A High-Efficient Image Scalar Algorithm for LCD Signal Processor

Tze-Yun Sung[1] Chun-Wang Yu[1] Yaw-Shih Shieh[1] Hsi-Chin Hsin[2]

[1]Department of Microelectronics Engineering, Chung Hua University, Hsinchu, Taiwan 300-12
[2]Department of Computer Science and Information Engineering, Formosa University, Hu-Wei, Taiwan 632-08

Abstract

In this paper, the image scalar algorithm and architecture for LCD monitor and television controller is presented. The program of hardware code is described to explain the proposed algorithm and architecture. The proposed image scalar has been applied on LCD monitor and TV controller or signal processor. The scalar has been evaluated and applied on the LCD display controller successfully.

Keywords: Image scalar, LCD monitor and television, hardware description language, average and distributed algorithm.

1. Introduction

The image scalar performs enlarge and reduce the input image. We avoid the distortion of the image, which has been scaled. In idea case, the large size of memory are required to store the image, and the high speed digital signal processors are used to do vector computation and optimize the image quality. But the hardware cost must be high, the hardware cost and image quality are trade-offs in the design of the commercial LCD and TV controller [1].

In this paper, the digital signal processors are not used in the controller in order to achieve the cost-down purpose. The simple multipliers are used to perform the algorithm. According to the proposed algorithm, the memory is saved. The architecture of the proposed image scalar is shown in Figure 1. The algorithm and architecture is described as follows:
In the proposed image scalar architecture, it involves eight modules: Y-buffer, Y-calculator, X-buffer, X-calculator, sc-fifo, sc-fifo4, sc-coeff and SC-main. sc-fifo and sc-fifo4 provide the memory module for Y-buffer and X-buffer, respectively. The input data of image signal is stored in Y-buffer. In order to save cost and keep the good quality, the memory size of Y-buffer is 1024 pixels × 3-bytes×8 Lines (8 scan lines) [2], [3], [4].

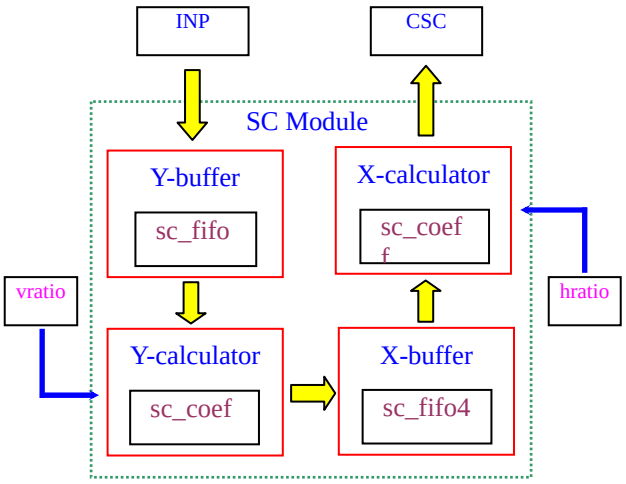

Fig. 1. The architecture of image scalar

2. Data Retrieval and Computation

A new horizontal line is performed by four neighboring lines in Y-buffer. The Y-calculator module performs the computation of a new line. The operation of Y-buffer is shown in Figure 2, and operation of a new line performing is shown in Figure 3.

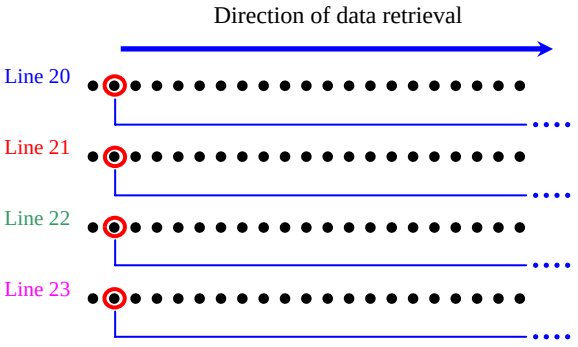

Fig. 2. The data retrieval in Y-buffer

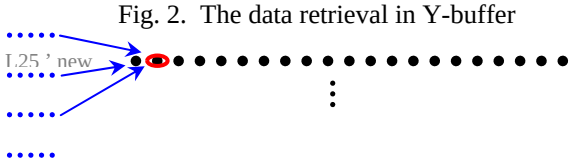

Fig. 3. Y-calculator generates the new pixels

Program 1:

```
if (ref_p_d2==`BIT1 && ref_p_d1==`BIT0)
begin
        addr_l0 <=#1 addr_l0 + inc_p[2:0];
        addr_l1 <=#1 addr_l1 + inc_p[2:0];

        addr_vp <=#1 addr_vp + inc_p;
```

Program 2:

```
assign prd[0] = ((addr_l0[2]) ? `BIT1 : `BIT0);
assign prd[1] = ((addr_l1[2]) ? `BIT1 : `BIT0);
assign prd[2] = ((addr_l2[2]) ? `BIT1 : `BIT0);
assign prd[3] = ((addr  l3[2]) ? `BIT1 : `BIT0);
```

Program 3:

```
if (ref_p_d1==`BIT1 && ref_p==`BIT1)
    addr_r <=#1 addr_r + `BIT1;
```

Program 4:

```
sc_fifo fifo0(…wr[0],rd[0],addr_w,addr_r,po_l0)
sc_fifo fifo1(…wr[1],rd[1],addr_w,addr_r,po_l1)
sc_fifo fifo2(…wr[2],rd[2],addr_w,addr_r,po_l2)
sc_fifo fifo3(…wr[3],rd[3],addr_w,addr_r,po_l3)
```

Program 5:

```
case({addr_l0,line_last,line_land})
        5'b00000:
                begin
                        p_m1 <=#1 po_l1;
                        p_z0 <=#1 po_l2;
                        p_p1 <=#1 po_l3;
                        p_p2 <=#1 po_l4;
                end
        5'b00100:
                begin
                        p_m1 <=#1 po_l2;
                        p_z0 <=#1 po_l3;
                        p_p1 <=#1 po_l4;
                        p_p2 <=#1 po_l5;
```

Program 6:

```
if (vid_en == `BIT1 && odd == `BIT0) wr <=#1 jwr;
else if (vid_en == `BIT1 && odd == `BIT1) wr <=#1 qwr
else wr <=#1 pwr;
```

Program 7:

```
assign ry = myr + { 8'b0, myr[16:8] } + 8'h80;
```

The data of initial horizontal line storing in inc p[2:0] is described in program 1. According to program 1, the data of four neighboring horizontal lines are read, when a new line is performing. The value of add_r is accumulated for generation of new pixels. The above operation is described in Program 2. In program 3 and 4, the memory structure and index method are described. In program 5, the data of a new line would be changed in memory address addr_10.

In program 6 and 7, the new pixel is generated by simple multiplication and shift operation. It doesn't require digital signal processor to perform the algorithm. The algorithm is described as follows:

$$\text{New Pixel} = \frac{C3 \times p\_m1}{256} + \frac{C2 \times p\_z0}{256} + \frac{C1 \times p\_p1}{256} + \frac{C0 \times p\_p2}{256} + 0.5$$

3. The Memory Optimization

Additional four memories are used to store the data of four lines synchronously, when the Y_calculator is generating a new line [3], [4].

The proposed image scalar has function of error detection. The input signals of image scalar may be video, S-video, HDTV and etc.[6]. The resolution of the input signal is half or quarter as the scalar needs, so that the resolution of the input image will be doubled by the scalar. It can avoid the distortion and improve the image quality. The doubling resolution algorithm is shown in Figure 4.

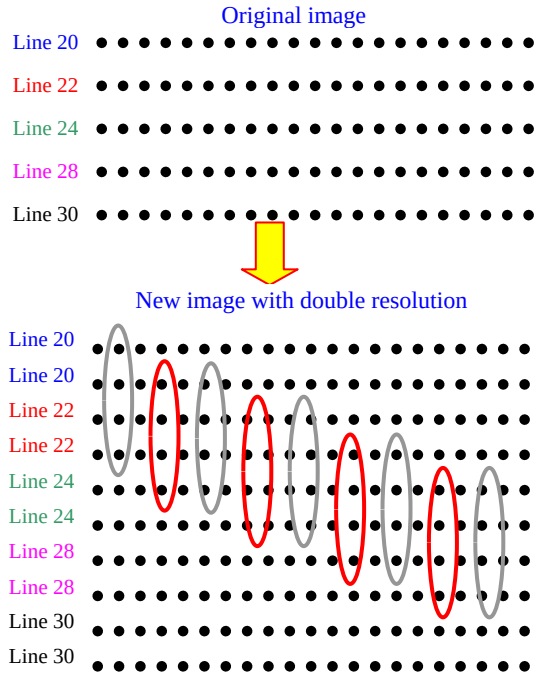

Fig. 4. The algorithm of doubling resolution

In program 8 and 9, 526 horizontal scan lines being expanded to 768 and 1024 lines is described.

Program 8:

```
assign my_m1 = p_m1[23:16] * coeff[34:27]; // C3
assign my_z0 = p_z0[23:16] * coeff[25:18]; // C2
assign my_p1 = p_p1[23:16] * coeff[16: 9]; // C1
assign my_p2 = p_p2[23:16] * coeff[ 7: 0]; // C0
```

Program 9:

```
assign qwr[0] = (addr_v[1:0]==2'b00)|| line_first;
assign qwr[1] = (addr_v[1:0]==2'b01);
assign qwr[2] = (addr_v[1:0]==2'b01);
assign qwr[3] = (addr_v[1:0]==2'b10);
```

4. The Algorithm of Line Scaling

In Y-buffer, we use the staying and skipping method to expand and reduce lines. For example, the 800 horizontal lines expands to 1024 lines, we choose every 3.57 lines to stay an address of horizontal scan in order to generate a new line.

4.1. Expanding Lines

The special algorithm for expanding 800 horizontal lines to 1024 lines is described as follows:
inc_p = 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1…
where inc_p is data set for generating a new horizontal line. 1 represents that it use a new data of the next horizontal line, and 0 represents that it use the original data. The operation of the algorithm is shown in Figure 6.

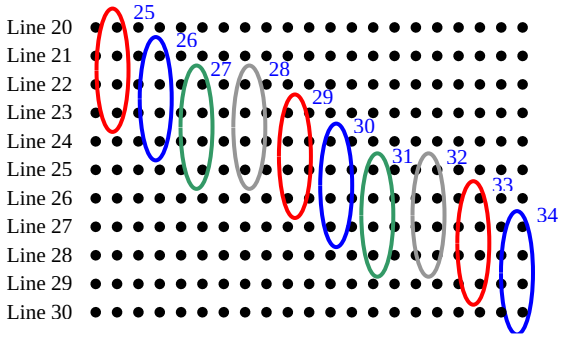

Fig. 5. The operation of line expansion

In Figure 5, a new line 27 is generated by the following equation:
L27 'new = a ×L22 + b× L23 + c× L24 + d × L25

The average and distributed algorithm is applied on line expansion, the same data can generate a different line. The coefficient determination is described in next section.

In program 10, the value of phase and inc_p will be updated, when a new line is generated.

Program 10:

```
if (ref_d1==`BIT0 && ref==`BIT1)
        inc_p <=#1 total_p[15:12];
if (ref_d2==`BIT1 && ref_d1==`BIT0)
begin
        total[15:12] <=#1 4'b0;
        total[11: 0] <=#1 total_p[11:0];
        phase <=#1 total_p[11:7];
```

4.2. Reducing Lines

The inc_p will be updated, when the image is reduced. The operation of line reduction is shown in Figure 6.
inc_p = 1, 1, 1, 2, 1, 1, 1, 2, 1, 1….

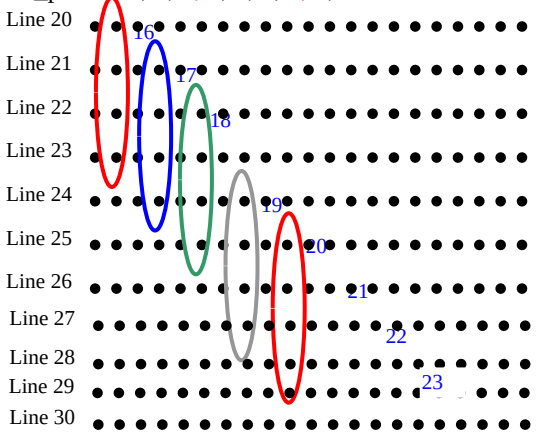

Fig. 6. The operation of line reduction

According to the average and distributed algorithm, the difference of the new line 18 and the new line 19 is two lines, and the value of inc_p is 2.

5. The Coefficients of Image Scalar

According to the average and distributed algorithm, the color distribution of the processed image will be followed that of the original input image. The coefficients are used to adjust the color distribution. We use green color to explain this algorithm, and the computation of distribution of the green color is shown in Figure 7.

In this algorithm, the value of vratio is 5120 and the coefficients from new line 37 to new line 40 are (0, 255, 0, 0), (-18, 221, 58, -6), (-16, 144, 143, -16,) and (-6, 58, 221, -18), respectively. The values of new lines are obtained as follows:
L37 'new:
(0×200+255×180+0×160 +0×140)÷256=179

The original values from line 31 to 34 are 180, 160, 140 and 120, respectively. The new values are 179, 154, 130, 125 and 120, respectively, when original four lines are expanded to new five lines. In 256-gray level, the difference of the new line (125) is ±5. The color of the processed image is very smooth.

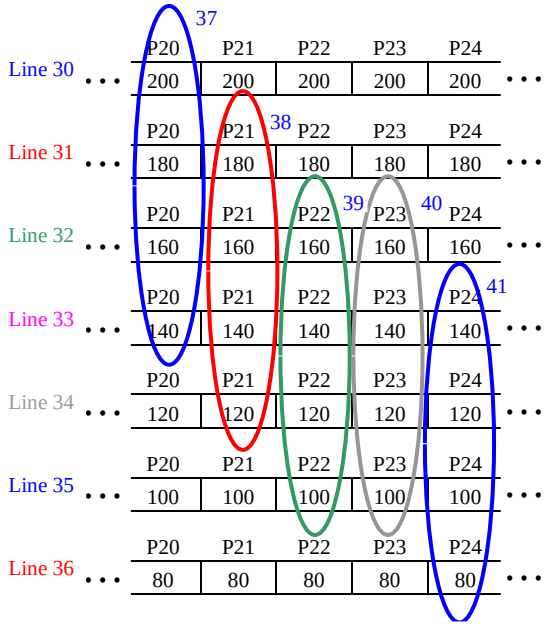

Fig. 7. The computation of distribution of the green color

6. The Algorithm of Pixel Scaling

In image scalar, the neighboring four vertical pixels in X-buffer are chosen to expand a new pixel.
Basically, the concept of data retrieval of X-buffer is the same as that of Y-buffer.

Program 11:

```
case(addr_type)
        2'b00:
                begin
                        p_m1 <=#1 po_l0;
                        p_z0 <=#1 po_l1;
                        p_p1 <=#1 po_l2;
                        p_p2 <=#1 po_l3;
                end
        2'b01:
                begin
                        p_m1 <=#1 po_l1;
                        p_z0 <=#1 po_l2;
                        p_p1 <=#1 po_l3;
                        p_p2 <=#1 po_l0;
```

Program 12:

```
case(addr_nl0[1:0])
        2'b00:
                begin
                        addr_r0 <=#1
addr_nl0[10:2];
                        addr_r1 <=#1
addr_nl1[10:2];
                        addr_r2 <=#1
addr_nl2[10:2];
                        addr_r3 <=#1
addr_nl3[10:2];
                end
        2'b01:
                begin
                        addr_r0 <=#1
addr_nl3[10:2];
                        addr_r1 <=#1
addr_nl0[10:2];
                        addr_r2 <=#1
addr_nl1[10:2];
                        addr_r3 <=#1
addr_nl2[10:2];

addr_type <=#1 addr_l0[1:0];
addr_l0 <=#1 addr_nl0;
```

In programs 11 and 12, the new data of pixel is generated at each display clock.

7. Conclusions

In this paper, we use the hardware description language (HDL) to implement the circuit of image scalar, and the cost of chip is also taken into consideration. The proposed algorithm is very suitable for VLSI implementation. The simple circuits and small area are achieved. In the future, we try to find more optimal algorithms to improve the image quality and computation speed.

Now, the test platform has been established and more optimal algorithms can be developed and evaluated easily.

References

[1] PHILIPS , SAA6721E SXGA RGB to TFT graphics engine data sheet, 1999.

[2] D. E. Thomas, P. H. Moorby, The Verilog Hardware Description Language, Fifth Edition, Kluwer Academic Pub. 2002.

[3] T. Y. Sung, Y. T. Chen, C. S. Chen, “Image Processing Algorithms Applying on Flat Panel Display Controller”, *2003 Computer Graphics Workshop*, Hualian, Taiwan, August, 28~29, 2003.

[4] T. Y. Sung, Y. T. Chen, “Implementation of Optimal Image Scaling Algorithm for LCD Monitor Controller,” *First Science and Technology Conference-Photonics and Communications*, Kaohsiung, Taiwan, Dec. 9-10, 2004, PJ-01.

[5] T. Y. Sung, C. S. Chen, “Implementation of Color Image Processing Algorithms for LCD Monitor Controller,” First Science and Technology Conference-Photonics and Communications, Taiwan, Dec. 9-10, 2004, A-04.

[6] R. C. Gonzalez, R. E. Woods, Digital Image Processing,” Addison-Wesley Publishing Company, 2000.