Embedded Demonstrator for Video Presentation and Manipulation

By Cato Marwell Jonassen
NTNU, Trondheim, Norway, 2010

Abstract: This embedded video demonstrator is built to show the importance of both hardware and software in a computer system. To show this the demonstrator presents video processed in either hardware or software based on selection. In addition it also shows the complexity of calculating decimal numbers (floating-point) when manipulating colors. The user should be able to conclude that the introduction of dedicated hardware to do complex and repeated tasks, can greatly increase performance.

Purpose

The purpose of the embedded video demonstrator is to give a visual demonstration of the importance of hardware/software codesign. Without the combination of hardware and software many familiar applications would not be possible. PCs, mobile phones, cameras etc. are examples of such applications.

Theory

Video

To show video this system uses a camera called D5M from Terasic. This camera is configured to read sensor data continuously. By reading the data fast enough it is possible to display multiple frames per second. The sensor consists of 2592×1944 pixels, and each pixel has an image sensor to record light at its location. It is not possible for the sensor to register all visible light, so a color filter is put on top of all the pixel sensors. This filter makes each sensor register only the light intensity of one color, either red, green or blue. The color filter used in this camera is called a Bayer filter. The Bayer filter consists of twice as many green filters than red and blue. This is done to compensate for the fact that the human eye is more sensitive to green light.

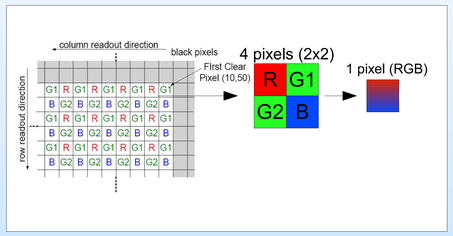

FIGURE 1: Color interpolation by using neighboring pixels to determine one full range RGB pixel

As Figure 1 shows, the data that is captured from the sensor only has one color per pixel. In order for a display to show a color image it needs three color data (red, green and blue) per pixel. To achieve this a demosaic algorithm is used to interpolate the missing color values for each pixel. This can be interpreted as "qualified" guessing. One of the simplest ways to do this is by using the closest neighbor colors to calculate the missing colors. Figure 1 shows how this is done by grouping pixels in 2×2. The interpolation can be done either in hardware or in software. Since an image consists of a lot of pixels arriving at a high rate it would be smart to do the interpolation in its most effective environment. The higher the resolution, the more pixels have to be processed per picture-frame.

Color Space

There are many different ways to represent colors, RGB is one of the most common. In this color space a color is created by an adaptive combination of red, green and blue. There are a lot of displays constructed to use this color space. RGB makes it very easy to create new colors, but the RGB color space does not have any easy way to alter brightness or contrast of the image.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0.000 & 1.596 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} (Y - 16) \\ (Cb - 128) \\ (Cr - 128) \end{bmatrix} \quad (2)$$

YCbCr is a color space that can be derived linearly from RGB. Y is the color's luma component (can be used to see a grayscale version of the image), Cb and Cr are the blue-difference and red-difference chroma components. Adjusting the brightness of the color is done by adding or subtracting a constant to the luma component. Equation 1 shows how to transform between the RGB and YCbCr color domain, while Equation 2 shows how to transform back from YCbCr to RGB color domain. This transformation can be done by a matrix multiplication of floating-point constants. Floating-point or decimal values require more computation than "simple" integer multiplication. To invert colors the RGB values are binary inverted, either by using a dedicated inverter or by taking the maximum value (based on the bit width, 8 bits gives the maximum value 255) and subtract the current color. These two methods are functionally equivalent, but subtracting is more time-consuming.

System overview

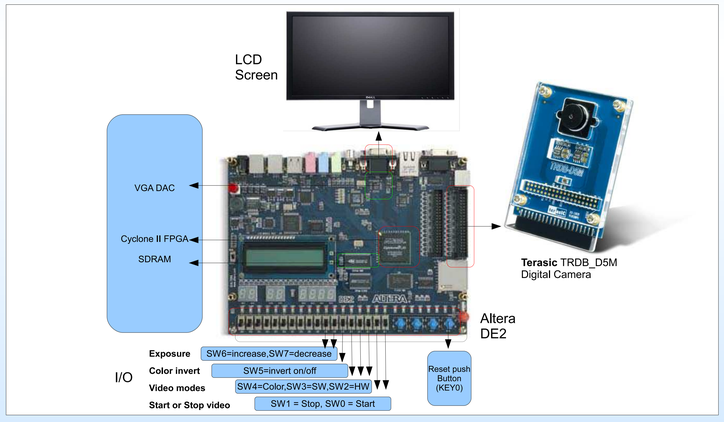

FIGURE 2: An overview of the demonstrator and its I/O

Figure 2 shows an overview of the whole system including a description of what the different switches do.

Switch	Purpose
KEY[0]	Reset the whole system
SW0	Start capturing video data from the camera
SW1	Stop capturing video data from the camera
SW2	Select hardware video processing (default)
SW3	Select software video processing
SW4	Select color demonstration mode
SW5	Invert colors on/off by toggling
SW6	Increase exposure
SW7	Decrease exposure

TABLE 1: Switches and their purpose

Table 1 is a list of the switches and their purpose in the system.

Demonstration

This list presents a possible run-through of the demonstration.

1. Start the demonstration by first choosing "hardware" video processing. This is done by toggling SW2
2. Start the video by toggling SW0, or stop the video by toggling SW1. SW5 is used to activate/deactivate inverted colors. If the picture is too bright or too dark it is possible to adjust the exposure by toggling SW6 or SW7
3. To start processing video in software, toggle SW3. This will reset the demonstrator and make it ready to begin processing in software
4. Whenever you are ready, start the processing by toggling SW0 and stop it by toggling SW1. SW5 is used to activate/deactivate inverted colors
5. Observe, what was the difference?
6. To start the color demonstration mode, toggle SW4. The system will stop displaying video and show a color palette image
7. Toggle SW4 further to go through the different stages. The LCD panel will display what the demonstrator is doing and the result can be observed on the monitor
8. Observe, what was the difference between processing the colors in hardware and using the software processor?

Conclusion

Video processing in hardware

When processing the video in hardware the video is very viewable and it has a high frame rate (around 15 fps). This is a result of using dedicated hardware to do a static task which repeats itself as long as there is video to process. Switching between normal color representation and inverted color representation happens instantly when activating this function. The hardware inverts the color by inverting the binary value of the color signal.

Video processing in software

When the video is processed in software the video is not very viewable and the frame rate is less than one frame per second. This is because the processor has to do all the job processing the video. The video data arrives quickly and in order for the processor to be able to keep up, the frame rate has to be lowered significantly. Switching between normal color representation and inverted color representation also works fine in software. The inverting is done by taking 255 and subtracting the color value for each component. This is more complex than inverting in hardware, but it manages this because of the low frame rate.

Color demonstration

Transformation between the color spaces by using floating-point constants demands more processing time than multiplication of integers. To be able to perform floating-point calculations in software, the values have to be translated to work in an integer multiplier. This translation takes time and makes multiplication more complex. By using a dedicated hardware floating-point multiplier this process is much faster than in software, because the floating-point or decimal values do not need to be translated.

References

[1] - Jonassen, Cato Marwell, Master Thesis, Department of Electronics and Telecommunication, Trondheim: NTNU, 2010, Unpublished