b010: {dat3_max, dat3_mid, dat3_min} <= {dat3_1,dat3_3,dat3_2};
95 |     3'b011: {dat3_max, dat3_mid, dat3_min} <= {dat3_3,dat3_1,dat3_2};
96 |     3'b100: {dat3_max, dat3_mid, dat3_min} <= {dat3_2,dat3_1,dat3_3};
97 |     3'b101: {dat3_max, dat3_mid, dat3_min} <= {dat3_2,dat3_3,dat3_1};
98 |     3'b111: {dat3_max, dat3_mid, dat3_min} <= {dat3_3,dat3_2,dat3_1};
99 |     default : {dat3_max, dat3_mid, dat3_min} <= 0; // {3{DATA_WIDTH{1'd0}}} ;
100 | endcase
101 | // -----
102 | wire b_cmp1_12, b_cmp1_23, b_cmp1_13 ;
103 | wire b_cmp2_12, b_cmp2_23, b_cmp2_13 ;
104 | wire b_cmp3_12, b_cmp3_23, b_cmp3_13 ;
105 | assign b_cmp1_12 = (dat1_max > dat2_max ) ? 1'd0:1'd1;
106 | assign b_cmp1_13 = (dat1_max > dat3_max ) ? 1'd0:1'd1;
107 | assign b_cmp1_23 = (dat2_max > dat3_max ) ? 1'd0:1'd1;
108 | assign b_cmp2_12 = (dat1_mid > dat2_mid ) ? 1'd0:1'd1;
109 | assign b_cmp2_13 = (dat1_mid > dat3_mid ) ? 1'd0:1'd1;
110 | assign b_cmp2_23 = (dat2_mid > dat3_mid ) ? 1'd0:1'd1;
111 | assign b_cmp3_12 = (dat1_min > dat2_min ) ? 1'd0:1'd1;
112 | assign b_cmp3_13 = (dat1_min > dat3_min ) ? 1'd0:1'd1;
113 | assign b_cmp3_23 = (dat2_min > dat3_min ) ? 1'd0:1'd1;
114 | // -----
115 | reg [DATA_WIDTH-1:0] b_max_max, b_max_mid, b_max_min ;
116 | reg [DATA_WIDTH-1:0] b_mid_max, b_mid_mid, b_mid_min ;
117 | reg [DATA_WIDTH-1:0] b_min_max, b_min_mid, b_min_min ;
118 | always @(posedge I_clk)
119 | case ({b_cmp1_12,b_cmp1_23,b_cmp1_13})
120 |     3'b000: {b_max_max, b_max_mid, b_max_min} <= {dat1_max,dat2_max,dat3_max};
121 |     3'b010: {b_max_max, b_max_mid, b_max_min} <= {dat1_max,dat3_max,dat2_max};
122 |     3'b011: {b_max_max, b_max_mid, b_max_min} <= {dat3_max,dat1_max,dat2_max};
123 |     3'b100: {b_max_max, b_max_mid, b_max_min} <= {dat2_max,dat1_max,dat3_max};
124 |     3'b101: {b_max_max, b_max_mid, b_max_min} <= {dat2_max,dat3_max,dat1_max};
125 |     3'b111: {b_max_max, b_max_mid, b_max_min} <= {dat3_max,dat2_max,dat1_max};
126 |     default : {b_max_max, b_max_mid, b_max_min} <= 0; // {3{DATA_WIDTH{1'd0}}} ;
127 | endcase
128 | always @(posedge I_clk)
129 | case ({b_cmp2_12,b_cmp2_23,b_cmp2_13})
130 |     3'b000: {b_mid_max, b_mid_mid, b_mid_min} <= {dat1_mid,dat2_mid,dat3_mid};
131 |     3'b010: {b_mid_max, b_mid_mid, b_mid_min} <= {dat1_mid,dat3_mid,dat2_mid};
132 |     3'b011: {b_mid_max, b_mid_mid, b_mid_min} <= {dat3_mid,dat1_mid,dat2_mid};
133 |     3'b100: {b_mid_max, b_mid_mid, b_mid_min} <= {dat2_mid,dat1_mid,dat3_mid};
134 |     3'b101: {b_mid_max, b_mid_mid, b_mid_min} <= {dat2_mid,dat3_mid,dat1_mid};
135 |     3'b111: {b_mid_max, b_mid_mid, b_mid_min} <= {dat3_mid,dat2_mid,dat1_mid};
136 |     default : {b_mid_max, b_mid_mid, b_mid_min} <= 0; // {3{DATA_WIDTH{1'd0}}} ;
137 | endcase

```

```

138 | always @(posedge I_clk)
139 |     case ({b_cmp3_12,b_cmp3_23,b_cmp3_13})
140 |         3'b000: {b_min_max, b_min_mid, b_min_min} <= {dat1_min,dat2_min,dat3_min};
141 |         3'b010: {b_min_max, b_min_mid, b_min_min} <= {dat1_min,dat3_min,dat2_min};
142 |         3'b011: {b_min_max, b_min_mid, b_min_min} <= {dat3_min,dat1_min,dat2_min};
143 |         3'b100: {b_min_max, b_min_mid, b_min_min} <= {dat2_min,dat1_min,dat3_min};
144 |         3'b101: {b_min_max, b_min_mid, b_min_min} <= {dat2_min,dat3_min,dat1_min};
145 |         3'b111: {b_min_max, b_min_mid, b_min_min} <= {dat3_min,dat2_min,dat1_min};
146 |         default : {b_min_max, b_min_mid, b_min_min} <= 0; // {3{DATA_WIDTH{1'd0}}}
147 |     endcase
148 | // -----
149 | reg [DATA_WIDTH-1:0] c_max , c_mid , c_min ;
150 | wire c_cmp_12 , c_cmp_23 , c_cmp_13 ;
151 | assign c_cmp_12 = (b_max_min > b_mid_mid) ? 1'd0:1'd1 ;
152 | assign c_cmp_13 = (b_max_min > b_min_max) ? 1'd0:1'd1 ;
153 | assign c_cmp_23 = (b_mid_mid > b_min_max) ? 1'd0:1'd1 ;
154 | always @(posedge I_clk)
155 |     case ({c_cmp_12,c_cmp_23,c_cmp_13})
156 |         3'b000: {c_max, c_mid, c_min} <= {b_max_min,b_mid_mid,b_min_max};
157 |         3'b010: {c_max, c_mid, c_min} <= {b_max_min,b_min_max,b_mid_mid};
158 |         3'b011: {c_max, c_mid, c_min} <= {b_min_max,b_max_min,b_mid_mid};
159 |         3'b100: {c_max, c_mid, c_min} <= {b_mid_mid,b_max_min,b_min_max};
160 |         3'b101: {c_max, c_mid, c_min} <= {b_mid_mid,b_min_max,b_max_min};
161 |         3'b111: {c_max, c_mid, c_min} <= {b_min_max,b_mid_mid,b_max_min};
162 |         default : {c_max, c_mid, c_min} <= 0; // {3{DATA_WIDTH{1'd0}}}
163 |     endcase
164 | // -----
165 | assign O_mid_dat = c_mid ;
166 | assign O_mid_vaild = line_vaild[3];
167 | endmodule

```

代码解释：

- 1, 将输入的数据打了一拍，总共花费了 4 个 clock。
- 2, 此程序优化了其他开源程序的是，每次三个数比较大小总共只比较了 3 次。并且在数据赋值的时候用了 case，而不是 if -else。
- 3, 命名方式：a_cmp_xx, b_cmp_xx, c_cmp_xx 分别对应这运算中的三次比较。max, mid, min 分别代表着最大值，中间值，最小值。
- 4, 为什么 case 中 max, mid, min 的赋值顺序是这样子的呢？比如：
{dat1_max, dat1_mid, dat1_min} <= {dat1_1,dat1_2,dat1_3};
这是因为根据 cmp 的真值表而来，这里有兴趣的可以自己列一下，这里就不列举了。

1.4.6 dpc_dat_judge 设计

这个模块就是一个判定功能，依据返回的差值，判定是否需要替换掉中间的那个数据。

```
24 // | Version | Designer | Update
25 // |-----|-----|
26 // | 00      | CobbPeng | File Created.
27 // |-----|-----|
28 // |         |         |
29 // |-----|-----|
30
31
32 module dpc_dat_judge #(
33     parameter DATA_WIDTH      = 8
34 )
35 (
36     input                clk
37     input [7:0]          I_dpc_thrd_value
38     input [3*DATA_WIDTH+2:0] I_line3_1
39     input [3*DATA_WIDTH+2:0] I_line3_2
40     input [3*DATA_WIDTH+2:0] I_line3_3
41     input                I_dat_vaild
42     input [DATA_WIDTH-1:0] I_mid_dat
43     output [DATA_WIDTH-1:0] O_judge_dat
44     output                O_judge_vaild
45 );
46 // ----- FIRST BL
47 reg [DATA_WIDTH:0] dat1r0,dat2r0,dat3r0,dat4r0,dat5r0,dat6r0,dat7r0,dat8r0,dat9r0;
48 reg [7:0]          thr1r0;
49 wire [7:0]         sig_bit ;
50 wire              comp_en ;
51 always @(posedge clk) {dat1r0,dat2r0,dat3r0,dat4r0,dat5r0,dat6r0,dat7r0,dat8r0,dat9r0}
52                               <= {I_line3_1,I_line3_2,I_line3_3};
53 always @(posedge clk) thr1r0<= I_dpc_thrd_value;
54 assign sig_bit = { dat1r0[DATA_WIDTH],dat2r0[DATA_WIDTH],dat3r0[DATA_WIDTH],dat4r0[DATA_WIDTH],
55                   dat6r0[DATA_WIDTH],dat7r0[DATA_WIDTH],dat8r0[DATA_WIDTH],dat9r0[DATA_WIDTH]};
56 assign comp_en = ((sig_bit==8'd0) | (sig_bit==8'hff)) ? 1'd1:1'd0 ;
57 // ----- SECON
58 reg [DATA_WIDTH-1:0] dat1r1,dat2r1,dat3r1,dat4r1,dat6r1,dat7r1,dat8r1,dat9r1;
59 reg [7:0]          thr1r1,thr2r1,thr3r1,thr4r1,thr6r1,thr7r1,thr8r1,thr9r1;
60 reg                comp_enr ;
61 always @(posedge clk) {thr1r1,thr2r1,thr3r1,thr4r1,thr6r1,thr7r1,thr8r1,thr9r1} <= {8{thr1r0}};
62 always @(posedge clk) comp_enr<= comp_en ;
63 always @(posedge clk)
64     if(dat1r0[DATA_WIDTH]) dat1r1 <= ~dat1r0[DATA_WIDTH-1:0];
65     else dat1r1 <= dat1r0[DATA_WIDTH-1:0];
66 always @(posedge clk)
67     if(dat2r0[DATA_WIDTH]) dat2r1 <= ~dat2r0[DATA_WIDTH-1:0];
68     else dat2r1 <= dat2r0[DATA_WIDTH-1:0];
69 always @(posedge clk)
70     if(dat3r0[DATA_WIDTH]) dat3r1 <= ~dat3r0[DATA_WIDTH-1:0];
71     else dat3r1 <= dat3r0[DATA_WIDTH-1:0];
72 always @(posedge clk)
73     if(dat4r0[DATA_WIDTH]) dat4r1 <= ~dat4r0[DATA_WIDTH-1:0];
74     else dat4r1 <= dat4r0[DATA_WIDTH-1:0];
75 always @(posedge clk)
76     if(dat6r0[DATA_WIDTH]) dat6r1 <= ~dat6r0[DATA_WIDTH-1:0];
77     else dat6r1 <= dat6r0[DATA_WIDTH-1:0];
78 always @(posedge clk)
79     if(dat7r0[DATA_WIDTH]) dat7r1 <= ~dat7r0[DATA_WIDTH-1:0];
80     else dat7r1 <= dat7r0[DATA_WIDTH-1:0];
81 always @(posedge clk)
82     if(dat8r0[DATA_WIDTH]) dat8r1 <= ~dat8r0[DATA_WIDTH-1:0];
83     else dat8r1 <= dat8r0[DATA_WIDTH-1:0];
```

```

84 always @(posedge clk)
85     if(dat9r0[DATA_WIDTH])      dat9r1 <= ~dat9r0[DATA_WIDTH-1:0];
86     else                        dat9r1 <=  dat9r0[DATA_WIDTH-1:0];
87 // -----
88 wire    [7:0]    flg;
89 assign  flg[0] = dat1r1 > thr1r1 ? 1'd1:1'd0 ;
90 assign  flg[1] = dat2r1 > thr2r1 ? 1'd1:1'd0 ;
91 assign  flg[2] = dat3r1 > thr3r1 ? 1'd1:1'd0 ;
92 assign  flg[3] = dat4r1 > thr4r1 ? 1'd1:1'd0 ;
93 assign  flg[4] = dat6r1 > thr6r1 ? 1'd1:1'd0 ;
94 assign  flg[5] = dat7r1 > thr7r1 ? 1'd1:1'd0 ;
95 assign  flg[6] = dat8r1 > thr8r1 ? 1'd1:1'd0 ;
96 assign  flg[7] = dat9r1 > thr9r1 ? 1'd1:1'd0 ;
97 // -----
98 reg  [DATA_WIDTH-1:0]    new_dat ;
99 reg  [DATA_WIDTH-1:0]    dat5r1 ;
100 reg  [DATA_WIDTH-1:0]    midr0, midr1 ;
101 reg  [2:0]                vaild_r ;
102 wire                    judge_flag ;
103 always @(posedge clk) dat5r1      <= dat5r0;
104 always @(posedge clk) {midr1, midr0 } <= {midr0 ,I_mid_dat};
105 always @(posedge clk) vaild_r      <= {vaild_r[1:0],I_dat_vaild};
106 always @(posedge clk)
107     if(judge_flag)                new_dat <= midr1 ;
108     else                          new_dat <= dat5r1 ;
109 assign judge_flag = comp_enr&(flg==8'hff) ;
110 assign O_judge_dat    = new_dat ;
111 assign O_judge_vaild  = vaild_r[2] ;
112
113 endmodule

```

代码解释:

- 1, 56行判定数据的正负类型。54-55两行, 将数据的符号位都提取出来作为判定的依据。
- 2, 有没有人对61行的操作感到迷惑的, 这就是改善时序的逻辑复制。这种一个数据, 需要在多个地方用到从而进行逻辑复制的操作, 在我的代码里还是比较常见的。因为这种扇出不大, 但是可能会影响时序的操作, 如果出现时序违例, 是没有办法通过限制扇出系数来解决的。
- 2, 63-86行, 绝对值操作。没有+1? 大丈夫不要在意这么多细节。大家都没有+1, 那就等于都+1咯。
- 3, 109行的 judge_flag 来判定, 是否需要替换中间值。

至此, DPC 的所有可综合代码都将设计完毕,

1.5 代码性能

用最少的代码实现最复杂的功能，代码性能是最重要的指标之一。主要看两个方面，第一是资源占用情况，资源占用越少越好。第二是时序收敛，代码能跑的速率越高越好。当然大家都知道 FPGA 中有速度与面积互换的情况，所以有时候，需要在面积和速率下做一个平衡。这种情况是你的代码架构和设计思想已经很优化了，实际工程中遇到的却是，改变一下设计思想，资源应用的更少，而代码性能更优，面积和速率的优化两者兼得。

下面我们看看在实际工程中上述代码的具体表现，平台如下所示，即为 MPSOC 里面的 4EV 平台，-2 的速率，商用级别芯片。

Hardware : xczu4ev-sfvc784-2-e
software : vivado 2022.1

1.5.1 utilization 概览

首先看看 vivado 2022.1 给出的 utilization

	LUT	Block Ram	DSP
dpc_top	660	2	1
line_buffer	246	2	0
dpc_delta_5x5	412	0	1
delta_3x3	56	0	1
mid_3x3	188	0	0
dpc_dat_judge	91	0	0

表 1.5.1.1 DPC 模块资源占用表

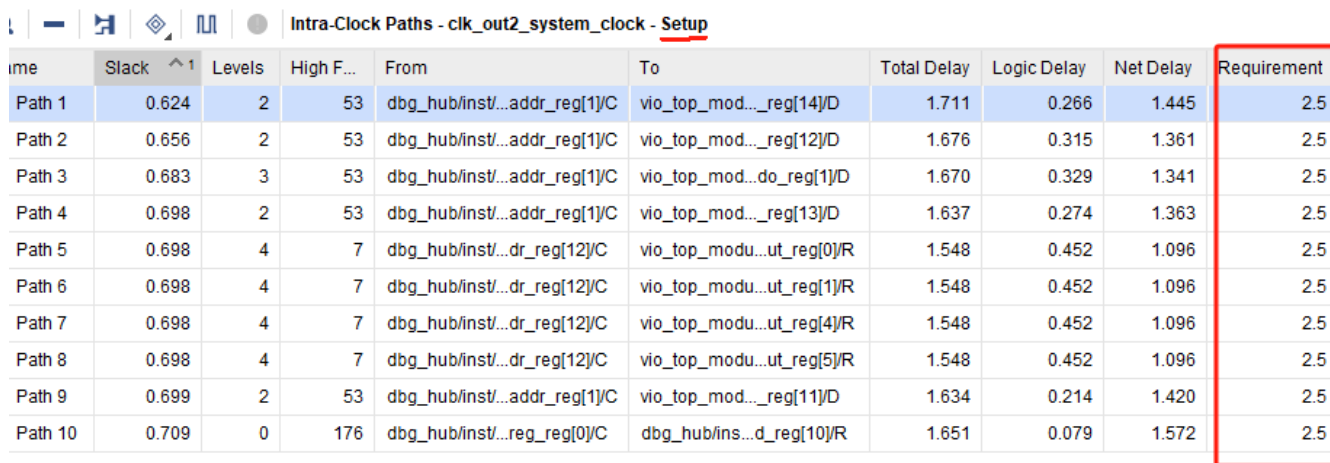
总资源占用 LUT 0.66k。在 1920*1080 的视频流下，缓存 4 行总占用 Bram 2 块，运算占用了 1 块，这一块 DSP 是故意占用的，注意看代码中有一个地方调用了 DSP，其他用 LUT 运算。DSP 在算法相关领域经常是珍贵资源。

有一个误区，就是有些人评估工作量，或者来说是评估这个模块的难易程度，根据资源占用，根据代码有多少行来判定。但这从不是用来评估高级工程师的指标，相反，是错误的指标。如果不进行优化，上述 DCP 模块资源占用率轻松翻 2-3 倍，代码行数也轻松翻 2-3 倍，届时在下一节时序收敛上，就没有这么轻松了。所以有时在 RTL 设计时，鱼（资源占用最少）与熊掌（运行的速率最快）是可以兼得的。

1.5.2 时序收敛

时序收敛在 RTL 设计中至关重要，它是代码在满足功能的前提下最能表明一个代码质量的最关键因素，比资源占用率的重要性更为突出。比如在有些设计中，为了让代码可以满足更高的时钟速率，而不得不牺牲资源来保障时序。现在就让我们来看看上述开源代码能抗住什么样的时序暴揍。

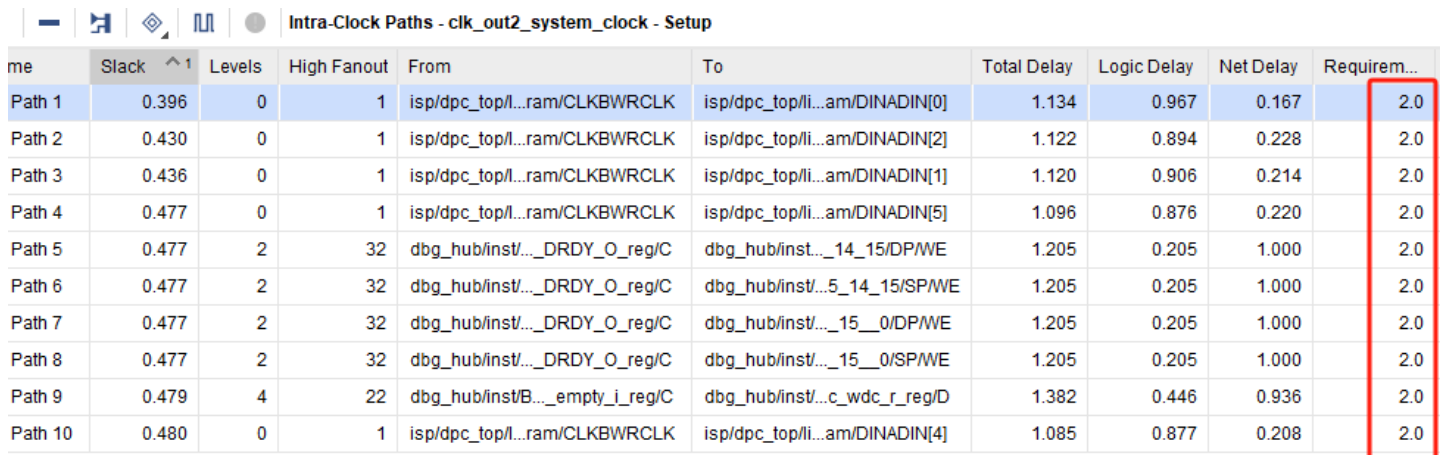
1.5.2.1 当时钟设定在 400M，也就是我通常按照项目实际要求最高频率的两倍去约束我的代码。setup 还能扛得住。setup 的 requirement 代表着 1 个时钟周期（在非多周期路径下），在时序 summary 下，vivado 默认给出前面 10 条最差时序裕量路径。从 slack 上来看，可以跑 500M 以上。



Path	Slack	Levels	High F...	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.624	2	53	dbg_hub/inst/...addr_reg[1]/C	vio_top_mod..._reg[14]/D	1.711	0.266	1.445	2.5
Path 2	0.656	2	53	dbg_hub/inst/...addr_reg[1]/C	vio_top_mod..._reg[12]/D	1.676	0.315	1.361	2.5
Path 3	0.683	3	53	dbg_hub/inst/...addr_reg[1]/C	vio_top_mod...do_reg[1]/D	1.670	0.329	1.341	2.5
Path 4	0.698	2	53	dbg_hub/inst/...addr_reg[1]/C	vio_top_mod..._reg[13]/D	1.637	0.274	1.363	2.5
Path 5	0.698	4	7	dbg_hub/inst/...dr_reg[12]/C	vio_top_modu...ut_reg[0]/R	1.548	0.452	1.096	2.5
Path 6	0.698	4	7	dbg_hub/inst/...dr_reg[12]/C	vio_top_modu...ut_reg[1]/R	1.548	0.452	1.096	2.5
Path 7	0.698	4	7	dbg_hub/inst/...dr_reg[12]/C	vio_top_modu...ut_reg[4]/R	1.548	0.452	1.096	2.5
Path 8	0.698	4	7	dbg_hub/inst/...dr_reg[12]/C	vio_top_modu...ut_reg[5]/R	1.548	0.452	1.096	2.5
Path 9	0.699	2	53	dbg_hub/inst/...addr_reg[1]/C	vio_top_mod..._reg[11]/D	1.634	0.214	1.420	2.5
Path 10	0.709	0	176	dbg_hub/inst/...reg_reg[0]/C	dbg_hub/ins...d_reg[10]/R	1.651	0.079	1.572	2.5

图 1.5.2.1 400M 时钟速率下的 setup 关键路径

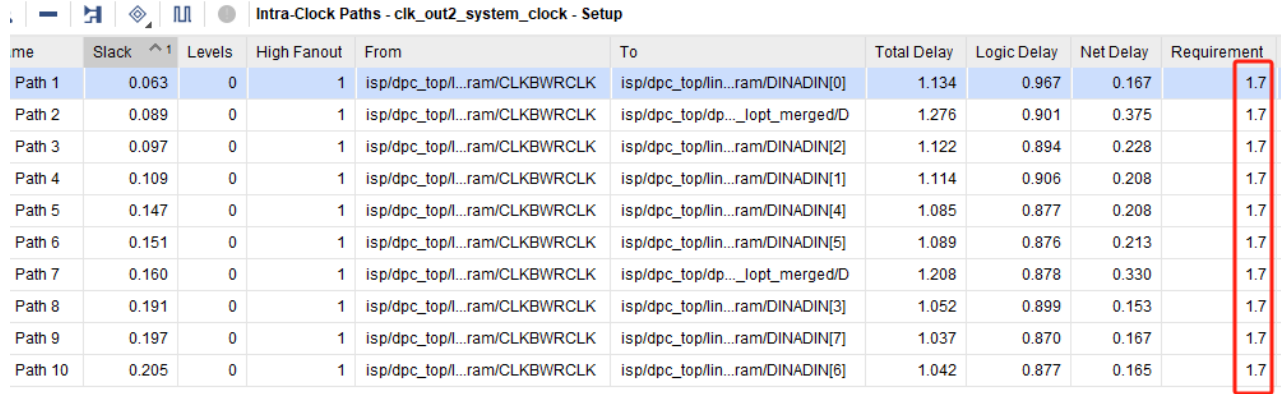
1.5.2.2 当时钟设定在 500M，setup 还能扛得住。从 slack 上看，大概在 1.6ns 的附近是它的极限。不逼 vivado 一把，怎么能挖掘它的潜能呢，于是继续逼近极限。



Path	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement...
Path 1	0.396	0	1	isp/dpc_top/li...ram/CLKBWRCLK	isp/dpc_top/li...am/DINADIN[0]	1.134	0.967	0.167	2.0
Path 2	0.430	0	1	isp/dpc_top/li...ram/CLKBWRCLK	isp/dpc_top/li...am/DINADIN[2]	1.122	0.894	0.228	2.0
Path 3	0.436	0	1	isp/dpc_top/li...ram/CLKBWRCLK	isp/dpc_top/li...am/DINADIN[1]	1.120	0.906	0.214	2.0
Path 4	0.477	0	1	isp/dpc_top/li...ram/CLKBWRCLK	isp/dpc_top/li...am/DINADIN[5]	1.096	0.876	0.220	2.0
Path 5	0.477	2	32	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/..._14_15/DPWE	1.205	0.205	1.000	2.0
Path 6	0.477	2	32	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/..._5_14_15/SPWE	1.205	0.205	1.000	2.0
Path 7	0.477	2	32	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/..._15_0/DPWE	1.205	0.205	1.000	2.0
Path 8	0.477	2	32	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/..._15_0/SPWE	1.205	0.205	1.000	2.0
Path 9	0.479	4	22	dbg_hub/inst/B...empty_i_reg/C	dbg_hub/inst/...c_wdc_r_reg/D	1.382	0.446	0.936	2.0
Path 10	0.480	0	1	isp/dpc_top/li...ram/CLKBWRCLK	isp/dpc_top/li...am/DINADIN[4]	1.085	0.877	0.208	2.0

图 1.5.2.2 500M 时钟速率下的 setup 关键路径

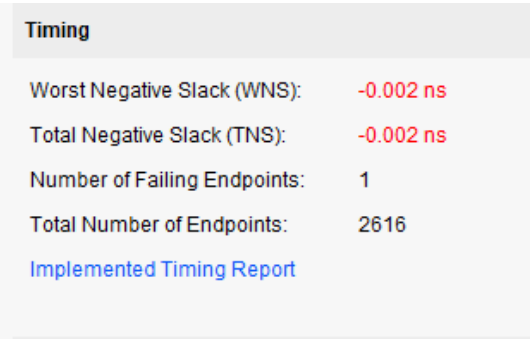
1.5.2.3 当时钟设定在 600M，slack 岌岌可危了。



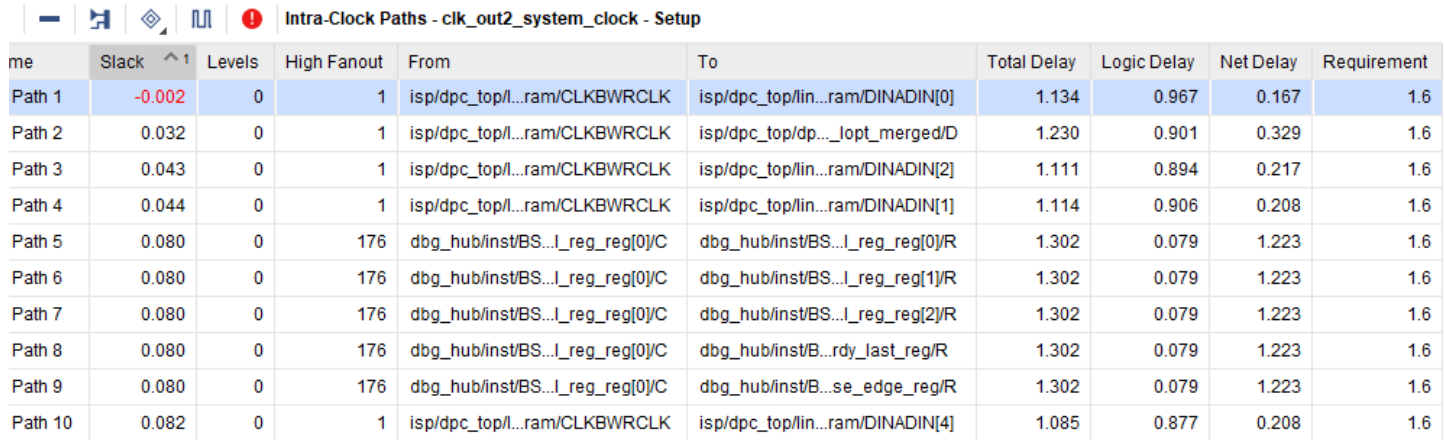
me	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.063	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[0]	1.134	0.967	0.167	1.7
Path 2	0.089	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/dp...lopt_merged/D	1.276	0.901	0.375	1.7
Path 3	0.097	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[2]	1.122	0.894	0.228	1.7
Path 4	0.109	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[1]	1.114	0.906	0.208	1.7
Path 5	0.147	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[4]	1.085	0.877	0.208	1.7
Path 6	0.151	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[5]	1.089	0.876	0.213	1.7
Path 7	0.160	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/dp...lopt_merged/D	1.208	0.878	0.330	1.7
Path 8	0.191	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[3]	1.052	0.899	0.153	1.7
Path 9	0.197	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[7]	1.037	0.870	0.167	1.7
Path 10	0.205	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[6]	1.042	0.877	0.165	1.7

图 1.5.2.3 600M 时钟速率下的 setup 关键路径

1.5.2.4 当时钟设定在 625M，终于扛不住暴揍，多次布局布线（没有调整综合指令）都绕不开这个时序违例。从路径上看是 xilinx RAM（FIFO 的内部就是 RAM）IP 中的写端口是关键路径，它有 -0.002ns 的时序违例。因为关键路径不是我写的代码，我也“束手无策”。有些情况下调整综合指令，大概率还是能修复部分时序违例的，但是通常不用，因为它会导致工程移植一致性变差。



Timing	
Worst Negative Slack (WNS):	-0.002 ns
Total Negative Slack (TNS):	-0.002 ns
Number of Failing Endpoints:	1
Total Number of Endpoints:	2616
Implemented Timing Report	



me	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	-0.002	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[0]	1.134	0.967	0.167	1.6
Path 2	0.032	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/dp...lopt_merged/D	1.230	0.901	0.329	1.6
Path 3	0.043	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[2]	1.111	0.894	0.217	1.6
Path 4	0.044	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[1]	1.114	0.906	0.208	1.6
Path 5	0.080	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[0]/R	1.302	0.079	1.223	1.6
Path 6	0.080	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[1]/R	1.302	0.079	1.223	1.6
Path 7	0.080	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[2]/R	1.302	0.079	1.223	1.6
Path 8	0.080	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/B...rdy_last_reg/R	1.302	0.079	1.223	1.6
Path 9	0.080	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/B...se_edge_reg/R	1.302	0.079	1.223	1.6
Path 10	0.082	0	1	isp/dpc_top/lin...ram/CLKBWRCLK	isp/dpc_top/lin...ram/DINADIN[4]	1.085	0.877	0.208	1.6

图 1.5.2.4 625M 时钟速率下的 setup 关键路径

下面我们来做事序分析，看看这条时序违例的关键路径，选中路径，按 F4，出现内部关键路径示意图，图太大，先给大家看一个预览图，这是两块 ram 之间传递数据的时候出现了时序违例。

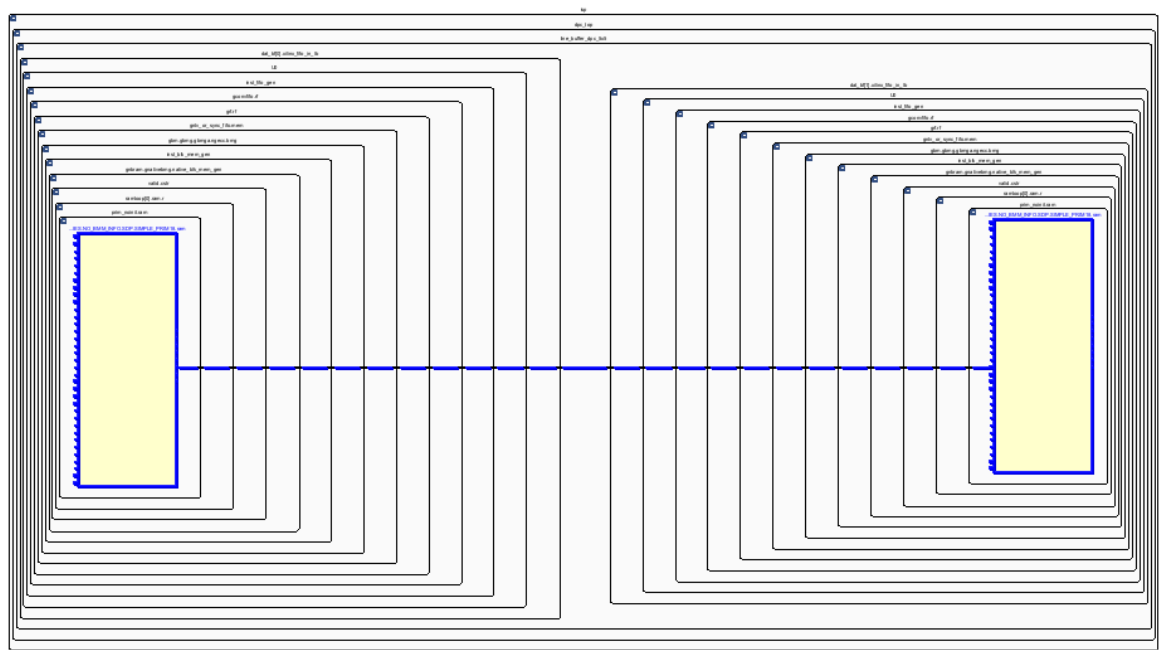


图 1.5.2.5 625M 时钟速率下的关键路径内部图

放大图片顶部层级关系，可以看到是 LineBuffer 中 FIFO0 与 FIFO1 之间传输数据的时候出现的时序违例。也就是说，在上一个 FIFO 读出数据给下一级 FIFO 的时候，这条路径似乎有点远。

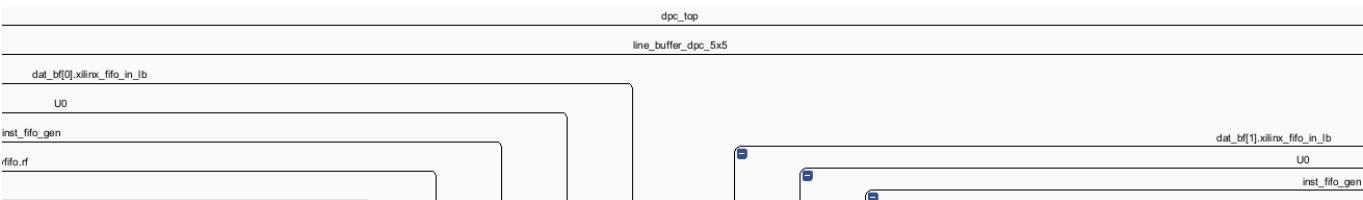


图 1.5.2.6 625M 时钟速率下的关键路径层级图

那么有没有办法解决呢，当然是有呀。

1.5.2.5 上一节说到时序违例的地方是两个 ram 之间传递数据之时时序性能不满足，那么就打开我的 fifo IP，在传递数据的时候中间打一拍。将原来的 latency 1 改成了 2，就是在输出之后打一拍了。设定如下图。

BasicNative PortsStatus FlagsData CountsSummary

Read Mode

☒ Standard FIFO
☐ First Word Fall Through

Data Port Parameters

☐ Asymmetric Port Width

Write Width

8

1,2,3...1024

Write Depth

2048

Actual Write Depth:

Read Width

8

Read Depth

2048

Actual Read Depth:

ECC, Output Register and Power Gating Options

☐ ECC

Hard ECC

☐ Sing
☐ ECC Pipeline Reg
☐ Dyn:
☒ Output Registers

Embec

BasicNative PortsStatus FlagsData CountsSummary

Block RAM resource(s) (18K BRAMs): 1

Block RAM resource(s) (36K BRAMs): 0

Clocking Scheme	Common Clock
Memory Type	Block RAM
Model Generated	Behavioral Model
Write Width	8
Write Depth	2048
Read Width	8
Read Depth	2048
Almost Full/Empty Flags	Not Selected/Not Selected
Programmable Full/Empty Flags	Not Selected/Not Selected
Data Count Outputs	Not Selected
Handshaking	Not Selected
Read Mode / Reset	Standard FIFO / Synchronous
Read Latency (From Rising Edge of Read Clock)	2

图 1.5.2.7 fifo 重新设定图

设定之后，再综合。果然在 625M 之下时序违例消失了。从 slack 裕量上看，还可以提提速。

Intra-Clock Paths - clk_out2_system_clock - Setup									
me	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.172	2	3	dbg_hub/inst/B...al_mode_reg/C	dbg_hub/inst/BS...m_full_i_reg/D	1.318	0.266	1.052	1.6
Path 2	0.229	3	22	dbg_hub/inst/BS...tate_reg[12]/C	dbg_hub/inst/BS...st_r_reg[0]/CE	1.163	0.382	0.781	1.6
Path 3	0.230	4	23	dbg_hub/inst/BS...tate_reg[18]/C	dbg_hub/inst/BS...sl_drdy_reg/D	1.246	0.531	0.715	1.6
Path 4	0.231	5	24	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/B...ec_wdc_r_reg/D	1.242	0.416	0.826	1.6
Path 5	0.238	5	24	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/B...addr_r_reg/D	1.247	0.363	0.884	1.6
Path 6	0.243	3	22	dbg_hub/inst/BS...tate_reg[12]/C	dbg_hub/inst/...l_mode_reg/CE	1.154	0.382	0.772	1.6
Path 7	0.247	5	24	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/BS...err_r_reg[1]/D	1.227	0.403	0.824	1.6
Path 8	0.254	5	24	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/BS...err_r_reg[0]/D	1.230	0.411	0.819	1.6
Path 9	0.262	0	1	dbg_hub/inst/B...IN_O_reg[0]/C	dbg_hub/inst/B...15_0_13/RAMA/I	1.157	0.079	1.078	1.6
Path 10	0.264	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/B...reg_reg[12]/R	1.127	0.080	1.047	1.6

图 1.5.2.8 第二次 625M 时钟速率下的 setup 关键路径

设定了 662.5M，时序裕量还不错，还可以继续提速。别忘了，FIFO 的输出 latency 由原来的 1 变成了 2，那么外面的逻辑记得也需要修改做到时序延迟一致。

Intra-Clock Paths - clk_out2_system_clock - Setup									
me	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.231	0	3	isp/dpc_top/lin...unt_d1_reg[6]/C	isp/dpc_top/li...DDRARDADDR[9]	0.918	0.079	0.839	1.5
Path 2	0.235	5	24	dbg_hub/inst/B...DGE.flag_reg/C	dbg_hub/inst/B...ec_wdc_r_reg/D	1.133	0.363	0.770	1.5
Path 3	0.247	3	22	dbg_hub/inst/B...timeout_reg/C	dbg_hub/inst/...l_mode_reg/CE	1.044	0.425	0.619	1.5
Path 4	0.247	3	22	dbg_hub/inst/B...timeout_reg/C	dbg_hub/inst/BS...st_r_reg[0]/CE	1.044	0.425	0.619	1.5
Path 5	0.250	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[7]/R	1.080	0.079	1.001	1.5
Path 6	0.250	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[9]/R	1.080	0.079	1.001	1.5
Path 7	0.253	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[4]/R	1.079	0.079	1.000	1.5
Path 8	0.253	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[5]/R	1.079	0.079	1.000	1.5
Path 9	0.253	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[6]/R	1.079	0.079	1.000	1.5
Path 10	0.253	0	176	dbg_hub/inst/BS...l_reg_reg[0]/C	dbg_hub/inst/BS...l_reg_reg[8]/R	1.079	0.079	1.000	1.5

图 1.5.2.9 662.5M 时钟速率下的 setup 关键路径

最后设定在了 725M，工程还算稳定的跑过去了。

Intra-Clock Paths - clk_out2_system_clock - Setup									
me	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.060	2	19	dbg_hub/...reg[3]/C	vio_top...[14]/D	1.247	0.314	0.933	1.4
Path 2	0.077	4	22	dbg_hub/...i_reg/C	dbg_h...eg/D	1.184	0.535	0.649	1.4
Path 3	0.093	3	19	dbg_hub/...reg[3]/C	vio_to...g[6]/D	1.215	0.309	0.906	1.4
Path 4	0.097	4	22	dbg_hub/...i_reg/C	dbg_h...reg/D	1.164	0.414	0.750	1.4
Path 5	0.107	2	19	dbg_hub/...reg[3]/C	vio_top...[13]/D	1.202	0.371	0.831	1.4
Path 6	0.109	3	19	dbg_hub/...reg[3]/C	vio_to...g[7]/D	1.206	0.325	0.881	1.4
Path 7	0.113	3	19	dbg_hub/...reg[3]/C	vio_to...g[8]/D	1.189	0.383	0.806	1.4
Path 8	0.115	3	19	dbg_hub/...reg[3]/C	vio_to...g[2]/D	1.187	0.311	0.876	1.4
Path 9	0.123	0	3	isp/dpc_...reg[9]/C	isp/d...R[12]	0.889	0.079	0.810	1.4
Path 10	0.126	3	53	dbg_hub/...reg[1]/C	vio_to...g[4]/D	1.181	0.328	0.853	1.4

图 1.5.2.10 725M 时钟速率下的 setup 关键路径

1.5.3 性能概述

模块性能概述：在资源充裕的情况之下，dpc 整个模块在 4EV 这个平台（vivado2022.1）上跑到了最佳 600M 的速率（不做时序优化的时候），只占用了 0.66k 的 LUT 资源。

实际应用中，满足 1920*1080p 60 的视频流，时钟仅需要 148.5M。此模块设计远远超过了实际所需。为了让代码稳定性更好，通常采用过约束的方式，让代码更加健硕。过约束会让你的时序裕量 slack 更大，抗住高温高湿等不确定因素的干扰。不过除了过约束，在实际项目应用中，还应该让代码具有可测性，自测性，以及一定的容错机制。这个在以后的课程中依托于具体的项目展开来讲。

1.6 仿真

仿真也是所有代码中的重中之重。一提到测试，应该想到的是，测试图哪里来，怎么测，需要上板子吗？从算法的角度，其实根本没有必要上板。下面我就一步步的详细说明，仿真还需要结合 Matlab 与 vivado 或者 modelsim。整体仿真架构如下：

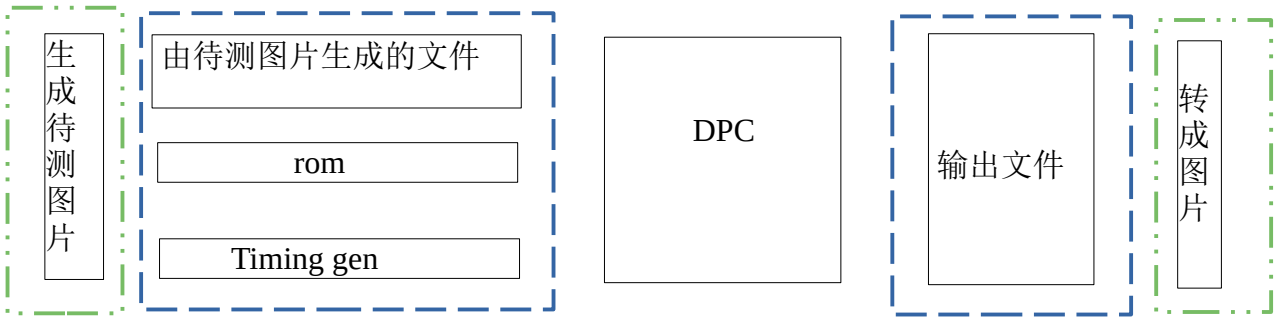


图 1.6.1 仿真框架图

其中绿色虚线框中的是 matlab 代码，蓝色虚线框里面的是测试代码。中间的 DPC 是待测程序。下面分别来介绍每一段代码是如何设计的。

1.6.1 生成待测图片

用随机数生成一个图片，1920*1080 大小，并且从中插入一些异常的点，类似在本章中第一节提到的。

```
1 % -----
2 clc;clear;close all;
3 tic;
4 % -----
5 % patten generate
6 for i=1:1080
7     for j=1:1920
8         rand_dat = randi([120,150]);
9         %insert dead pixels
10        if (i == 100 | i== 200 | i == 300 | i== 400 | i == 500 | i== 600) ...
11            &(j==100 | j == 110)
12
13            rand_dat = 250 ;
14        elseif (i == 1 )&(j==2 )
15            rand_dat = 100 ;
16        elseif (i == 100 | i== 200 | i == 300 | i== 400 | i == 500 | i== 600)...
17            &(j==130 | j == 140)
18
19            rand_dat = 10 ;
20        end
21        patten(i,j)= rand_dat ;
22    end
23 end
24
25 patten = uint8(patten);
26 figure ;
27 imshow(patten);
28 imwrite(patten,'dpc_patten.png');
```

代码解释：

- 1, 8 行, 生成随机数, 当成图像的 raw 数据。
- 2, 10-19 行, 随意添加的几个异常点, 模拟坏点的状态。
- 3, 14-15 行是 debug 时用来定位的, 也可以删除不用。
- 4, imshow 出来的图片是缩略图, 想看具体现象还是看 png 图。坏点部分放大之后显示如下图, 就是下图的白点和黑点。

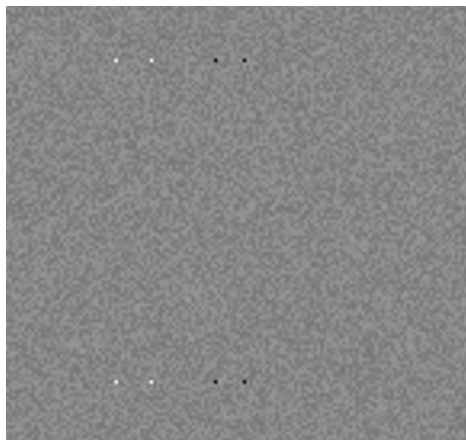


图 1.6.1 坏点部分放大

1.6.2 待测图片生成txt 文件

这一部分就是比较简单, 仅仅是在 matlab 中读入文件, 写入 txt 即可。

```
5  %% file head -----
6  clear all;clc;close all;
7  %% user setting -----
8  img = imread('dpc_patten.png');
9  %% -----
10 [m,n,k]=size(img);
11 u_dat = uint8(img);
12 if k==1
13     print_dat = u_dat';
14     fid = fopen('dpc_patten.txt','wt');
15     for i=1: (m*n)
16         fprintf(fid,'%2x\n',print_dat(i));
17     end
18     fclose(fid);
19     print_img= u_dat ;
20 end
21 %% display -----
22 figure,
23 subplot(1,1,1),imshow(print_img);
24 title('org')
25 %% -----
```

1.6.3 生成VESA 标准视频同步信号

在 1.6.2 中我们得到了一个 txt 文档。将 txt 文档导入到 verilogHDL 中，配合 VESA timing 生成器，就可以生成视频流，达到了类似一个固定帧画面的视频输入的效果。如此就可以只专注于算法部分的 FPGA 代码实现效果，而不用担心板子上的各种视频接口调试，sensor 配置等问题。

视频流生成器虽然在 vivado 里面也有 IP，但还是自己写的调用起来方便。

```
40 module vesa_timing_gen (
41     input          I_video_clk ,
42     input          I_reset_n   ,
43     input          [3:0] I_pattern_format,
44     input          [3:0] I_pattern_colour,
45     output         [29:0] O_pattern_rgb ,
46     output         [2:0]  O_pattern_syn
47 );
48 // -----
49 reg [11:0] h_timing_syn, h_timing_ba, h_timing_ac, h_timing_total ;
50 reg [11:0] v_timing_syn, v_timing_ba, v_timing_ac, v_timing_total ;
51 always @(posedge I_video_clk)
52 if(!I_reset_n) begin
62 else case (I_pattern_format)
63 4'b0000 : begin // ----- 1920*1200 pixl clk
76 4'b0001 : begin // ----- 1920 *1080p60 pixl
77 // syn + back + active + front = total
78 // 44 + 148 + 1920 + 88 = 2200
79 // 5 + 36 + 1080 + 4 = 1125
80 h_timing_syn    <= 12'd44 ;
81 h_timing_ba     <= 12'd192 ; //44 +148
82 h_timing_ac     <= 12'd2112 ; //44 + 148 + 1920
83 h_timing_total  <= 12'd2200 ;
84 v_timing_syn    <= 12'd5 ;
85 v_timing_ba     <= 12'd41 ; // 5 + 36
86 v_timing_ac     <= 12'd1121 ; // 5 + 36 + 1080
87 v_timing_total  <= 12'd1125 ;
88 end
89 4'b0010 : begin // ----- 1280 *1024p60 pixl
102 4'b0011 : begin // ----- 1024* 768 pixl
115 4'b0100 : begin // ----- 1280* 720 pixl
128 default : begin // ----- user define -----
138
139 endcase
140 // -----
141 wire h_timing_out ,v_timing_out;
142 reg [11:0] pix_cnt , hyc_cnt ;
143 always @(posedge I_video_clk)
144 if((!I_reset_n)|h_timing_out) pix_cnt <= 12'd1 ;
145 else pix_cnt <= pix_cnt + 12'd1 ;
```

```

146 always @(posedge I_video_clk)
147     if(!I_reset_n) hyc_cnt <= 12'd1 ;
148     else if (h_timing_out)
149         if (v_timing_out) hyc_cnt <= 12'd1 ;
150         else hyc_cnt <= hyc_cnt + 12'd1 ;
151 assign h_timing_out = (pix_cnt==h_timing_total);
152 assign v_timing_out = (hyc_cnt == v_timing_total);
153 // -----
154 reg line_vaild ;
155 reg d_signal, h_signal, v_signal;
156 always @ (posedge I_video_clk)
157     if (!I_reset_n) line_vaild <= 1'd0 ;
158     else if(hyc_cnt == v_timing_ba) line_vaild <= 1'd1 ;
159     else if(hyc_cnt == v_timing_ac) line_vaild <= 1'd0 ;
160     else line_vaild <= 1'd0 ;
161 always @ (posedge I_video_clk)
162     if(!I_reset_n) d_signal <= 1'b0;
163     else if((pix_cnt == h_timing_ba) & line_vaild) d_signal <= 1'b1;
164     else if((pix_cnt == h_timing_ac) & line_vaild) d_signal <= 1'b0;
165     else d_signal <= 1'b0;
166 always @ (posedge I_video_clk)
167     if(!I_reset_n) h_signal <= 1'b0;
168     else if(h_timing_out) h_signal <= 1'b1;
169     else if((pix_cnt == h_timing_syn)) h_signal <= 1'b0;
170 always @ (posedge I_video_clk )
171     if(!I_reset_n) v_signal <= 1'b0;
172     else if(v_timing_out&h_timing_out) v_signal <= 1'b1;
173     else if((hyc_cnt == v_timing_syn)&(h_timing_out)) v_signal <= 1'b0;
174     else v_signal <= 1'b0;
175 // -----
176 reg syn_v, syn_h , syn_d ;
177 always @ (posedge I_video_clk )
178     {syn_v, syn_h , syn_d} <= {v_signal, h_signal, d_signal};
179
180
181 assign O_pattern_syn = {v_signal, h_signal, d_signal};
182 assign O_pattern_rgb = {10'h3ff, 10'h3ff, 10'h3ff} ;
183 // -----
184 initial begin
185     #100 ;
186     wait (I_reset_n);
187     #100 ;
188     force O_pattern_syn[2] = 0 ;
189     wait(O_pattern_syn[1]==1);
190     wait(O_pattern_syn[1]==0);
191     #500 force O_pattern_syn[2] = 1 ;
192     wait(O_pattern_syn[1]==1);
193     wait(O_pattern_syn[1]==0);
194     wait(O_pattern_syn[1]==1);
195     wait(O_pattern_syn[1]==0);
196     #500 release O_pattern_syn[2] ;
197     end
198
199 endmodule

```

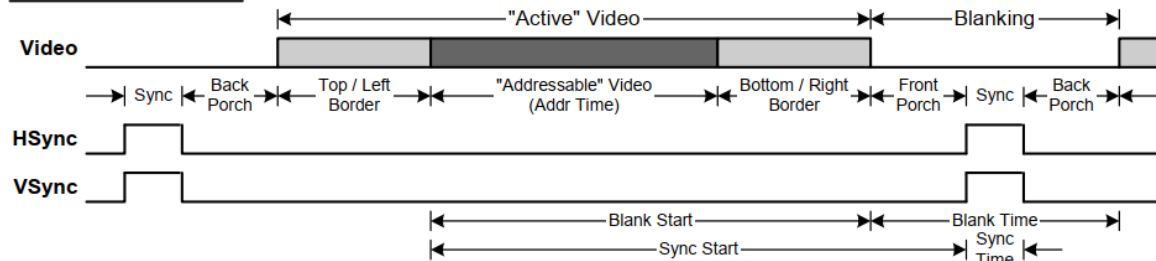
代码解释:

1, 51-139 行在依据外部的 `I_pattern_format` 设定来选取不同的分辨率。`I_pattern_format` 设定的不同, 记住外部的时钟频率也应当与之匹配。

2, 141-175 用来生成 VESA 的同步信号, `V`, `H`, `DE`。都是正向信号。有些 VESA 视频标准的同步信号与我的程序中稍有不同, 比如 `V`, `H`, `DE` 是负的, 这也是可以的。

3.1 DMT Video Timing Parameter Definitions - Positive H & Positive V Syncs:

Definition of Terms



3.5 DMT Video Timing Parameter Definitions - Total Frame Timing:

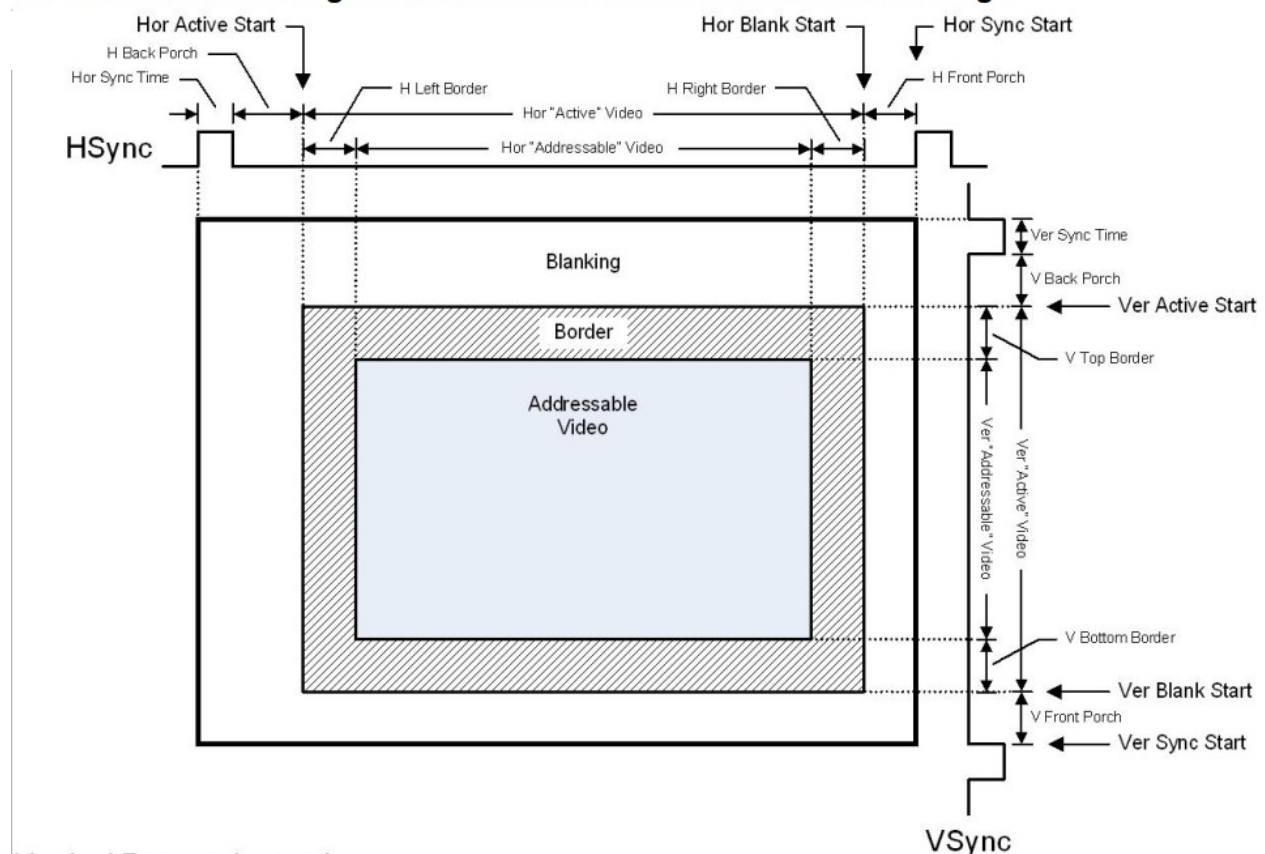


图 1.6.3.1 VESA 视频标准

以上截图来自 VESA 标准说明文档。

1.6.4 用txt 生成视频流

此模块的目的，是将txt 文件导入到HDL 中，然后依据vesa_timing_gen 生成视频流。因为此次导入的是raw 数据，所以只需要一个rom 就可以了。

```
33
34 module file_to_video (
35     input          I_video_clk      ,
36     input          I_reset_n        ,
37     output [2:0]    O_video_syn      ,
38     output [7:0]    O_video_r        ,
39     output [7:0]    O_video_g        ,
40     output [7:0]    O_video_b        ,
41 );
42 // -----
43 wire [2:0] patten_syn ;
44 wire [29:0] patten_rgb ;
45 // -----
46 localparam R_FILE_PATH = "E:/peng/ISP/coms/matlab/dpc_patten.txt" ;
47 localparam G_FILE_PATH = "E:/peng/ISP/coms/matlab/dpc_patten.txt" ;
48 localparam B_FILE_PATH = "E:/peng/ISP/coms/matlab/dpc_patten.txt" ;
49
50
51 // ***** I_pattern_format *****
52 //4'b0000 ---> 1920*1200
53 //4'b0001 ---> 1920*1080p60
54 //4'b0010 ---> 1280*1024p60
55 //4'b0011 ---> 1024*768
56 //4'b0100 ---> 1280*720
57 // ***** I_pattern_colour *****
58 //4'b0000 ---> full colour bar
59 //4'b0001 ---> almost full colour bar
60 //4'b0010 ---> gray bar
61 // -----
62 vesa_timing_gen vesa_timing_gen (
63     //pattern_gen_10bit pattern_gen_10bit(
64         .I_video_clk      (I_video_clk      ),
65         .I_reset_n        (I_reset_n        ),
66         .I_pattern_format  (4'b0001         ),
67         .I_pattern_colour  (4'b0010         ),
68         .O_pattern_rgb     (patten_rgb       ),
69         .O_pattern_syn     (patten_syn       ) // V ,H ,DE
70     );
71 // -----
72 reg [23:0] d_count ;
73 always @(posedge I_video_clk)
74     if(patten_syn[0]) d_count <= d_count + 24'd1; //
75     else if(patten_syn[2]) d_count <= 24'd0 ;
76 // -----
77 wire [7:0] video_r, video_g, video_b;
78 vir_rom #(
79     .ROM_ADDR_WIDTH      (24
80     .ROM_DATA_WIDTH      (8
```



```

81         .ROM_MEMORY_TYPE      ("distributed"      ), // auto
82         .ROM_INITIAL_FILE      ( R_FILE_PATH      )
83     )vir_rom_r(
84         .I_rd_clk               (I_video_clk       ),
85         .I_rd_en                (patten_syn[0]     ),
86         .I_rd_add               (d_count          ),
87         .O_rd_dat               (video_r           )
88     );
89
90
91 ⊞vir_rom #(
102
103 ⊞vir_rom #(
114
115     // -----
116     reg      [2:0]  video_syn ;
117     reg      [29:0] video_patten_rgb ;
118     always @(posedge I_video_clk)video_syn<= patten_syn;
119     always @(posedge I_video_clk)video_patten_rgb<= patten_rgb;
120
121     assign  O_video_syn = video_syn;
122     assign  O_video_r = {video_r} ;
123     assign  O_video_g = {video_g} ; //8'd120; // video_patten_rgb
124     assign  O_video_b = {video_b} ;
125
126 ⊞//
142
143     endmodule

```

代码解释:

- 1, 由 vesa_timing_gen 模块产生的时序, 读出 rom 中的 txt 数据, 形成视频流。
- 2, 由于此次使用的是 raw 图, 所以只用到一个 rom。也就是 78 行代码中的 ram。其他两个 rom 折叠起来了。
- 3, 那么这个 rom 的设计方法如下。把 rom 类型拉到宏定义端口上, 初始化路径也拉入到宏定义端口上。

```

32
33 ⊞module vir_rom #(
34     parameter  ROM_ADDR_WIDTH  = 11,
35     parameter  ROM_DATA_WIDTH  = 8, //
36     parameter  ROM_MEMORY_TYPE = "distributed" , // auto // block
37     parameter  ROM_INITIAL_FILE = "E:/peng/"
38 )
39 ⊞(
40     input              I_rd_clk      ,
41     input              I_rd_en       ,
42     input  [ROM_ADDR_WIDTH-1:0]      I_rd_add    ,
43     output reg [ROM_DATA_WIDTH-1:0]  O_rd_dat
44 );
45     (*rom_style=ROM_MEMORY_TYPE*)
46     reg [ROM_DATA_WIDTH-1:0] mem_rom_gen[2**ROM_ADDR_WIDTH-1:0];

```

```

47 initial begin
48     $readmemh(ROM_INITIAL_FILE, mem_rom_gen);
49 end
50 always @ (posedge I_rd_clk)
51     if(I_rd_en) O_rd_dat <= mem_rom_gen[I_rd_add];
52
53 endmodule

```

1.6.5 视频流生成 txt

那如果要把 HDL 中输出的视频信号转成 txt 呢？当然是可以的。不过设计的时候需要注意：

1，有些算法里面处理完的第一帧不是我们需要的数据，或者第二帧都不是。所以设计的时候需要帧数可以选择。

2，有时候在 debug 的时候，输出了 xx 或者 zz 的不正常数据，需要通知到仿真器的显示界面上，而不用在大量的数据里面去查找是否有 xx 或者 zz。

```

33
34 module video_to_file #(
35     parameter VIDEO_DATA_WIDTH = 11 ,
36     parameter FRAME_PRINT_NUMBER = 2 , // 0-9
37     parameter FILE_NAME_DEFINE = "dat.txt"
38 ) (
39     input I_video_clk ,
40     input I_video_rst_n ,
41     input I_video_v ,
42     input I_video_d ,
43     input [VIDEO_DATA_WIDTH-1:0] I_video_dat
44 );
45 // -----
46 reg syn_v ;
47 wire syn_v_pos ;
48 always @(posedge I_video_clk) syn_v <= I_video_v;
49 assign syn_v_pos = (!syn_v) & I_video_v;
50 // -----
51 integer dat_file ;
52 reg [9:0] frame_select ;
53 always @(posedge I_video_clk)
54     if(!I_video_rst_n) frame_select <= 10'b00_0000_0001;
55     else if(syn_v_pos) frame_select <= frame_select << 1;
56 always @(posedge I_video_clk)
57     if(frame_select[FRAME_PRINT_NUMBER] & I_video_d) begin
58         $fdisplay(dat_file, "%h", I_video_dat);
59         if((I_video_dat == 8'hxx) | (I_video_dat == 8'hzz))
60             $display("data error \n");
61     end

```



```

62 | // -----
63 | initial begin
64 |     wait (frame_select[FRAME_PRINT_NUMBER-1]);
65 |     dat_file = $fopen(FILE_NAME_DEFINE);
66 |     wait (frame_select[FRAME_PRINT_NUMBER+1]);
67 |     $fclose (dat_file);
68 |     end
69 |
70 | endmodule

```

代码解释：

1, 52 行, frame_select 就是用来选择输出帧, 不过选择的帧数越靠后, 仿真所需要的时间就越长, 所以尽量选择靠前的, 节约时间。

2, 59 行, 防止因为出错而输出了错误的符号, 给到仿真器的用户界面上一个提示。txt 中写入了 zz 或者 xx 也没什么问题, 而是在 debug 的时候数据量大了, 到了 matlab 中, 需要等很久才知道 txt 中有非数据的其他字符导致 matlab 中程序运行失败。所以这句话主要是为了节约 debug 成本的。

1.6.6 txt 生成图片

这也是一个很简短的 matlab 小程序, 几行代码就满足条件了。

```

5 | % file to image . for gray
6 | %
7 | %% -----
8 | clear all;clc;close all;
9 | % -----
10 | usr_file = 'video_r_dat.txt' ;
11 | % img size
12 | img_w = 1920;
13 | img_h = 1080;
14 | % -----
15 | %% process -----
16 | usr_dat = textread(usr_file, '%s');
17 | usr_img_dat = hex2dec(usr_dat);
18 | usr_img = reshape(usr_img_dat, img_w, img_h );
19 | usr_img = uint8(usr_img');
20 | imshow(usr_img);title('user')
21 | imwrite(usr_img,'usr_img.jpg');

```

1.6.7 testbench 全貌

综合上面所有为tb准备的代码，然后看看完整的TB张啥样。

```
2  `timescale 1ns/1ps
3  //
4  // *****
5  //
6  //
7  //
8  //
9
10 //  COPYRIGHT      :  COBB.PENG
11 //  URL            :  www..com
12 //  TEMPLATE VERSION :  V00
13 //
14 //
15 //  FileName       :
16 //  ModuleName     :
17 //  HardWare Version :
18 //  Software Version :
19 //  Verilog Version :  verilog 2001
20 //  Target Devices :
21 //  Description    :
22 //
23 //
24 // |-----|
25 // | Version | Designer | Update
26 // |-----|-----|-----|
27 // | 00      | CobbPeng | File Created.
28 // |-----|-----|-----|
29 // |         |         |
30 // |-----|
31
32
33 module isp_tb ;
34 reg clk ;
35 reg video_rst ;
36
37 wire [2:0] video_syn ;
38 wire [7:0] video_raw ;
39 // -----
40 wire [7:0] dpc_thrd_value ;
41 wire [2:0] bayer_format ;
42
43 wire [7:0] blc_r_value ;
44 wire [7:0] blc_gr_value ;
45 wire [7:0] blc_gb_value ;
46 wire [7:0] blc_b_value ;
47
48 assign dpc_thrd_value = 8'd80 ;
49 assign bayer_format = 3'd0 ; //000:BGGR 001:GBGR
```

```

50
51 assign      blc_r_value = 8'd16 ;
52 assign      blc_gr_value = 8'd17 ;
53 assign      blc_gb_value = 8'd18 ;
54 assign      blc_b_value = 8'd15 ;
55 // -----
56 wire [7:0]    video_r, video_g, video_b ;
57 file_to_video file_to_video(
58     .I_video_clk          (clk          ),
59     .I_reset_n            (!video_rst   ),
60     .O_video_syn          (video_syn    ),
61     .O_video_r            (video_r      ),
62     .O_video_g            (video_g      ),
63     .O_video_b            (video_b      )
64 );
65 // -----
66 wire [23:0]    isp_dat ;
67 wire [2:0]     isp_syn ;
68 isp #(
69     .VIDEO_RAWDATA_WIDTH      (8          )
70 ) isp (
71     .I_video_clk              (clk          ),
72     .I_video_rst              (video_rst    ),
73     .I_video_syn              (video_syn    ),
74     .I_video_raw              (video_r      ),
75     // -----
76     .I_bayer_format           (bayer_format[2:0] ),
77     .I_dpc_thrd_value         (dpc_thrd_value ),
78     // -----
79     .I_blc_r_value            (blc_r_value   ),
80     .I_blc_gr_value           (blc_gr_value  ),
81     .I_blc_gb_value           (blc_gb_value  ),
82     .I_blc_b_value            (blc_b_value   ),
83     // -----
84     // -----
85     .O_video_syn              (isp_syn       ),
86     .O_video_dat              (isp_dat       )
87 );
88 // -----
89 video_to_file #(
90     .VIDEO_DATA_WIDTH          (8          ) ,
91     .FRAME_PRINT_NUMBER        (2          ) ,
92     .FILE_NAME_DEFINE          ( "video_r_dat.txt" )
93 ) video_to_file_r (
94     .I_video_clk              (clk          ),
95     .I_video_rst_n            (!video_rst    ),
96     .I_video_v                (isp_syn[2]    ),
97     .I_video_d                (isp_syn[0]    ),
98     .I_video_dat              (isp_dat[7:0]  )
99 );
100 /*
124 // -----
127 always #10 clk = ~clk ;

```

```

128
129
130 initial begin
131     clk = 0 ;    video_rst = 1 ;
132     #100 ;
133     video_rst = 0 ;
134     #80 ;
135     wait (isp_syn[0]);
136     wait (!isp_syn[0]);
137     wait (isp_syn[0]);
138     wait (isp_syn[2]);
139
140     wait (!isp_syn[2]);
141     wait (isp_syn[2]);
142
143     wait (!isp_syn[2]);
144     // wait (isp_syn[2]);
145
146
147     $stop ;
148     end
149     //-----
150
151 endmodule

```

代码解释：

- 1, 100 行和 124 行折叠部分是没有用到的 video_to_file_g, video_to_file_b。
- 2, 130 行的 wait 就是在根据同步信号做延迟。比如 140, 143 行，这三个 wait 的时间就是一帧。

至此，所有的 DCP 代码完毕。

1.6.8 仿真波形图

在上述的仿真代码下，依据仿真图，一步步 debug 自己的程序。

如图在 top 层，一帧的前面 4 行会无法凑齐 5*5 的矩阵，如图 1.6.8.1 所示，当输入到第五行的时候，第一次出现一个完整的 5*5 矩阵框，如图 1.6.8.2 所示。还有人记得我特意放置的 100 这个参数吗？

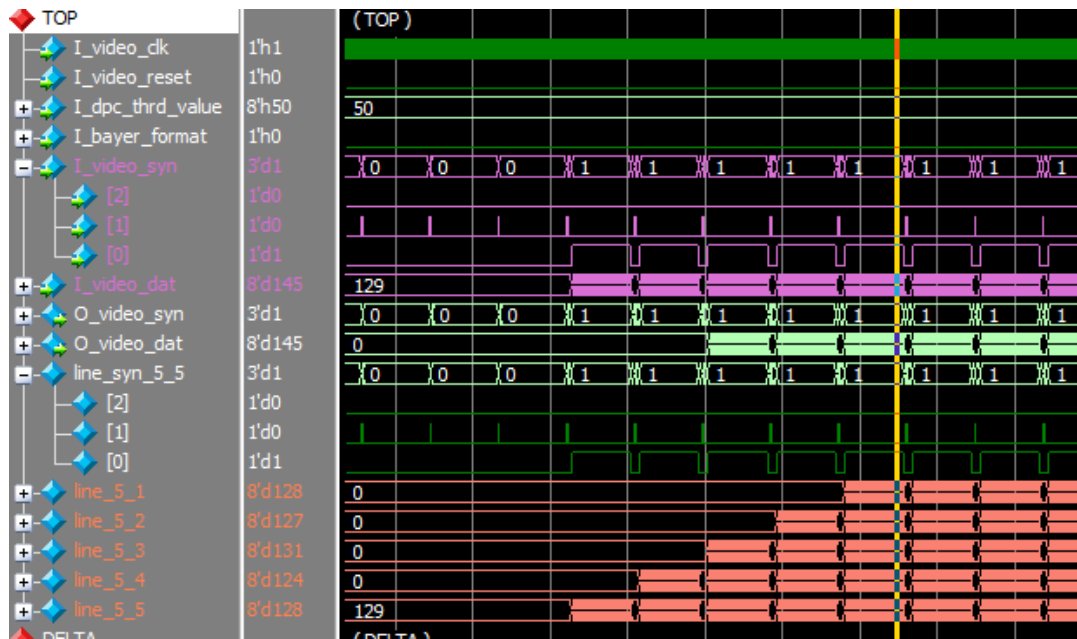


图 1.6.8.1 top 层仿真波形图

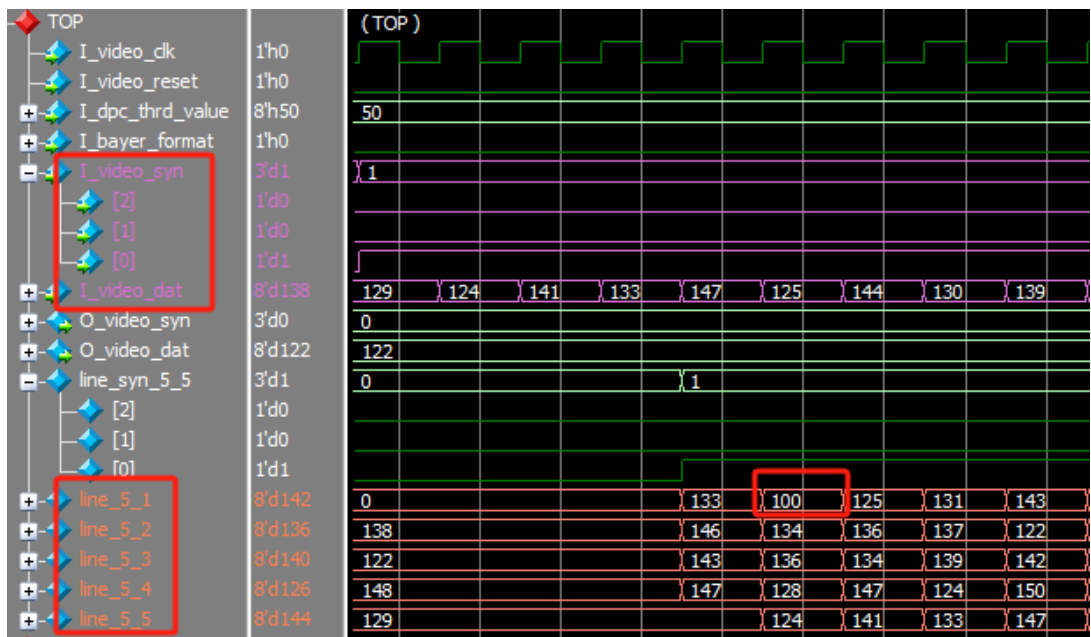


图 1.6.8.2 5*5 矩阵波形图

在 1.6.1 中可以看出我一共放置了 6 组坏点，每组坏点有四个。来看仿真概览图，如下图 1.6.8.3 所示。

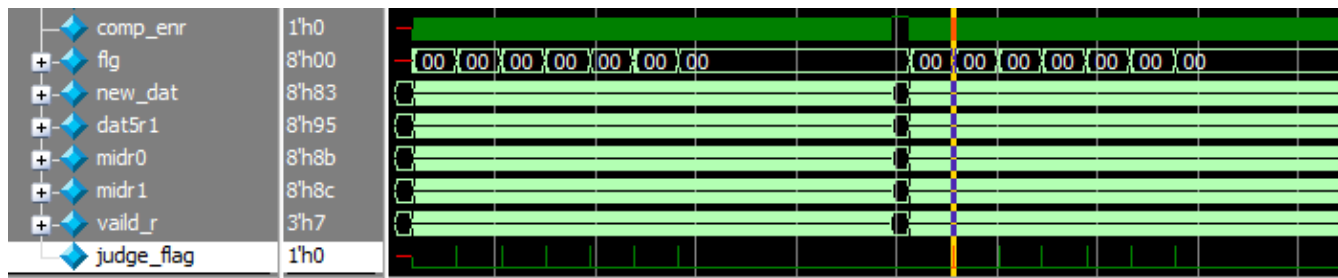


图 1.6.8.3 每帧的坏点

这是两帧的仿真缩略图，可以看到每帧有六个发现坏点的地方。放大光标的位置可以看到每个坏点的地方是 4 组数据，如图 1.6.8.4 所示：

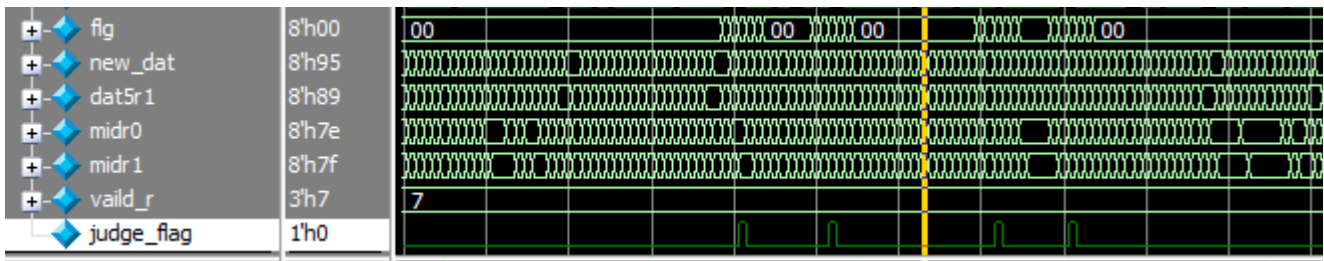


图 1.6.8.4 一组坏点波形图放大

当然，1.6.1 中的坏点生成程序，对于坏点的插入位置，可以自行调整。最后将仿真得到的 txt 文档导入到 matlab 中，再生成图片，如此，DPC 从算法实现，到 FPGA 实现，仿真，完美收工。

1.7 后记

1.7.1 关于时序收敛

在 1.5 中优化后，当我把时钟频率调到 762.5M 的时候，跑到了接近这个芯片的频率的极限，也就是接近了 FPGA 时钟网络的极限。时序分析显示没有 setup，hold 违例，只有时钟脉宽违例。

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):		0.058 ns	Worst Hold Slack (WHS):		0.009 ns
Total Negative Slack (TNS):		0.000 ns	Total Hold Slack (THS):		0.000 ns
Number of Failing Endpoints:		0	Number of Failing Endpoints:		0
Total Number of Endpoints:		2641	Total Number of Endpoints:		2625
			Worst Pulse Width Slack (WPWS):		-0.044 ns
			Total Pulse Width Negative Slack (TPWS):		-0.174 ns
			Number of Failing Endpoints:		4
			Total Number of Endpoints:		1524

Intra-Clock Paths - clk_out2_system_clock - Setup

Time	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.058	5	23	dbg_hub/inst/...DRDY_O_reg/C	dbg_hub/inst/B...ec_wdc_r_reg/D	1.174	0.525	0.649	1.3
Path 2	0.059	4	24	dbg_hub/inst/BS..._state_reg[6]/C	dbg_hub/inst/BS..._sl_drdy_reg/D	1.146	0.476	0.670	1.3
Path 3	0.063	4	24	dbg_hub/inst/BS..._state_reg[6]/C	dbg_hub/inst/B..._addr_r_reg/D	1.140	0.499	0.641	1.3
Path 4	0.064	3	22	dbg_hub/inst/B.../timeout_reg/C	dbg_hub/inst/BS...st_r_reg[0]/CE	1.032	0.300	0.732	1.3
Path 5	0.065	3	1	dbg_hub/inst/BS...l_reg_reg[1]/C	dbg_hub/inst/BS...addr_reg[12]/D	1.093	0.393	0.700	1.3
Path 6	0.066	1	16	dbg_hub/inst/B.../addr_reg[9]/C	vio_top_module...est_reg[15]/CE	0.983	0.228	0.755	1.3
Path 7	0.066	3	22	dbg_hub/inst/B.../timeout_reg/C	dbg_hub/inst/...l_mode_reg/CE	1.030	0.300	0.730	1.3
Path 8	0.082	3	1	dbg_hub/inst/BS...l_reg_reg[1]/C	dbg_hub/inst/BS...addr_reg[13]/D	1.076	0.423	0.653	1.3
Path 9	0.088	4	7	dbg_hub/inst/BS...addr_reg[14]/C	vio_top_module/...e_out_reg[0]/R	0.936	0.355	0.581	1.3
Path 10	0.088	4	7	dbg_hub/inst/BS...addr_reg[14]/C	vio_top_module/...e_out_reg[1]/R	0.936	0.355	0.581	1.3

附录 1： 版本说明

版本	贡献者	说明	勘误
v00	彭晓恩	DPC 模块设计	

附录 2：参与讨论

有微信群和 QQ 群，欢迎入群讨论

个人 wechat



(来者注明身份，拉入群聊)

QQ 群

