First Prize

Network Data Security System Design with High Security Insurance

Institution: **Department of Information Engineering, I-Shou University**

Participants: **Jia-Wei Gong, Jian-Hong Chen, and Zih-Heng Chen**

Instructor: **Professor Ming-Haw Jing**

Design Introduction

Because of the availability of relatively large bandwidth, computers worldwide are connected via a network. Computer users can transfer information and exchange data easily on the Internet, and as a result, many network data servers have been set up. With this growth, ensuring data security during storage and transmission has become very important. In our project, a channel coding and encryption system protects data, ensuring data security in case of network invalidation. It also prevents data loss for cases in which a single packet of data is revealed (including losing an encryption key). Our goal is to make a new network data security system with high-security insurance.

A traditional network data security system (e.g., network-attached storage devices or a storage area network) achieves high security through encryption/decryption algorithms such as advanced encryption standard (AES), data encryption standard (DES), RC6, and other symmetric key cryptosystems. However, with these schemes data errors have become more frequent, particularly as network transmissions enter the gigabit per second (Gbps) range. Therefore, our new network data security system must consider data accuracy and privacy.

We used the AES block cipher to encrypt the data and we used an RSA algorithm to encrypt the AES key. The key and encypted data are encoded using a Reed-Solomon (RS) encoder and stored, via Ethernet, onto multiple archive data servers. When the data is read from the server, the reverse process occurs and decoding is performed on each server. Because of fault tolerance mechanisms, the server works normally even when data errors occur during transmission or if the server is damaged.

To develop our proposed embedded system, we used a Nios® II processor (a 32-bit RISC soft-core processor) to integrate various peripherals that we downloaded via the Internet. The design's AES, RS encoder, RSA algorithm, and network communication protocol require many look-up tables (LUTs) and hardware design. We used system-on-a-programmable-chips (SOPC) concepts and hardware/software co-design to shorten the design process.

The Nios II processor supports user-defined custom instructions as well as hardware acceleration. It can also perform other functions such as processing, controlling, decision making, and ordering. These features allow the software designer to write the control flow program in C/C++, while most of the computing is done in hardware. The software application includes a custom instruction that operates as a high-speed, hardware-accelerated computing module. In addition to supporting custom instructions, the Nios II Integrated Development Environment (IDE), which includes the GNU C/C++ assembler and Eclipse IDE, is ideal for the entire design process. Furthermore, the designer can perform simulation, development, debugging, integration, and validation in real time on the Development and Education (DE2) board. All of these features make development efficient.

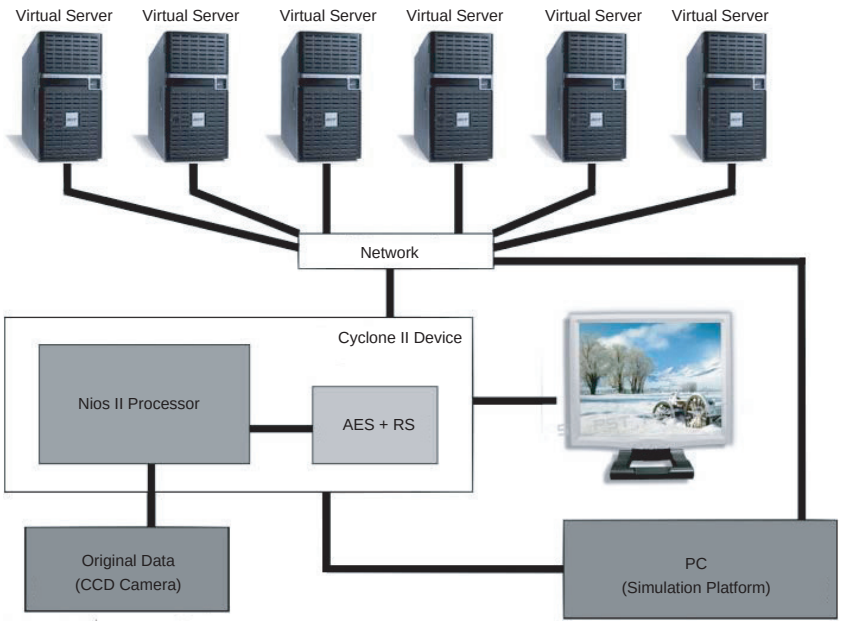
Design Concepts

Using hardware/software co-design helped us complete the network data security system rapidly. The design can be divided into 3 parts, as shown below:

- *System*—The Nios II processor performs the major system control functions. These functions include flow control, general computing, system monitoring, event decision making, receiving and sending data, user interfacing, and implementing an interrupt service routine.
- *Hardware*—The hardware includes an RS encoder, AES, and RSA high-speed operation components, charge-coupled device (CCD) image capture components, dual-port SDRAM controller, Ethernet controller, VGA controller, and SRAM controller.
- *Software*—The software provides a testing and validating platform for the communication between the software and the hardware. We also use it for software simulation and development.

Figure 1 shows the test platform for the system.

Figure 1. Network Data Security System Test Platform



The AES algorithm has two parts: encryption and decryption (see Figure 2). Table 1 shows the operation of these parts (which are described in detail later in this paper).

Figure 2. AES Encryption/Decryption Conceptual Diagram

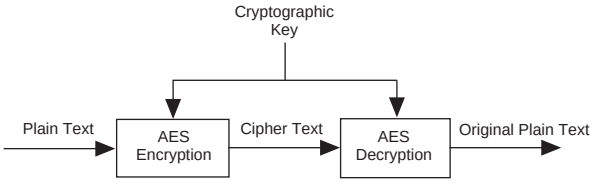


Table 1. AES Work Flow

AES Encryption	AES Decryption
AddRoundKey for round=1 to N-1 SubBytes ShiftRows MixColumes AddRoundKey end for SubBytes ShiftRow AddRoundKey	(Inv)AddRoundKey for round=1 to N-1 (Inv)ShiftRows (Inv)SubBytes (Inv)AddRoundKey (Inv)MixColumes end for (Inv)ShiftRows (Inv)ShbBytes (Inv)AddRoundKey

We use four key components to implement the AES block cipher: SubBytes, ShiftRows, MixColumes, and AddRoundKey. AES security is based on the encryption/decryption key. Therefore, we transmit the key with an open-key encryption system (RSA), which is performed in hardware. See Figure 3. Because the RSA component requires a large amount of integral power and modular operation, we implemented it as a custom instruction and incorporated it into the Nios II arithmetic logic unit (ALU) to improve overall efficiency (see Figure 4). Other hardware elements include the (Inv)SubBytes, (Inv)ShiftRows, (Inv)MixColumes, and (Inv)AddRoundKey AES functions.

Figure 3. Open-Key Encryption/Decryption Conceptual Diagram

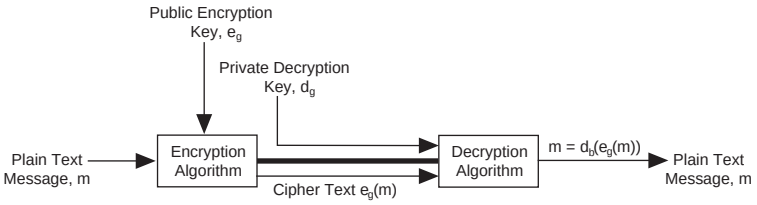
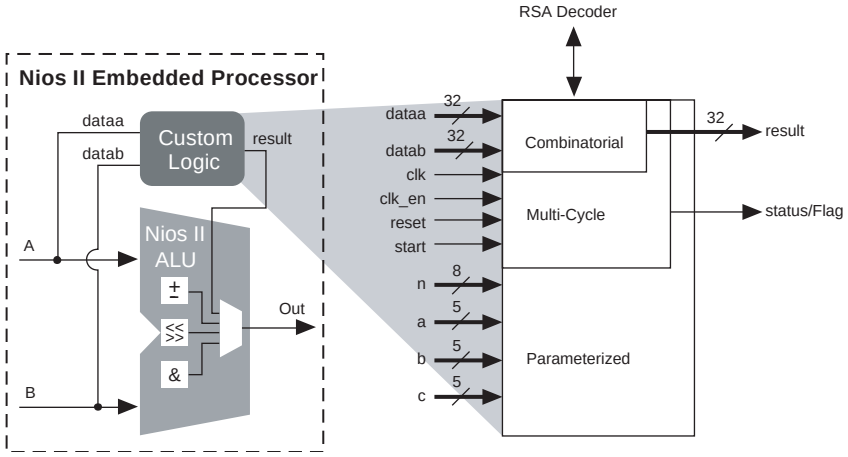
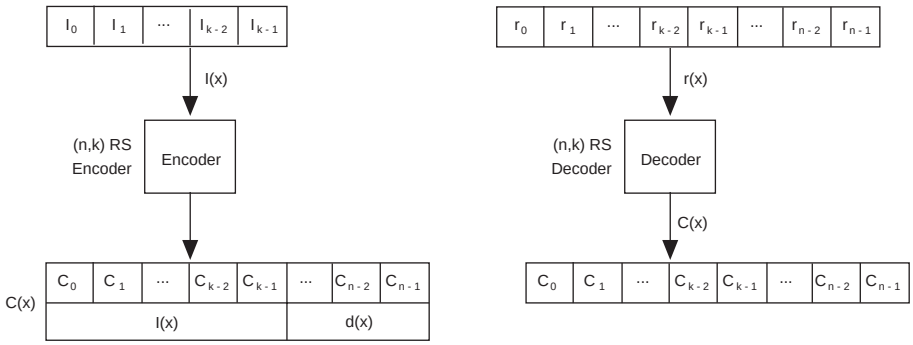


Figure 4. RSA Decoder Custom Instruction



Our design implements an RS encoder and decoder (see Figures 5). Both functions are made up of basic mathematical operations, including multipliers, inverses, and finite field squares. The encoder uses Berlekamp-Massey and chain search concepts, and is implemented in hardware.

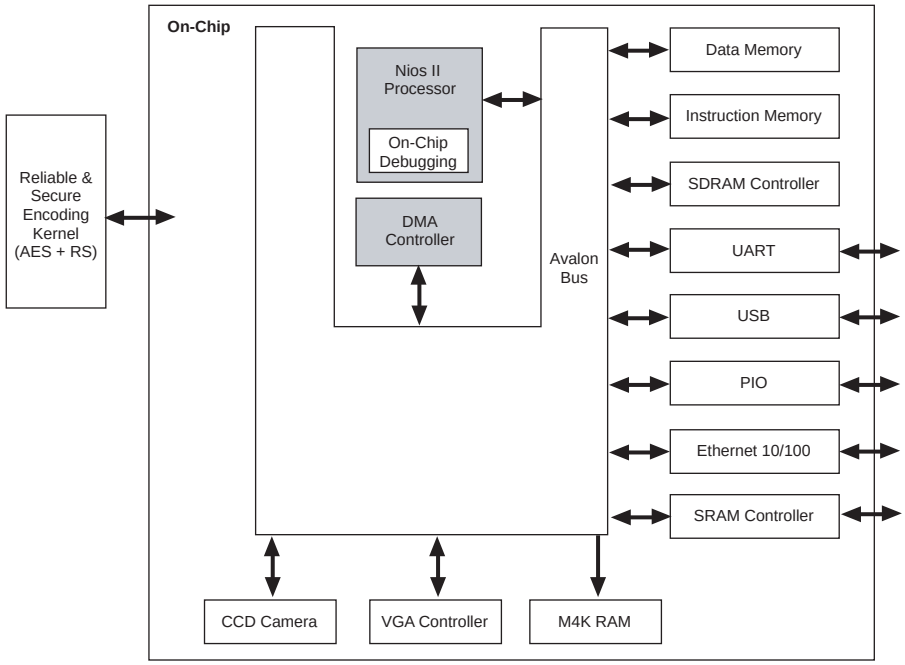
Figure 5. RS Encoder/Decoder Conceptual Diagram



The core infrastructure has the following elements, as shown in Figure 6:

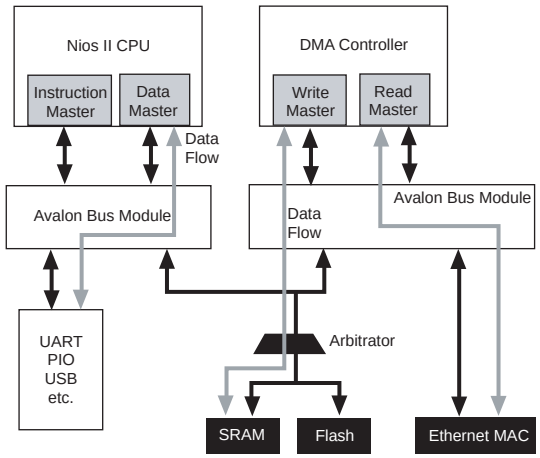
- *High-Speed Data Processor*—Includes the RS encoder/decoder, AES, and RSA operations.
- *CCD Controller*—Connects to the CCD camera via a programmable I/O (PIO) and direct memory access (DMA) to capture images.
- *VGA Controller*—Displays the image captured by the CCD camera on a computer display.
- *SDRAM Controller*—Reads/writes data at high speed.
- *Network Controller*—Performs network card initialization, receives packets, and sends interrupt processing and user datagram protocol (UDP) communications.
- *System Software*—Provides LUT operation, hardware core component control, peripheral control, operation control, data flow control, interface control, interrupt control, etc.

Figure 6. Core Infrastructure



The system's communication interfaces are UART, USB, and Ethernet. They use DMA technology to increase CPU usage time, and they are connected via the Avalon® bus, allowing improved data channel and SDRAM efficiency (see Figure 7).

Figure 7. Communication Interfaces



Application Scope

Our project can be used in the following applications:

- Distributed archive security system
- Archive server

- Network hard drive or mail server
- High-speed database storage system
- Disk array

Target Users

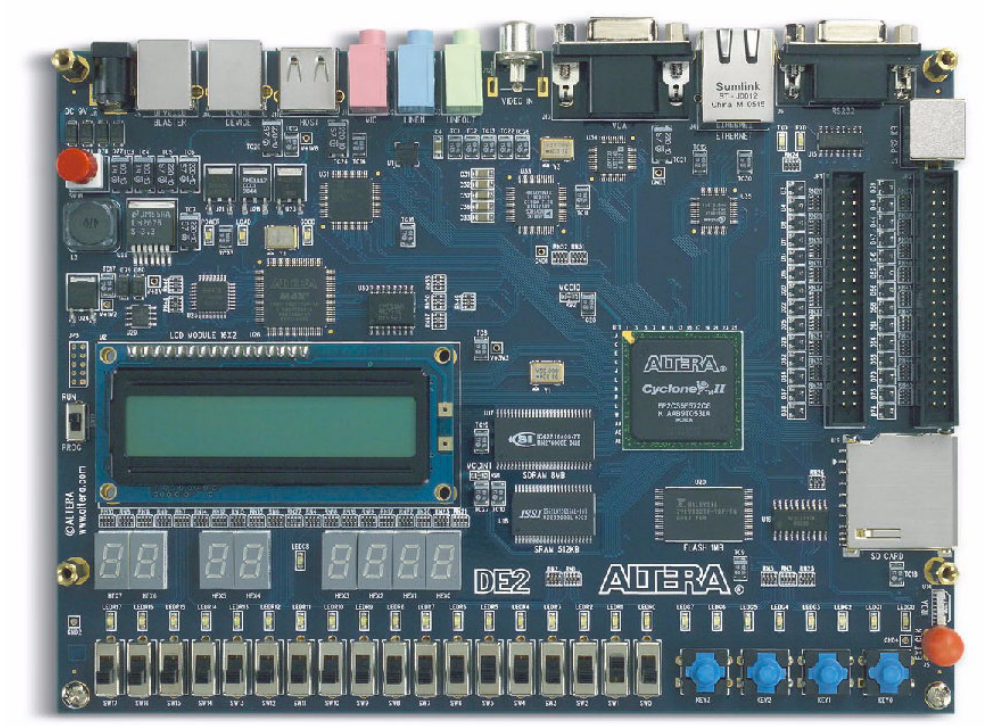
Our project targets the following users:

- Companies that need a high-security data storage system
- Designers of network data security systems that require high security insurance
- Portable communication or storage system producers
- Personal data storage system designers

Development Board

Our design uses the Altera® DE2 development board, which contains a Cyclone® II EP2C35F672C6 FPGA, EPCS16 serial configuration device, 8-Mbyte SDRAM, 512-Kbyte SRAM, 4-Mbyte flash, secure digital (SD) memory card slot, 10/100 Ethernet, RS-232 port, infrared port, etc. See Figure 8.

Figure 8. DE2 Development Board



Function Description

This section describes the function of our design.

Development Steps

We used the following general steps to implement the design.

1. Implement the design's operational blocks (AES, RSA, and RS encoding) in HDL (VHDL and Verilog HDL) using the Quartus® II development environment.

2. Capture an image using a CCD camera, which is displayed in real time with a VGA controller. The captured image is the data source to be encoded and encrypted. The SDRAM image data on the DE2 development board outputs from the computer over the network.

3. Transmit data for real-time encoding/decoding using Ethernet and DMA control.

4. Develop custom instructions to implement the RSA decoder. Return the result directly to the ALU.

5. Build the entire system using SOPC Builder. Complete the hardware/software co-design and demonstration.

6. Simulate AES-128 encryption and RS (6,4,1) encoding/decoding with C++ Builder.

7. Use C++ Builder to complete the network communication GUI interface with the DE2 development board.

8. Simulate several data servers with one PC (for example, it could simulate a situation in which the service of one or more servers is suspended).

9. Test the systems using test data.

Implementation

This section describes how we implemented the design.

Create the Components

Using the Quartus II software version 5.1, we created the AES, RSA and RS encoding functions in VHDL and Verilog HDL. Specifically, we:

1. Analyzed the AES algorithm and developed its four key components. We integrated the encoding and decoding functions for increased CPU efficiency and data property. This functionality is recursive.

2. Analyzed the RS function. We implemented the RS decoder and encoder using a general reliable error-correcting mechanism.

3. Created all components according to the input requirements: multiplier, square, and finite field inverse, KeyExpansion, (Inv)SubBytes, (Inv)ShiftRows, (Inv)MixColumes, (Inv)AddRoundKey, RSA decoder, RS decoder, and RS encoder.

4. Developed the VHDL components using the Quartus II software version 5.1, and performed functional validation and timing simulation.

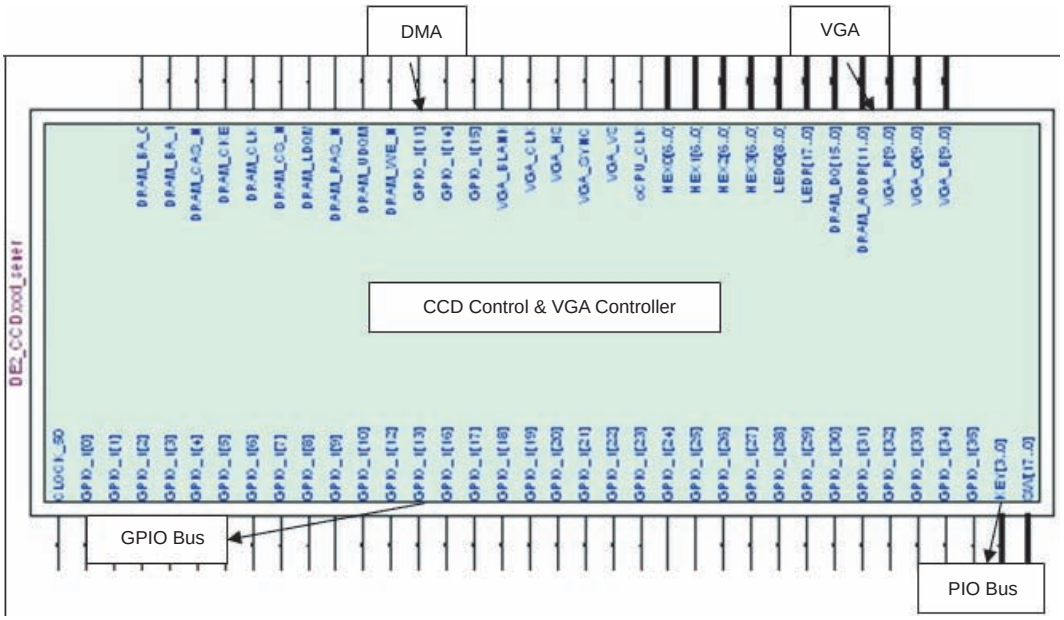
Capture and Display the Image

We captured an image via a CCD camera and displayed it in real time on-screen using a VGA controller. The captured image is the data source to be encoded and encrypted. The following steps describe this process.

1. Use Verilog HDL to connect the CCD controller and VGA controller with the Nios II processor (using a PIO function) as shown in Figure 9.

2. Display the captured image on-screen using a VGA controller. Alternatively, use the Nios II processor to transmit it back to the PC via Ethernet for use by the software and hardware encryption/decryption and encoding validation.

Figure 9. CCD and VGA Controller



Transmit the Data

We implement real-time data encoding/decoding transmission via Ethernet, the USB interface, and DMA control as described in the following steps.

1. Incorporate the Ethernet and USB interfaces by selecting the components in SOPC Builder as shown in Figure 10.

2. Set the arbitration parameter and connect the interfaces with the Avalon bus as shown in Figure 11.

3. Use the Ethernet port to transfer encoding/decoding data between the PC validation platform and the virtual server on the network.

Figure 10. SOPC Builder Components

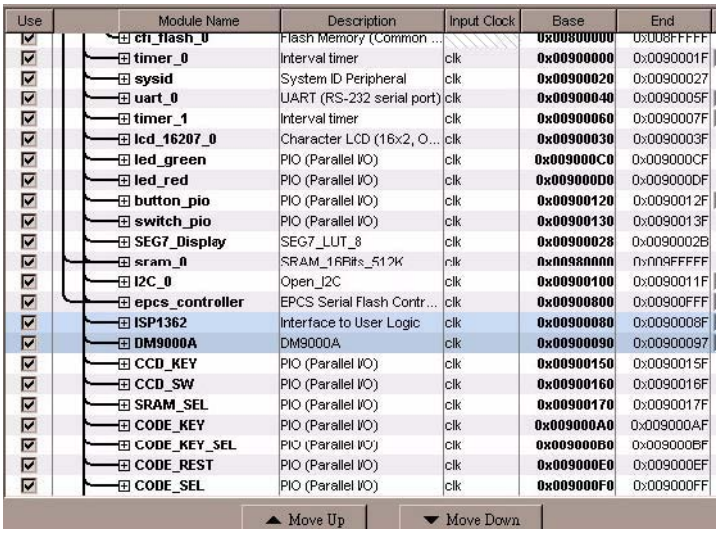
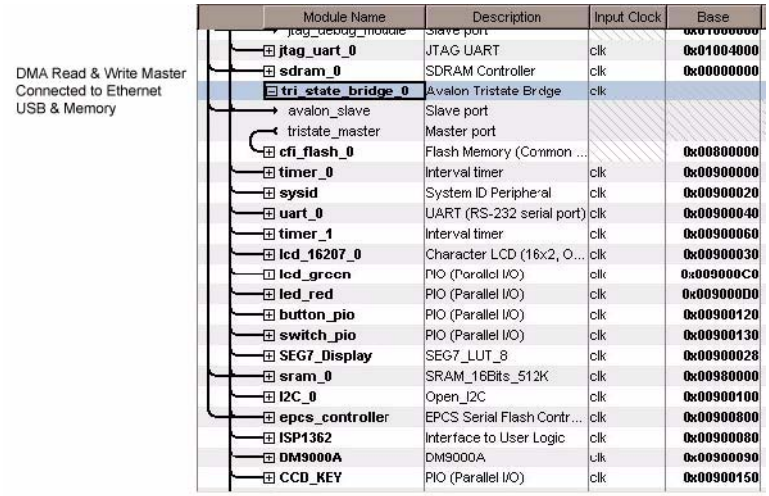


Figure 11. Avalon Bus and Arbitration



Generate the System

We generated the entire system with SOPC Builder as described in the following steps.

1. Open the Altera-provided Cyclone II standard CPU example.

2. Add the user-defined PIO pins, as shown in Figure 12. Table 2 shows the setting of each pin.

3. Design the system infrastructure with SOPC Builder, and generate the **system.h** header file, peripheral drives, and EDIF file.

4. Generate the executable file using the GNUPro encoder, which is downloaded to the Cyclone II device after validation and debugging.

Figure 12. Add User-Defined Pins

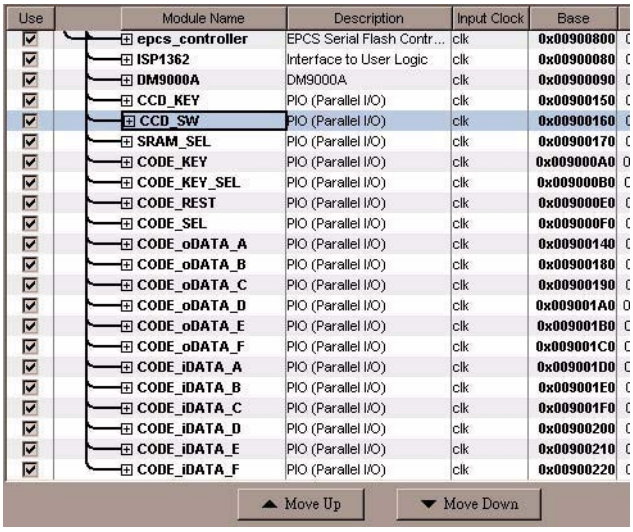


Table 2. User-Defined Pin Functions

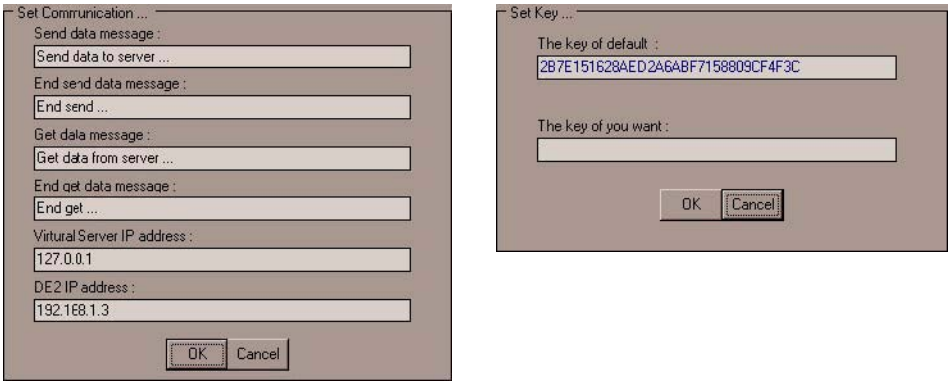
Name	Size (Bits)	Direction	Functions
CCD_KEY	?	Output	Control the CCD control.
CCD_SW	18	Output	Initialize the CCD micro-adjusting.
SRAM_SEL	1	Output	Control the SDRAM.
CODE_KEY	32	Output	Set the encryption key.
CODE_KEY_SEL	6	Output	Control the number of keys.
CODE_REST	1	Output	Initialize the encoding hardware core component.
CODE_SEL	1	Output	Control encoding/decoding.
CODE_iDATA_A~F	32	Input	Transmit encrypted data.
CODE_oDATA_A~F	32	Output	Transmit encrypted data.

Encrypt and Encode Data

We used C++ Builder to complete 128-bit AES encryption and RS encoding/decoding as described in the following steps.

1. Write a user interface program with C++ Builder. Figure 13 shows an example of the parameter settings.

Figure 13. Parameter Settings



2. Write reference rules according to the specification of the Federal Information Processing Standard Publication 197 and RS encoding concepts.

3. Finish the software testing platform gradually and use the platform as the reference for the whole system (see Figures 14 through 20).

Figure 14. Validate Whether the Encoding/Decoding Is Correct

Figure 15. Primary Data

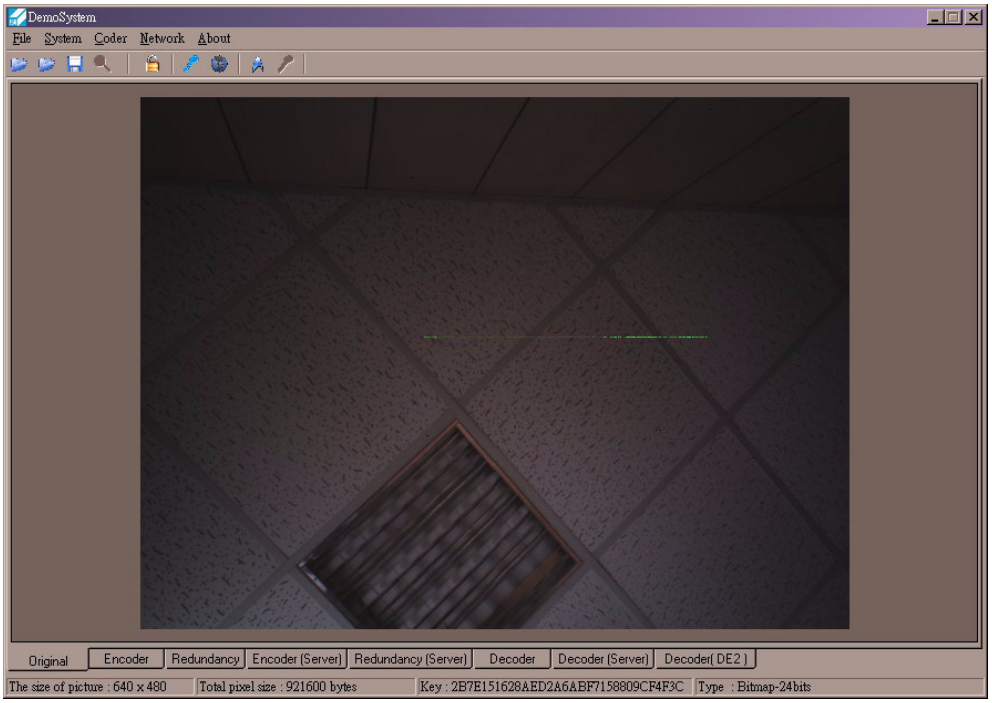


Figure 16. Encode a Large Amount of Data

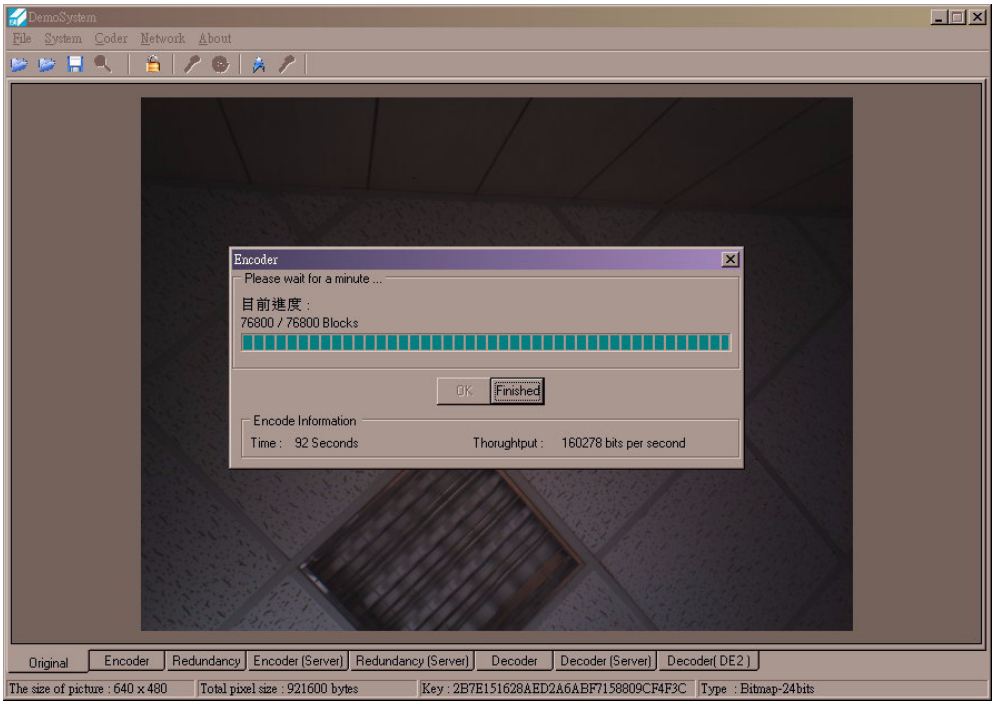


Figure 17. Encoded Data (Encrypted via AES)

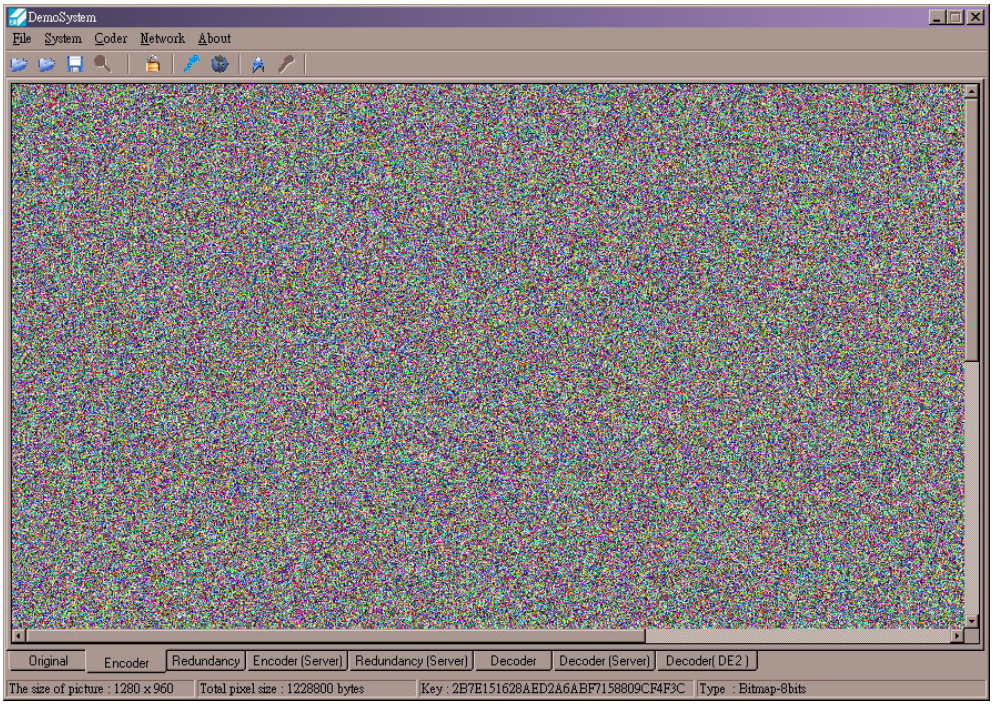


Figure 18. Encoded Data (Redundancy Encoded Using an RS Code)

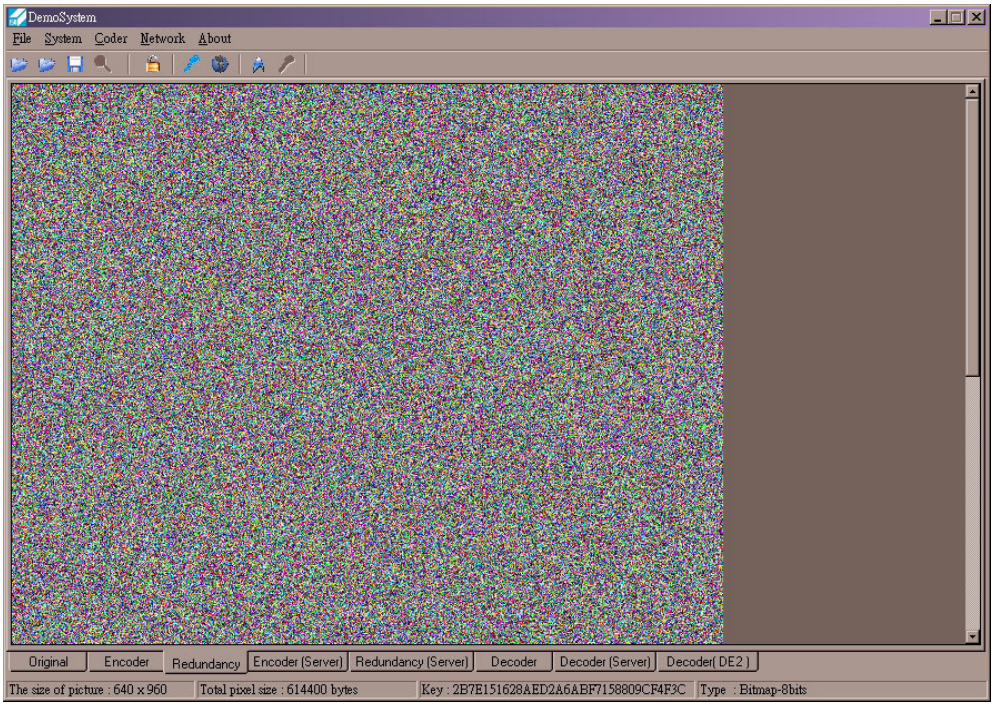


Figure 19. Decode the Data

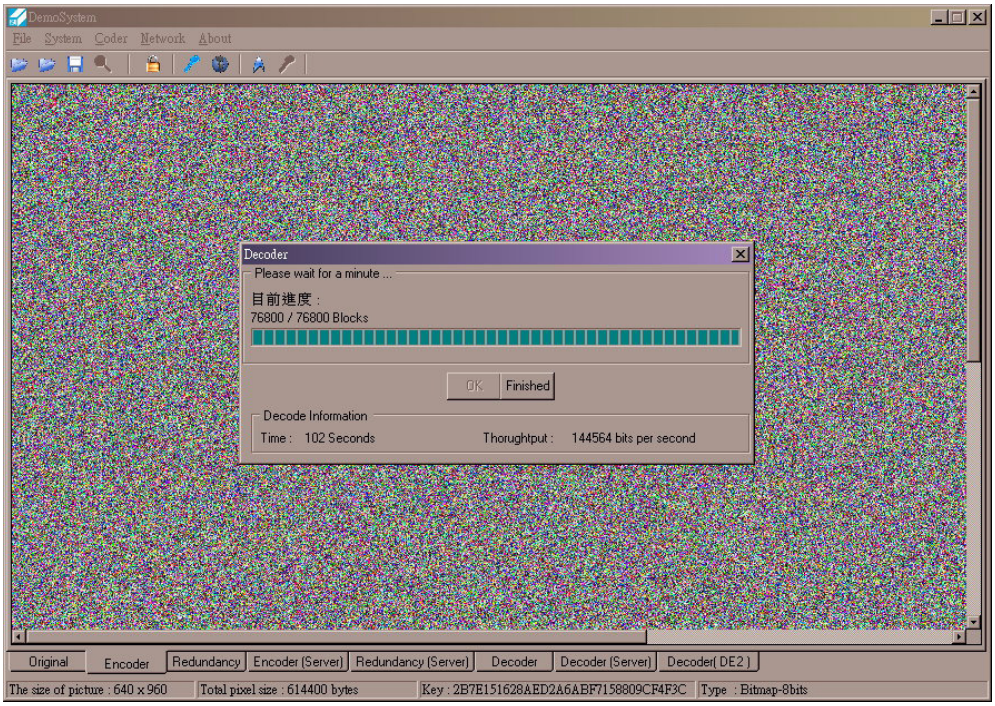
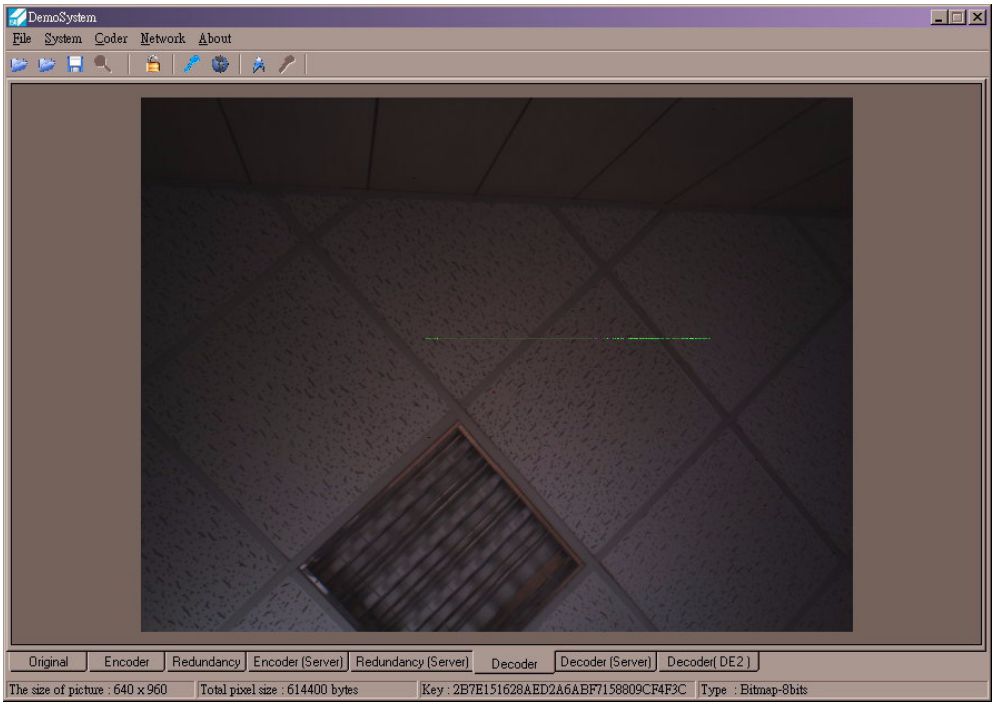


Figure 20. Decoded Data



4. The software application checks whether the encoded data is the same as the decoded data, and later validates it with the data encoded by the Nios II processor for development and debugging.

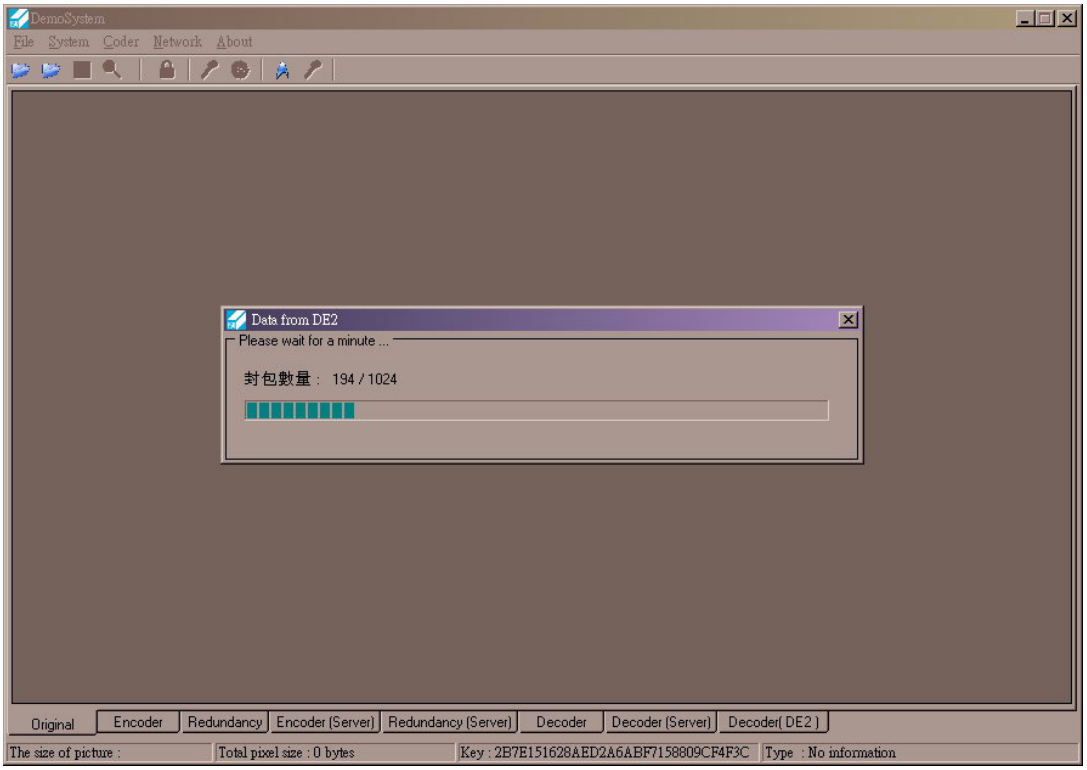
Create the GUI

We built the GUI interface that communicates with the DE2 development board using C++ Builder as described in the following steps.

1. Use the UDP communication protocol to communicate with the DE2 development board over Ethernet.

2. Transmit the CCD image from the virtual server to the Nios II processor for validation, as shown in Figure 21.

Figure 21. Transmit the CCD Image to the Validation Software

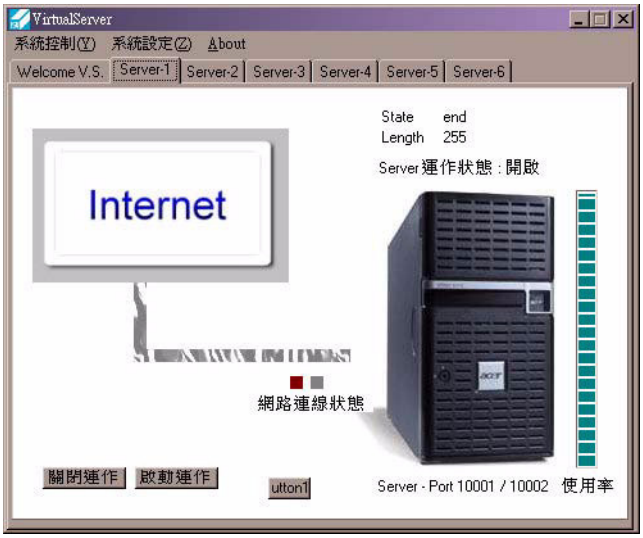


In this case, the data source of the validation software on the PC side will vary, which makes the data more flexible. Additionally, the network function makes sending/receiving of data between virtual servers possible.

Simulate

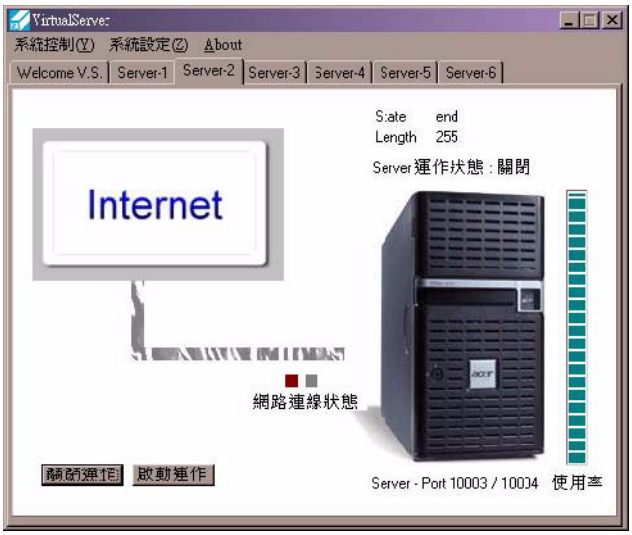
We simulated several data servers using one PC, including simulating situations in which service for one or more servers is suspended. We wrote the back-end GUI using C++ Builder, as shown in Figure 22.

Figure 22. Back-End GUI



We simulated the situation in which service is suspended for one or more servers, as shown in Figure 23.

Figure 23. Service Suspension Simulation



Test the System

We used test data to test the system for the four modes shown in Table 3.

Table 3. Test Modes

Test Number	Test Data	Encoding Mode	Decoding Mode	Performance Comparison
1	CCD camera	Nios II encoding	Nios II decoding	Fastest
2	CCD camera	Validation software encoding	Nios II decoding	Second fastest
3	CCD camera	Nios II encoding	Validation software decoding	Third fastest
4	CCD camera	Validation software encoding	Validation software decoding	Slowest

If a single server is suspended, the data is sent to the virtual server after encoding by the Nios II processor, and then is downloaded to the software validation platform for decoding through the network (as shown in Figures 24 through 27). This flow completes the mode shown in test number 3 in Table 3.

Figure 24. Primary Image Data Captured by CCD

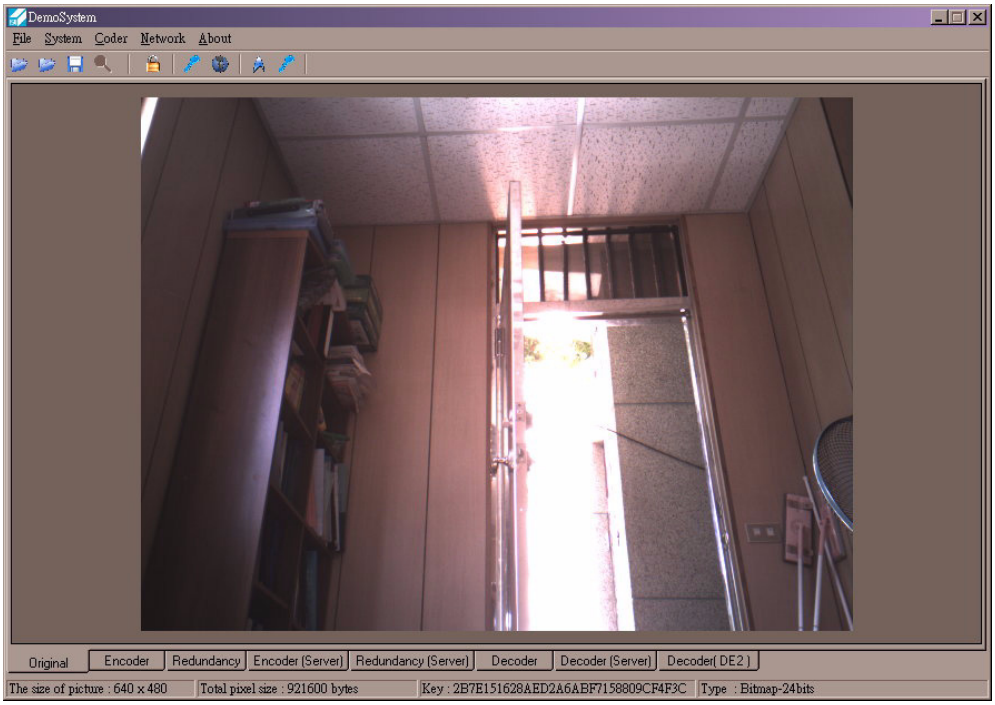


Figure 25. Data after Software Encoding

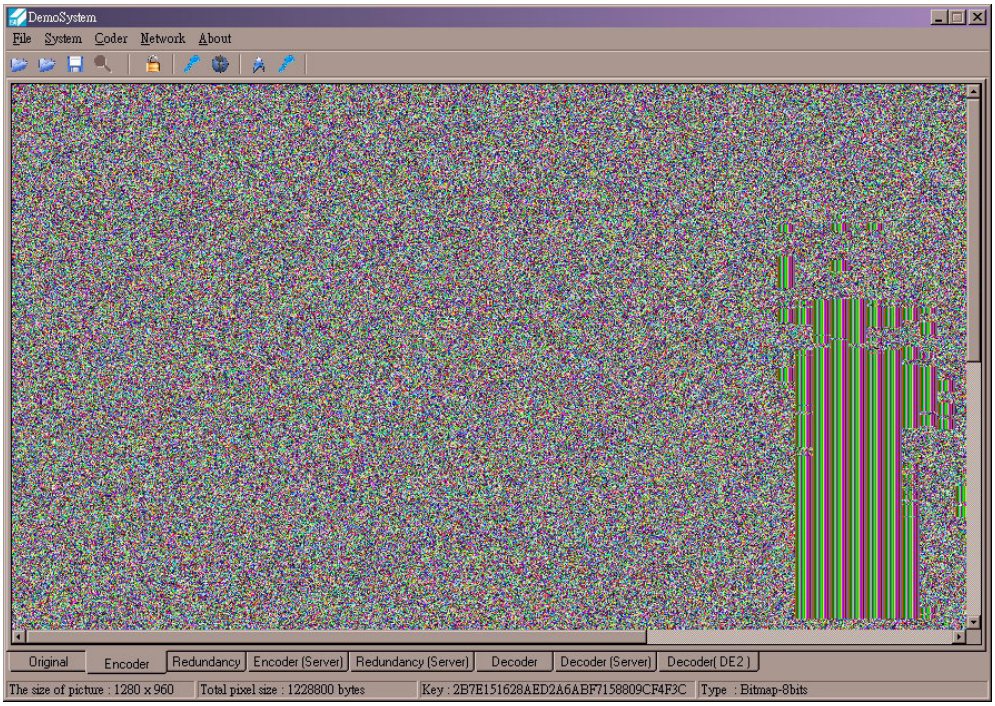


Figure 26. Data Sent to Virtual Server In Case of Service Suspension

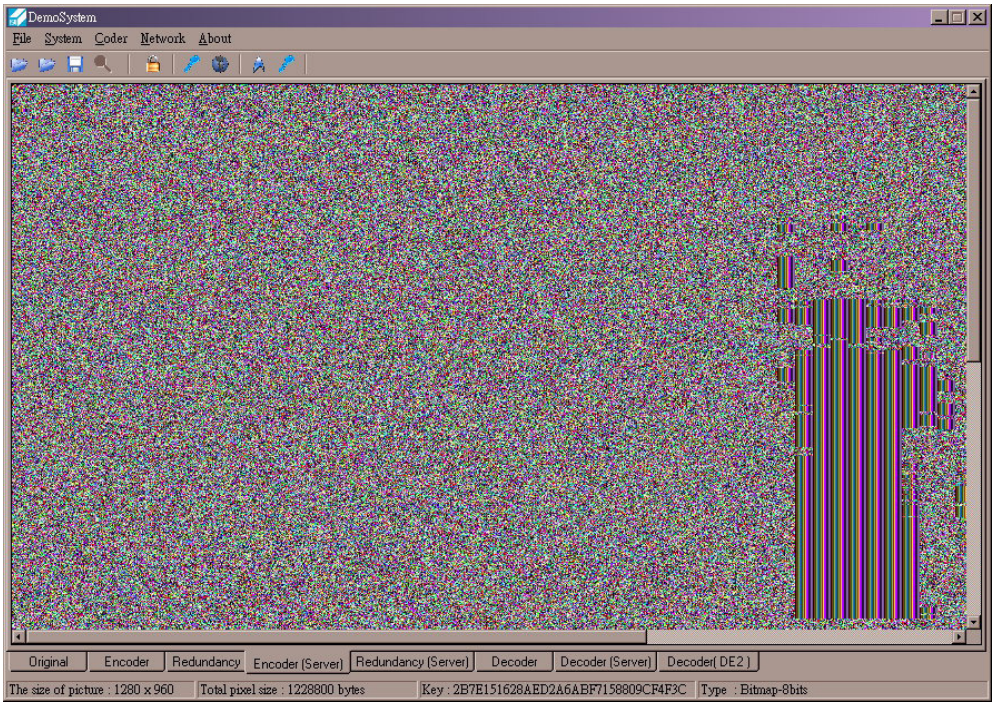
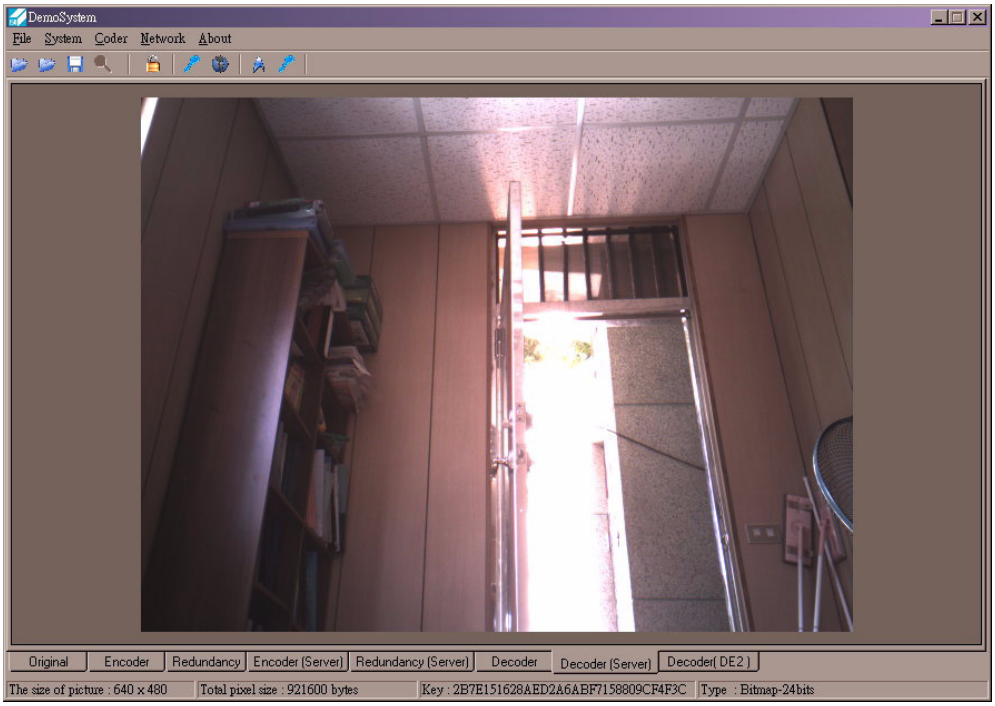


Figure 27. Decoded Result



DE2 Development Board Operation

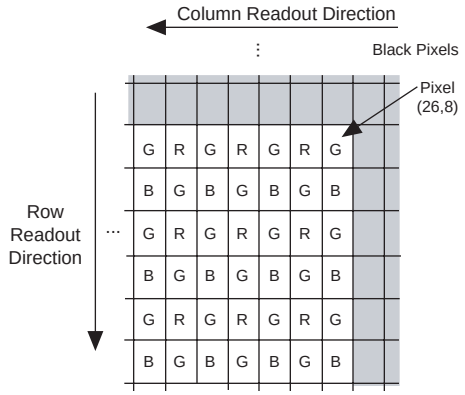
We used the DE2 development board in this design as described below:

- If SW1 is 0 and Key0 is unlocked, the CCD camera captures images and saves them to SDRAM.
- If SW1 is 0 and Key1 is unlocked, the CCD camera stops capturing images.
- If SW1 is 0 and Key2 is unlocked, the Nios II processor transmits the CCD camera image back to the validation software for display on the network.
- If SW1 and SW0 are 0 and Key3 is unlocked, the Nios II processor encodes the CCD camera image and transmits it to the virtual server over the network.
- If SW1 is 0, SW1 is 1, and Key3 is unlocked, the Nios II processor downloads the data from the virtual server through the network and transmits it to the validation software, which displays it after decoding.
- If SW1 is 1 and Key0 is unlocked, the Nios II processor waits for the validation software to transmit the new AES key back so that the RSA decoding process can update the key.

Performance Parameters

Our design provides channel encoding, and after encoding, the data is transmitted over the network. We analyzed the performance of the encoding/decoding process. For test purposes, we used the low-level graphic data of the CCD camera as data source (see Figure 16).

Figure 28. CCD's Low-Level Data Sequence



After the CCD camera captures an image, two sets of image data are generated: RG (8 bits of red and 8 bits of green) and BG (8 bits of blue and 8 bits of green). Because we set the CCD camera to capture 640 x 480 images, our encoded graphics are composed of 640 x 480 pixels at 16 bits times 2 (RG and BG). AES encoding processes 128 data bits at a time, therefore, it needs to encode 640 x 480 x 16 x 2/128 = 76,800 times total. RS encoding can process 192 data bits at a time, therefore, it needs to decode 640 x 480 x 16 x 2 x 1.5/192 = 76,800 times. Each encoding/decoding process uses a large amount of memory for the reads/writes (we use the SDRAM as the memory). Tables 4 and 5 show the performance analysis.

Table 4. Encoding Performance

Function Description	Memory		Expected Cycles (Times)
	Read (Times)	Write (Times)	
Load the encoding data from memory (128 bits)	4	4	200
Encoder	16	24	500
Add the encoded data into memory (192 bits)	6	6	300
Flow and peripheral control	0	0	1,000

Table 5. Decoding Performance

Function Description	Memory		Expected Cycles (Times)
	Read (Times)	Write (Times)	
Load the encoding data from memory (192 bits)	6	6	300
Decoder	24	16	500
Add the encoded data into memory (128 bits)	4	4	200
Flow and peripheral control	0	0	1,000

Table 6 shows the performance for the software encoding/decoding (on the PC) and Nios II encoding/decoding. We found that the Nios II encoding/decoding performance is significantly faster than the PC's software encoding/decoding.

Table 6. DE2 and PC Performance

CPU Type	CPU Frequency	Memory Types & Frequencies
PC	AMD Athlon XP 2600+	DDRAM, 400 MHz
DE2	Nios II processor at 50 MHz	SDRAM, 50 MHz
DE2	Nios II processor at 100 MHz	SDRAM, 100 MHz

Tables 7 and 8 show our performance analysis.

Table 7. Performance Anaylsis (50 MHz)

Operation	Expected Nios II Implementation Duration (s)	Nios II at 50 MHz (Time/Data Rate)	PC Software (Time/Data Rate)	Performance Improvement (Times)
Encoding	[200 + 500 + 300 + 1000] x 20 ns x 76,800 = 3.072 (s)	2.46 s/3.996 Mbps	101 s/0.097 Mbps	41.057
Decoding	[200 + 500 + 300 + 1000] x 20 ns x 76,800 = 3.072 (s)	2.46 s/3.996 Mbps	107 s/0.092 Mbps	43.435

Table 8. Performance Analysis (100 MHz)

Operation	Expected Nios II Implementation Duration (s)	Nios II at 100 MHz (Time/Data Rate)	PC Software (Time/Data Rate)	Performance Improvement (Times)
Encoding	[200 + 500 + 300 + 1000] x 10 ns x 76,800 = 1.536	2.26 s/4.350 Mbps	101sec./0.097 Mbps	44.691
Decoding	[200 + 500 + 300 + 1000] x 10 ns x 76,800 = 1.536	2.26 s/4.350 Mbps	107sec./0.092 Mbps	47.345

We set the network packet size at 1,200 bytes, and there must be some delay betwen packet transmissions. Table 9 shows the overall network transmission performance.

Table 9. Network Performance

Network Platform Used	Expected Network Speed (Mbps)	Packet Delay (ms)	Actual Network Speed (Mbps)
Nios II processor at 50 MHz	20	4	3
PC	20	2 to 4	3 to 6
Nios II processor at 100 MHz	20	2	6
PC	20	2 to 4	3 to 6

Although the network performance is not as good as we expected, the frequency is good enough. If we can improve the network frequency speed, the overall system performance would improve significantly.

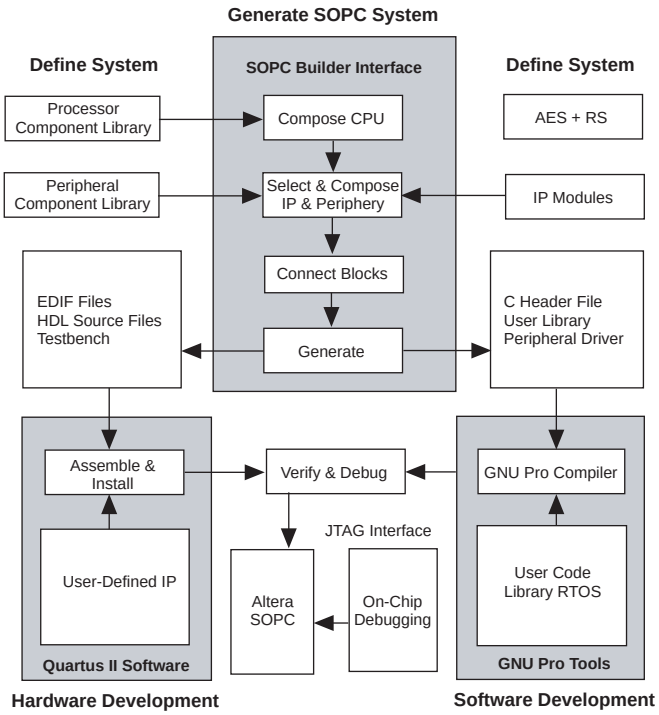
Design Architecture

This section describes our design architecture for the project.

System Development Flow

Figure 29 shows the project development flow.

Figure 29. Development Flow



System Design

Figure 30 shows the system design. The client must have the device acquire the functions of a new network data security system.

Figure 30. System Design

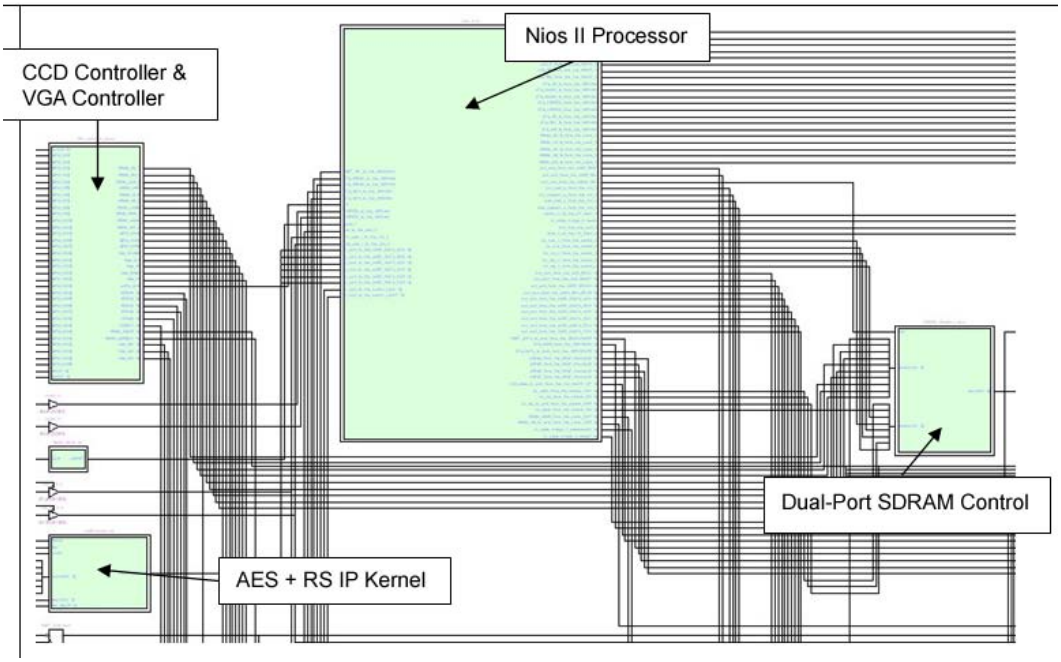


Table 10 shows the parameters in the system user link library.

Table 10. System User Link Library

Parameter	Function
ethernet_interrupts	Interrupts the adapter.
InitHardware	Sets the hardware initialization.
StartCCDToCatchImg	Captures the CCD images and sends them to SDRAM.
StopCCD	Stops capturing the CCD images.
CatchImgSendToPC	Sends the captured CCD images to the PC verification software through the Internet.
CodeDdata	Encodes and decodes data and sends/receives it to the virtual server/PC verification software.
splitPackage	Splits network packets to be sent.
combinePackage	Combines the received network packets.

Hardware Design

This section describes the hardware design.

AES-128 Encryption/Decryption

We designed the AES components recursively. Figure 31 shows the AES hardware diagram. Figure 32 shows the waveform after AES combines with the RS code. We used this design for the following reasons:

- The AES algorithm repeats four specific components many times. If we used a parallel or pipelined design, it would use a substantial number of LEs (about 23,000) although the speed

would be very high (throughput is about 13 Gbps). Additionally, this number of LEs does not provide real-time data. Therefore, we did not adopt this design.

- Using the AES algorithm, encryption and decryption are almost the same. Therefore, we can co-design both functions and use a multiplexer to select which function to use.

- By setting one round as the unit element and designing the AES algorithm recursively, we reduced the LE usage significantly (about 2,300) but the design is slower. However, the overall efficiency is sufficient because the hardware computes the AES algorithm rapidly (the CPU can receive a value 3 cycles after sending datd).

Figure 31. AES Hardware

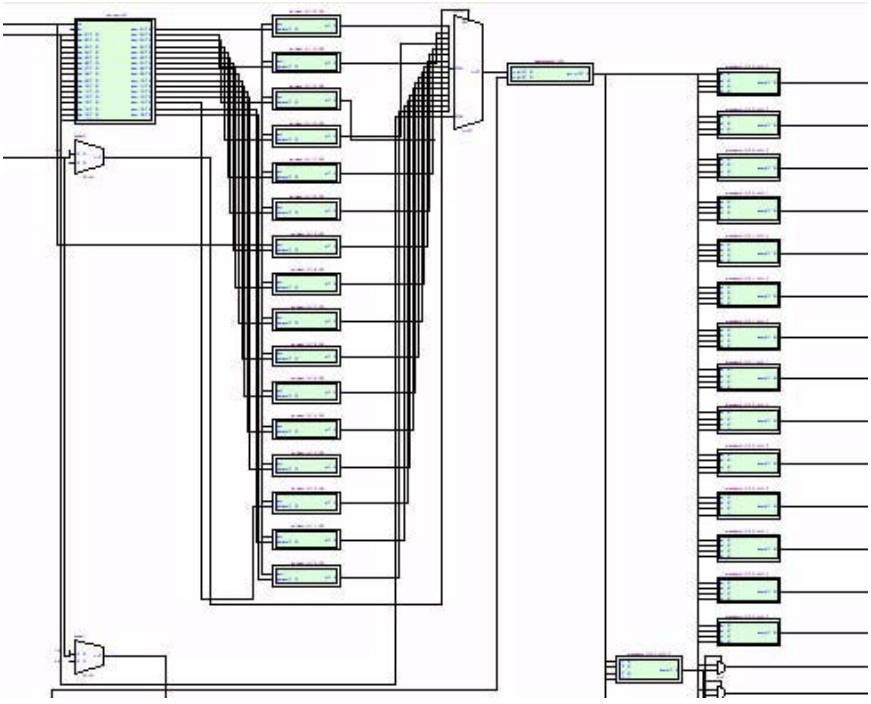
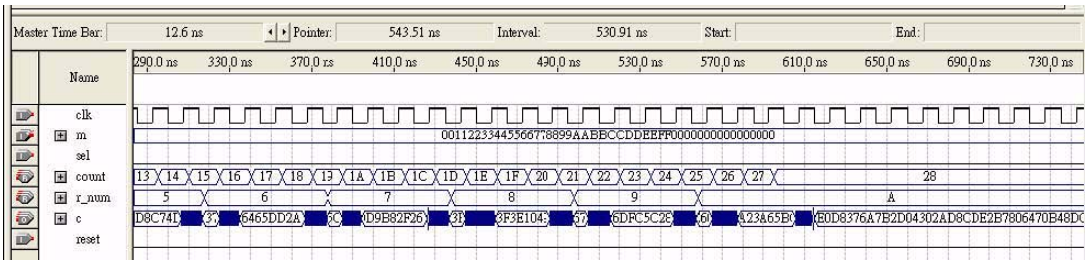


Figure 32. AES and RS Code Waveform



(Inv)SubBytes

We implemented the (Inv)SubBytes function using a LUT. The encryption LUT is referred to as S-Box, and the decryption one is referred to as Inverse S-Box. Both LUTs are 256 bytes and are stored in M4K RAM. A muliplexer selects whether encryption or decryption is performed (see Figure 33). In contrast, parallel computing of the encryption /ecryption requires 16 LUTs and the overall design requires a memory space of 256 x 2 x 16.

Figure 33. Realize (Inv)SubBytes Components Using a LUT

(Inv)MixColumes

We use the xtime component, mentioned by Rijndael (the author of the AES algorithm) in the specifications, to design the (Inv)MixColumes component. a(x) represents a 4-byte polynomial expression and the xtime component effect is the value of mod (x4 + 1) after multiplying x by the a(x) polynomial. We needed to use the component many times in the (Inv)MixColumes function, so we decided to co-design the encryption/decryption MixColumes components. The MixColumes component has two outputs and a multiplexer that selects which function to use. See Figures 34 and 35.

Figure 34. (Inv)MixColumes Design

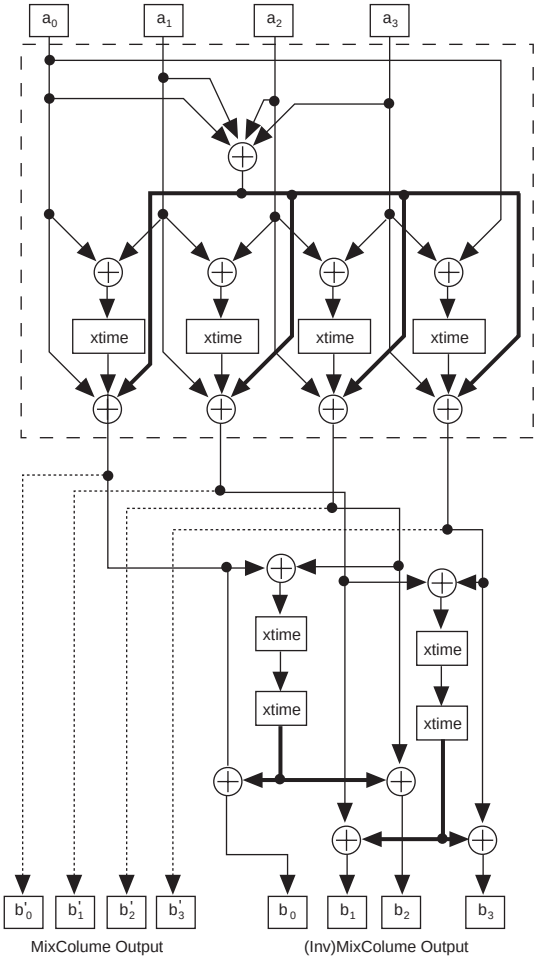
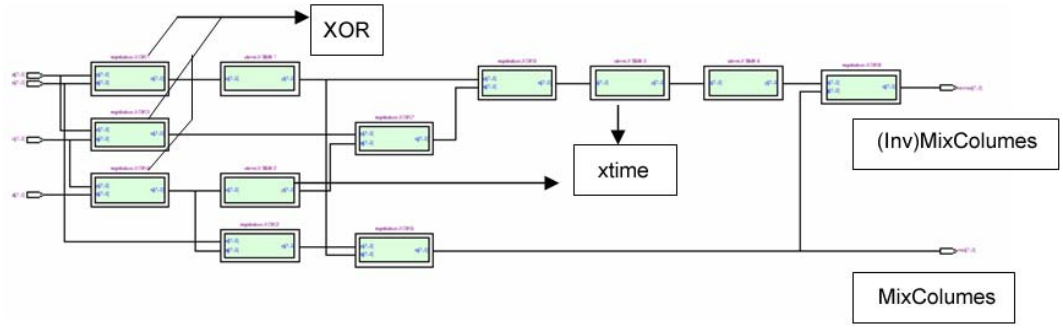


Figure 35. (Inv)MixColumes Implementation



(Inv)ShiftRow and (Inv)AddRoundKey

We created the (Inv)ShiftRow component by changing lines of hardware (see Figure 36). (Inv)AddRoundKey is composed of XOR components (see Figure 37). These designs are quite simple, so we will not describe them further.

Figure 36. (Inv)ShiftRow Components

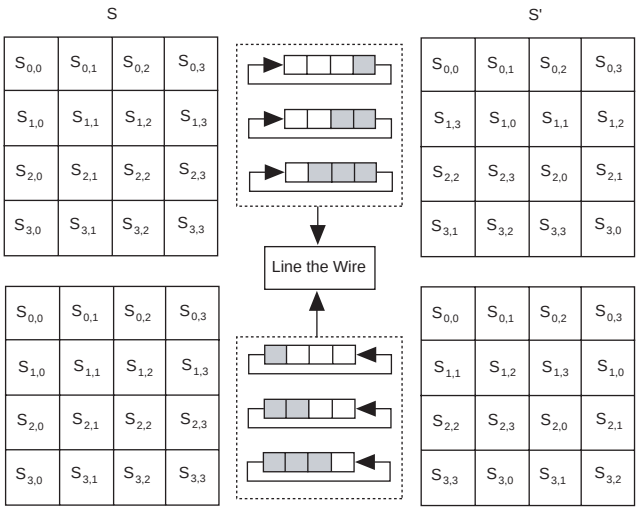
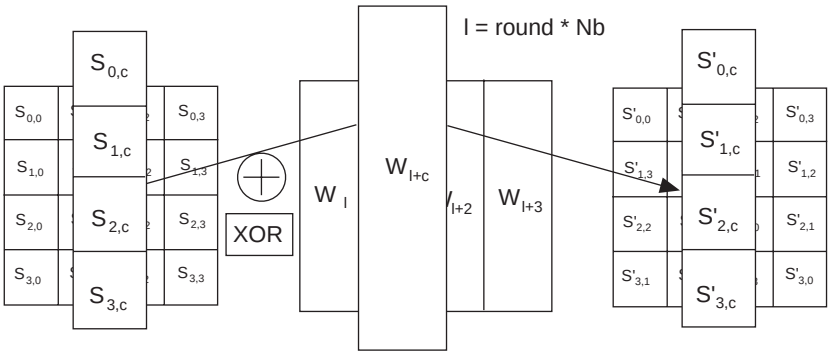


Figure 37. AddRoundKey Components



RS Encoder/Decoder (6,4,2)

The RS algorithm uses system encoding. It replenishes the r(x) with m(x) during encoding. After encoding, the message can remain on the string; therefore, this method greatly improves the decoding efficiency.

RS Encoder

Figures 38 and 39 show the RS encoder hardware design. The original data, M(X), is divided, generating r(x), which is added to m(x) and forms a new codeword. Figure 40 shows the RS encoder waveform.

Figure 38. RS Encoder Conceptual Diagram

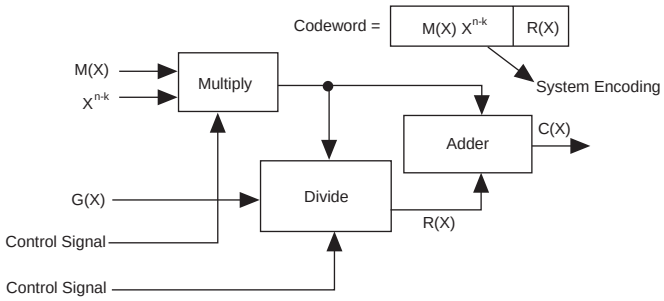


Figure 39. RS Encoder Hardware

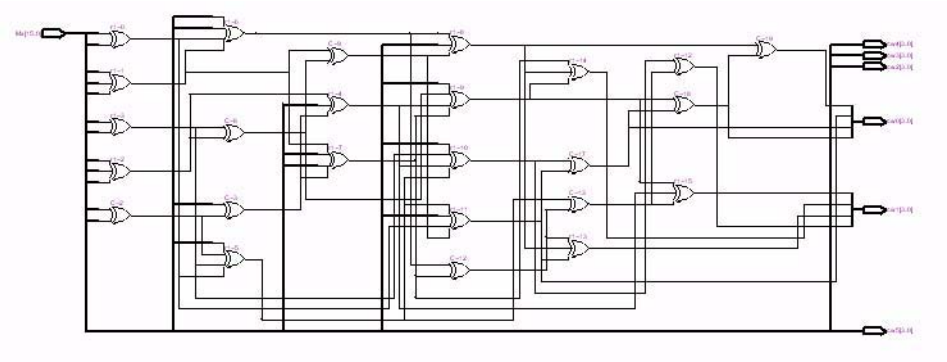
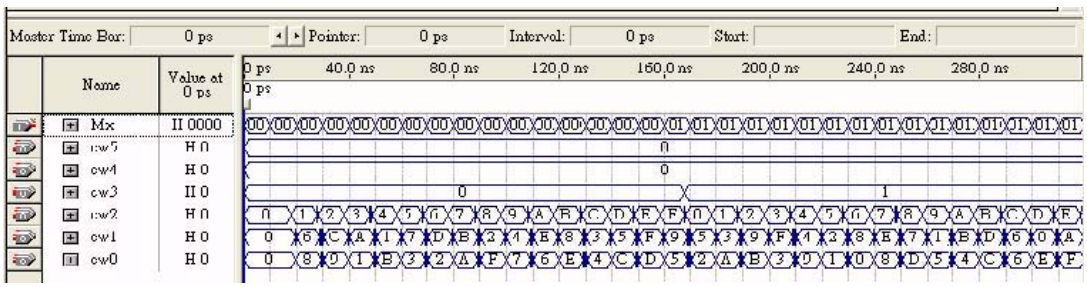


Figure 40. RS Encoder Waveform



RS Decoder

Figures 41 and 42 show the RS decoder hardware design. The decoder decodes the codeword that was output from the encoder via the hardware kernel. The error location and value can be obtained using the

Berlekamp-Massey algorithm and Forney algorithms, respectively. Figure 43 shows the RS decoder waveform.

Figure 41. RS Decoder Conceptual Diagram

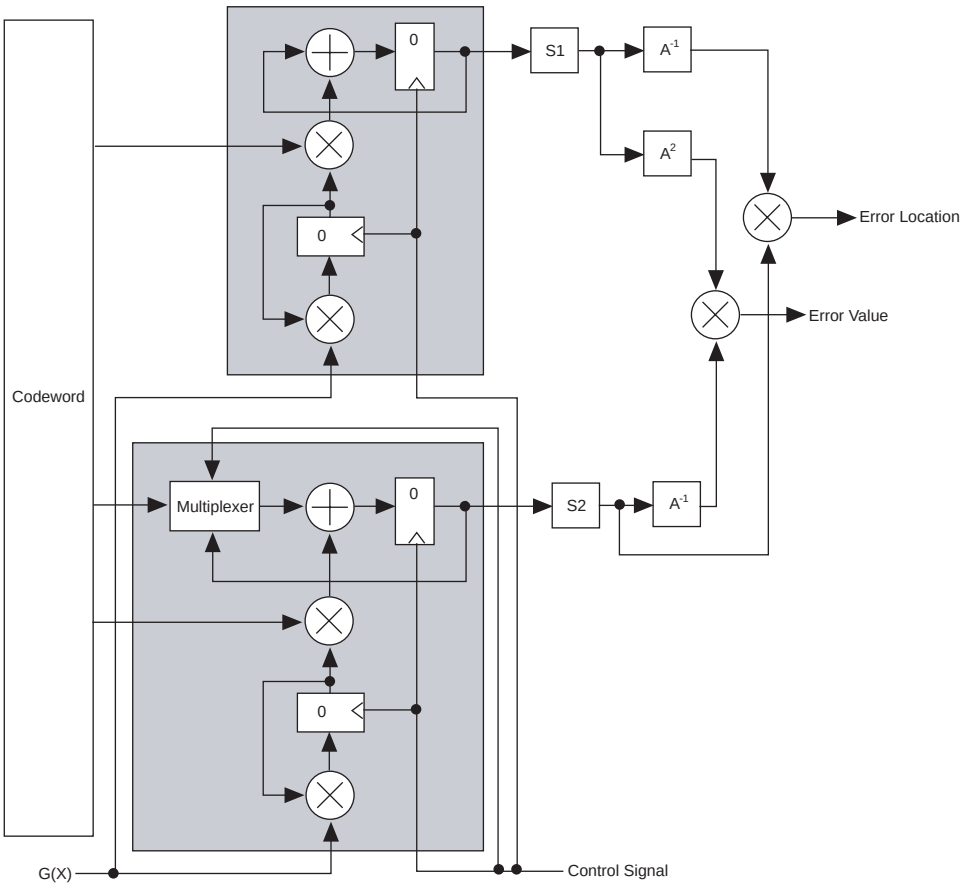


Figure 42. RS Decoder Hardware

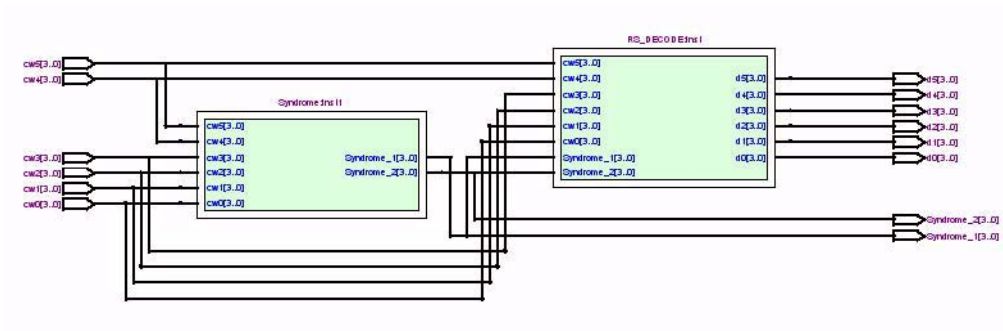
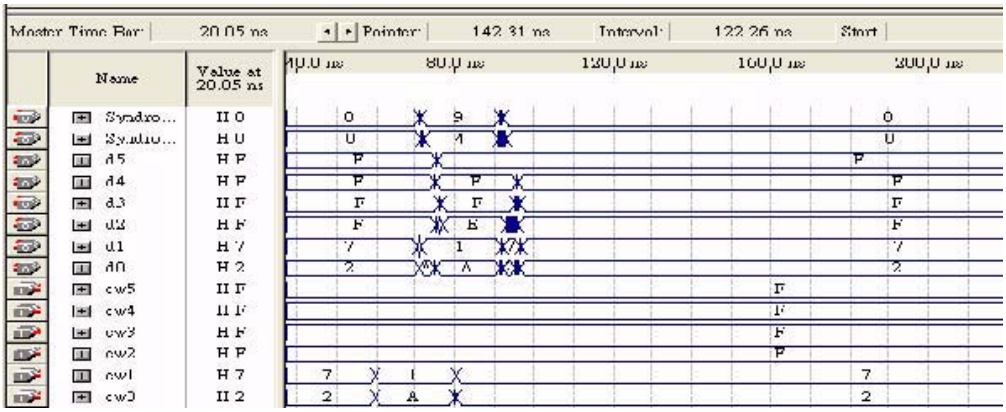


Figure 43. RS Decoder Waveform



RSA Decoder

In the RSA decoder, one byte is a unit. Table 11 shows the RSA parameters. RSA decoding involves integer powers and modular arithmetic, and is time consuming. Therefore, we used a multi-cycle custom instruction that operates using three cycles. Table 12 shows the pin descriptions. Figure 44 shows the RSA decoder waveform.

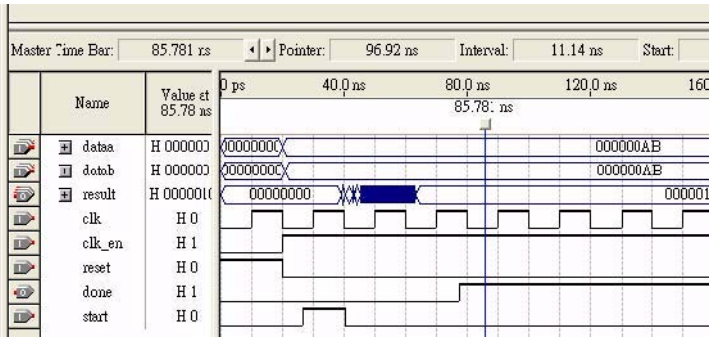
Table 11. RSA Software Pseudo-Code

Variable Name	Value
p	31
o	11
n = pxo	341
z = (p - 1) x (q - 1)	300
e: public encryption key	79
d: primate decryption key	19

Table 12. RSA Pins

Pin Name	Direction	Description
dataa	Input	Operand 1
datab	Input	Operand 2
result	Output	Result
clk	Input	Frequency
Clk_en	Input	Frequency enabled
reset	Input	Reset
done	Output	Signals are finished
start	Input	Capture operand

Figure 44. RSA Decoder Waveform



Software Flow

This section describes the software flow.

Function Flow

Figure 45 shows the function flow. The flow is also described as follows.

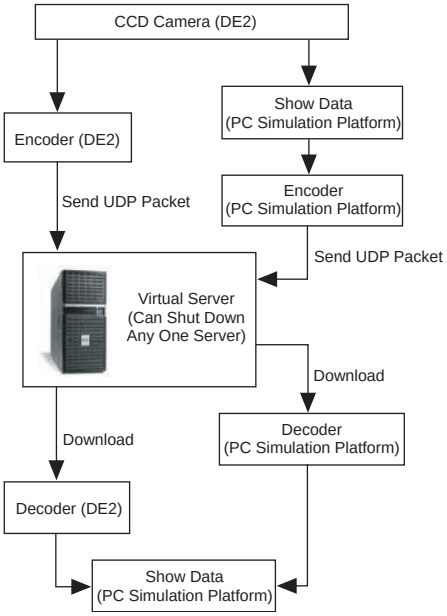
- *Software encoding/decoding simulation*—Send the captured CCD camera images to the DE2 development board to the verifying software. Then send the images to the virtual server through the Internet via the software-simulated coding. The virtual server then ends its active state. Download the data via the Internet to the verifying software to decode and verify the result.

- *Hardware encoding/decoding*—Upload the captured CCD camera images to the DE2 development board and upload the encoded data to the virtual server via the Internet. The virtual server then ends its active state. Download the data via the Internet to the DE2 development board to decode. Send the data back to the verifying software and check the result.

- *Software encoding and hardware decoding*—Send the captured CCD camera images to the DE2 development board to the verifying software. Then, upload them to the virtual server through the Internet via the software-simulated encoding. The virtual server ends its active state. Download the data via the Internet to the DE2 development board to decode. Return the data to the verifying software and check the result.

- *Hardware encoding and software decoding*—Upload the captured CCD camera image to the DE2 development board and upload the encoded data to the virtual server via the Internet. The virtual server then ends its active state. Download the data via the Internet to the verifying software to decode and check the result.

Figure 45. Software Function Flow



Software Pseudo-Code and Function Description

Table 13 shows the software pseudo-code.

Table 13. Software Pseudo-Code

Encoder	Decoder
Load data from DE2 (CCD). Show data on simulation platform. Encode data. Send data to virtual server.	Download data from virtual server. Decode data. Show data on simulation platform.

Table 14 describes the software functions.

Table 14. Software Functions

Function	Description
Encoder	AES + RS encoding core
Decoder	AES + RS decoding core
splitPackage	Network package split
combinePackage	Network package combine
LowDataToRGB	Transform the low data of captured CCD images into RGB
Encryption	AES encryption core
Decryption	AES decryption core
RS-Encoder	RS encoding core
RS-Decoder	RS decoding core

Function Highlights

The function highlights are:

- *Environment setting*—The user can set environment variables and increase the use of flexible spaces (see Figure 46).
- *Image processing*—The data resources are diversified, as shown by the captured CCD camera images on the DE2 development board (see Figure 47).
- *Data encoding*—We provide functions such as encoding/decoding time analysis optimization (see Figure 48).
- *Upload to the virtual server through the Internet*—The user uploads the encoded data through the Internet (see Figure 49).
- *Virtual server stops operation*—The user can easily stop the virtual server with a button (see Figure 50).
- *Download from the virtual server through the Internet*—The user downloads data onto the virtual server via the Internet (see Figure 51).
- *Data decoding*—The application decodes the downloaded data and provides specific time analysis and decoding optimization (see Figure 52).

Figure 46. Environment Setting

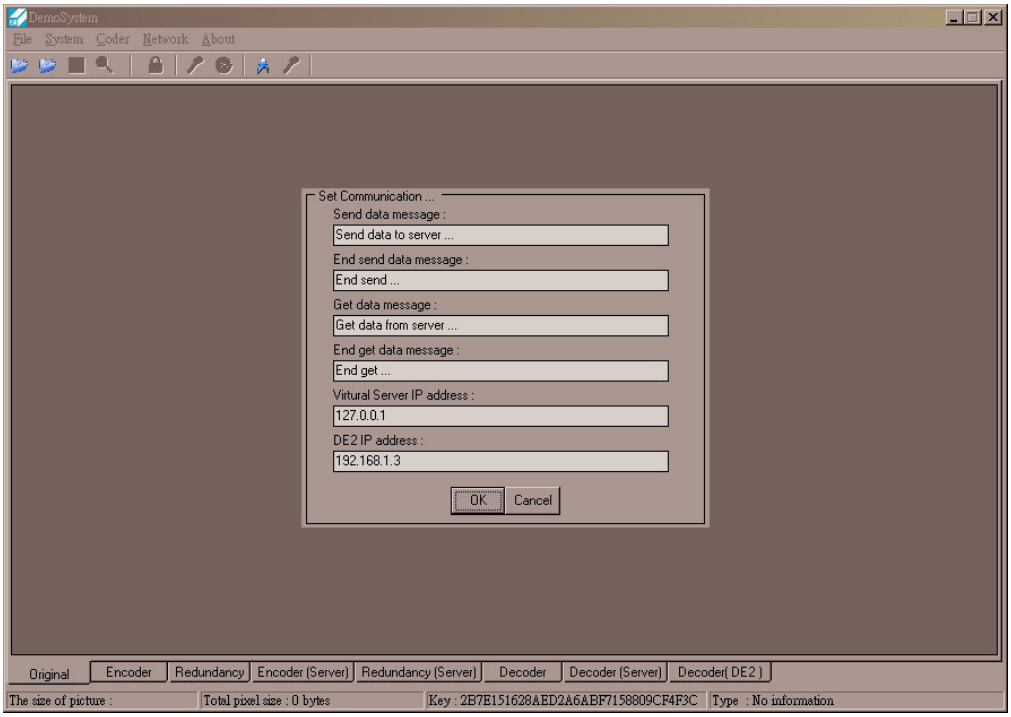


Figure 47. Image Processing

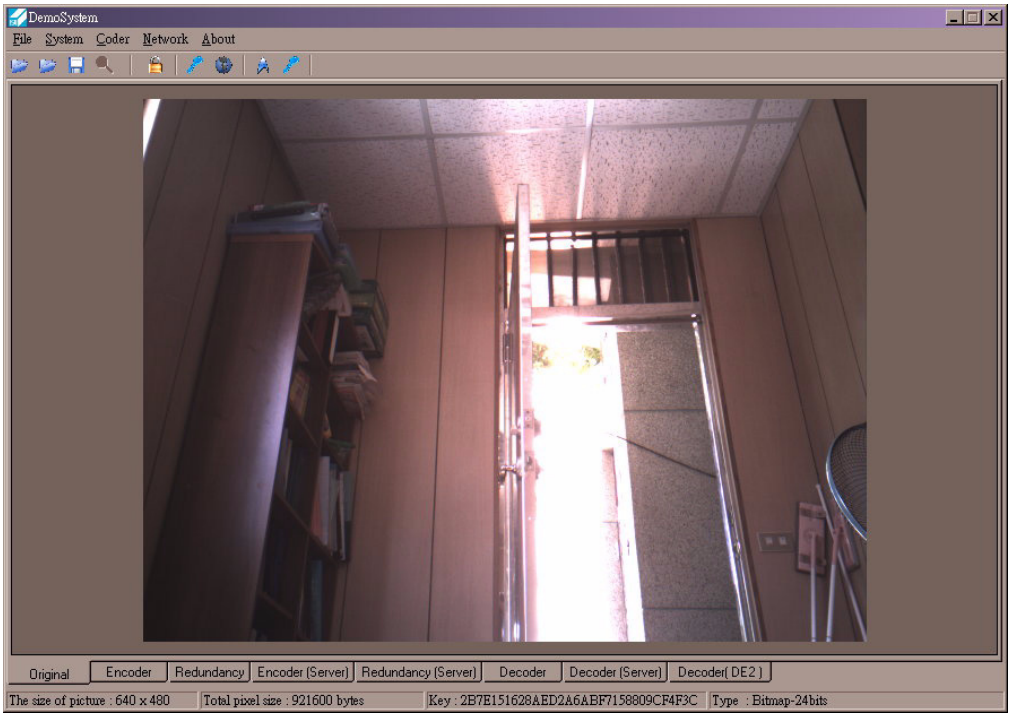


Figure 48. Data Encoding

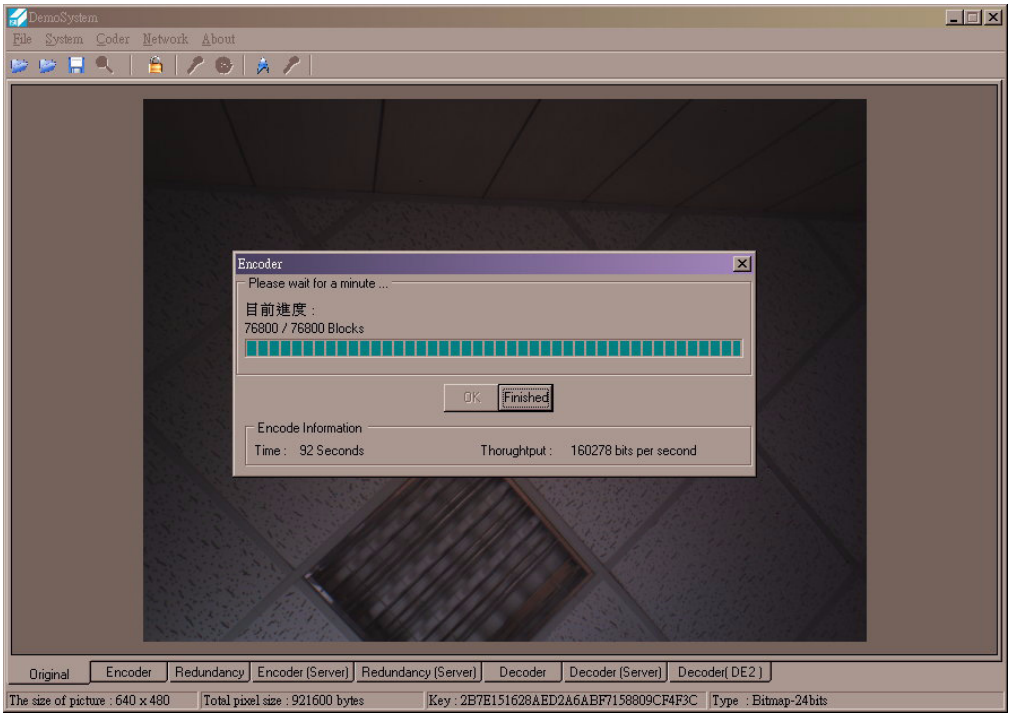


Figure 49. Upload to Virtual Server

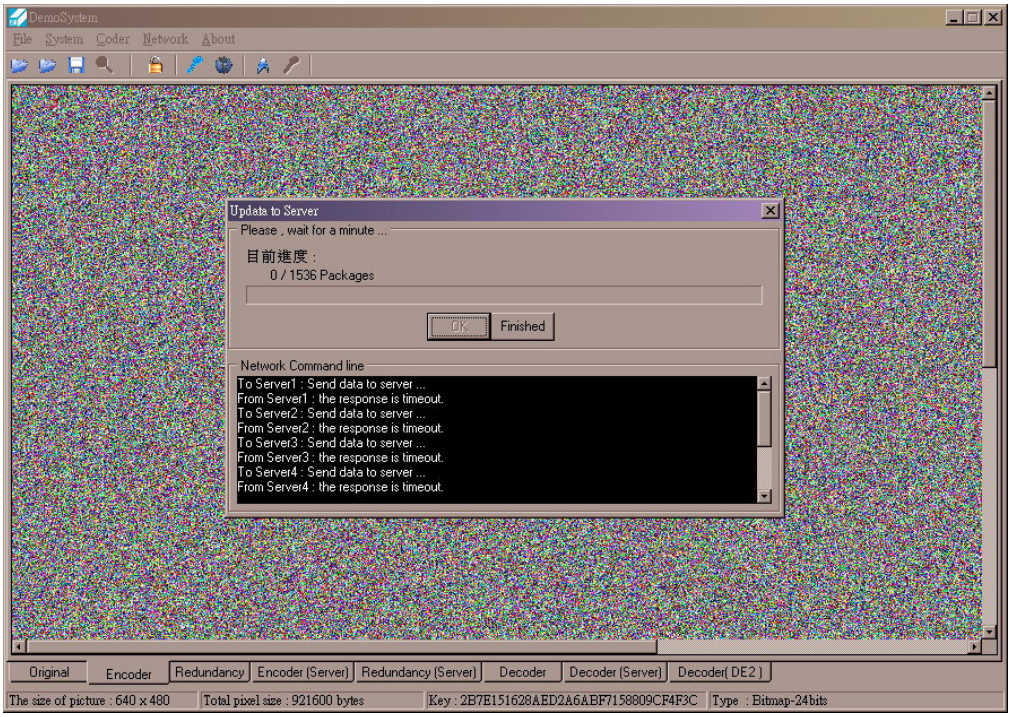


Figure 50. Stop Virtual Server

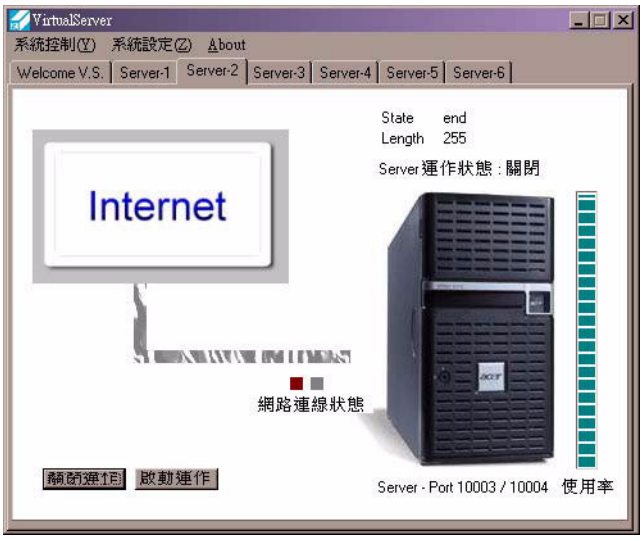


Figure 51. Download from Virtual Server

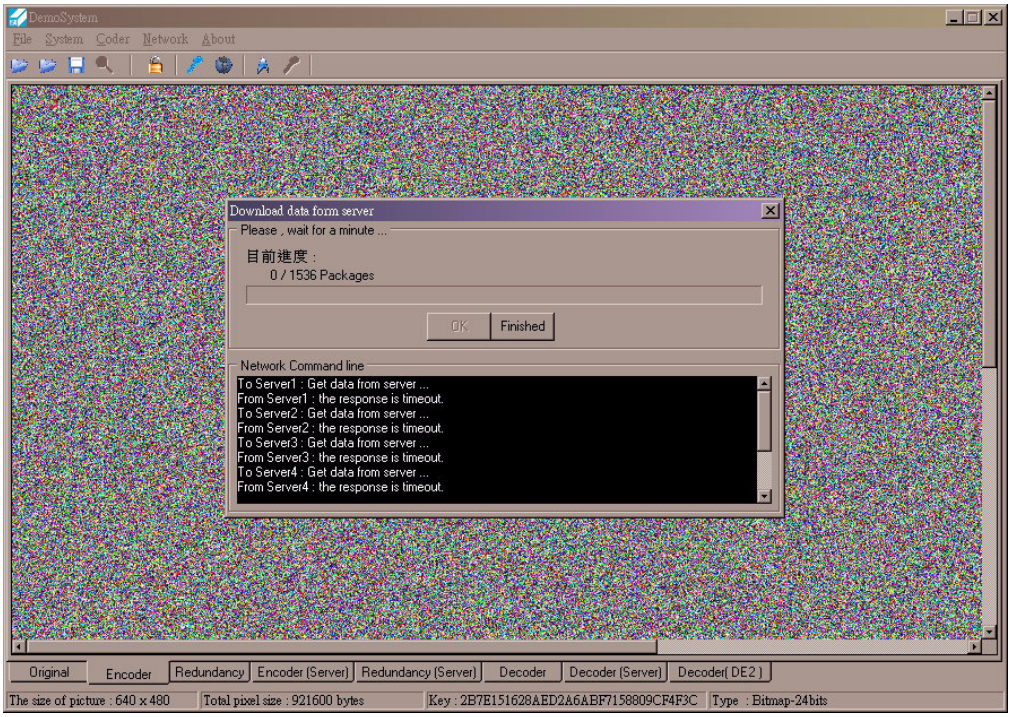
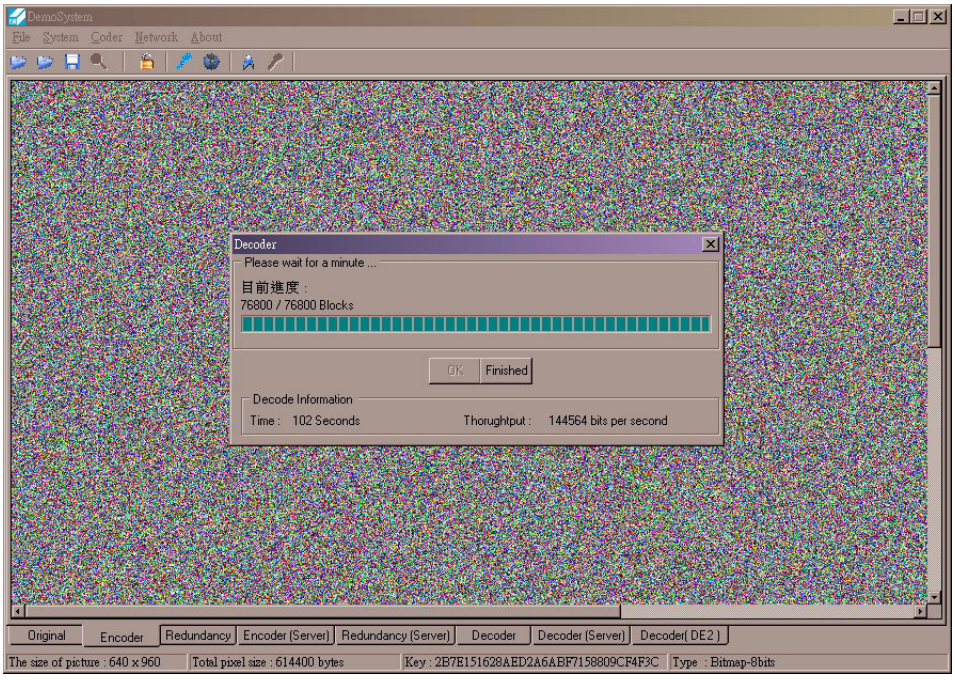


Figure 52. Data Decoding



Design Methodology

The process we used to create our design is described as follows:

- *Define the system*—Include processors, memory, peripheral components, and pins that connect the peripheral components.
- *Generate the system*—Use SOPC Builder to create a **.ptf** file, which is used by the Nios II IDE to generate the **system.h** file. This file contains all of the component information.
- *Create the hardware design*—Use VHDL code to write and build the required components. Combine, encode, decode, and simulate the circuit.
- *Create the software design*—Use the Nios II IDE to create relevant header files and drivers. Write applications, and encode and decode them to **.elf** files.
- *Simulate*—Use the ModelSim software to simulate. If problems occur, modify the system and redesign the software and hardware.
- *Verify*—Download the software and hardware through JTAG to the RAM on the Cyclone II development board and perform physical verification.
- *Test*—Test the system using the PC GUI to send test data.

SOPC Builder provides a platform to integrate software and hardware, and provides an environment for mutual development. Our design has three major parts: logic (intellectual property (IP) design), storage (RAM), and the computing core (CPU or digital signal processor). The design procedure is as follows:

- *Select the algorithms and IP functions*—We use Rijndael's AES algorithm and RS algorithm. The dynamic table generated after the addition of input variables is stored in on-chip RAM. Then we use the Nios II CPU to control the AES + RS encoding and decoding.
- *Select IP and custom IP components*—For our project, we used ready-made IP components; the Altera web site provides documentation that aids in designing the software and hardware. We also developmed custom IP components according to the project requirements. These components are attached to the Avalon bus.
- *Software/hardware design*—We used the hardware/software co-design for the project. Co-design is challenging because software development involves the planning and distribution of hardware resources and plays an important role in the performance of the entire system. SOPC Builder and the Nios II IDE provide an integrated hardware/software design system, which sped up the development process.

Design Features

Our design's features are as follows:

- *Create dynamic tables to accelerate operation*—AES encryption/decryption requires many finite field mathematical calculations. Replacing finite field mathematical calculations with LUTs can increase the design's speed.
- *Connect ten slaves to the Avalon bus*—Include data memory, instruction memory, CCD controller, SDRAM controller, UART, PIO, Ethernet, AES algorithm, and RS encoding operation components. Implement complicated IP integration and use the LEs in the Cyclone II device efficiently.

- *Accelerate instruction calculations*—RSA encoding/decoding uses many multiplication and modulus calculations. Compiling this function in custom instructions and adding them to the Nios II ALU improves the effectiveness of complicated mathematical and logic calculations.

- *Connect custom IP outside of the Nios II core*—The Nios II embedded processor can be modified easily, so users can design PIO pins for external communication according to their own needs. We also integrated the AES + RS hardware core to speed up encoding and decoding.

- *Provide high data security, data fault tolerance, and correction capability*—Academic circles have recognized the safety of AES encryption/decryption technology. A user can only open the encrypted text if he/she possesses the golden key. The golden key is further protected by RSA encryption. Data fault tolerance is implemented using the RS coding technology, i.e., normal services can be provided when one server host is inactive. We performed sophisticated calculations on the FPGA and improved efficiency by using the Nios II CPU to control the operation components, implement custom instructions, and support peripheral resources.

- *Provide a user-friendly demo program*—The user-friendly software GUI displays the flow of a whole set of AES + RS encoding/decoding, helping users quickly grasp the design concept.

- *Use storage space efficiently*—Traditionally, encrypted data (such as network-attached storage, storage area networks, etc.) is stored by copying data to the file servers. Data on each file server belongs to the same stock, so the space occupation is N x S (where N is the number of servers and S refers to the occupied space). When reclaiming data, data is taken from a close server.

  In our new network data security system, however, the storage encodes a stock of data with the RS code and disperses them on several file servers. The data is then drawn in an orderly fashion from the file servers. The occupied space is S because the same stock of data is dispersed to several file servers during storage. Additionally, the RS encoding technology enables normal service when one server host is inactive. In our experiments, tests can be facilitated using several I/O mechanisms, including 0/100 Ethernet, USB, and RS-232. We tested all of these connections using the DE2 development board.

Conclusion

Our design team benefited greatly from the Altera Nios II Design Contest 2006. Our work included system integration, hardware development, and software design:

- *System integration*—SOPC Builder and the Nios II IDE enable the flexible design and prototyping of soft CPUs. With this solution, the designer first develops network interfaces and then builds development and test platforms on PCs to accelerate the development process. This contest familiarized our design team with the importance of integration, providing faster development flow for consumer electronic products that have increasingly shorter life cycle, and reducing the costs of human resources and materials. We believe this flexible system design will become the mainstream in the near future. Our only problem was that the network performance of the DE2 development board was not good enough for our project and could not implement dynamic image processing.

- *Hardware development*—Hardware/software co-design is a hot topic. The costs and timing of developing hardware are higher than that of software development; therefore, making full use of hardware is critical. In this contest, we learned that using hardware to perform repeated operations improves system efficiency. However, this effect must be achieved by using the Nios II processor because it contains highly integrated interfaces, such as the Avalon bus, PIO, and custom instructions, which integrate the hardware and software interfaces. With this combination, a designer can easily implement hardware/software co-design.

- *Software design*—I feel honored to have participated in the Nios II design contest. I am responsible for the software interfaces, focusing on the communication and information exchange with the Nios II processor. We used Ethernet to do the work, and learned how to transmit

information. We use the C++ Builder to finish the software design and use it as a verifying tool. The software verifies the design's correctness and implementes the Nios II communication debugger to test each phase. During the design process, we learned Nios II internal communication and information exchange. Now that the contest is over, I feel that I know more about the Nios II processor hope to continue my learning process.