CAC hardware module

CTL Alibaba.

Revison History

Revision	Date	Author	Description
0.1	2022.8.22	Wanwei Xiao	Initial draft

1. Introduction

1.1 Request & Purpose

This document presents chromatic aberration correction algorithm and specification in CTL ISP pipeline architecture. It defines the features with high–level diagram and modules design. The team members can follow this document to do detailed design and implementation.

1.2 Definitions & Abbreviations

Name	Description
CAC	chromatic aberration correction

1.3 Reference

[1]

2. Overview

The CAC module is used to eliminate chromatic aberrations in the image generated by the lens, generally including horizontal and axial chromatic aberrations, as well as the purple fringing phenomenon of some images

Figure 2–1 Function of cac module[2]

2.1 CAC Location

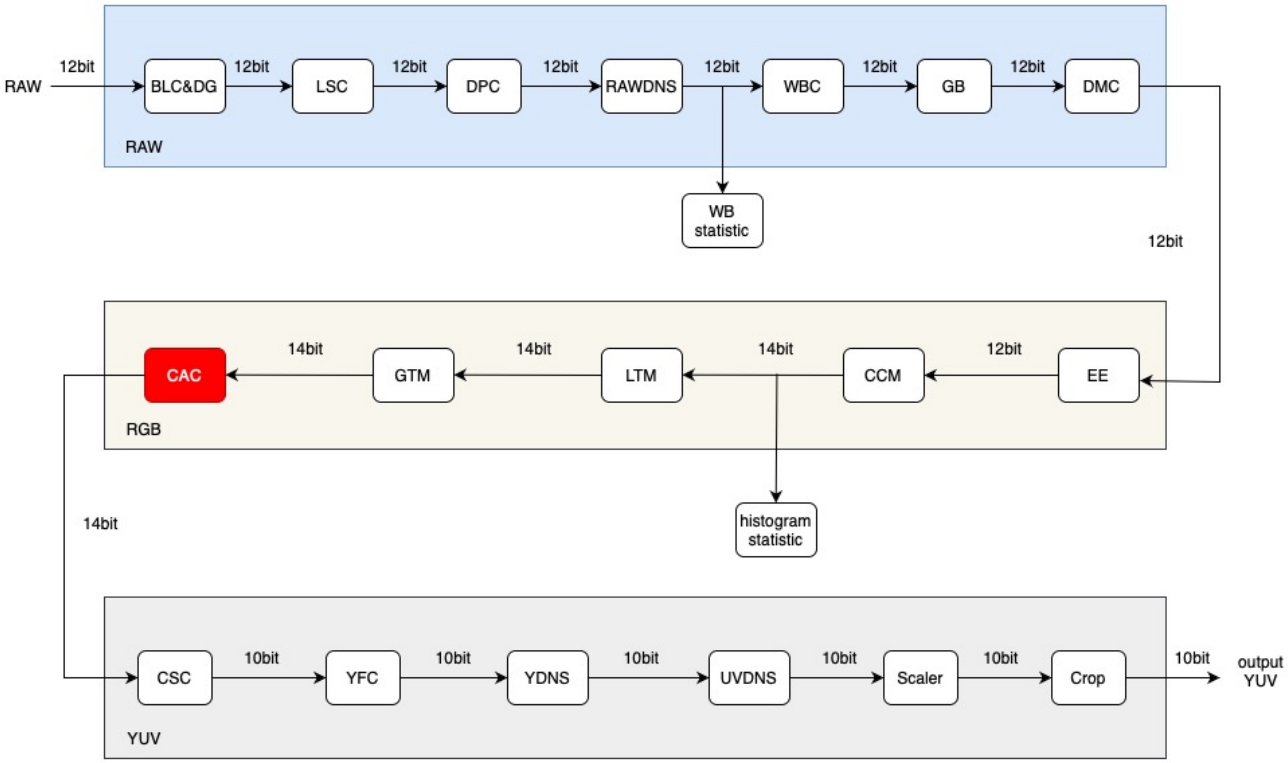

Figure 2–2 CAC location in ISP pipeline

2.2 CAC Register

Name	Bits Width	Range	Default Value	Shadow	Description
eb	1	0 or 1	1		enable signal for CAC module
t_transient	17	[–65536,65535]	7000		determination threshold for transient region
t_edge	17	[–65536,65535]	4000		determination threshold for edges of transient region

2.3 Top–level function interfaces

Name	Bits Width	Description
top_reg	/	ISP top_level parameters
cac_reg	/	CAC module parameters
src	42	the dataflow of input
dst	42	the dataflow of output

3. CAC Aglorithm&Architecture

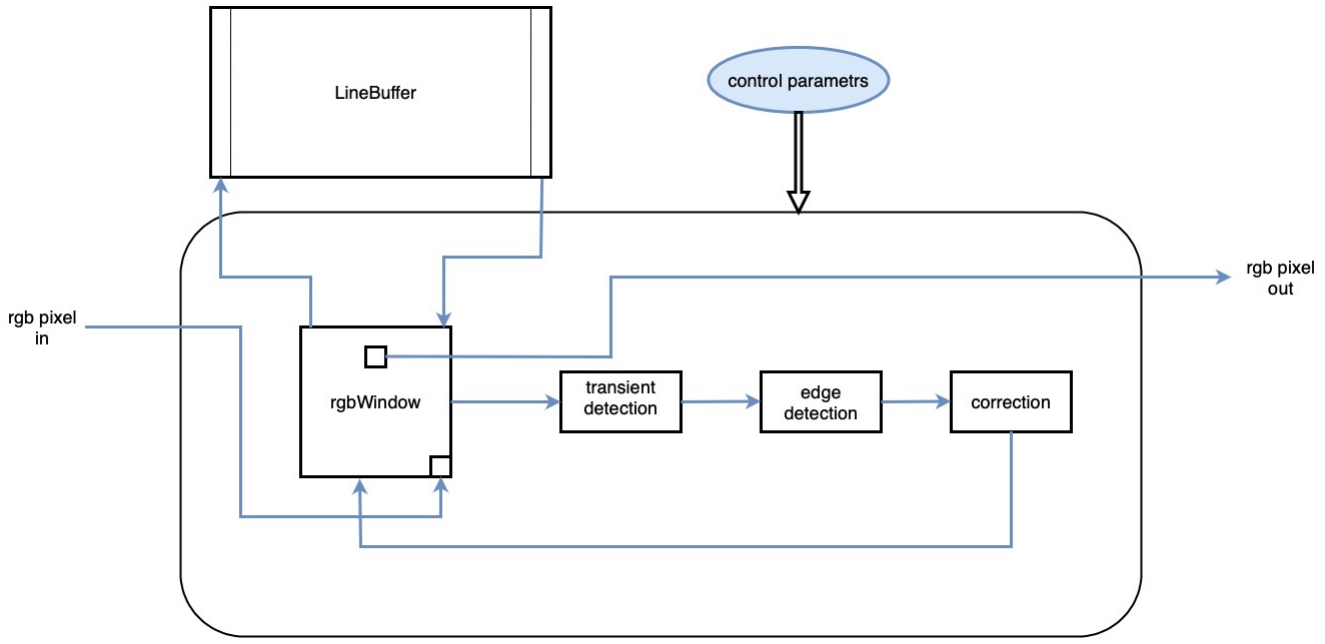

Figure 3–1 CAC module

- top register

Name	Bits Width	Range	Description
frameWidth	13	[0, 8192)	the width of image
frameHeight	13	[0, 8192)	the height of image
inputPattern	2	0, 1, 2, 3	the pattern of bayer, 0:r 1:Gr 2:Gb 3:b
blc	9	[0, 512)	black pixel value

3.1 Start of frame

The CAC module frame starts with just two thresholds t_transient and t_edge loaded, as well as the data input to rgbWindow

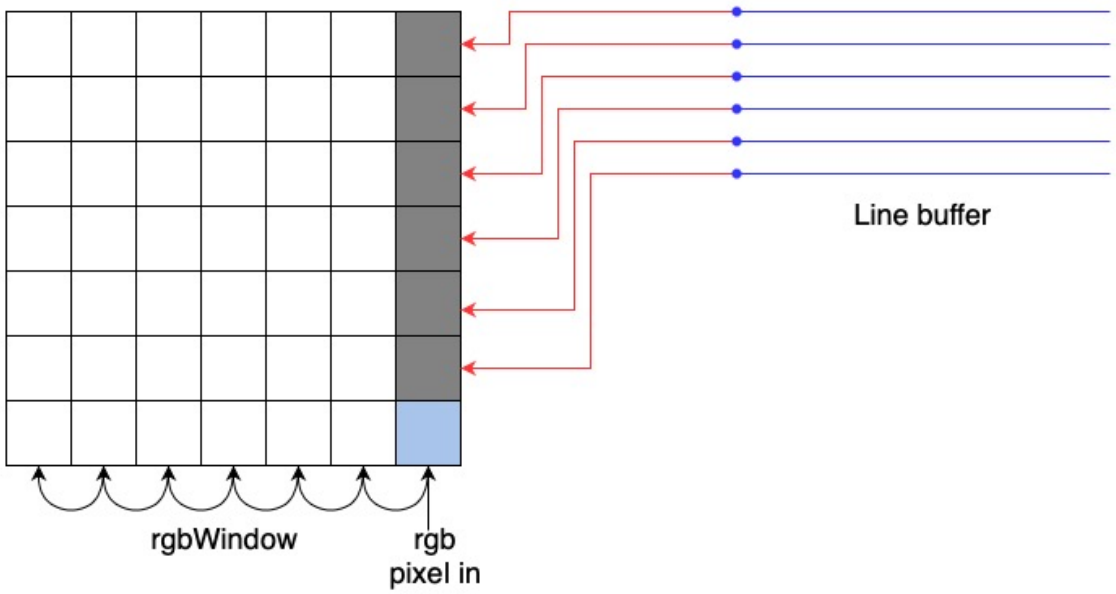

Figure 3–2 rgb window schedule

3.2 Pixel Preprocessing

3.2.1 Algorithm

algorithm details you can reference the paper [1]. The whole process is divided into horizontal and vertical correction, but this process execute in a 7x7 rgbWindow.

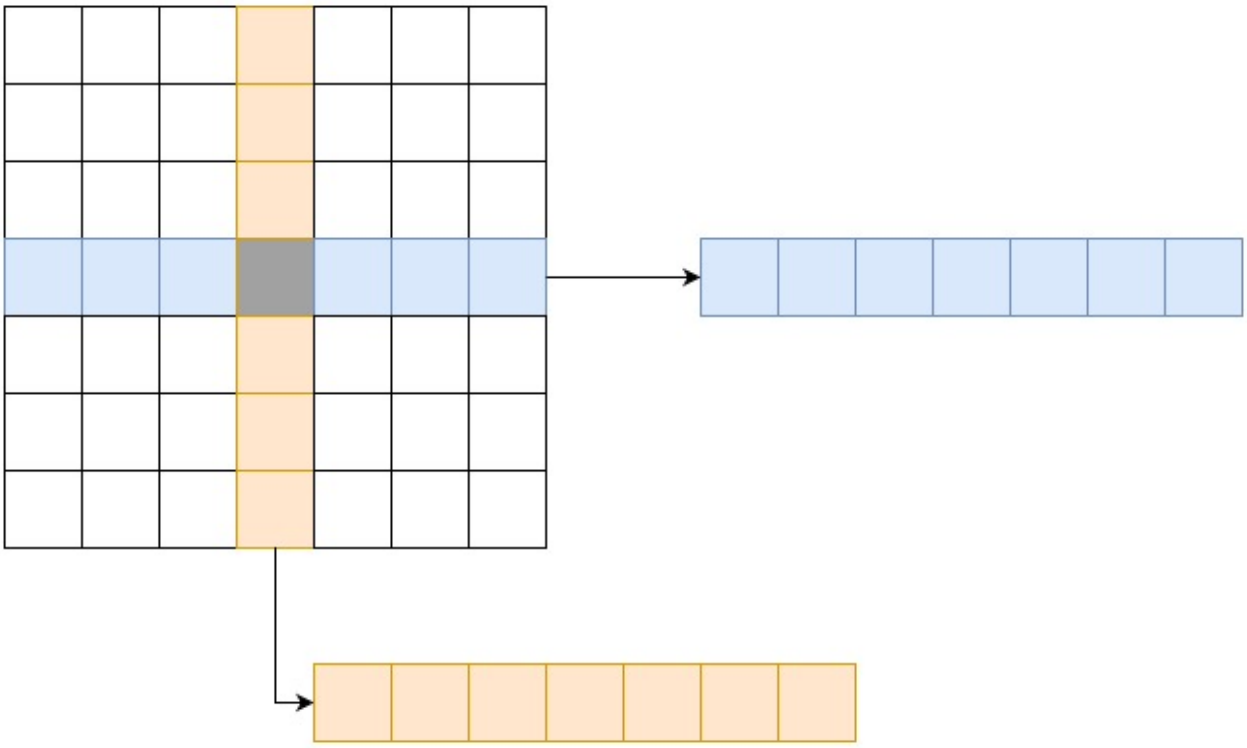

Figure 3–3 pixel preprocessing

3.2.2 Hardware implementation

The whole procedure can be divided into 2 stages.

Stage1: load the fourth row pixel into register.

Stage2: load the fourth column pixel into register.

3.2 Chromatic difference and edge information calculation

3.2.1 Algorithm

Calculate the difference and edge information, difference information include the difference between blue channel and green channel and the difference between red channel and green channel. The edge information is calculated by sobel mask

3.2.2 Hardware implementation

The whole procedure can be divided into 2 stages.

Stage1: calculate the difference information

- rg_diff = inPixel_int.r – inpPixel_int.g
- bg_diff = inPixel_int.b – inPixel_int.g

Stage2: calculate the edge information

C_edge_d = Ec(i,j) = C(i–1,j–1) + 2C(i,j–1) + C(i+1,j–1) – C(i–1,j+1) – 2C(i,j+1) –C(i+1,j+1) //C is the {R,G,B}, d represent the direction, horizontal or vertical

3.4 Transient region and edges of transient detection

3.4.1 Algorithm

This step include two stage. Take the horizontal direction as an example

First of all, the pixel in the image of a green channel gradient judgment, if its gradient is greater than the set threshold t_transient, then the pixel p is considered to be located in a transient.

The second stage is based on the expansion of the pixel to both ends, Satisfy the following conditions， the pixel i is l_t and r_t

(max(sigma_temp * r_edge_h[i], max(sigma_temp * g_edge_h[i], sigma_temp * b_edge_h[i]) < t_edge //the sigma_temp = 1 or –1, 1, the same direction, –1, the opposite direction

There are two edges, l_t and r_t.

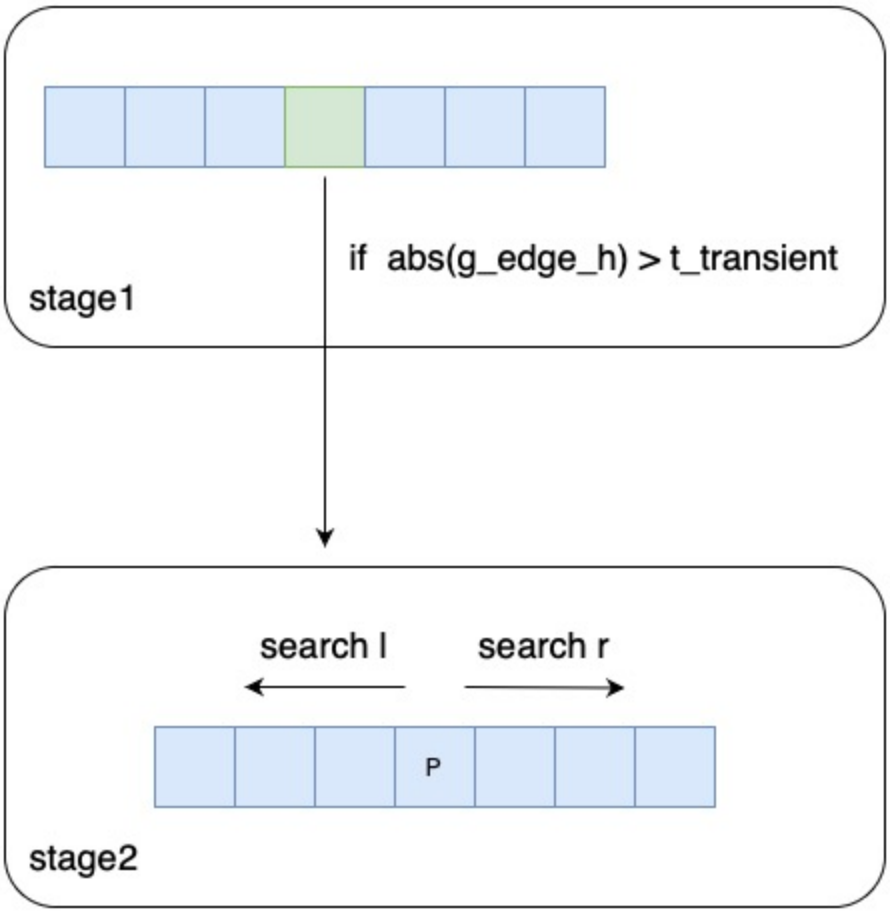

Figure 3–4. Transient region and edges of transient detection

3.4.2 Hardware implementation

The whole procedure can be divided into 2 stages.

Stage1: get the transient region

if abs(g_edge_h[3]) > t_transient

execute the stage2

else

execute next pixel cycle.

Stage 2: calculate the l_t and r_t

if g_edge_h[3] > t_transient

search to the left and right, until (Eg < t_edge && Er < t_edge && Eb < t_edge)

get l_t and r_t

if can not get l_t or r_t, l_t = 0 and r_t = 6

else if g_edge_h[3] < –t_transient

search to the left and right, until (Eg > –t_edge && Er > –t_edge && Eb > –t_edge)

get l_t and r_t

if can not get l_t or r_t, l_t = 0 and r_t = 6

else

execute next pixel cycle.

3.5 Correction

3.5.1 Algorithm

This step is the pixel correction, which includes all pixel values from l_t to r_t, with l_t and r_t as reference pixels. The main constraint of the correction process is as follows.

- min{rg_diff[l_t], rg_diff[r_t]} <= rg_diff[i] <= max{rg_diff[l_t], rg_diff[r_t]} //i is the pixel between l_t and r_t
- min{bg_diff[l_t], bg_diff[r_t]} <= bg_diff[i] <= max{bg_diff[l_t], bg_diff[r_t]} //i is the pixel between l_t and r_t

Figure 3–5 is a correction example, and the l_t = 1, r_t = 4.

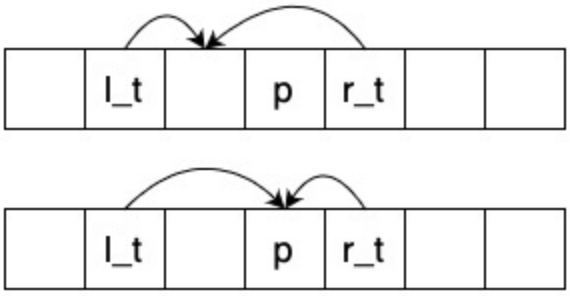

Figure 3–5. correction example

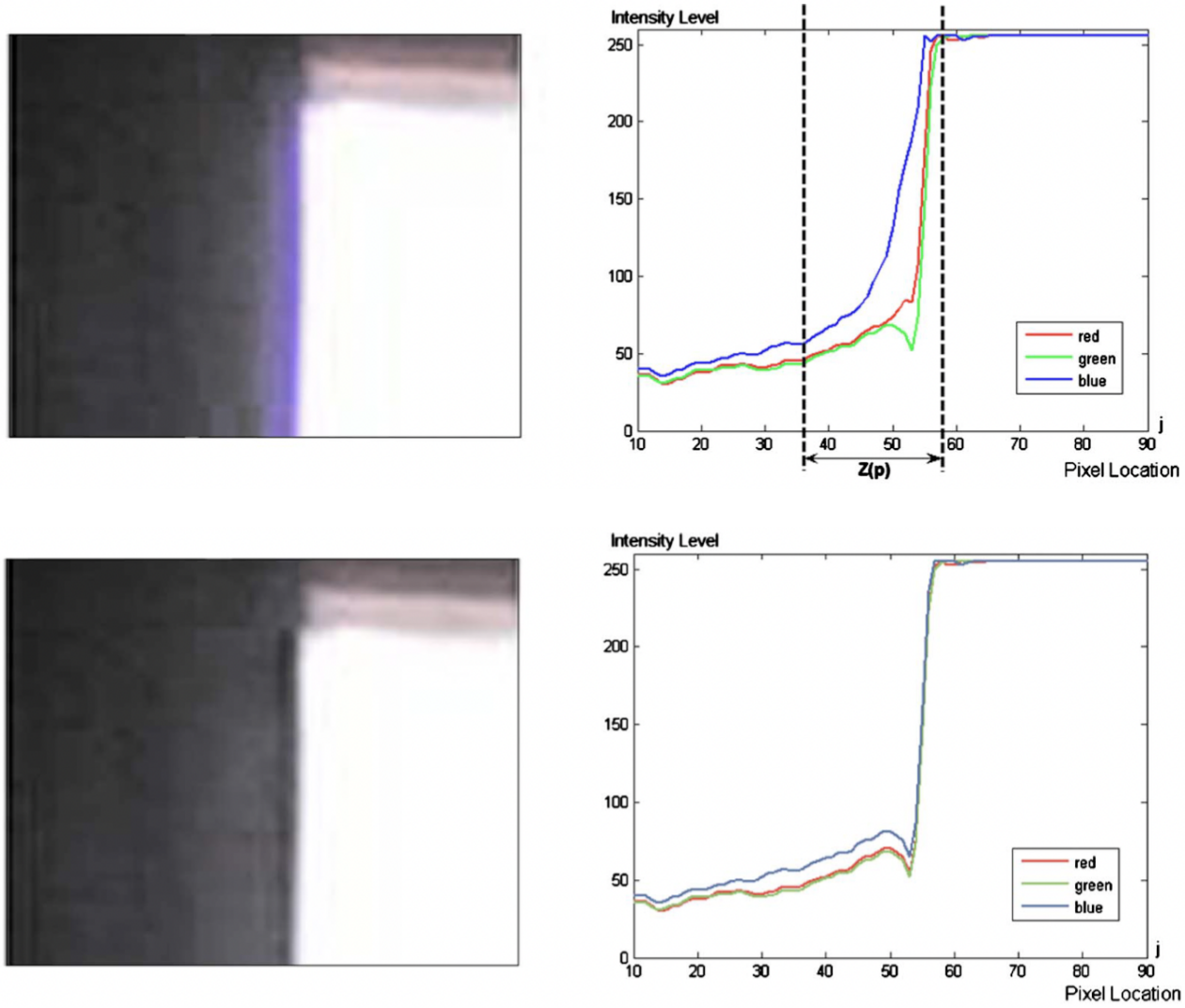

Figure3–6. correction effect and rgb intensity[1]

3.5.2 Hardware implementation

The whole procedure can be divided into 2 stages.

Stage1: Determine if the pixel in the transient area needs to be corrected

select pixel with l_t to r_t

if (bg_diff(pixel) > max(bg_diff(l_t), bg_diff(r_t))

or bg_diff(pixel) < min(bg_diff(l_t), bg_diff(r_t))

or rg_diff(pixel) > max(rg_diff(l_t), rg_diff(r_t))

or rg_diff(pixel) < min(rg_diff(l_t), rg_diff(r_t)))

execute stage2

else

execute next pixel cycle.

stage2: calculate correct values, correction effect and rgb intensity are shown as figure 3–6.

if (bg_diff(pixel) > max(bg_diff(l_t), bg_diff(r_t)))

output_pixel = inPixel_int.g + max(bg_diff(l_t), bg_diff(r_t))

else if (bg_diff(pixel) < min(bg_diff(l_t), bg_diff(r_t)))

output_pixel = inPixel_int.g + min(bg_diff(l_t), bg_diff(r_t))

if (rg_diff(pixel) > max(rg_diff(l_t), rg_diff(r_t)))

output_pixel = inPixel_int.g + max(rg_diff(l_t), rg_diff(r_t))

else if (rg_diff(pixel) < min(rg_diff(l_t), rg_diff(r_t)))

output_pixel = inPixel_int.g + min(rg_diff(l_t), rg_diff(r_t))

3.6 Write back

3.6.1 Algorithm

Since a single cycle currently outputs one pixel, the additional pixel values need to be written back to the rgbWindow as well as the line buffer. as shown in Fingure 3–7

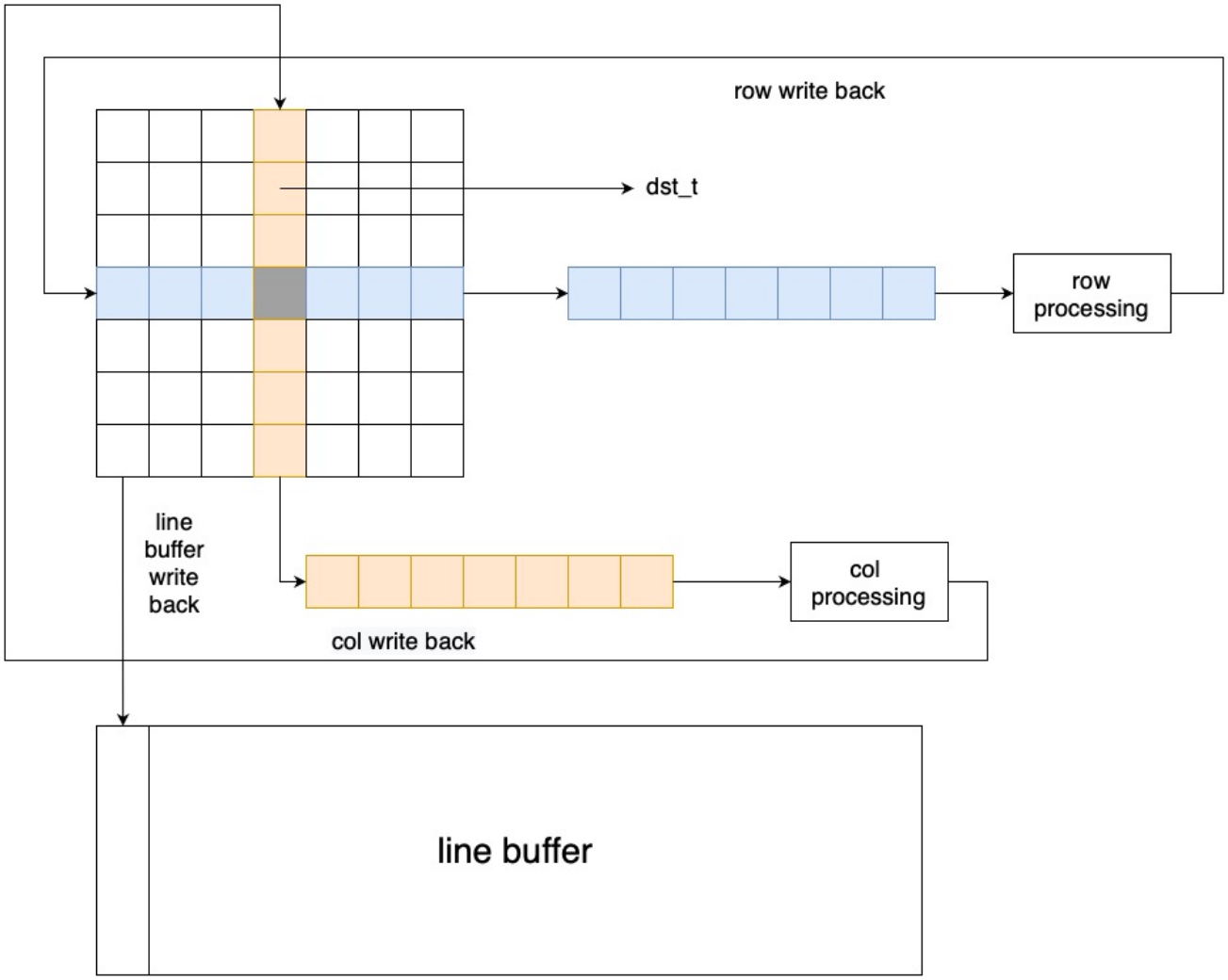

Figure 3–7 pixel write back

3.6.2 Hardware implementation

The whole procedure can be divided into 3 stages.

Stage1: row write back and col write back.

write back horizontal pixels to the corresponding rgbWindow position,write back vertical pixels to the corresponding rgbWindow position

Stage2: output pixel rgbWindow(1,3)

Stage3: write back the first column pixels to the first column of line buffer.

4. Testing and Verification

4.1 hardware function test

- run c model and open the DUMP_FIGURE in common/top.h
- copy the ouput figure to the path hardware/tv/
- cd tcl/
- make cac
- if no pixel mismatch, the test pass

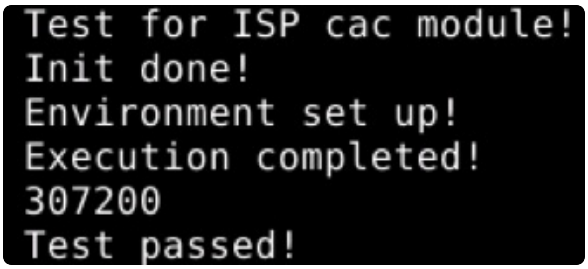

Figure 4–1. test result

4.2 HLS synthesis results

4.2.1 synthesis

when you run the hardware function test, if you open the vivado_hls in hardware/src/top.h, you can get a report in the path hardware/tcl/cac/cac/syn/report/cac_csynth.rpt

4.2.1 systhesis results

frequency	BRAM_18k	DSP48E	FF	LUT	URAM
50Mhz	118	0	4803	9904	0

5. Reference

[1] Chung S W , Kim B K , Song W J . Removing chromatic aberration by digital image processing[J]. Optical Engineering, 2010, 49.

[2] Chang J , Kang H , Kang M G . Correction of Axial and Lateral Chromatic Aberration With False Color Filtering[J]. IEEE Transactions on Image Processing A Publication of the IEEE Signal Processing Society, 2013, 22(3):1186–1198.