Real-time Cartoon Work

[1]LU Kai, [2]LIN Yin

[1,2]Department of Electronic Engineering, Hangzhou Dianzi University
Hangzhou, 310018, China

[1]lukai.neo@gmail.com, [2]linyin1112@163.com

***Abstract*— In this paper, a real-time cartoon work for video effect and entertainment based on FPGA is presented. Users are able to produce a cartoon effect video at once instead of using Photoshop to process it. The design approach of this system consists of four major parts. First, collect video information from the camera or DV and transfer to the RGB format. Second, use bold edges and color simplification to create cartoon-like video of real world, approaches here includes Gaussian filter, edge detection, timer/space average and color simplification. Third, the cartoon-like images could also be stored in the SD card controlled by SPI bus written in verilog HDL. Forth, the stored images and original video feed could be displayed in the monitor with a picture in picture effect. Our system implements the idea on Altera-DE2 board and provides a flexible device to produce cartoon style video or picture in real- time.**

***Keywords*— Cartoon Work, PIP, Edge Detection, Color Simplification, FPGA Design**

I. INTRODUCTION

Human demand is the original motivation for the development of science and technology. The strong curiosity of our team to create a cartoon video and the innovative spirit encourages us to accomplish the project, real-time cartoon work, which could create a real-time cartoon from a camera feed. The present work use bold edges and color simplification to create cartoon-like renditions of real world, and the inspiration came from the modern video games and TV cartoons. Cartoon work is the act of modifying an image to make it looks as if it were rendered by a computer or sketched by an artist. This includes bolding any edges or contours in the image and making colors brighter and more vibrant. In addition the amount of colors used in the image is reduced and smoothed out.

We use Altera DE2 FPGA board provided by Terasic to implement real time cartoon work. Video feed is sent to the decoder chip from a camera via a composite cable. Once decoded to RGB, the signal is sent to the cartoon work module and then passes through edge detection, time averaging and color simplification modules. The cartoon style pixel data is then handed off to the board's VGA driver and showed in a monitor. Besides, once key pressed, the cartoon style image could be stored in SD card and displayed in a picture in picture (PIP) mode, which the cartoon work image is on the background layer while the original video is on the overlay.

Users are able to select both the color simplification scheme for each color channel and also the edge detection threshold by using the DE2's switches. There are options to display in gray-scale and brighter color schemes selectable by the switches as well. Users could also choose the cartoon work display or the PIP display by using the DE2's keys.

II. SYSTEM ARCHITECTURE

In this section, we are going to introduce the design architecture of our system. The block diagram of system architecture is shown in Fig. 1.

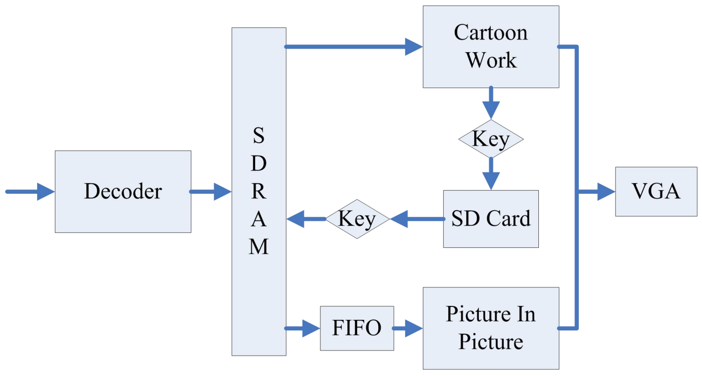

Fig. 1 System architecture

Camera collects the video signal and sends it to the decoder module. The decoder chip transforms the video signal to the CCIR656 data flow. The signal is transformed from CCIR656 to YCrCb and then from YCrCb to RGB by FPGA, it is written to the SDRAM next. The present work includes two modes: cartoon work and PIP. In the cartoon work mode, the RGB signal is sent to the cartoon work module and then passes through edge detection, time averaging and color simplification modules to make it just like cartoons. The cartoon style pixel data is then handed off to the board's VGA driver and showed in a monitor. The cartoon style image could be stored in SD card when key 1 is pressed. The controller of SD card using SPI interface is designed based on FPGA system and FIFO. In the PIP mode, the image in the SD card is also written to the SDRAM when the key2 is pressed. The stored image and the original video are read from SDRAM at the same time to meet the timing of VGA driver and then sent to the PIP module via FIFO. PIP module sets the cartoon work image as the backgrounds while the original video as a floating window on the overlay. The mix pixel data is then handed off to the VGA and showed in a monitor.

The principle of the proposed work includes the following three parts: Decoder, Cartoon work, PIP.

A. Decoder

The following block diagram and the description of DE2_TV design as shown in Fig. 2 are taken from Altera's DE2 User Manual.

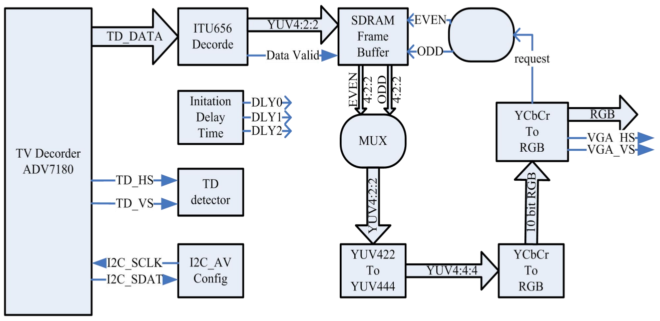

Fig. 2 DE2_TV design

Fig. 2 shows the block diagram of the design. There are two major blocks in the circuit, called I2C_AV_Config and TV to VGA. The TV to VGA block consists of the ITU-R 656 Decoder, SDRAM Frame Buffer, YUV422 to YUV444, YCrCb to RGB, and VGA Controller. This figure also shows the TV Decoder (ADV7181) and the VGA DAC (ADV7123) chips used.

As soon as the bit stream is downloaded into the FPGA, the register values of the TV Decoder chip are used to configure the TV decoder via the I2C_ AV_Config block, which uses the I2C protocol to communicate with the TV Decoder chip. Following the power-on sequence, the TV Decoder chip will be unstable for a time period; the Lock Detector is responsible for detecting this state.

The ITU-R 656 Decoder block extracts YCrCb 4:2:2 (YUV 4:2:2) video signals from the ITU-R 656 data stream sent from the TV Decoder. It also generates a valid control signal indicating the valid period of data output. Because the video signal from the TV Decoder is interlaced, we need to perform de-interlacing on the data source. We use the SDRAM Frame Buffer and a field selection multiplexer (MUX) which is controlled by the VGA controller to perform the de-interlacing operation. Internally, the VGA Controller generates data request and odd/even selected signals to the SDRAM Frame Buffer and filed selection multiplexer (MUX). The YUV422 to YUV444 block converts the selected YCrCb 4:2:2 (YUV 4:2:2) video data to the YCrCb 4:4:4 (YUV 4:4:4) video data format.

Finally, the YCrCb_to_RGB block converts the YCrCb data into RGB output.

B. Cartoon Work

In the cartoon work module, the video signal is then split off to two different pipeline paths: one for edge detection and one for color time and spatial averaging and color simplification. Along the edge detection path, the streaming pixel data is passed through a Gaussian filter to attenuate image noise and enable cleaner edge detection.Then the pixel data is fed into a large set of shift registers that holds pixel data for three entire lines worth of pixels. This is required because our edge detection algorithm requires not only the pixel in question, but the eight neighboring pixels as well. A 3x3 block of pixels centered on the pixel in question is chosen from the shift registers and sent to an intensity calculation module. The intensity calculation module calculates the intensity of each of the 9 pixels in the block and feeds the intensity values to both horizontal and vertical edge detection modules. These modules apply a Sobel filter algorithm to the appropriate pixels to detect an edge. If there is an edge, a select signal is sent to a final MUX to select black as the pixel color.

In the color manipulation pipeline, the pixel data is first sent to an identical set of shift registers as in the other pipeline just to keep the two paths synchronized. The output of this module is sent to a time and spatial averaging module. This module interacts with SRAM and saves a history of past values for every pixel in the raster to time average out noise to reduce flickering. It also does 2x2 block spatial averaging to further smooth out the image. After some additional pipeline registers to match timing with the other path, the colors are simplified based on user input from the switches, which change the number of colors that each channel can represent.

Finally the two paths converge at the input to the final MUXing that determines the value of the pixel in question. If there is an edge, black is chosen, otherwise color is chosen. There is also the option to choose a gray-scale image, which is simply the intensity value for the pixel that was calculated earlier in the edge detection path. The output of the MUX is sent to the VGA controller provided to us by Terasic. The controller module drives the VGA output port on the DE2 board and is hooked up to an external monitor to display the raster, which is updated in real time.

The block diagram of cartoon work is shown in Fig. 3.

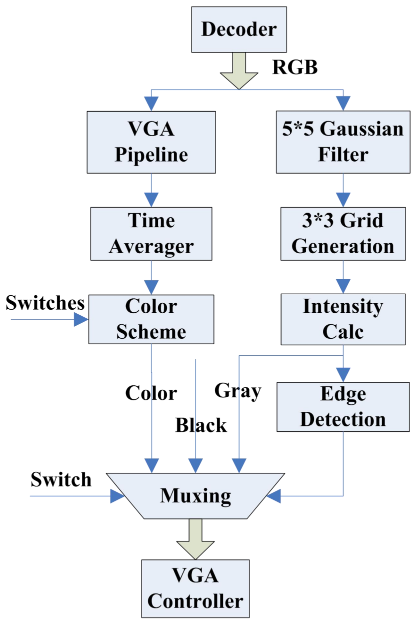

Fig. 3 Cartoon work architecture

C. PIP

To implement the PIP effect, we set the cartoon style image store in the SD card as the background. Additionally we lessen the original NTSC video to 320*240 and set it as a moving window on the overlay. Finally these two pictures are shown in the monitor according to the VGA controller timing. The block diagram of PIP module is shown in Fig. 4.

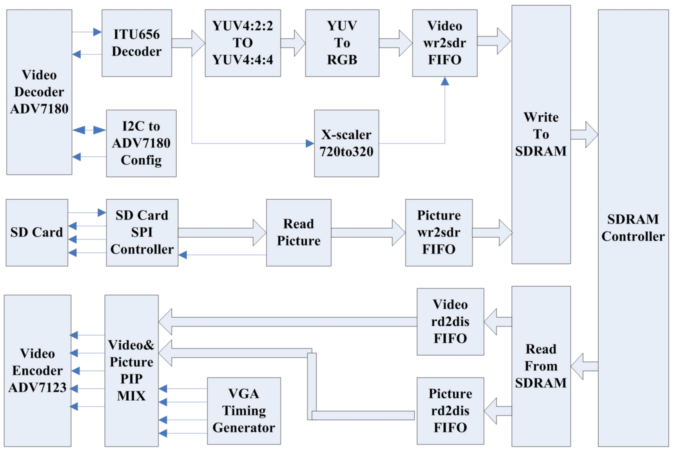

Fig. 4 PIP

In the write to SDRAM part, we create a write-to-SDRAM arbitration before the SDRAM to get together the original video data and the image data read from the SD card. We lessen the streaming video at the X-scaler before writing it to SDRAM to make the moving video window shown as 320*240. The present work extracts 320 pixel from 720 pixel equably because it is the most economical and straightforward way. That is to say 4 pixels are extracted from every 9 pixels in the original data just as shown in Fig. 5.

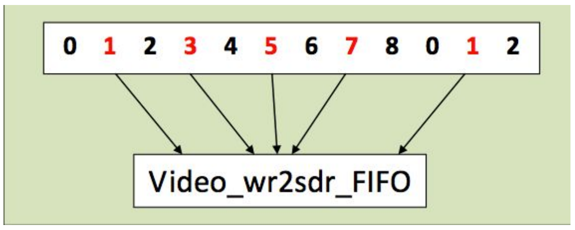

Fig. 5 Extraction algorithm

When it comes to display, for we need to read two different data to show in a nesting way, we create two FIFO for the streaming data to buffer the data read from SDRAM respectively between read-from-SDRAM module and the VGA controller module. That is, the PIP effect can be regarded that big image-display window and small video-display window are two monitors in different size and they are synchronous when showed, however the small monitor's blank time is a litter longer.

Finally, we make the small window moving in the big image-display window area all the time to implement the whole PIP effect. VS signal counts once every frame so that the small window shifts a pixel both in the X direction and Y direction and reverses until it reaches the setting edge. Thus, the small video-display window is just like a marbles in a box, seeking for the exit forever but cannot go out.

D. SD Controller

The SD card module includes READ_FIFO, SD_WRITE, SPI_CONTROLLER, CLK_SD, SD_INITIAL, WRITE_FIFO, SD_READ. These modules are written in hardware language, verilog HDL. The block diagram of SD controller module is shown in Fig. 6.

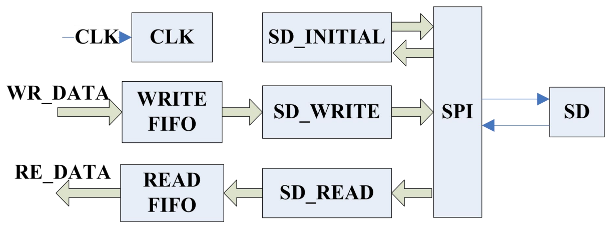

Fig. 6 SD controller

1) SPI_CONTROLLER

SD card supports three kinds of transmission modes: 1-bit SD BUS, 4-bits SD BUS and SPI BUS. The data transfer rate of 1-bit SD BUS and 4-bits SD BUS is high, but it's difficult to achieve it in hardware design language because its transmission protocol is complex. While SPI bus mode has only a data transmission lines, it is easy to implement and greatly shortens the system development since transmission protocol is simple. So we use SPI BUS mode.

SPI interface includes SCK,MISO,MOSI and CS. When CS signal is active, the parallel data of the host input is stored in the shift register and converted into serial data stored on the SD card. Meanwhile, serial data of SD card is read and converted into parallel data by the shift register.

2) *CLK_SD*

The data transfer clock of SD card is 0~50MHZ,the data transfer rate is 0~25MB/s, and initialization clock of SD card is 0 ~ 400kHz. While the system clock is 50MHZ. Thus, in order to In order to ensure the data transfer and initialization of SD card, we need to the need to divide system clock in the CLK_SD module, making the initialization clock does not exceed 400kHz, data transfer clock does not exceed 50MHz.

3) *SD_INITIAL*

We should initialize the SD card prior to operations of read and write. After the SD card is powered, it is supposed to delay at least 74 clocks to ensure stability, but the delay time can not exceed 1ms.After 74 clocks delay, the master controller send a reset command (CMDO) to the SD card in order to come into the SPI BUS mode when CS signal is active (low). Reset SD card when receiving a response(0x01h) to the SD card. Then we send an initialization command (CMD1), master controller pulls up the CS signal after receiving the response of the SD card(0x00h), now the initialization of SD card is complete.

4) *SD_WRITE, SD_READ*

Read/write data controller has two parts, command_controller and data_controller. The block diagram of read/write data controller is shown in Fig. 7.

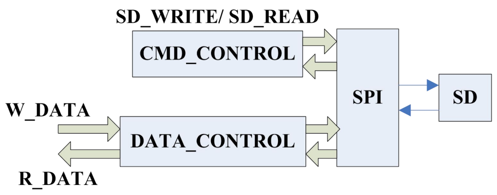

Fig. 7 Read/ write data controller

After the initialization, the command_controller sends a write command or a read command to the SD card when CS signal is active (low). Once the SD card respond 0x00h, status jumps to data_controller.

The write_data_controller sends a data's start bit(0xFE) after receiving the correct response of SD card. Then it's time to send your data. The default length of data here is 512 byte. After this, the response of SD card would be received so that status goes into the DATA_ACK to complete the process of write data. The block diagram of write_data_controller is shown in Fig. 8.

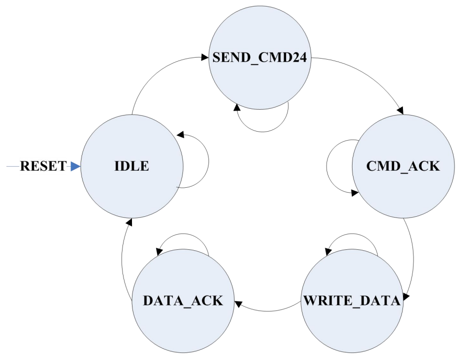

Fig. 8 Write_data_controller

The read_data_controller sends a data's start bit(0xFE) after receiving the correct response of SD card. Then it's time to read your data in SD card's sectors. The default length of data here is 512 byte. After reading the whole 512 byte data, we complete the process of read data. The block diagram of read_data_controller is shown in Fig. 9.

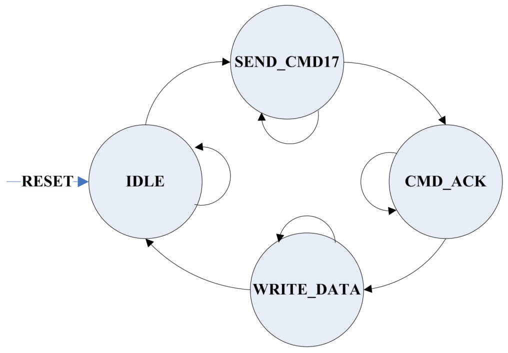

Fig. 9 Read_data_controller

III. Design Method of Cartoon Work

In this section, we are going to introduce the design method used in cartoon work module.

A. Gaussian Filter

The system uses a 5x5 Gaussian kernel with equal elements to simulate a Gaussian radius of infinity. The filter thus effectively acts as an averaging function.

0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04

Fig. 10 5 x 5 Gaussian Kernel

Each pixel in the map is influenced by 24 of its neighbors. Spatial averaging helps get rid of several unwanted video artifacts including specular reflection. The filter is implemented by convolving the the kernel with the camera output.

The Gaussian filter module takes a 5x5 window of pixels and performs the aforementioned computation using adders and a fixed point multiplication to multiply the result by the correct fraction.

The filter needs an array of wires. However, since Verilog modules cannot have arrays of ports, the array of wires is packed into one wide bus which is the input port. In the case of the Gaussian Filter, the port is 250 bits wide. Within the code, the ports are unpacked into a wire array which is then operated upon.

B. Grid Generation

There are two grid generator modules that store arrays of pixels read from the video input. The array of pixels uses line buffers to store horizontal lines of pixels until they are overwritten by new pixels. There are two grid generators because one is for a 3x3 window of pixels and the other is for a 5x5 window of pixels. The line buffers are 640 elements long to accommodate a single horizontal line of the VGA frame.

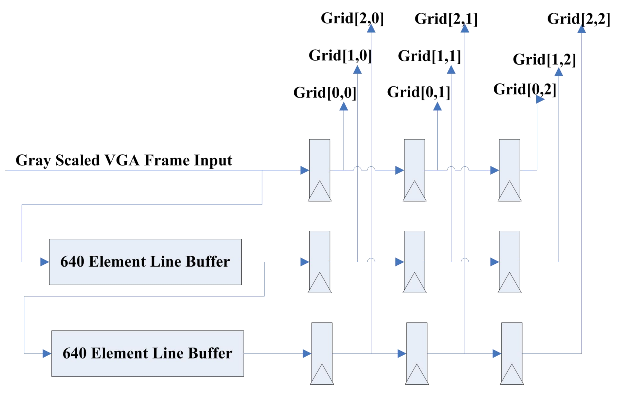

Fig. 11 Data path diagram of a 3x3 Grid Generator

The Gaussian Filter operates on a 5x5 array of pixels so it connects to the 5x5 grid generator, storing the output Gaussian value to each of the pixels. The 3x3 grid of pixels is connected to the Sobel Detection module because the calculation performed there only requires a 3x3 pixel window. Similarly to the Gaussian filter, port packing and unpacking is done to simplify the code and reduce the number of port names.

C. Intensity Calculator

The intensity calculator module is a simple hardware implementation of a weighted averaging between the different color channels of the pixel. The red, green, and blue values of the pixel in question are given to the intensity calculator during each cycle while the VGA is not blanking. We chose to weight the intensity of a pixel as 25% red value, 50% green, and 25% blue. We chose green to be greater than red and blue because the human eye is most sensitive to this color. The coefficients are chosen as powers of 2 so we can use shift and add operations instead of costly multiplications. We let the compiler optimize the addition and shift operations as necessary and register the result each cycle, which is then outputted to the edge detectors. We instantiated 9 intensity calculator modules in our final design to account for the block of 9 pixels that the edge detection algorithm requires.

D. Edge Detectors

The Sobel operator performs discrete differentiation to get the approximate vertical and horizontal gradients of the image intensity. The Sobel filter is implemented by convolving a 3x3 kernel with the image twice, once for each dimension.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} A_{i-1,i-1} & A_{i,i-1} & A_{i+1,i-1} \\ A_{i-1,i} & A_{i,i} & A_{i+1,i} \\ A_{i-1,i+1} & A_{i,i+1} & A_{i+1,i+1} \end{bmatrix}$$

$$G_y = \begin{bmatrix} +1 & +2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \begin{bmatrix} A_{i-1,i-1} & A_{i,i-1} & A_{i+1,i-1} \\ A_{i-1,i} & A_{i,i} & A_{i+1,i} \\ A_{i-1,i+1} & A_{i,i+1} & A_{i+1,i+1} \end{bmatrix}$$

$$G = \sqrt{G_x{}^2 + G_y{}^2}$$

Fig. 12 Sobel convolution

The Sobel operator is inexpensive in hardware computationally. Iimplementation of the sobel filter requires multiple line buffers that will add delay to the system, like the Gaussian filter. However the edge-detection accelerator does not violate the real-time operation of the system. The trade-off for the computational simplicity of the Sobel operator is its inability to handle high frequency variations in image intensity.

There are two distinct modules for edge detection, one that does the vertical edges, and one that does the horizontal edges. The only difference between the two is which convolution filter it uses (described in above), the rest of the execution is exactly the same. The input grid of 9 pixels comes from the intensity calculator. The module then convolves this grid of pixels with the filter, and does the necessary additions and subtractions. This sum is then checked against a threshold. We calculate two versions of the sum, since there is a possibility it will be negative. We calculate the original sum and the negative of it, which guarantees that one of the two will be positive. Then we check which one is positive, and check that value against the threshold. This avoids some possibly disastrous complications, which include difficulty with sign on comparisons, or the need for two thresholds, 1 positive and 1 negative.

The threshold for this module is variable, and can be set using the switches and a push button to register the new threshold value. We originally had the threshold variable at 16 bits, but this was causing timing errors that resulted in smearing on the screen. We reduced the threshold to only be a 10 bit number, which removed the timing issues. The module outputs 1 if the pixel is an edge, and 0 if the pixel is not an edge.

E. **Time Averager**

We decided to average the pixel data both in time and space to reduce the noticeable flicker caused by color simplification. The module uses the on-board SRAM to store a history of past values of blocks of pixels to perform a weighted average on along with new incoming pixel data. The module is split into two components: an averager module that performs a combinational weighted average and the hardware associated with SRAM communication.

The averager module takes in the color information of the current pixel, the history for that pixel, and outputs the new value for the pixel to be saved in SRAM. We designed the averager to keep a running average of the last 4 frames worth of pixel data for each pixel in the raster. This means we want to add ¾ of the old value to ¼ of the new value to get the updated average. We chose to average 4 frames because it is a power of 2 and can be calculated using shift and add operations, thus avoiding costly multiplications. Instead of trying to calculate ¾ of the old value directly, we use (1 - ¼) times the old value. The final equation for the new average is:

newAvg = oldAvg - oldAvg>>2 + newColor>>2.

The averager module assumes that each color channel will have 5 bits of precision. This is determined by the word size of the SRAM, which is 16 bits. We have three 5 bit colors, which is 15 bits, and 1 unused bit that is stored in the SRAM. All three colors are averaged at the same time within the averager, and the value is outputted without registering.

The time averager module is responsible for communicating with the SRAM and feeding its data to and from the averager module. To implement a 2x2 spatial averager, first we needed to change the addressing of the pixels by removing the least significant bit of the X and Y coordinates. This makes 4 adjacent pixels all reference the same location in SRAM. This reduces the total required SRAM and allows us to use a full word for pixel data instead of sharing 2 pixels per 16 bit word of SRAM as we have done in the past.

The SRAM takes a cycle to read and another to write, and it is a problem if a new pixel is required each cycle. Luckily we are averaging 2 pixels at a time and adjacent horizontal pixels are requested one after another. This means we only need the history once for both the pixels together instead of both individually, which gives us 2 cycles per 2 pixel horizontal pair. We take advantage of this by reading the history for the two pixels in the first cycle and write the updated history for the two pixels in the second cycle. The history gets updated with both new pixel values for that 2 pixel block by using an intermediate history value that is stored in a register.

The current address requested is always sent to the SRAM address port with the LSB of X and Y coordinates removed. This makes the address the same during the first and second cycle of a pixel pair. The write enable for the SRAM is only set high in the second cycle, which is determined by the LSB of the X coordinate. There is a single data port to the SRAM that must be floating during the first cycle for reading, and set to the updated history value from the averager during the second cycle. The write enable to the SRAM can only be set high when the VGA controller is not “Blanking” because we want to update the pixel history with valid data being requested by the VGA controller.

The input to the averager module’s “old” port during the first cycle is the history data read from the SRAM. The input to the current colors is always the current color channels coming from the SDRAM requested by the VGA controller. The output of the averager is registered after the first cycle and used as the “old” input to the averager in the second cycle. The output during the second cycle is sent to the SRAM. This system uses both current adjacent pixel colors to update the average, while only needing to update the SRAM once. The output of the averager is also set as the output of the timeAverager module for each cycle. This is in a 5-5-5 RGB format.

IV. RESULT

In this section, we will represent the final effect of different modes.

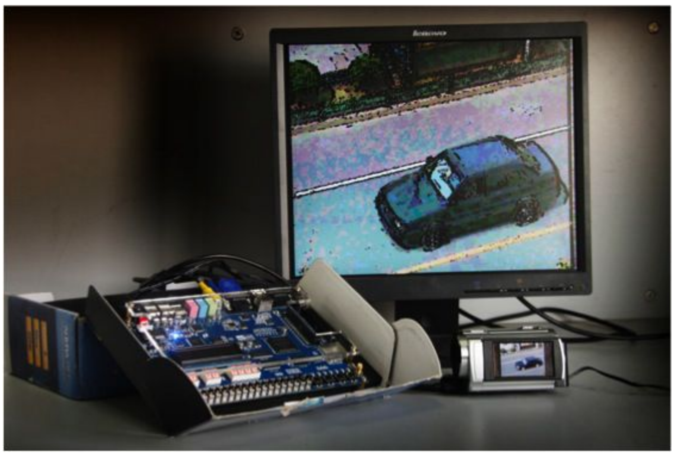

Fig. 13 Overall projects

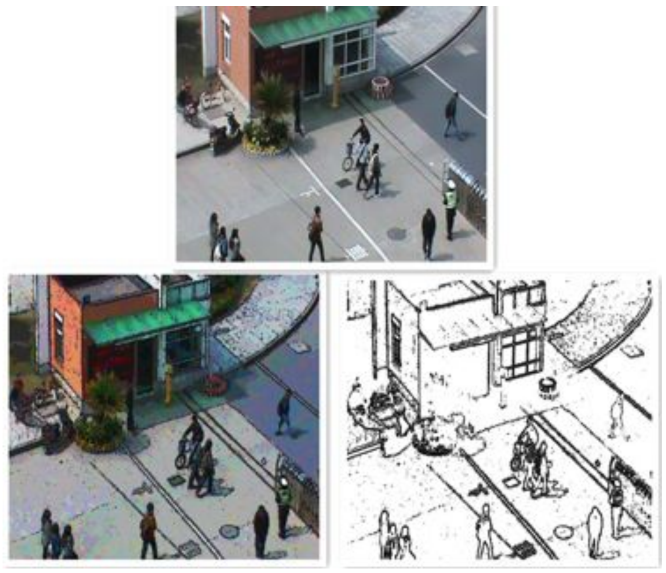

Fig. 14 Original image&cartoon work image&edge detected image

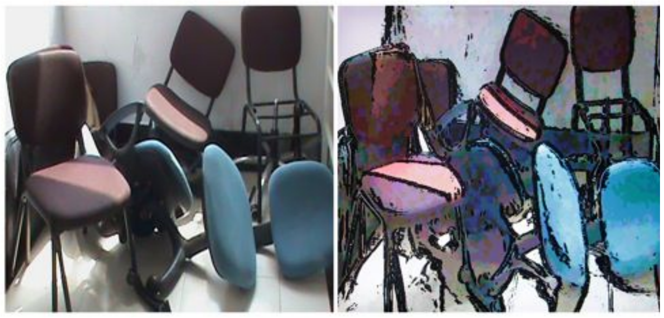

Fig. 15 Original image&cartoon work image

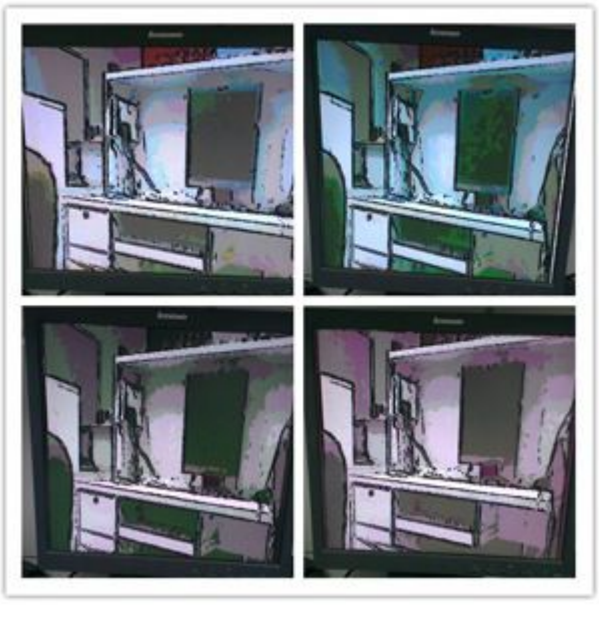

Fig. 16 Gray-scale&3 brighter color schemes selected by switches

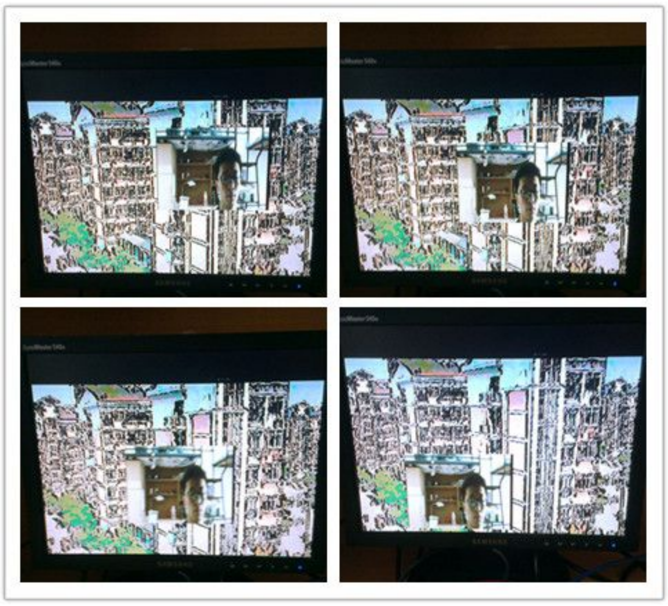

Fig. 17 PIP

V. CONCLUSIONS

Overall we are very satisfied with the results of our project. We implement a real-time cartoon work that uses Gaussian filter, edge detection algorithm, time/space averaging, and selectable color simplification. The images we produce are convincing and in many cases look like they were sketched by an artist. In addition, we store the cartoon image into the SD card and display the image and the video in picture in picture effect. We are able to accomplish all of this when keeping the hardware fast enough to keep up with the constantly updating stream of video data. This is a very interesting project that challenges us in video processing, real-time calculation, and embedded system design.

ACKNOWLEDGMENT

We would like to express our gratitude to Altera® Inc. and Terasic® Inc. for their sponsor and support for our project in the Innovate Asia Altera Design Contest 2013. Moreover, we also wish to acknowledge Professor Jiye Huang and Department of Electrical Engineering, Hangzhou Dianzi University for the excellent training and providing us with the chance to fulfill this project.

REFERENCES

[1] W.H. Tsang, P.W.M. Tsang, “Suppression of false edge detection due to specular reflection in color images”, Pattern Recognition Letters, Volume 18, Issue 2, February 1997.

[2] Alex Stark J.Adaptive image contrast enhancement using generalizations of histogram equalization. IEEE Transactions on Image Processing . 2000.

[3] L Tao,Vijayan Asari.An integrated neighborhood dependent ap-proach for nonlinear enhancement of color images. Information Technology:Coding and Computing,2004.Proceedings . 2004.

[4] ALTERA.Edge detection reference application Note 364.ver.1.0. . 2004.

[5] Yu Wang, Rong Luo, et al. Development of an effective design tool of a real-time image processing hardware, final report for cooperation with Mitsubishi Heavy Industries, Ltd, 2012-3-19.
[6] Gonzalez, Digital Image Processing [M]. Beijing: Electronic Industry Press, 2005.9:179.
[7] DE2 User_Manual, www.altera.com, 2009
[8] Altera Co.Cyclone FPGA Data Sheet. . 2005.
[9] http://www.fpga4fun.com
[10] http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2010/kaf42_jay29_teg25/teg25_jay29_kaf42/index.html
[11] http://en.wikipedia.org/wiki/Sobel_operator
[12] http://en.wikipedia.org/wiki/Gaussian_filter