A novel hardware-software co-design for automatic white balance

Chin-Hsing Chen, Sun-Yen Tan, and Wen-Tzeng Huang

***Abstract*—**As electronic techniques is continuous improved rapidly cameras or video camcorders used for image retrieval technology and development become digitalized. The color of the photographs would look very different due to differences in light projection illumination when we take a picture. Human eyes are able to automatically adjust the color when the illuminations of the light source vary. However, the most frequently used image sensor, charge coupled device, CCD device can not correct the color as human eyes. This paper presents a hardware-software co-design method based on Lam's automatic white balance algorithm, which combines both Gray World Assumption and Perfect Reflector Assumption algorithms. The execution steps of Lam's algorithm were divided into three stages. The hardware-software co-design and analysis for each stage was realized. Three factors including processing time, Slices and DSP48s of hardware resources were used to formulate the objective function, which was employed to evaluate the system performance and hardware resource cost. Experimental results shows suitable partitions of hardware-software co-designs were achieved. An embedded processor, MicroBlaze developed by Xilinx and a floating point processor were used to deal with the software part of the algorithm. The hardware part of the algorithm was implemented using an IP-based method. It is able to reduce the memory and CPU resources of PC as well as to have the properties of easy modification and function expansion by using such system on programmable chip architecture.

***Keywords*—**Automatic white balance, embedded processor, hardware-software co-design, system on a programmable chip.

I. Introduction

GRAY World Assumption (GWA) [13] and Perfect Reflector Assumption (PRA) [14] are two common methods used to realize automatic white balance algorithms. GWA can offer a better effect for photographs with rich color environment and background due to its characteristics. However, processed images may have an undesirable shift in the entire color range when the acquired images are with a predominant color. On the contrary, the images with a predominant color can be processed without a cast as people can see if PRA is applied. However, PRA may not correctly deal with the automatic white balance of images with multicolor. The processed images may have an undesirable shift in the entire color range. In 2005, Lam proposed an automatic white balance algorithm which combines both GWA and PRA. The algorithm can deal with automatic white balance of images with both multicolor and a predominant color correctly [1].

Three approaches may be used to construct image retrieval platforms dealing with white balance algorithms. The first approach contains storing acquired images into memory and executing specific software program in computer hardware resources to process the images. Operating systems and application programs as well as some memory and CPU resources are required when this method is applied. The second approach contains some additional image processing ASICs on the image retrieval platforms. Such ASICs can be applied to deal with images processes. However, the image processing functions is limited by their specifications. The third approach is to construct image retrieval platforms using SoPC. This method usually contains field-programmable gate array (FPGA) chips and embedded processors with Intellectual Property (IP) based hardware designs. This design methodology may not consume large amount of memory and CPU resources of PC as well as to have the properties of easy modification and function expansion, which can not be achieved if pure hardware architectures are used. Some ICs may be replaced by such SoPC architecture to contain their functions and reduce the difficulty of the PCB designs [2].

In this research, the Xilinx's ML402-Virtex-SX35 development board was used. The embedded processor is using MicroBlaze Soft Core developed by Xilinx. A floating point implementation was used to deal with the software part of Lam's algorithm. The hardware part of Lam's algorithm was implemented using an IP-based design.

The paper is organized as follows: In Section II, we review the automatic white balance algorithms. Then, in Section III, we describe our implementing methods for Lam's algorithm, including hardware-software partitioning and the evaluation of the cost functions. In Section IV, experimental results are presented to show the implementations and a brief comparison is discussed. Finally, a conclusion is given in Section V.

Manuscript received March 4, 2007; Revised received August 13, 2007.

Chin-Hsing Chen is with the Department of Management Information Systems, Central Taiwan University of Science and Technology, Taichung, Taiwan (e-mail: chchen@ctust.edu.tw).

Sun-Yen Tan is with the Electronic Engineering Department, National Taipei University of Technology, Taipei, Taiwan (phone: 886-2-27712171; fax: 886-2-27317120; e-mail: sytan@ntut.edu.tw).

Wen-Tzeng Huang is with the Electronic Engineering Department, National Taipei University of Technology, Taipei, Taiwan (e-mail: wthuang@ntut.edu.tw).

II. Procedure for Paper Submission

Within this paper $\hat{I}_R$, $\hat{I}_G$, and $\hat{I}_B$ are used to denote the red, green, and blue values after image processing as well as I_R, I_G, and I_B are used to denote the red, green, and blue values before image processing, respectively.

A. Gray World Assumption

Currently GWA algorithm is one of the most frequently used automatic white balance algorithms. The algorithm is based on the assumption, "all photographs acquired by cameras are colorful images.". In other words, the occurrence probabilities of red, green, and blue pixels of a picture are the same. The gray level of a color is composed from the averages of red, green, and blue colors. For real situations the shooting pictures are usually with colorful enough. It matches the assumption of GWA. Therefore, it is necessary to adjust each average of red, green, and blue colors to be the same when we use GWA algorithm. An image consists of brightness and chromaticity information. Human eyes are more strongly sensitive to the brightness than to the chromaticity of an image.

$$\hat{I}_R(x,y) = K_R I_R(x,y) \tag{1}$$

$$\hat{I}_B(x,y) = K_B I_B(x,y) \tag{2}$$

$$K_R = \frac{G_{avg}}{R_{avg}} = \frac{\dfrac{\sum_{i=0}^{m}\sum_{j=0}^{n} I_G(x,y)}{mn}}{\dfrac{\sum_{i=0}^{m}\sum_{j=0}^{n} I_R(x,y)}{mn}} \tag{3}$$

Equations (1) and (3) show the main computations of the red channel [3]. The similar computation is shown in (2). It can be used to obtain the blue value for the blue channel [3]. The advantage of the GWA algorithm is to have a better recovery of the original appearance of the scene when the input images are colorful. However, processed images may have an undesirable shift in the entire color range when the acquired images are with a predominant color.

B. Perfect Reflector Assumption

PRA is another famous algorithm to deal with automatic white balance. Let us discuss the relationship between lights and object colors first. Objects may not show colors themselves. However, their colors can be shown through different wavelengths of the radiations from the illumination in absorption, reflection, or transmission. We could not see any object colors if there is no radiation from the illumination. On the other hand, the object color is white if all the radiations are reflected. Therefore, white color objects or regions are called as perfect reflectors. The PRA theory assumes that perfect reflectors can be used as the reference value of a white color in dealing with an acquired image. The red, green, and blue for a white color object inside any color temperature image could be the maximum values. To achieve automatic white balance of images the perfect reflector may be used as a reference to correct other colors.

$$\hat{I}_R(x,y) = K_{MaxR} * I_R(x,y) \tag{4}$$

$$\hat{I}_B(x,y) = K_{MaxB} * I_B(x,y) \tag{5}$$

$$K_{MaxR} = \frac{\underset{x,y}{Max}\{I_G(x,y)\}}{\underset{x,y}{Max}\{I_R(x,y)\}} \tag{6}$$

Equations (4) and (6) show the main computations of the red channel. The similar computation shown in (5) can be used to obtain the blue value for the blue channel [3]. The advantage of PRA algorithm is to have a better recovery of the original appearance of the scene when the input images are with a predominant color. However, processed images may have an undesirable shift in the entire color range when the acquired images are with multicolor.

C. 2005 Lam's algorithm

Both PRA and GWA algorithms still have the disadvantages as mentioned above. Lam proposed an automatic white balance method which combines above two algorithms in 2005. In this paper the method is called as Lam algorithm. Lam presented two equations, as shown in (7) and (8), to deal with automatic white balances. The main computations for the coefficients of the red channel are shown in (9) to (12). The similar computation is shown in (8). It can be used to obtain the blue coefficients for the blue channel [1].

$$\hat{I}_R(x,y) = u_R I_R^2(x,y) + v_R I_R(x,y) \tag{7}$$

$$\hat{I}_B(x,y) = u_B I_B^2(x,y) + v_B I_B(x,y) \tag{8}$$

$$\sum_{x=1}^{M}\sum_{y=1}^{N} \hat{I}_R(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} I_G(x,y) \tag{9}$$

$$\underset{x,y}{Max}\{I_R(x,y)\} = \underset{x,y}{Max}\{I_G(x,y)\} \tag{10}$$

$$u_R \sum_{x=1}^{M}\sum_{y=1}^{N} I_R^2(x,y) + v_R \sum_{x=1}^{M}\sum_{y=1}^{N} I_R(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} I_G(x,y) \tag{11}$$

$$u_R \underset{x,y}{Max}\{I_R^2(x,y)\} + v_R \underset{x,y}{Max}\{I_R(x,y)\} = \underset{x,y}{Max}\{I_G(x,y)\} \tag{12}$$

Equations contain the processing of the squares of pixels. Therefore, the processed images have the property of enhancing the contrast of images. As with PRA and GWA, the values of the green channel are kept unchanged. Only the values of the red and blue channels are adjusted. Lam's algorithm has the advantages of both GWA and PRA. When input images with multicolor the processed images can have the effect as well as the GWA method. When input images with a predominant color the processed images can not have the effect as well as the PRA method. But it removes the saturation condition during the GWA algorithm processing. Therefore, Lam's algorithm is better than GWA and PRA to be a suitable method used for dealing with automatic white balances of any different images.

III. Hardware-software co-design

The execution steps of Lam's algorithm were divided into three stages. The first stage is the pre-processing of the Lam's algorithm. It is to obtain the required parameters of the computations for Lam's algorithm. The second stage is to obtain the solutions of the equations of Lam's algorithm. There two common methods to be applied to obtain the solutions. One is by Gauss Elimination. Another is by Cramer's Rule. The third stage is to deal with the automatic white balance computations for each pixel. The hardware-software co-design and analysis for each stage was realized and evaluated. The partitions of these three stages are shown in Fig. 1. When the computations of the parameters for the first stage were implemented using hardware each pixel needs 5 adders, 2 multipliers and 3 comparators.

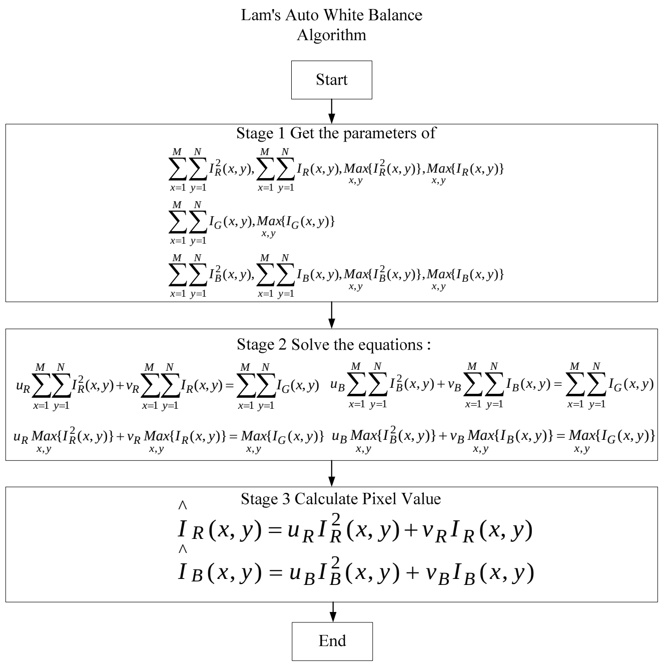
Lam's Auto White Balance Algorithm
Start
Stage 1 Get the parameters of
$\sum_{x=1}^{M}\sum_{y=1}^{N} I_R^2(x,y), \sum_{x=1}^{M}\sum_{y=1}^{N} I_R(x,y), \max_{x,y}\{I_R^2(x,y)\}, \max_{x,y}\{I_R(x,y)\}$
$\sum_{x=1}^{M}\sum_{y=1}^{N} I_G(x,y), \max_{x,y}\{I_G(x,y)\}$
$\sum_{x=1}^{M}\sum_{y=1}^{N} I_B^2(x,y), \sum_{x=1}^{M}\sum_{y=1}^{N} I_B(x,y), \max_{x,y}\{I_B^2(x,y)\}, \max_{x,y}\{I_B(x,y)\}$
Stage 2 Solve the equations :
$u_R\sum_{x=1}^{M}\sum_{y=1}^{N} I_R^2(x,y) + v_R\sum_{x=1}^{M}\sum_{y=1}^{N} I_R(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} I_G(x,y)$
$u_B\sum_{x=1}^{M}\sum_{y=1}^{N} I_B^2(x,y) + v_B\sum_{x=1}^{M}\sum_{y=1}^{N} I_B(x,y) = \sum_{x=1}^{M}\sum_{y=1}^{N} I_G(x,y)$
$u_R \max_{x,y}\{I_R^2(x,y)\} + v_R \max_{x,y}\{I_R(x,y)\} = \max_{x,y}\{I_G(x,y)\}$
$u_B \max_{x,y}\{I_B^2(x,y)\} + v_B \max_{x,y}\{I_B(x,y)\} = \max_{x,y}\{I_G(x,y)\}$
Stage 3 Calculate Pixel Value
$\hat{I}_R(x,y) = u_R I_R^2(x,y) + v_R I_R(x,y)$
$\hat{I}_B(x,y) = u_B I_B^2(x,y) + v_B I_B(x,y)$
End

Fig. 1 Three stages of Lam's algorithm

The hardware implementation of the red channel of the data-path model is shown in Fig. 2. The same hardware was also used for the blue channel. The similar hardware was used for the green channel. There was no summation of multipliers inside this similar hardware.

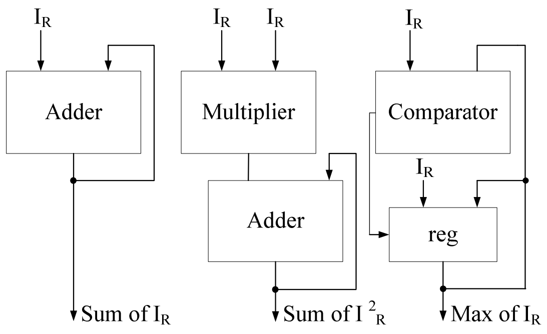

Fig. 2 The hardware model for obtaining the red channel parameters (stage 1)

The computation of the second stage is to obtain the solutions of the linear equations with two unknown. Four methods were discussed in this paper. They are Guass Elimination Software, Guass Elimination Hardware, Cramer's Rule Software, and Cramer's Rule Hardware. Floating point computations are required for the second and the third stages. Therefore, the term computation is for the floating point computation. The employed computation blocks were floating point operators [7].

When Guass Elimination Software was applied 6 dividers, 6 multipliers, and 6 sub-tractors were required for a frame. The hardware implementation of the red channel of the data-path model is shown in Fig. 3. The same hardware was also used for the blue channel. When Cramer's Rule Software was applied 12 multipliers, 6 sub-tractors, and 4 dividers were required for a frame. The hardware implementation of the red channel of the data-path model is shown in Fig. 4. The same hardware was also used for the blue channel.

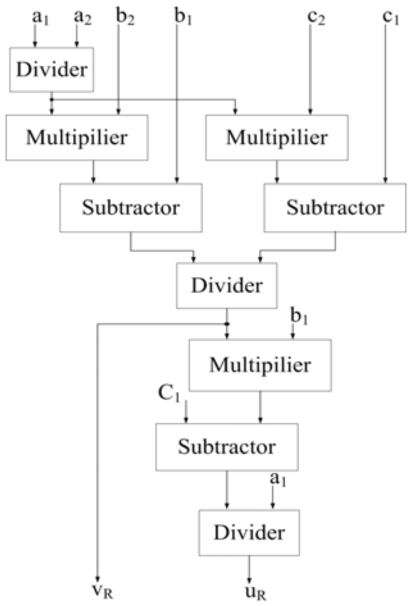

Fig. 3 The hardware implementation of Guass Elimination for the red channel parameters u_R and v_R (stage 2)

The computations of the parameters for the third stage is write back the pixel data when the software implementation were applied. 3 multipliers and 1 adder were required for a pixel. The hardware implementation of the red channel of the data-path model is shown in Fig. 5. The same hardware was also used for the blue channel.

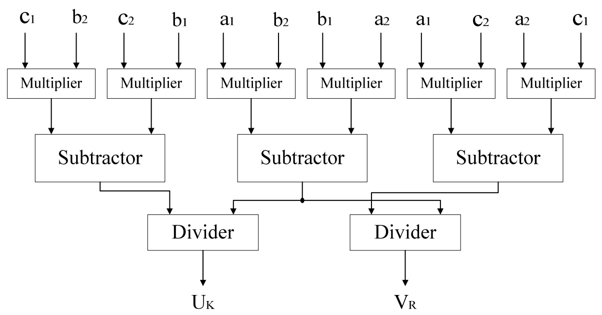

Fig. 4 The hardware implementation of Cramer's Rule for the red channel parameters u_R and v_R (stage 2)

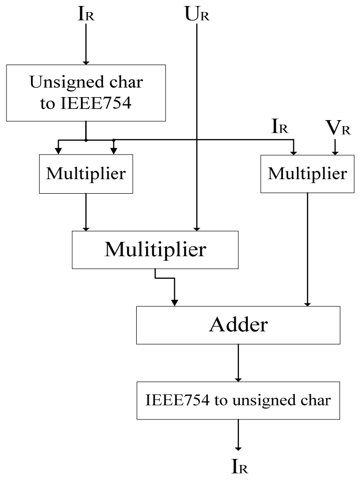

Fig. 5 The data-path model of writing back the pixel data

A. *Sixteen hardware-software partitions*

There sixteen partitions were used to implement SOPC to deal with the Lam's algorithm. Three factors including processing times, usages of Slices and DSP48 of hardware resources were used to formulate a cost function, which was employed to evaluate the system performance and hardware resource cost. Each Slice contains a set of D-FlipFlop and 4-bit programmable logic. Table I shows these sixteen hardware-software partitions.

B. *Software operations using floating-point operation units*

The employed algorithm contains floating point operations. Micro-Blaze embedded processor 4.0 version [8] can support additional floating point operation units. Additions, subtractions, multiplications, and divisions of the floating point operations can speed-up from 15 to 266 times [9]. In this paper, MicroBlaze embedded processor 4.0 version with floating point operation units were used for the software implementation.

TABLE I
SIXTEEN HARDWARE-SOFTWARE PARTITIONS

	Stage1	Stage2	Stage3
Case1	SW	Gauss Elimination SW	SW
Case2	SW	Gauss Elimination SW	HW
Case3	SW	Gauss Elimination HW	SW
Case4	SW	Gauss Elimination HW	HW
Case5	SW	Cramer's Rule SW	SW
Case6	SW	Cramer's Rule SW	HW
Case7	SW	Cramer's Rule HW	SW
Case8	SW	Cramer's Rule HW	HW
Case9	HW	Gauss Elimination SW	SW
Case10	HW	Gauss Elimination SW	HW
Case11	HW	Gauss Elimination HW	SW
Case12	HW	Gauss Elimination HW	HW
Case13	HW	Cramer's Rule SW	SW
Case14	HW	Cramer's Rule SW	HW
Case15	HW	Cramer's Rule HW	SW
Case16	HW	Cramer's Rule HW	HW

C. *Hardware operations using floating point operators*

The software for realizing the design and synthesis of the partitioned hardware is using Xilinx Integrated Software Environment 7.1 iSP2. The floating point operators used for the second and the third stages are constructed using Floating Point Operator v1.0 [7] generated by Xilinx CoreGenerator. The width of the buses, data bit definitions, and operator types of the floating point operations can be defined using adjusting the setup of CoreGenerator. In this paper the format of the floating point operations was set as single precision of IEEE-754 standard. One bit is used as the sign bit. 8 bits are used as the exponent. 23 bits are used as the mantissa.

D. *Defining an objective function*

An objective function [4] which was used to evaluate the system is shown in Eq (13). Two factors were considered in our systems. They are processing times and usages of hardware resources. To evaluate the usages of hardware resources the usages of hardware resources are further divided into the usages of Slices and the usages of DSP48 slices (DSP48s). To realize the objective function and calculate it for various conditions of the partitions ***time***, ***slices*** and ***DSP48s*** denote the maximum values of the sixteen partitions, respectively. Let $\boldsymbol{time_{real}}$, $\boldsymbol{slices_{real}}$, and $\boldsymbol{DSP48s_{real}}$ denote the actual value for each partition, respectively.

After computing the sixteen evaluation values can be obtained from the Objective Function. If the evaluation value is closed to 1 it means that the system can have the expected performance with a faster execution time, a smaller amount of Slices, and a smaller amount of DSP48s. The parameters, α , β , and γ of (13) are used to achieve a balance of the execution efficiency and the hardware resources. They represent the specified weights of the execution time, the usage of Slices, and the usage of DSP48s. In this research, we set $\alpha = \frac{1}{2}$, $\beta = \frac{1}{4}$, and $\gamma = \frac{1}{4}$.

The applications of the automatic white balance algorithms of digital images are very wide, such as digital cameras, image capturing cards, smart cameras, etc. For different performance requirements the three parameters of the objective function can be modified to meet the desired constraints.

$$\text{Objective Function} = \frac{time - time_{real}}{time} * \alpha + \frac{slices - slices_{real}}{slices} * \beta + \frac{DSP48s - DSP48s_{real}}{DSP48s} * \gamma \quad (13)$$

To enable readers understanding the objective function, an example is used to explain it. First the weight ratio of the processing time and hardware resources were set to be 1:1 (see Eq (14)). It was not only for obtaining execution efficiency but also for reducing hardware resources. Then the sum of the weights were set to be 1 (see Eq (15)). Similarly, it is to achieve that the weights of Slices and DSP48s of the hardware are the same. That is, $\beta = \frac{1}{4}$ and $\gamma = \frac{1}{4}$.

$$\alpha : (\beta + \gamma) = 1 : 1 \quad (14)$$

Let $\alpha + \beta + \gamma = 1$

$$=> \alpha = \frac{1}{2}, (\beta + \gamma) = \frac{1}{2} \quad (15)$$

Alternatively, we define a cost function to show the increasing percentage ratio of the execution efficiency and the cost. It is shown in (16).

$$\text{Cost Function} = \frac{\text{execution efficiency increment\%}}{\text{hardware resource increment\%}} = \frac{\frac{\text{time-time}_{real}}{\text{time}}}{1-(\frac{\text{slices-slices}_{real}}{\text{slices}} * \frac{1}{2} + \frac{\text{DSP48s-DSP48s}_{real}}{\text{DSP48s}} * \frac{1}{2})} \quad (16)$$

In the cost function the numerator denotes the efficiency improvement variation of the execution time. The denominator denotes the cost increment caused by the usage increment of the hardware resources (Slices and DSP48 slices). The greater ratio values mean that the design has shorter execution time and lower cost.

IV. Experimental Results

A typical system architecture was used to implement the system for this research [5]. MicroBlaze is a Soft Core processor developed by Xilinx. The DDR SDRAM Controller is an external memory controller. The On Chip Peripheral (OPB) is a peripheral bus. The IP Interface is a bus interface. The CameraLink Deserial is an input interface for digital cameras. The Region of Interesting (ROI) denotes the some region processing circuits. The Auto White Balance Stage1 HW and Stage3 HW denote the designed circuits for the partition for Case14.

The flow for fetching images is shown in Fig. 6. The first step is that the SVS282 camera sent out Bayer arrangement CCD sensed images [10] to the Low Voltage Differential Signal (LVDS) pin [11] of the ML402 development board by CameraLink interface. The second step is that the Virtex4-SX35 FPGA image chip executes the function of the Region of Interesting (ROI) and stores the fetched images into the Double Data Rate Synchronous Dynamic Random Access Memory (DDR SDRAM) of the development board to finish the image fetching processing. The third step is to execute the automatic white balance algorithms. Due to the automatic white balance processed images are still CFA color patterns [12]. For the verification and the convenience of observations, the fourth step is to execute the pixel generation of the differences of adjacent pixels using MicroBlaze. Various software-hardware partitioning implementations were downloaded into FPGA by the Joint Test Action Group (JTAG) interface and estimated experimental results are shown in this section. To verify the experimental results the images were transmitted to the computer via the Universal Asynchronous Receiver/Transmitter (UART) interface. Then a Visual Basic application program was used to receive the automatic white balance processed images at the computer end. They were stored into the files using Bitmap (BMP) format. The whole experimental environment is shown as Fig. 7.

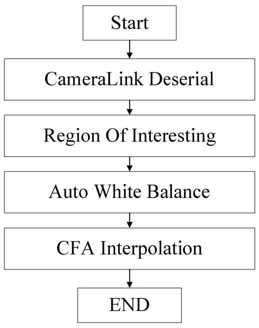

Fig. 6 The flow of fetching images

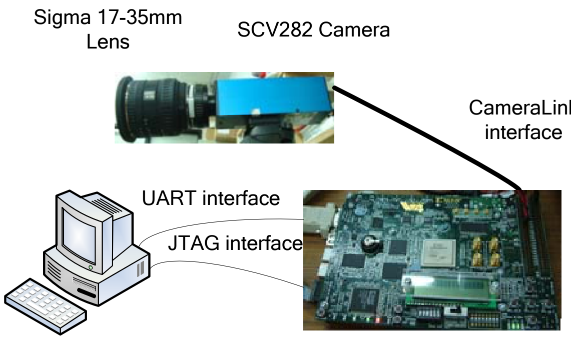

Fig. 7 The system connections and the whole experimental

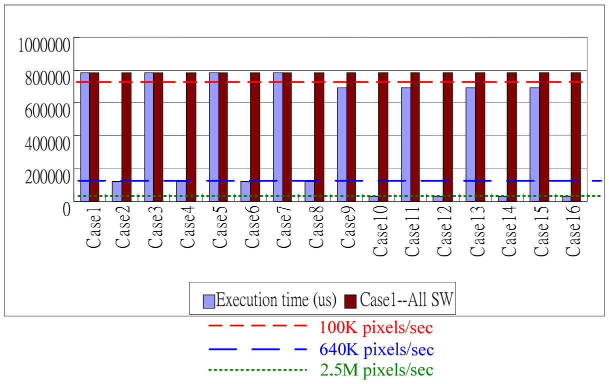

Fig. 8 Execution time comparisons of 16 different partitions and the pure software implementation case (Case1)

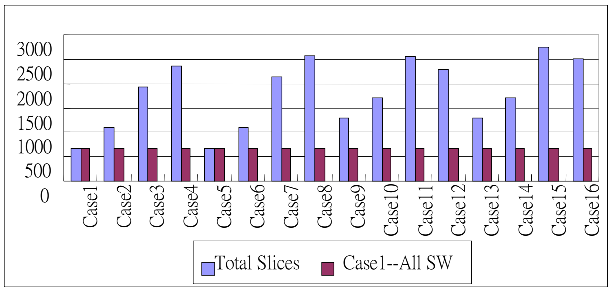

Fig. 9 Comparisons of hardware resource Slices usages of 16 different partitions and the pure software implementation case (Case1)

In the experiments the execution time was counted for the 320 x 240 resolution images. The experimental statistics of execution times, Slices, and DSP48s are shown in Figures 8, 9 and 10 respectively. The evaluation values of objective function and the cost function are shown in Table II and Fig. 11.

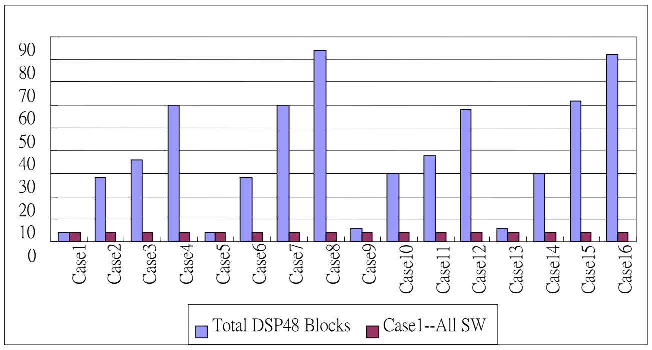

Fig. 10 Comparisons of hardware resource DSP48 usages of 16 different partitions and the pure software implementation case (Case1)

As shown evaluation of the objective function, Case6 and Case2 both were rank one. Case10 and Case14 both were rank two. Both Case2 and Case6 implemented the computations of the first stage using the software. They also implemented the computations of the second stage using the software. They implemented the computations of the third stage using the hardware. However, they used different methods for the solutions in the second stage. Their performance is able to deal with a 640 k pixels picture per second. It is equal to have an 800 x 800 resolution. The difference of the evaluations between the best case and Case10 and Case14 is only 0.0021. Their performance is able to deal with a 2.5 M pixels picture per second. It is equal to have a 1580 x 1580 resolution. The difference between Case10 and Case14 is using different methods for the solutions in the second stage. Case14 is 0.49 μs faster than Case10 under the same amount of hardware resource usages. For the requirements of several M pixels digital picture resolutions we decided to take Case14 which is able to deal with a 2.5 M pixels picture per second as well as have a balance between performance and hardware resources. The system architecture is shown in Fig. 12.

As shown in Table III the execution time of Case14 is 30000.747 μs. It is 26 times faster than the pure software case (782701.244 μs). A floating point unit (FPU) is about 678 Slices. As shown in Table IV the hardware resources of several cases are equal to 4 times of a FPU.

For parallel processing the second stage data the hardware implementation of the stage required 1266 Slices if Gauss elimination method is applied. Alternatively, it required 1474 Slices if Cramer's rule is applied. The Slice usage of both hardware is over 2 times of a FPU. The hardware execution time of Gauss elimination method was about 9.26 μs. The hardware execution time of Cramer's rule was about 6.5 μs.

TABLE II
OBJECTIVE FUNCTION AND COST FUNCTION

	Stage 1	Stage 2	Stage 3	Objective Function	Cost Function
Case1	SW	Gauss Elimination SW	SW	0.4267	0.0001
Case2	SW	Gauss Elimination SW	HW	0.7384	2.3018
Case3	SW	Gauss Elimination HW	SW	0.2169	0
Case4	SW	Gauss Elimination HW	HW	0.5286	1.0728
Case5	SW	Cramer's Rule SW	SW	0.4267	0.0001
Case6	SW	Cramer's Rule SW	HW	0.7384	2.3018
Case7	SW	Cramer's Rule HW	SW	0.1266	0
Case8	SW	Cramer's Rule HW	HW	0.4384	0.8724
Case9	HW	Gauss Elimination SW	SW	0.4248	0.4405
Case10	HW	Gauss Elimination SW	HW	0.7365	1.9679
Case11	HW	Gauss Elimination HW	SW	0.2149	0.1720
Case12	HW	Gauss Elimination HW	HW	0.5999	1.2624
Case13	HW	Cramer's Rule SW	SW	0.4248	0.4405
Case14	HW	Cramer's Rule SW	HW	0.7365	1.9679
Case15	HW	Cramer's Rule HW	SW	0.1247	0.1363
Case16	HW	Cramer's Rule HW	HW	0.5097	1.0206

TABLE III
TOTAL EXECUTION TIME

	Stage 1	Stage 2	Stage 3	Total execution Time (μs)
Case1	SW	Gauss Elimination SW	SW	782701.244
Case2	SW	Gauss Elimination SW	HW	122701.244
Case3	SW	Gauss Elimination HW	SW	782709.260
Case4	SW	Gauss Elimination HW	HW	122709.250
Case5	SW	Cramer's Rule SW	SW	782700.747
Case6	SW	Cramer's Rule SW	HW	122700.747
Case7	SW	Cramer's Rule HW	SW	782706.500
Case8	SW	Cramer's Rule HW	HW	122706.500
Case9	HW	Gauss Elimination SW	SW	690001.244
Case10	HW	Gauss Elimination SW	HW	30001.244
Case11	HW	Gauss Elimination HW	SW	690009.260
Case12	HW	Gauss Elimination HW	HW	30009.250
Case13	HW	Cramer's Rule SW	SW	690000.747
Case14	HW	Cramer's Rule SW	HW	30000.747
Case15	HW	Cramer's Rule HW	SW	690006.500
Case16	HW	Cramer's Rule HW	HW	30006.500

From the evaluations of the objective function the sequence of the optimal solutions is Case2 = Case6 > Case14 = Case10 > Case12 > Case4 > Case16 > Case8 > Case1 = Case5 > Case9 = Case13 > Case3 > Case11 > Case7 > Case15. From the evaluations of the cost function the sequence of the optimal solutions is Case2 = Case6 > Case14 = Case10 > Case12 > Case4 > Case16 > Case8 > Case9 = Case13 > Case11 > Case15 > Case1 = Case5 > Case3 = Case7.

TABLE IV
TOTAL USAGES OF HARDWARE RESOURCES (SLICES)

	Stage 1	Stage 2	Stage 3	Total Slices
Case1	SW	Gauss Elimination SW	SW	678
Case2	SW	Gauss Elimination SW	HW	1103
Case3	SW	Gauss Elimination HW	SW	1944
Case4	SW	Gauss Elimination HW	HW	2369
Case5	SW	Cramer's Rule SW	SW	678
Case6	SW	Cramer's Rule SW	HW	1103
Case7	SW	Cramer's Rule HW	SW	2152
Case8	SW	Cramer's Rule HW	HW	2577
Case9	HW	Gauss Elimination SW	SW	1288
Case10	HW	Gauss Elimination SW	HW	1713
Case11	HW	Gauss Elimination HW	SW	2554
Case12	HW	Gauss Elimination HW	HW	2301
Case13	HW	Cramer's Rule SW	SW	1288
Case14	HW	Cramer's Rule SW	HW	1713
Case15	HW	Cramer's Rule HW	SW	2762
Case16	HW	Cramer's Rule HW	HW	2509

View these two evaluations the sequences are very similar. The difference is that the cost function evaluations of Case1, Case5, Case3, and Case7 are close to the worst case. Due to the execution efficiency increment percentage approach to 0 they are at the last four places in the sequence. On the other hand, due to the objective function is with the linear accumulation property and these four cases consumed less hardware resource they are not at the last four places in the objective function evaluation sequence. Due to the objective function is with the linear accumulation property it is able to show the system performance at the execution efficiency increment and the hardware economy. The optimal solution of the objective function evaluation is also with the optimal ratio of efficiency and cost.

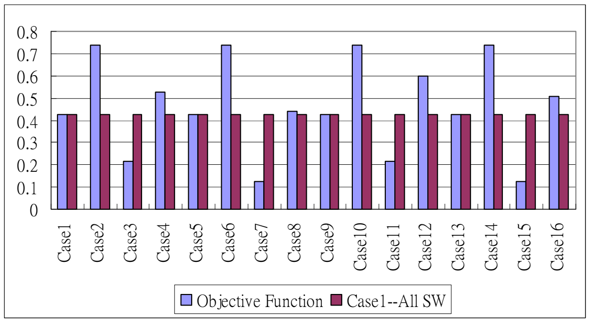

Fig. 11 Evaluations

For the experiment Philips PL18114 table lamp and PL-L827 fluorescent tube were used. As shown in Fig. 13 Color boards, dolls, thermos mugs were put inside the images to view the situations of processed images by Lam's automatic white balance algorithm.

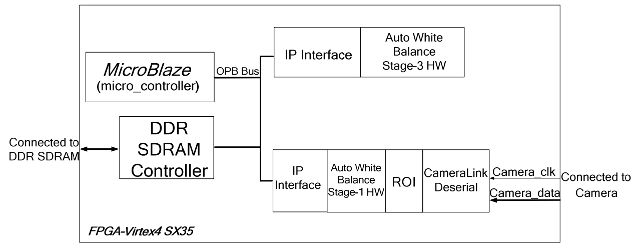

Fig. 12 The system architecture

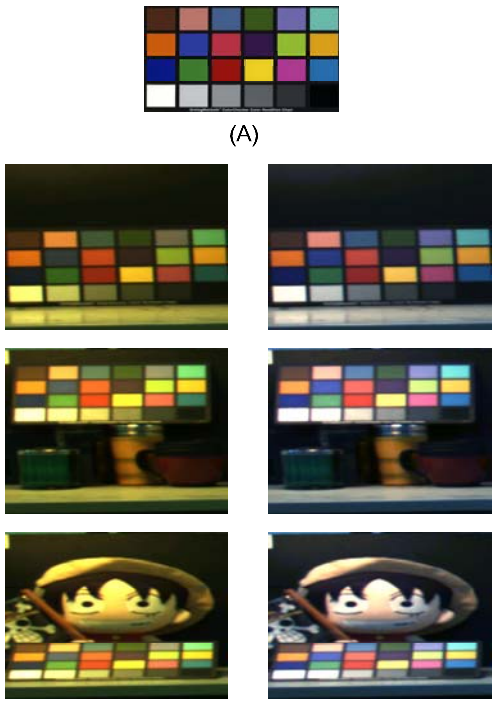

Fig. 13 2700° K illuminant experiments. (A) Color board, (B) original images, and (C) automatic balance processed images

Color temperature 2700° K was used for the experiment. Fig. 13 (A) shows a standard color board. Fig. 13 (B) shows the original fetched images. Fig. 13 (C) shows the images which were processed by the automatic white balance SoPC. It is clearly shown that the color board images in Fig. 13 (C) are close to the color board shown in Fig 13 (A). Compare colors of gray scales at the fourth rows of color boards shown in Fig. 13 (A), (B), and (C). The color of gray scale at the fourth row of the color board shown in Fig. 13 (B) biased to yellow. This is a feature of images with low color temperature.

After we computed the cost function of (13) using the actual execution times and the usages of Slices and DSP48s for the sixteen partitions. The evaluation values of sixteen cases were obtained. They are shown in Table V and Fig. 11.

TABLE V
THE COST FUNCTION EVALUATIONS

	Stage 1	Stage 2	Stage 3	Evaluations
Case1	SW	Gauss Elimination SW	SW	0.4264
Case2	SW	Gauss Elimination SW	HW	0.7486
Case3	SW	Gauss Elimination HW	SW	0.2143
Case4	SW	Gauss Elimination HW	HW	0.5365
Case5	SW	Cramer's Rule SW	SW	0.4264
Case6	SW	Cramer's Rule SW	HW	0.7486
Case7	SW	Cramer's Rule HW	SW	0.1223
Case8	SW	Cramer's Rule HW	HW	0.4445
Case9	HW	Gauss Elimination SW	SW	0.4243
Case10	HW	Gauss Elimination SW	HW	0.7465
Case11	HW	Gauss Elimination HW	SW	0.2122
Case12	HW	Gauss Elimination HW	HW	0.5957
Case13	HW	Cramer's Rule SW	SW	0.4243
Case14	HW	Cramer's Rule SW	HW	0.7465
Case15	HW	Cramer's Rule HW	SW	0.1202
Case16	HW	Cramer's Rule HW	HW	0.5037

V. CONCLUSION

In this paper, a hardware-software co-design methodology was used to implement automatic white balance functions, which is based on FPGA chips and a SoPC architecture. The hardware design using an IP-based method with an embedded processor is able to reduce the memory and CPU resources of PC as well as to have the properties of easy modification and function expansion.

The chosen automatic white balance algorithm which combines and is based on the gray world assumption and the perfect reflector assumption theories to achieve the automatic white balance correction for acquired images. It causes that the color temperature of images is the same as human eyes. By the automatic white balance algorithm it is able to automatically adjust the internal image color. It is able to show the correct color as human eyes when the illumination of the light source vary.

We used the hardware-software co-design methodology to implement the automatic white balance algorithm on a SoPC. An objective function was used to evaluate sixteen hardware-software partition cases and to achieve a balance between execution performance and hardware resources. The performance of the partitioned result was 26 times faster than the pure software implementations. It is able to deal with a 2.5 M pixels picture per second. The amount of the additional hardware resources is 4 times of a FPU. The system architecture has the properties of easy modification and function expansion. It shows the advantage of the SoPC design. As FPGA process techniques is continuous improved, the system may have a better performance if energy consumptions can be further considered together.

REFERENCES

[1] E. Y. Lam, “Combining Gray World and Retinex Theory for Automatic White Balance in Digital Photography,” *Proceedings of the Ninth International Symposium on Consumer Electronics*, June 2005, pp. 134-139.

[2] W. Wolf, “A Decade of Hardware/Software Codesign,” *Computer*, vol. 36, Issue 4, April 2003, pp.38-43.

[3] F. Gasparini and R. Schettini, “Color correction for digital photographs,” *Proceedings of the 12th International Conference on Image Analysis and Processing*, 2003, pp. 646-651.

[4] Y. Zou, Z. Zhuang and H. Chen, “HW-SW Partitioning Based on Genetic Algorithm,” *Congress on Evolutionary Computation*, vol. 1, June 2004, pp. 628-633.

[5] S. Kawamura, “Capturing images with digital still cameras,” *IEEE Micro*, vol. 18, Issue 6, Nov.-Dec. 1998, pp.14-19.

[6] Chan-Pang Kuok, “The Design of a Dectection System of Copper Foil Defects on Printed Circuit Boards under System-on-chip Structure ,” Master Thesis, Department of Electrical Engineering, National Cheng Kung University, Taiwan, Jun. 2004.

[7] Xilinx, Inc., “Xilinx Logicore Floating-Point Operator v1.0,” April 2005.

[8] Xilinx, Inc., “MicroBlaze Processor Reference Guide,” Ver.5.2 May 9, 2005.

[9] An Enhanced 32-Bit Processor Core for FPGA Integration, http://ramp.eecs.berkeley.edu/Publications/MBforRAMP2.ppt

[10] SVS-VISTEK CAMERAS Inc., “Digital Progressive Area Scan Camera SVCam User`s Manual Monochrome / Color Version LVDS-Version / Camera Link Version with 10/12 Bit Digitization,” Ver. 1.6, May 2003.

[11] Automated Imaging Association, “Camera Link Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers,” Ver.1.1, Jan. 2004.

[12] T. Chen, “A Study of Spatial Color Interpolation Algorithms for Single-Detector Digital Cameras,” Information System Laboratory Department of Electrical Engineering Stanford University, http://www-ise.stanford.edu/~tingchen/.

[13] M. Fedor, “Approaches to color balancing,” PSYCH221/EE362course project, Department of Psychology, Stanford University, U.S.A., 1998.

[14] J. Chiang and F. Al-Turkait, “Color balancing experiments with the HP-photo smart-C30 digital camera,” PSYCH221/EE362 course project, Department of Psychology, Stanford University, U.S.A., 1999.