A New Pipelined Architecture for Fuzzy Color Correction

Jer Min Jou , Shiann-Rong Kuang and Yeu-Horng Shiau

Department of Electrical Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.

Abstract

Color correction, which nonlinearly converts the color coordinates of an input device such as the scanner into that of an output device such as the printer, is important for multimedia applications. In this paper, we present a novel dynamic pipelined VLSI architecture for the fuzzy color correction algorithm proposed in [1] to meet the speed requirement of time-critical applications. To prompt the performance, the presented architecture is dynamically pipelined with unfixed latencies (or data initiation intervals), then the problem of impossible pipelining (and then slow executing) the fuzzy color correction algorithm due to the variable execution length of each iteration in it is solved completely. As a result, a significant (about 2 times) speed-up of the dynamic pipeline architecture with a slight hardware overhead relative to the sequential architecture has been achieved.

I. Introduction

The reproduction of color documents between different devices has become one of important problems in multimedia applications due to the rapid progress on the technologies of VLSI and color input/output devices such as scanning/printing devices. The major problem of a color correction system is how to maintain the original color quality of documents or images very faithfully when they are transferred between different input/output devices. The most common example is the color documents being scanned in the scanner and then being printed out by the printer, it also is this paper's focusing problem. However, since the nonlinear color mapping problem exists between them, the coordinates obtained by the color scanner can't be used directly by the color printer. Therefore, a mechanism to efficiently and fast convert the scanner's color coordinates into that of the printer is necessary and critical for a time-critical color environment.

In the past, there are some methods [2, 3] proposed to deal with color correction (or conversion) between the scanner and the printer. However, some of them are with high computation costs and/or need large storage area, the other are slow and unsuitable for time-critical applications. A slow fuzzy correction algorithm and a FPGA (also slow) implementation were presented in [4]. In [1], an efficient fuzzy-tree correction algorithm, denoted as FCC, is proposed. The advantages of FCC lie in its simplicity, adaptability, and good correction effect. A sequential hardware of it had also been implemented in [1]. But, its processing speed is still slow and is a problem for time-critical applications. Making a fast hardware realization for FCC to prompt the correction speed is then the subject of this paper.

In this paper, we present a dynamic pipelined architecture for FCC to increase the correction speed significantly. Pipelining [6-7] is a powerful and popular technique in designing high throughput digital circuits. For a functional pipeline design, consecutive iterations of the same loop are initiated at a time interval called the latency. In general, latency is fixed [6] or has some fixed values [7]. However, in the main FCC processing loop, variant execution length of each iteration and time-relative data dependencies between them make its pipeline latencies unfixed and hard to pipeline. In order to construct a high throughput pipelined architecture for FCC, we first identify and modify some complicated data dependencies in it such that pipelining is easy to carry out, and then FCC is partitioned into two sections. In the first section, the whole second section is viewed as an unbound delay operation whose delay is data-dependent. The two sections then are separately pipelined to form a combined dynamic pipeline. After dynamic pipelining, the pipeline latency of the FCC loop is depended on the execution length of the second section, and is unfixed naturally and varies between 5 to 13. The average value is about 9.5 (measured by the hardware simulation). Thus, the processing speed of FCC can be prompted significantly and is about 2 times speed-up to its sequential version only having a little hardware overhead.

II. Overview of FCC and Its Sequential Architecture

In [1], the fuzzy color correction process is modeled as a three-level fuzzy tree inference process. The function of each fuzzy subtree is to do one color conversion which is performed by finding a decision path in it. The basic structure of the FCC algorithm is outlined in Fig. 1 [1]. In it, each input color value X_i is processed with eight fuzzy sets (s_1, s_2, …, s_8) for the

color. L denotes the current level of the three-level fuzzy tree and $1 \le L \le 3$. $Path_L$ denotes the decision path of the current subtree and $0 \le Path_L \le 7$. Additionally, a 146 bytes ROM is used as the rule memory. The address for retrieving the s^1 (the supported value of fuzzy set s_1) and d (the fixed distance between any two neighbor fuzzy sets of each subtree) from ROM can be calculated as

$$\text{address} = \begin{cases} 144, & \text{if } L = 1, \\ 128 + Path_{L-1} * 2, & \text{if } L = 2, \\ Path_{L-2} * 16 + Path_{L-1} * 2, & \text{otherwise.} \end{cases} \quad (1)$$

The inference result X_o is calculated by

$$X_o = \begin{cases} w_1, & \text{if } k = 0, \\ \dfrac{|D| * w_k + (d - |D|) * w_{k+1}}{d}, & \text{if } 1 \le k \le 7, \\ w_8, & \text{if } k = 8. \end{cases} \quad (2)$$

where $|D|$ is the distance distance between X_i and the fuzzy set s_k, (w_1, w_2, …, w_8) are the supported values of the fuzzy sets which represent the various intensities of color space CMY. The execution length of each iteration in the FCC algorithm is variable because the iteration number I of the second loop in Fig. 1 is changed according to the input data ($0 \le I \le 8$). For more details, please refer to [1].

The corresponding sequential architecture [1] of FCC is shown in Fig. 2. It includes one comparator, one two's complementor, one adder, one subtractor, and one ALU which performs multiplication and division operations in Eq. (2). When the input X_i is transmitted to the architecture, it calculates the rule address according to level number L and Eq. (1), and then reads s^1 and d from ROM. The notation $<<n$ ($>>n$) in it denotes that the value is shifted left (right) n bits. Next, the architecture finds out the values of k and $|D|$ to determine $Path_1$ and $Path_2$. The storage used to store k is a 4-bit up-count counter to perform its increment operation. The condition $E8$($E0$) is set to 1 if k=8 (k=0). Another condition $L0$ is set to 1 if D>0. Finally, the center-average method is used for defuzzification by Eq. (2) and the defuzzification (inference) result X_o is obtained. In Fig. 2, the (w_1, w_2, …, w_8) are stored in a 8 bytes ROM (denoted as ROM2) and k is used as its address.

III. Dynamic Pipelined Architecture Design of FCC

The architecture of Fig. 2 performs FCC sequentially, and its processing speed is limited by the recursive dependence between different iterations and is slow. Conventionally, the processing speed of the sequential architecture can be improved by using pipelining techniques economically and efficiently. However, since the part of its operations with variable execution delay, FCC is difficult to be pipelined by using conventional pipelining techniques with a fixed latency. The dynamic pipelined architecture with variant latencies must be developed to pipeline FCC to prompt its performance.

To design a high performance dynamic pipelined FCC architecture with minimum hardware resource, all operations are first partitioned into two sections to increase the probability of pipelining and the chance of hardware sharing. The operations of the second while loop and the **if** instruction in Fig. 1 which make the FCC main loop execution length to be variable are grouped into the second section, and the remaining operations are grouped into the first section. The aim of the partition is to divide the unbounded second section from the first main loop so that they can be separately pipelined to combine a high performance dynamic pipeline. Note that the multiplication and division operations need two clock cycles and the other operations like addition, subtraction, or comparison need one clock cycle to execute by using the 0.8μm cell library [5].

After the original algorithm is partitioned, the second section of it is separately scheduled by applying performance optimization pipelining techniques to achieve a high performance. The second section is scheduled without regard to the interaction and precedence constraints between it and the first section temporarily. Its operations may be scheduled into 4 states by using the conventional pipeline scheduling: the operations in line 6 of Fig.1 are scheduled at the first state; the operations in line 7 of Fig.1 are scheduled at the second state; and the operations in line 8 are scheduled at the third and fourth states respectively. However, this schedule is very slow and can't be pipelined due to data dependency between operations $k < 8$ and k++ (see Fig. 1). To get a high performance schedule, we apply the concept of speculative computation in [8] to schedule the second section, and all operations in the second section are scheduled into one state S_α. However, at state S_α how to prevent some conditional operations such as operation $D = D - d$, whose execution depends the result of logical operation: (k<8 && D>0) in line 6, overrunning (when its executing condition is false) becomes a pressing problem. Some new control signals must be generated and relative circuits must be designed to solve it. Let $c6$ and $c8$ be the conditions produced by logical operations (k<8 && D>0) and ($1 \le k \le 7$ && $|D| \le d/2$) respectively (see Fig1 and Fig.2), and let L_D, L_Path_1, L_Path_2, and C_k be the original loading or up-counting control signals of registers D, $Path_1$, $Path_2$, and counter k, respectively. Signals L_D, L_Path_1, L_Path_2, and C_k are set to 1 by the controller when the FCC architecture reaches state S_α. To prevent the speculative overruns occurring, new signals must be designed to replace them to control respective registers or counters. Let their corresponding new control signals be L_D', L_Path_1', L_Path_2', and C_k',

respectively. In addition, let signal In_α be a control signal which indicates whether the FCC architecture is at state S_α and is set to 1 if it is at state S_α; otherwise, signal In_α is set to 0. Then, signal *L_D'* is generated by

$$L\_D' = \begin{cases} L\_D, & \text{if } In_\alpha = 0; \\ c6, & \text{otherwise.} \end{cases} \quad (3)$$

Signal L_Path_1' and L_Path_2' are generated by

$$L\_Path_L = \begin{cases} L\_Path_L, & \text{if } In_\alpha = 0; \\ 1, & \text{if } In_\alpha = 1 \text{ and } (c6 = 1 \text{ or } c8 = 1); \\ 0, & \text{if } In_\alpha = 1 \text{ and } c6 = 0 \text{ and } c8 = 0; \end{cases} \quad (4)$$

where $_L$ is equal to 1 or 2. Then, the circuits to implement Eq. (3) and Eq. (4) are shown in Fig. 3. By the above design, all operations of the second section can be scheduled speculatively and efficiently into state S_α, which makes the performance of the FCC architecture further high.

Subsequently, the first section will be pipelined and then combined with the second section to form a dynamic pipeline with variable latencies. Performance optimization of the first section is more important and difficult since the more complex interactive precedence relations between respective sections must be considered and the execution length of the second section is unknown (data dependent). We incrementally unwind the first section main loop, to pipeline the first section. After a polynomial (and in practice small) number of iterations have been unwinded and parallelized, a repeating pipeline body will provably emerge. During pipelining, the original location of the operations in the second section is replaced by an unbound delay operation whose delay is dependent on operation *c6* and *c8*. Fig. 4 shows the four times unwinding of the loop body. The operations of calculating the ROM address in each iteration must be scheduled after the unbound delay operation of its previous iteration due to the data dependency caused by $Path_1$ and $Path_2$. After the repeating pipeline body of the first section is found, all operations of different iterations or different sections which are executed at the same time are formed a state such as $S_1 \sim S_4$ as shown in Fig. 4. The latency of the circuit is equal four clock cycles plus the delay of the second section (i.e. the unbound delay operation) and is variable.

Finally, the state transition graphs of the two sections are combined into a final state transition graph as shown in Fig. 5. The dynamic pipelined data path for them is designed and is depicted in Fig. 6. In it, the ALU of Fig. 2 is split into a multiplier and a divider due to their activated time conflict.

IV. Comparing Results

Four pictures were used to test the processing speed of the dynamic pipelined design. The compared results with the sequential design are reported in Table 1. In it, columns "sequential" and "dynamic pipelining" show the total state number required for sequential and dynamic pipelined designs, respectively. The results show that about 2 times speedup can be obtained. The average latencies $\overline{L}$ for the dynamic pipelined design are also listed in Table 1.

Comparing the sequential and dynamic pipelined data paths shown in Fig. 2 and Fig. 6, the penalty paid for the speedup is extra area overhead including more pipelined registers and a somewhat larger controller. In addition, the total area of one multiplier and one divider used in the dynamic architecture is larger than the area of one ALU in the sequential architecture which can perform multiplication and division operations. In the future, how to reduce the area of data path such as using the table-look-up method to replace some larger functional units is our next goal.

V. Conclusions

This paper has presented a dynamic pipelined architecture for the fuzzy tree color correction algorithm. The data dependencies in the original sequential architecture have been analyzed and loosened by some simple techniques, and then a new dynamic pipelining architecture is designed to construct a high-throughput FCC circuit with variable latencies. It obtains about 2 times speedup with a slight area overhead.

References

[1] Jer-Min Jou, Shiann-Rong Kuang, and Ren-Der Chen, "A New Efficient Fuzzy Algorithm for Color Correction," will appear in *IEEE Transactions on Circuits & Systems Part I*, 1998.

[2] R. Takeuchi, M. Tsumura, M. Tadauchi, and H. Shio, "Color image scanner with an RGB linear image sensor," *J. Image Technology*, Vol. 14, pp. 68-72, 1988.

[3] H. R. Kang, and P. G. Anderson, "Neural network application to the color scanner and printer calibrations," *J. Electronic Imaging*, Vol. 1, pp. 125-135, 1992.

[4] B. D. Liu and C. Y. Huang, "Design and Implementation of the Tree-Based Fuzzy Logic Controller," *IEEE Trans. on Systems, Man, and Cybernetics—Part B: Cybernetics*, Vol. 27, No. 3, pp. 475-487, June 1997.

[5] 0.8 μm SPDM technology manual, Computer & Communication Laboratory, Industry Technology Research Institute, R.O.C., 1993.

[6] N. Park and A. C. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 3, pp. 356-370, March 1988.

[7] H. S. Jun and S. Y. Hwang, "Automatic Synthesis of

Dynamically Configured Pipelines Supporting Variable Data Initiation Intervals," *IEEE Trans. on VLSI Systems*, Vol. 4, No. 2, pp. 279-285, July 1996.

[8] U. Holtmann and R. Ernst, "Experiments with Low-Level Speculative Computation Based on Multiple Branch Prediction," *IEEE Trans. on VLSI Systems*, Vol. 1, No. 3, pp. 262-267, 1993.

```
1:  L=1;
2:  while (input pattern X_i ≠ NULL) {
3:      Calculate the address of ROM using Eq. (1);
4:      s^l=ROM[address++];   D= X_i − s^l;
5:      k=0;   Path_L=0;   d=ROM[address];
6:      while (k<8 && D>0) {
7:          D= D − d;   Path_L=k;   k++;
        }
8:      if (1 ≤ k ≤ 7 && |D| ≤ d/2) Path_L=k;
9:      Calculate X_o using Eq. (2);
10:     if (++L==4) L=1;
    }
```

Fig. 1. The fuzzy-tree color correction algorithm.

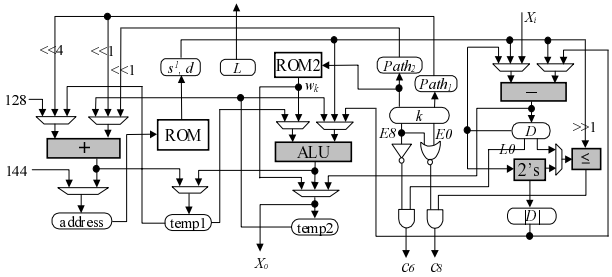

Fig. 2. The sequential architecture of FCC algorithm.

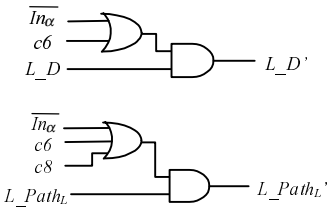

Fig. 3. Circuits generating signals *L_D'* and L_Path_L'.

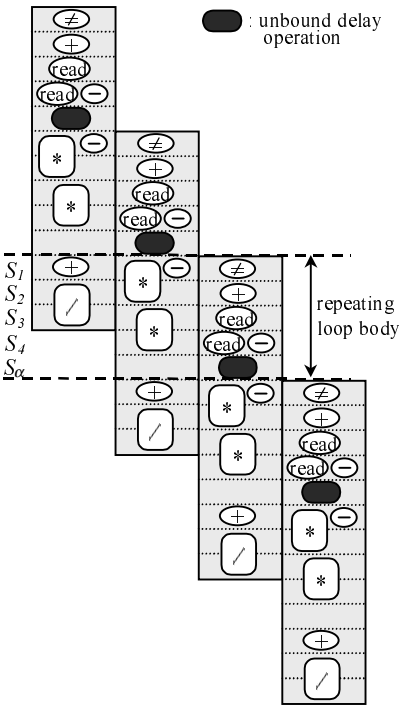

Fig. 4. Result of unwinding of the first section four times.

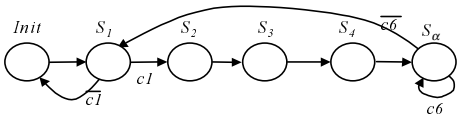

Fig. 5. The final state transition graph.

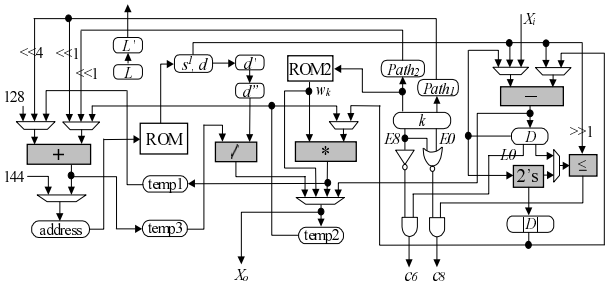

Fig. 6. Dynamic pipelined architecture of FCC algorithm.

Table 1. The comparing results of the color correctors.

pictures	file size (bytes)	sequential	dynamic pipelining	$\bar{L}$	speedup
Pic1	148416	2704900	1370164	9.25	1.97
Pic2	230604	4345076	2193924	10.39	1.98
Pic3	974916	16898438	8139846	8.35	2.08
Pic4	1137198	20268056	10356836	9.11	1.96