A Flexible and Low-Cost Hardware Implementation for Real-Time Video Stitching in 2D

Leilei Huang*, Yanheng Lu, Xiaoyang Zeng, Yibo Fan*
State Key Lab of ASIC & System, Fudan University, Shanghai 200433, China
Email:10300720005@fudan.edu.cn, fanyibo@fudan.edu.cn

***Abstract*—Due to the complexity of general image stitching methods, it is difficult to realize real-time video stitching at a reasonable level of cost. To solve this problem, this paper proposes a flexible and low-cost hardware implementation for such application, which can be easily reconfigured or redesigned to support different number of cameras arrayed in different ways in 2 dimensions. When supports 4 cameras arrayed as a 2x2 square, it cost only 5 small line buffers, which are one-port rams, and limited control and calculation resources to fulfill the stitching task. After configuration, it can be simply regarded as one camera with a higher resolution, generating frame valid, line valid and pixel data signals. When works at only 78MHz, it still can realize a real-time video stitching of 2000x1200@30fps from 4 channels of 1280x720@30fps videos. The results of simulation and FPGA verification show no apparent artifacts in blended pictures.**

I. INTRODUCTION

Video stitching is to generate videos with higher resolution from lowers ones, which can be used in many applications such as video conference, film making or in-vehicle cameras. However, due to the complexity of image stitching methods, it is hard to realize video stitching with hardware at a reasonable level of cost, not mention doing it in real time. Figure 1 shows a general flow to stitch images, which explains the complexity to a certain extent.

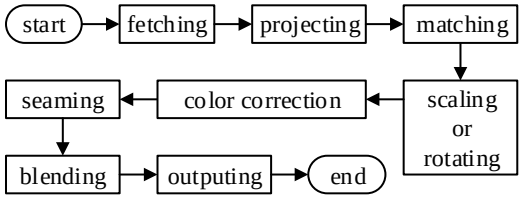

Fig. 1. General flow of image stitching

Till now, few papers proposed hardware implementations. In software area, work [5], [6] and [7] explored this problem based on CPU or DSP. Though the stitching results are satisfying, both the speed and the cost remain to be improved. Besides, only one dimension is supported by these works, which limits the practicability.

It has to be pointed out that the bottleneck of above works or general video stitching methods mainly lies in that the processing unit is one frame. For example, only after the projecting to a whole frame is done, matching could be done; or only after the seam line of a whole frame is found, blending could be done, which causes the dilemma of either using huge on-chip ram to store frames or consuming long time to fetch and store frames from or to off-chip rams. In other words, if methods of reducing the processing unit are found, stitching can be implemented at a lower cost and a faster speed, which inspired this papers.

II. BRIEF WORK FLOW

A. Assumptions

This design is aimed at source videos with:

1) *Negligible Size Mismatch*
2) *Negligible Angle Mismatch*
3) *Fixed Camera Position and Angle*

Before stitching, source videos have to be zoomed or rotated, even be projected, if the size is mismatched, or the angle is mismatched. No matter in which situation, data from multiple lines are needed to generate the wanted data, which cost not only a great amount of the storage space but also a great amount of control and calculation resources, especially for hardware implementations. While the last assumption makes sure that the image-match process can be done only once, because fixed camera position and angel means fixed overlapped area of source videos.

Fortunately, in some real applications, like video conference, these two assumptions can be easily satisfied.

B. Work Flow

A brief schematic of this design is shown in figure 2.

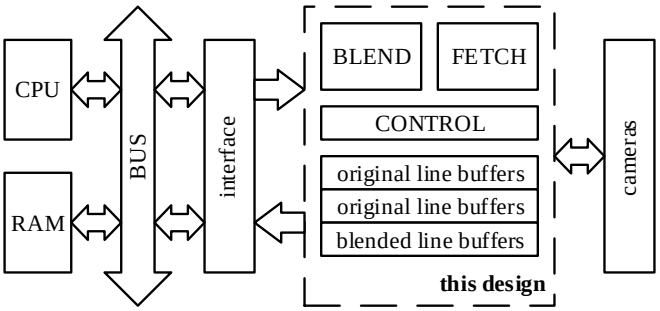

Fig. 2. Brief schematic of proposed design

The proposed design has two modes.

In one mode, it simply fetches one frame from a certain camera and outputs the data, with which, CPU could apply algorithms like SURT [1] to generate configuration values for the other mode. Just as mentioned above, match algorithm is needed only once, so it can be very complex and time-consuming in order to get a better result.

In the other mode, module FETCH fetches lines of pixels, does some color correction or conversion, and then stores them into one group of original line buffers. At the same time, module BLEND finds the seam line and blends data in the other group of original line buffers to generate blended pixels and store them to blended line buffers. Module CONTROL takes charge of the control of above two modules, the Ping-Pong access to line buffers and the output of final data.

It should be pointed out that the minimum process unit in this design is one line of pixels instead of one frame, which makes the real time stitching can be realized at a very low cost of storage space or bandwidth. More specifically, all of the processes in this design are done in unit of line, so the Ping-Pong flow, shown in figure 3, can be done at the cost of adding only one group of line buffers instead of frame buffers.

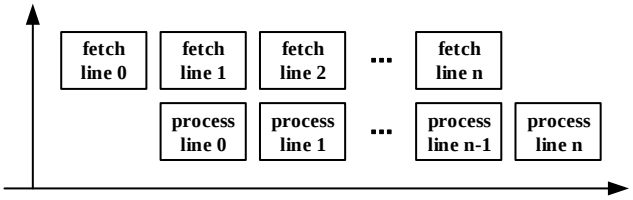

Fig. 3. Ping-Pong work flow

From above, it can be inferred that the processing time to get one blended frame T satisfies the following equation:

$$T = t_{FETCH} + (n-1)\,max(t_{FETCH}, t_{PROCESS}) + t_{PROCESS} \quad (1)$$

where t_{FETCH} denotes the time for module FETCH to process one line, $t_{PROCESS}$ denotes the total time to blend and output one line while n denote the amount of lines in blended images. This equation will be used later to prove the real-time possibility of this design in theory.

III. Detailed Implementation

A. Fetching

As a matter of fact, the above Ping-Pong flow implicitly establishes on the assumption that the data of cameras can be fetched in snatches. If not, then the data may be continuously dumped into original line buffers, so when one group of lines is filled but the blending and outputting of the other group are not done, the loss of valid data appears. Fortunately, this kind of cameras is very common, like OV7670 camera [2] from OMNIVISION, and other cameras can be regarded as OV7670 when used with (off-chip) FIFOs. For this reason, cameras used in this design can be taken as FIFOs which contain one frame of image data, and the fetch time of one line only depends on the line length and data format. For example, if the line length is 1280 and the RGB value of one pixel can be got in one cycle, t_{FETCH} should be exactly 1280.

B. Color Correcting and Converting

In real application, apart from the above problem, color correction and format converting in module FETCH should also be paid attention to.

Color correction is due to the mismatch between and automatic calibration of cameras. It can be simply realized by three multipliers for each color channel in RGB format. One group of multiplication facts, of course, should be the value of RGB and the other group is the correction coefficients which can be calculated by CPU. For the same reason as match process, the calculation of these coefficients can be done only once.

While format converting is due to different formats used by different cameras, such as Raw RGB, GRB (4:2:2), RGB 565, RGB 555, RGB 444, YUV (4:2:2) or YCbCr (4:2:2). In our design, full channels of each pixel is needed, so converting process is usually inevitable.

C. Seam Line Searching

In traditional way, seam line is searched by the following equations [3]:

$$E_{i,j} = e_{i,j} + min(E_{i-1,j}, E_{i-1,j-1}, E_{i-1,j}) \quad (2)$$

where i and j denoted coordinates, E denotes the cumulative error of overlapped regions, e denotes the error which is calculated by the value of pixels in overlapped region B_1^{ov} and B_2^{ov} from two source images [3]:

$$e_{i,j} = (B_{1,i,j}^{ov} - B_{2,i,j}^{ov})^2 \quad (3)$$

In the above method, only after the minimum cumulative error of the last line is worked out, the seam line of two sources images can be got by backtracking from the minimum entry. In other words, source images have to be loaded at least two times to find a feasible seam line, which is a bottleneck in real-time processing.

To deal with this problem, a new method is proposed in this paper:

1) First Step: The first line is searched to get the best entry i_1 using the absolute difference d of corresponding pixels as the measurement, which is

$$d_{i,j} = \left|B_{1,i,j}^{ov} - B_{2,i,j}^{ov}\right|;\ i = 1, \dots, m;\ j = 1 \quad (4)$$

2) *Second Step:* The neighboring pixels of i_1 in second line is searched to get the second point i_2 of seam line, then the third line, the fourth line, until the last line.

$$d_{i,j} = \left|B_{1,i,j}^{ov} - B_{2,i,j}^{ov}\right|; i = i_{j-1} - 1, i_{j-1}, i_{j-1} + 1; j = 2,3, \dots n \quad (5)$$

From above, it can be inferred that the proposed method has at least three advantages:

1) *Bandwidth:* Source images are processed line by line, and seam line is worked out immediately. It is unnecessary to load one frame twice, even blending process can be done at the same load of one line pixels, which greatly reduces the bandwidth to store and fetch image data.

2) *Storage Space:* only one information is needed to search the next line j, which is i_{j-1}. Storage space to store all possible seams for each entry in traditional method can be saved.

3) *Expansibility:* In order to get a better seam line, it is easy to expand this method. For example, the neighboring boundary can be expanded from$[-1,+1]$ to $[-2,+2]$; the measurement can be expanded to absolute difference of corresponding blocks, and the middle point of the best block is taken as the current seam point:

$$d_{i,j} = \sum_{x=i-1}^{i+1} \left|B_{1,x,j}^{ov} - B_{2,x,j}^{ov}\right|; \quad (6)$$

Attention should be paid on the fact that it is possible to find several equal minimum d in one line. In this situation, it

is better to choose coordinate which closed to the middle as the current seam point, otherwise the seam line would easily decline to the left or the right boundary. When implemented by hardware, an easy way to realize this logic is shown in figure 4(a).

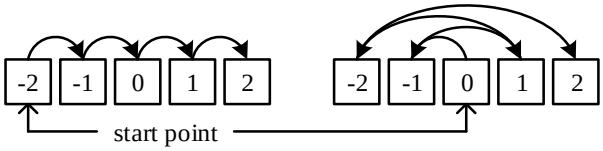

Fig. 4(a). Two different search orders

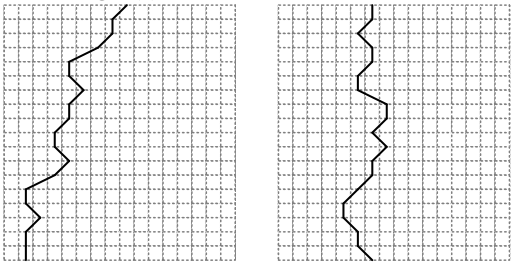
Fig. 4(b). Corresponding seam line

Instead of simply searches from right to left, this design searches from middle to the right, then to the left, then right again left again, until to the left neighboring boundary.

D.Seam Line Boundary

Some notation should be clarified at the beginning of this section, which is overlapped range, search range and blend range. Overlapped range is half the horizontal length of overlapped area. Search range is the maximum distance from seam line to the middle of overlapped area. Blend range is the range to do blending along the seam line. All of these amounts are marked in figure 5.

It is natural to understand that overlapped range should be greater than the sum of search range and blend range, or there wouldn't be data existing to do blending. In traditional method, because there is no restricts on the above three amounts, in some situation, the seam line could be very far from the middle, leading to a narrow margin for blend range. If this happens, it is very likely for human eye to detect the blend line. While in this design, a search boundary is put on the search of seam line, which restricts the seam line into a reasonable range, leaving enough blend margins.

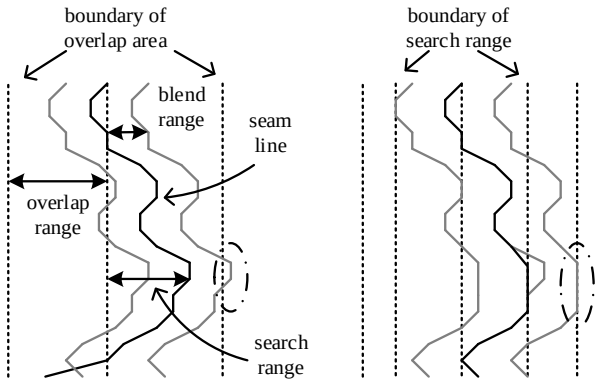

Fig. 5. Boundary added to search range

E. Blending

In this design, instead of the original linear method [4], a better blend method is proposed and adopted, which takes horizontal gradient of pixels into account to reduce the blur:

$$F(p) = \alpha * I_1(p) + (1-\alpha) * I_2(p)$$
$$\alpha = \max(1, d_1/(d_1+d_2) * (1 + 0.1\partial I_1(p))), p \le i_{seam} \quad or$$
$$\alpha = \min(0, d_1/(d_1+d_2) * (1 - 0.1\partial I_2(p))), p > i_{seam} \quad (7)$$

where p denotes the position of the pixel to be generated, α denotes the blend coefficient, d denotes the distance between p and blend boundary, $I(p)$ denotes the value of the source pixels and $F(p)$ denotes the blended value.

This optimization is based on an intuitive feelings which is that properly raising the proportion of source image with a higher gradient would make blended image less blurred.

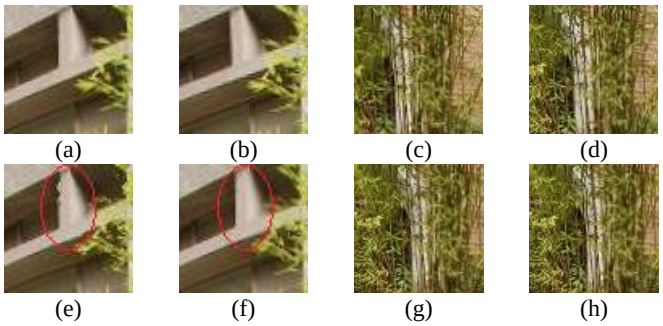

Fig. 6. (a)-(d) Source images; (e)&(g) Blend with original linear method; (f)&(h) Blend with proposed linear method

F. Controlling

Since module FETCH and module BLEND only process data in one line, the control logic to them becomes very simple.

Getting the signal of which cameras to fetch would be enough for module FETCH to function properly. Still using 4 cameras arrayed like a 2x2 square as an example, figure 7 shows the correct signal in this situation:

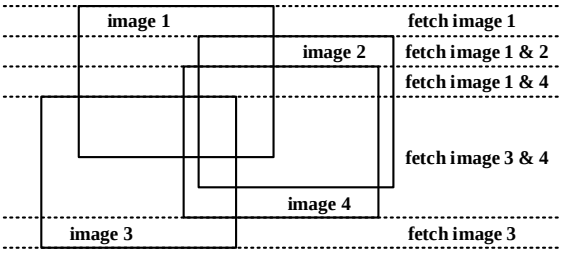

Fig. 7. An example of fetch signal

The output data from module FETCH should be stored in one group of original line buffers which can be organized in two ways. Using 3 cameras arrayed like a horizontal line as an example, 3 full-line buffers or 2 full-line buffers and 2 half-line of one-port rams are used respectively, shown in figure 8.

In the serial way, data from each camera is stored in one corresponding full-line buffer. Because these buffer consists of one-port rams, two overlapped area in buffer 2 can't be read at the same time, so blend process has to be serial.

In the parallel way, data from the middle camera is stored in two buffers, size of which is half of full-line buffers. In this

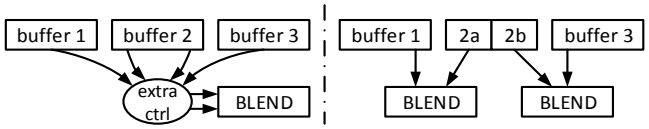

Fig. 8. (a) Serial Organization Way (b) Parallel Organization Way

manner, blend process can be done concurrently, which greatly reduces the complexity of control logic at the cost of one extra BLEND module. Furthermore, due to this simplification, the proposed design can be easily modified to support different number of cameras arrayed in different ways.

After blend process to one line is done, module CONTROL could start dumping final data. Since all the final data is stored in buffers, different clock domain can be adopted in this design if two-port rams are used as buffers. That is to say, a lower clock can be used to fetch data from camera due to restriction of board level communication; a faster clock can be used to export data to enhance the timing performance.

IV. Experimental Results

A. Stitching Results

Using 4 cameras arrayed like a 2x2 square as an example:

Fig. 9. Stitching results

B. Comparison with Other Works

The synthesis results are got under SIMC65 when the aim frequency is set to 400MHz. As mentioned above, the number of BLENDs and buffers depends on the number of cameras. For example, with mxn 720p cameras, m BLENDs are needed and the total size of original line buffers is 2mx720 pixels.

The timing performance can be worked out in the following way. When the neighboring boundary is 2, blend range is 64, $t_{PROCESS}$ should be 2132 cycles, if the output video has a horizontal length of 2000. Using equation 1, it takes 2559680 cycles to generate one 2000x1200 blended image from 4 channels of 1280x720 videos arrayed as a 2x2 square. In other words, this design could realize a real-time video stitching of 2000x1200@30fps when works at only 78MHz.

Thus, it can be inferred from figure 9 and table I, this design fulfills a satisfying 2-dimension stitching result with limited hardware resources and consuming time.

TABLE I
Comparison with Other Works

work	resources	timing performance	notes
this work	BLEND 1445gates CONTROL 2517gates FETCH 644gates	2000x1200@30fps from 2x2 channels of 1280x720	2-D 78MHz
[5]	embedded system R0P7724LC0011RL	XVGA@1.67fps from 2 channels	1-D 512MHz
[6]	DSP DM643	TV resolution @22fps from 2 channels	1-D 600MHz
[7]	Intel i7 3930K CPU	real time video stitching from 4 channels of D1	1-D 2.3GHz

V. Conclusions

This paper proposes a flexible and low-cost hardware implementation for real-time video stitching, which can be easily reconfigured or redesigned to support different numbers of cameras arrayed in different ways in 2 dimensions. When supports 4 cameras arrayed as a square, it cost only 5 small line buffers, which are one-port rams, and limited control and calculation resources to fulfill the stitching task. After configuration, it can be simply regarded as one camera with a higher resolution, generating frame valid, line valid and pixel data signals. The results of simulation and FPGA verification show no apparent artifacts in blended pictures. When works at only 78MHz, it still can realize a real-time video stitching of 2000x1200@30fps from 2x2 channels of 1280x720@30fps videos.

Acknowledgment

This paper is supported by National Natural Science Foundation of China (61306023), Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP, 20120071120021), STCSM(13511503400), National High Technology Research and Development Program (863,2012AA012001).

References

[1] H. Bay, T. Tuytelaars, L. Van Gool, "SURF: Speeded Up Robust Features", in *Computer Vision–ECCV*, pp. 404-417, 2006.

[2] http://www.ovt.com/products/

[3] A. A. Efros and W. T. Freeman, "Image Quilting for Texture Synthesis and Transfer", in *Proc. 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 341-346, 2001.

[4] Szeliski, R., "Image Mosaicing for Tele-Reality Applications", in *Proc. 2th IEEE Workshop on Applications of Computer Vision*, pp. 44-53, 1994.

[5] T. C. Chang, C. A. Chien, J. H. Chang, J. I. Guo, "A Low-Complexity Image Stitching Algorithm Suitable for Embedded Systems", in *IEEE International Conference on Consumer Electronics*, pp. 197-198, 2011.

[6] Z. K. Zhang, Z. H. Liu, J. B. Jiao, "DSP implementation of a multi-channel video display system with image stitching", in *IEEE Youth Conference on Information, Computing and Telecommunication*, pp. 204-207, 2009.

[7] K. C. Huang, P. Y. Chien, C. A. Chien, H. C. Chang and J. I. Guo, "A 360-Degree Panoramic Video System Design", in *International Symposium on VLSI Design, Automation and Test*, pp. 1-4, 2014.