Third Prize

Nios II-Based Intellectual Property Camera Design

Institution: **Xidian University**

Participants: **Jinbao Yuan, Mingsong Chen, Yingzhao Shao**

Instructor: **Ren Aifeng**

Design Introduction

With the development of network technology, people have higher requirements for monitoring functions. By revolutionizing the traditional monitoring methodology, an intellectual property (IP) camera provides a good solution for remote real-time monitoring. With this technology, the user can check the safety, in real time, of the locations such as the home, office, etc. via a web site or video browser.

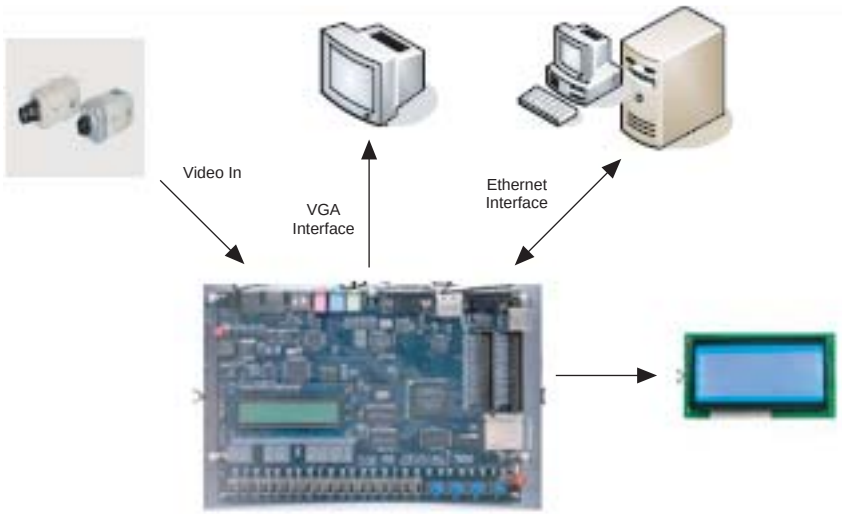
At present, most IP cameras on the market are implemented with hard-core microprocessors (MCUs) such as a special media processor. Our design proposes a network video transmission solution based on the Altera® Nios® II embedded soft-core CPU, implementing video data transmission via Ethernet. To reduce the data traffic and improve the video image's network transmission speed, the design uses an extraction algorithm and implements the video data transmission over the local area network (LAN) with the user datagram protocol (UDP). On the receiving side, it successfully displays data on the video terminal with the National Instruments LabWindows/CVI software. Altera's FPGAs are well known by engineers for their superb performance and configurability. The Nios II processor-embedded system features excellent flexibility, scalability, and upgrade options. Additionally, the Nios II soft-core processor embedded in a Cyclone® II FPGA is low cost and has performance up to 100 DMIPs, making it very competitive in the marketplace.

Hardware Platform

Using the Terasic Technologies, Inc Development and Education (DE2) board as the core hardware platform, our design implements video data collection, transmission, and remote display with an

National Television System Committee (NTSC) camera, liquid crystal display (LCD), etc. Figure 1 shows a block diagram of the system.

Figure 1. System Block Diagram



Software Platform

The following sections describe the software platform.

Embedded Development Software

We used Altera's Quartus® II version 7.0, SOPC Builder, the Nios II Integrated Development Environment (IDE), etc. to develop the FPGA design in hardware logic and embedded system software.

As Altera's fourth-generation programmable logic development tool, the Quartus II software provides the complete multi-platform development environment designers need. It integrates such development environments as system and programmable logic components design, combination, layout and threading, as well as verification and simulation. The overall development environment specially designed for a system-on-a-programmable-chip (SOPC) is the basis for SOPC design.

SOPC Builder is an SOPC development tool integrated into the Quartus II software that helps users create a complete system. Designers can use a variety of free components that are integrated in SOPC Builder or define their own peripherals or commands. As long as designers make configuration choices as required during the design process, they can build a system with reasonable resource usage. Furthermore, SOPC Builder automatically generates an in-chip bus structure, arbitration, and interrupt logic for each hardware component, and generates headers compliant with subscribed system features for successive software design (these headers define memory mapping, interrupt priority, and the data structure of each peripheral register space). When the hardware system changes, SOPC Builder automatically updates these headers and provides software designers with an automatic configuration interface, showing flexibility that ordinary hard-core processors cannot achieve.

The Nios II IDE is the basic development tool for the Nios II embedded processor: the user can complete all software development tasks, including editions, compilation, debugging, and program downloading, with the Nios II IDE. The Nios II IDE provides a uniform development platform for all Nios II processor systems. With only a PC, an Altera FPGA, and a JTAG-capable download cable, software developers can write data into the Nios II processor system and communicate with it.

Camera Display Development Software

The display is an integral part of an IP camera. Although various PC applications are available on the market, due to time limitations we chose a development platform that was easy to learn, easy to develop, and had the functions we needed.

In our project, we created an Ethernet terminal application using the LabWindows/CVI software, which is an interactive C language development platform. By combining the powerful, flexible C language platform with a professional measuring and control tool for data collection, analysis, and display, LabWindows/CVI utilizes its IDE, interactive programming method, function board, and rich function library to strengthen available C language functions. It provides an ideal software development environment for C language developers and designers to compile a detection system, automatic testing environment, data collection system, process monitoring system, etc.

The functionality of LabWindows/CVI lies in its rich function library, which not only has regular program design but also supports complex data collection and device control systems development. Additionally, the LabWindows/CVI user interface editor enables graphical user interface (GUI) creation and editing. The user interface library functions allow the design to create and control the GUI in the program. LabWindows/CVI offers a variety of professional controllers for designing GUI panels that help the designer build excellent user interfaces.

Function Description

This section describes the design functionality.

Video Collection

The NTSC camera and video decoder chip collect video images and convert them into digital video data that is compliant with the ITU-R656 standard. The data is sent to the FPGA for additional processing.

Video Compression

The resolution output from the video processing control module is 640 x 480; therefore, each data frame is 9,216,000 bit (640 x 480 x 30 bits), which is too much bandwidth. To solve this problem, we converted the resolution to 320 x 240 for lower data traffic, and take the five higher bits in the 10-bit RGB component signals output by the video decoding module as the lower 15 bits of the 16-bit data transmitted (add a 0 onto the highest bit).

Local Video VGA Display

Besides remote monitoring, this design is also applicable to short-distance monitoring using a long video cable. Without processing the data, the transmitted video is smoother and more clear when displayed at a shorter distance. Figure 2 shows an example of the local video display.

Figure 2. Local VGA Display Example

Ethernet Video Transmission

We implemented the remote monitoring function (which was the original purpose of our design) by adding an Ethernet control chip to the system with the help of a light-weight TCP/IP (LwIP) protocol stack. To ensure real-time operation and avoid frame loss, we chose the UDP communication protocol. Compared to the TCP protocol, the UDP protocol is more appropriate for multi-media data transmissions. However, to achieve this error-free control protocol, the design includes automatic packaging and automatic header additions in the transmitter's UDP error control module. The receiver performs the reverse function to provide a normal display.

Remote Display using the Ethernet Terminal

In this project, the computer's Ethernet terminal connects to the server (which is the DE2 board), receives and displays video data via Ethernet, and saves and plays back video fragments in real time. In the display's real-time window, the user can see the remote video transmitted via the network. The Save function allows the user to save the video data of interest (e.g., when an abnormality occurs during monitoring). When the user clicks Playback, the saved video plays. The Exit button logs out of the application. Figure 3 shows an example of the remote video display.

Figure 3. Real-Time Remote Video Display Example

IP Address Display

An LCD panel automatically displays the IP address allocated by the video collection server (the DE2 board) and the name of the system. Figure 4 shows the LCD panel display.

Figure 4. LCD Display

Performance Parameters

This section describes the design's performance parameters.

Video Parameters

The video parameters are:

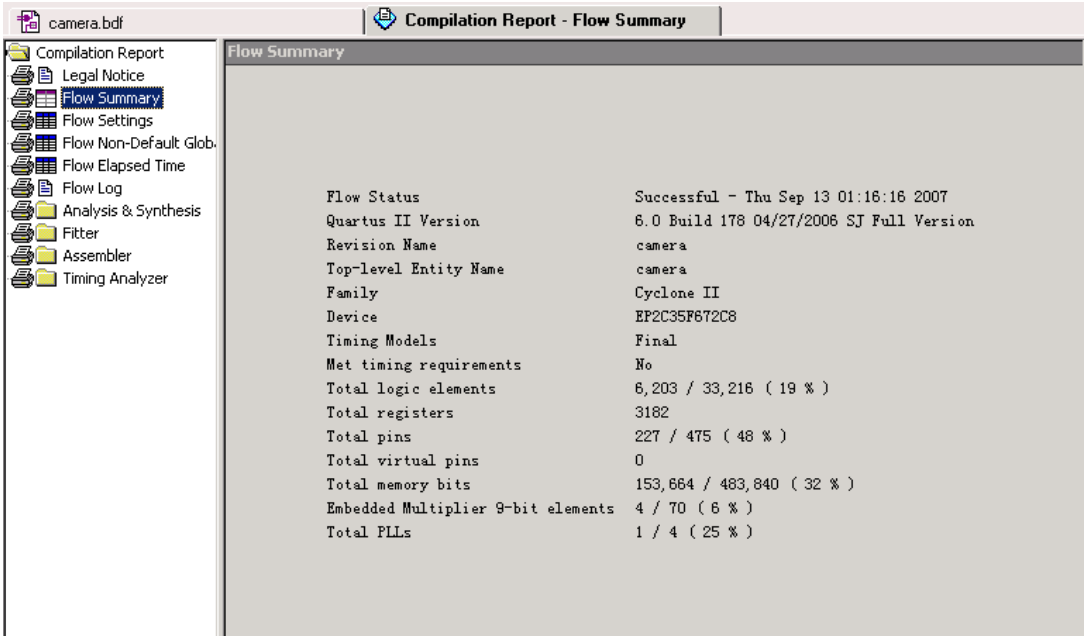
- Input video standard: NTSC
- Input video resolution: 768 x 494
- Output video standard: RGB

- Output video resolution: 320 x 240
- Data traffic: 320 x 240 x 15 bits/frame

FPGA Resource Utilization

Figure 5 shows the FPGA resources used in this design.

Figure 5. FPGA Resource Utilization



DE2 Development Board Resource Utilization

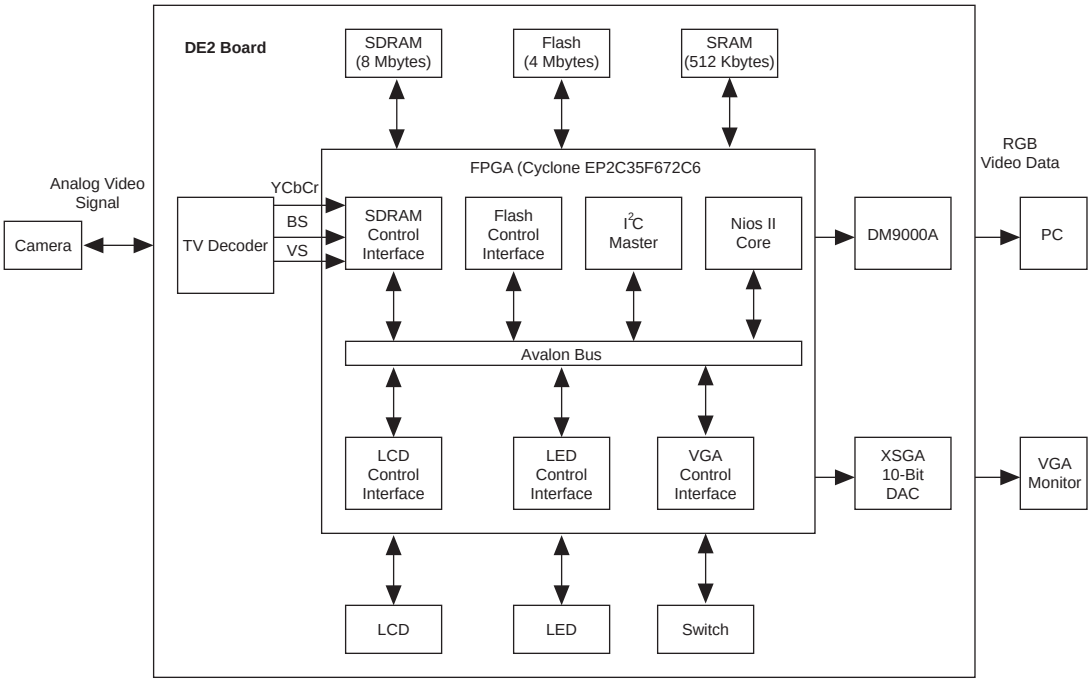
The design uses the following DE2 board resources:

- FPGA: Altera Cyclone II EP2C35 FPGA
- Dynamic memory: 8-Mbyte SDRAM
- Static memory: 512-Kbyte SRAM
- Flash memory: 4-Mbyte flash device
- 50-MHz, 27-MHz crystals, oscillators
- Switch, key, LED, LCD
- Video decoder chip: ADV7181B device
- 10-bit video digital-to-analog (D/A) converter: ADV7183 device
- XSGA video port

Design Architecture

The system design makes full use of the DE2 board's hardware resources. The whole system includes a camera, the DE2 board, a PC, and a monitor. The system provides video collection, video processing, and video transmission and display. Figure 6 shows the overall system structure.

Figure 6. Overall System Structure



The camera generates the video source for the system, and inputs analog video signals to the DE2 board via the video-in port. The DE2 board's ADV7181B video decoder chip decodes the analog video signal, converting it to a digital video signal that is compliant with the ITU-R656 standard. The system sends the signal to the FPGA's internal video decoding module to convert the YCbCr color-difference signal to an RGB tri-chromatic signal. The signal is then sent to the video compression module, which saves the compressed RGB three-way data into SRAM. Every time a frame is saved, the data transmission control module sends RGB data in the SRAM to the client for display via the DM9000A device. Controlled by a toggle switch, the user can display compressed or uncompressed RGB data on the VGA display.

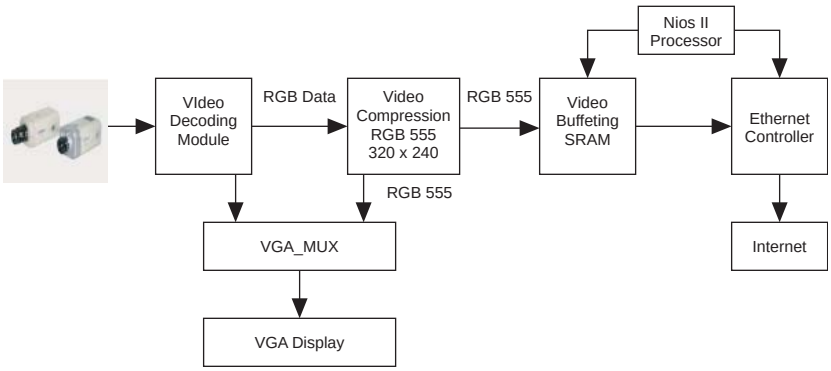
Design Methodology

This section describes our design methodology.

System Function Design

The system design includes a video decoding module, a video compression module, an SRAM bus switch module, and a video transmission module. Figure 7 shows the system diagram.

Figure 7. System Diagram



Embedded System Design

The embedded system design consists of hardware and software.

System Hardware Design

The following sections describe the modules in the hardware design.

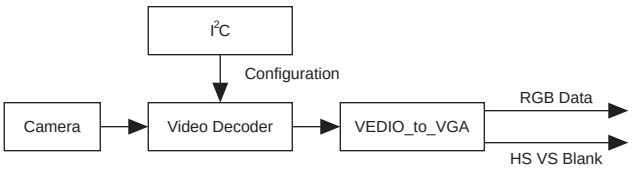
Video Decoding Module

The video decoding module has two functions:

- Configuring the video decoder chip ADV7181B.
- Converting video data from the video decoder chip to RGB format.

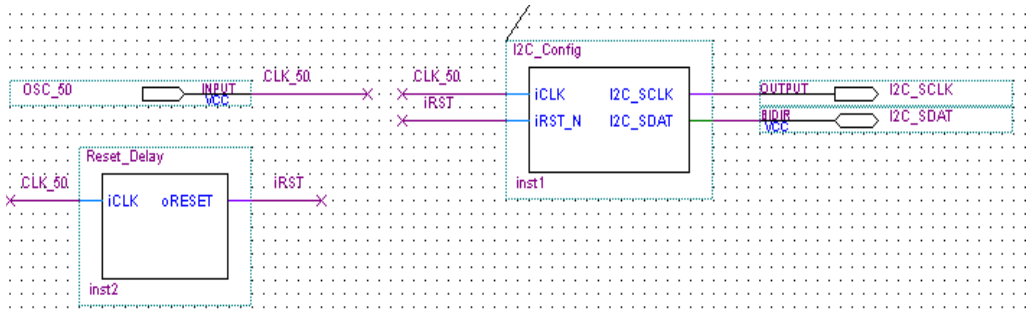
Figure 8 shows the video decoding process.

Figure 8. Video Decoding Schematic Diagram



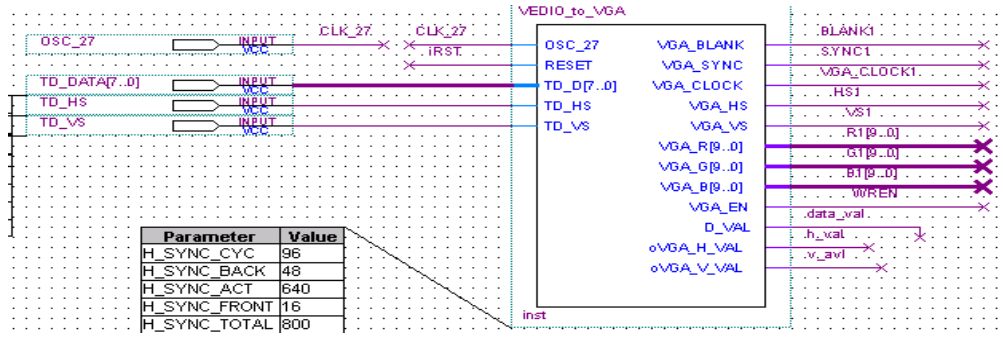
This system has an external NTSC (768 H x 494 V) camera connected to the DE2 board's video-in RCA interface. The design uses the I^2C bus to configure video decoder chip properly. Figure 9 shows the hardware implementation.

Figure 9. ADV7181B I²C Configuration Schematic



After configuration, the video decoder chip outputs YCBCR video data and control signals according to the ITU-R656 standard. Because the output data is for interlaced scanning, we designed the VEDIO_to_VGA module to process the data source interlacing and provide color space conversion. The module generates RGB video data, horizontal/vertical synchronization in line with the VGA sequence and blanking signal, etc. Figure 10 shows the hardware implementation.

Figure 10. Video Decoding Schematic



Video Compression Module

Because too much video data can slow the network transmission, we compressed the original video. We used a simple extraction algorithm to compress the video data: the design extracts every other line and one of every two points from the original video. Using the algorithm, the video data decreases to one fourth of the original. Because human eyes are insensitive to color signals, the design takes the five higher bits of the RGB components to compress the video data to an RGB555 video stream with 320 x 240 resolution. At the same time, the address adds 1 for each pixel point generated. Figures 11 and 12 show the video compression block diagram and hardware schematic.

Figure 11. Video Compression Module Block Diagram

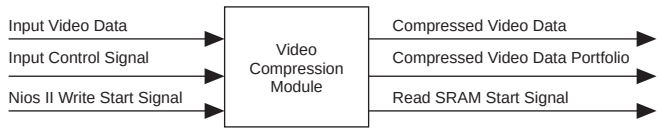
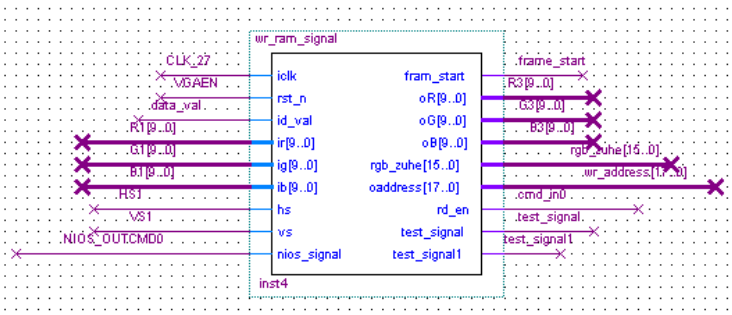


Figure 12. Video Compression Module Schematic Diagram



When the remote Ethernet terminal receives a video request, the Nios II processor sends the `NIOS_OUTCMD0` control signal to the hardware. When the module receives the command, it compresses the current image frame to generate the pixel to be extracted and the pixel's address in SRAM. When a frame is full, the read RAM start signal is sent to the Nios II processor.

SRAM Bus Switching Module

The image is obtained and stored using a hardware description language (HDL), and the Nios II processor controls the video's network transmission. The SRAM bus switching module sends the video data obtained by the hardware to the Nios II processor, and then the Nios II Ethernet controller sends it to the remote terminal. Figures 13 and 14 show the video compression block diagram and hardware schematic.

Figure 13. SRAM Bus Switching Module Block Diagram

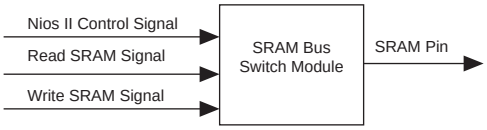
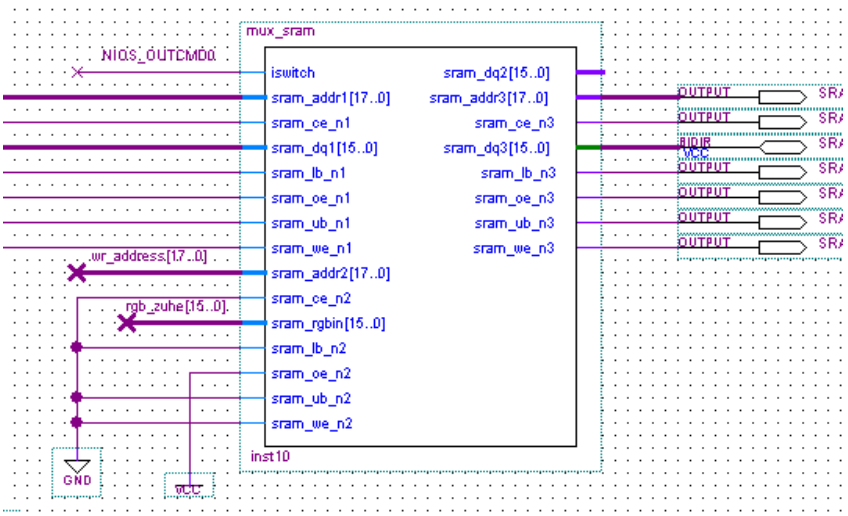


Figure 14. SRAM Bus Switching Module Schematic Diagram



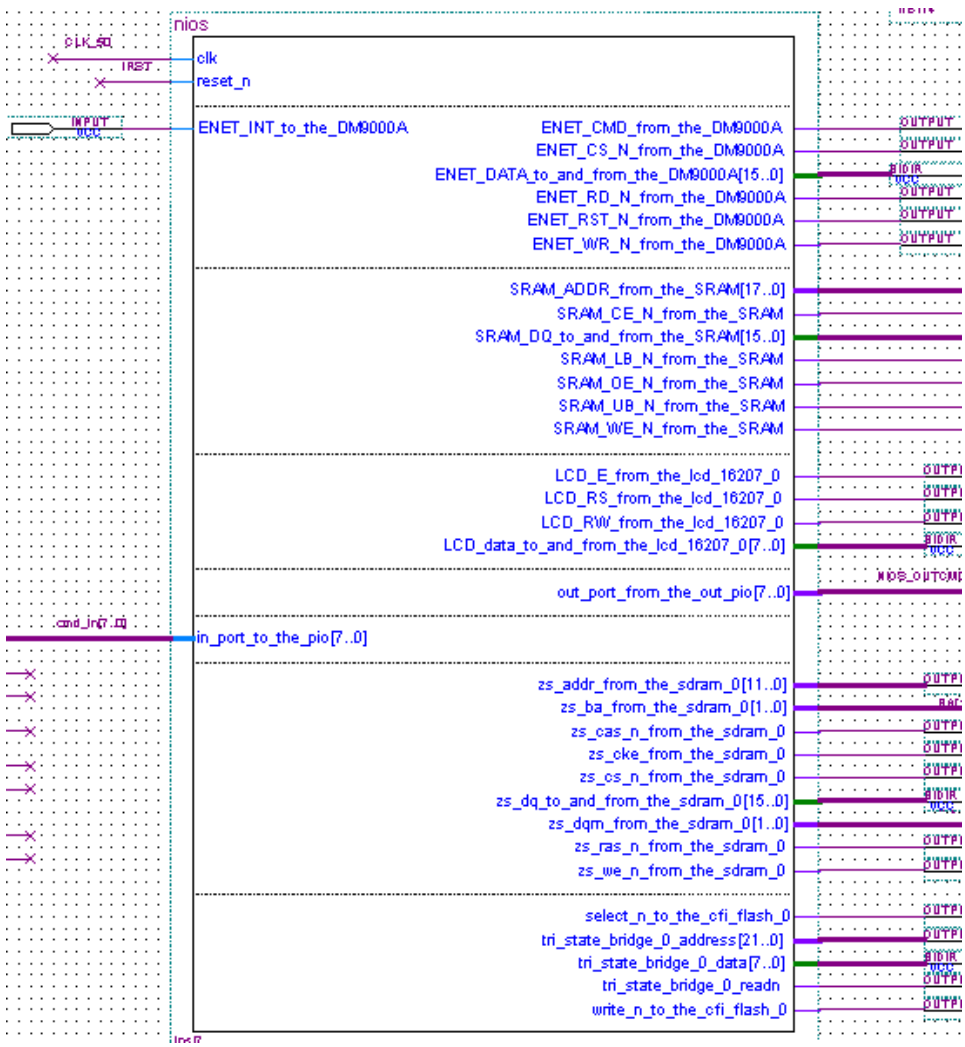
When the Nios II processor's `NIOS_OUTCMD0` control signal is high, the SRAM bus switching module writes the output video data from the video compression module into RAM. The hardware notifies the

Nios II processor to read using an interrupt. When it receives the read signal, the Nios II processor sends the NIOS_OUTCMD0 control signal as 0 and the SRAM bus switching module controls the SRAM bus to connect externally with the Avalon® bus.

Data Transmission Control Module

The data transmission control module generates the read/write control signals. When the client has a video request, the Nios II processor writes a video frame to the video buffer module's SRAM and waits for an external interrupt before it reads data in the buffer module and transfers it to the client. Figure 15 shows the hardware schematic diagram.

Figure 15. Data Transmission Control Module Schematic Diagram

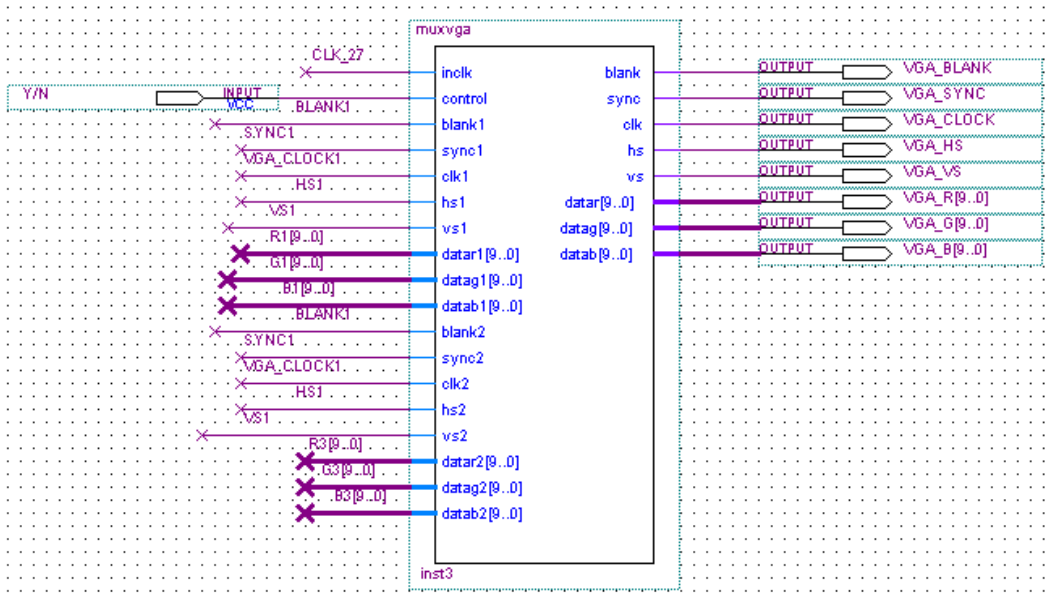


We implemented the data transmission control module in the Nios II soft-core processor. The processor conducts network initialization to acquire the MAC and IP addresses, receive the remote client's video request, control video compression and read/write video buffering, and transfer video data.

Local VGA Display Selection Module

The VGA display selection module controls whether the video displayed on the VGA monitor, i.e., it chooses whether the video is displayed compressed or uncompressed using a toggle switch. Figure 16 shows the hardware schematic diagram.

Figure 16. VGA Display Selection Module Schematic Diagram



System Software Design

The following sections describe the system's software design.

µC/OS-II OS

Due to the embedded system's limited resources and real-time video transmission requirements, we chose to use the µC/OS-II real-time operating system (RTOS). µC/OS-II is a portable, scalable, preemptive real-time, multi-tasking OS kernel. It distributes a separate stack for each task and provides multiple system services for interrupt management. It is easy to load µC/OS-II in the Nios II IDE provided the µC/OS-II option is selected when constructing the project. More important, the OS is loaded dynamically according to the hardware platform chosen by the designer in SOPC Builder. To develop a system, the designer modifies and adjusts the hardware platform to implement the best configuration. The Nios II IDE loads µC/OS-II automatically according to the modified hardware platform; therefore, designers do not need to worry about mismatches when changing the hardware platform.

LwIP Protocol Stack

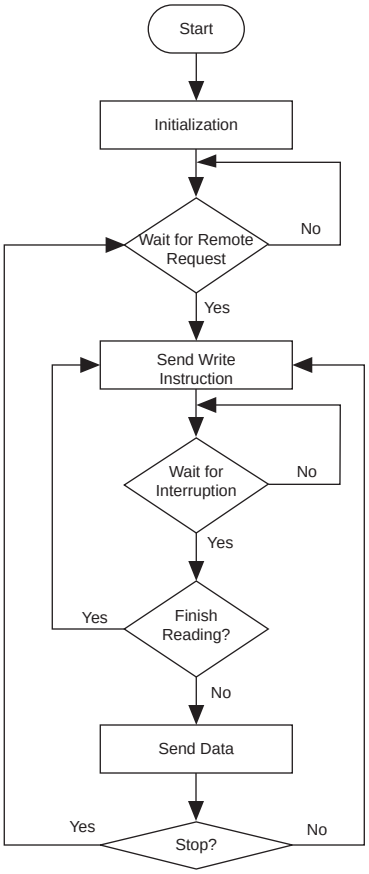
Our system requires network transmission, so it loads a network protocol. The LwIP protocol stack is integrated in the Nios II IDE, so we chose that protocol. Altera provides a Nios II port of LwIP, which enables a fast, open-source access to an Ethernet connection stack. Altera's LwIP port includes socket API encapsulation to provide a standard socket API with complete document description. After loading µC/OS-II, users can add the protocol stack by selecting the corresponding LwIP option.

System Software Process

The system software consists of three parts (see Figure 17):

- Initialize the LCD and acquire the MAC and IP addresses.
- Send control signals to the hardware and control the video data access.
- Wait for remote client requests and output video data to the video buffer.

Figure 17. System Software Process Flow Chart



Client Application Software Design

We developed the client video display application using the LabWindows/CVI software. The client communicates with the server using socket programming.

The client's key function is to receive and display a complete image frame. The client negotiates with the Nios II processor to regulate the frame's transmission time. The Nios II processor transfers RGB data in 5:5:5, but the client displays it in 24-bit bitmap format to enhance the effect. Therefore, the application uses an algorithm to convert the data from 16-bit RGB to 24-bit RGB image data.

A display function provides these operations by:

- Stipulating the number of UDP packages that a frame is divided into when programming the client application, so that we can easily know the image size and provide a timely display.

- Using an algorithm to convert from 16-bit RGB to 24-bit RGB image data. The code is as follows:

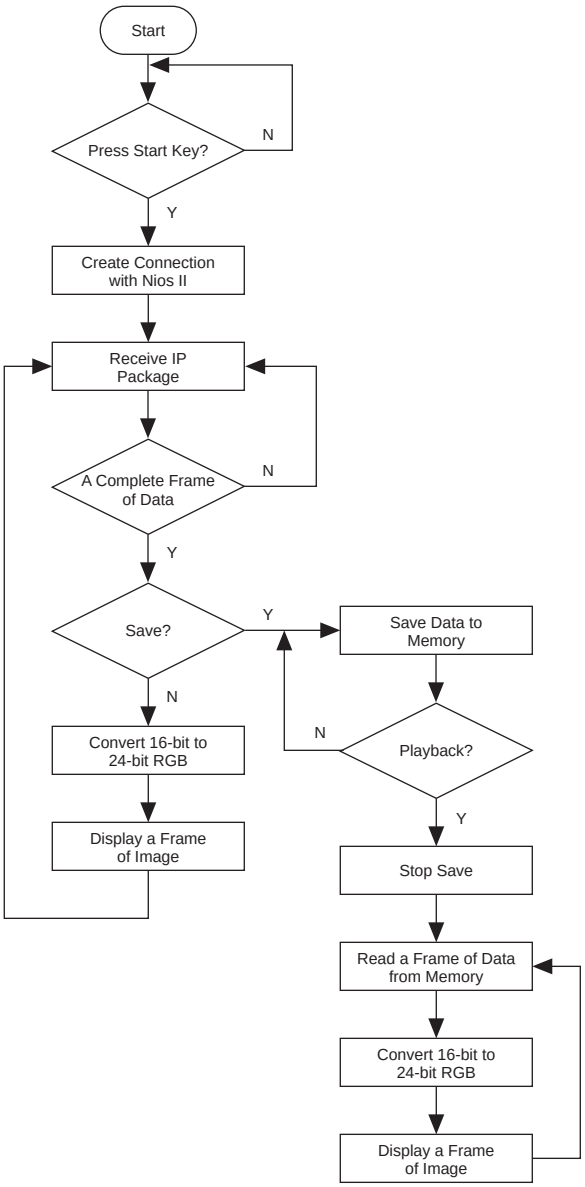
```
for(k = 0; k < 153600; k++)
 {
  rgb555 = (int)buff[k];
  rgb555 = rgb555<<8;
  rgb555 = rgb555|(int)buff[k + 1];
  k = k + 1;
  r = ((rgb555 >> 7) & 0xF8);
  g = ((rgb555 >> 2) & 0xF8);
  b = ((rgb555 << 3) & 0xF8);
  buff24[n++] = (unsigned char)r;
  buff24[n++] = (unsigned char)g;
  buff24[n++] = (unsigned char)b;
 }
```

The received `unsigned char` data (buff, and two as a unit), is put into the `unsigned int` data (`rgb555`). Then it shifts `rgb555` and conducts an 0xFF AND operation to extract three component data (r, g, and b), puts them into the `buff24` memory, and uses them for display.

For the save and playback functions, the problem lies in correctly selecting the data storage time in the program. Considering the image integrity, we set a mark in software to identify when the client application starts receiving a complete frame, and then save the data to ensure the integrity of the video data.

Figure 18 shows the client application flow chart.

Figure 18. Client Application Flow Chart



Design Features

Our design has the following features:

- *SOPC technology*—Making full use of SOPC features, the system uses an FPGA and embedded soft-core processor and uses the FPGA hardware to collect and analyze data for parallel processing. With SOPC Builder's custom peripheral feature, we added the DM9000A device to the system according to our requirements, enabling the Nios II processor to transmit data quickly. This method enhances system reliability and reduces power consumption.

- *Real-time data transmission by compressing video data with an extraction algorithm*—Before transmission, the system compresses the video data using an HDL module. With a simple compression algorithm, the overall performance of system is greatly enhanced, the visual effect is ensured, and the transmission efficiency is doubled. Thus, the system will better meet market demands.

- *SRAM bus switching technology*—In this system, we used SDRAM as the program buffer. Considering the capacity of the single-chip SRAM and SRAM on the development board, we used SRAM as the image buffer. Because SRAM cannot conduct dual-port operation, we used bus switching technology. When image data needs to be updated, the SRAM is connected to the data compression module. After a frame of video data is written into SRAM, the SRAM is connected to the Avalon bus and the Nios II processor reads the SRAM data for transmission.

- *Custom peripherals*—With custom peripherals, any hardware can be connected to the Avalon bus, and peripherals defined in SOPC Builder can be added into the system. For example, by adding custom peripherals (such as the DM9000A, I^2C bus interface, or SRAM interface) to the system to expand system performance, the Nios II processor strengthens its peripheral support.

- *Embedded µC/OS-II and LwIP protocol stack*—It is difficult for hard-core processors such as ARM processors to use µC/OS-II. In contrast, the Nios II IDE makes µC/OS-II and the LwIP protocol stack very easy to use. By integrating µC/OS-II and the LwIP protocol stack into the Nios II IDE, the system makes OS configuration and other operations part of a friendly GUI, which speeds software development. Considering the real-time requirements of video transmission as well as the convenience of adopting standard socket programming, we chose µC/OS-II OS. Because the LwIP protocol stack supports standard socket programming, software development times are further reduced.

- *Nios II technology*—Compared with inquiry methods, Nios II interrupt technology greatly improves the CPU's efficiency. In the Nios II processor, interrupt processing is easy to use. When an abnormality occurs, all functions except the interrupt service request (ISR) are performed by the hardware abstraction layer (HAL) system library code without operations required by the designer. Designers only need to compile the interrupt processing program in a specified format and register the interruption to the HAL during system initialization. In our system, the Nios processor can only execute operations when it receives the read RAM enable signal from the video processing control module. Because inquiry mode lowers CPU efficiency, we used interrupts to solve this problem.

- *Enhanced display of upper computer images*—Outputting video data through Ethernet transmission technology, the client application software can view, save, and play video. To enhance the display, we used an algorithm in the client application to convert 16-bit RGB data to 24-bit data.

Conclusion

This design gave us a better understanding of SOPC technology and the design process allowed us to learn, attempt, and innovate. During the short period of time that Altera hosted the SOPC embedded processor contest, we received an overview of SOPC solutions and the Nios II soft-core processor, and changed our viewpoint on FPGAs. SOPC design makes an FPGA single function, because all external control devices (e.g., single-chip) operations are integrated into the FPGA. The designer can add or remove Nios II peripherals and interfaces as required, facilitating the design process. Because the hardware does not have to be changed when the design changes, SOPC design provides a seamless interface between the processor and hardware logic, free from the problems of hardware threading and ensuring system stability. In SOPC design, the software and hardware are developed collaboratively, enabling synchronized FPGA logic development and Nios II soft-core program development in the same FPGA, which greatly increases the design efficiency.

We had a variety of problems to solve during our development process with the DE2 board. For example, we started using an inquiry method but found it to be inefficient. We then tried using

interrupts. As long as the status of programmable I/O (PIO) interfaces changes, the embedded system interrupts, improving the efficiency of the video data collection and the transmission speed.

Some design upgrades we would like to implement in the future are:

- Using a standard compression algorithm (e.g., H.263) to reduce the data bandwidth and implement access to the Internet. In this design, we planned to use a simple extraction method as our algorithm to compress video data. Due to time constraints and the complex HDL or C language required for the video data compression algorithm, we eventually abandoned the idea. The current design is based on a LAN, but a data compression algorithm is necessary when the system accesses a wide area network (WAN). By improving our HDL design, we could implement compression in the FPGA. Alternatively, we could design the algorithm using C and convert it into HDL using the C2H Compiler.

- Using the real-time transmission protocol (RTP) to improve the transmission speed.

During this contest, we learned the importance of collaboration. Together, we shared the experience of studying a technical problem overnight. In the final stages, although facing a lot of pressure, we were able to finish the project sucessfully. We thank our tutor Mr. Ren Aifeng who supported us during the project. Without his help, we would not have completed it. Although our design is imperfect, we gained valuable knowledge and friendship by participating in the contest. If possible, we plan to improve our design in the future.

Finally, we thank Altera for hosting the contest.