Second Prize

Nios II Embedded Electronic Photo Album

Institution: **Electrical Engineering Institute, St. John's University**

Participants: **Hong-Zhi Zhang, Wei-Ming Yeh, and Wei-Min Yang**

Instructor: **Rui-Xi Chen**

Design Introduction

Digital photos are very popular on the Internet; unfortunately, unauthorized distribution and use of these photos is very common. To prevent unauthorized use, an author can place a watermark on the image, but this technology has not been implemented in an embedded system. Without an instant watermark and encryption option on digital cameras and e-albums, the user must manipulate online photos with a computer or leave them unmarked.

To protect the author's copyright effectively, we used hardware/software co-design to create a Nios® II embedded photo album that provides instant raw data preset processing. Typically, digital photos are saved in compressed JPEG format. To encrypt a digital watermark, you must recover the photo from the archive in which it was saved. However, if you encrypt the photo before it is compressed, you do not have to perform as many operations. We implemented the watermark encryption technology using the µClinux v2.6.11 embedded system and a hardware drive.

Our design can be incorporated into digital cameras and camera phones that can save photos or deliver them to the Internet via a wireless network instead of a computer. If the author's watermark is added to the photo before it is stored into an archive, the photo can then be uploaded to the Internet directly without requiring additional processing software.

Function Description

Our watermark encryption technology and the resulting marked photos must be transparent, robust, secure, and unambiguous. Additionally, the embedded system watermark (ESWM) photos can display

functions of the picture. We used the Nios II processor and hardware/software co-design concepts to perform the following functions:

- Use a discrete cosine transform (DCT) operation to separate the major images and the elements that are not sensitive to the human eye.
- Determine which frequency domain to use when adding the watermark so that the photo is not damaged.
- Use random number seeds to scatter the watermark.
- Retrieve the scattered image points.
- Store the program and photo onto a compact flash (CF) card.
- Migrate the µClinux 2.6.11 kernel, establish a file system, create a Boa network server and application program.
- Use a built-in network server to display the watermarked photo on a web site.
- Directly embed the watermark.

Performance Parameters

The time it takes to compute an algorithm determines whether hardware is more efficient for the operation. We used profiling software to detect the time spent on all of the the ESWM program's operations. Additionally, the profile software assessed the algorithm's efficiency for each sector.

We used a standard test image of "Lena" (a 24-bit, 128 x 128 color image) as the photo to watermark. See Figure 1.

Figure 1. Standard Testing Image of Lena

We use a grayscale 32 x 32 image for the watermark (see Figure 2).

Figure 2. Watermark

When we execute the ESWM program, we receive information that can be used to apply or remove the watermark. Figures 3 and 4 show the efficiency analysis for applying and removing the watermark. The colored pie slices represent the amount of time spent on a specific operation.

Figure 3. Efficiency Analysis of the Watermark Application Program

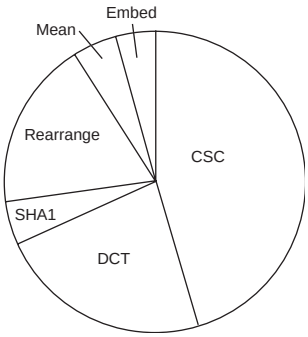
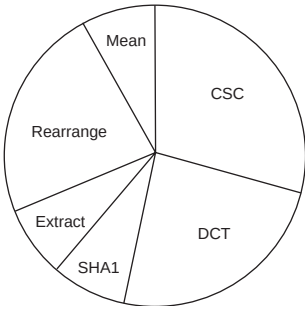


Figure 4. Efficiency Analysis of the Watermark Removal Program



When we analyzed the efficiency of the watermark application and removal process, the color space converter (CSC), discrete cosine transform (DCT), and rearrange (embedding and quantization) operations occur most frequently and use the most resources. Seen from the program code of the photo, the CSC and DCT functions read pixel information to operate; therefore, the number of operations performed vary with the size of the photo. That is, the bigger the photo, the more time is required for the computation. The embedding and quantization process uses a random number generator. If the generator repeats numbers, it will cause collisions when the application rearranges the photo. Most collisions are resolved by re-selecting the random numbers. A larger photo leads to a greater possibility of collision, which is the source of the bottleneck.

When we created our Nios II system, we used our watermark application and removal program, as well as the operational analysis. When the system applies the watermark, it first treats the photo with a picture process (PICP) program that includes CSC and a forward discrete cosine transform (FDCT). Then, we apply the watermark process (WMP) operations (CSC, FDCT and quantization (Q)). Finally, we integrate the frequency domain and recover the three areas of the operation program in the time domain. Table 1 shows the watermark application process for the 128 x 128 Lena photo and the 32 x 32 watermark implemented on a Cyclone® II EP1C20 50-MHz device.

Table 1. Watermark Application Process

Process	CSC (Types)	DCT (Types)	Q (Types)	Embed (Types)	PICP *(1)*	WMP *(2)*	Integration Process *(3)*	Total Time
FLOAT_CESWM	Float	Float	Float	Short	24.1	3.23	10.47	37.77
FLOAT_CI _CESWM	Float custom instruc-tion (CI)	Float CI	Float CI	Short	4.34	0.72	2.82	7.88

Notes to Table 1:
(1) PICP: CSC (RGB to YCbCr) + FDCT
(2) WMP: CSC (RGB to YCbCr) + FDCT + Q
(3) Integration Process: Embedded + inverse discrete cosine transform (IDCT) + CSC (YCbCr to RGB)

To remove the watermark, the program executes the decode watermark process (DeWMP), performs dequantization, returns from the frequency to the time domain, and converts from the YCrCb to RGB color space. This process provides a deintegration operation, which occurs in two areas to remove the watermark.

Table 2. Watermark Removal Process

Process	CSC (Types)	DCT (Types)	Q (Types)	Extract (Types)	Deintegration *(1)*	DeWMP *(2)*	Total Time
FLOAT_CESWM	float	float	Float	Short	21.91	3.17	25.08
FLOAT_CI _CESWM	float CI	float CI	float CI	Short	4.11	0.41	4.52

Notes to Table 2:
(1) Deintegration Process: Dequantization + inverse DCT (IDCT) + CSC (YCbCr to RGB)
(2) DeWMP: CSC (RGB to YCbCr) + FDCT + Extracted

Design Architecture

Figure 5 shows the system architecture.

Figure 5. System Architecture

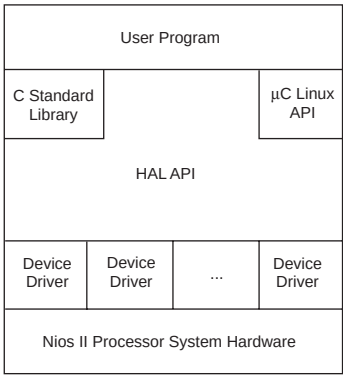
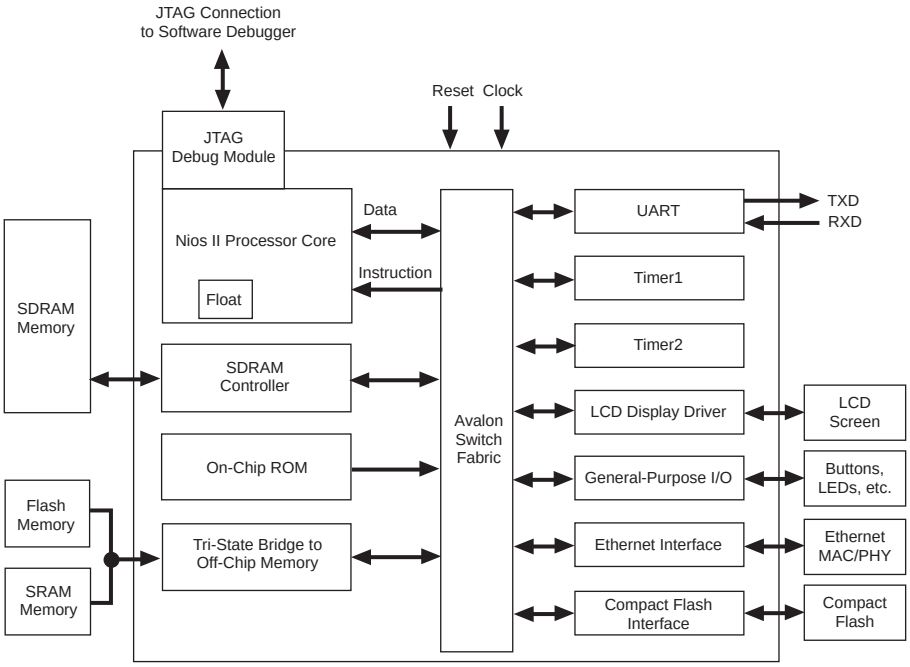


Figure 6 shows the system block diagram.

Figure 6. System Block Diagram



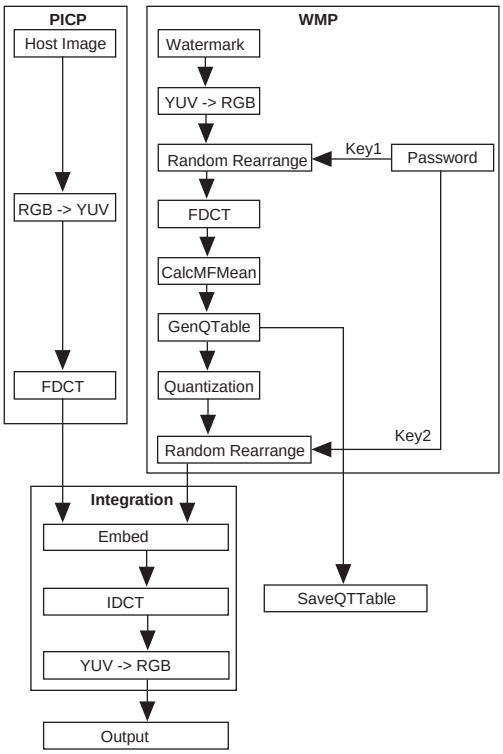
Design Methodology

This section describes our design methodology, including the method we used to apply the watermark and the steps we used to implement the system. To apply the watermark, we used a DCT to convert the image to the frequency domain, examined the position of the frequency band, and added random number functions to scatter the watermark throughout the photo, making it robust. To implement the system, we used a watermark algorithm as the program code. We tested each stage and did not add new functions until the established functions were stable. According to our efficiency analysis, we needed to hardware accelerate the CSC and DCT operations.

Implementation Method

Figure 7 shows the architecture for applying the watermark.

Figure 7. Watermark Application Architecture



We used the following method to apply the watermark.

1. Read the original image and perform a CSC operation, converting the original image from RGB to YCbCr.

2. Starting with YCrCb in the original image, perform an FDCT conversion and work out the DCT coefficient of the image. Look in the coefficients to find a medium or low frequency parameter that will be the starting position to apply the watermark.

3. Read the user's passwords, use an encryption algorithm to create two groups of random number seeds, read in the watermark (converting it from RGB to YCrCb), and use the first group of random number seeds for embedding and Q to scatter the watermark.

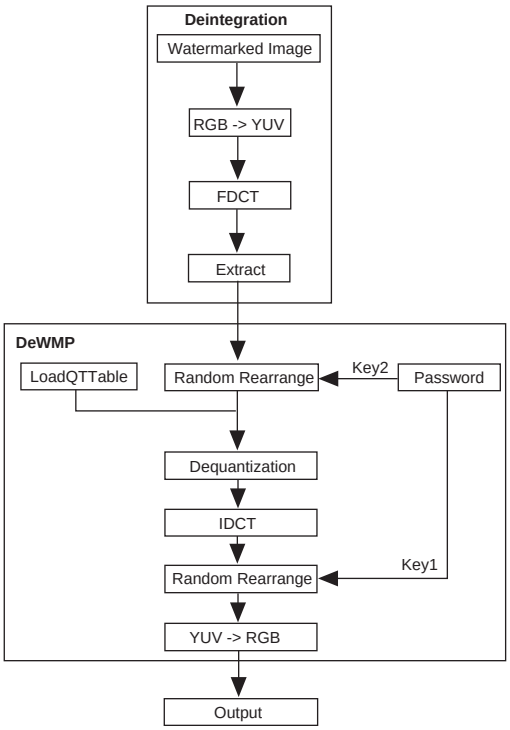
4. Perform an FDCT conversion of the scattered watermark to obtain the watermark's DCT coefficient. Take the average of the medium frequency, and let the quantification form make a digital quantization of all the coefficients, thus obtaining the quantified watermark DCT coefficient.

5. Use the second group of random number seeds on the quantified watermark DCT coefficient to replace, in random order, the low frequency parameter value of the original image.

6. Perform an IDCT conversion of the modified image to get the YUV image embedded with watermark information. Perform a CSC operation, converting from YCbCr to RGB, thus obtaining the image embedded with the watermark.

Figure 8 shows the architecture for removing the watermark.

Figure 8. Watermark Removal Architecture



We used the following method to remove the watermark.

1. Read the watermarked image and perform a CSC operation, converting from RGB to YCbCr.

2. Perform an FDCT conversion of YCbCr. Find the position of the medium and low frequency parameter and use it as the starting point to remove the watermark.

3. Read the user's password. Create two groups of random number seeds with an encryption algorithm. Use the second group of random number seeds to remove the watermark from the medium and low frequency parameters in a non-random order.

4. Read the quantization form and perform dequantization processing of the material that is removed. Perform an IDCT operation on the processed information to obtain the original watermark.

5. Use the first group of random number seeds to non-randomly rearrange the watermark. Convert from YCbCr to RGB to the original watermark image.

Software Implementation

When we began this project, we started by creating a basic flow and then added functionality. We tested the software at each stage to ensure that it operated correctly. The basic flow in our first ESWM

software version was: read the archive, perfom FDCT, incorporate the watermark into the image, perform IDCT, and output to the archive. The following sections describe the functionality we added to subsequent versions of the ESWM software.

ESWM v0.2: Reading and Managing the Watermark Image

We used a bitmap (BMP) file as our watermark image. The BMP image format is easy to use and is universal. Additionally, the color information in BMP files is stored as RGB and is not compressed, making it easy to work with.

ESWM v0.4: Applying and Removing the Watermark

In this version we implemented basic watermark application and removal functions, such as FDCT, IDCT, quantization, and frequency domain analysis, and experimented with different ways of applying the watermark. We needed to add a strong watermark that could resist attack. The major technologies we used are described as follows:

- *DCT*—The DCT operation is divided into FDCT and IDCT, with conversion from space domain information to frequency domain information and back. This operation allowed us to distinguish the photo's high, medium, and low frequencies and designate the correct area in which to add the watermark.

- *Quantization*—Quantization adjusts the information after the watermark image goes through DCT conversion, so that the watermark information value is similar to the value of the medium and low frequency parameters of the original image to be watermarked. This process cushions the impact of the watermark on the original photo.

ESWM v0.6: Using a Random Number Technique

In this version, we focused on keeping the watermark from damaging the original image. In our previous software version, applying the watermark only uses FDCT and IDCT, and a slight breakage can cause massive loss of the hidden watermark media (see Figure 9). To solve this problem, we added a random number function before applying the watermark to protect the watermark even if the image is damaged. Essentially, we scattered the physical breakage to each part of the watermark so that the watermark would still be distinguished (see Figures 10 and 11).

Figure 9. Damaged Image

Figure 10. Watermark Removed without Random Number Technique

Figure 11. Watermark Removed with Random Number Technique

ESWM v0.8: Improving JPEG Compression Resistance

In this version we focused on improving resistance to breakage during JPEG compression. Before applying the watermark, we used the YCbCr function and then performed the CSC operation on the original photo. Then, we converted from RGB to YCbCr and hid the watermark in the Y data, preserving the watermark when the image is JPEG compressed. Additionally, we improved software efficiency. These improvements are described as follows:

- *YCbCr*—YCbCr is also a color space. JPEG compression converts the information in the RGB color space into YCbCr, in which Y refers to the brightness, and Cb and Cr refer to the chroma. The data is converted into this mode because the human eye is more sensitive to brightness than to chroma. Therefore, JPEG compression preserves the brightness data but only preserves part of the chroma by sampling some proportion of it to reduce the information. Hiding the watermark in the Y segment protects the watermark from any damage resulting from reducing the samples.

- *Quantified Breakage of JPEG Compression*—JPEG compression also involves quantization processing. The JPEG built-in standard quantization form removes most of the DCT high-frequency coefficients, leaving only half of the information. That is, half of the DCT coefficient values in an 8 x 8 area are changed to 0, according to the device and adjustment standard for the compression quality. The quantization affects the amount of information saved in the images. To strengthen the ability of the watermark to resist JPEG compression, we performed numerous tests to find the most appropriate place to store the watermark information.

ESWM v1.0: Use a Hash Function for Security

Previous versions of the ESWM program used fixed random number seeds to execute the scattering work. In this version, we used a secure hash algorithm (SHA-1) encryption technique to ensure the

security of the system. We added a function to request the user to enter a password. Then, we used the SHA-1 to convert the password into two groups of random number seeds to improve security.

ESWM v1.2: Improve DCT Conversion

In this version, we used the quick DCT calculating program created by Mr. Takuya Ooura to enhance software efficiency. The ESWM software can execute the algorithm correctly, read the bitmap image, and apply a watermark. Alternatively, it can remove the watermark from a watermarked image.

CSC and DCT Principles and Architecture

This section describes the CSC and DCT functions, which we implemented in hardware to accelerate the functionality.

CSC and Improving Logic Efficiency

The CSC operation requires many multiplication and addition operations. Converting an RGB pixel to a YUV pixel requires 9 multiplication and 6 addition operations, which equates to 2,359,296 multiplications and 1,572,864 additions for a 512 x 512 image. Because the software spends a long time on these operations, we decided to use the Nios II processor with a CSC hardware accelerator to enhance the circuit's efficiency. We define the CSC matrix algorithm in Petri net mode.

Petri net N = (P, T, A, w,) of the CSC operation has the following architectures:

P = {p0, p1, p2, p3,.... p14}

T = {t1, t2, t3,.... t9}

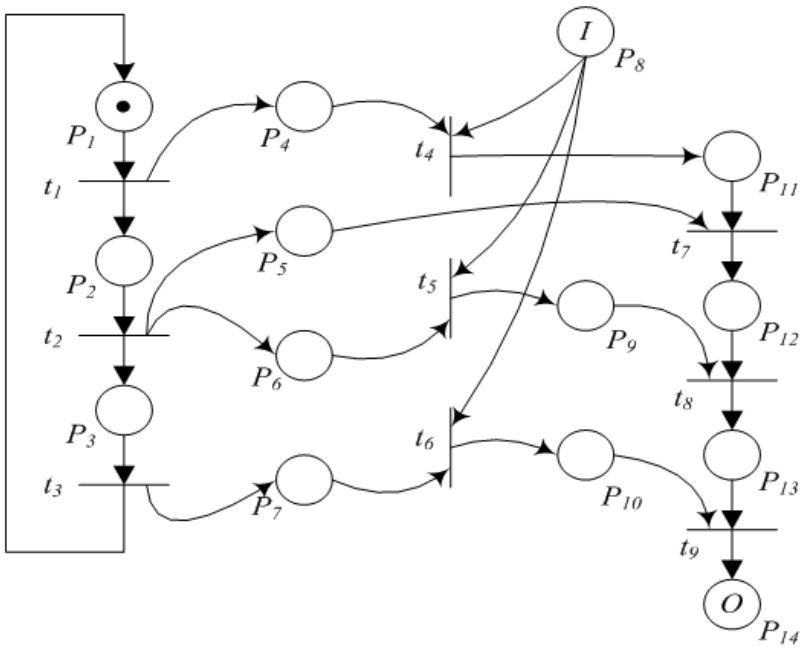
A = {(p1, t1), (t1, p2), (p2, t2), (t2, p2), (p3, t3), (t3, p1), (t1, p4), (t2, p5), (t2, p6), (t3, p7), (p4, t4), (p5, t7), (p6, t5), (p7, t6), (p8, t4), (p8, t5), (p8, t6), (t4, p11), (t5, p9), (t6, p10), (p9, t8), (p10, t9), (p11, t7), (t7, p12), (p12, t8), (t8, p13), (p13, t9), (t9, p14)}

$$w(a) = 1 \forall a \in A$$

$$\vec{x}_0 = [1, 0, 0, 0, 0, 0, 0, I, 0, 0, 0, 0, 0, O]$$

This model is concurrent, including 14 places, 9 transitions, and 28 arcs. See Figure 12.

Figure 12. Petri Net Algorithm of the CSC Accelerator



As shown in Figure 12, the CSC algorithm circuit architecture we implemented is a four-layer pipeline with a two-layer value mode adjustment and a limitation pipeline. See Figure 13.

Figure 13. CSC Implementation Architecture

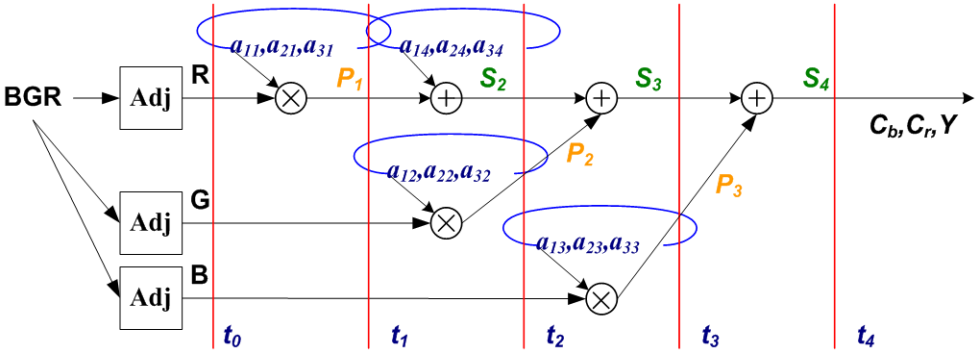
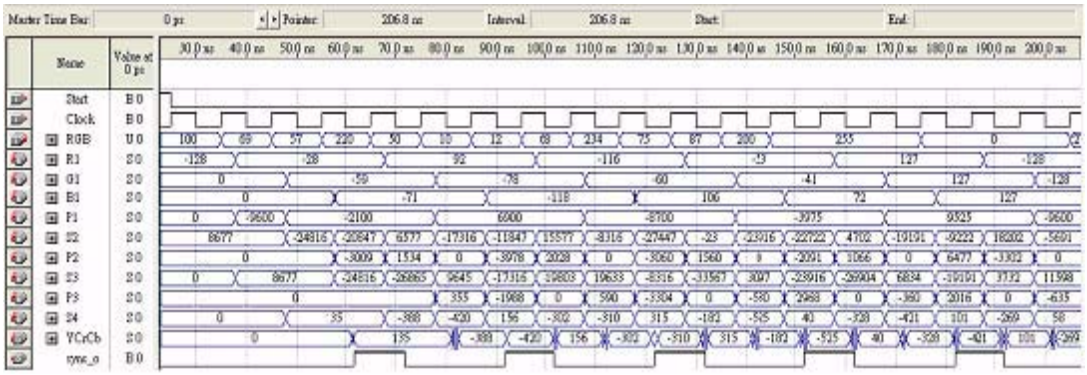


Figure 14 shows the CSC simulation waveform.

Figure 14. CSC Simulation Waveform



DCT Algorithm

According to the efficiency testing we performed, the DCT operation is a bottleneck in the watermark algorithm. Therefore, we decided to implement the DCT by integrating hardware and software to improve performance. For the two-dimensional DCT/IDCT, we implemented a straightforward design that compresses and decompresses the image using a Nios II processor.

1-dimensional (1D) DCT formula:

$$X(k) = C(n)\sum_{n=0}^{N-1} x(n) \cdot \cos(\frac{(2k+1)n\pi}{2N})$$

$$k = 0,1,...,n-1$$

c(n)= { 1/(square root of 2) if n = 0 1, otherwise }

To simplify the cosine operation in 1D DCT conversions, it helps to establish the value of the DCT cosine ahead of time, as shown in Figure 15. Because floating-point operation is not ideal for implementation in a hardware circuit, the values in Figure 15 are multiplied by 256 (i.e., moved left 8 bits) to determine the closest fixed-point numbers (see Figure 16).

Figure 15. DCT Cosine Values

	1	2	3	4	5	6	7	8
1	0.35355	0.35355	0.35355	0.35355	0.35355	0.35355	0.35355	0.35355
2	0.49039	0.41573	0.27779	0.097545	-0.097545	-0.27779	-0.41573	-0.49039
3	0.46194	0.19134	-0.19134	-0.46194	-0.46194	-0.19134	0.19134	0.46194
4	0.41573	-0.097545	-0.49039	-0.27779	0.27779	0.49039	0.097545	-0.41573
5	0.35355	-0.35355	-0.35355	0.35355	0.35355	-0.35355	-0.35355	0.35355
6	0.27779	-0.49039	0.097545	0.41573	-0.41573	-0.097545	0.49039	-0.27779
7	0.19134	-0.46194	0.46194	-0.19134	-0.19134	0.46194	-0.46194	0.19134
8	0.097545	-0.27779	0.41573	-0.49039	0.49039	-0.41573	0.27779	-0.097545

Figure 16. Fixed-Point Values

	1	2	3	4	5	6	7	8
1	91	31	91	91	91	9′	91	91
2	126	106	71	25	-25	-7′	-106	-126
3	118	49	-49	-118	-118	-49	49	118
4	106	-25	-126	-71	71	126	25	-106
5	91	-31	-91	91	91	-9′	-91	91
6	71	-126	25	106	-106	-25	126	-71
7	49	-118	118	-49	-49	118	-118	49
8	25	-71	106	-126	126	-106	71	-25

Figure 17 shows the equation for the case of the eight-point 1D DCT operation.

Figure 17. 8-Point 1D DCT Operation Equation

$$\begin{pmatrix} X0 \\ X2 \\ X4 \\ X6 \end{pmatrix} = 1/2 \begin{pmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{pmatrix} * \begin{pmatrix} X0+X7 \\ X1+X6 \\ X2+X5 \\ X3+X4 \end{pmatrix}$$

$$\begin{pmatrix} X1 \\ X3 \\ X5 \\ X7 \end{pmatrix} = 1/2 \begin{pmatrix} D & E & F & G \\ E & -D & -G & -F \\ F & -D & -G & E \\ G & -F & E & -D \end{pmatrix} * \begin{pmatrix} X0-X7 \\ X1-X6 \\ X2-X5 \\ X3-X4 \end{pmatrix}$$

A=cosπ/4
B= cosπ/8
C= sinπ/8
D= cosπ/16
E= cos3π/16
F= sin3π/16
G= sinπ/16

Design Features

Our design contains the following features:

1. Provides digital watermarking technology as the core function. Using this technology, the design allows the user to embed a signature and establish a photo authentication mechanism to protect the rights of the owner.

2. Develops the system for a Cyclone II EP1C20 device using the Nios II processor and µClinux environment.

3. Makes it difficult to break the watermark, and uses the most appropriate frequency domain for the tests.

4. Allows image manipulation functions, such as rotating, cutting, and changing the image size. The watermark can still be identified in the edited photos.

5. Uses CF cards to store the images and display the watermarked images on a web site.

Conclusion

In our project, the Nios II embedded e-album, the watermark encryption algorithm uses YUV space conversion, as well as DCT, to convert the image to the frequency domain. Quantization processing enhances the image robustness, and we explore the appropriate frequency band in which to apply the watermark to achieve transparency and resistance to JPEG compression breakage. We added a random number function to the embedded algorithm to scatter the watermark throughout the image, protecting the watermark even if the photo is cut or damaged.

After completing the watermarking software, we analyzed the time required for each function using profile software to determine the efficiency. We added in the Nios II version 6.0 floating-point ordering command, accelerating the operational speed of the CSC, DCT, and quantization floating points. We then compared the time differences when using the floating-point ordering command, not using it, or using only software. Our result showed that using the floating-point ordering command is 5 times faster than not using it.

To fully implement the e-album, we needed to have a strong combination of input, output, and human-computer interface. Therefore, we used the μClinux version 2.6.11 kernel on a Cyclone II EP1C20 device and established a file system. Then we created the application program, controlled the CF storage device, generated a Boa web server to display the results, and used a web site as the development environment for the whole system. To implement these functions, we used the Nios II architecture to deploy the relevant algorithms, generate the major input and output (I/O) devices, and drive the program. When selecting operating system features, we needed to consider support, adjustability, and memory allocation flexibility of the Nios II system and the required device so the system can perform properly.