Nios II Embedded Processor Design Contest

Outstanding Designs 2007

Introduction

One thing hasn't changed since Altera created the world's first reprogrammable logic device more than 20 years ago: a commitment to innovation: ours and yours. We help you take your design ideas from concept to reality with a portfolio of products that includes the Nios® and Nios II soft-core embedded processors. Since they were introduced in June 2000, academic and professional designers worldwide have innovated using Altera's Nios and Nios II processors, integrating them in a wide range of commercial applications using high-performance Stratix® series FPGAs, low-cost Cyclone® series FPGAs, and HardCopy® series ASICs. Today, more than 20,000 development kits have been shipped, and over 5,000 companies—including the world's top 20 original equipment manufacturers (OEMs)—are licensed and implementing Nios processors in their designs.

The versatility of the Nios processor supports creativity and innovation. You can tailor Nios systems with the exact peripherals, memory, and interfaces required, and add your own proprietary functions—your own differentiation—to create a unique competitive advantage. Altera solves the IP integration problem with the SOPC Builder productivity tool that allows you to drag and drop the exact mix of functions required, so you can focus on higher, system-level requirements instead of mundane, error-prone, manual tasks. Plus, Altera® tools work seamlessly with other third-party industry-standard tools, minimizing training time and improving time to market. The soft-core processor implementation enables easy software and design upgrades, effectively making your design obsolescence-proof. With the added advantages of FPGA flexibility, fast time-to-market, and system integration, you have a risk-free path to a custom embedded solution. Your possibilities are unlimited.

Altera continually cultivates designers and supports innovation in Asia and around the world through our University Program. As part of that program, the Nios II Embedded Processor Design Contest aims to increase student interest in embedded processors, improve their design and creative abilities, and ultimately motivate the continued development of FPGA-based embedded processor designs. The significant growth of the program is yet more proof of how rapidly the Nios design community is expanding.

The winning entries presented in this book showcase not only the breadth of possibilities that can be addressed using Altera's embedded solutions—including everything from police vehicle support systems, medical equipment and telemedicine, robots, fingerprint identification, and a solar tracking control system to video processing and a cordless phone—but also technology trends in the industry. When you have the tools and flexibility you need, your potential for great design is unlimited.

Congratulations to all the Nios II Design Contest winners and their professors. Keep the fires of innovation burning!

Contents

Introduction iii
Contents v
Automotive Applications 1
Auto Audio Equalizer Using Digital Signal Analysis, *Third Prize* 3
Institution: Hanyang University
Participants: Sung-Wook Kim, Eun-Chan Kim, Bum-Su Jeong
Instructor: Professor Jae-Myoung Jeong
Consumer Applications 7
H.264 VBS-BMA-Based Hardware Infrastructure Implementation on an FPGA, *Second Prize* .9
Institution: Ching Yun University/ Department of Electronic Engineering
Participants: Wenxian Qian, Songzhi Gu
Instructor: Ou Qianmin
An Internet-Based Smart Terminal, *Third Prize* 21
Institution: Shanghai Jiao Tong University
Participants: Shanwen Zhang, Jie Zhang, Faheng Zang
Instructor: Zhigang Zhang
Multi-Functional Digital Albums Based on the Nios II Processor, *Third Prize* 37
Institution: Information Science Institute, Beijing Jiaotong University
Participants: Cheng Hong, Rui Deng, Yongxin Ye
Instructor: Xiaoming Ding
Communications Applications 57
SOPC-Based Cordless Phone, *First Prize* 59
Institution: National Institute of Technology, Tiruchy
Participants: Dhirendrakumar Tripathi, P. D. S. Prasad Reddy, Rashmi HM
Instructor: Dr. B. Venkataramani
Nios II-Based Intellectual Property Camera Design, *Third Prize* 87
Institution: Xidian University
Participants: Jinbao Yuan, Mingsong Chen, Yingzhao Shao
Instructor: Ren Aifeng

Industrial Applications..... 105

Police Vehicle Support System with Wireless Auto-Tracking Camera, *First Prize*.....107
Institution: Inha University, Korea Aerospace University, Hongik University
Participants: Sung Woong Joo, Ho Seong Suh, Young Je Moon
Instructor: Professor Tak Don Han
Smart Self-Controlled Vehicle for Motion Image Tracking, *First Prize* 125
Institution: Department of Information Engineering, I-Shou University
Participants: Chang-Che Wu, Shih-Hsin Chou, Chia-Hung Chao, Chia-Wei Hsu
Instructor: Dr. Ming-Haw Jing
RTOS Acceleration Using Instruction Set Customization, *Star Award*.....143
Institution: Centre for High Performance Embedded System (CHiPES), Nanyang Technological University (NTU)
Participants: Muhamed Fauzi Bin Abbas, Ku Wei Chiet
Instructor: Professor Thambipillai Srikanthan
Aerial Photographic System Using an Unmanned Aerial Vehicle, *Second Prize*.....157
Institution: Chungbuk National University
Participants: Hyuk Joong Kwon, Woo Joong Kim, Jang Geun Kim, Sang Bae Park
Instructor: Professor Jung-Kwan Seo
Laser Direct Writing Digital Servo Controller Based on SOPC Technology, *Second Prize*....173
Institution: Ultra-Precision Photoelectric Instrument Engineering Research Institute, Harbin Institute of Technology
Participants: Lei Yan, Tao Cheng, Ya Gao
Instructor: Wang Lei
Smart Bus Station Sign, *Third Prize* 195
Institution: Oriental Institute of Technology
Participants: Jian Jinrong, Zhan Yilin, Lin Taida
Instructor: Xiao Ruxuan
FPGA-Based Smart Induction Motor Controller Design, *Third Prize*.....205
Institution: Electrical Engineering Department, Yuan Ze University
Participants: Zhong Zhaoming, Lin Minghong, Chen Yilong
Instructor: Lin Zhimin
Intelligent Solar Tracking Control System Implemented on an FPGA, *Third Prize* 217
Institution: Institute of Electrical Engineering, Yuan Ze University
Participants: Zhang Xinhong, Wu Zongxian, Yu Zhengda
Instructor: Professor Huang Yingzhe
Nios II Processor-Based Fingerprint Identification System, *Third Prize*.....247
Institution: College of Communication Engineering, Chongqing University
Participants: Ji Wang, Liang Wu, Yong Liu
Instructor: He Wei
Fingerprint Identification System Based on the Nios II Processor, *Third Prize* 281
Institution: Huazhong University of Science and Technology
Participants: Linchuan Li, Yao Zhang, Chengdong Ge
Instructor: Xiao Kan

Medical Applications..... 297
MRI Spinal Segmentation Based on the Nios II Processor, *First Prize*.....299
Institution: Information Science Institute, College of Computer and Information Technology, Beijing Jiaotong University
Participants: Weiming Li, Ruiqiong Shi, Bo Li
Instructor: Xiaoming Ding
Nios II Processor-Based Self-Adaptive QRS Detection System, *Second Prize*.....319
Institution: Indian Institute of Technology, Kharagpur
Participants: Sai Prashanth, Prashant Agrawal
Instructor: Professor Agit Pal
Portable Telemedicine Monitoring Equipment, *Second Prize* 333
Institution: HuaQiao University
Participants: Huafeng Hong, Qianjiang, Yongjie Li
Instructor: Ling Chaodong
FPGA-Based Clinical Diagnostic System using Pipelined Architectures in the Nios II Soft-Core Processor, *Third Prize* 373
Institution: Jadavpur University, Calcutta
Participants: Shubhajit Roy Chowdhury
Instructor: Professor Hiranmay Saha
Appendix: Nios II Embedded Processor Family..... 391
Appendix: Nios II Embedded Processor Design Contest Winners . 405

Automotive Applications

This section includes projects that can be used as automotive applications, including:

Auto Audio Equalizer Using Digital Signal Analysis..... 3

Third Prize

Auto Audio Equalizer Using Digital Signal Analysis

Institution: **Hanyang University**

Participants: **Sung-Wook Kim, Eun-Chan Kim, Bum-Su Jeong**

Instructor: **Professor Jae-Myoung Jeong**

Design Introduction

Our project, an auto audio equalizer using digital signal analysis, is an equalizer that enables low-end speakers to deliver high-quality audio like that found in expensive speakers. Consumers can copy the characteristics of speakers with the desired audio quality and apply those characteristics to their own low-cost, low-audio-quality speakers. The structure and application of our project is simple, and it allows users of low-cost speakers to enjoy better sound quality.

Our project is an economical device that can also be applied to high-end monitoring equipment to achieve a flatter response without upgrading the existing system.

Function Description

The equalizer has three major functions:

- Measure and save the response of the speaker system.
- Analyze and compare the saved speaker's response with that of the current speaker system, process the input signal, and adjust the current speaker system's response characteristics to mimic those of the selected speaker system.
- Measure the applied system's response and evaluate the performance.

To execute these functions, we implemented the design in four parts:

- We implemented the fast Fourier transform (FFT)/inverse FFT (IFFT) block's input signals using functions from the MegaCore® library.
- We configured the FFT/IFFT hardware as an equalizer by connecting the coefficients of each frequency ranges to the registers of a Nios® II processor.
- We included an SDRAM controller, flash controller, user logic FIFO buffer, and audio CODEC interface logic using SOPC Builder. We ported µClinux for the system that uses the Nios II processor.
- We included PC monitoring software that has a graphical user interface (GUI) that displays and saves the stored speaker's response and the measured response. The response can also be downloaded to the Cyclone® device on the development and education (DE1) board.Communication between the device and the PC uses the DE1 board's RS-232 interface.

We chose µClinux as the system's operating system because many programmers are familiar with the Linux environment, which helps shorten the development period, and there are plenty of available development resources. Additionally, µClinux dramatically reduces the need for low-level coding such as interrupt handling without overloading the CPU.

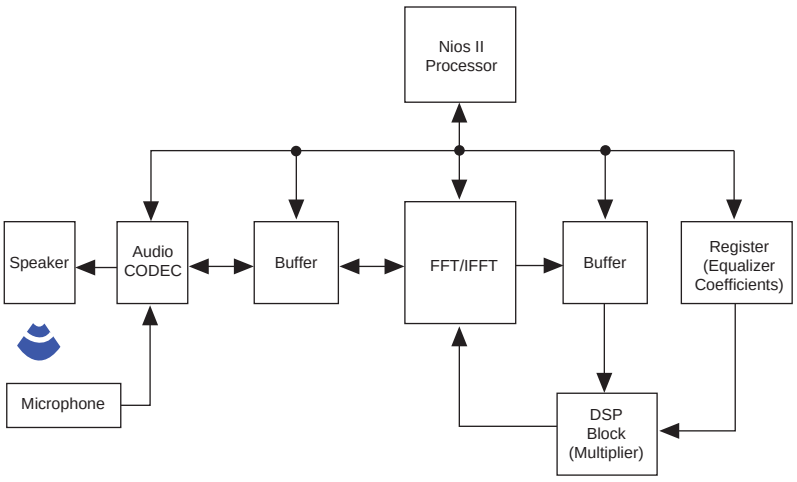
Performance Parameters

The equalizer applies the response of an already measured separate speaker to the current speaker system and then measures the current system's response a second time. The system numerically displays the correlation between the desired response and the improved response. Users can monitor how close the responses are using a PC-based GUI.

Design Architecture

Because the contest required us to use the DE1 development board, the system has an abbreviated structure that only implements the core functions. Figure 1 shows the system's block diagram. The digital signal processing (DSP) block performs most of the signal processing. The Nios II processor implements the module control and monitoring. We could expand the tasks performed by the Nios II processor later using the C-to-Hardware Acceleration (C2H) Compiler or user-defined custom instructions.

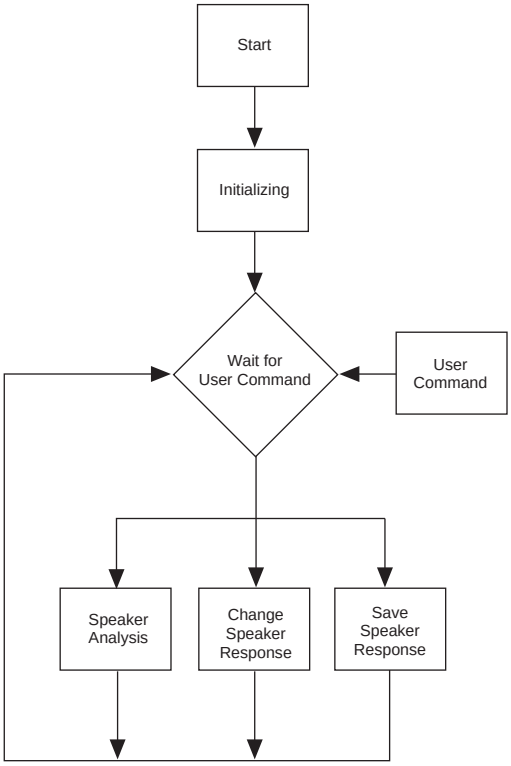
Figure 1. System Block Diagram



We inserted a digital signal analyzer (DSA) to measure the speaker's response. The microphone is not speaker dependent; therefore, once the frequency characteristics are met, other response characteristics are not critical.

Figure 2 shows the software flow chart.

Figure 2. Software Flow Chart



The system performs three key functions as instructed by the user:

- *Speaker analysis*—The system measures the response of a speaker that is currently connected to the system. After measurement, it graphically display the result so that the user can decide whether to save it.
- *Save speaker response*—The system can store the measured speaker response in flash memory; later, the system can change a speaker's response based on this information.
- *Adjust speaker response*—If the user selects the response of the speaker currently connected to the system and the desired speaker response, the system can change the current speaker's response through DSP.

Equalizers currently available in the marketplace can perform similar functions, but the user must manually adjust each variable. In the future, we would like to add a feature that imports calculated variable values (like those used in commercial equalizers) to the PC. This feature would allow users to use Winamp or Windows Media Player without a commercial system.

Design Description

The most difficult part of designing the hardware is debugging and verifying the system operation at every step. We only added an unverified block after the rest of the system had been fully verified. We also verified the intellectual property (IP) features using simulation. This process helped us avoid rechecking existing parts of the system while debugging and increased our development efficiency.

Fortunately, the Altera® Quartus® II software, SOPC Builder, and Nios II Integrated Development Environment (IDE) allowed us to use already verified IP functions and design methodologies, significantly reducing the debugging burden. For example:

- Because we used the FFT-IFFT block without modifications, we only needed to perform a simple data format check and timing simulation.
- We implemented the Nios II processor, memory controller, FIFO interface, and audio CODEC interface using SOPC Builder. These blocks are basic features of SOPC Builder; therefore, we easily designed and tested the firmware using the Nios II IDE. Test programs are readily available at communities such as the Nios II forum, speeding up the entire process.
- The JotSpot Wiki (nioswiki) provides a good explanation of how to port µClinux. We just needed to follow the instructions to implement all of the IP blocks with SOPC Builder.
- We used Windows-based programming to design the PC monitoring program. After we developed the monitoring program, we simulated the monitored signals using the MATLAB software and other tools that manipulate data easily. Then, we ported the compiled code to the DE1 board and tested the hardware.

Design Features

Our design has the following features:

- *Creative*—Using DSP, the design makes a low-end speaker produce high-end speaker sound.
- *Inexpensive*—All functions except the audio CODEC and microphone are implemented in an Altera FPGA, reducing the bill of materials (BOM). Also, the system performance is not affected by using a cheap microphone, further reducing the total system cost.
- *Easy and prompt development*—The Nios II processor and user-defined logic (buffer, FFT/IFFT block, register, and audio CODEC) connect directly to the Avalon® bus, which controls them. Therefore, we did not need to provide controls using a general-purpose I/O (GPIO). In the future, we will not need to design the DSP block separately; instead we can implement it with the Nios II processor using user-defined custom instructions.

Conclusion

The Cyclone FPGA on the DE1 board was sufficient to implement a 2,048-sample FFT/IFFT with a 16-bit, 44.1-kHz sampling rate, the Nios II processor, and other IP functions. However, our design only measured and processed the magnitude as a speaker's response, ignoring phase delay and other distortions, which limits the system performance. In the future, improving the signal processing will provide better performance.

We often perform projects using the DE2 board, mainly for fast prototyping, because Altera's FPGA solution provides proven tools and IP blocks, enabling us to implement systems quickly. Additionally, active Internet communities provide quicker, higher quality feedback than local distributors. We ask Altera to continue events like the Nios II design contest to support and encourage these communities.

Consumer Applications

This section includes projects that can be used as consumer applications, including:

H.264 VBS-BMA-Based Hardware Infrastructure Implementation on an FPGA 9

An Internet-Based Smart Terminal..... 21

Multi-Functional Digital Albums Based on the Nios II Processor 37

Second Prize

H.264 VBS-BMA-Based Hardware Infrastructure Implementation on an FPGA

Institution: **Ching Yun University/ Department of Electronic Engineering**

Participants: **Wenxian Qian, Songzhi Gu**

Instructor: **Ou Qianmin**

Design Introduction

The block matching method, which is used for motion estimation, plays a key role in motion picture coding systems. Replacing the fixed-block-size block matching algorithm (FBS-BMA) with the H.264 variable-block-size block matching algorithm (VBS-BMA) addresses the issue that video object changes cannot be processed effectively, further improving video compression efficiency. Our design reduces the complex H.264 VBS-BMA calculation and features low latency, low power, and high throughput, delivering better coding performance.

H.264 VBS-BMA supports 4 x 4, 4 x 8, 8 x 4, 8 x 8, 8 x 16, 16 x 8, and 16 x 16 blocks. When serial frame data is transmitted via a network, the user can choose the most appropriate block for matching according to the current bandwidth, thereby obtaining the best transmission speed and frame quality. However, the H.264 VBS-BMA calculation is more complex. Therefore, we designed an efficient very-large-scale integration (VLSI) hardware structure that features high computing throughput to cut the complex calculation time required by H.264 video coding, reduce the calculation frequency, and improve coding performance. Our design uses the following elements:

- *Hardware*—Because the design has numerous complicated calculations for compression, we needed an effective system to process video frames in real time. To process serial frame data of a specific size and frequency effectively, we used Altera's Nios® II development kit. The Nios II processor is a soft-core processor based on the RISC architecture. It synthesizes a processor circuit (but not a hard core) on FPGA fabric, permitting scalable development. The Nios II processor can serve as the hierarchy setting for memory, and add processor instructions

independently to perform special calculations. While using Altera's development board and FPGA to develop hardware, we used the Quartus® II software and SOPC Builder, which is integrated in the Quartus II software, to develop the test platform. Besides adding our block matching circuit to the test platform via SOPC Builder, we also used Ethernet chip-embedded control hardware and components, and then transmitted them to the development board for rapid prototyping.

- *Software*—We used the Nios II Integrated Development Environment (IDE) to write and compile test software, implement network transmission between PCs, and migrate µC/OS-II into the hardware structure of the Nios II processor system.

- *PC*—We designed a graphical interface program that can be executed in the Microsoft Windows operating system (OS). The PC is connected to the test platform through the network, transmitting motion frames and vectors, and accomplishing calculation, estimation, compensation, and decoding. It displays real-time estimation frames on the PC screen to facilitate users' viewing.

Function Description

Users load serial frame data through the graphical user interface (GUI) as shown in Figure 1. Next, the test platform is connected as the the connection status shows (see Figure 2). After the frame is sent to the test platform via the network, the system performs the motion vector calculation, motion estimation, and motion compensation. Then, the estimation frame is sent back to the PC via the network and displayed on the monitor (see Figure 3).

Figure 1. User Loads Serial Frame Data on PC

Figure 2. PC Connects to the Test Platform

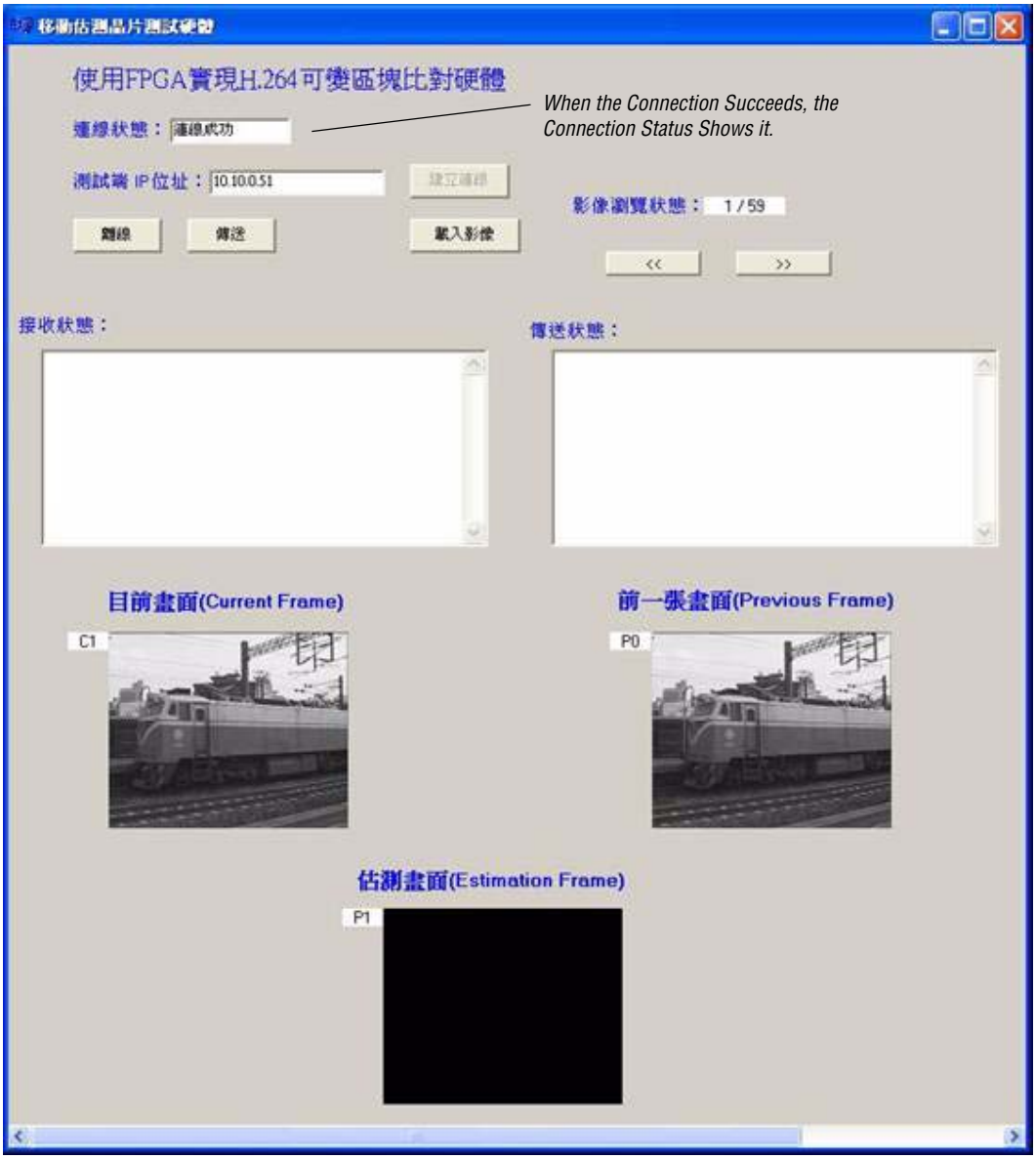


Figure 3. Test Platform Returns Estimation Frames after Processing

Performance Parameters

Our design has the following performance parameters.

- The design targets the Cyclone® EP1C20F400C7 device, which has 20,060 logic elements (LEs), 294,912 memory bits, and 301 pins. The design uses 8,881 (44%) LEs, 110,352 (34%) memory bits, and 176 (58%) pins.

- The system adds some necessary interface modules in the Nios II processor to improve system integration. Because most of the peripherals have related intellectual property (IP) cores that are well designed and tested, they can be used to accelerate hardware validation and software

development. We used many Altera IP cores, including SDRAM, Ethernet PHY chip, UART, SRAM, and flash memory. Additionally, the Nios II processor's 32-bit processing significantly enhanced the image processing efficiency and networking, and improved serviceability.

- The Nios II processor serves the design's control and algorithm cores. Because many IP cores and user IP cores can be added as peripherals, the system is more flexible.
- We enabled the motion estimation test function. After testing, we could estimate the frame result in real time.
- Currently, the device implements 4 x 4 block matching. In the future we can add other blocks using the same principles.

Design Architecture

Figure 4 shows the design concept of the H.264 VBS-BMA FPGA hardware structure. Users can load serial frame data through the GUI, transfer it to the test platform using Ethernet, test the H.264 block matching circuit, send the data back via the network, and (after test platform computation) display the frames on the PC interface for real-time viewing.

Figure 4. H.264 VBS-BMA System Design Concept

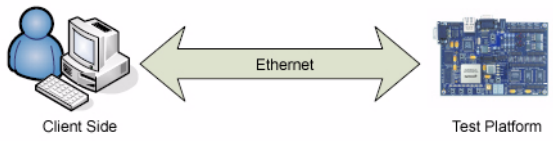


Figure 5 shows the PC operation. The current frame of the loaded serial frame data is transmitted to the Ethernet with a TCP socket. The PC receives motion vectors that are sent back from the test platform and decodes them to obtain the new estimation frame.

Figure 5. PC Operation

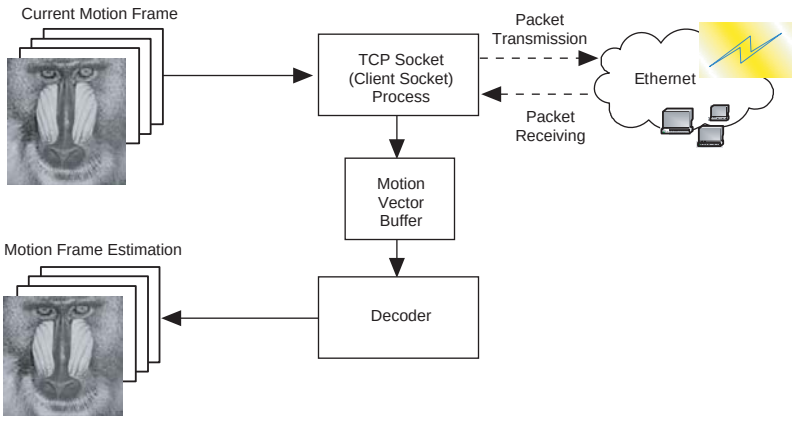
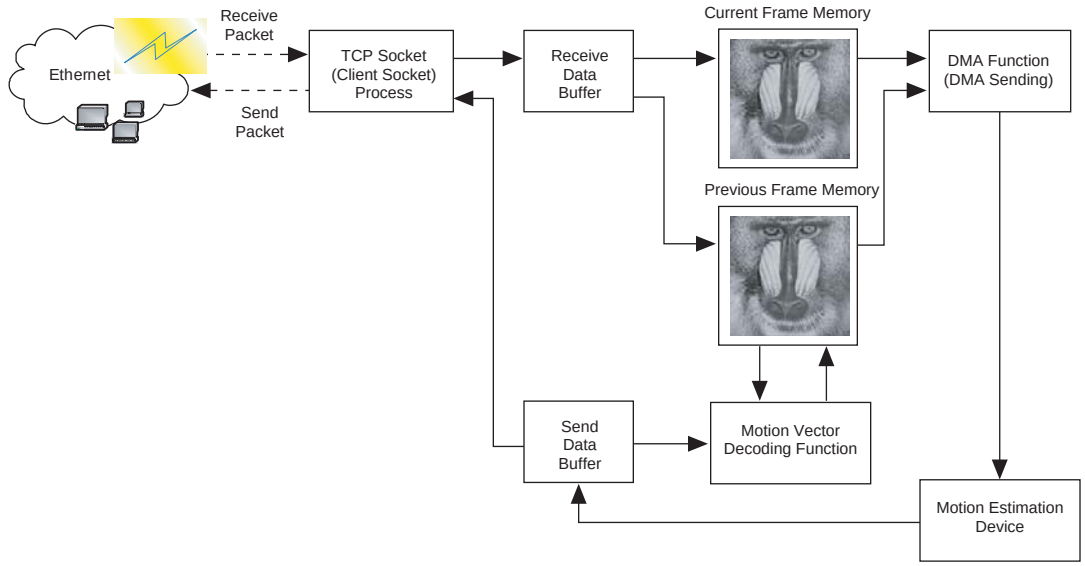


Figure 6 shows the test platform operation. First, the TCP socket receives the frame data package that the PC transmits from the Ethernet and caches it in the Receive Data Buffer. The first frame received is stored in the Previous Frame Memory as the previous frame, and the last frame is stored in the Current Frame Memory as the current frame. After the current frame is received, the current and previous frames are transferred to the block matching circuit for motion estimation and motion vector calculation. The DMA device accelerates data transmission from the memory to the block matching

circuit. Motion vectors obtained from the calculation are stored in the Send Data Buffer and are sent back to the PC via a TCP socket. During transmission to the PC, motion vectors are decoded to rebuild the estimation frame using the original previous frame. The new estimation frame is then put into the Previous Frame Memory to replace the original previous frame and become the previous frame for the next calculation.

Figure 6. Test Platform Operation



As shown in Figure 7, after the test platform is connected to the PC, it receives frame data (Cx) over the network from the PC. The first frame (C0) is transmitted directly to the Previous Frame Buffer for initialization; all other frames are transmitted to the Current Frame Buffer. After the current frame is received, the current and previous frames are transmitted to the motion estimation device for motion vector calculation. After the test platform conducts the motion compensation on the motion vector and previous frame, the generated estimation frame (Px) is transmitted to the Previous Frame Buffer to serve as data for the next motion estimation. Simultaneously, motion vectors are sent back to the PC via the network. Px and Px-1 (the previous frame) are stored in the PC. The system then determines whether data transmission is finished; if it is not, it continues to transmit Cx and repeats the whole process until it finishes.

Figure 7. System Flow Chart

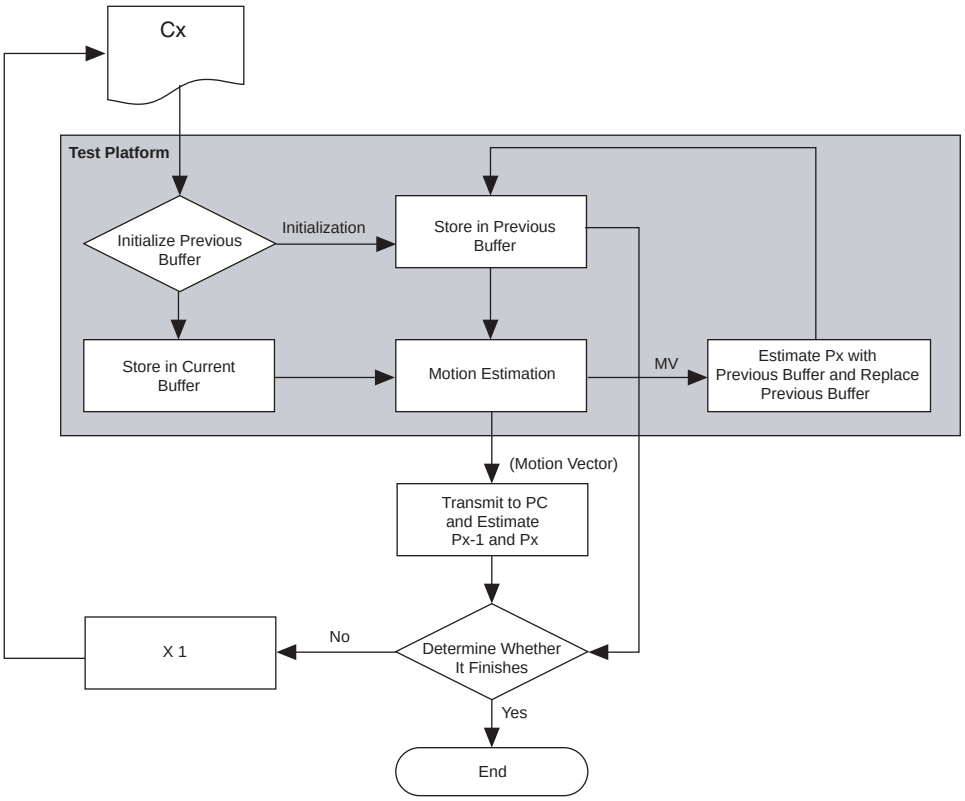


Figure 8 shows the test platform's hardware structure. The Nios II processor creates the hardware structure of the matching circuit. We used the Avalon® bus to connect peripherals like traditional peripheral models; for example, an LCD shows the test platform's IP address, an Ethernet MAC/PHY transmits and receives data packets, and SDRAM stores the previous and current frames. We used a direct memory access (DMA) model for effective data transmission between the peripherals and memory.

Figure 8. Test Platform Hardware Structure

Adapted from the Nios II Processor Reference Handbook

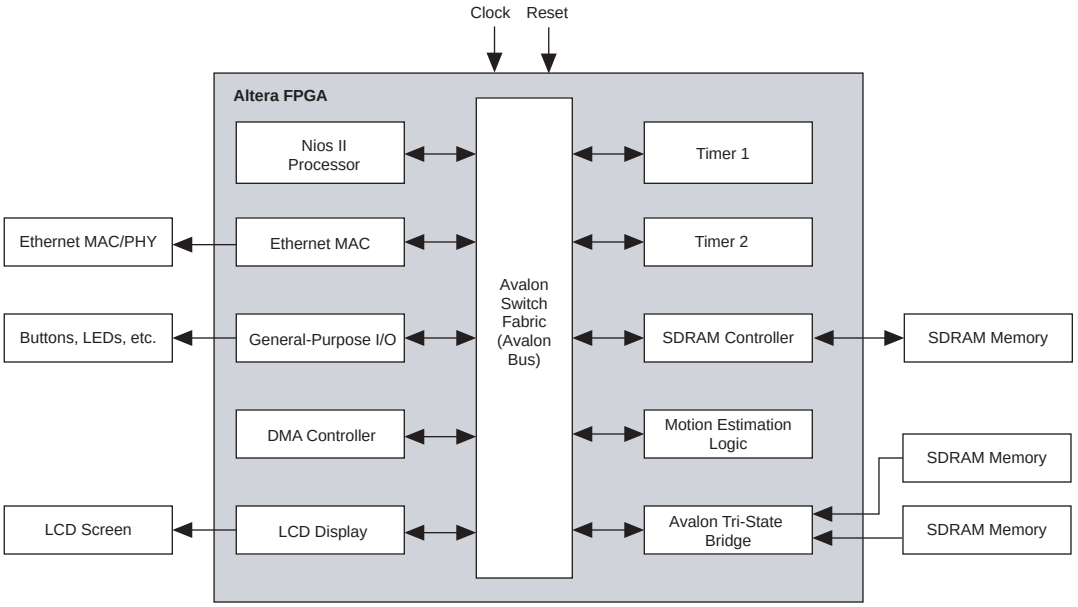


Figure 9 shows the test platform's software structure. It provides support from the inside to the outside. The first internal layer is the matching circuit's hardware structure, which supports the whole software structure implementation. The second layer is the hardware driver, which is generated automatically during the creation of the hardware structure and allows components in the outside layer to use hardware in the first layer. The third layer is Altera's hardware abstraction layer (HAL) library that enables components in the outside layer to use hardware. The other two layers are for the embedded operating system (OS) and protocol stack. The final layer is our application.

Figure 9. Test Platform Software Structure

Adapted from Using Lightweight IP with the Nios II Processor Tutorial

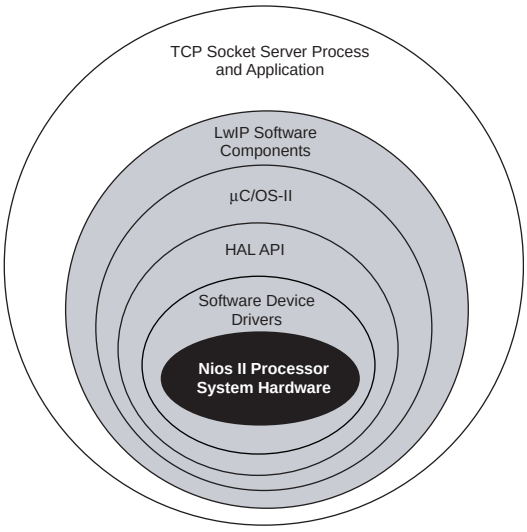
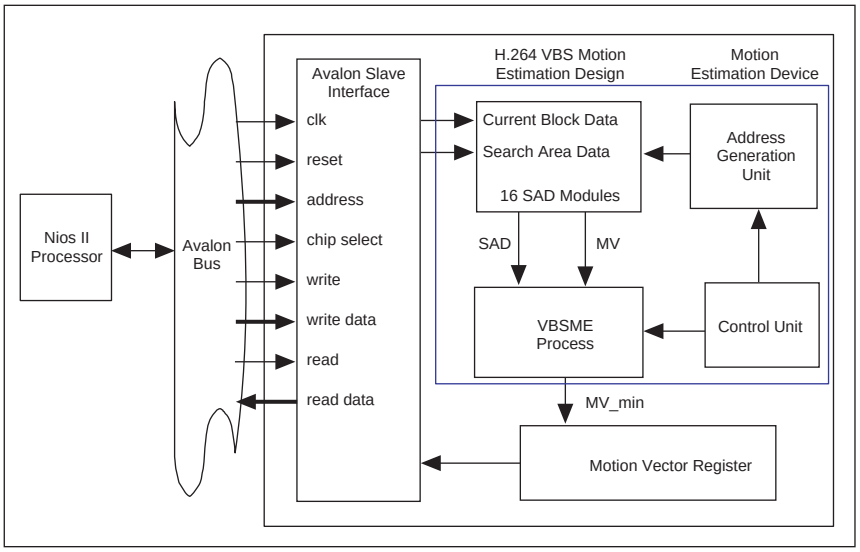


Figure 10 shows the system structure of the motion estimation device. The Avalon bus sends frames to the matching circuit. After the calculation, the Avalon bus reads motion vectors from the circuit's motion vector register and sends them to the PC.

Figure 10. Motion Estimation Device System Structure



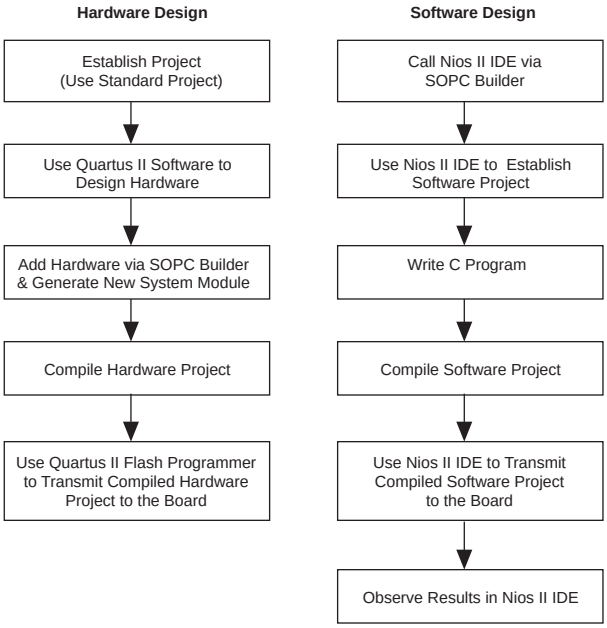
Our motion estimation architecture contains 16 sum of absolute differences (SAD) modules, a VBSME processor, an address generation unit (AGU), and a control unit. These modules operate as described below:

- Each of the 16 SAD modules handles a different SAD calculation based on the sub-block and its search area.
- The VBSME processor sums the SAD calculation from 16 basic blocks generated by the 16 SAD modules, while composing a SAD of different-sized sub-blocks and primary blocks and searching for the best motion vector.
- The AGU controls memory data reading and writing.
- The control unit controls coordination between the other modules.

Design Methodology

Figure 11 shows the hardware and software design flow charts.

Figure 11. Hardware and Software Design Flow Charts



Hardware Design

As shown in Figure 11, we used a standard project based on Nios II examples and wrote the hardware program using the concept of layered modules. After compilation, we opened SOPC Builder, integrated custom user IP, and added a DMA device to enhance the system performance. After we built the hardware structure, SOPC Builder automatically organized the hardware devices and generated the related drivers and files needed for software development. We transmitted the compiled test platform to the development board using the Quartus II software, using the FPGA combined with other hardware required by the test platform to perform rapid prototyping.

Next, we used the Nios II IDE to test the software code and compile the test software in the test platform. The Nios II IDE contains µC/OS-II (a real-time OS) and a protocol stack (light-weight IP), and a small TCP/IP protocol used in the embedded system, which helped us implement network communication between the TCP server socket and the PC. µC/OS-II provides quick responses, we migrated it to the hardware structure of the Nios II processor.

Software Design

As part of the software design, we modified the Altera-provided simple socket server IP. We set the IP address, subnet mask and port name, modified the read, receive, write, and transmit program, set the system library's send and receive buffer sizes, and downloaded the compiled project to the test platform.

To allow users to test the platform, we wrote a graphical interface program executed in Microsoft Windows using the Borland C++ Builder software version 6.0. Because we used TCP/IP, we used the client TCP socket to connect the PC with the test platform. Then, we were able to transmit motion frames and vectors, load serial frame data in the PC to test the block matching circuit, and send the estimation images generated by the test platform back to the PC for the user to view in real time.

Design Features

Our design has the following features:

- The H.264 block matching circuit uses Altera's DMA device to manage the transmission between the block matching circuit and SDRAM, i.e., to provide effective transmission without using the Nios II processor.
- We used SOPC Builder to add Ethernet chip control hardware and automatically generate the related drivers required for the system's network functions.
- The user IP is customized and integrated in SOPC Builder to perform complex computation. Additionally, the layered modular hardware circuit design reduces the complexity and allows users to customize the design.

Conclusion

In an era of software, designers seem to pay more and more attention to the study and development of software. However, our design shows that the H.264 VBS-BMA algorithm used for software can also be implemented in hardware to provide better performance. Additionally, Altera's simple socket server IP allows users to observe the estimation frame in real time without processing complex pins with a logic analyzer and signal generator and writing a complex testbench.

The Nios II processor allows users to create various devices—such as SDRAM, an Ethernet PHY chip, UART, SRAM, and flash memory—and embed them in the processor. The user can develop these modules directly with SOPC Builder, reducing the peripheral hardware design's complexity and development time.

In addition to gaining a deeper understanding of the Nios II processor during the design contest, we learned how to solve problems and achieve peace of mind, which is the most important asset that cannot be obtained from classroom teaching.

Third Prize

An Internet-Based Smart Terminal

Institution: **Shanghai Jiao Tong University**

Participants: **Shanwen Zhang, Jie Zhang, Faheng Zang**

Instructor: **Zhigang Zhang**

Design Introduction

Households, schools, and administrative organizations are using more and more electrical equipment. With the increase in additional equipment, management becomes an issue. Because the equipment is usually maintained and managed in wired mode, extra investment will be needed specifically for equipment management. Our project considers using wireless mode to organize and manage equipment and centrally control it using the Internet. Internet access has become a standard interface for families and offices. Therefore, with a wireless control terminal and a cost-efficient receiver for each device, one monitor can connect to and display information about electrical appliances (such as electric lights, air conditioners, etc.), saving enterprise administration costs. Our product is applicable to large office buildings, high-end residential buildings, administrative enterprises, smart villas, schools, hospitals, etc. that have a lot of electrical equipment to be managed.

The system is very scalable. Users can continuously add new equipment and functions, as long as the terminal firmware and control software is updated. For example, this system can deliver video and broadcast to each unit using a high-bandwidth LAN, making it a powerful method for ad broadcasting in office buildings. Additionally, a touch screen can provide users with more public information.

If electronic equipment vendors accept this scheme as the standard for equipment intercommunication, preset communication interface, information technology will increase user satisfaction.

This system uses the Altera® Development and Education (DE2) development board as the terminal control core. Embedded system-on-a-programmable-chip (SOPC) technology conveniently integrates Ethernet, SDRAM, a programmable I/O (PIO) interface, digital-to-analog (D/A) switch interface, and a universal serial bus (USB) controller into the system, which greatly contributes to the system implementation. With the μClinux cross-compilation development environment, we can immediately begin work, which greatly enhances efficiency. By migrating a clipped μClinux system, managing the

development board resources is very easy and convenient, which reduces development problems and shortens the development cycle.

Function Description

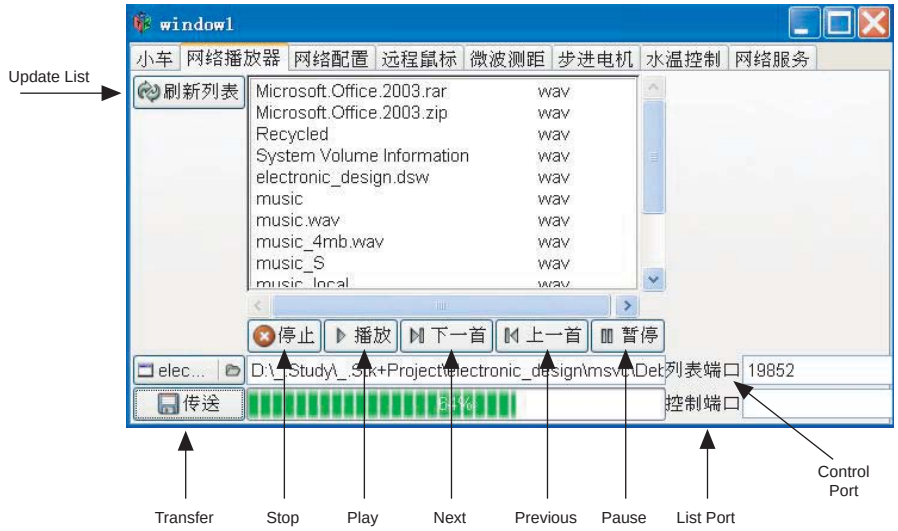
Our project is an experiment to provide proof of concept for smart network equipment. To this end, we completed two functions in this design:

- Network **.wav** music playback
- Remote mouse

Network .wav Music Playback

This functional module sends files from a PC to the DE2 board. The DE2 board stores the files into a mobile hard disk. The PC can obtain music files from the music folder and select any file to play on the DE2 board. See Figure 1.

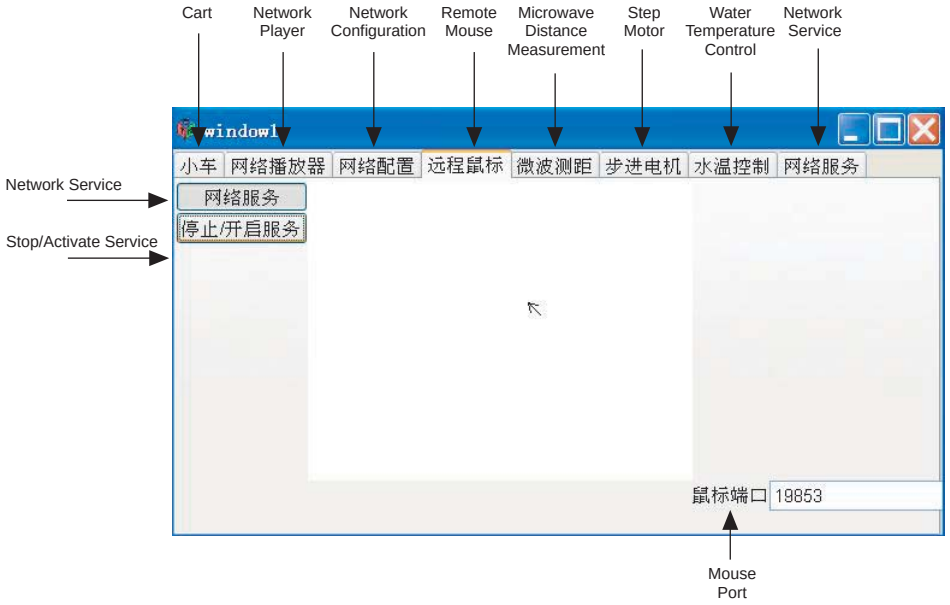
Figure 1. Network .wav Music Player



Remote Mouse

We can connect a USB mouse to the DE2 board's USB controller. The mouse pointer then appears on the PC and is controlled through the DE2 board. See Figure 2.

Figure 2. Remote Mouse



Performance Parameters

This section describes the performance parameters of our design.

Music Playing Performance

We broadcast a **.wav** file in our design. To play the music, the design writes 16-digit pulse-code modulation (PCM) codes into an audio FIFO buffer. This method is simple and direct, but has the following weaknesses:

- Large **.wav** files are inconvenient for network transmission.
- The files use excessive memory resources, causing low memory utilization.

Because of these limitations, we plan to use MP3 files in future designs. However, our current project is an experiment to demonstrate and materialize the concept of smart network equipment. Therefore, **.wav** files are fine for this project.

During our experiments, we found inferior sound quality and a noisy background. Through analysis, we determined that these issues are caused by the **.wav** file playback algorithm, i.e., cyclic query. Due to assimilated processing capability of **nios2kernel**, if we used the CPU query algorithm the design would overutilize the CPU and affect system processing. We will need to optimize this function in the future.

When we use an FTP server to upload files, the DE2 board's fastest receiving speed is 300 k/second, which is also the DE2 board's limit for processing data. The **.wav** file speed is 1,411 bps (or 176 k/second) and the design consumes some system processes and network services, making playback difficult to complete in software query mode. Therefore, this mode occupies too many processor resources, which is a problem we should address first in our future design work.

Remote Mouse Performance

The system can successfully collect real-time mouse movements and clearly the simulate mouse movement on a PC.

Design Architecture

The goal of our project is to convert equipment without information processing capability into an Internet node using a small communication module. Using powerful Internet interactivity, we can use and control the equipment in centralized management mode, simplifying office communication lines. Figures 3 and 4 show the block diagrams for our design.

Figure 3. Smart Terminal Block Diagram

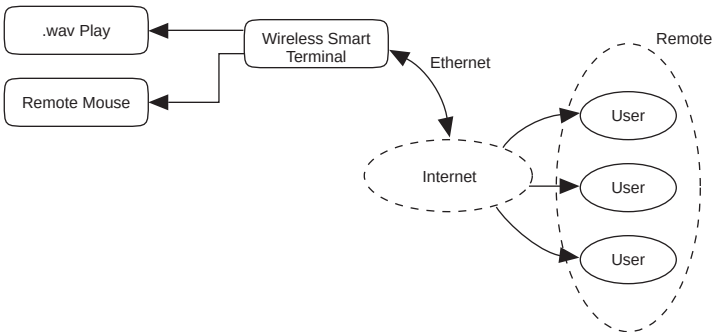
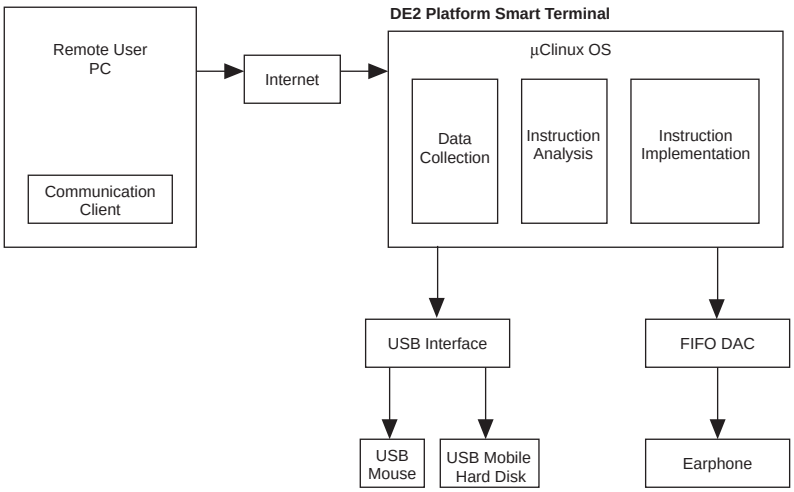


Figure 4. Functional System Block Diagram



The remote module uses special communication software to communicate with the DE2 board via the Internet and sends control instructions. The DE2 board obtains remote messages via Ethernet, interprets the message packet into specific instructions, and sends them out using a wireless transmission device. Equipment within the system's coverage analyzes the instruction upon receiving it and confirms

whether it is the destination device for the instruction. If yes, it carries out the instruction, implementing remote control and achieving our goal. The entire system has two core modules:

- An intelligent, integrated, personalized remote control (i.e., the remote equipment management software)
- Networking capability, functional expandability, reliability, and low power consumption from the DE2 board at the smart terminal

The DE2 board has abundant resources to meet the design demands, and we expect them to be expanded.

Design Methodology

The following sections describe our design methodology.

Migrating µClinux to the DE2 Board

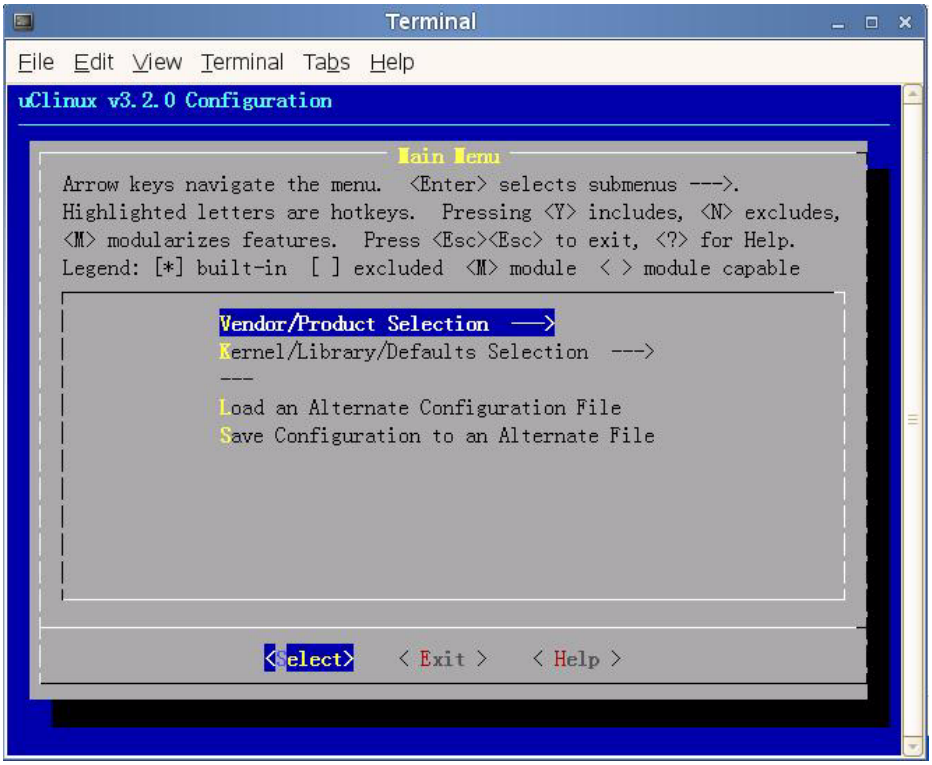
The µClinux system is a reduced Linux kernel without a memory management unit (MMU); we used a version 2.6 kernel. First we downloaded and installed the kernel:

1. We downloaded the µClinux kernel source code from the Internet at **http://www.uclinux.org/dls.uclinux.org/uClinux-dist-20070130.tar.bz2**.
2. We downloaded a µClinux kernel modification file from the Internet at **http://nioswiki.jot.com/WikiHomeOperatingSystems/µclinux/UClinuxDist/uClinux-dist-20070130-nios2-02.diff.gz?cacheTime=1170915312300**.
3. We downloaded a cross-compilation environment file from the Internet at **http://nioswiki.jot.com/WikiHome/OperatingSystem/µClinux/BinaryToolchain/nios2gcc.tar.bz2**.
4. We installed the cross-compilation environment, after which we could compile the kernel.

We used the following steps to compile the kernel:

1. At a command prompt, we typed `/make menuconfig` in the **UClinuxDist** directory, which opened the compilation interface shown in Figure 5.

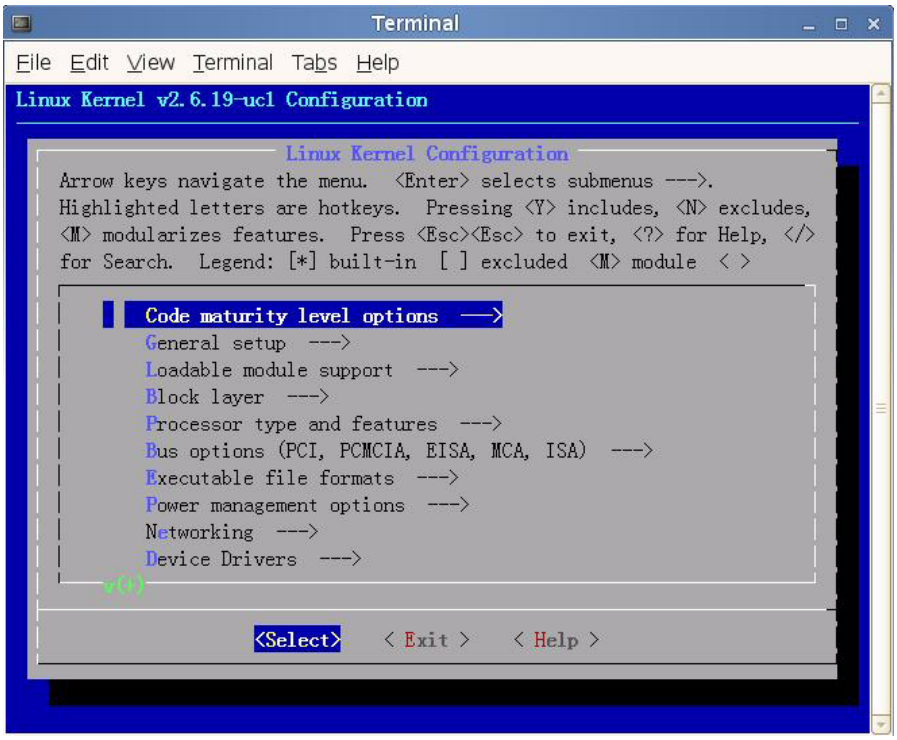
Figure 5. Compilation Interface



2. We selected the **Vendor/Product Selection** option.
3. For the **Select the Vendor you wish to target** option, we chose **(Altera) Vendor**.
4. For the **Select the Product you wish to target** option, we chose **(nios2nommu) Altera Products**.
5. Then we went back to the **Main Menu** and choose the **Kernel/Library/Defaults Selection**, and made the following selections:
   - (Linux-2.6x) Kernel Version
   - (None) Libc Version
   - [ ] Default all settings (lose changes)
   - [*] Customize Kernel Settings
   - [*] Customize Vendor/User Settings
   - [ ] Update default Vendor Settings
6. We chose **Select** to exit. The kernel driver and service configuration program displayed next (see Figure 6).

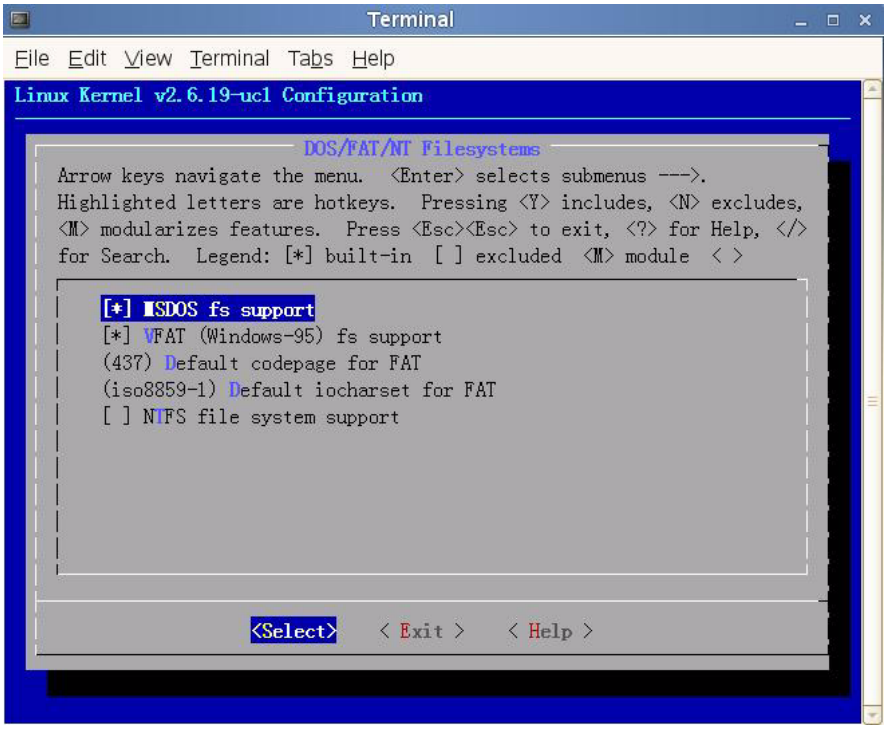
   We needed to configure options so that we could use the USB equipment and the network modules.

Figure 6. Linux Kernel Configuration



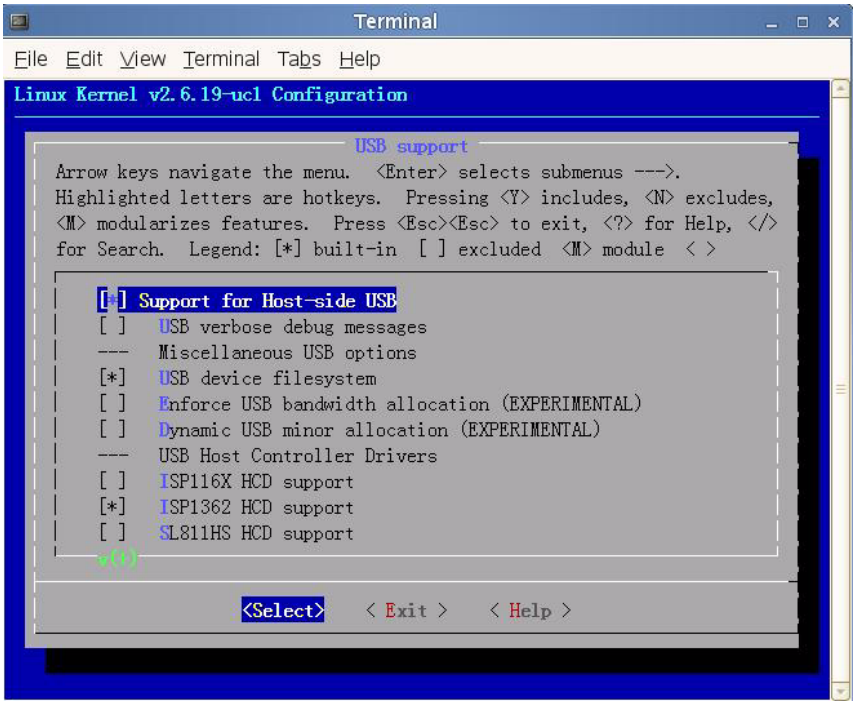
7. We made the selections shown in Figure 7 in the **File systems > DOS/FAT/NT Filesystems** screen to configure the USB file system.

Figure 7. DOS/FAT/NT Filesystems Screen



8. In the **Native Language Support** screen, we made the following selections:
   - (iso8859-1) Default NLS Option
   - [*]Codepage 437 (United States, Canada)
   - [*] NLS ISO 8859-1 (Latin 1; Western European Languages)
9. To configure the USB equipment driver, we made the following selections in the **Device Drivers > USB support** screen (see Figure 8).
   - [*] Support for Host-side USB
   - [*] USB device filesystem
   - [*] ISP1362 HCD support
   - [*] USB Mass Storage support
   - [*] USB Human Interface Device (full HID) support
   - [*] HID input layer support

Figure 8. USB Support Screen



10. We made the following selections to configure support for the DM9000A device.

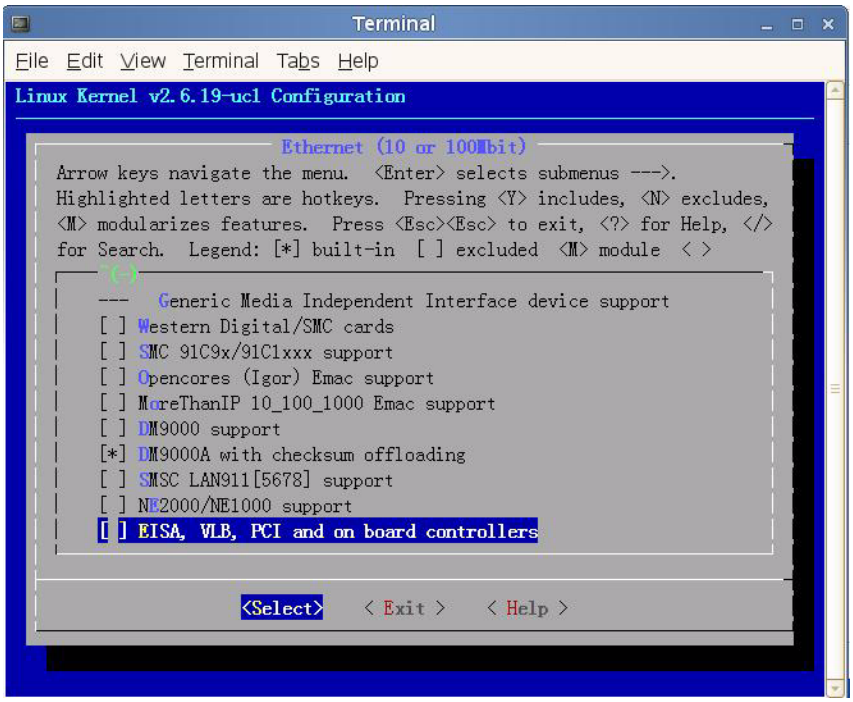
    - In the **Enter Network device support** screen, we selected:

      [*] Network device support

    - In the **Ethernet (10 or 100Mbit)** screen (see Figure 9), we selected:

      [*] DM9000A with checksum offloading

Figure 9. Ethernet (10 or 100 Mbit) Screen



After we finished the configuration we exited the kernel compilation program. We could compile the kernel using the following commands at the command line:

1. `: /make romfs` ↵

2. `: /make` ↵

3. `: /make linux image` ↵

The system compiles the kernel and creates a zImage file in the image contents directory. The next step was to download the kernel to DE2 board using the Nios® II Integrated Development Environment (IDE) terminal and activate the µClinux operating system (OS).

Operating µClinux on the DE2 Board

To run µClinux on the DE2 board, we entered the following commands in the Nios II 6.1 Command Shell:

`: /nios2-configure-sof de2-net.sof` ↵ (this command configures the hardware circuit)

`: /nios2-download-g zImage` ↵ (this command downloads the system)

`: /nios2-terminal` ↵ (this command activates the µClinux system)

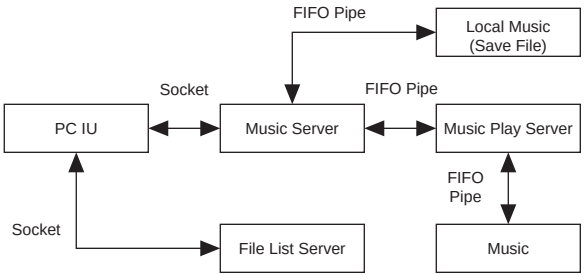
Figure 10 shows the interface after activation.

Figure 10. Activating µClinux

.wav File Network Uploading and Playing

Figure 11 shows the block diagram for uploading and playing **.wav** files on the network.

Figure 11. .wav File Uploading and Playing Block Diagram



The system functions are:

- *User interface*—The PC's user interface (UI) sends control information to the DE2 board, such as obtaining the music file list, playing specified music, suspending music, and stopping playback. The UI invokes network socket ports on the DE2 board for control:
  - 19851—Playing port
  - 19852—File list port
  - 19853—Remote mouse port
- *Music server process*—By transmitting control information, the music server process on the DE2 board automatically recognizes and controls the data. The music server process receives the control information from the PC via the network. It transmits the information to different energy-changing processes, and instructs them to finish various functions. When sending files, the music server process transfers the file name and file data to the music local process, which saves the files according to the designated file name. To control playback, the music server process sends control information to the music play server process along with the name of the file to play and when to pause or stop.
- *Music process*—When the music process finishes a task, it keeps reading data from the pipe and writes it into the audio FIFO buffer. If there is no data, it stops, which is equivalent to the function of a stop or pause.
- *Music play server process*—The music play server process obtains information required for playback, such as the file to play and control information. When a file must be played, the music play server process reads it and write its content into the playback pipe. After receiving the content, the music process begins playing immediately. Meanwhile, if the music play server process receives a pause or stop command, it stops writing audio information into the playback pipe and finishes the pause function.
- *List server process*—When a list request occurs, the list server process sends all file names in the music folder to the PC. The PC displays the received file names on the interface for the user to choose.
- *Music local process*—The music local process receives and saves files to the mobile hard disk. Files sent can be seen with the PC's list function.

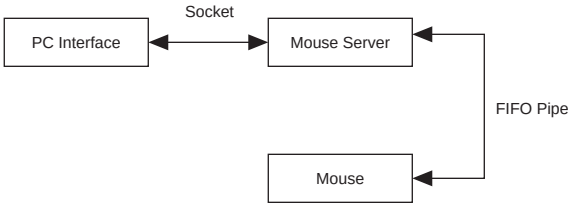
Remote Mouse

Figure 12 shows the remote mouse modules, including:

- *PC interface*—Because of performance considerations, the PC interface inquires about mouse movements every 10 ms, obtains 32-byte movement information, and shows it on the interface.

- *Mouse server process*—The process continuously reads the piped data, saves it in 32-byte buffers, and waits for the PC acquisition commands. When there is a PC inquiry, null information is sent out to ensure that the program performs even if there is no data in the buffer.

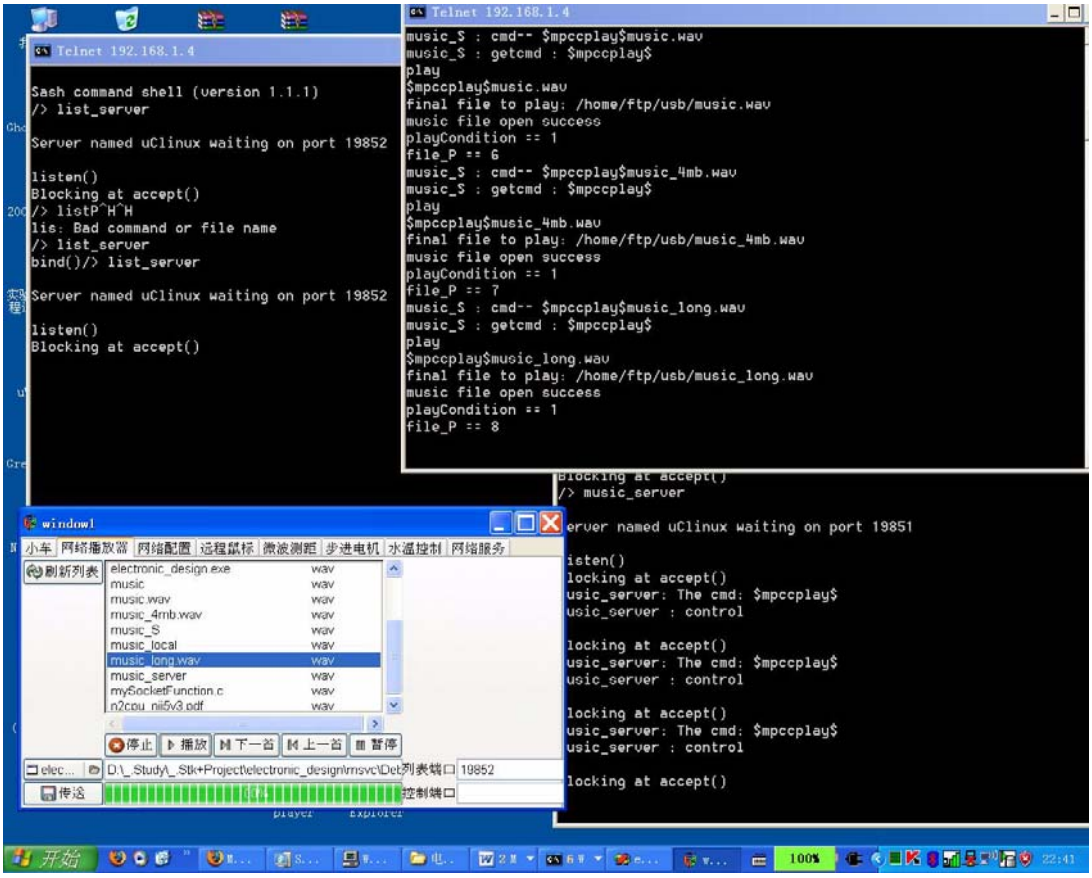
Figure 12. Remote Mouse Modules



Implementing On-Line Debugging

The obvious difference between embedded design and ordinary PC program debugging is the debugging difficulty because every debugging change requires the user to download the entire system file. With µClinux, however, the network interface simplifies the debugging process. To perform debugging using telnet, we entered µClinux, configured the network, activated the **inetd** server, logged onto the DE2 board using telnet, uploaded the program to debug using ftp, and debugged the program online. See Figure 13.

Figure 13. Telnet Debugging Interface



User Interface Compilation

We chose the GTK+ version 2.0 library to compile the UI for the following reasons:

- GTK+, a cross-platform library, can be run on the Windows, Linux, and Mac OS X operating systems. With this program, we can release software on current mainstream platforms without transforming it.
- It possesses a Glade item and can configure the UI quickly, saving a lot of mechanical work.

Our system has two program UIs:

- *Network player*—As shown in Figure 1 on page 22, the network player interface integrates two functions: a file list that displays the music files on the mobile hard disk and a network player that selects, plays, pauses, and stops the file listed previously.
- *Remote mouse*—When the user plugs a mouse into the DE2 board's USB port and activates the network service, the mouse pointer on the monitor moves when the user moves the mouse (see Figure 2 on page 23). The user can pause or activate this function.

Development Environment

Our system has the following development environment:

- *Development board*—Altera DE2 board
- *FPGA synthesis tools*—Quartus®II version 6.1 development kit
- *Cross-compiling development environment*—Suse Linux 10.2 x86
- *Kernel compilation environment*—Suse Linux 10.2 x86
- *User interface*—GTK+ version 2.0

Design Features

Our design uses the network as information media. With highly integrated technology, the design saves system costs and shortens the system configuration cycle.

The design positions the PC as a network terminal, integrating the network into every day life. Just imagine, you can select goods you need at home using a touch-screen LCD on your refrigerator. Suppliers can sell their goods online, shortening shopping time and minimizing retailers' costs for real stores.

In schools, enterprises that manage the electronic equipment has become an important aspect of daily maintenance. With a smart network platform, administrators can sit in their offices and click a mouse to control buildings, electric lights, and air conditioners throughout the campus. Costing only a little, the platform can save huge costs in long-term maintenance.

Technologically, the DE2-based system integrates:

- Embedded μClinux OS
- SOPC hardware/software integration
- Wave music file playback on the DE2 board
- Loading and using USB equipment on the DE2 board
- USB mouse
- Mobile hard disk
- On-line μClinux debugging
- Embedded Ethernet
- Audio D/A converter

We developed a smart control design concept and have essentially finished it. The system integrates network technology and an SOPC embedded system, and implements a comprehensive technology application. With the development of additional technologies, integration will provide better applications.

Conclusion

We spent over three months in the Zhang Zhigang Innovation Lab where we learned advanced SOPC technology, learned how to use Altera's FPGAs and software, and gained a lot of experience. Theory is abstract. Undergraduates who get used to accepting theories lack training, hands-on experience, and innovation. This contest gave us the opportunity to put what we have learned into practice. From program design to implementation, it showed us the project stages and the gap between theory and practice.

Usually we are very confident about what we have learned in school, but we may be unable to deal with a real project. Because it is systematic engineering, this project covers almost all of our undergraduate curriculum. With this benefit, we gained a lot from participating in the contest. We had to solve many hard problems in the contest, and using three different Altera software design programs was the first one. During the project, a trivial problem could cost us several hours work, so accumulating knowledge we could use in daily life was very important. A solid foundation will help us solve problems smoothly and efficiently.

We transplanted μClinux to the DE2 board, which made it convenient and efficient to use the DE2 board resources. The process plays a particularly important role in multi-tasking development. Modular concepts are critical for the design of complex tasks. An embedded OS offers an ideal operating environment for modular programs. In our design, we can perform debugging on each module, which increases debugging flexibility.

By exposure to Altera's FPGA hardware/software system, we learned about embedded equipment trends. Altera's Nios II IDE supports the developers in hardware and software, saving a lot of time. Additionally, the products have considerable flexibility: the Nios II CPU can be clipped according to users' demands, the system bus interface supports Verilog HDL and VHDL for hardware interfaces, and users only need to load configured equipment onto the SOPC Builder bus.

By participating in the contest, we learned that system engineering requires comprehensive knowledge, and its performance is determined by the weakest segment. A faulty segment aborts the entire system.

Finally, we extend our gratitude to Instructor Zhang and the assistant instructors in the innovation lab for their support in equipment and technology. Without them, we would not have completed the program.

References

[1] Zhigang, Zhang, *DE2 Platform Application and DSP Builder Technology,* Training Material for 2007 Altera Cup Shanghai Jiao Tong University Electronic Design Contest, 2007.

[2] Zhigang, Zhang, *SOPC Technology,* Reference for 2007 Altera Cup Shanghai Jiao Tong University Electronic Design Contest, 2007.

[3] Thomas, Nathan, *Developing on Linux An Introduction to Development on Linux*, Red Hat, Inc., nthomas@redhat.com.

[4] Corbet, Jonathan, Rubini, Alessandro, and Kroah-Hartman, Greg, *Linux Device Drivers, Third Edition*, O'Reilly, Beijing: February 2005.

[5] Strahnen, Dr. Ing. M., *HOWTO: NIOS-CPU with additional hardware driven with μClinux*.

[6] *Linux Application Program Development Guide: to Use Gtk+Gnome Library*

[7] *GTK+ Reference Manual*, GNOME Documentation Library.

[8] **http://nioswiki.jot.com**

Third Prize

Multi-Functional Digital Albums Based on the Nios II Processor

Institution: **Information Science Institute, Beijing Jiaotong University**

Participants: **Cheng Hong, Rui Deng, Yongxin Ye**

Instructor: **Xiaoming Ding**

Design Introduction

As digital cameras and high-pixel mobile phone cameras have become more popular, users save their photos in a variety of storage devices (e.g., PC hard disks, semiconductor storage devices, and CD-ROMs). Today, most printed photos are from digital cameras or mobile phone cameras. As users create more and more digital photos, they will spend a lot of money developing these photos, and traditional photo frames and albums do not store these photos well. Additionally, directly editing photos on a portable hardware platform independent of a PC would be very convenient when traveling.

Today, the main functions of digital albums on the market are photo storage and playing, which wastes valuable resources. It would be significant to integrate most functional modules into one system that fully utilizes the processor, implements more user operations, and meets more requirements.

Our design can be used together with a digital camera. It allows users to edit photos stored in the camera directly and then transmit the photos to the home or office through a network interface, relieving pressure on storage devices such as secure digital (SD) cards. A digital album that features digital photo storage, playing, viewing, editions, and network transmission will satisfy users' requirements in today's information society.

Our design provides users with a complete do-it-yourself (DIY) space that helps them create personalized photos. When the communication interface between the album and the mobile phone is developed in the future, users can send DIY photos directly to their friends' mobile phones.

Application Scope and Targeted Users

The concept of a digital photo is well received in society. Our design plays a role in any field in which digital photos exist. With the unique editions, receiving, and transmission functions of our design, travelers do not need to worry about mass storage space for their photos. The user can use this convienent, portable design at home or while traveling; our design can be widely applied to all groups.

Nios II Processor Design Advantages

Using the Nios® II processor provides the following advantages:

- *Configurability*—Compared with traditional processors, the Nios II processor features configurability to facilitate users' designs. We can customize the system according to our demands. Because Altera provides an abundant intellectual property (IP) core library, we can implement our design ideas quickly, shortening the development period. Additionally, the multi-functional technology cuts costs significantly and is more flexible.

- *Integrated development environment (IDE)*—The complete Altera®-provided IDE, from the Quartus® II software to SOPC Builder and to the Nios II Embedded Design Suite (EDS), creates all the hardware and software conditions for our design. Based on Altera's FPGAs, we can complete the system configuration and implementation with higher performance and stability.

- *Custom instructions and peripherals*—The Nios II soft-core processor can integrate 256 custom instructions and peripherals to accelerate system operation. The Nios II software development environment provides a standard program interface to facilitate program migration.

- *Unique C-to-Hardware Acceleration (C2H) Compiler functions*—The C2H Compiler has played a great role in the time since its launch. With this application, we do not need to worry that the complex algorithms in the Nios II EDS environment will influence the system performance. Instead, we can accelerate algorithm bottlenecks with the C2H Compiler to easily speed system operation.

Function Description

This design uses the Development and Education (DE1) multimedia development board to design a multi-functional digital product—a multi-functional digital album. The album supports all basic functions of its kind on the market while adding some new functions to meet the requirements of different user groups. This product contains the following functions:

- *Digital photo storage*—It is necessary for a digital album to have a certain amount of photo storage. Because the DE1 board has an SD card interface, we only need to load the SD card's data, demand, and address line to the Avalon® bus and the Nios II processor can control the SD card data reading and writing.

- *Digital photo viewing and playing*—It is a basic requirement for a digital album to allow users to view their photos at any time. With the µC/FS file system, the album can conveniently notify the Nios II CPU of photo files in the attached storage media, and users can choose any photo to view and play.

- *Special music effect*—Multimedia is popular in daily life. Most of Altera's FPGA development boards can implement multimedia playing. Considering the pressure on the storage media, we chose compressed G.729 code streams as the music format and embedded a laboratory decoding algorithm into our digital album so that users can listen to beautiful music while viewing photos, allowing it to be a "voice album."

- *Digital photo and audio file management*—After migrating the file system, we can easily group and archive files for convenient operation.

- *Digital photo editions and processing effects*—With an embedded platform we can provide a photo processing effect similar to that on a PC by using software like Adobe PhotoShop.

- *Photo format compression and decompression*—Because photos are usually stored in JPEG format, our multi-functional digital album must meet that requirement. Therefore, we embedded a JPEG decoder module into the Nios II processor, conducted all processing with decoded RGB data, and then compressed the photos into JPEG format for network transmission.

- *Digital photo perfection*—Traditional photos are usually stored in frames and can fade over time. In contrast, with a digital format we can add beautiful frames to photos for an amazing effect.

- *Digital photo network transmission*—The digital album design can receive JPEG code streams through the network and transfer them to the FPGA for processing. Then, the album compresses the photo data into JPEG code streams through the FPGA and transmits the code streams over the network, allowing the user to share photos with others. Implementing this function requires us to develop a network interface on the DE1 board, so we created a circuit board with a network function that connects with the FPGA through the general-purpose I/O (GPIO) pins. Therefore, our design can partner with digital cameras, allowing travelers to share photos with others anytime and anywhere.

- *User interface*—The DE1 board provides us with a PS/2 interface, so we can load a mouse and keyboard onto the Avalon bus. To respond simultaneously to the mouse and keyboard data, we modified the on-board circuit, allowing the Nios II processor to respond to the PS/2 peripheral any time and constructed the whole album.

- *Digital camera partner*—This design provides direct photo viewing and processing operations.

- *Digital watermark embedding and extraction*—When processing photos, users often hope the photos are not modified by others. The digital watermark embedding function of multi-functional digital album can be used as a photo copyright management tool without changing the effect of the photo.

When implementing the functional modules, we fully used the Nios II processor features to invoke the functional modules with the µC/OS-II embedded operation system.

Performance Parameters

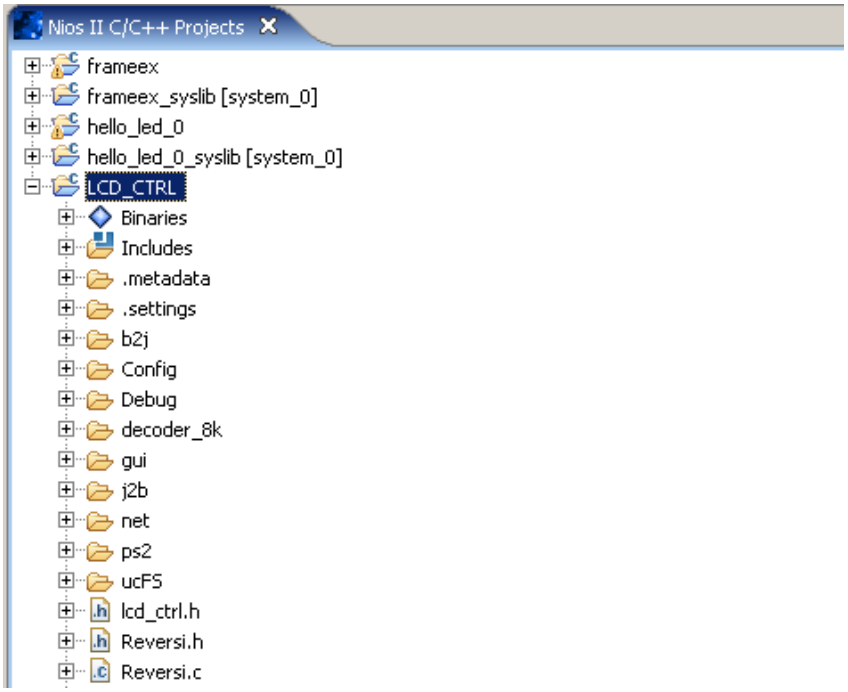
Figure 1 shows the hardware resource utilization of the multi-functional digital album.

Figure 1. Hardware Resource Utilization

Flow Status	Successful - Wed Sep 12 21:04:51 2007
Quartus II Version	6.1 Build 201 11/27/2006 SJ Full Version
Revision Name	DE1_SD_Card_Audio
Top-level Entity Name	DE1_SD_Card_Audio
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	Yes
Total logic elements	5,515 / 18,752 (29 %)
Total combinational functions	4,781 / 18,752 (25 %)
Dedicated logic registers	3,364 / 18,752 (18 %)
Total registers	3481
Total pins	273 / 315 (87 %)
Total virtual pins	0
Total memory bits	79,360 / 239,616 (33 %)
Embedded Multiplier 9-bit elements	4 / 52 (8 %)
Total PLLs	2 / 4 (50 %)

Figure 2 shows the system software architecture.

Figure 2. Software Architecture



The system has the following specifications:

- System performance index parameters
  - Photo parameter: 320 x 240 JPEG format
  - LCD display parameters: 320 x 240 resolution, 5.7 inches
- System resource utilization (see Figure 1 on page 40 for details)
  - FPGA: Altera Cyclone® II EP2C20F484C7
  - Program operation space: 8-Mbyte SDRAM
  - Static storage: 512 Kbyte SRAM
  - User interface: key switch, LED, seven-segment nixie tube
  - VGA interface
  - Stereo audio codec
  - PS/2 mouse and keyboard interface
  - External network module control board for the GPIO interface and LCD display
  - Storage: Kingston SD card 2.0G/1.0G
- Expanded resources
  - SD card, four-line SD mode
  - Network controller circuit board
  - PS/2 one-for-two interface
  - LCD 65,000-color screen

Design Architecture

Figure 3 shows the hardware design and Figure 4 shows the system hardware and flow.

Figure 3. Hardware Design Block Diagram

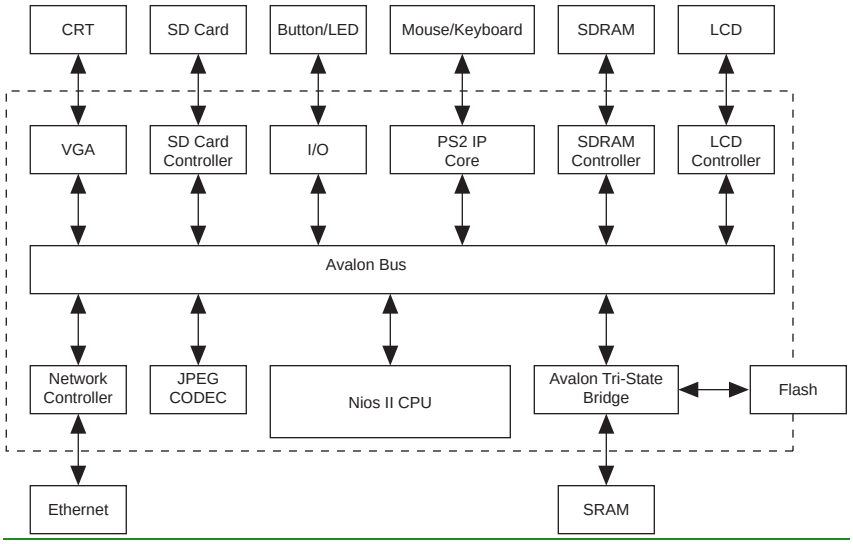
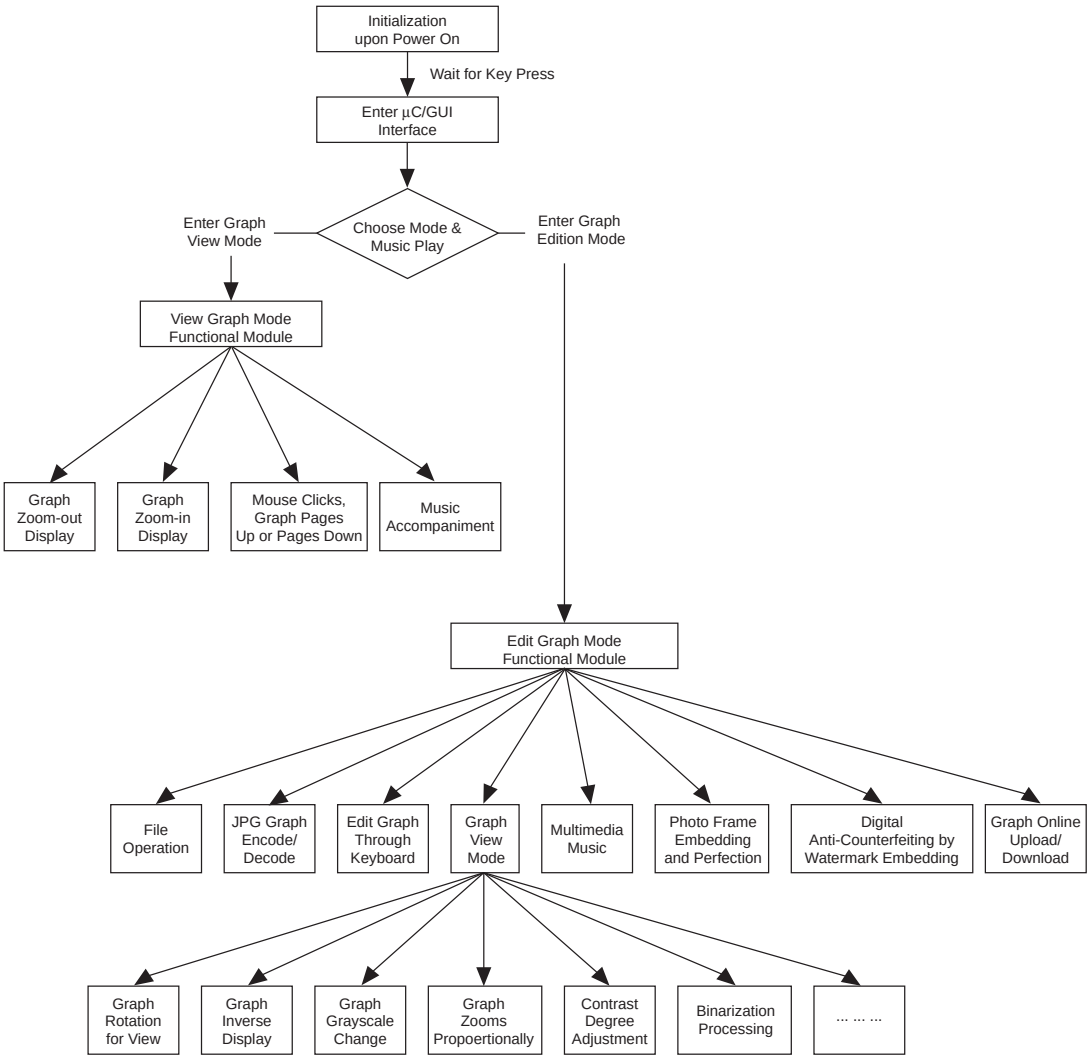


Figure 4. System Flow Chart



Design Methodology

This section describes the design methodology.

System Design

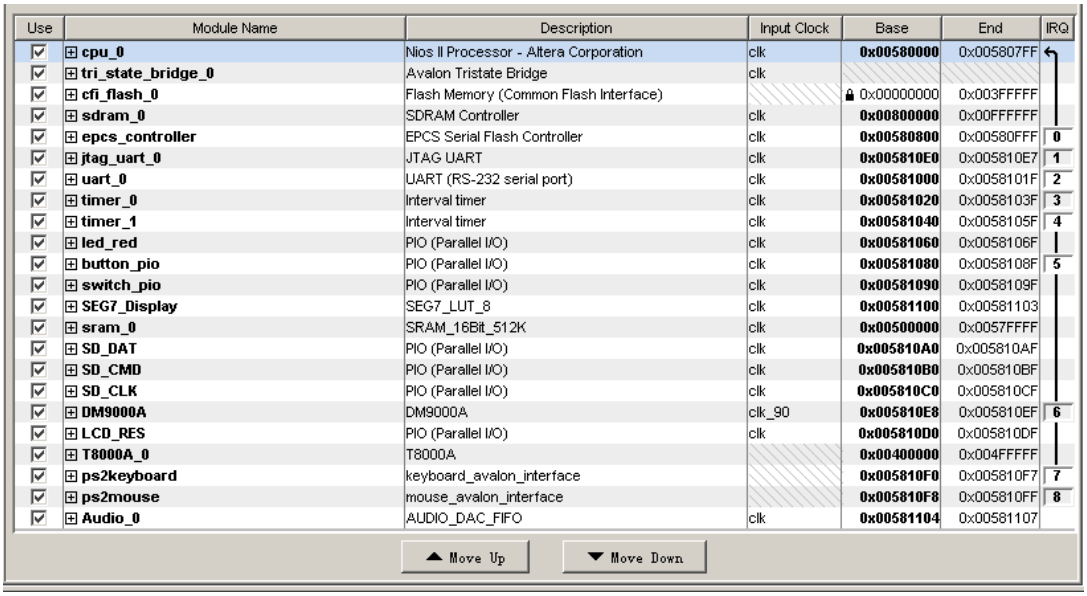
The system design contains these functional modules:

- SD card file system
- Keyboard input
- Audio digital-to-analog D/A converter (DAC)
- DM9000A network controller

- TCB8000A LCD controller
- VGA display
- JPEG codec
- Image processing
- Audio decoding

All functional modules are integrated into one system using SOPC Builder as shown in Figure 5.

Figure 5. SOPC Builder System Architecture



Implementation

The whole design adopts a top-down methodology to build the system hardware module in SOPC Builder. All required functional modules are integrated through the Avalon bus to improve system stability. Because a multi-functional digital album particularly needs a stable hardware system when performing multiple functions, we tried to use only SOPC Builder for the hardware.

SOPC Builder's powerful system integration feature enabled us to shorten the design cycle, build a stable system in a short time, complete the system software design on the basis of the hardware, and integrate the software and hardware into a complete system.

SD Card Storage Module

As a totally open standard, SD cards are used in MP3 players, digital video cameras, digital cameras, e-books, audio-visual (AV) appliances, and particularly in ultra-slim digital cameras. SD cards are the same shape as multimedia cards (MMC), but they are a little thicker than MMCs and have larger capacity. Additionally, SD cards are compatible with MMC interface specifications. The nine-pin SD card changes the transmission mode from serial to parallel, improving the transmission speed. It reads and writes faster than MMCs and is more secure.

To make the system more applicable and compatible, we decide to use an SD card as the primary storage media to store photos, music, etc. The one-line SD card read setting on the DE1 board is limited in speed. We modified the setting to a four-line mode to make it more compliant with the system demands.

Table 1 defines the SPI mode time sequence signals, Table 2 defines the four-line SD mode time sequence signals, and Figures 6 through 9 show the time sequences for the SPI and SD card modes.

Table 1. SPI Mode Time Sequence

CS	Host-to-card chip selection signal.
CLK	Host-to-card clock signal.
DataIn	Host-to-card data signal.
DataOut	Card-to-host data signal.

Table 2. Four-Line SD Card Mode Time Sequence

CLK	Host-to-card clock signal.
CMD	Control and answer signal.
DAT0-DAT3	4-bit data line.
VDD, VSS1 and VSS2	Power supply and grounding signal.

Figure 6. SPI Mode Read Time Sequence

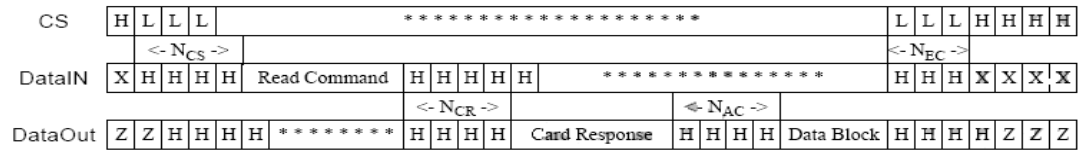


Figure 7. SPI Mode Write Time Sequence

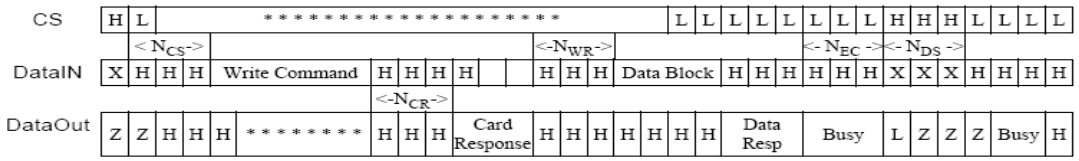


Figure 8. Four-Line SD Card Mode Read Time Sequence

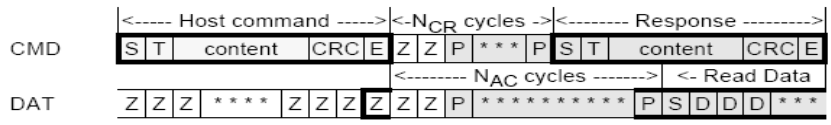
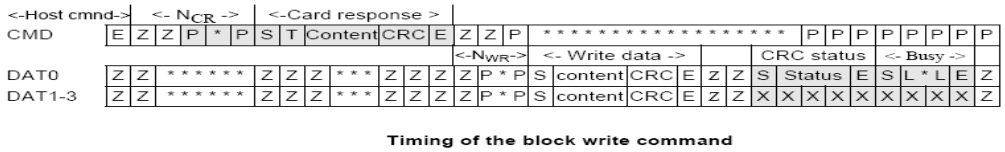


Figure 9. Four-Line SD Card Mode Write Time Sequence



PS/2 Keyboard and Mouse IP Core Module

The PS/2 mouse and keyboard protocols are the same. The PS/2 mouse and keyboard perform a bidirectional synchronous serial protocol; each cycle, the data line transmits a bit of data and a clock line transmits a pulse that is read. The keyboard/mouse can transmit data to the host and the host can transmit data to the devices. Because the host has priority on the bus, it controls communication from the keyboard/mouse by pulling the clock low.

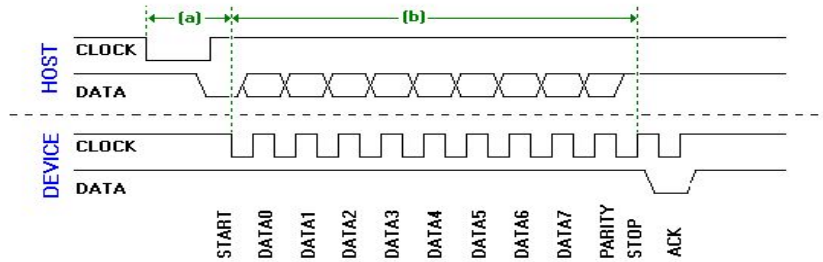
A standard PS/2 mouse supports the horizontal displacement, X, vertical displacement, Y, and mouse left, middle, and right clicks. It reads the input at a fixed frequency and then marks the reflected movement and click status. A standard mouse has two counters to track displacement. The X and Y displacement counters can store 9-bit binary complements, and each counter has a corresponding overflow flag. The overflow flag content and the three mouse-click statuses are transmitted together to the host in the form of a three-byte mobile data packet. The displacement counter represents the displacement from the last displacement data packet sent to the host. Table 3 shows the data format.

Table 3. Displacement Counter Data Format

Original Bit	8 Data Bits	Parity Bit	Stop Bit	Answer Bit
Logic 0	Low bit first	Determine the parity according to the number of 1s in the data bit.	Logic 1	Used in host/device communication.

Figure 10 shows the host/device communication.

Figure 10. Host/Device Communication



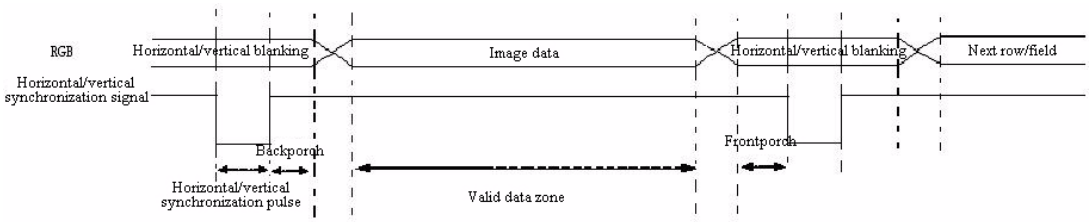
According to standard PS/2 protocol, we compiled the mouse and keyboard hardware modules (that are Avalon slaves in SOPC Builder) to complete the user interface.

VGA Image Display Module

Although this project uses an LCD display, we also designed a VGA interface. The VGA time sequence includes horizontal and vertical time sequences. Both time sequences contain the horizontal (vertical) synchronization pulse, the width from the end of horizontal (vertical) synchronization pulse to the beginning of valid data display zone (backporch), the width of the valid displaying zone, and the width from the end of valid data display zone to the beginning of horizontal (vertical) synchronization pulse (frontporch) parameters. The logic zone between the width of the horizontal valid display zone and the width of the vertical valid display zone is regarded as the video zone, and other zones are regarded as blank zones.

Figure 11 shows the time sequence of one row or one field.

Figure 11. Horizontal/Vertical Time Sequence



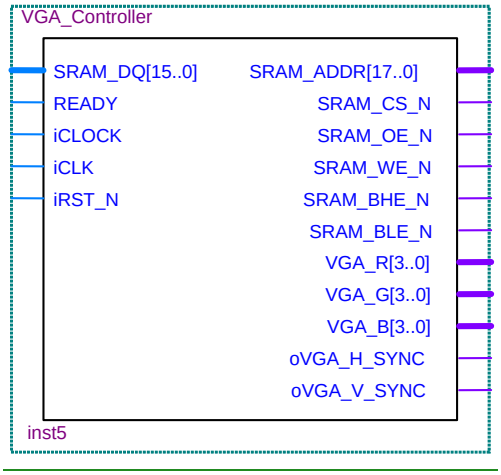
Based on the time sequence in Figure 11, Table 4 shows the time during each stage of the horizontal/vertical signal.

Table 4. Time During Each Stage of Horizontal/Vertical Signal

Parameter	VGA(640×480)
Horizontal scanning frequency/vertical scanning frequency	31.469 KHz/59.94Hz
Scanline time	31.77 μs/16.68 ms
hsync/vsync	3.77 μs/0.06 ms
Horizontal/vertical frontporch	0.94 μs/0.35 ms
Horizontal/vertical backporch	1.89 μs/1.02 ms
Valid video zone	25.17 μs/15.25 ms

According to the time sequence requirements, we compiled the VGA controller to display images. We used SRAM as the image data memory to display the images after decompression and processing. Figure 12 shows the VGA controller.

Figure 12. VGA Controller



LCD Display and TCB8000A Controller Module

We used a 5.7-inch, 65,000-color thin film transistor (TFT) as the display and the TCB8000A device as the controller. During the design, we only needed to integrate the interfaces (including an 8-bit data line,

write transmission signal, address signal, reset signal, and chip select signal) between the controller and the CPU in SOPC Builder, and load the module as a peripheral to the Avalon bus to implement the image data display.

System Software Design

The following sections describe the system software design.

µC/OS-II Operating System

As a free, embedded operation system with public source code, µC/OS-II has been applied widely in many fields worldwide, such as mobile phones, routers, hubs, aerocrafts, medical instruments, etc. µC/OS-II is suited to small control systems because it features high efficiency, small size, excellent real-time performance, good scalability, etc. Software development of the system is completed in the Nios II-integrated µc/OS-II embedded operating system (OS).

In our multi-functional digital album system, we created Task Main (main control tasks), Task_Gui (graphical user interface [GUI] user interface display task), and Task_Music (music playing control task). µC/OS-II schedules the three tasks using a message mailbox.

JPEG CODEC Module

The JPEG CODEC module converts the image format. Digital cameras also embed the CODEC for JPEG images, and this module is indispensable for our digital album. Figures 13 and 14 show the JPEG encoding and decoding procedure.

Figure 13. DCT-Based JPEG Encoding Procedures

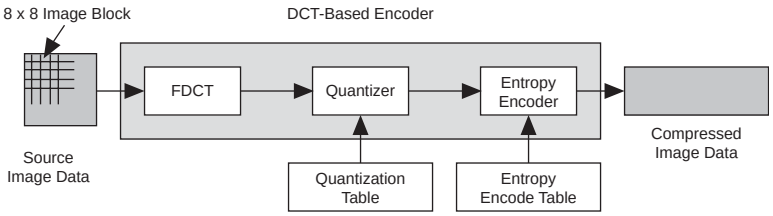
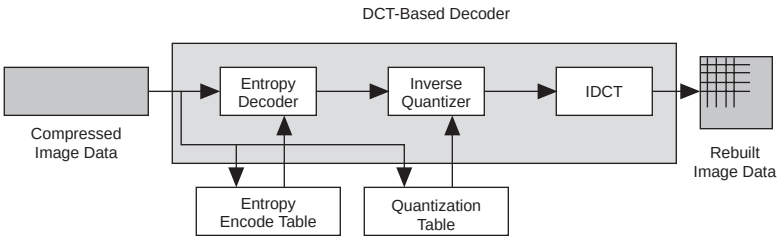


Figure 14. DCT-Based JPEG Decoding Procedures



In the procedures, the core algorithms are discrete cosine transform, quantization, inverse-quantization, and encoding/decoding of the quantization coefficient with a Huffman variable-length encoder. Additionally, the procedures involve technologies such as transforming the color model, a zigzag arrangement of quantization coefficients, the differential pulse-code modulation of the DC coefficients, and the run-length encoding of the AC coefficient. Figures 15 and 16 show the encoding and decoding flow charts, respectively.

Figure 15. Encoding Flow Chart

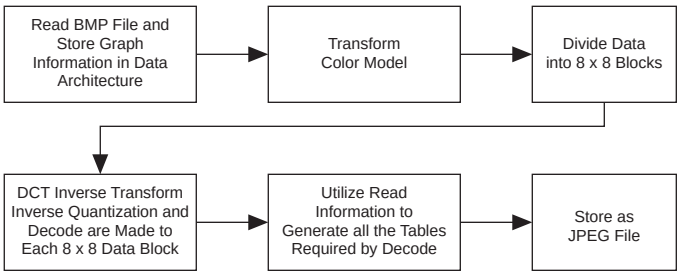
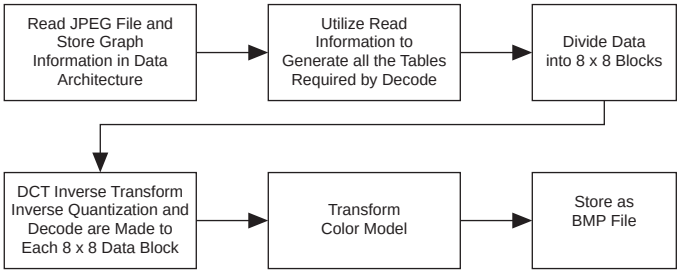


Figure 16. Decoding Flow Chart



With these procedures, users can view real-time photos from digital cameras, including JPEGs with different encodings, making the album a real partner to the digital camera and a good photo processing tool during traveling.

Network Transmission and Receiving Module

Migrating µC/OS-II enables us to make a network expansion board in the hardware infrastructure. Our network template creates its own data transmission method by invoking bottom receiving and transmission packet programs with the Nios II processor. The data is packaged at the Nios II transmission terminal and the data is unpacked at the network receiving terminal. Figure 17 shows the multi-functional digital album's transmitting/receiving interface on a PC terminal.

Figure 17. Transmitting and Receiving Terminal Interface

Accessing the network functions means the system can be edited and viewed on-line, which significantly improves the system's application scope. On the network, the system can receive remote image data without depending on a local SD card data source. Additionally, users can transmit their photos to friends for sharing while traveling.

µC/FS File System

µC/OS-II is contained in the Nios II software development integration environment so that users can apply it in their own software engineering. To make the concurrent task processing and CPU sharing more reasonable, we used µC/OS-II with a file system. In the beginning, we selected the zlg file system, but its speed was unsatisfactory during testing. The reading and writing of 1-Mbyte data into the SD card required 37 seconds and 57 seconds, respectively. We analyzed why it was so time consuming, and found that one problem was the SD card's read/write limitation. On the DE1 development board, the SD card uses one-line reading and writing, i.e., it only has one data line. By adding data lines, we modified the mode to four-line reading and writing. Our tests showed that the speeds were improved to 17 seconds and 27 seconds, respectively. This result was still not quadruple the original speed as we expected.

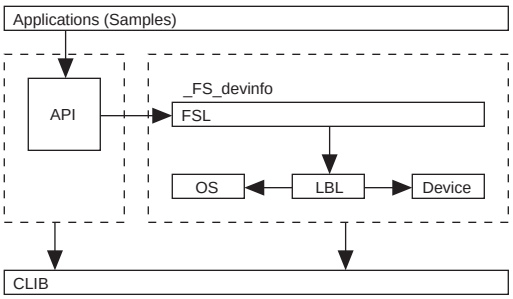
We determined that another reason for the speed problem was the file system. By searching the Internet, we found that zlg/fs had low performance and consumed a lot of processing time. So we decided to use Micrium's µC/FS, which has excellent compatibility with µC/OS-II and high performance. After several weeks effort, we successfully migrated µC/FS version 1.34 to the DE1 board to establish a file system for a four-line SD card. According to the comparison test, the reading and writing speeds were greatly increased, taking just 3.6 seconds and 11 seconds, respectively for 1 Mbyte of data. This speed basically satisfied our speed requirement for accessing data files. However, writing was still slow because when data is written into the SD card, each block needs to calculate a 16-bit cyclic redundancy code (CRC), occupying a lot of transmission time. Therefore, we accelerated this part with a custom instruction. Additionally, if the CPU operating frequency improves, the write speed will also increase.

We briefly introduce the µC/FS file architecture in the following sections. Table 5 shows the µC/FS software packet files. Figure 18 shows the µC/FS block diagram.

Table 5. µC/FS Software Packet Files

Application Program Interface (API)	Some external interfaces.
CLIB	Implements some basic standard C functions, including processing strings and memory.
CONFIG	The configuration items of each target, including all configuration types.
DEVICE	The packaging of the device layer, currently including hard disk, RAM, smc, windows (I/O interface) etc. It primarily implements the FS__device_type interface.
FSL	File system layer (supporting all file systems), currently including the FAT file system.
LBL	Some OS-level operations (number of signals) of the logic block layer (the packaging of the OS and device layer).

Figure 18. µC/FS Block Diagram



Some functions available in the file system are:

- File system control functions
  - *FS_Exit()*—Stop file system.
  - *FS_Init()*—Start file system.
- File access functions
  - *FS_FClose()*—Close a file.
  - *FS_FOpen()*—Open a file.
- Direct input/output functions
  - *FS_FRead()*—Read data from file.
  - *FS_FWrite()*—Write data to file.
- Directory functions
  - *FS_CloseDir()*—Close a directory.
  - *FS_MkDir()*—Create a directory.

- *FS_OpenDir()*—Open a directory.
- *FS_ReadDir()*—Read from a directory.
- *FS_RewindDir()*—Reset position in directory stream.
- *FS_RmDir()*—Remove a directory.

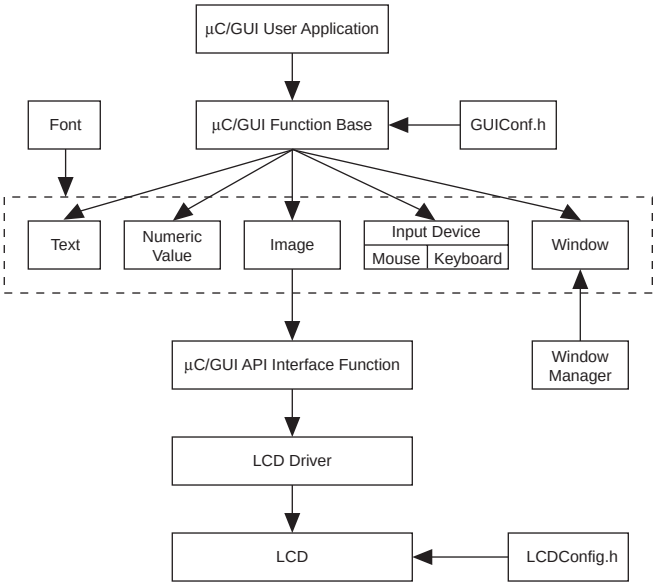
µC/GUI Interface

µC/GUI is an excellent graphic software for embedded systems, featuring open source code, portability, configurability, stability, and reliability. µC/GUI provides rich interface elements, such as buttons, edit boxs, sliders, etc. Additionally, it supports an efficient windows callback mechanism to provide interfaces between interfaces and application functions. We developed the user interface of our multi-functional digital album system with this tool.

µC/GUI Architecture

Figure 19 shows the µC/GUI system architecture.

Figure 19. µC/GUI System Architecture



The µC/GUI function library provides GUI interfaces to user programs. It contains functions such as text, numeric values, 2-dimensional (2-D) images, input devices, and various window objects. The input device can be a keyboard, mouse or touch screen. 2-D images contain pictures, beelines, polygons, circles, ellipses, arcs, etc. Window objects include buttons, edit boxs, progress bars, check boxs, etc.

The µC/GUI function library can be configured via the **GUIConf.h** file, including whether to use a memory device or window management device, whether to support an OS or touch screen, the size of the dynamic memory to be configured, etc. Various hardware-related attributes are defined in the **LCDConf.h** file, such as LCD size, color, and interface function. The LCD driver interprets the µC/GUI functions into the liquid crystal interface function defined in the **LCDConf.h** file regardless of the hardware connection. With the driver, the µC/GUI-LCD hardware interface converts the hardware interface function into an LCD read/write function as defined in the **LCDConf.h** file.

µC/GUI Migration

The first step in migration is modifying the **GUIConf.h** and **LCDConf.h** configuration files. According to the display module requirements of the digital album system, we configured the relevant parameters in the configuration files.

µC/GUI provides drivers for different LCD controllers. For example, the KS0713, SED1335, and T6963 controllers have corresponding LCD drivers. However, our system's display module is the TOPWAY TCB8000A LCD controller with a TFT 65,000 color LCD screen, and µC/GUI does not provide a driver for it. Furthermore, unlike the LCD controllers for which µC/GUI provides drivers, the TCB8000A controller has an independent screen control instruction system. These factors made it difficult to migrate the controller for µC/GUI.

During migration, we first used the TCB8000A instruction system to let µC/GUI provide the most API functions for the upper application function. Then, for API functions that the TCB8000A controller does not support in hardware, we repaired the drivers using software. Last, by testing a large quantity of controls, we adjusted the TCB8000A display instruction parameters used in the LCD drivers, optimizing the LCD driver performance and seamlessly migrating µC/GUI for the TCB8000A controller.

µC/GUI provides mouse and keyboard drivers. However, when we implemented the mouse and keyboard modules in the system, we needed to consider driver problems when combining the IP module with µC/GUI. Therefore, we compiled the keyboard data receive function in the Nios II processor so that it is invoked by the µC/GUI keyboard input driver and the obtained hardware keyboard value is transmitted to µC/GUI to complete a keyboard event. Then, µC/GUI processes the key during a message processing loop and transmits the key to the current focus form in µC/GUI. With this system module, we can transmit any input information to µC/GUI and implement text input for system editions.

The mouse driver is more complex. Fortunately, µC/GUI has good mouse driver support and its form management and message processing mechanism allows the mouse information to be updated and processed in real time.

The album includes the following functions:

1. The µC/FS file system informs µC/GUI of the photo data stored in the attached SD card.
2. After JPEG decoding, the RGB data is output.
3. The system conducts image processing, output display, editing, and processing in various modes.

Figures 20 and 21 show the album in view and edit mode, respectively.

Figure 20. Album Schemes in View Mode

Figure 21. Album Menu Schemes in Edit Mode

Digital Watermark Embedding

This module is embedded as a menu management sub-block, so that we can add a digital watermark to our photos to protect them. Due to time and system resource limitations, we did not integrate this module into the system.

Applying SOPC Concepts

For this design, our hardware structure is completely integrated using system-on-a-programmable-chip (SOPC) concepts. SOPC design has the advantage that hardware can be rebuilt and reconfigured, which is a unique feature of FPGAs in hardware development. Additionally, the top-down SOPC design flow dramatically shortened our design cycle. At the very beginning of the design, we considered using SOPC design concepts for our system. In our opinion, the biggest advantage of embedded systems is not making a perfect design, but using the least number of resources to complete the design while optimizing power consumption and integrity. We also must consider future development of the system. If the current design is limited in future development and expansion, it will not be a successful design. In our design, we preserve many operations for future system expansion to allow users to bring their ideas into the system to perform more and better functions.

Design Features

Our system has the following features:

- The µC/OS real-time operating system (RTOS) is a key factor in the normal operation of all functional modules in the system. From a system perspective, this design integrates a digital album, image processing, compression and decompression, transmission, and receiving into one system for the first time. The Nios II CPU and Avalon bus arbitration played an important role in implementing the system. To achieve the integration and operation of functional modules in the multi-functional album, we introduced an OS embedded in the Nios II processor to schedule the functional tasks, providing stability, improving system performance, and simplifying our design.

- This design employs multiple user interfaces including SD card memory, LCD interface, Ethernet interface, PS/2 mouse and keyboard interfaces, etc. We loaded these peripherals onto the Avalon bus using SOPC Builder. In spite of the large number of interfaces, the modules operate correctly in the integrated SOPC environment.

- For the software interface, we successfully migrated µC/GUI into the the Nios II environment. The user interface makes the design friendly and photos are easy to manage.

- To use the mouse and keyboard interfaces simultaneously, we expanded the original PS/2 interface so that one PS/2 interface can allow the Nios II processor to respond to the mouse and keyboard interrupt signals simultaneously. To provide fast reading and writing to the system for

the SD card interface, we modified the original one-line SPI mode into a four-line SD mode, improving the read/write speed by four times and satisfying the system requirements.

- Implementing µC/GUI, µC/FS, and µC/OS-II in the Nios II environment provided a good user interface for our album. The whole system must operate under a control environment, and combining the three elements meets the system requirements and provides good driver support to the hardware infrastructure.

- The expanded GPIO helped us load the network control module and LCD. The high-integrity SOPC design ensures stable communication in the whole system.

Conclusion

For this contest, we successfully implemented a multi-functional digital album based on the Nios II processor. We learned a lot from participating in the contest and now understand Nios II embedded systems more.

The system design targets an FPGA. We fully used Nios II features to control the whole system's operation. With OS scheduling, the tasks are managed according to the task priority. Where conflicts exist, the tasks are delivered to the OS for management. For example, in the beginning, the mouse and keyboard IP cores needed to be interrupted continuously and the Nios II processor needed to respond continuously, causing conflicts in some programs and causing the mouse not to operate in the GUI. After we introduced the OS, there were no system conflicts.

By participating in the contest, we learned about the high integrity of SOPC Builder, the stability of the Nios IDE, and the convenience of system debugging. The SOPC-based reconfigurable hardware and software design and top-down design method provided a flexible system implementation, higher system integrity, and a shorter design cycle.

Finally, thanks to Altera for giving us this rare opportunity to create the design.

Communications Applications

This section includes projects that can be used as communications applications, including:

SOPC-Based Cordless Phone 59

Nios II-Based Intellectual Property Camera Design 87

First Prize

SOPC-Based Cordless Phone

Institution: National Institute of Technology, Tiruchy

Participants: Dhirendrakumar Tripathi, P. D. S. Prasad Reddy, Rashmi HM

Instructor: Dr. B. Venkataramani

Design Introduction

A cordless telephone has a base unit and a handset, and electromagnetic radiation allows the two to communicate. Cordless phones have become increasingly popular both at home and in the workplace, and modern cordless phones offer a wide range of features. The primary benefit of the cordless phone is that the handset does not need to be in a fixed location or close to the base unit. Cordless phones allow the user to free him or herself from a wired connection to the local telephone line and move about freely.

Most cordless phones come with a built-in speaker phone. Color screens, like those in mobile phones, have become common in cordless phones as well. These phones support a variety of ring tones and even wallpapers and screen savers. Cordless phones can also be used to send and receive short message service (SMS) messages.

Our design provides a digital implementation of a cordless phone operating in the range of 43 to 50 MHz in a Nios® II processor. We used the Nios II processor to build a system-on-a-programmable-chip (SOPC)-based cordless phone that supports real-time applications. The design enables easy reconfiguration and low development costs. Altera® SOPC designs let us implement real-time, critical functions in hardware. Custom instructions make it easy to implement the whole system on an SOPC platform, with better software partitioning and hardware implementation of the cordless phone.

Design Purpose

In this project, we digitally implemented a cordless phone operating in the 43 to 50 MHz range. So far, cordless phones operating in this range have only been implemented using analog techniques. We implemented a significant portion of the cordless phone design using an FPGA. A spread spectrum technique provides security and isolation between cordless phones. The complete digital transmitter

and receiver are implemented on an FPGA and interfaced with the external world using analog-to-digital (A/D) and digital-to-analog (D/A) converters.

Application Scope

The application is a cordless phone that provides enhanced security and advanced features such as an address book, ring tones, answering machine, etc. at an affordable price. With small modifications, the same design could be used for secure military communication applications. Low-cost, high-density, low-power Altera FPGAs could be used to mass produce these phones at an affordable price.

Nios II Processor and SOPC Builder Role

A cordless phone system commonly includes a programmable embedded processor that controls the phone's operation. In addition to the real-time requirement, the system also requires interfaces for the keyboard and display. The Nios II processor is useful for incorporating these features because all non-real-time tasks can be implemented efficiently in software. Furthermore, we can configure the number of I/O ports and the size of each port using SOPC Builder. Given these parameters, we implemented the digital transmitter and receiver blocks as custom instructions in the Nios II processor.

The Nios II processor offers embedded design versatility and reconfigurability to implement all requirements of the cordless phone. It has built-in memory, peripherals, and interfaces, as well as a variety of intellectual property (IP) functions. Productivity tools--such as SOPC Builder, the Nios II Integrated Development Environment (IDE), and Quartus II Compiler combined with an FPGA target device, allowed us to reduce our turnaround time and costs.

The Nios II processor is a perfect fit because we could select the peripheral, performance, processor variant, and cost that best suits to our needs. The Nios II processor supports multi-processor systems, and using SOPC Builder we could implement multiple processor cores if our design required it. We can efficiently implement a complete system when we integrate high-density FPGAs with high-capacity RAMs and the Nios II processor. High-bandwidth memories, embedded digital signal processing (DSP) blocks, phase-locked loops (PLLs), and high-speed interfaces can be programmed into the FPGA, which facilitated the cordless phone implementation.

Function Description

The SOPC-based cordless phone consists of two primary digital parts, the FM modulator/demodulator and frequency hopping spread spectrum (FHSS) modulator/demodulator.

FM Modulator/Demodulator

The SOPC-based cordless phone uses a radio frequency link to transmit/receive voice signals between the base unit and the remote hand set. In this design, voice signal frequency modulation is performed in the digital domain using the coordinate rotation digital computer (CORDIC) algorithm. The system demodulates the FM signal in the digital domain using the CORDIC algorithm and a special sampling technique.

The following sections provide a brief explanation of the CORDIC algorithm and the special sampling scheme. For more details on these topics refer to "References" on page 80.

CORDIC Algorithm

The CORDIC algorithm uses shifts and adds to compute a wide range of functions, including trigonometric, hyperbolic, linear, and logarithmic functions. The CORDIC algorithm is used in diverse applications such as mathematical co-processor units, calculators, waveform generators, and digital modems. The CORDIC algorithm uses shifts and adds to perform vector rotations iteratively. In rotation mode, the CORDIC converts one vector in rectangular form to another vector in rectangular form. In vector mode, it converts a vector in rectangular form to polar form.

CORDIC Rotation Mode

The CORDIC algorithm for rotation mode is derived from the general rotation transform:

$$x_{fin} = x_{in}\cos\Theta - y_{in}\sin\Theta \quad (1)$$

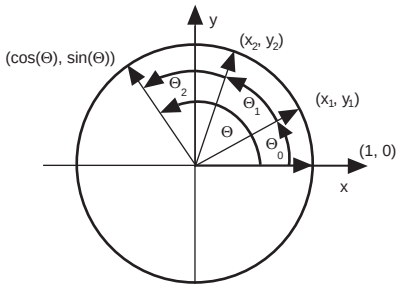
$$y_{fin} = y_{in}\cos\Theta + x_{in}\sin\Theta \quad (2)$$

The transform rotates a vector (x_{in}, y_{in}) in a Cartesian plane by an angle $\Theta$to another vector with the coordinates (x_{fin}, yf_{in}). Rotation is achieved by performing a series of successively smaller elementary rotations Θ_0, Θ_1, Θ_2... Θ_N such that:

$$\Theta = \sum_{0}^{N} \Theta_i$$

Figure 1 shows the case where rotation of a vector of magnitude 1 by an angle Θ is achieved using three elementary rotations Θ_0, Θ_1, and Θ_2.

Figure 1. Vector Rotation by an Angle Θ_i Using Steps



Rotation of the vector by an angle Θ_i can be rewritten as:

$$x_{i+1} = x_i\cos\Theta_i - y_i\sin\Theta_i \quad (3)$$

$$y_{i+1} = y_i\cos\Theta_i - x_i\sin\Theta_i \quad (4)$$

$$x_{i+1}/\cos\Theta_i = x_i - y_i\tan\Theta_i \quad (5)$$

$$y_{i+1}/\cos\Theta_i = y_i - x_i\tan\Theta_i \quad (6)$$

The computational complexity of equations (5) and (6) can be reduced by rewriting them as:

$$x_{i+1} = x_i - y_i\tan\Theta_i \quad (7)$$

$$y_{i+1} = y_i - x_i\tan\Theta_i \quad (8)$$

$$(x_{fin}, y_{fin}) = \left(\frac{x_N}{\prod_{1}^{N}\cos\Theta_i}, \frac{y_N}{\prod_{1}^{N}\cos\Theta_i}\right) \quad (9)$$

and performing the division by $\cos\Theta_i$ together for all N iterations by dividing the value of (x_N,y_N) by $\prod_{1}^{N}\cos\Theta_i$.

Further, the value of Θ_i for i = 1, 2 … N is chosen such that $\tan\Theta_i$ is 2^{-i}. Table 1 shows the values of the angles for i = 0 to 9.

Table 1. Values of Θ_i $\tan^{-1}(2^{-i})$

i	Θ_i	$\tan\Theta_i$
0	45	1
1	26.5	0.5
2	14	0.25
3	7.1	0.125
4	3.57	0.0625
5	1.78	0.03125

This process reduces the multiplication by the $\tan\Theta_i$ to a simple shift operation. As the iteration increases, Θ_i becomes smaller and smaller. When the difference between $\Theta$and the sum of Θ_i from 1 to N becomes very small for some value of N, the iteration terminates. The remaining angle by which the vector must be rotated after the completion of i iterations is indicated by the parameter z_{i+1} as given in equation (10).

$$z_{i+1} = z_i - \Theta_i \quad (10)$$

$$z_0 = \Theta_0 \quad (11)$$

Θ_i is considered to be positive when the rotation required is counter-clockwise and negative otherwise. To approximate an arbitrary angle using Θ_i of the form $\tan^{-1}(2^{-i})$, Θ_i may be negative for some values of i.

The sign (sgn) of z_i indicates whether, in the next iteration, the rotation should be counter-clockwise or clockwise. Because, $\tan\Theta_i$ is $+2^{-i}$ when Θ_i is positive and -2^{-i} otherwise, the iterative equations may be rewritten as:

$$\delta_i = \text{sgn}(z_i) \quad (12)$$

$$x_{i+1} = x_i - \delta_i y_i 2^{-i} \quad (13)$$

$$y_{i+1} = y_i + \delta_i x_i 2^{-i} \quad (14)$$

$$z_{i+1} = z_1 - \delta_i \tan^{-1}(2^{-i}) \quad (15)$$

The computation of $\prod_{1}^{N}\cos\Theta_i$ can be simplified.

Because $\cos\Theta_i = 1$ for very small values of Θ_i, the equation can be computed for N = 6 (therefore K = 0.6073), and can be used for any other value of N > 6.

CORDIC Vector Mode

In this mode, an initial vector with the x, y coordinates of (x_{in}, y_{in}) is rotated such that its y coordinate becomes zero. The procedure used for rotation may be adopted for vector mode with the following modifications: rotation is carried out clockwise (so that the y coordinate can be made 0), and the total angle by which the vector has been rotated from the initial position after i rotations is indicated by the parameter z_{i+1} and z_0 is defined as 0. See equations 16 through 19. To obtain equation 16, when y_i is positive, the rotation is clockwise in the next iteration. When it is negative, it is counter-clockwise.

$$\delta_i = -\text{sgn}\,(y_i) \quad (16)$$

$$x_{i+1} = x_i - \delta_i\, y_i\, 2^{-i} \quad (17)$$

$$y_{i+1} = y_i + \delta_i\, x_i\, 2^{-i} \quad (18)$$

$$z_{i+1} = z_i - \delta_i \tan^{-1}(2^{-i}) \quad (19)$$

As i becomes large, y_i goes to 0 and x_{fin}, the magnitude of the vector after N iterations and z_{fin}, the angle of the vector, are obtained as:

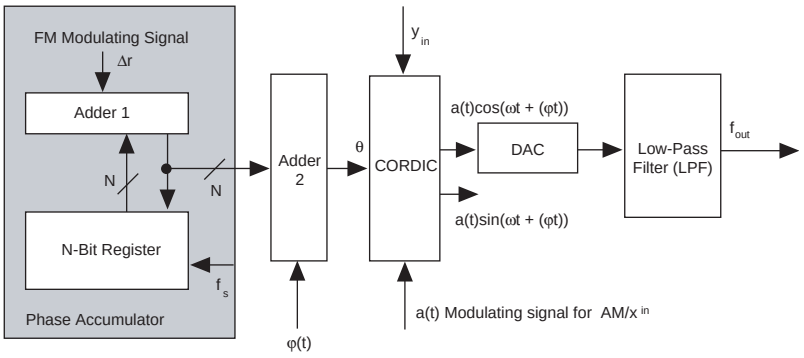
$$x_{fin} = \frac{x_N}{\prod_{1}^{N} \cos\Theta_i} \quad (20)$$

$$z_{fin} = \tan^{-1}\left(\frac{y_0}{x_0}\right) \quad (21)$$

CORDIC as a Universal Modulator

In this design, we used the CORDIC as a universal modulator, which can be obtained by including an adder at the output of the phase accumulator. Figure 2 shows the resulting circuit. (x_{in}, y_{in}) denote the inputs for amplitude modulation. φ(t) and Δ_r denote the inputs for phase and frequency modulation, respectively. When (x_{in}, y_{in}, φ(t)) are (1, 0, 0) and Δ_r is a constant (as shown in Figure 2) the CORDIC generates two unmodulated carrier signals in quadrature (i.e., sinωt and cosωt). When modulating signals are applied, modulated carriers are obtained. When amplitude modulation is required, the modulating input is applied to the x_{in} input and y_{in} is zero. For generating the carrier with phased, frequency modulation, the modulating inputs are fed to φ(t) and Δ_r, respectively.

Figure 2. CORDIC as a Universal Modulator



CORDIC as a Universal Demodulator

If the carrier frequency, amplitude, and phase of the received signal are f_i, 2b(t) and Θ(t) respectively, then the received signal r(t), is given by

$$r(t) = 2b(t)\sin(\omega_i t + \Theta(t)) \quad (22)$$

One way to demodulate this signal is by generating the in-phase signal I(t) and quadrature signal Q(t) using a local oscillator of frequency f_0 with a known initial phase and equations 23 and 24.

$$I(t) = b(t)\{\sin(2\pi(f_i - f_0)t + \Theta(t))\} \quad (23)$$

$$Q(t) = b(t)\{\cos(2\pi(f_i - f_0)t + \Theta(t))\} \quad (24)$$

The in-phase and quadrature signals are fed to the two inputs of the CORDIC, which is operated in vector mode, to obtain the demodulated output.

I and Q Signal Generation Using a Special Sampling Scheme

The I and Q components can be generated using a quadrature mixer or a special sampling scheme. With the special sampling scheme, if the local oscillator frequency is f_0, r(t) is sampled at a rate of $f_S = 4f_0$ then

$$t = nT_S = n/4f_0 \quad (29)$$

$$2\pi tf_0 = 2\pi(nT_S)f_0 = 2n\pi/4 = n\pi/2 \quad (30)$$

Let us denote the values of r(t), I(t), and Q(t) at $t = nT_S$ as r(n), I(n), and Q(n), respectively. Then I(n) and Q(n) can be written as:

I(n) = r(0), 0, -r(2), 0, r(4), ….

Q(n) = 0, r(1), 0, -r(3), 0, r(5), ….

If the local oscillator frequency f_0 is chosen to be the same as the input frequency f_i, then f_S can be chosen as $4f_0$. Table 2 shows r(t), I(t), and Q(t) at various sampling instants. For more details on the special sampling technique refer to reference [2]. Figure 3 shows a block diagram of the I and Q generation using the special sampling scheme.

Figure 3. I and Q Signal Generation Using Special Sampling Scheme

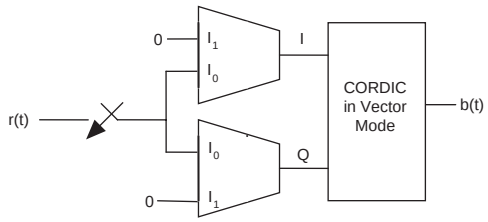


Table 2. Received Signal Samples for Different Instants

Sample at $t = t_n = nT_S$ for n =	0	1	2	3	4
R(t)	b(0)sin(Θ)	$b(t_1)\cos(\Theta)$	$-b(t_2)\sin(\Theta)$	$-b(t_3)\cos(\Theta)$	$b(t_4)\sin(\Theta)$
V_{if1}	b(0)sin(Θ)	0	$-b(t_2)\sin(\Theta)$,	0	$b(t_4)\sin(\Theta)$
V_{if2}	0	$b(t_1)\cos(\Theta)$,	0	-b(t3)cos(Θ)	0

With this method we can generate the in-phase and quadrature components without using mixers at the cost of a higher sampling rate.

FHSS Modulator/Demodulator

The FHSS technique, which changes the carrier frequency randomly with time, provides cordless phone communication with enhanced security and privacy. Transmitted signals that have a carrier frequency hopped by a pseudo-random code are difficult to demodulate by receivers other than the intended one. The technique makes the system less vulnerable to accidental or deliberate reception by a third party and protects privacy. We used the CORDIC as the frequency synthesizer for FHSS modulation and demodulation.

FHSS

Our cordless phone operating in the 43- to 50-MHz band is allocated 25 sets of frequency sub-bands. A cordless phone selects a set of frequency bands (one for the base unit to the handset and another for the handset to the base unit) from these 25 sets and operates at this fixed set of frequencies. If two cordless phones in the vicinity have the same frequency set, they could interfere with each other or cause security threats.

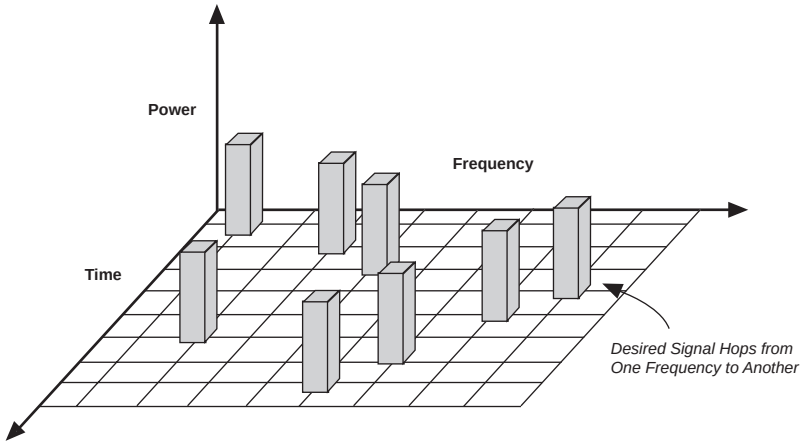
To solve these problems, we used FHSS. In a frequency hopping system, the frequency or channel in use is changed rapidly. The transmitter hops from one channel to another in a pre-determined, pseudo-random manner. The receiver has the same channel list (the hop set) and pseudo-random sequence generator. Therefore, the cordless phone only stays in a particular channel for a short time, lowering the probability of interference and security threats.

In FHSS, the frequencies used in the hopping sequence can be selected by the user. To “tune in,” a listener must know the number of frequencies selected in the system, the actual frequencies, the order in which these frequencies are used (the hopping sequence), as well as the dwell time (time for which a hopping frequency exists). In this way, the FHSS modulation acts as a layer 1 encryption process.

Concerning noise effects, FHSS systems can operate with a signal-to-noise ratio (SNR) of about 18 dB. Frequency modulation, or frequency shift keying (FSK), is often used in frequency hopping systems. However, it is used infrequently in direct sequence systems because when a direct sequence signal passes through a squaring or frequency doubling circuit, a carrier at twice the signal’s center frequency is produced. This twice frequency narrowband carrier contains any modulation impressed on the direct sequence signal. Thus, with analog modulation it is possible for the signal to be demodulated without any prior knowledge of the pseudo-random spreading code.

Figure 4 shows the frequency hopping process. The carrier signal is at different frequencies at different times while retaining the same transmitting power.

Figure 4. Frequency Hopping



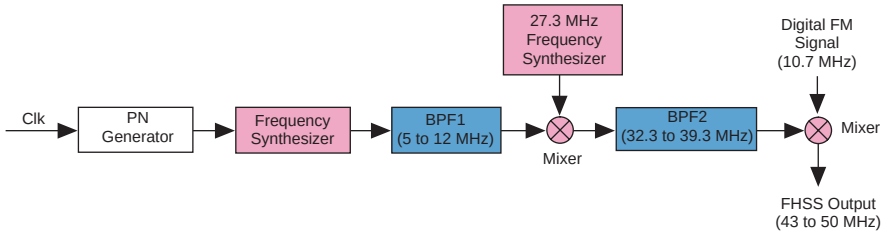
FHSS Modulator

An FHSS modulator (see Figure 5) is the integral element of the cordless phone transmitter. It contains one pseudo-noise (PN) sequence generator, which generates a pseudo-random code. A 6-bit PN sequence generator generates 64 hopping frequencies. The characteristic equation and the seed value of the PN sequence generator decide the output pseudo random number sequence. The PN sequence generator output is fed to a frequency synthesizer, which generates 64 frequencies in a 5- to 12-MHz band in a pseudo-random fashion. See references [4] and [5] on page 80 for more details.

The CORDIC algorithm implements the frequency synthesizer. We used a slow-frequency hopping scheme in our project. In this scheme, a particular carrier frequency exists for more than one data symbol duration. The hopped frequency pattern is up-converted to the 32.3- to 39.3-MHz frequency range so that the final output frequencies of the cordless phone are in the range of 43 to 50 MHz.

See "Design Description" on page 71 for the Simulink and Quartus® II implementation details.

Figure 5. FHSS Modulator



Key:
Light blue blocks were designed using Altera IP blocks.
White blocks were designed by us.
Pink blocks were implemented by instantiating Altera IP blocks.

FHSS Receiver

At the receiver, the FHSS demodulator needs a synchronizing circuit, which ensures that the receiver's pseudo-random code generator is synchronized with the transmitter. When the transmitter and receiver are synchronized, the user is unaware that the transmitter and the receiver are rapidly changing carrier

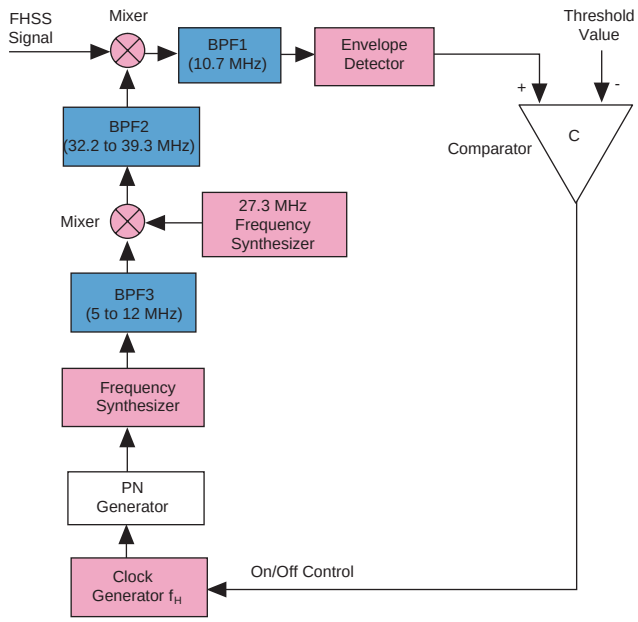
frequencies. The following sections describe the synchronization scheme implementation (see references [3] and [4] for more information).

FHSS Synchronization

The synchronization circuit (see Figure 6) has the following elements:

- Mixer
- Bandpass filter centered at the FM intermediate frequency (10.7 MHz) with a bandwidth that has twice the spacing between any two adjacent frequency channels in the hopped spectrum $2f_h$ where f_h = the available bandwidth/number of hopping channels. In our project, $2f_h$ is equal to 2 x (12 - 5)/64 MHz or 0.21875 MHz.
- Envelope detector
- Comparator
- PN generator
- Frequency synthesizer
- Clock generator

Figure 6. FHSS Demodulator and Synchronization Circuit



Key:
Light blue blocks were designed using Altera IP blocks.
White blocks were designed by us.
Pink blocks were implemented by instantiating Altera IP blocks.

We used the special sampling technique and CORDIC algorithm to implement the digital envelope detector. We implemented the frequency synthesizer digitally using a direct digital frequency synthesizer (DDFS) and the CORDIC scheme.

For insight into the functionality of synchronization circuitry and the FHSS demodulator, assume that the receiver has a carrier frequency that does not match the transmitted frequency. In this case, the mixer does not generate the desired modulated signal centered on f_{IF} (i.e., 10.7 MHz) and the narrow bandpass filter's output amplitude (BPF1) is very low, resulting in a low amplitude at the envelope detector's output.

The comparator compares the output of envelope detector with a threshold value. When the detector output is less than the threshold value, the comparator produces a zero output. The comparator output works as an on/off control for the PN sequence generator's clock generator. If the on/off control output is zero, the PN generator output is fixed at the seed value. This setup causes the receiver's frequency synthesizer to generate a fixed frequency until the PN generator output changes.

If the incoming frequency matches the receiver's synthesizer frequency, the mixer generates the modulated signal centered on f_{IF} (or 10.7 MHz). Then the BPF1 output and the envelope detector are greater than the threshold voltage of the comparator, which in turn generates a high output. The comparator's high output enables the receiver PN generator clock, and afterwards the receiver frequency matches the transmitter frequency and full synchronization is maintained.

Synchronization Challenges

FHSS synchronization has many technical design issues, specifically:

- *Threshold voltage*—Deciding the comparator's threshold voltage is complicated. There is no algorithm for deciding the threshold voltage, and the threshold voltage is fixed after performing several iterations.

- *Synchronization latency*—Our scheme uses a serial synchronization architecture, which uses the least area on the FPGA. The design could implement more than one block with each block having a different seed value and expecting a different transmitted frequency. Searching for the transmitted frequency in parallel reduces the synchronization latency.

Nios II Processor Advantages

Using the Nios II processor gave us the following advantages:

- The customizable Nios II processor provides high performance and supports flexible, low-cost product development.

- The Nios II processor's customizable instruction set accommodates complicated arithmetic operations. Additionally, it accelerates algorithm processing, which provides faster execution than implementing these operations in software.

- A configurable design enhances system performance and allows us to upgrade the hardware and software on-site easily. Additionally, it helps prevent our system from becoming obsolete.

- The Nios II IDE enables phased design implementation. We can implement system blocks separately, integrate them with the Nios II processor, and test them on the chip. This design style provides a controlled development environment.

- Implementing the processor, peripherals, memory, and I/O interface in a single FPGA reduces the total system cost.

- We can perform rapid prototyping and final system implementation of the original design concept in a short time with the Nios II processor.

- The Nios II processor can be customized and reconfigured. For example, it supports three processor cores, peripherals, the Avalon switch fabric, custom instructions, and hardware acceleration. All of these functions can be implemented using commonly available Altera FPGAs.

- Using IP optimized for the FPGA architecture, we can easily redesign standard functions, rapidly customize hardware peripherals, focus on design partitioning, and improve our design knowledge.

- Integrated development kits, the Quartus II software, SOPC Builder, and the SignalTap® II embedded logic analyzer provide a complete set of test and debug tools for hardware design. With the Nios II IDE, we can simplify the software design and development tasks, such as program editing and debugging.

Performance Parameters

To simplify the design, we preferred to use a development board that has an FPGA with a large number of logic elements (LEs), an on-board analog-to-digital converter (ADC), and on-board digital-to-analog converter (DAC). Therefore, we implemented the design using the Stratix® II development board, which features the Stratix II EP2S60F1020C4 FPGA. Using this FPGA avoids area concerns because Stratix II devices are the biggest FPGAs available in the market. According to our estimates, this project consumes more than 90% of the LEs in the Cyclone II EP2C35 device, which is on the Development and Education (DE1) board.

Table 3 shows the resource usage for our project using the Stratix II board.

Table 3. Resource Usage

Parameter	Value
Device	EP2S60F1020C4
Tool Used	Quartus II version 6.0
Total ALUTs	25,731 out of 48,352 (53.22%)
Total Memory Bits	426,164 out of 2,544,192 (16.75%)
Total PLL	10 out of 12 (83.33%)
Total Registers	11,790

Design Architecture

This section describes the hardware and software design architecture.

Hardware

Figure 7 shows the cordless phone's general transmitter architecture (analog components are shown in lavender). The ADC receives the input from the microphone, which is converted into a digital signal. This digital signal frequency modulates a carrier of 10.7 MHz using a digital FM modulator. The FM modulator output is converted into a serial data stream and is fed as the input to the digital FHSS modulator (for details see "FHSS Modulator/Demodulator" on page 65). The signal is up-converted to the 43- to 50-MHz range and fed to the antenna after amplification by the power amplifier. In our project, the ADC, DAC, power amplifier, BPF, and microphone are analog components. The FM modulator and FHSS modulator are completely implemented as a digital system in the FPGA.

Figure 7. Cordless Phone Transmitter

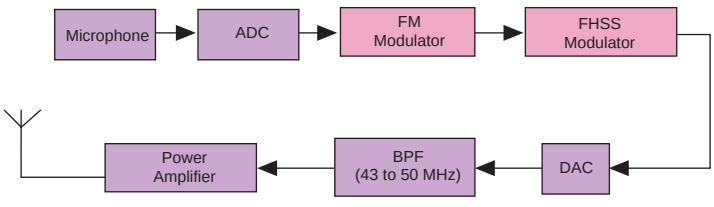
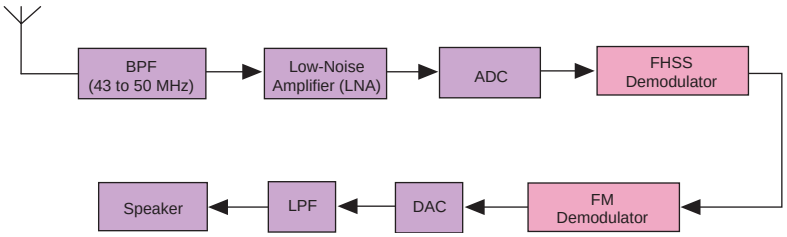


Figure 8 shows the cordless phone receiver. The received signal is passed through a bandpass filter (BPF) (43 to 50 MHz), whose output is fed to a low-noise amplifier. The signal is digitized using an ADC. The ADC output is fed to a digital FHSS demodulator, which has frequency down conversion and synchronization blocks. When there is perfect synchronization between the transmitter and receiver, the FM modulator activates. The FHSS demodulator output is an FM signal centered around 10.7 MHz. This signal is demodulated using a digital FM demodulator and is fed to the DAC. After conversion of the digital signal into analog, the DAC output is fed to a low-pass filter (LPF) with a 3.4-kHz cutoff frequency and finally fed to the speaker.

Figure 8. Cordless Phone Receiver



Software Flow Chart

Figures 9 and 10 show the software flow charts for the cordless phone's transmitter and receiver, respectively. The whole transceiver can be called using a custom instruction in the Nios II processor. Using software, we can change the seed value of PN generators and the modulation technique to quadrature phase shift keying (QPSK). However, our design is currently focused on the FM/FHSS modulation technique.

Figure 9. Transmitter Flow Chart

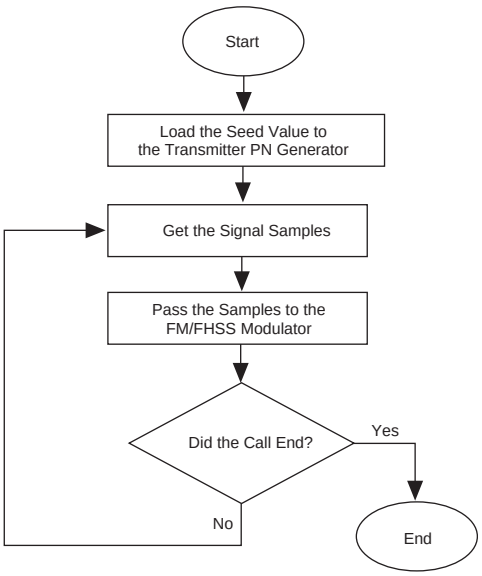
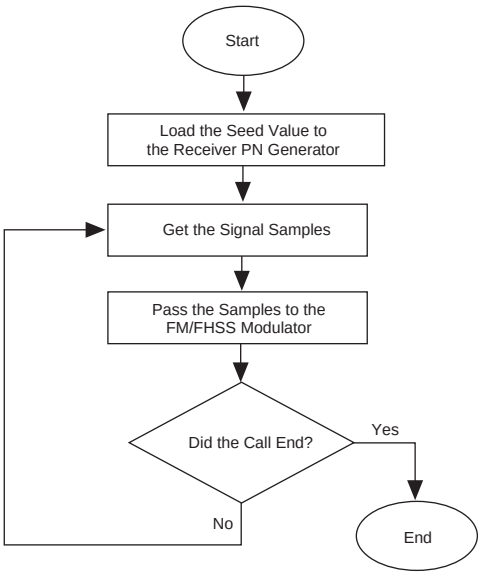


Figure 10. Receiver Flow Chart



Design Description

We implemented the SOPC-based cordless phone project in 5 phases:

1. Modeling and conceptualization of digital FM generation/demodulation.
2. Modeling and conceptualization of digital FHSS modulation/demodulation.
3. Designing cordless phone transmitter (FM generation and FHSS modulation).
4. Designing cordless phone receiver (FM and FHSS demodulation).
5. Creating the whole system as a Nios II custom logic block.

Phase 1 and 2 Implementation

We implemented phases 1 and 2 using The MathWorks MATLAB software and Altera's DSP Builder, which links the MATLAB and Simulink software with the Quartus II software. In this phase, we used Altera-provided DSP Builder blocks and we imported our Verilog HDL design entities into the Simulink model file using HDL import blocks. This method let us quickly prototype the design and verify the design concepts. Refer to "Appendix" on page 80 for the Simulink block diagrams.

Figure 11 shows the FM generation simulation result in Simulink. We used the CORDIC as the universal modulator for generating the FM wave. The message signal is a 3,300-Hz sinusoidal wave. The FM signal has a center frequency of 10.7 MHz and a frequency deviation of 75 kHz.

Figure 11. FM Generation

Channel 1 is the message signal, channel 2 is the FM wave

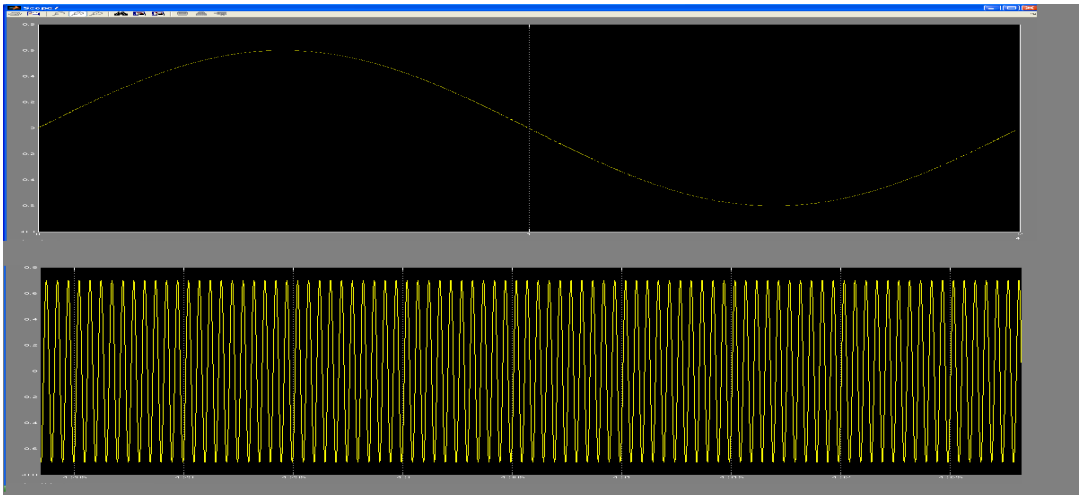


Figure 12 shows the result of FM demodulation using the special sampling technique. As shown in the figure, after a small delay, the final demodulated wave has the same period but different amplitude.

Figure 12. FM Demodulation using CORDIC and Special Sampling Theorem

Channel 1 is the message signal, channel 2 is the FM signal, and channel 3 is the FM demodulated signal using the special sampling theorem

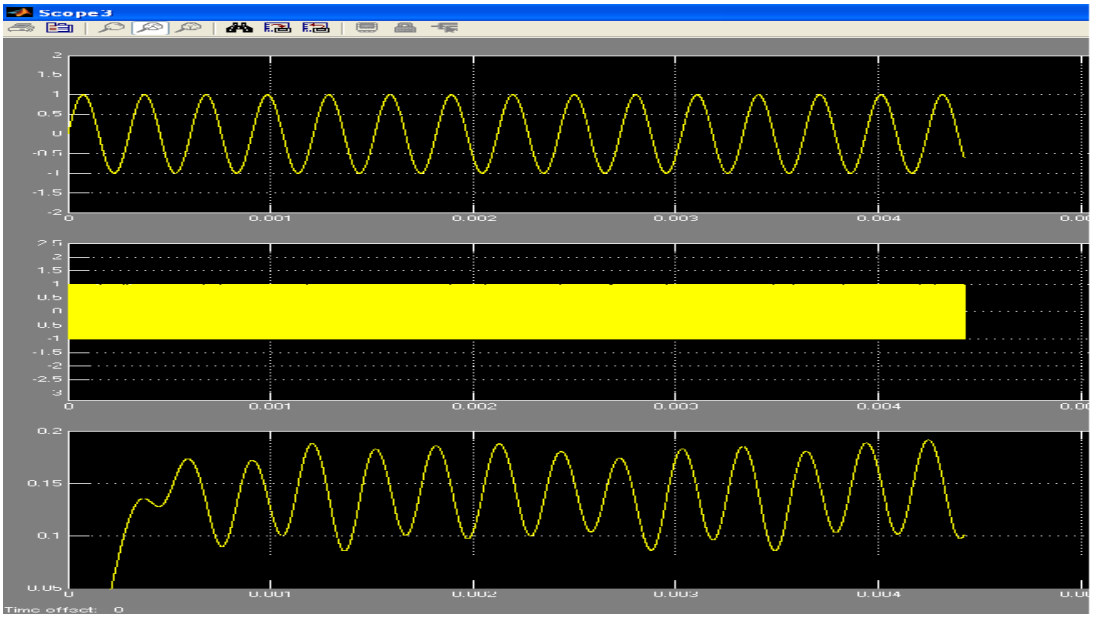


Figure 13 shows output of the 6-bit PN generator with a count from 1 to 63. We implemented the PN generator using the HDL import block. The second channel shows the corresponding frequency generated using direct digital frequency synthesis (DDFS) and CORDIC. As the PN sequence generator output changes, different frequency components can be obtained.

Figure 13. PN Generator Output (Channel 1) and FHSS Signal (5 to 12 MHz)

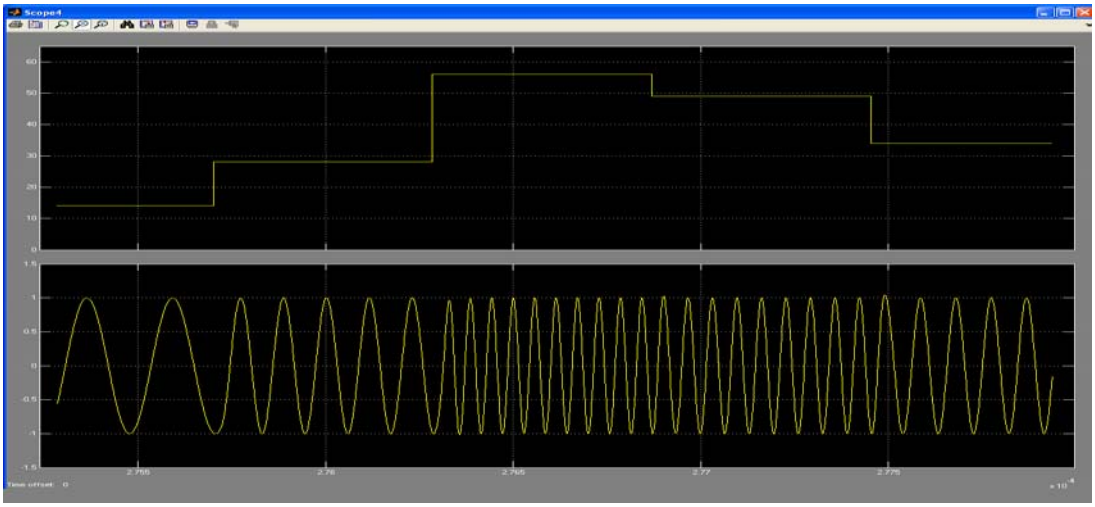


Figure 14 shows the 10.7-MHz FM signal that has been spread to 43 to 50 MHz.

Figure 14. 10.7-MHz FM Signal (Channel 1) and 43- to 50-MHz FHSS Signal (Channel 2)

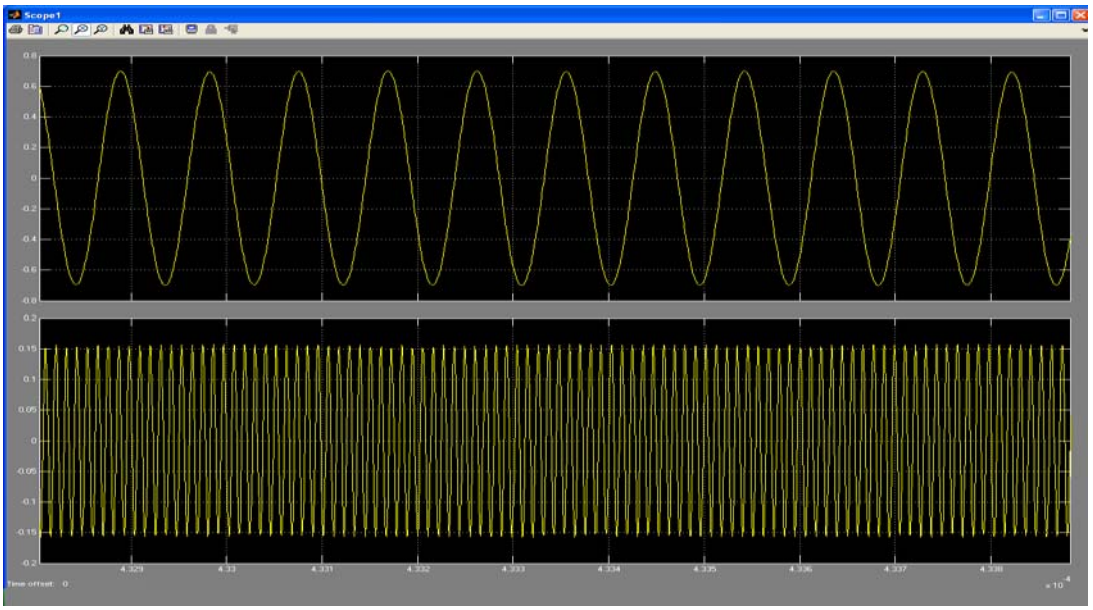
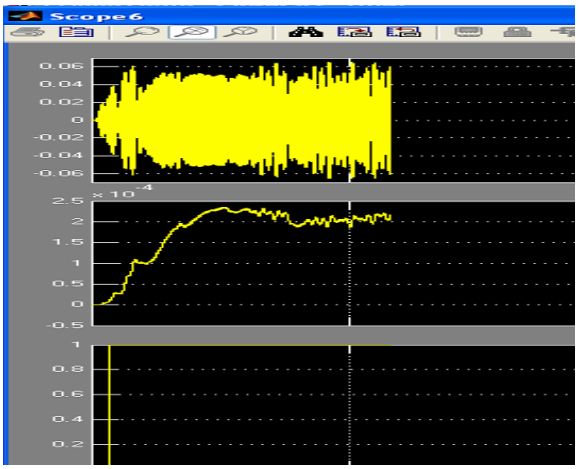


Figure 15 shows the output at various points in the synchronization loop. Channel 1 shows that the BPF output is negligible if the receiver's PN generator sequence is not synchronized with the transmitter. In this case, the receiver's frequency synthesizer output frequency is not equal to the transmitter frequency synthesizer; therefore, the product of the received signal with the receiver frequency synthesizer's output frequency is not centered at 10.7 MHz. The BPF output (centered at 10.7 MHz and a bandwidth of $0.21875(2f_h)$) is negligible. Because the BPF output is negligible, the envelope detector output is negligible.

Figure 15. Synchronization Process

Channel 1 is the BPF output, channel 2 is the envelope detector output, and channel 3 is the threshold detector output going from 0 to 1

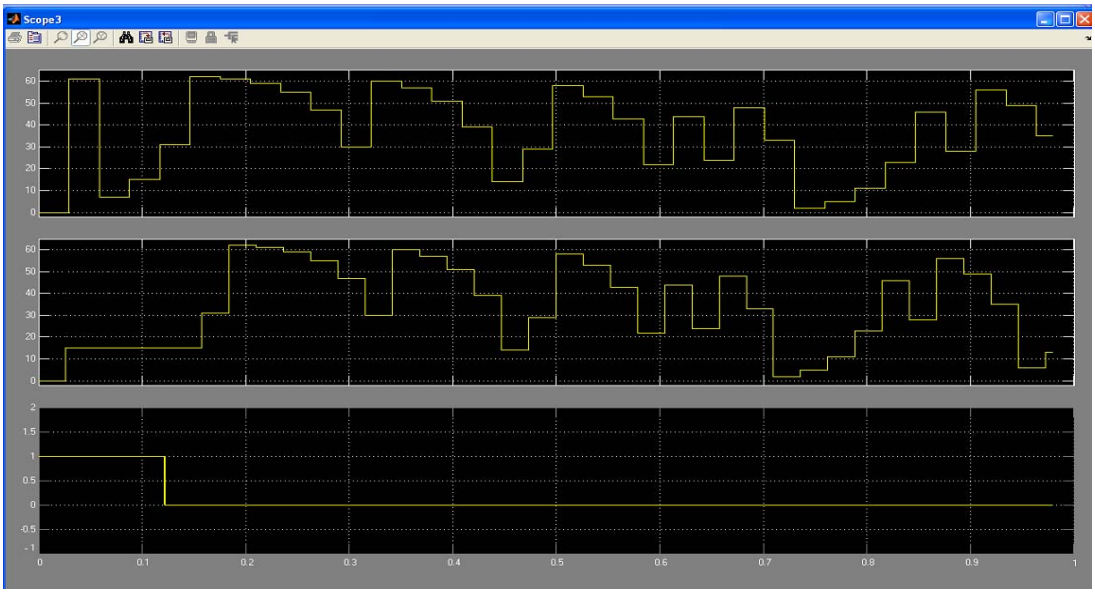


When the receiver's PN sequence generator is synchronized with the transmitter's PN sequence generator, the BPF produces a larger amplitude output. The envelope detector detects the signal's envelope and its output is compared to the threshold value of the comparator (in our Simulink model the threshold value is .8E-5). As soon as the envelope detector output crosses the threshold value, the comparator outputs a 1, which enables the receiver PN generator's clock. Thereafter it remains synchronized with the transmitter PN sequence generator.

Figure 16 shows the synchronization between the transmitter PN sequence generator and receiver PN generator. When they are synchronized, the on/off control (channel 3) goes low and activates a counter in the receiver PN sequence generator. If they are not synchronized, the receiver PN sequence generator remains at its seed value, which is 15 in our Simulink model. We kept the receiver clock a little bit faster to compensate for the inherent delay in the synchronization loop.

Figure 16. Synchronization of the Transmitter and Receiver PN Generators

Channel 1 is the transmitter PN generator, channel 2 is the receiver PN generator, and channel 3 is the clock enable for the receiver PN generator (active-low signal)



Phase 3 and 4 Implementation

In the third and fourth phases, we wrote Verilog HDL code for the digital blocks and testing using real-time signals.

Our design requires two ADCs and DACs. We used the Stratix II DSP development kit, which has two high-end ADCs and DACs with a maximum sampling rate (and number of bits) of 125 million samples per second (MSPS) (corresponding to 12 bits) and 165 MSPS (corresponding to 14 bits), respectively. The on-board FPGA is the EP2S601020C4 device.

We faced the following implementation problems:

- The development board's ADC does not support low-frequency signals, such as voice signals. Therefore, we had to upconvert the voice signal using an AM modulator to the range of hundreds of kHz. This up-converted signal is processed further in the FPGA after it is passed through the ADC. In the actual implementation scheme, the AM modulator must be implemented using an analog circuit. We could use an integrated circuit (IC) such as the MC1496 device for this purpose.

- The Stratix II DSP development board has a 14-bit DAC and a 12-bit ADC. Therefore, we had to reduce the digital filter resolution to either 14 or 12 bits, which affected the final output resolution.

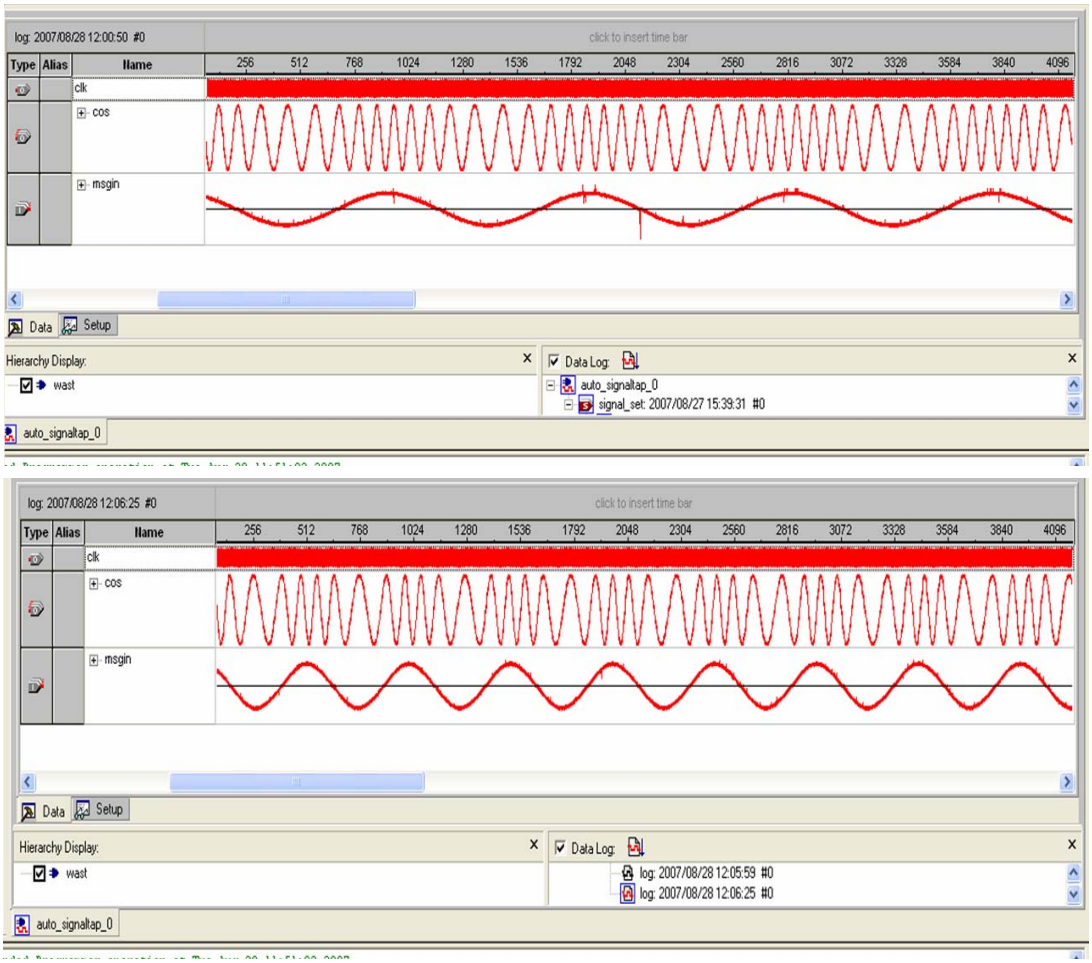
We used the Quartus II SignalTap II logic analyzer to view the signals coming from and into the FPGA on the board. This feature allowed us to debug and verify the design.

For the digital frequency modulator and FHSS modulator design, we used an unrolled pipelined CORDIC as the universal modulator. This CORDIC has five pipelined stages and can generate a higher frequency compared to a simple CORDIC. For FM modulator testing and verification, we fed a high-frequency message signal to the ADC.

Figure 17 shows the FM modulation result captured by the SignalTap II logic analyzer.

Figure 17. 100-kHz Signal Digital FM Generation SignalTap II Logic Analyzer Output

Channel 1 is the ADC clock, channel 2 is the digital frequency modulated wave, and channel 3 is the message signal

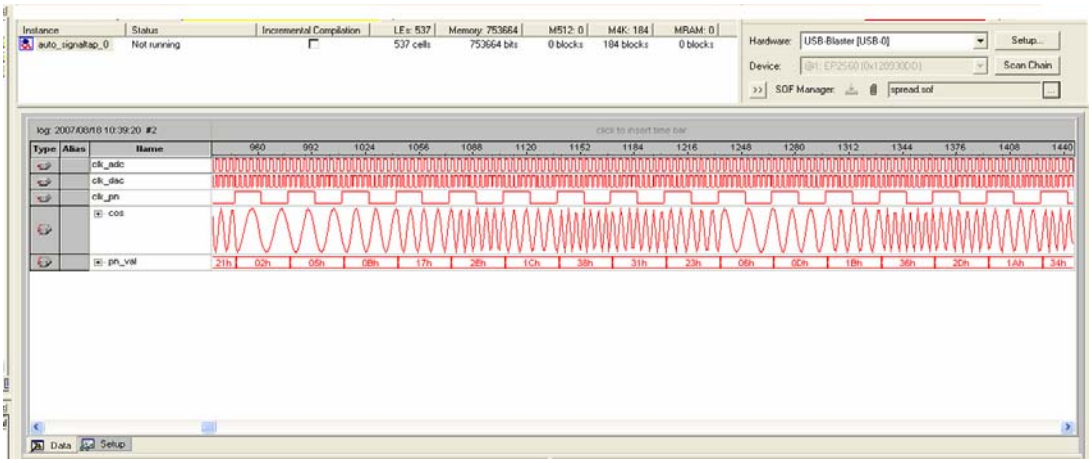


To generate the hopping frequency pattern, we used a digital frequency hopping modulator. The 6-bit PN sequence generator determines the unrolled CORDIC output frequency. This system generates 64 pseudo-random frequencies in the 5- to 12-MHz band.

Figure 18 shows the system output captured the SignalTap II logic analyzer.

Figure 18. Digital FHSS Generation SignalTap II Logic Analyzer Output

Channel 1 is the ADC clock, channel 2 is the DAC clock, channel 3 is the digital frequency hopped wave, and channel 4 is the PN generator



As mentioned previously, FHSS synchronization requires a narrow-band BPF (BPF1 in Figure 6 on page 67) that is centered at an FM intermediate frequency (i.e., 10.7 MHz) and has a bandwidth twice the hopped frequency ($2f_h$). We generated the BPF using the Altera-provided FIR Compiler MegaCore® function. The BPF has 100 taps and is implemented as a distributed arithmetic full parallel filter with level 1 pipelining. It uses signed 2's complement arithmetic. The Stratix II DSP development board has a 12-bit ADC. To feed the BPF output, we truncated the full resolution filter output to 12 bits. Figure 19 shows the output of this filter when the 10.7-MHz signal is applied (as tapped by the SignalTap II logic analyzer).

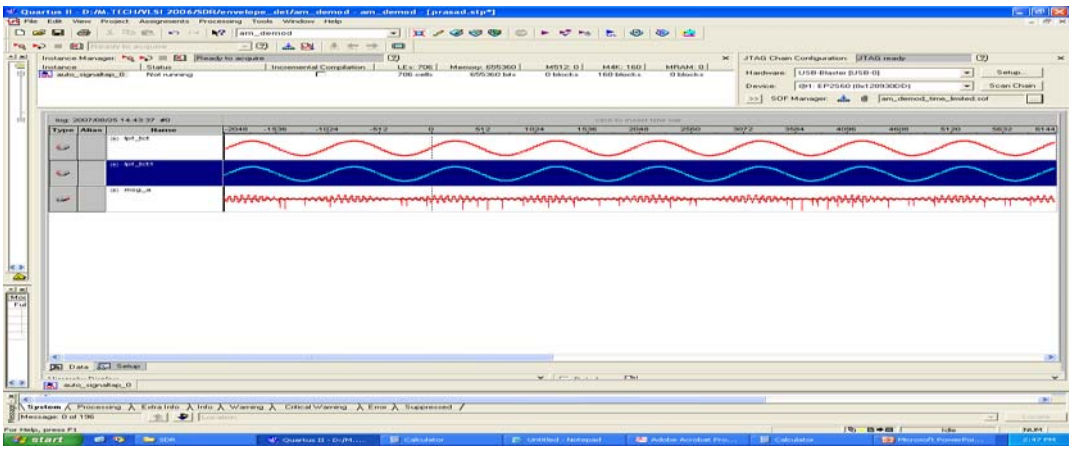
Figure 19. Cordless Phone Receiver's Narrow-Band BPF Output

Channel 1 is the full resolution BPF output, channel 2 is the 10.7-MHz signal, channel 3 is the DAC clock, and channel 4 is the 12-bit BPF output

As shown in Figure 6 on page 67, an envelope detector is an essential part of the synchronization loop. The envelope detector is simply an AM demodulator. We implemented the envelope detector using the special sampling theorem and CORDIC algorithm. Figure 20 shows the envelope detector output when the AM signal is applied (as tapped by the SignalTap II logic analyzer). The AM signal is generated on the FPGA using a pipelined CORDIC algorithm.

Figure 20. Cordless Phone Receiver's Narrow-Band BPF Output (Envelope Detector)

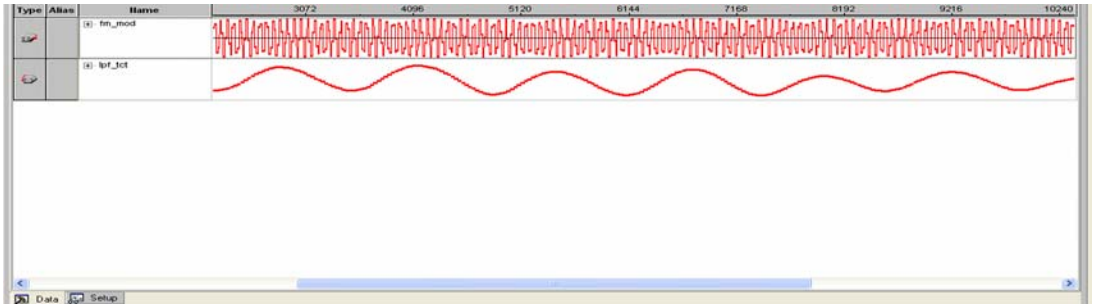
Channel 1 is the full resolution envelope detector output, channel 2 is the envelope detector truncated output, and channel 3 is the AM signal input message to the envelope detector



We built the FM demodulation process using an unrolled, pipelined CORDIC and the special sampling technique. Figure 21 shows the FM demodulator result. The FM modulator generates the FM input, which is fed to the digital FM demodulator.

Figure 21. Digital FM Demodulation

Channel 1 is the FM signal and channel 2 is the 100-kHz FM demodulated signal



To implement the mixers, we proposed implementing the digital AM demodulator using CORDIC. See "CORDIC as a Universal Modulator" on page 63 for the theory of AM modulation using CORDIC.

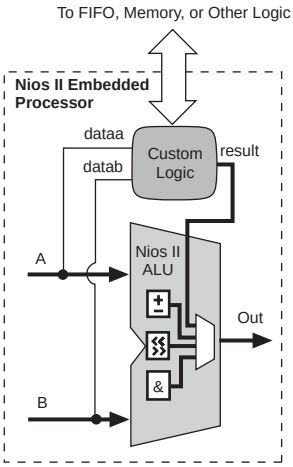
Phase 5 Implementation

The fifth design phase involved using SOPC Builder and the Nios II processor to accelerate our time-critical application by converting our design into a Nios II custom instruction. The custom instruction has two elements:

- *Custom logic block*—This block is the hardware that performs the user-defined operation. The Nios II processor can include up to five user-defined custom logic blocks. Our custom instruction logic is directly connected to the Nios II arithmetic logic unit (ALU) as shown in Figure 22. We designed two custom logic blocks, one for the transmitter and one for the receiver, and placed them on separate FPGAs.

- *Software macro*—The macro allows the system designer to access the custom logic via the software code. We can add 256 custom instructions to the Nios II processor. The transmitter and receiver custom logic blocks can be accessed using custom instructions.

Figure 22. Nios II ALU Custom Instruction Logic



The Nios II processor sets the transmitter and receiver seed values. It can also choose the modulation scheme. We can use the processor to use another digital modulation scheme (such as QPSK) instead of FM and study the performance benefits.

Design Features

Our design has the following features:

- It implements an FM modulator and demodulator for real-time applications using the CORDIC algorithm and an unrolled pipelined CORDIC structure.
- It implements a digital FHSS transmitter and receiver.
- It implements a FHSS receiver and transmitter synchronization scheme.
- Because the Nios II processor is reconfigurable, in the future we can enhance our design to meet real-time and non-real-time application requirements with very low development costs.
- The Nios II processor enables optimum hardware/software development by executing more computationally intensive tasks in hardware and the remaining tasks in software.
- The Nios II embedded processor delivers a good price-to-performance ratio.
- The Nios II processor lowered our technical development hurdles. We can use the Nios II processor to implement the processor, peripherals, memory, and I/O interface on a single FPGA, thereby reducing the total system cost.
- Altera's comprehensive development tools such as DSP Builder, the SignalTap II logic analyzer, and optimized IP functions reduced our software development cost, letting us focus more attention on the cordless phone's design details.

In the future, we would like to enhance the cordless phone in the following ways:

- Using the Nios II embedded processor, the cordless phone could support VoIP and Skype services.
- By upgrading the operation frequency, the cordless phone with the FHSS scheme can be used for secure military communication.
- By using the flexibility of the Nios II processor, we could add many innovative features such as support for a color LCD screen, SMS service, screen savers, and wallpapers.
- Using SOPC Builder concepts, we could enhance the cordless phone base station to handle more than one handset.
- We could implemented the transmitter and receiver on the same FPGA.

Conclusion

During the design process, we learned how the Nios II processor's flexibility and versatility can help us and we learned how the Nios II processor can keep our designs in pace with the emerging industrial requirement trends. Using the Nios II processor, we reduced our development costs, material costs, and turnaround times, which improved our competitiveness. Using Nios II custom instructions provided an added advantage for design customization.

SOPC Builder gave us the exact set of CPUs, peripherals, and interfaces needed for our application. On the whole, SOPC concepts allowed us to create a more flexible, dynamically reconfigurable, and computationally intensive implementation.

The support and guidance we received from the Altera web site helped us complete the implementation in time.

References

[1]Ray Andraka. "A survey of CORDIC algorithms for FPGA based computers," *International Symposium on Field Programmable Gate Arrays 1998 Proceedings*, (1998): pp 191 - 200.

[2] James Tsui. "Digital Techniques for Wide Band Transmission," Artech House Publishers, (1995).

[3] Taub, Herbert and Schilling, Donald L. *Principles of Communication Systems*. New Delhi, Tata McGraw Hill Publication.

[4] Dixon, R.C. *Spread Spectrum Systems*. New York: John Wiley & Sons, Inc., 1976.

Appendix

This appendix provides Simulink models and block diagrams for our design.

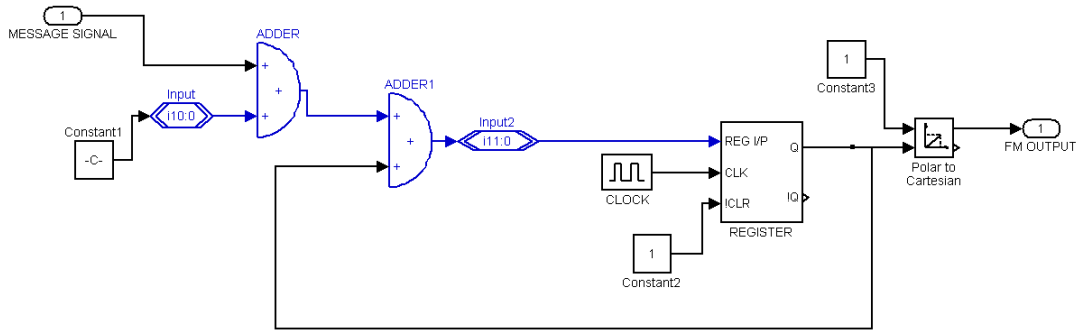
Simulink Models

The following sections provide Simulink models for the modulators and demodulators in the system.

FM Modulator

Figure 23 shows the FM modulator implementation using CORDIC as the universal modulator. A polar-to-cartesian converter acts as the CORDIC. Adders were instantiated from the DSP Builder library.

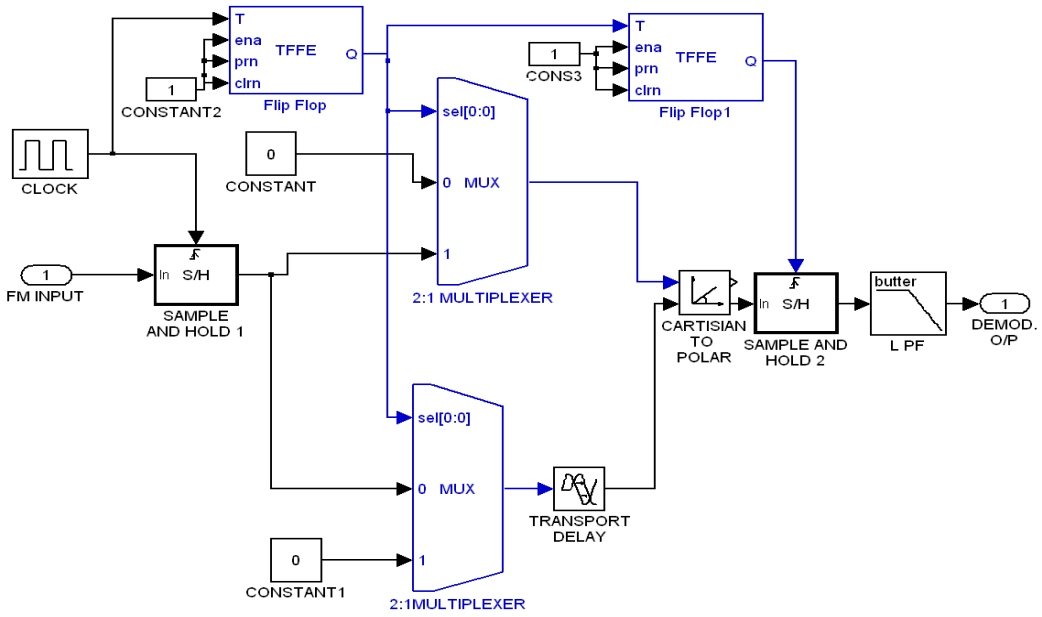
Figure 23. FM Modulator Simulink Model



FM Demodulator

Figure 24 shows the FM demodulator implementation using the special sampling technique (“I and Q Signal Generation Using a Special Sampling Scheme” on page 64).Two T flipflops were used as the frequency divider. The first T flipflop divides the sampling frequency (it was 171.2 MSPS in this design) by two, another divides it by four. We added a transport delay element at the path of Q channel output, which insured that the I and Q channel outputs would be available at the same time.

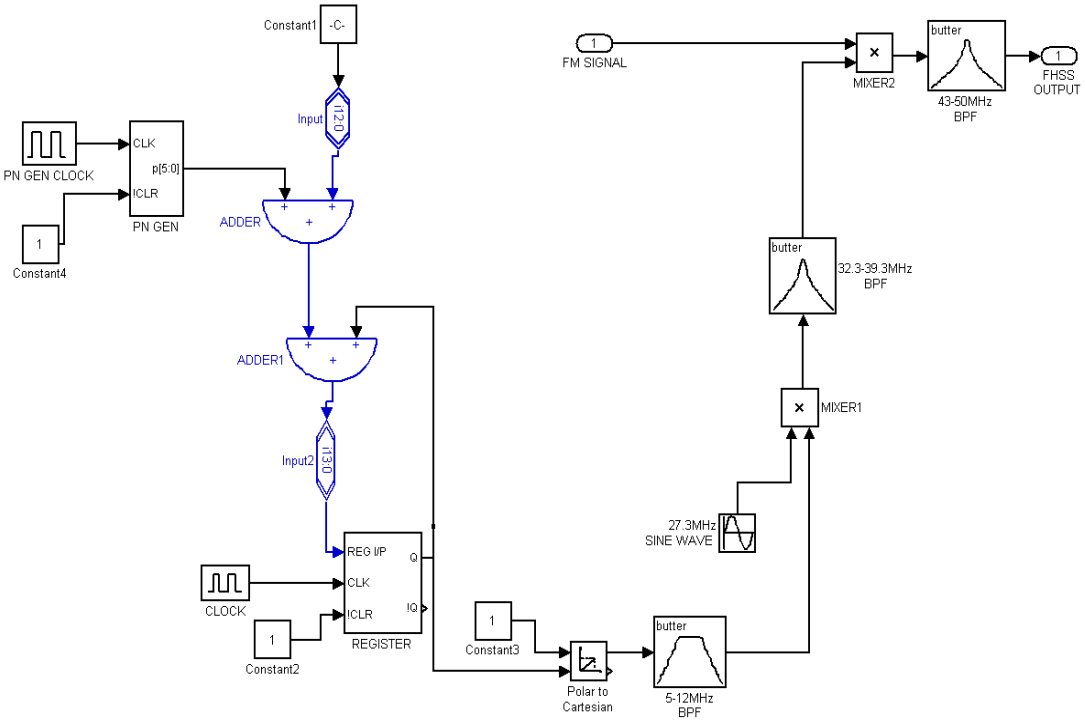
Figure 24. FM Demodulator Simulink Model



FHSS Modulator

Figure 25 shows the FHSS modulation scheme. We used CORDIC as the frequency synthesizer. We instantiated adders from the DSP Builder library. The PN generator contains an HDL import block, which imported HDL code for the PN generator.

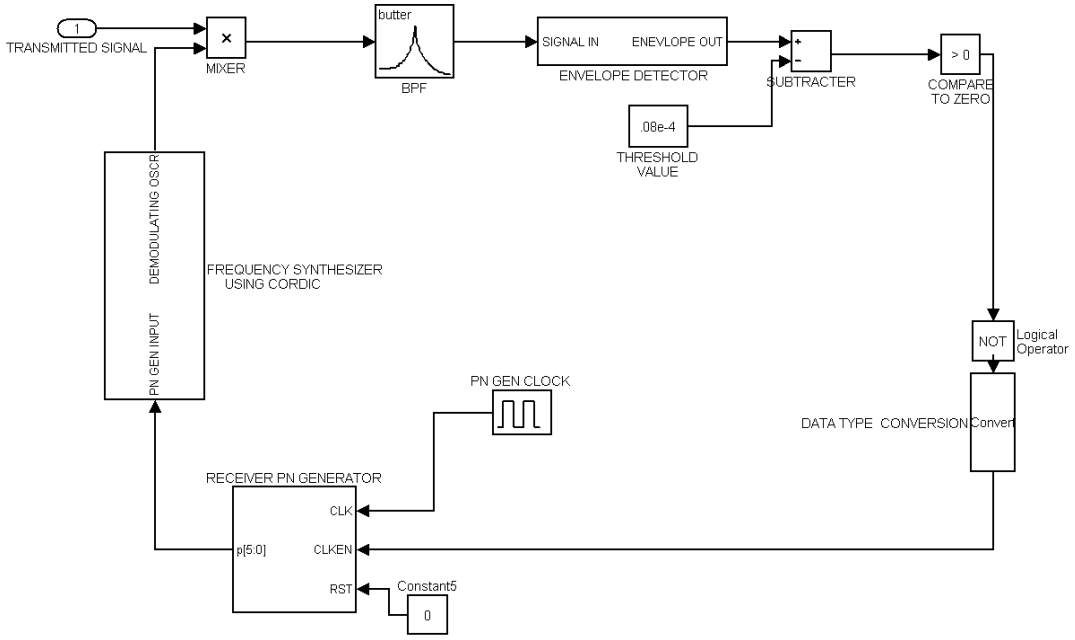
Figure 25. FFHSS Modulator Simulink Model



FHSS Demodulator

Figure 26 shows the FHSS demodulator. The envelope detector was implemented as the AM demodulator. We designed the frequency synthesizer using CORDIC [1]. The receiver PN generator contains an HDL import block, which imported the PN generator HDL code.

Figure 26. FHSS Demodulator Simulink Model



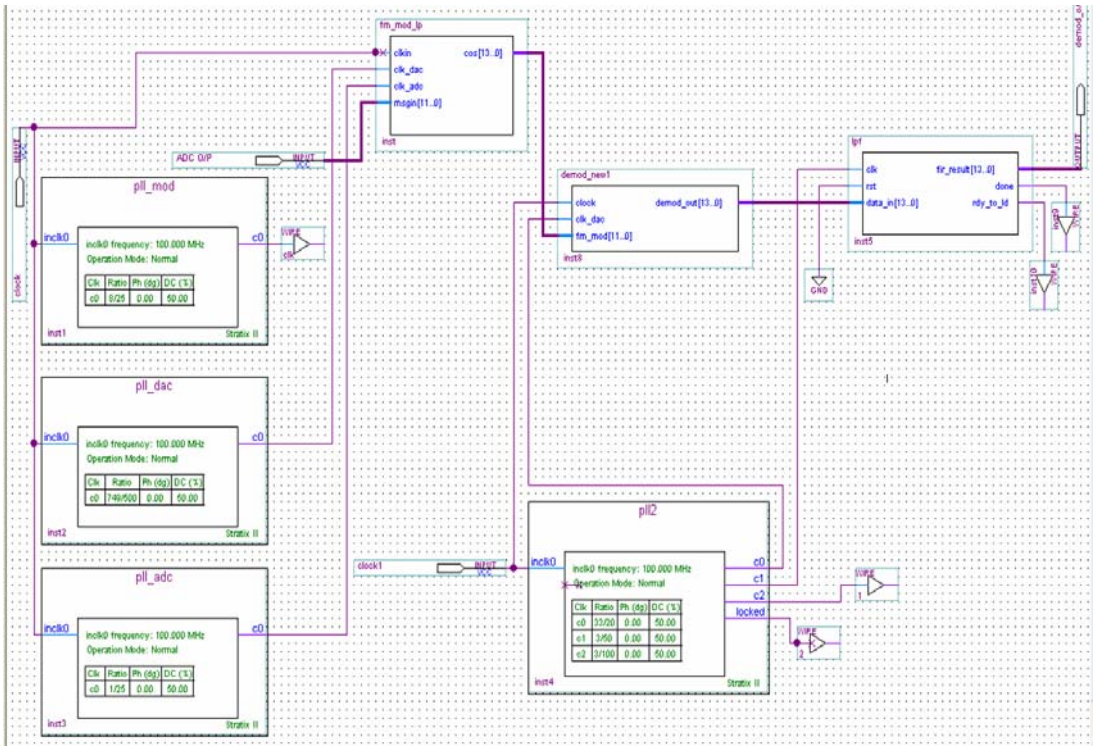
Block Diagrams

The following sections provide block diagrams for various system blocks.

FM Modulator/Demodulator

Figure 27 shows the FM modulator and FM demodulator block diagram. The figure combines the FM modulator and FM demodulator Block Design Files (**.bdf**) to show the overall design.

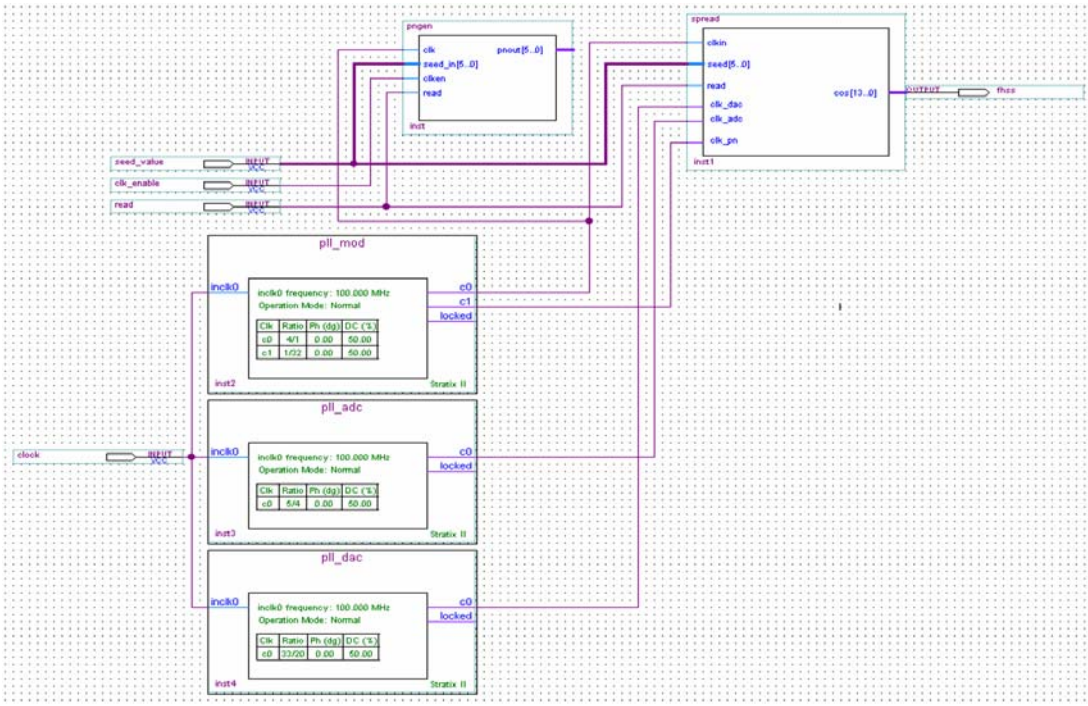
Figure 27. FM Modulator/Demodulator Block Diagram



FHSS Modulator

Figure 28 shows the FHSS generation BDF. The external environment provides the seed value. The module spread contains one FM modulator, which generates the output frequency depending on the PN generator output value.

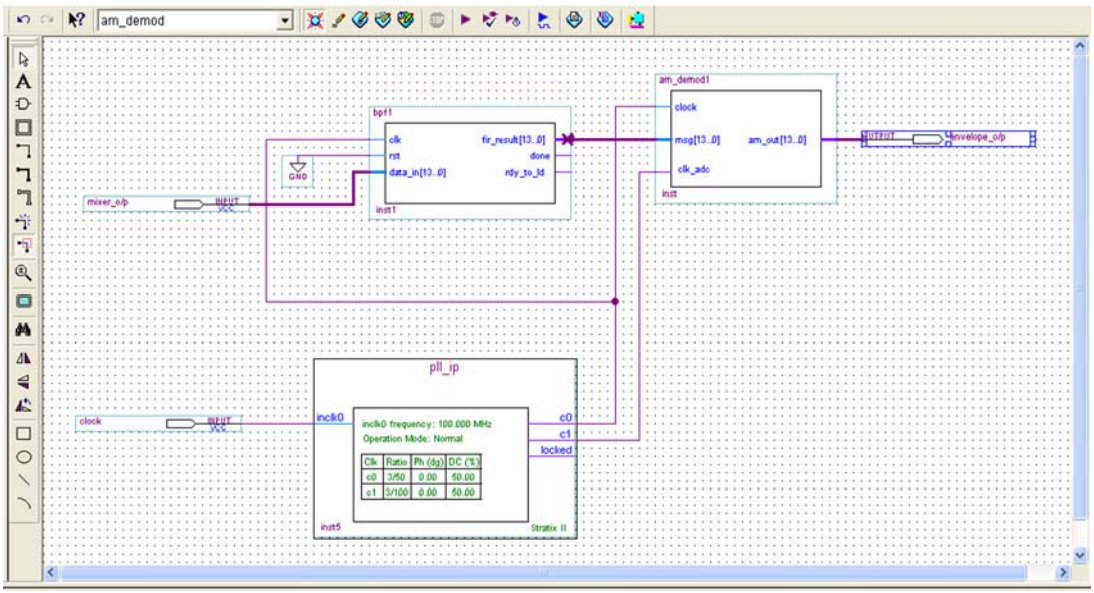
Figure 28. FHSS Generation Block Diagram



AM Demodulator or Envelope Detector

Figure 29 shows the envelope detector or AM demodulator. **bpf1** is the bandpass filter, which is used for synchronizing the FHSS receiver (see "FHSS Synchronization" on page 67).

Figure 29. AM Demodulator or Envelope Detector Block Diagram



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.

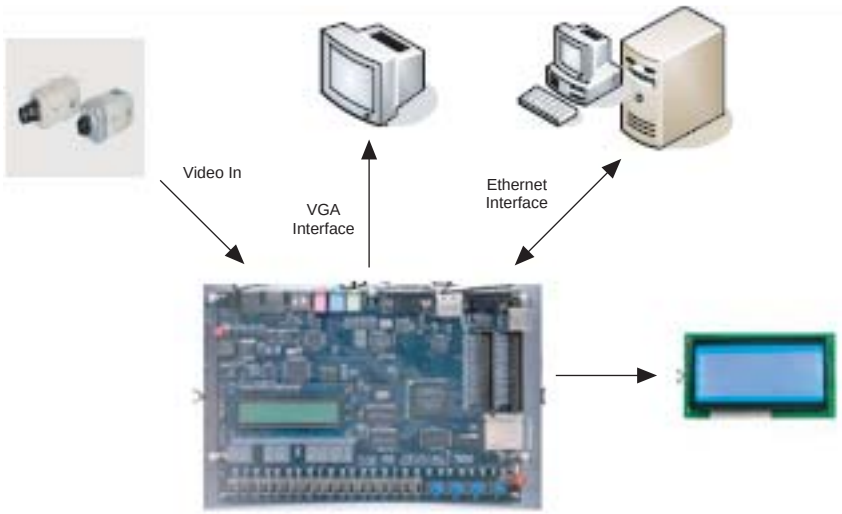
At present, most IP cameras on the market are implemented with hard-core microprocessors (MCUs) such as a special media processor. Our design proposes a network video transmission solution based on the Altera® Nios® II embedded soft-core CPU, implementing video data transmission via Ethernet. To reduce 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:

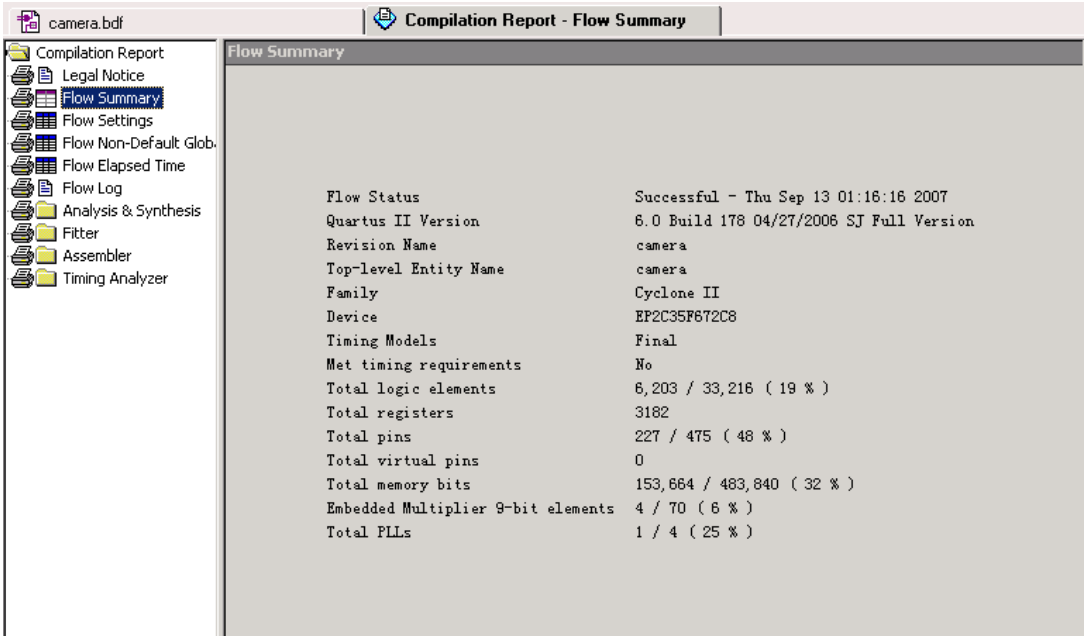
- Input video standard: NTSC
- Input video resolution: 768 x 494
- Output video standard: RGB

- 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

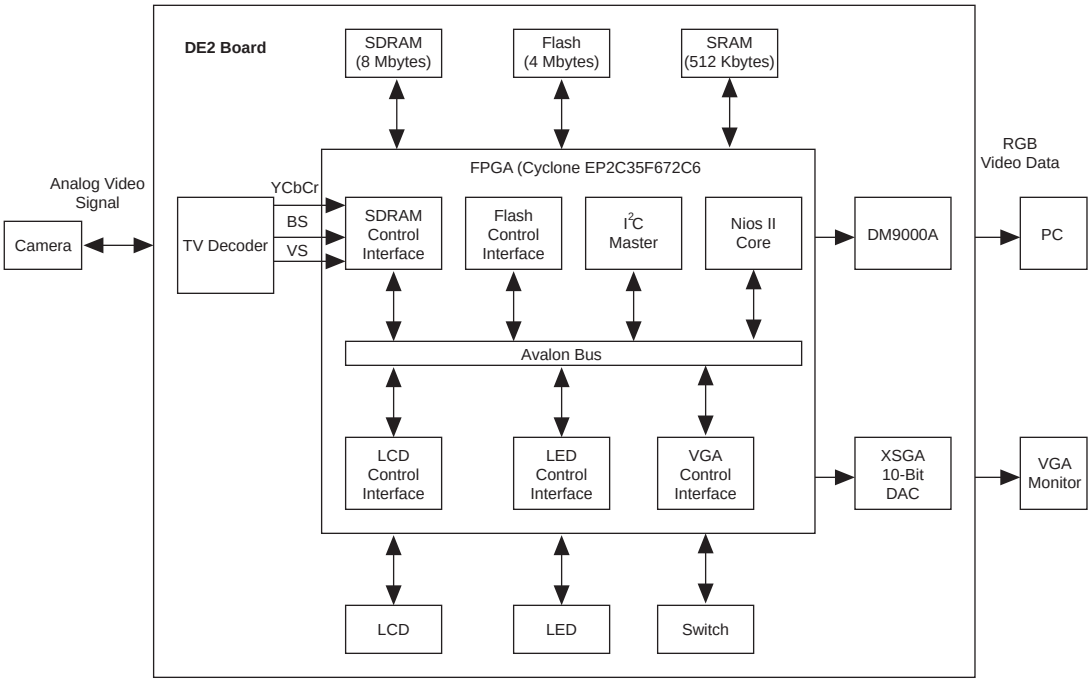
The design uses the following DE2 board resources:

- 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.

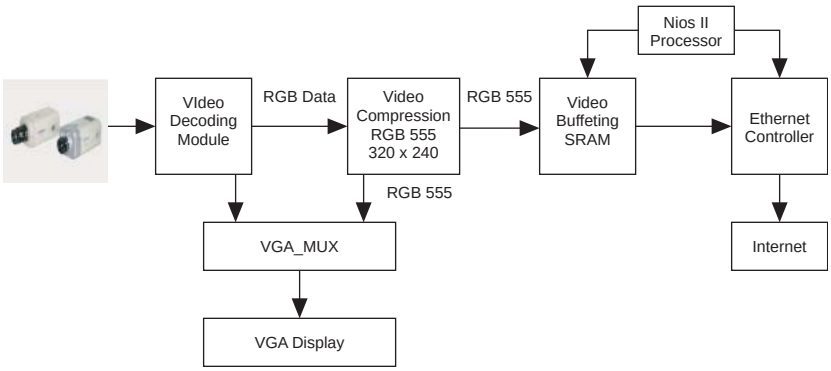
Design Methodology

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.

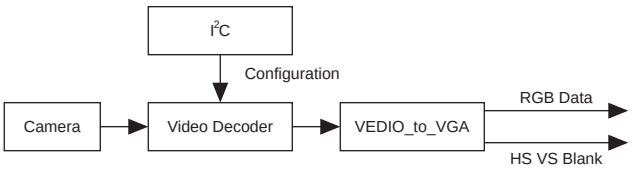
Video Decoding Module

The video decoding module has two functions:

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

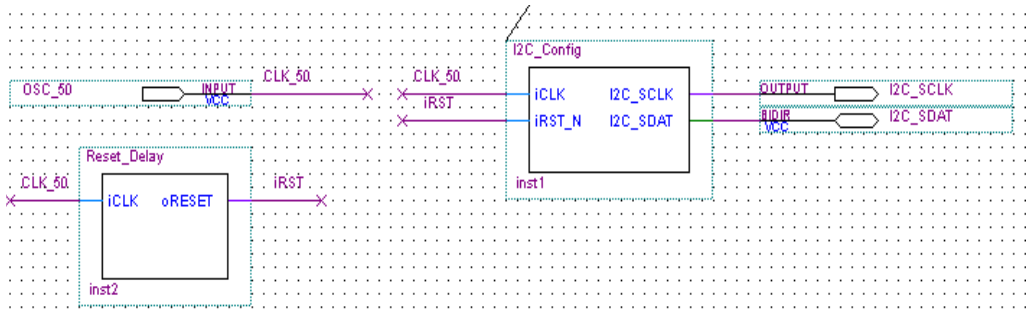
Figure 8 shows the video decoding process.

Figure 8. Video Decoding Schematic Diagram



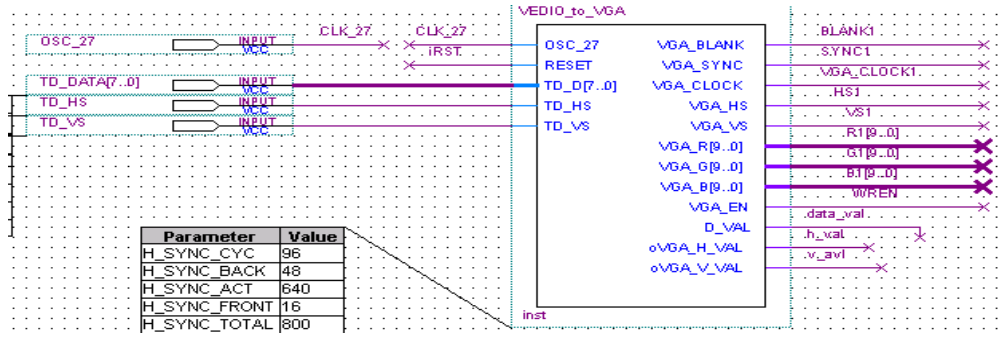
This system has an external NTSC (768 H x 494 V) camera connected to the DE2 board's video-in RCA interface. The design uses the I^2C bus to configure video decoder chip properly. Figure 9 shows the hardware implementation.

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

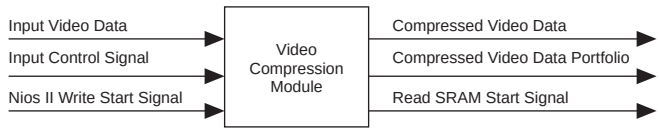
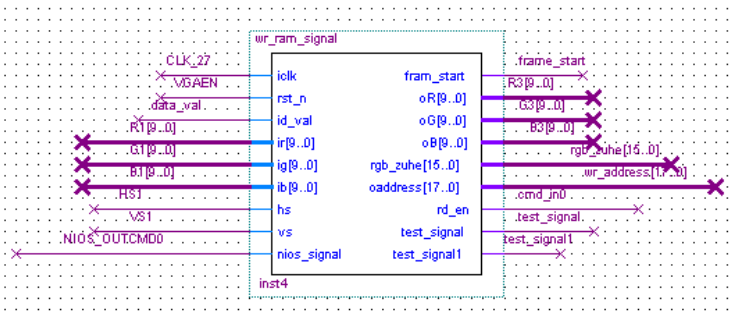


Figure 12. Video Compression Module Schematic Diagram



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

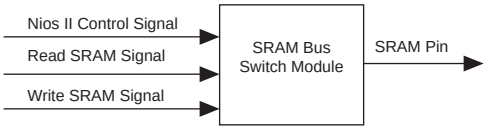
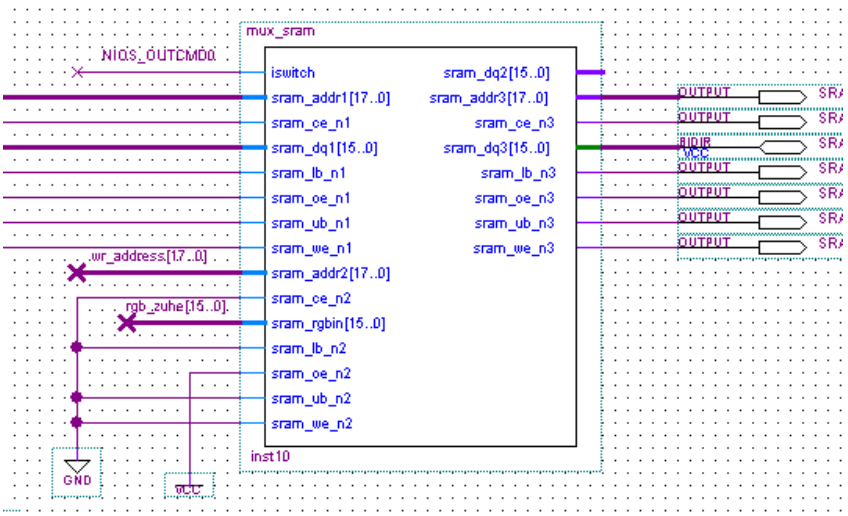


Figure 14. SRAM Bus Switching Module Schematic Diagram



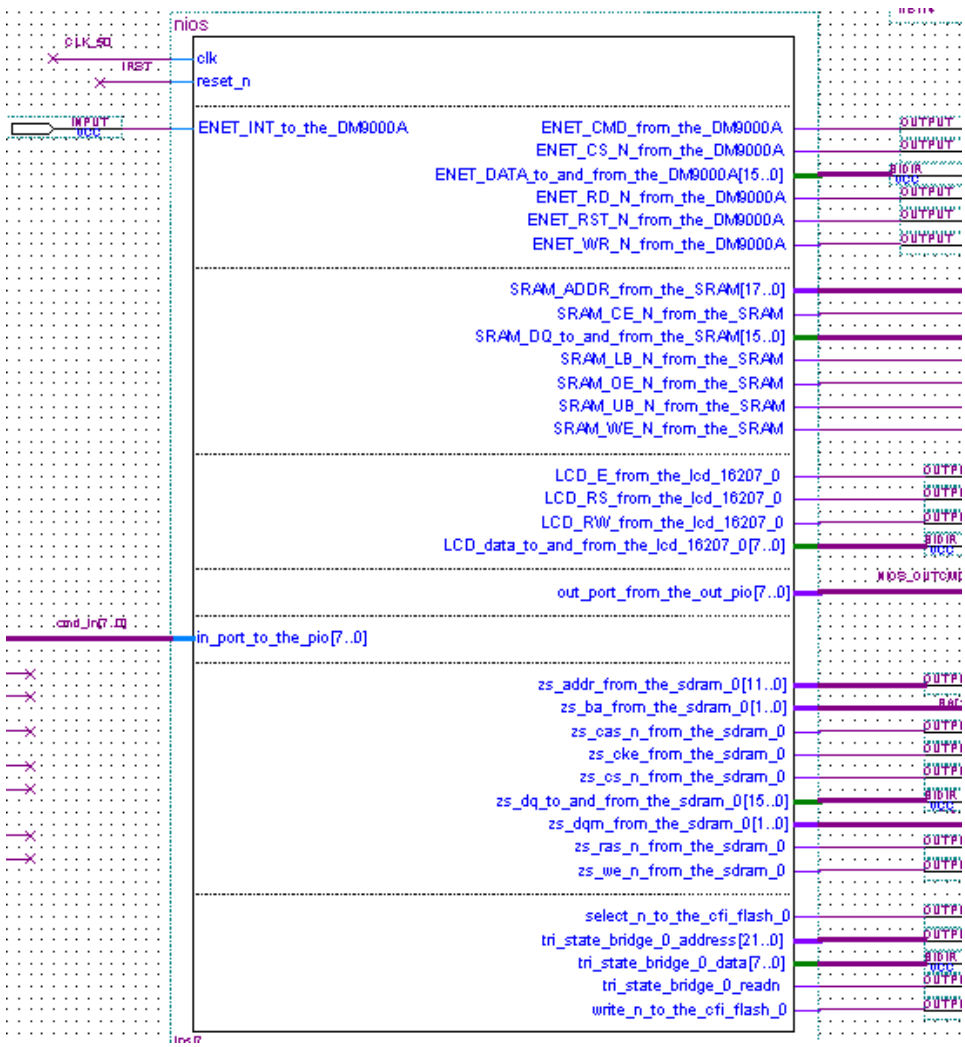
When the Nios II processor's `NIOS_OUTCMD0` control signal is high, the SRAM bus switching module writes the output video data from the video compression module into RAM. The hardware notifies the

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

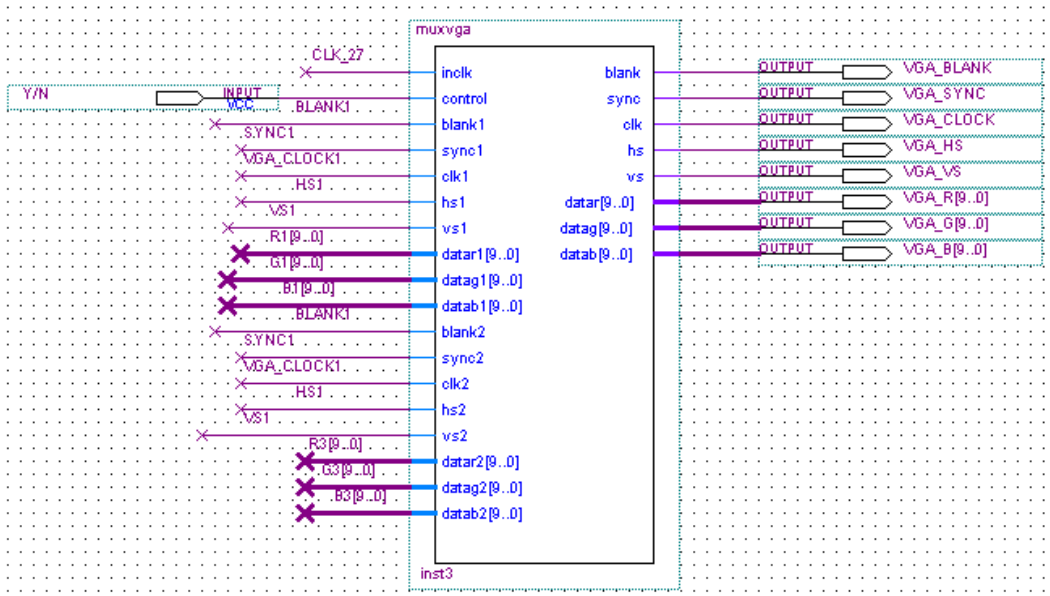


We implemented the data transmission control module in the Nios II soft-core processor. The processor conducts network initialization to acquire the MAC and IP addresses, receive the remote client's video request, control video compression and read/write video buffering, and transfer video data.

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

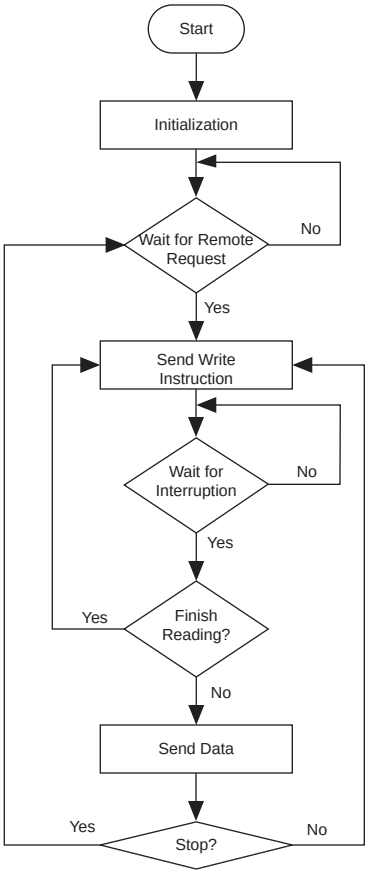
Our system requires network transmission, so it loads a network protocol. The LwIP protocol stack is integrated in the Nios II IDE, so we chose that protocol. Altera provides a Nios II port of LwIP, which enables a fast, open-source access to an Ethernet connection stack. Altera's LwIP port includes 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:

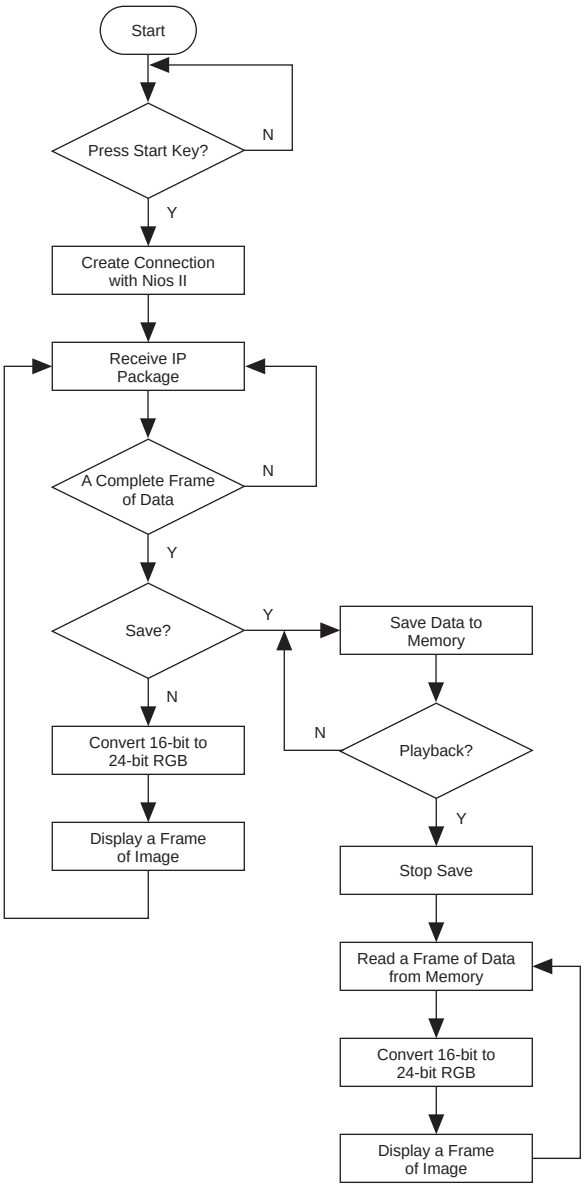
```
for(k = 0; k < 153600; k++)
 {
  rgb555 = (int)buff[k];
  rgb555 = rgb555<<8;
  rgb555 = rgb555|(int)buff[k + 1];
  k = k + 1;
  r = ((rgb555 >> 7) & 0xF8);
  g = ((rgb555 >> 2) & 0xF8);
  b = ((rgb555 << 3) & 0xF8);
  buff24[n++] = (unsigned char)r;
  buff24[n++] = (unsigned char)g;
  buff24[n++]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

Industrial Applications

This section includes projects that can be used as industrial applications, including:

Police Vehicle Support System with Wireless Auto-Tracking Camera 107

Smart Self-Controlled Vehicle for Motion Image Tracking 125

RTOS Acceleration Using Instruction Set Customization 143

Smart Bus Station Sign 195

Aerial Photographic System Using an Unmanned Aerial Vehicle 157

Laser Direct Writing Digital Servo Controller Based on SOPC Technology 173

FPGA-Based Smart Induction Motor Controller Design 205

Intelligent Solar Tracking Control System Implemented on an FPGA 217

Nios II Processor-Based Fingerprint Identification System 247

Fingerprint Identification System Based on the Nios II Processor 281

First Prize

Police Vehicle Support System with Wireless Auto-Tracking Camera

Institution: **Inha University, Korea Aerospace University, Hongik University**

Participants: **Sung Woong Joo, Ho Seong Suh, Young Je Moon**

Instructor: **Professor Tak Don Han**

Design Introduction

Our project, a police vehicle support system with a wireless auto-tracking camera, has three main goals:

- To improve police vehicles' ability to collect and interpret data.
- To provide real-time image transfers and an information sharing system between police vehicles and the command center.
- To create a high-performance, affordable solution using system-on-a-programmable-chip (SOPC) design concepts.

Existing police vehicle tracking systems can lose a suspected vehicle on screen because the vehicle's camera is fixed. Our project provides an auto-tracking solution that continuously shows the suspected vehicle's position on screen. To achieve this goal, we use an automated threshold calculation method that reduces the effect of light.

The pan-tilt camera uses a step motor. The camera moves right, left, up, and down. We designed the FPGA step motor controller to react quickly. In fact, the FPGA step motor controller's reaction time is faster than the software controller; therefore, the motor controller also helps the camera focus on the suspected vehicle continuously, even when the vehicle moves fast.

We designed an automatic voice alert system, which operates with the pursuit camera. We implemented hardware acceleration using Nios® II custom instructions for MPEG audio playback, eliminating the

need for an extra chip to perform MPEG audio decoding. We developed a µClinux audio driver for the WM8731DAC device on the Development and Education (DE2) development board.

We developed an FPGA-based on-board diagnostic system (OBD-II) interface to obtain vehicle information such as velocity and fault state from the engine control unit (ECU), which is a vehicle control system. The OBD-II interface measures the relative velocity and monitors the vehicle's state, replacing a high-cost laser measuring instrument. We designed the JPEG compression module using the `libjpeg` library, which is commonly used in Linux systems, and Altera's C-to-Hardware Acceleration (C2H) Compiler. The compression module provides image storage and wireless transfers. Our design improves the compression performance without requiring us to modify the code.

Another main feature of this project is a global, wireless, high-speed downlink packet access (HSDPA) function. The collected real-time data and image information is transferred wirelessly from the police vehicle to the command control center.

Suitability of In-Vehicle FPGAs

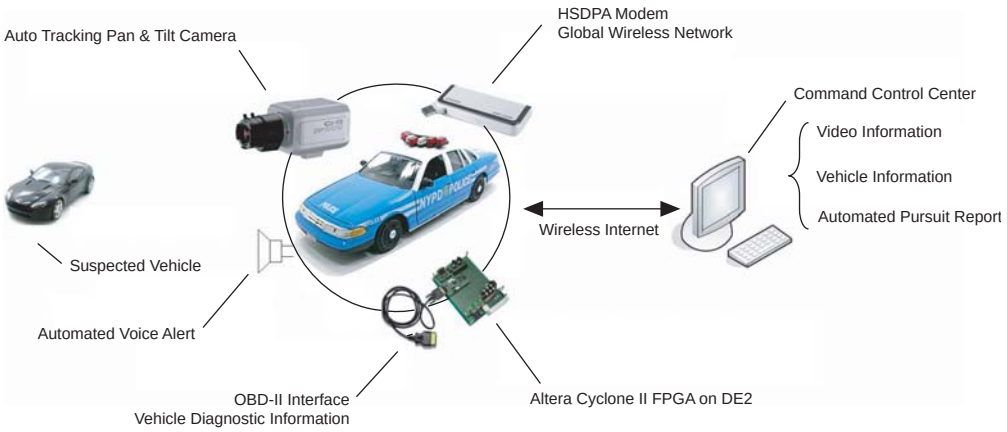
The police vehicle support system is a complex field that demands image, voice, communication, and sensor data processing. The vehicle systems increase in complexity as the amount of loaded equipment increases. For the OBD-II interface, different vehicles have different systems because their protocols are different. FPGAs are suitable for this field because they are easy to reorganize and re-synthesize.

We developed our SOPC system using the Quartus® II software and Nios II Integrated Development Environment (IDE) version 7.0. We implemented the operating system (OS) and applications using GNU tools. SOPC Builder made it easy to configure the system, and µClinux and the GNU tools offer familiar development environments.

Police Vehicle Needs

According to police pursuit policy, the officer must activate warning and recording equipment and report to the command control center upon engaging in a pursuit. It is difficult to perform all of these actions at the same time, so our project provides an automated, integrated solution. Figure 1 shows the design concept.

Figure 1. Design Concept



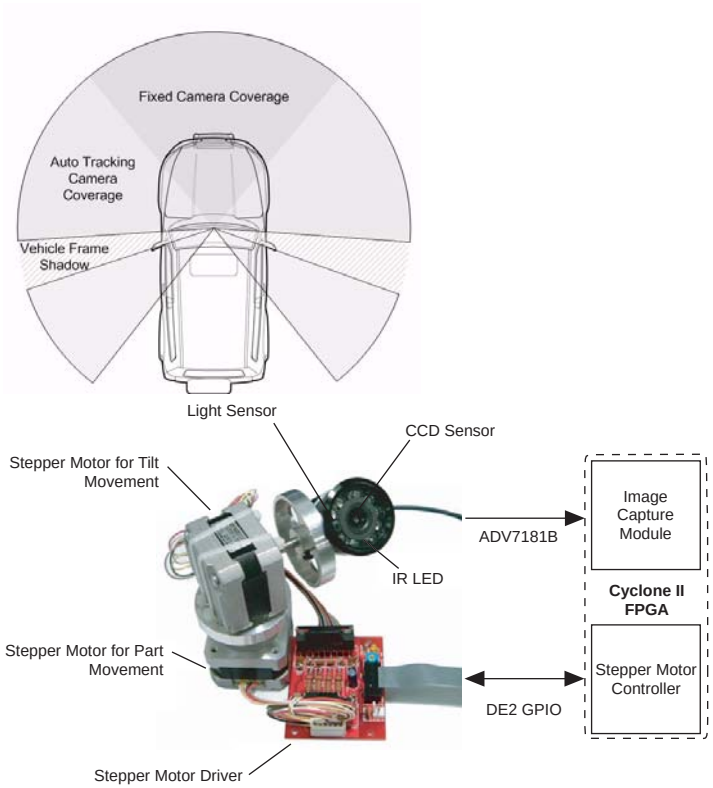
Function Description

This section describes the functionality of our design.

Auto-Tracking Camera

We made a camera module that moves horizontally and vertically so that the target vehicle's position is at the center of the screen at all times. A critical part of the pan-tilt camera module is the reaction time. A faster reaction time reduces the chance of losing the target vehicle. Figure 2 shows the pan-tilt camera module.

Figure 2. Pan-Tilt Camera Module



We chose a hardware-controlled method instead of a software-controlled method. We designed the stepper motor controller using Verilog HDL. The pan-tilt motion commands from the image processing module are transferred directly to the stepper motor controller on the FPGA. Then, the stepper motor controller receives the commands and generates operating signal pulses. Finally, the controller sends signals to each motor.

To enter tracking mode, the user aligns the camera to the target vehicle and presses a button on the DE2 board. The image processing module extracts the average color feature from the target vehicle and estimates the target vehicle's location. The pan-tilt camera tracks the vehicle as soon as the vehicle moves.

The captured 640 x 400 images are saved in USB storage and transferred to the control command center simultaneously. We tested the auto-tracking camera on the road (see Figure 3) and it works well in most cases. Our tracking mechanism does not operate at night and has a weakness with some colors because the tracking algorithm is based on color differences. Figure 4 shows the embedded systems in the vehicle.

Figure 3. Road Tracking Test

Figure 4. In-Vehicle Embedded System Configuration

Automated Voice Alert

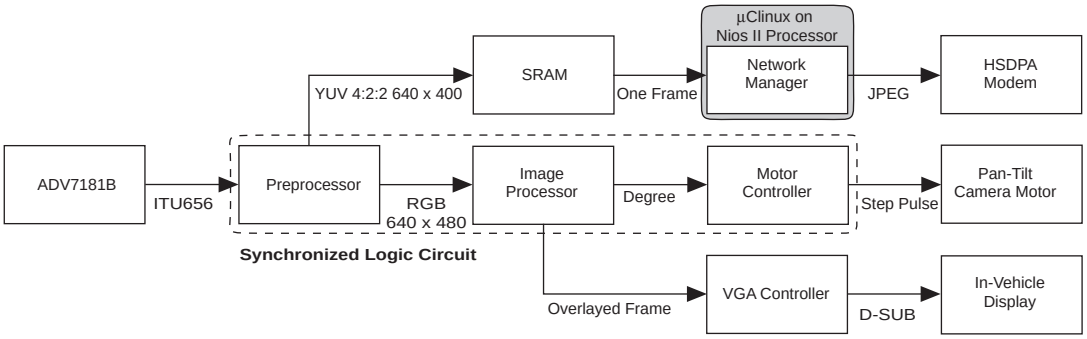
For the convenience of the officer in the vehicle, the automated voice alert system begins operating as soon as the auto-tracking camera enters tracking mode. MPEG audio data is played using a Nios II custom instruction without any extra processors. The Nios II processor operates at 100 MHz on the DE2 board.

The development board can play 128-Kbps, 44.1-KHz MPEG1 layer-3 mono-channel audio without acceleration, although the processor does need to reduce its load for multi-tasking. Therefore, we added a 64-bit multiplier, which improves the playing capacity approximately 2.5 times.

Image Capture Module

Figure 5 shows the image capture, processing, and transfer block diagram.

Figure 5. Image Capture, Processing, and Transfer Block Diagram



The camera's analog image information is converted into an ITU656 standard digital stream on the DE2 development board. This stream is used in three ways:

- It controls the auto-tracking camera's left, right, up, and down operation.
- It provides a vision sharing system and JPEG compression of the wireless transfers.
- It provides the in-vehicle display.

This image capture module preprocesses the image by modifying the image size, removing interlacing mode, and calculating the frame buffer memory address.

Image Processing Module

The camera's auto-tracking function requires a motion tracking algorithm. We used an adapted color tracking algorithm, which is easily supported on an FPGA. This algorithm calculates the average value of the changing vehicle color depending on the direction, and accordingly creates a new binary threshold value.

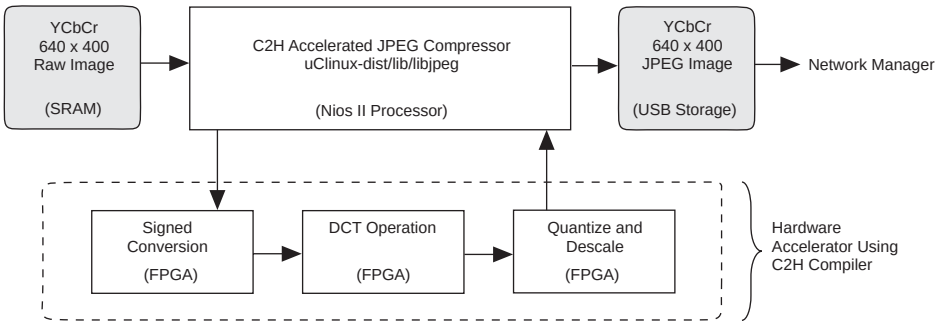
This algorithm processes by line unit, which is synchronized with the display input module in the FPGA without software processing. The design does not need outer frame buffer memory. The main benefit of this method is performance. The system transfers control commands to the motor controller every 1/30th of a second. Figure 6 shows the tracking algorithm being tested in the lab.

Figure 6. Tracking Algorithm Lab Test

C2H Accelerated JPEG Compression on μClinux

The images are 640 x 400 pixels and are compressed using JPEG. We replaced the `libjpeg` forward discrete cosine transform (DCT) function with an accelerator that we developed using the C2H Compiler. The accelerator can be accessed in the μClinux environment. Combining the C2H accelerator and μClinux is a very important feature because the acceleration operates concurrently with other tasks. By accelerating `libjpeg`, a standard library, we improved compression performance without using an extra digital signal processing (DSP) chip or other typical software. Applications using `libjpeg` have improved compression performance through recompiling without modifying any code. Figure 7 shows the JPEG compression diagram with C2H acceleration.

Figure 7. JPEG Compression Diagram



Custom OBD-II Interface

Vehicles, including police vehicles, have ECUs for system management. The ECU is a very important component in recently manufactured vehicles because it unifies the engine and various electronic controls. OBD-II is an interface that provides communication between devices and connects a computer or diagnostic tools to the ECU for vehicle maintenance.

There are many OBD-II standards depending on the vehicle manufacturer. Our project adopts the ISO9141-2 international standard. Using OBD-II, we can determine the driving speed, fuel state, and vehicle fault state. OBD-II has a 5-baud initialization procedure and a 10.4-k baud communication speed. Received information bytes have to be complemented and sent back to the ECU for communication. We used the SOPC Builder UART component because it is similar to serial communication. Figure 8 shows the DE2 board and extension equipment for the OBD-II interface. Figure 9 shows the captured image and OBD-II information display.

Figure 8. DE2 Board Extension Equipment and OBD-II Interface

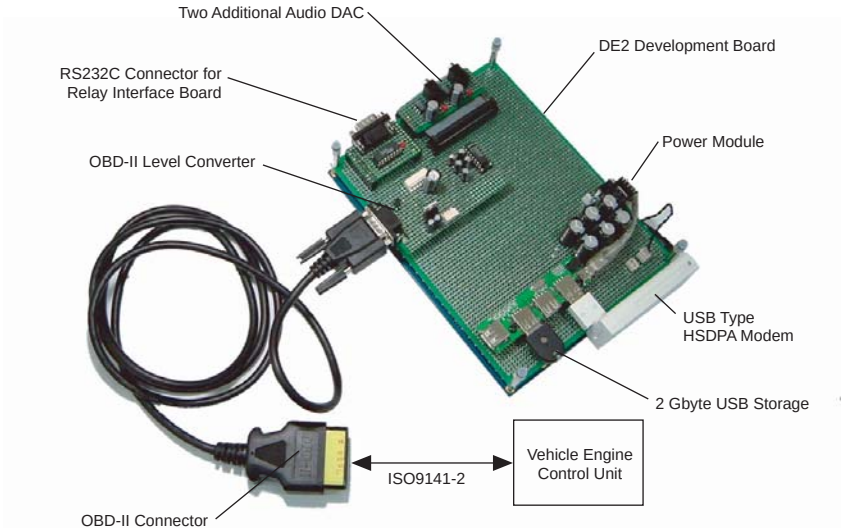
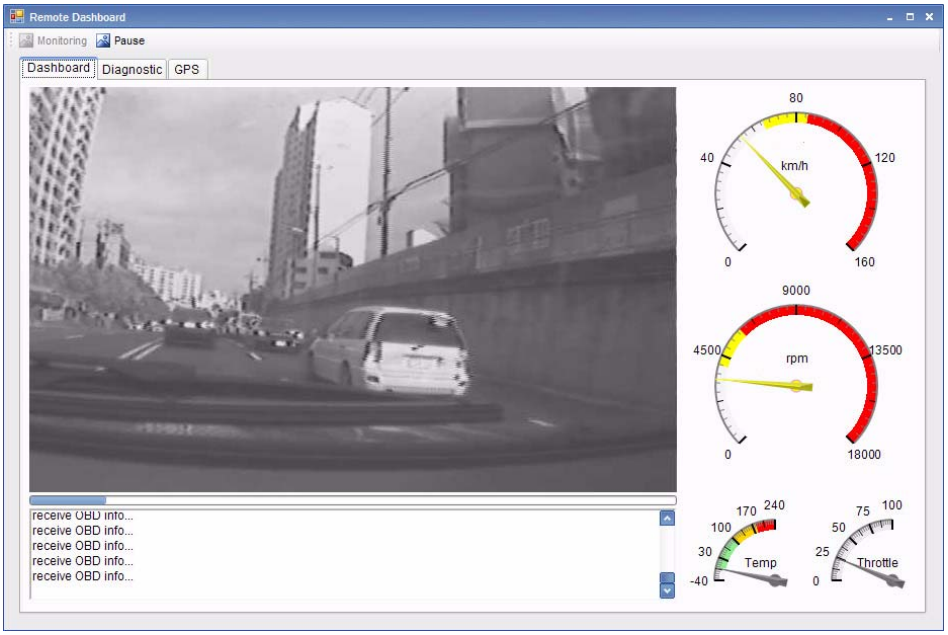


Figure 9. Captured Image and OBD-II Information Display



Performance Parameters

This section describes the performance parameters of our design. Table 1 shows the time interval between when the image processing module sends control signals and the stepmotors receive the initial operation signal. We measured the time interval using an oscilloscope. In the software program

controller, the Nios II processor receives interrupt signals and generates an operation signal, after which the stepmotor starts through the general-purpose I/O (GPIO) interface.

Table 1. Stepper Motor Pulse Generation Method Comparison

Latency	Using Interrupt Routine and Timer	Full FPGA Controlled Method
Average latency (µs)	60	35

The velocity of the auto-tracking camera is mostly determined by the image processing performance. Table 2 shows the tested frame rate of each tracking algorithm for different platforms. The DE2 board's frame rate is almost 60 frames per second (fps) because the image processing module operates in interlace mode; however, we show 29 fps, which is the number of effective frames.

Table 2. Tracking Algorithm Performance Comparison

	PC	ARM	DE2 Board
Clock	2.4 GHz	520 MHz	100 MHz
Implementation	Software	Software	Full FPGA
Frame rate (fps)	29	10	29

Table 3 shows the compression performance of the `libjpeg` DCT function accelerated using the C2H Compiler. The 640 x 400 x 24-bit bitmaps are compressed 20 times for accurate measurement. Compiling with the C2H Compiler shows worse performance than the design that has no accelerator. To solve this problem, we changed the buffer management method, resulting in a 4x performance improvement after modifying the DCT function.

Table 3. JPEG Compression Performance Comparison

	libjpeg	libjpeg	mod-libjpeg
Compiler	gcc	gcc, C2H	gcc, C2H
Accelerated function	N/A	forward_dct()	forward_dct()
Compression time (seconds)	210	281	52

Table 4 shows the modem performance test. When we designed the system, we were concerned about low performance when using a USB modem with the µClinux system. According to our test, it showed almost the same network performance as the PC environment.

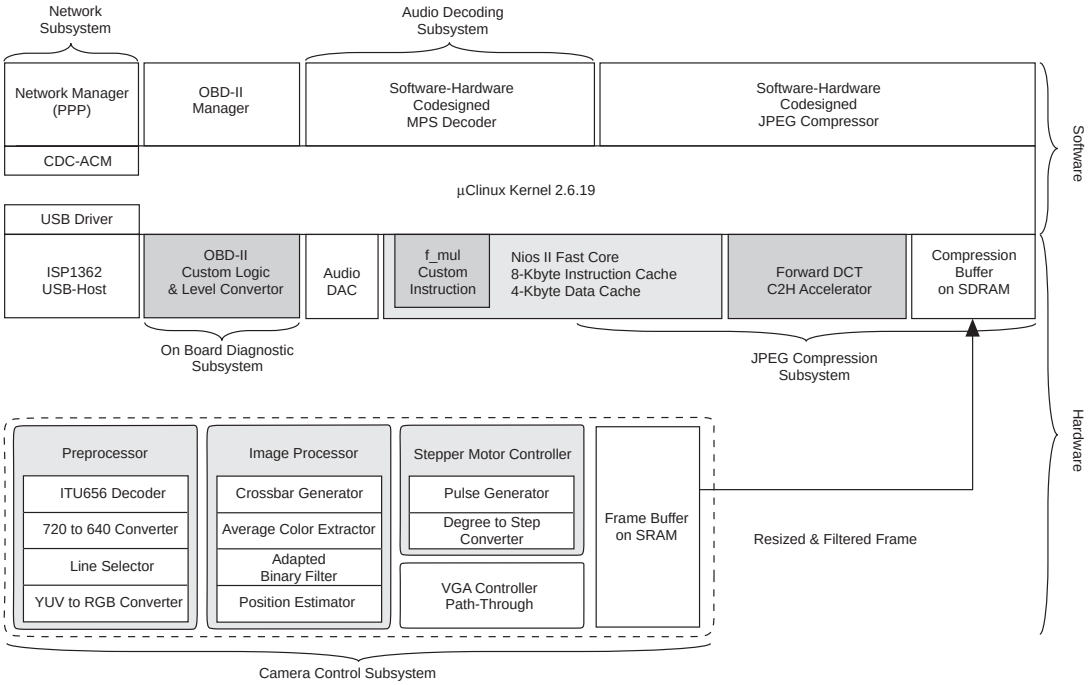
Table 4. USB HSDPA Modem Performance Test on µClinux

	Expected (PC)	DE2 Board
OS	Linux 2.6.22	µClinux 2.6.19
Download (Kbytes/second)	32	30
Upload (Kbytes/second)	28	25

Design Architecture

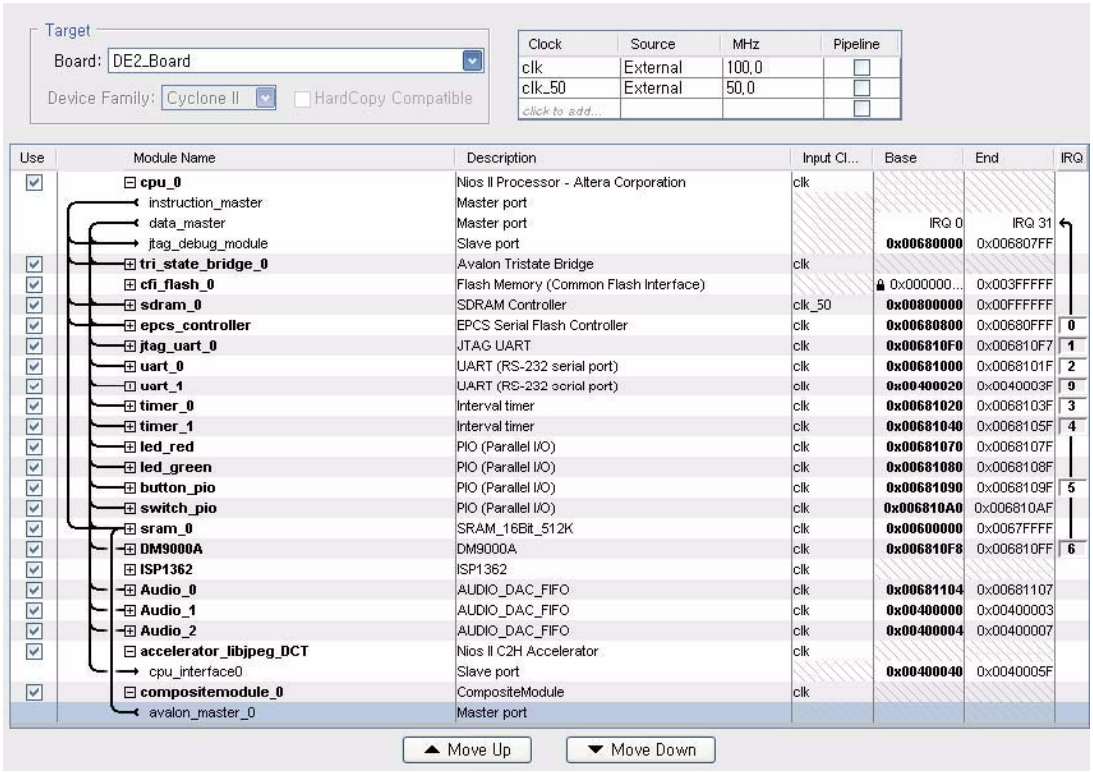
Figure 10 shows the system architecture.

Figure 10. System Architecture



µClinux controls the software systems. The FPGA contains the camera control system and sub-systems, including the image processing modules. Grey boxes in Figure 10 indicate custom-designed SOPC components. Figure 11 shows the system in SOPC Builder.

Figure 11. SOPC Builder Configuration



We moved the `libjpeg` DCT function into the C2H accelerator. We combined the image processing module, VGA controller, and stepmotor controller into a unique SOPC component. The design uses 31,000 logic elements (LEs). Figure 12 shows the Quartus II compilation report.

Figure 12. Quartus II Compilation Report

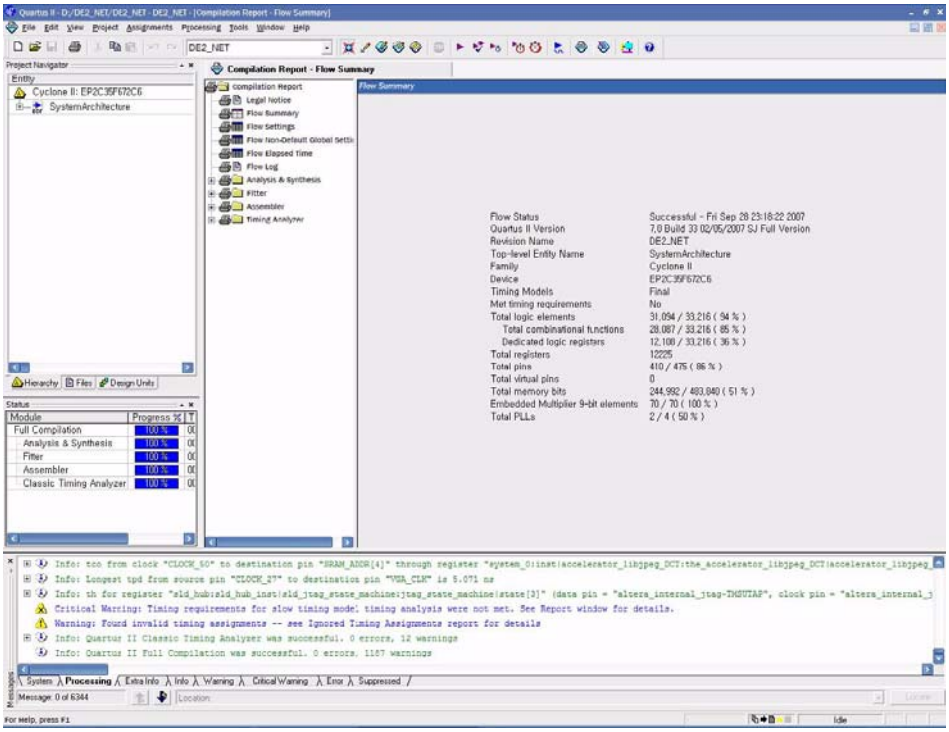


Figure 13 shows the top-level entity.

Figure 13. System Top-Level Entity

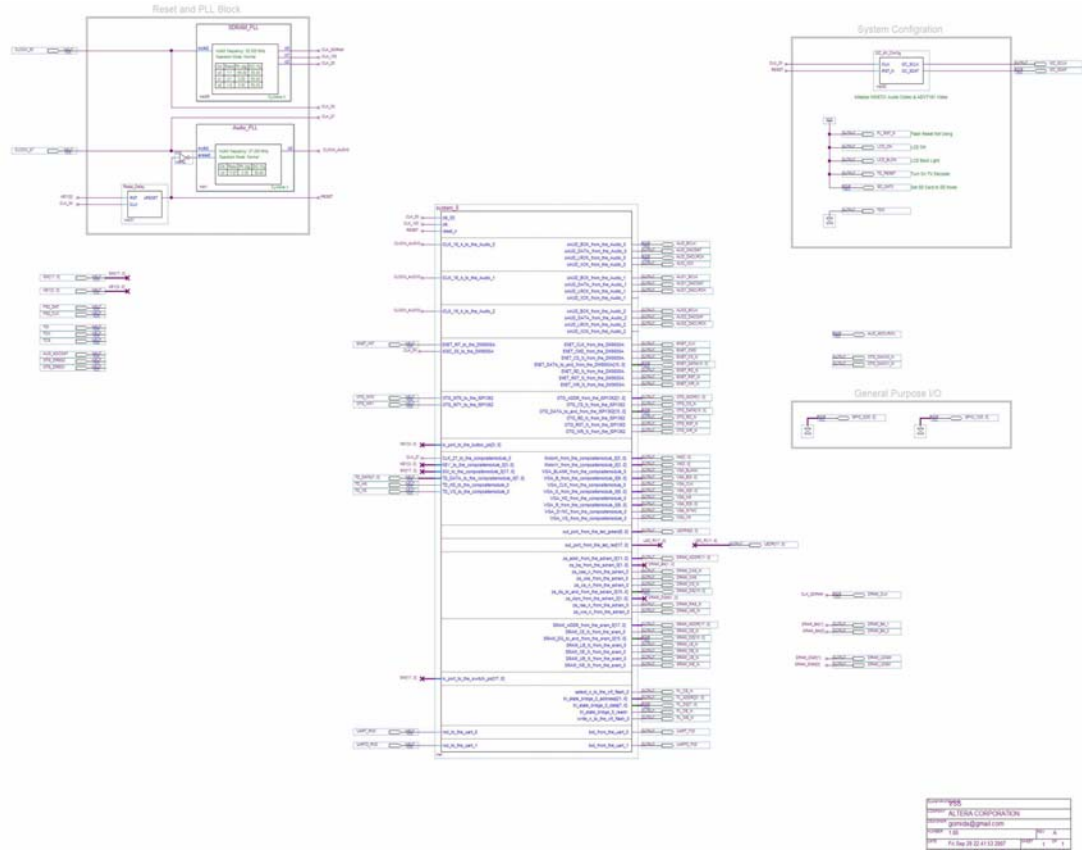
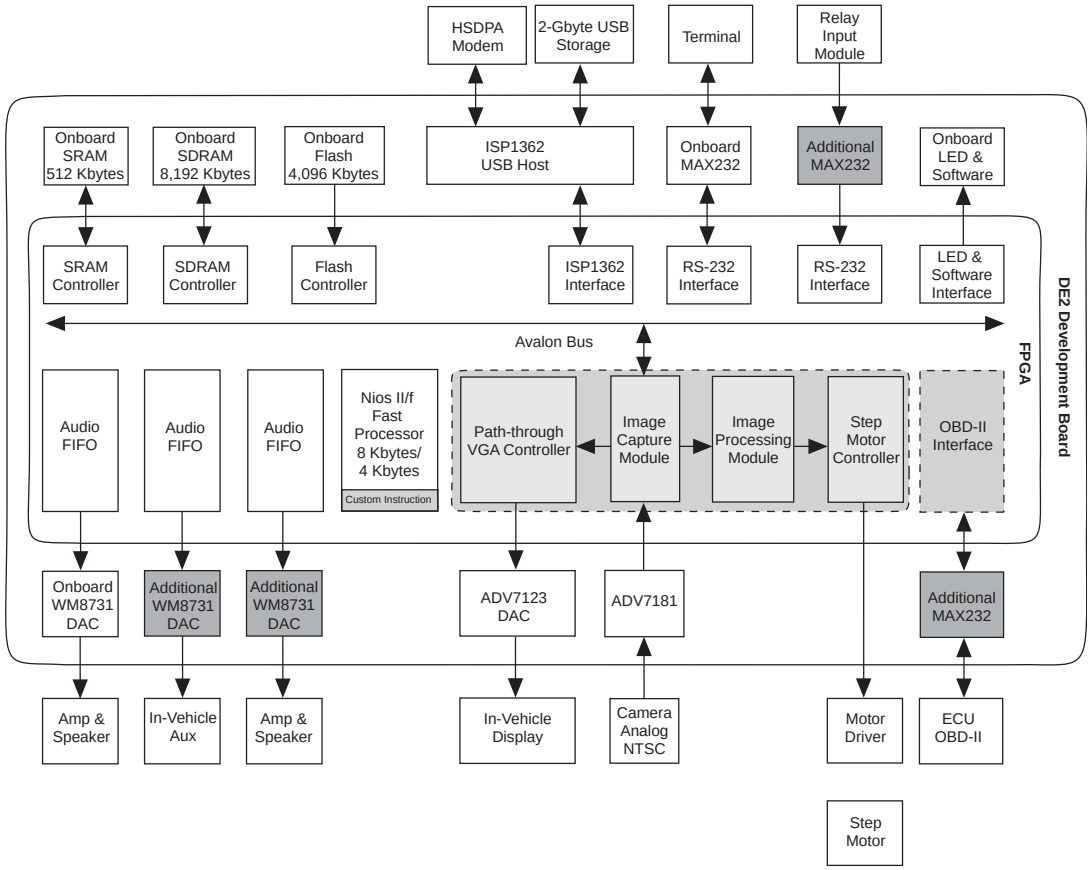
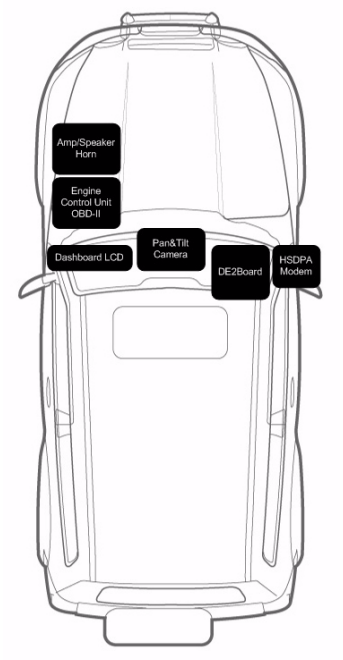


Figure 14 shows the on-chip and in-vehicle block diagram.

Figure 14. On-Chip and In-Vehicle Configuration





Design Description

This section describes our implementation method and the steps we used to build our design.

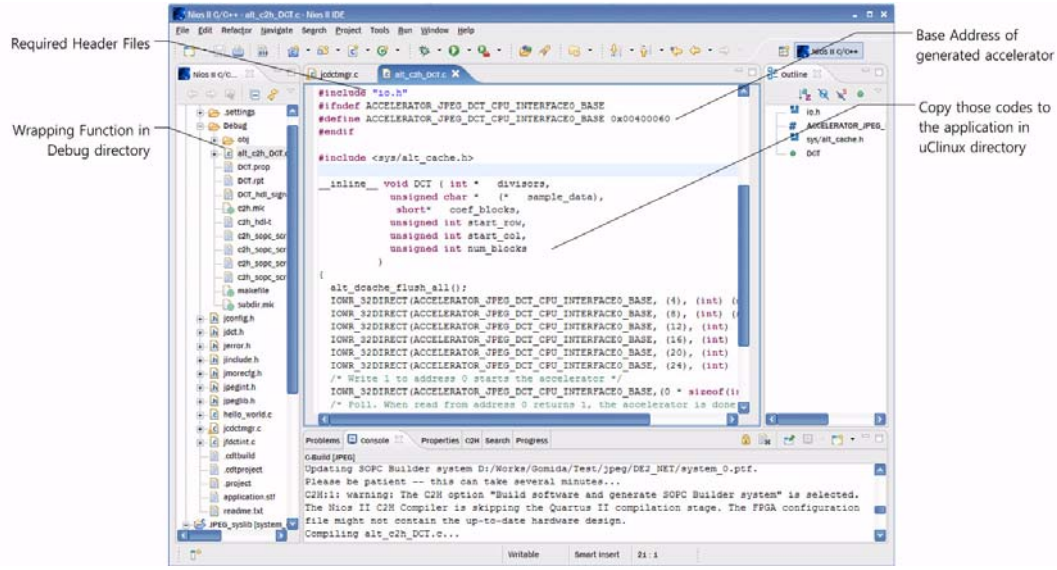
Combining µClinux and C2H

Using an operating system offers development flexibility for complex multi-device systems. The µClinux kernel is appropriate for non-memory management unit (MMU) processors. Because µClinux does not have an MMU, the Nios II processor application that accesses the custom hardware accelerator is simpler. We compiled the code we wrote in the Nios II IDE to operate in a multi-tasking environment under µClinux with few or no changes because there is no limit writing to memory-mapped addresses in µClinux. We used the C2H accelerator on µClinux with typical techniques. The following discussion describes the steps required to move the C2H accelerator from the Nios IDE into µClinux (see Figure 15).

First we made a temporary project. Then, we compiled and generated the accelerator in the Nios II IDE. After generation, the accelerator's wrapping function is saved in the **debug** directory. We copied the header files and wrapper to the µClinux development directory and programmed the FPGA.

Next, we compiled the accelerated application using Nios II gcc tools with the `elf2flt` option, we first made sure that the required header files such as **system.h** and **io.h** existed. We then copied the generated execution file to the development board. This implementation is faster than a software-only system in most cases. Unfortunately, we faced a performance problem when converting the `libjpeg` DCT function to the accelerator. "Optimizing the JPEG Library for the C2H Compiler" on page 120 describes how we solved the performance problem.

Figure 15. Moving the C2H Accelerator Wrapper into µClinux from the Nios II IDE



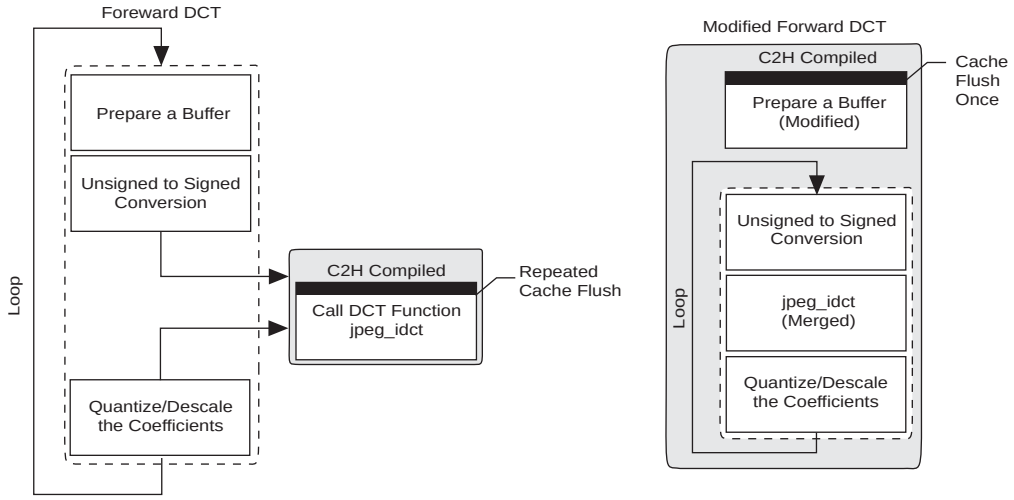
Optimizing the JPEG Library for the C2H Compiler

Generally, developers use a digital signal processor for JPEG compression, but a processor requires software to support it. Using the C2H Compiler to accelerate `libjpeg` is an interesting solution because many existing applications use `libjpeg`, which is a standard JPEG library.

When converting an original DCT function with the C2H Compiler, however, the function had lower performance than the software-only design. Flushing the data cache, which occurs every 64 bytes of data processing work, caused the performance problem. Therefore, we designed an optimized buffer

management system that was suitable for the C2H Compiler. This solution improved the performance 4 times. Figure 16 shows the optimized DCT function block diagram.

Figure 16. Optimized DCT Function Block Diagram



Creating the Custom SOPC Component

We combined the image processing module, VGA controller, and stepmotor controller as a unique component because these functions need to work together closely. We designed each block separately in Verilog HDL and added them to SOPC Builder as components. The components write image data in the SRAM as an Avalon® master. Figure 17 shows the custom component in SOPC Builder and Figure 18 shows the multiplier custom instruction.

Figure 17. Custom SOPC Builder Component

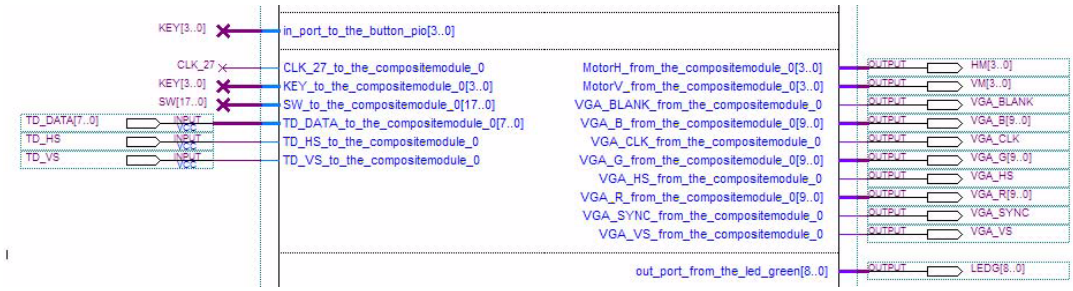
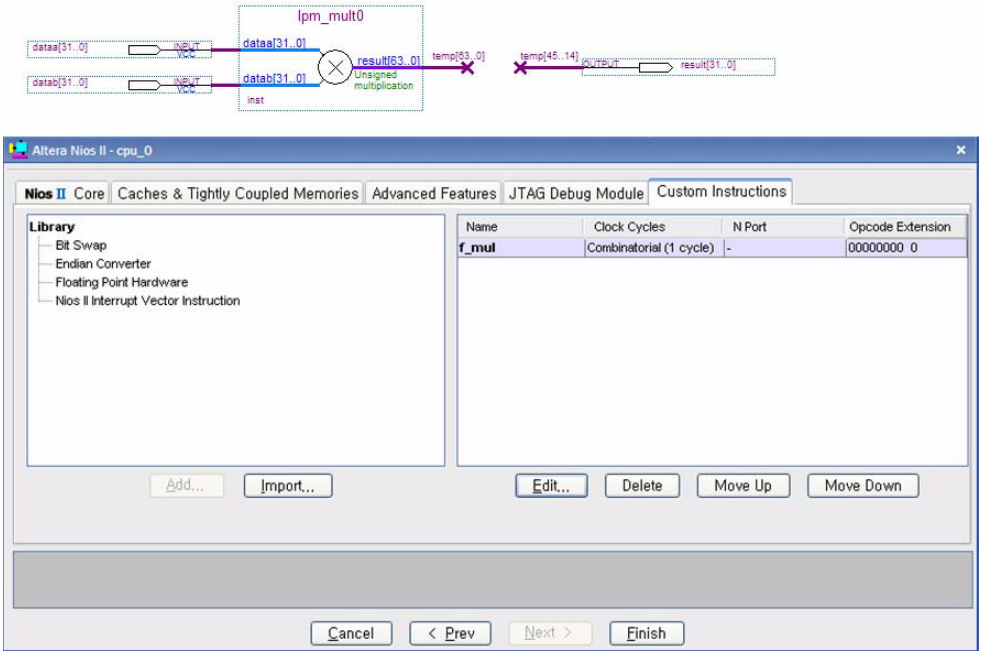


Figure 18. 64-Bit Multiplier Custom Instruction



MPEG Audio Decoding Custom Instruction

There are three main issues to consider when playing MPEG audio in a Nios II processor/µClinux environment

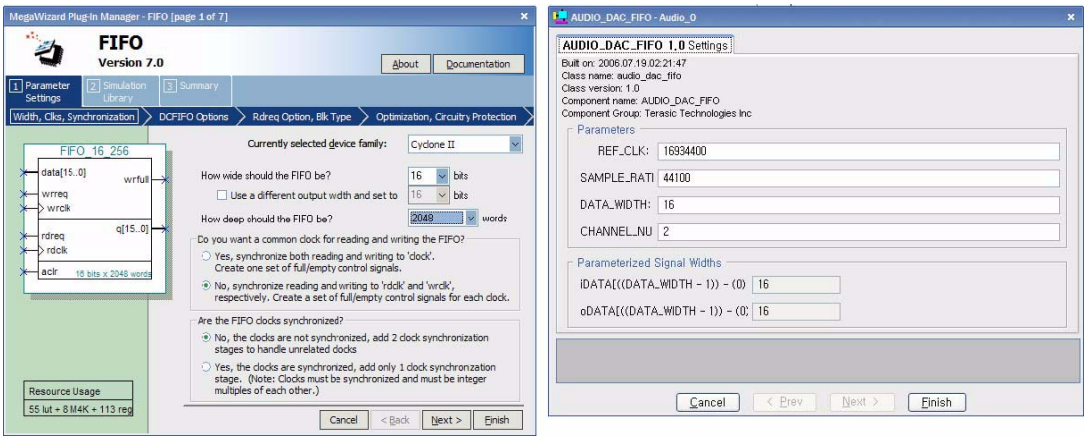
- Processor performance
- FIFO buffer size
- µClinux device driver output

The design had poor performance when the 100-MHz Nios II processor in the Cyclone® II device decoded the stereo 128-Kbps, 44.1-KHz MPEG1 layer 3 audio. With a big enough FIFO buffer, the system could play mono-channel audio but the CPU allocated all of its time to playing audio.

To solve this problem, we added a 64-bit multiplier custom instruction to the Nios II processor to implement a 64-bit multiply calculation that is frequently used by the `libmad` library. With this change, we increased the audio playback performance about 2.5 times and decreased the number of clock cycles required for the calculation.

Other issues can also cause audio playback problems, such as a bad sampling rate, a lack of buffer space, and a multi-tasking environment. Figure 19 shows a configuration that solves this problem with a 17-MHz audio reference clock.

Figure 19. DAC FIFO Setting for 44.1-KHz MPEG Audio



Design Features

The most fascinating part of this system is its integrity. One FPGA performs the whole function, including image processing, compression, transferring, MPEG audio decoding, step motor control, and OBD communication. Each block is an SOPC Builder component, so it is very easy to recycle components for different projects.

We designed the device driver so that all systems can operate in the µClinux environment. The access method connects the µClinux application, custom instruction accelerator, and C2H technology. We unified the software as well as the hardware.

The main image processing feature minimizes memory access. We only used the frame buffer memory for JPEG compression because the image processing is implemented by a line unit. Eventually, we could dramatically save Avalon bus bandwidth. Table 5 shows the design's key features.

Table 5. Key Features and Related Modules

Feature	Related Module	Implementation
Auto-Tracking Camera	Stepper Motor Controller	Custom SOPC Builder Component
	Image Capture Module	
	Image Processing Module	
Automated Voice Alert	MPEG Decoder	Nios II Custom Instruction
	WM8731 DAC Driver for µClinux	Character Device Driver
Command Control Center (Remote Dashboard)	OBD-II Interface Module	Custom SOPC Builder Component
	JPEG Compressor	C2H Compiler

Conclusion

We developed and tested a police vehicle support system representing an application in the telematics field. The auto tracking camera tracks a typical vehicle and positions it at the center of the screen at all times. The OBD interface is based on FPGA technology and captures information about the vehicle's state. A remote system distantly verifies the vehicle's image and state information and communicates using a global wireless HSDPA module.

We started the project with a fixed hardware platform and limited resources. FPGAs offer amazing technology that can easily adapt to design configuration changes. Our simple, effective design method involved modifying the hardware design when there were performance problems or problems that could not be solved in software. We noted that FPGAs are particularly valuable in the field of image processing.

We used the Altera® C2H Compiler for JPEG compression. We started out with the `libjpeg` DCT function, which is commonly used. We achieve high performance in our design by accelerating the `libjpeg` DCT function with C2H technology. We believe that the C2H Compiler, which translates C programs into HDL, is driving a changing software paradigm. The C2H Compiler has challenges, but we expect it will contribute to the development of software-to-hardware translation technology.

First Prize

Smart Self-Controlled Vehicle for Motion Image Tracking

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

Participants: **Chang-Che Wu, Shih-Hsin Chou, Chia-Hung Chao, Chia-Wei Hsu**

Instructor: **Dr. Ming-Haw Jing**

Design Introduction

Automobile electronics pose high requirements for safety, and the number of automobiles on the road is huge. According to Global Automotive Components, there were 760 million automobiles in 2004 and 850 million in 2005. General Motors projects that by 2020, there will be 1.1 billion automobiles worldwide. Data from IC Insights shows that until 2010, automobiles with on-board electronics will account for 40% of these vehicles. Therefore, there is a huge potential for growth in the automobile electronics market, which can generate substantial profit for automobile vendors as well as provide an opportunity for rapid growth of Taiwan's high-tech vendors. In these circumstances, auto electronics will become the highlight of Taiwan's research and development.

Our project uses image processing technology to provide identification and implement self-controlled auto guidance. For example, if the driver is not familiar with backing an auto into the garage, he/she can start the electronic automated guide setting for automated guidance, speed control, identification, etc.

In October 2007, Toyota's latest Lexus models, the LS 460 and LS 460L, were marketed in U.S. These models are equipped with Toyota's latest Advanced Parking Guidance System. The system uses a backward camera and a sonar sensor to detect the vehicle's surroundings. If the driver presses a button and turns on the brakes to control the auto's speed next to a parking space, the system automatically hands over the power steering to finish parking.

Similarly, the automatic guided vehicle (AGV), which is most commonly used in automated factory systems, can move forward, stop, and turn following the commands and routes of the program, and can be linked to a material handling system. A type of fully automated material handling equipment, an AGV can load goods automatically at a fixed location and move to another location for unloading. The

basic function of the AGV is to walk automatically along a fixed track. Although this technology has been used in factories, it cannot be used in a complex environment because its route must be planned in advance and the track must be drawn on the ground.

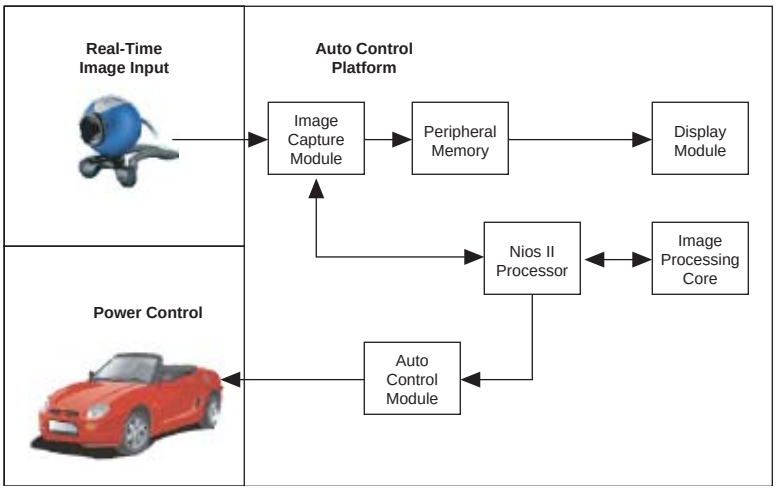
Unlike the AGV, our project uses images to identify a mark, the position of which can be altered according to the application environment. Compared with traditional AGVs, our design has wider application. Additionally, this design can search for a specific mark, automatically analyze the existing image information, lock in on the object, and control the automobile. Besides assisting people with parking and backing into a garage, the system can be used for airplanes and any other power vehicle. This project implements a machine vision algorithm based on a hardware acceleration module, and uses the efficient, multi-core embedded Nios® II processor to implement the self-controlled automobile guidance platform. In the future, the design can implement many applications such as auto security, including anti-bumping, driveway deviation alarms, driveway retaining (i.e., guiding the driver to the original path), rear obstacle alarm, pedestrian monitoring, distance monitoring (i.e., keeping the driver some distance away from the auto ahead), night viewing, automatic headlight adjustment, transportation/speed-limit sign identification, blind spot monitoring, etc.

Using two embedded Nios II soft-core processors, this design integrates complex peripheral circuits and memory modules into an auto control platform through the easily designed, highly integrated Avalon® bus. With the high-performance Nios II processor, the design can easily implement real-time image processing and high-speed automated control products.

Project Description

Using hardware and software co-design, we integrated the image input port, auto-controlled platform, and power control module into an experimental platform for automated target tracking. We built the smart image-tracking embedded system platform with a high-performance soft-core CPU controlling the peripheral modules and a VHDL image processing core circuit. See Figure 1.

Figure 1. Embedded System Featuring Smart Image Tracking



The core components of the system are:

- CMOS sensor hardware module
  - *CMOS sensor controller*—Drives the CMOS sensor, continuously captures images, and imports the motion image data flow.

- *Data simplification*—Compresses the image data (GB and GR) captured by the CMOS sensor, reducing the computing workload and analysis time.
- *SDRAM controller*—With six FIFO controllers, it allots SDRAM resources to two CMOS sensor controllers and VGA controllers (three for writing and three for reading).

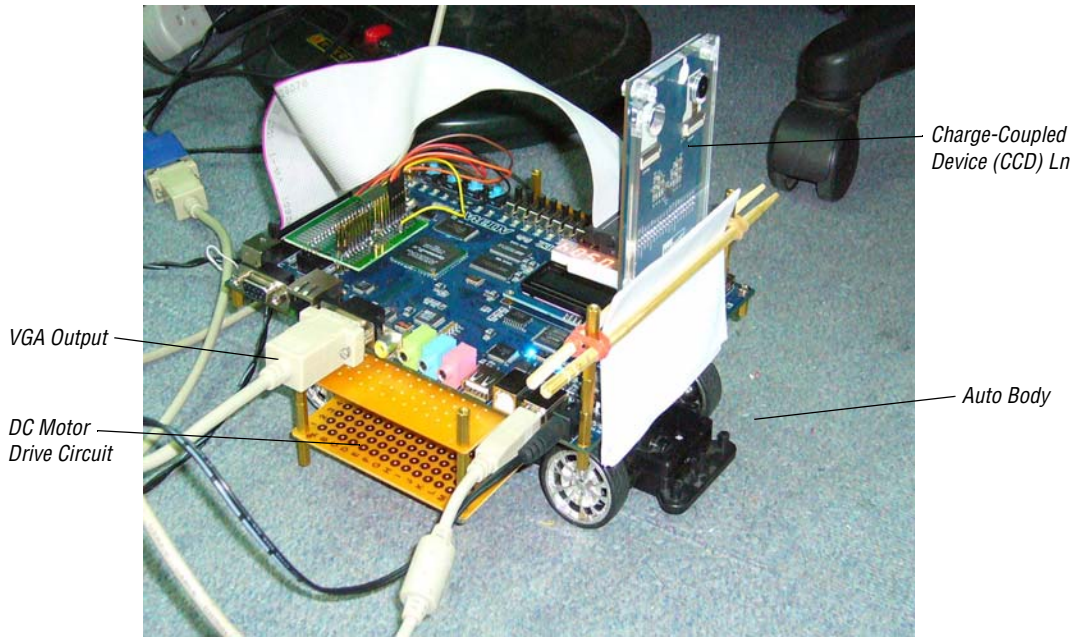
■ VGA hardware modules

- *VGA controller*—Uses its components to display images on VGA directly in real time.
- *XY histogram*—Marks the target's position with XY coordinates and makes a histogram of the X and Y axes for real-time images.

■ *Power control*—With a second soft-core CPU executing instructions from outside, the CPU drives the wheel's control circuits with four groups of programmable I/Os (PIOs), controlling whether the auto goes forward/backward or turns left or right.

Figure 2 shows a photo of the smart image-tracking auto.

Figure 2. Smart Image-Tracking Auto



Application Areas

The design can be used for the following applications:

■ Smart auto electronic devices applied in guidance systems for backing the autos into the garage, pulling over, and automated driving

■ AGV automated material handling systems

■ Automated airplane piloting and positioning to gates or runways

- Automated image and visualization control interfaces, such as input interfaces used for toy autos or boats, or control commands (e.g., video game handsets)
- Automated guidance for assistance equipment such as wheelchairs and electric autos

Target Users

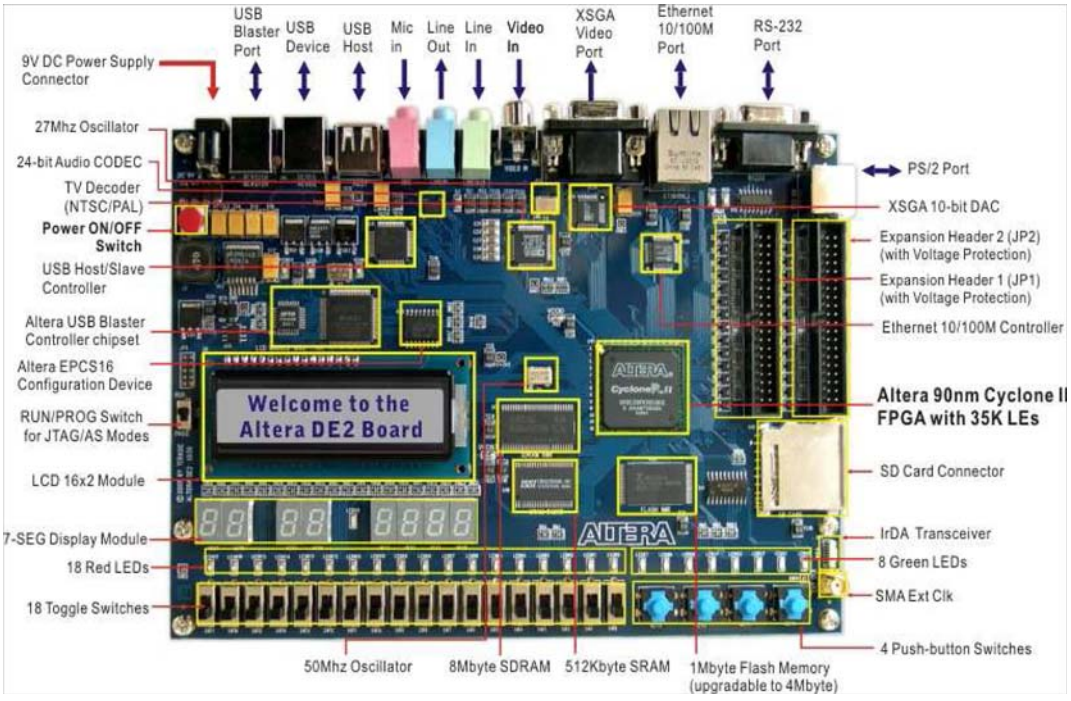
Our design targets the following users:

- Automobile electronics equipment manufacturers
- AGV automated guidance auto-controlled system vendors
- Airplane automated piloting and control system manufacturers
- Home entertainment product manufacturers
- Assistance equipment manufacturers

Development Board

For this design, we used the Development and Education (DE2) board, which includes an Altera® Cyclone® II EP2C35 FPGA with 35,000 logic elements (LEs), 8-Mbyte (1 Mbyte x 4 x 16) SDRAM, 4-Mbyte flash memory, a secure digital (SD) card interface, a USB master-slave controller with class A and B USB interfaces, a 10/100 Ethernet physical layer/media access controller (PHY/MAC), two serial connectors (RS-232 DB9 ports), etc. See Figure 3.

Figure 3. DE2 Development and Education Board



Function Description

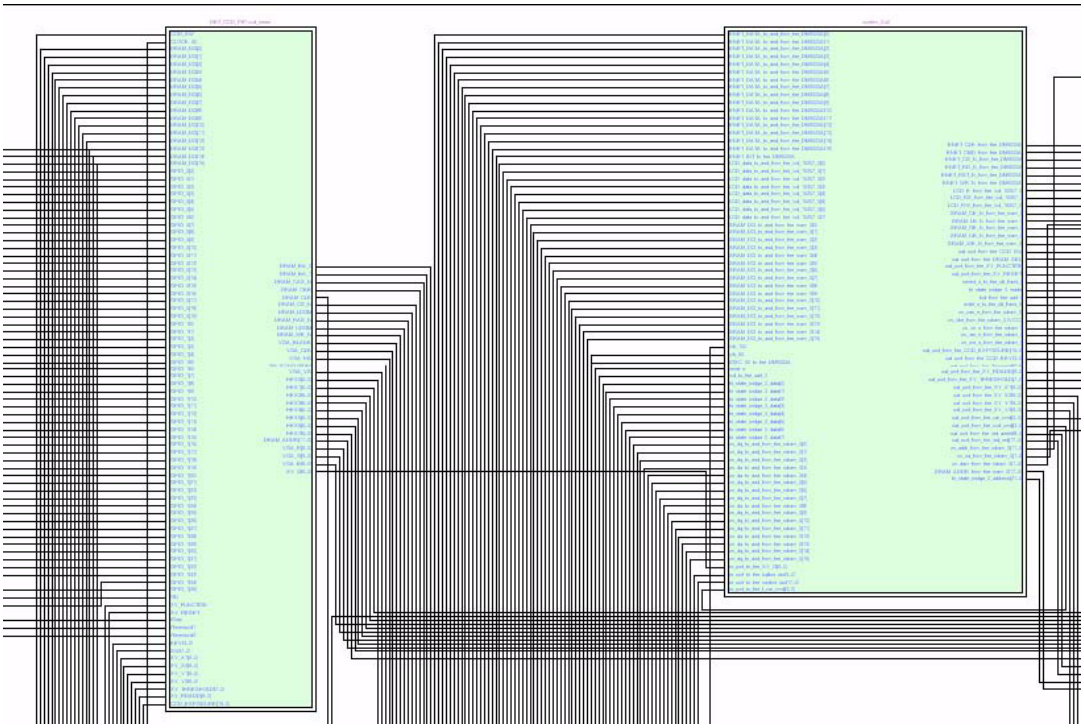
We used the Quartus® II software version 7.1 to design the Data Compress and XY Histogram cores of the smart vehicle guidance system. We created the cores using VHDL and Verilog HDL. Our design has the following functionality:

- Captures images with a CMOS sensor, reduces the number of images using data simplification, and displays them on-screen in real time via the VGA controller.
- Analyzes the reduced image data and generates statistics on the X and Y axis for analysis.
- Builds the whole system's infrastructure with SOPC Builder, complete hardware and software co-design, and 100% demonstration.
- Implements the self-controlled auto body guidance system and demonstrates automatic stop, left or right turn, automatic searching for special targets, etc.
- Implements the SDRAM controller with multiple interfaces and distributes the SDRAM writing and reading in a FIFO structure.
- Delivers a dual-CPU embedded system for the CMOS sensor image processing and auto body power control.

Major Hardware Components

We created the dual-CPU core with SOPC Builder, designed hardware circuit components in Verilog HDL, conducted timing simulation and verification with waveforms, and connected the CPU using the PIO method. In addition to the peripheral circuits provided by SOPC Builder, the design has a dual-CMOS sensor image capturing circuit, a 6-port SDRAM controller, and a VGA controller that contains the image processing circuit. Figure 4 shows the hardware circuit, which has been integrated into a bigger module (the block on the left of Figure 4). This block is a dual-CPU module created with SOPC Builder.

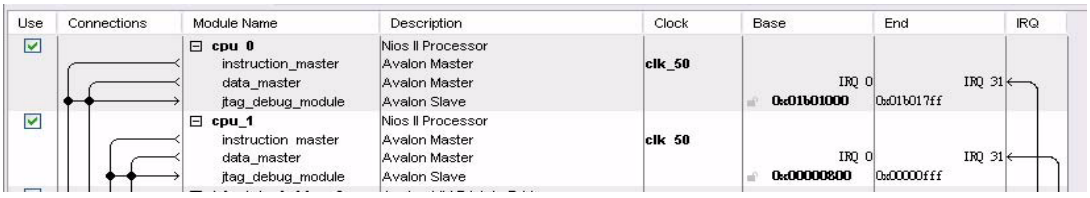
Figure 4. Hardware Components



Dual-Core Processors

Figure 5 shows the dual-core processors. cpu_0 controls the CMOS sensor and image processing, and cpu_1 controls the auto-controlled power.

Figure 5. Dual-CPU System



Dual-CMOS Sensor Grabber

The dual-CMOS sensor grabber compiles the control hardware circuit grabbed onto the lens image, and simultaneously grabs images from the dual lenses using two Integrated Development Environment (IDE) interfaces (expansion headers 1 and 2) on the DE2 development board.

Multi-Port SDRAM Controller

The multi-port SRAM controller generates six FIFO interfaces (implemented in embedded RAM), three for reading and three for writing, with the Quartus II MegaWizard® Plug-In Manager. The controller enables two groups of CMOS sensor grabbers and one group of VGA controllers to read and write to the SDRAM.

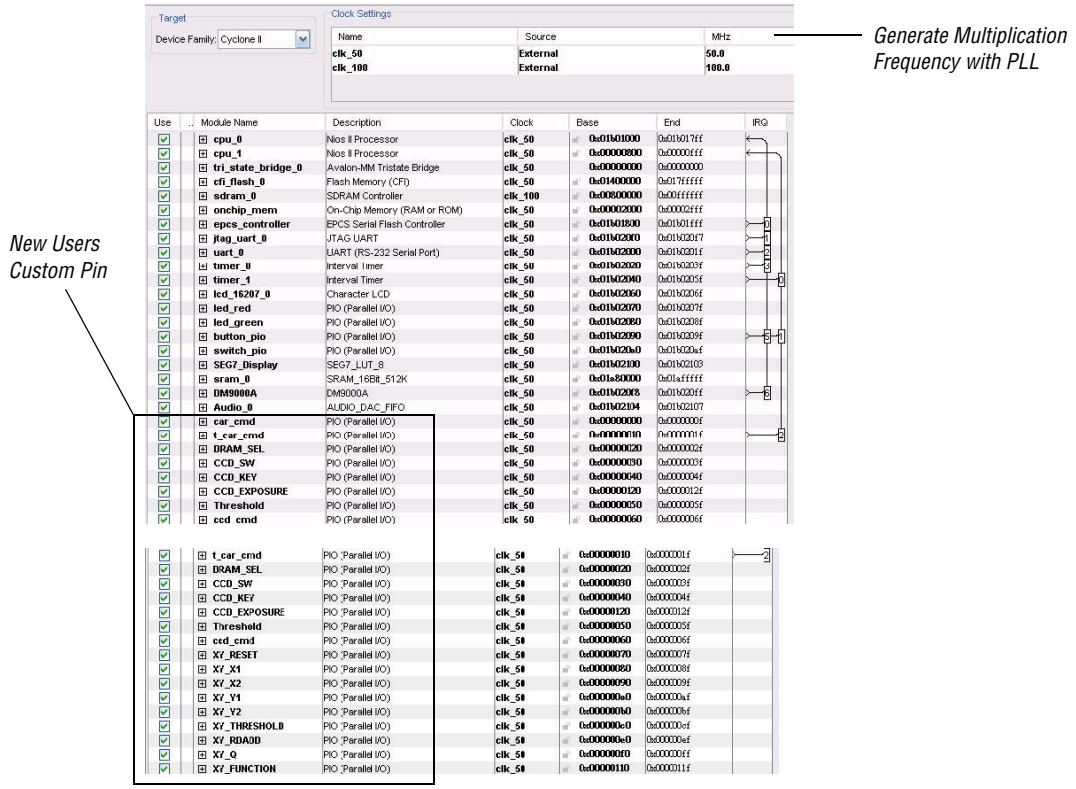
VGA Controller and Image Processing

The VGA controller and image processing function write the hardware control circuit output by the VGA monitor, generate image statistics on the X- and Y-axis while outputting images, and store the results in on-chip memory for the Nios II processor to read.

SOPC Builder Settings

We used SOPC Builder to create the Nios II system. For example, we added user-defined pins on the DE2 development board to control custom peripheral circuits, and used a phase-locked loop (PLL) to generate a 100-MHz clock source for the SDRAM. See Figure 6.

Figure 6. SOPC Builder Settings



Memory Configuration During System Software Execution

Because the CMOS sensor grabber and VGA controller occupy the development board's SDRAM, the cpu_0 and cpu_1 programs are stored in flash memory. The exceptional vectors of cpu_0 are put in SRAM at the time of program execution, and cpu_1 is put in on-chip memory. When we developed the CPU software in the Nios II IDE, we assigned variable stack areas to relevant memories, as shown in Figures 7 and 8.

Figure 7. cpu_0 Memory Configuration in SOPC Builder (up) and Nios II IDE (down)

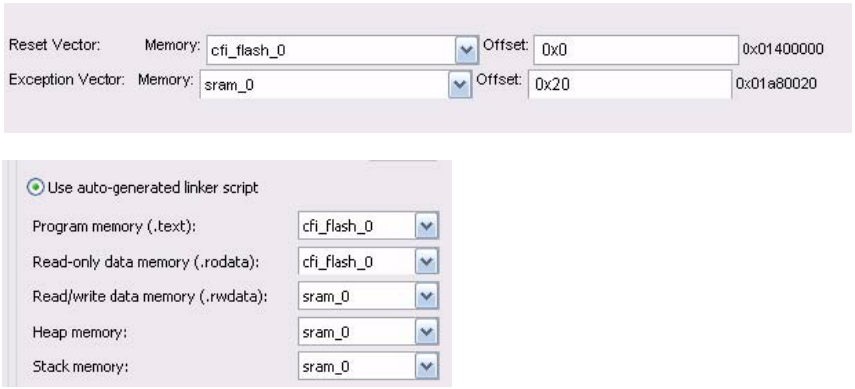
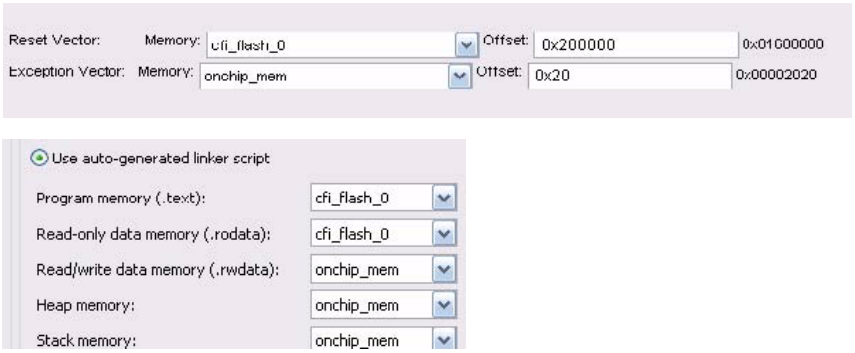


Figure 8. cpu_1 Memory Configuration in SOPC Builder (up) and Nios II IDE (down)



Performance Parameters

The application differentiates 640 x 480, 24-bit full-color real-time images for each of the 10 frames per second. It must process 8.78-Mbytes of data per second while conducting binarization and image processing of the X- and Y-axis histograms. Because the massive image information must be handled quickly, we used an architecture accelerated in hardware and controlled by software. Additionally, the SDRAM resources can be switched to the Nios II processor, and the Nios II processor can read and process the SDRAM image. When we tested the image processing algorithm, we also developed a PC simulation program with BCB (Intel 1.6-GHz dual-core, 1-Gbyte RAM, and 1.3 mega CMOS sensor). Table 1 compares the experimental data.

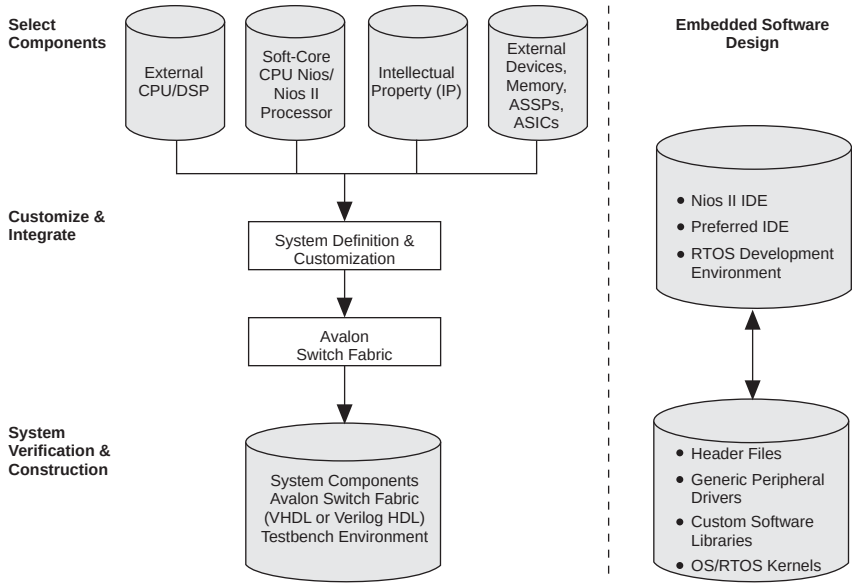
Table 1. Performance Analysis for Three Image Processing Platforms

	PC Software Simulation	Nios II Software	Nios II Software plus Hardware Acceleration
Frames handled per second	1 to 2	3 to 4	9 to 11
Notes	Developed with a Video For Windows (VFW) function for binarization, histogram analysis, etc.	Includes switching SDRAM master, reading SDRAM, binarization, histogram analysis, etc.	Includes reading SDRAM, binarization, histogram analysis, etc.

Design Architecture

Figure 9 shows the system development flow chart.

Figure 9. System Development Flow Chart



System Architecture

As shown in Figure 10, the design has a dual-core system: one CPU controls the CMOS controller module and another controls most peripheral components. Between the two CPUs are input and output pins for information communication. With this design, one CPU handles massive image information at full speed and the other operates the auto control system. Thus, when a deviation or a crash is detected in the images, the system triggers the intermission of another CPU via a PIO and informs the auto control system in real time that it should give commands to rectify the direction or avoid a bump. Peripheral components used in the design include: flash memory, SDRAM, SRAM, M4K RAM, LCM, JTAG-UART, RS-232, general-purpose I/O (GPIO), button, switch, timer, LED, segment, VGA, CMOS sensor, etc.

Figure 10. System Block Diagram

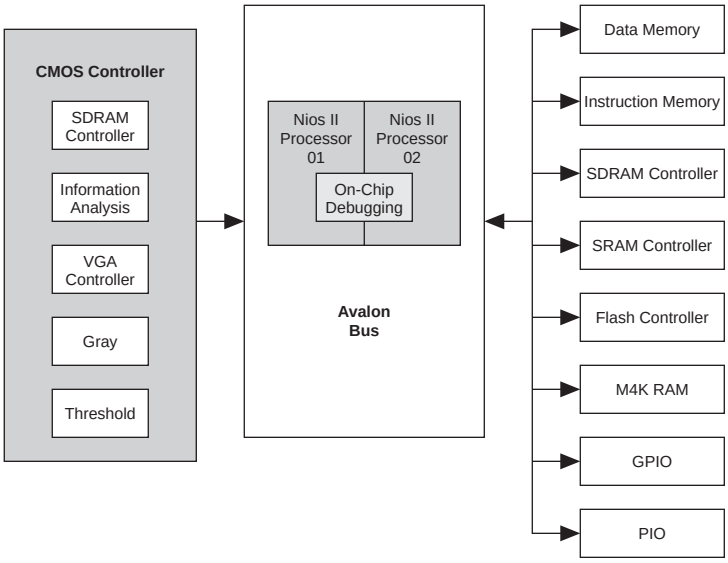
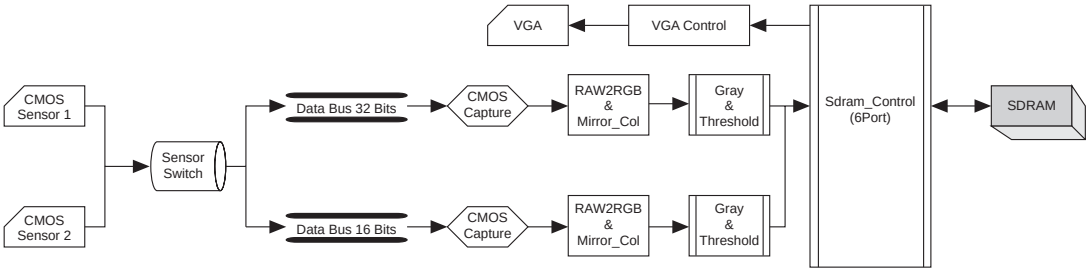


Image Processing

Figure 11 shows how the design supports dynamic real-time image tracking. When the CMOS sensor captures an image through hardware components, RAW data is converted into RGB. After binarization and grayscale application, the data is easy to process and the image is immediately shown on the VGA display. The whole process is performed in hardware. For image tracking, the Nios II processor marks the target using the X- or Y-axis histogram statistics. When a new image arrives, the Nios II processor can read out the values needed without doing anything, achieving the advantages of hardware acceleration.

Figure 11. Image Processing Block Diagram



Software Flow Chart

Figure 12 shows the software flow chart, which is described as follows:

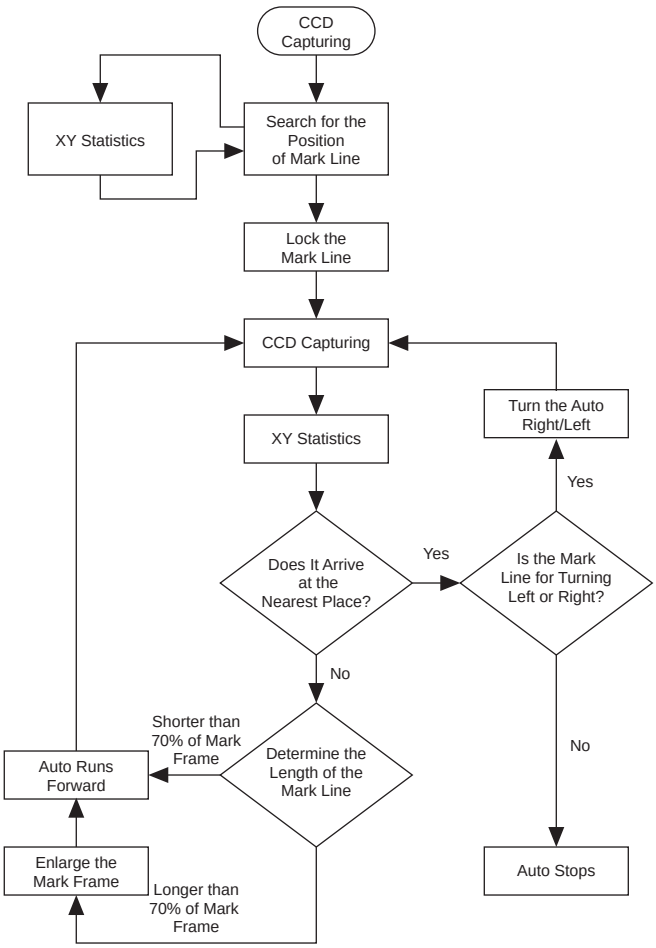
1. For computational acceleration, binarization and statistics are performed in hardware.
2. The system searches for the position of the target mark.
3. The system dynamically locks the mark.

4. The system determines the length of the mark and whether the distance between the self-controlled auto and the mark is the shortest. If it is not, the system determines whether the mark is longer than 70% of the mark frame. If it is, the system enlarges the mark frame.

5. The PIO sends the forwarding command to the self-controlled auto.

6. If the self-controlled auto is at the closest point to the mark, the system determines whether it is a mark for turning left or right. If it is, it turns left (or right) according to the mark; otherwise it stops the auto.

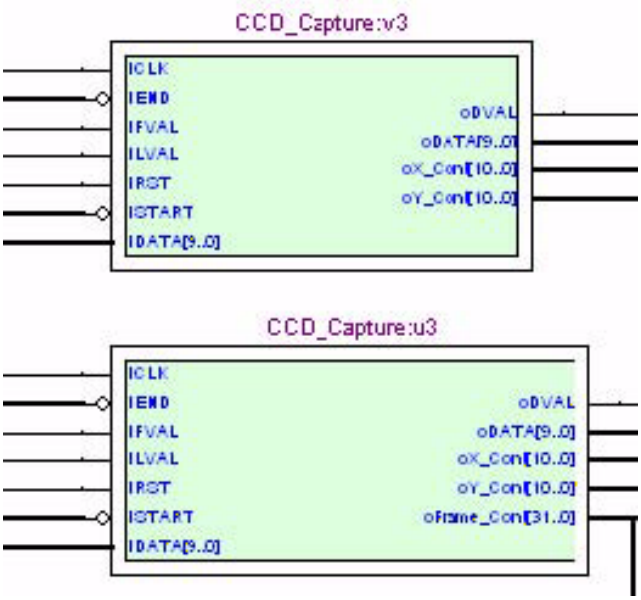
Figure 12. Software Flow Chart



Hardware Circuit Diagram

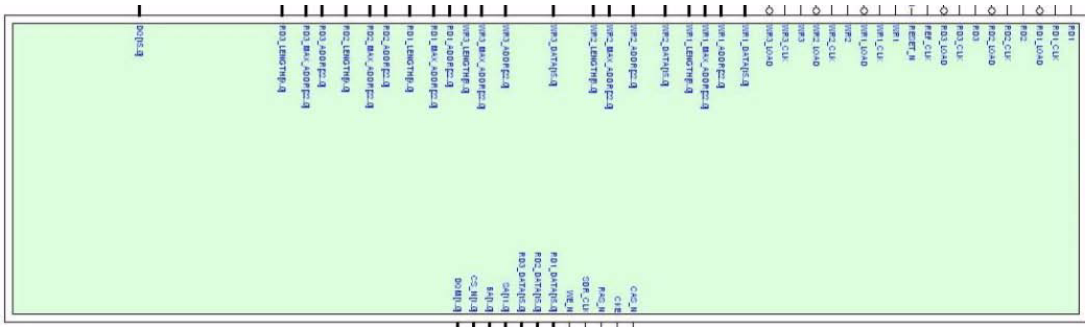
The dual-CMOS sensor image capturing component switches between main and sub-pictures. One of CMOS sensors controls the frame speed, by which the value output at once is 10 bits. Meanwhile, the pixel's X and Y axis are output for reading. See Figure 13.

Figure 13. Dual-CMOS Sensor Module



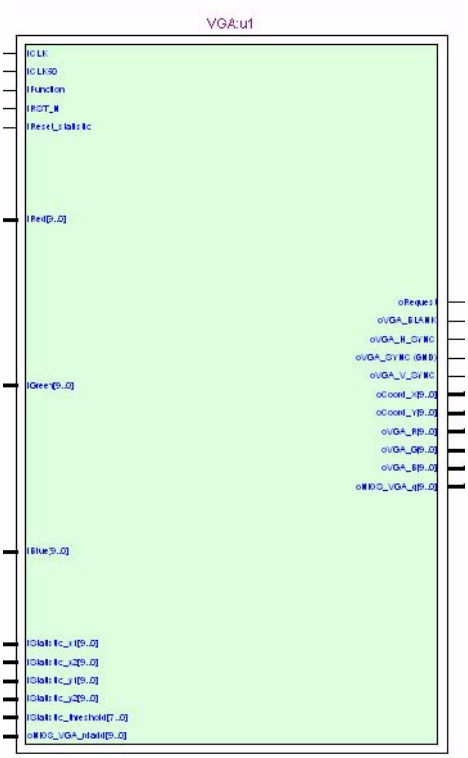
The multi-port SDRAM controller uses six FIFO buffers to provide three read and three write SDRAM controllers. Each FIFO is 2 Kbytes and is generated by M4K RAM. See Figure 14.

Figure 14. 6-Port SDRAM Controller



For the VGA controller and image processing functions (see Figure 15), the system combines the values read by SDRAM with the appropriate `H_sync` and `V_sync` signals, and sends out pixels one by one. Meanwhile, it generates the X- or Y-axis histogram statistics and stores the results in another M4K RAM block where the values can be read immediately when the Nios II processor needs them.

Figure 15. VGA Controller and Image Processing Modules



DC Motor Driver Circuit

The wheel's forward or backward rotation is controlled by a full bridge circuit. The Nios II processor controls the auto's body movement with `CAR_CMD [3..0]`, the PIO. `CAR_CMD[1..0]` is the back-wheel switch and `CAR_CMD[3..2]` is the front-wheel switch. Figure 16 shows a full-bridge circuit switch that controls forward or backward electrical flow for the back wheels (the case of the front wheels is similar). Table 2 shows the detailed control commands.

Figure 16. Back Wheel Full-Bridge Circuit Switch

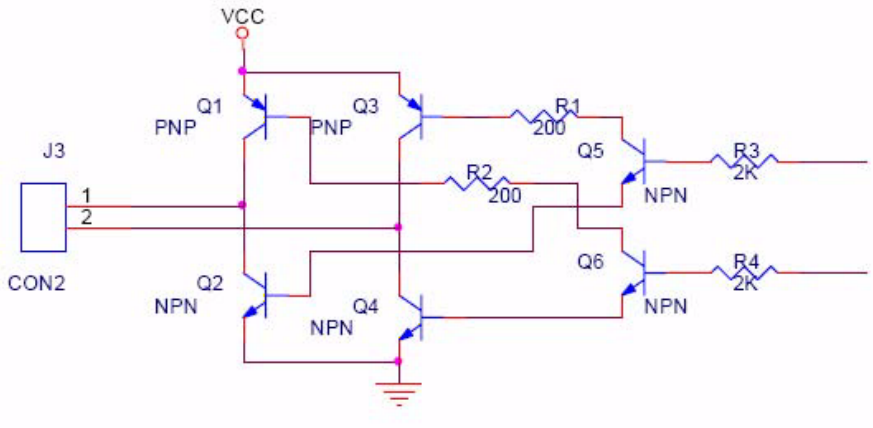


Table 2. Auto Control Commands

CAR_CMD[3]	CAR_CMD[2]	CAR_CMD[1]	CAR_CMD[0]	CAR_CMD	Car Body Movement
0	0	0	0	0x0	Stop
0	0	0	1	0x1	Forward
0	0	1	0	0x2	Backward
0	1	0	0	0x4	Front wheel turning right
1	0	0	0	0x8	Front wheel turning left
0	1	0	1	0x5	Right forward
0	1	1	0	0x6	Right backward
1	0	0	1	0x9	Left forward
1	0	1	0	0xA	Left backward

Test Scenarios

In Figure 17, the smart image tracking auto has locked onto a special target and is heading towards it. The image clearly shows that, in addition to the white arrow, there are many other white obstacles such as white walls, tissues, etc. Figure 18 shows the VGA output of the smart image tracking auto, and a binary image can be seen clearly. The smart image tracking auto locked onto the white arrow (within a green rim) and when the auto body moves forward, the green rim automatically grows bigger and locks onto the white arrow.

Figure 17. Smart Image Tracking Auto Moves Towards Target

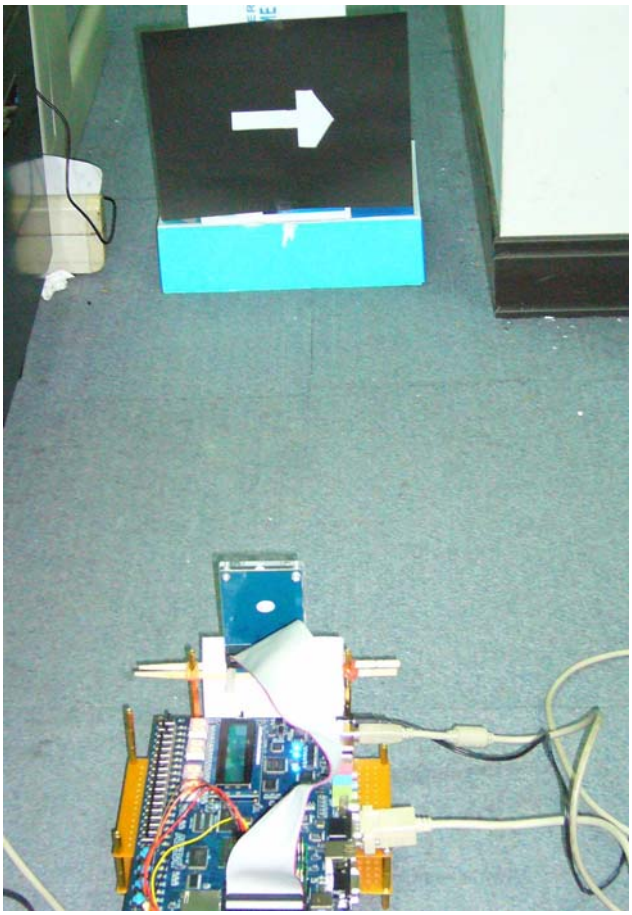
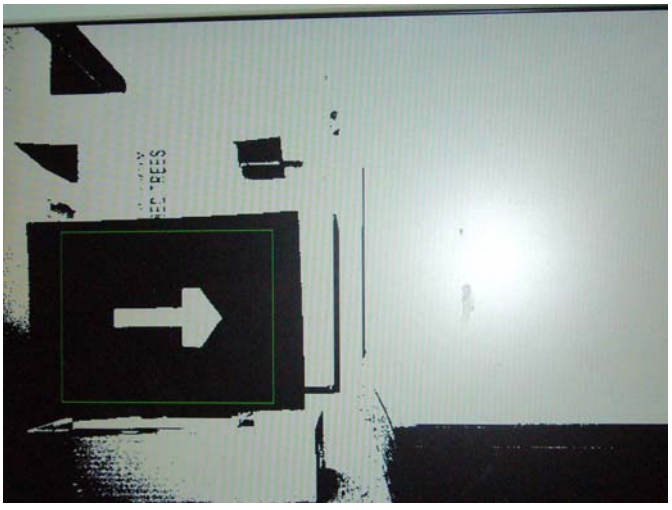


Figure 18. Target Locked by Forwarding Smart Image Tracking Auto



Design Methodology

This section describes our design methodology.

Implementation Method

We used the following implementation methods:

1. *Define the system*—We included processors, memory, peripheral components, and pins connecting the peripheral components.

2. *Generating system*—We produced a system **.ptf** file using SOPC Builder.

3. *Hardware design*—We compiled and set up the required components using Verilog HDL. We integrated, compiled, and simulated circuits, designed the auto body driver circuit, and measured the required electrical characteristics.

4. *Software design*—We used BCB to check the image processing algorithm, using the Nios II IDE to generate relevant header documents and drivers, and wrote system applications and compiled them into **.elf**.

5. *Simulation*—We used the ModelSim software for simulation. When we found a problem, we went back to step 2 and redesigned the software and hardware.

6. *Verification*—Using the JTAG interface, we downloaded the hardware and software to RAM or flash in the DE2 development board for physical verification.

7. *Test*—We combined the application with the auto body power control to conduct a product test on a rudimentary system.

Design Steps

To our knowledge, system-on-a-programmable chip (SOPC) design can provide an integrated, hardware/software platform with high elasticity. Because our design involves applying the embedded system to mechanical controls, the design planning consisted of four phases:

- *Define the input port and output ports*—Define modules needed by each stage from outside to inside, and determine their input or output pins. For example, the input port of the outermost system is the CMOS sensor while the output port is the VGA output and power control of the self-controlled auto. The inside modules include the CMOS grabber, SDRAM controller, VGA controller, Nios II CPU, automated auto driver module, etc.

- *Select cores and custom intellectual property (IP) components*—The design uses two full-edition CPUs. Aside from the peripheral interfaces attached to SOPC Builder, we needed to create the multi-port SDRAM controller and VGA controller.

- *Perform hardware and software system design*—We conducted the hardware and software design jointly, which was a challenge because software development involves planning and distributing hardware resources as well as the system performance. SOPC Builder enables an integrated development interface with high elasticity, which accelerated the system planning process. The Nios II IDE enables a complete environment for software development, including setting up break points, debugging, simulating instructions, etc., all of which accelerate product development.

- *Perform mechanical integration*—The design combines modules such as a DC motor, pulse-width modulation (PWM) sign control, full bridge switch, transistor enlarger circuit, etc. at which we are not adept (for we are majors in information engineering). However, being enthusiasts, we enjoyed the learning process and felt successful. We present our work using mechanical integration, allowing the prototype development system to be presented more dynamically.

Design Features

Our design has the following features:

- *Dual-CPU communication*—We used two CPUs to control and manage the smart image tracking system and obtained dual-CPU communication with I/O and interrupts.
- *100% hardware and software implementation*—We perfectly implemented the smart image tracking auto functions, such as target searching, target locking, auto body automated leading control, automated stop, etc.
- *Custom peripherals with hardware acceleration*—At first we developed the CMOS sensor controller and SDRAM controller with a hardware circuit; we stored the CMOS sensor image directly in SDRAM and made the SDRAM a Nios II master. But the experiments showed that the speed could not meet our requirements because the Nios II processor spent too much time reading SDRAM and image processing. Therefore, we developed circuits combing the VGA controller with image processing and implemented the VGA output and image processing at the same time. This design delivered the effect that we wanted.
- *Three IP functions connected outside of the Nios II processor*—Because the Nios II processor can be set with elasticity, PIO pins communicating outside can be easily designed according to the user's needs. By combining the hardware circuits such as the VGA controller, multi-port SDRAM controller, image processing function, etc., we improved the performance of the massive image data processing.
- *Fully uses the FPGA resources*—The design uses the Cyclone II EP2C35 high-capacity FPGA. However, because we used a dual-core CPU and developed many complex IP cores in which many components use M4K RAM (e.g., the FIFO buffer and image processing components), the whole system uses a total of 21,732 logic elements (LEs) (65%) and 347,248 RAM bits (72%), taking full advantage of the FPGA resources. See Figure 19.

Figure 19. FPGA Resources Used

Flow Status	Successful - Mon Sep 17 10:36:39 2007
Quartus II Version	7.1 Build 178 06/25/2007 SP 1 SJ Full Version
Revision Name	DE2_NET
Top-level Entity Name	DE2_NET
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	No
Total logic elements	21,732 / 33,216 (65 %)
Total combinational functions	19,275 / 33,216 (58 %)
Dedicated logic registers	12,472 / 33,216 (37 %)
Total registers	12477
Total pins	433 / 475 (91 %)
Total virtual pins	0
Total memory bits	347,248 / 483,840 (72 %)
Embedded Multiplier 9-bit elements	8 / 70 (11 %)
Total PLLs	1 / 4 (25 %)

Conclusion

We benefited greatly from participating in the Altera Nios II Embedded Processor Design Contest 2007. We divided our work into systems integration, hardware development, and control circuit design, which are described as follows:

- *System integration*—Thanks to the Nios II IDE and convenient SOPC Builder, we implemented the soft-core CPU in the prototype machine with a flexible design, accelerating the development process. With quick executive efficiency provided by a dual-core PC, the new Quartus II version 7.1 software and Nios II IDE version 7.1 largely reduced the time we had to wait for hardware integration or software compilation.

- *Hardware development*—Thanks to the contest, we are now acquainted with SDRAM controller design. With timing adjustments, we designed multiple high-speed peripherals that share one SDRAM device. Besides the VGA controller, we learned how to process real-time images while outputting image pixels, etc.

- *Control circuit design*—We are honored to participate in this Nios II design contest. Although the challenge of working with unfamiliar mechanical controls gave us a lot of trouble, we finally saw the clumsy, smart image-tracking auto operate on various surfaces. This contest brought all members in our lab together to solve problems jointly. Thank you for providing young students with such a precious opportunity to make our dreams come true.

Star Award

RTOS Acceleration Using Instruction Set Customization

Institution: **Centre for High Performance Embedded System (CHiPES), Nanyang Technological University (NTU)**

Participants: **Muhamed Fauzi Bin Abbas, Ku Wei Chiet**

Instructor: **Professor Thambipillai Srikanthan**

Design Introduction

As embedded system designs become increasingly more complex, the use of real-time operating systems (RTOS) becomes essential to meet time-to-market pressures and to contain non-recurring engineering costs. However, an RTOS consumes precious CPU cycles in return for the services it provides. Further, the RTOS is typically treated as a pure software entity and is subject only to minor adaptations. In this project, we leverage work done by our research group towards instruction set customization for RTOS acceleration using dedicated hardware and apply it to the Nios® II processor. We present our design, findings, and results in this paper.

In this project, we implemented RTOS acceleration by customizing the Nios II instruction set. As part of the process, we identified RTOS operations that are executed frequently and converted them into custom instructions (CIs), thereby collapsing a series of instructions into fewer operations and reducing the RTOS CPU overhead.

By reducing the time consumed by system tasks, greater processing time is made available for user applications. The system becomes more responsive, and this effect is further noticeable if the CPU is clocked at a lower clock frequency, where the same number of clock cycles constitutes a higher percentage of system overhead. By allowing more time for user tasks, RTOS acceleration benefits applications that are run on the system without changing the applications. This acceleration provides a drop-in method for improving system performance and responsiveness.

The effect of our work is very noticeable in systems that run a large number of tasks. For our project we use the Nios II/s CPU and μC/OS-II.

We decided to use the Nios II processor because of the following factors:

- The Nios II processor is a soft-core CPU that allows instruction set customization. In our opinion, using instruction set customization for RTOS acceleration is extremely attractive because the instruction execution is serialized with respect to other instructions, retaining the sequential nature of the original function being accelerated. However, all parts of the custom instruction can execute in parallel when the instruction is executed. Finally, this instruction can be accessed either in assembly language or as an intrinsic function from a high-level language such as C or C++, making integration with the software RTOS simpler.

- SOPC Builder makes it extremely easy to add custom instructions to the CPU core. In the future, we plan to explore the use of the C-to-Hardware Acceleration (C2H) Compiler for this work. This implementation would potentially make it easy to create new RTOS customization instructions because the RTOS itself is typically written in C.

At the same time, it seems that the Nios II processor is typically targeted at medium complexity applications (about a 100-MHz clock with multiple tasks). We found that in this configuration, RTOS overheads can be very significant in systems with a large number of tasks. These types of systems stand to benefit significantly from our work. However, as we shall demonstrate later in the paper, our proposed design can be easily parameterized to cater to different workloads and requirements.

Function Description

For this project, we used µC/OS-II running on the Nios II/s processor. By analyzing the RTOS, we determined that the two most frequently executed portions are:

- *Timer tick routine*—The timer tick routine executes at the system clock frequency, which typically ranges from 100 Hz (10 ms) to 1,000 Hz (1 ms). Because all RTOS operations are expressed relative to this parameter, the system clock frequency has a direct impact on the system resolution. This routine runs as part of the interrupt service routine (ISR) in response to the timer tick, therefore, it is executed very frequently, and acceleration results in significant savings.

- *Scheduler*—The scheduler is called every time the status of one or more tasks changes in the system. Further, some portion of the scheduler is called with interrupts disabled. Any improvement in these areas results in reduced interrupt latency. In general, the scheduler is called as the final operation of many system calls and acceleration improves the system call processing time.

µC/OS-II handles 64 tasks with unique task priorities. The task priority is the same as the task ID.

Time Management Module

The timer services offered by µC/OS-II consist of timeouts and delays. The timer services' basic unit of measurement is the system timer tick, which is generated as a periodic interrupt by a hardware timer in the system. Timer management affects the following areas:

- All delays and timeout requests are expressed in number of ticks. µC/OS-II, which has a 16-bit internal timer delay variable, supports delays of up to 65,535 ticks.

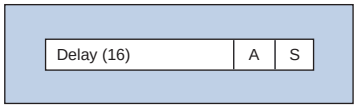
- When a timer tick occurs, the tick ISR decrements the waiting task's delay variable. If the variable reaches 0, the task can be added to the ready list.

- If the task is suspended when its delay value reaches 0, the task delay value is reloaded with 1 to delay it by another tick. This process continues until the task is resumed.

The timer module's basic architecture comprises a 16-bit counter to store the delay value for each supported task. Additionally, two extra bits are stored for every task: one bit stores whether the delay

counter is active and the other stores whether the task is suspended. Figure 1 shows the basic unit for each task, which is replicated for the number of tasks expected in the system.

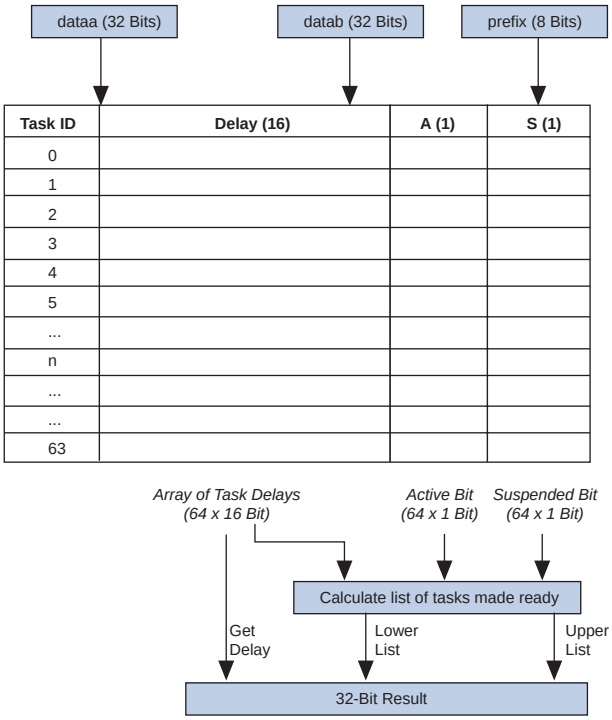
Figure 1. Timer Management Basic Unit Architecture



Name	Width	Function
Delay	16	Delay counter.
A	1	Active. Indicates whether the counter is in use.
S	1	Suspended. Indicated whehter the task is suspended.

Figure 2 shows the architecture for a full implementation with 64 units. The instruction usage is described next.

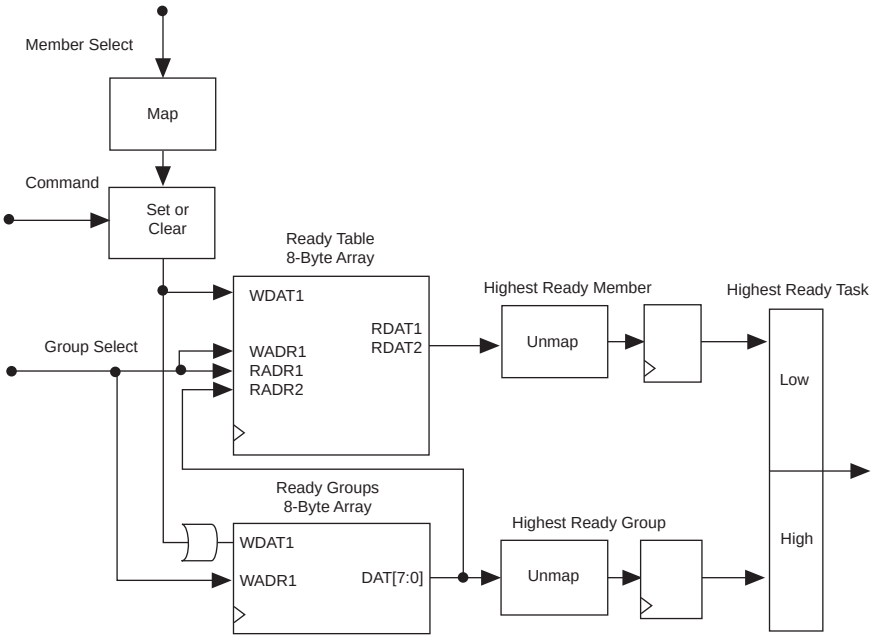
Figure 2. Custom Instruction for 64 Tasks



Scheduler Module

In an associated project in our research group, the basic scheduler was accelerated using instruction set customization [1]. Figure 3 shows the architecture diagram for that implementation. We implemented the scheduler so that it could be instrumented and tested in the same manner as the other work done in this project.

Figure 3. Scheduler Custom Instruction Architecture Block Diagram



Performance Parameters

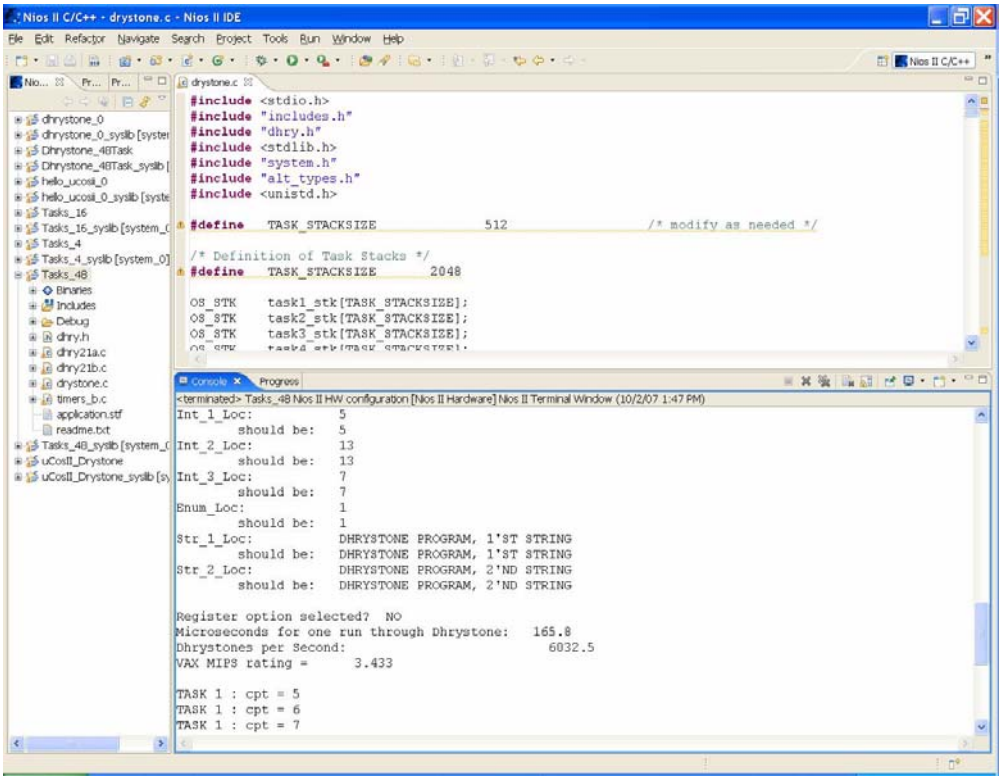
The design uses 3,748 registers and 41% of the logic elements (LEs) available in the Cyclone® II device on the Development and Education (DE1) board, as shown in Table 1. Details about the hardware consumption are discussed later in this section.

Table 1. Design Resource Usage

Resource	Description
Flow status	Successful - Tue Oct 02 09:52:15 2007
Quartus II version	7.1 Build 156 04/30/2007 SJ Full Version
Revision name	DE1_SD_Card_Audio
Top-level entity name	DE1_SD_Card_Audio
Family	Cyclone II
Device	EP2C20F484C7
Timing models	Final
Met timing requirements	Yes
Total logic elements	7,749 / 18,752 (41%)
Total combinational functions	7,278 / 18,752 (39%)
Dedicated logic registers	3,631 / 18,752 (19%)
Total registers	3748
Total pins	287 / 315 (91%)
Total virtual pins	0
Total memory bits	47,104 / 239,616 (20%)
Embedded multiplier 9-bit elements	4 / 52 (8%)
Total phase-locked loops (PLLs)	1 / 4 (25%)

Figure 4 shows the debug messages sent from the DE1 board to the PC console through the JTAG connection. The messages show how we benchmarked the system using the Dhrystone benchmark program.

Figure 4. Debug Messages



The custom instructions need to be carefully added into specific parts of the RTOS and are called from multiple places in the system. Therefore, there is an overall system performance impact from our project. In this section, we report the main results and improvements by using custom instructions for RTOS acceleration.

Performance Improvement of Specific Functions

Although the custom instructions are used in many places in µC/OS-II, their main impact is seen in a few specific functions.

Timer Tick Custom Instruction

For the timer tick custom instruction, the main impact is in the timer tick ISR (OSTimeTick). At every system timer interrupt (typically every 1 or 10 ms), the timer tick ISR is run in response to the timer tick interrupt.

The timer tick ISR has 2 main operations:

- The wait count of each task is decremented by 1 by iterating through the task control blocks for all tasks present in the system.
- For unsuspended tasks whose wait count becomes 0, the scheduler is called to update the task's ready list.

The timer tick custom instruction only affects the decrement operation. Because the kernel iterates through the waiting tasks, the function's execution time directly depends on the number of tasks in the system. Table 2 shows the results for the OSTimeTick module. The performance improvement

(calculated as a percentage) is shown in brackets. We tabulated the results with 4 tasks (light load), 16 tasks (medium load), and 48 tasks (high load) in the system.

Table 2. Timer Tick Performance

CPU Types (# of Tasks)	Software *(1)*		Hardware (with Custom Instructions) *(2)*	
	st *(3)*	lt *(4)*	st	lt
4 Tasks	584	612	107 (81.60%)	183 (70.10%)
16 Tasks	921	2525	249 (72.90%)	298 (88.20%)
48 Tasks	4,057	4,844	219 (94.60%)	260 (94.60%)

Notes:

(1) The Software column indicates performance when only the pure software RTOS is executed (i.e., without any custom instruction support).
(2) The Hardware (with Custom Instructions) column indicates performance when the RTOS has been accelerated using the custom instruction.
(3) st refers to the typical shortest time spent in the execution of the relevant portion.
(4) lt refers to the typical longest time spent in execution.

As shown in Table 2, the performance improvement is in the range of 70% to 81%, even for systems with a light load. As the number of system tasks increases, the software RTOS starts to show very high overhead. In contrast, the RTOS supported by the custom instruction scales better and less time is consumed in the ISR, showing more than 90% improvement.

Scheduler Instruction

The scheduler instruction significantly affects many areas of the RTOS. The main impact is noticed in OS_Sched. As with the timer tick, the performance depends on the number of system tasks. Table 3 shows the performance data (see Table 2 for the column definitions). The typical improvement is 40%.

Table 3. Scheduler Instruction Performance

CPU Types (# of Tasks)	Software		Hardware (with Custom Instructions)	
	st	lt	st	lt
4 Tasks	295	423	140 (52.50%)	251 (40.66%)
16 Tasks	287	440	154 (46.34%)	269 (38.86%)
48 Tasks	286	437	154 (46.15%)	268 (38.67%)

Dhrystone Benchmark

While the performance improvement in individual functions is impressive, it is not representative of the performance improvement that a system will actually experience because that depends on the custom instruction usage in the actual system. However, because these instructions are part of the RTOS, some improvement can be measured without actual applications.

The Dhrystone benchmark is typically used to measure the CPU performance with respect to the amount of work it performs. In recent years, Dhrystone use for comparing CPUs has diminished. However, it is still a viable measure of the amount of work that the CPU can perform.

In our case, we are measuring the performance of the same CPU with different amounts of hardware to offload certain tasks. We are trying to measure the ability of the same CPU to do work when supported by the custom instructions. On the same CPU, the Dhrystone benchmark is affected only by the number of CPU cycles that are available for the software to execute. Therefore, we can use the Dhrystone as a

benchmark to measure of the amount of work that can be performed as a result of adding RTOS custom instructions.

Table 4 shows the Dhrystone benchmark for the system running at 50 MHz with the Dhrystone task running at medium priority. Results are reported for the following four system types with a different number of system tasks because the number of tasks impacts the RTOS primitive execution times:

- *Original RTOS*—The system without any custom instructions.
- *Scheduler modified*—The system includes the scheduler custom instruction.
- *Timer modified*—The system includes the timer tick custom instruction.
- *Combined modified*—The system includes both custom instructions.

The numbers are reported for the system with timer interrupts occurring at 100 Hz (10 ms) and 1,000 Hz (1 ms). The corresponding percentage improvement is shown in parentheses. As required by the Dhrystone benchmark, we did not apply a compiler optimization. As Table 4 shows, the performance is improved by nearly 54%.

Table 4. Dhrystone Tests without Compiler Optimization

# of Tasks	Original RTOS		Scheduler Modified		Timer Modified		Combined Modified	
	100 Hz	1000 Hz	100 Hz	1000 Hz	100 Hz	1000 Hz	100 Hz	1000 Hz
16	3.995	3.257	4.001 (0.15%)	3.305 (1.47%)	4.023 (0.70%)	3.587 (10.13%)	4.045 (1.25%)	3.722 (14.28%)
48	3.486	2.153	3.491 (0.14%)	2.202 (2.28%)	3.592 (3.04%)	3.194 (48.35%)	3.604 (3.38%)	3.316 (54.02%)

Although we collected the data in Table 4 without compiler optimization, we feel that it is not representative of commercial systems. Table 5 presents the same numbers when compiler optimization (O2) is applied. Although the differences are not as large as before, we can see that the Dhrystone benchmark does improve by up to 9.4%. Therefore, the system can perform almost 10% more work when RTOS acceleration is applied.

Table 5. Dhrystone Tests with Compiler Optimization

# of Tasks	Original RTOS		Scheduler Modified		Timer Modified		Combined Modified	
	100 Hz	1000 Hz	100 Hz	1000 Hz	100 Hz	1000 Hz	100 Hz	1000 Hz
16	4.014	3.787	4.023 (0.22%)	3.806 (0.50%)	4.019 (0.12%)	3.85 (1.66%)	4.026 (0.30%)	3.908 (3.20%)
48	3.536	3.159	3.538 (0.06%)	3.18 (0.66%)	3.56 (0.68%)	3.4 (7.63%)	3.566 (0.85%)	3.456 (9.40%)

Hardware Resource Requirements

The base hardware was designed to support 64 system tasks. Table 6 shows the resources (logic cells and registers) required for the scheduler and the timer.

Table 6. Hardware Requirements

Optimization	Logic Cells	Registers
Scheduler	592	80
Timer	3,657	1,184

The hardware data is for a custom instruction that supports all 64 system tasks. However, it is possible to obtain comparable performance while consuming less hardware based on application-specific parameters because the Nios II processor is a soft-core processor and we can create the CPU to match the specific application requirements. The system has two main optimization areas:

- µC/OS-II is extremely customizable and we can restrict the number of tasks that are present in the system. Depending on the application requirements, we can support fewer tasks. Similarly, we can restrict the number of tasks that require hardware. For example, if the system needs to support 16 tasks, we only need 16 counters in the timer custom instruction. This method allows us to achieve the same performance without consuming too much hardware.

- In software, it is convenient to use entities that are multiples of 8 bits. Therefore, the µC/OS-II timer wait value is 16 bits. A system that ticks at a rate of 1,000 Hz waits for a maximum of approximately 65 seconds. A system that ticks at a rate of 100 Hz waits for a maximum of 655 seconds. For a system that typically has small waits, we can reduce the timer tick custom instruction counter width. For example, a 12-bit counter allows a wait of more than 4,000 ticks (4 seconds at 1,000 Hz or 40 seconds at 100 Hz). µC/OS-II already supports delays longer than the time supported by the resolution of the counter. By adapting that area, we can use a smaller hardware counters width, thereby reducing the hardware requirements.

Table 7 shows the hardware resources required for different numbers of tasks and counter resolution for the timer tick custom instruction. The hardware required to support 16 tasks at 12-bit resolution is approximately 20% of the custom instruction (23% logic cells and 20% registers) that supports all 64 tasks at 16-bit resolution.

Table 7. Hardware Results

Tasks	Timer Resolution (bits)	Logic Cells	Registers
64	16	3,657	1,184
64	12	2,834	928
24	16	1,301	432
16	16	917	304
16	12	865	240

This result is possible because the Nios II processor is a soft-core CPU and can be easily customized to closely match the CPU to the application requirements. The custom instructions can be easily parameterized, i.e., SOPC Builder can generate the custom instruction with the correct bit-width and task support the designer requires.

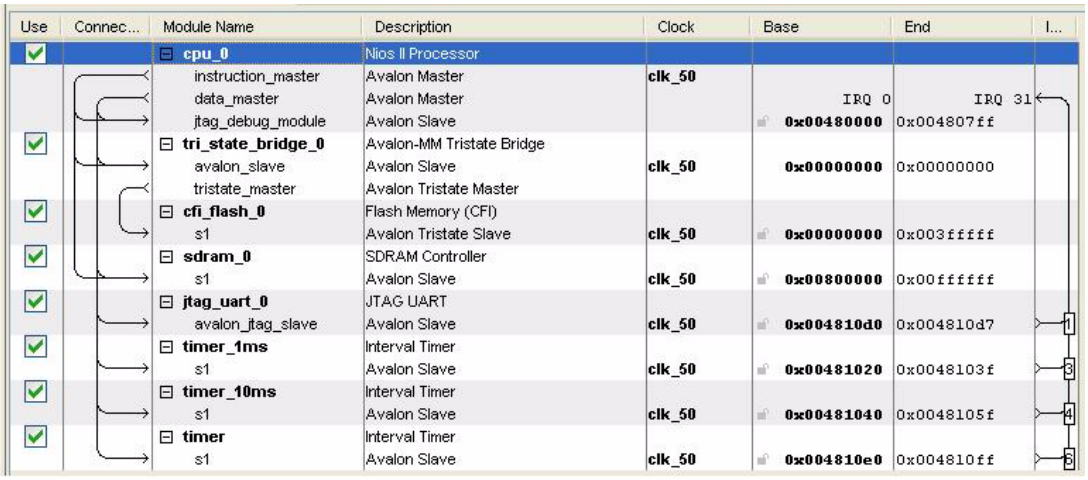
Design Architecture

We constructed our basic Nios II platform with µC/OS-II obtained from the Nios II Integrated Development Environment (IDE) Project Wizard. We developed the custom instruction for the time

management and scheduler modules and then we modified the µC/OS-II source code to work with our custom instructions. We also adapted and used the Dhrystone code that came with the Nios II IDE to benchmark the system's performance improvement.

The main CPU is the Nios II processor, which was clocked at 50 MHz as the reference design. The design uses the Nios II/s processor, flash memory, and SDRAM controllers, which are connected to the Nios II processor by a tri-state Avalon® bridge, `ext_ram_bus`. We used additional modules, such as a timer, JTAG UART, etc., to run and debug the Nios II processor. Figure 5 shows the hardware modules in SOPC Builder.

Figure 5. Nios II Processor and Hardware Modules in SOPC Builder



We designed the following two custom instructions to make the system faster:

```
#define ALT_CI_TIMER (n, A, B) __builtin_custom_inii(0x20+(n&((1<<5)-1)), A, B)
#define ALT_CI_SCHD (A, B)  __builtin_custom_inii(0x0, A, B)
```

The custom instruction depends on a number of input parameters that cannot be passed to it directly. Additionally, the design requires functions to load and store values inside the instruction. Therefore, we set up the instruction to perform a number of different operations. The function is selected by changing the value of the prefix parameter. Because the custom instruction stores values, it requires two clock cycles to run. The time management custom instruction has the following operations:

- Set delay for task n.
- Clear delay for task n.
- Get delay for task n.
- System tick reduces all active counts by 1.
- Update suspended tasks and read status of tasks 0 to 31.
- Update suspended tasks and read status of tasks 32 to 63.
- Clear result registers.
- Update active bit for all tasks.

- Mark task n as suspended.
- Mark task n as unsuspended.

The scheduler custom instruction has the following operations:

- Put a task into ready list.
- Remove a task from ready list.
- Find the highest priority task from ready list.
- Return the ready group value.
- Clear all the task bits from ready list.
- Get the bit mask from 0 to 7.
- Get the bit information from OSUnMapTbl.
- Get one element value of OSRdyTbl[].

Design Features

Our design has the following main features:

- *Significantly reduces RTOS overhead*—While RTOS usage is considered essential in many modern CPU-based systems, the overhead can be significant. Our approach helps contain it.
- *More time for user tasks*—As demonstrated, properly using these instructions results in more time for user tasks, thereby allowing the same system to do more work at the same frequency. Alternatively, the system can be clocked at a lower frequency.
- *Improved determinism*—When using custom instructions, the execution time does not increase at a very steep rate as the number of system tasks increases. This situation provides better scalability and improved determinism because the custom instruction consumes almost the same amount of time, regardless of the number of tasks.
- *Potentially reduces interrupt latency*—Code that uses the custom instructions often runs in system areas that are either in ISRs or in areas that run with disabled interrupts. Reducing execution time of these modules reduces the system's worst-case interrupt latency.
- *Parameterized instructions*—The design can be easily parameterized, allowing the designer to tailor the custom instructions to the target application's specific requirements. This process is made even simpler by including SOPC Builder options such that the RTOS design is also generated when producing the designer's CPU software development kit (SDK).
- *Drop-in optimization*—Application-specific hardware/ software partitioning is a time-consuming task that requires many design and test iterations. In CPU-based software-intensive systems that require an RTOS, our approach offers an attractive method to improve the system performance by simply reducing the RTOS overhead, an area that system designers seldom touch. Additionally, because the custom instructions are independent and parameterized, the designer can utilize unused FPGA resources in a final system for RTOS acceleration (the hardware requirements can be managed by selecting the specific instruction, the number of tasks, and their timer solutions, etc.), which has a positive impact on the whole system. Because the RTOS and instruction

generation can be automated, this option can be provided as a customizable, pre-verified, drop-in optimization.

- *Wide applicability*—Because our work makes more CPU time available to user tasks, it is applicable to any system that is based on a customizable CPU on an FPGA, regardless of the target application. All software-based systems can benefit from our work.

Conclusion

This project builds on work that was done previously in our research group using the Nios II processor and an older version of µC/OS-II. Since then, the Quartus II software and SOPC Builder have evolved significantly, making a number of tasks easier. It is interesting to note that while the participants in our project group were mentored and supported by the people who did the earlier research, one of us used the Nios II processor for the first time in this project.

The project was a cumulative learning and knowledge enrichment experience regarding the Nios II processor and FPGAs. In addition to the fact that we achieved RTOS acceleration, most applications can reap immediate benefits when run on the modified RTOS.

In our opinion, the Nios II processor is very useful for embedded system engineers and is also an excellent tool for research engineers. The designer can easily change external peripherals and interfaces from within SOPC Builder, making a seamless interface between processor and hardware logic. Instead of passively adapting to the hardware processor, we can customize both hardware and software, particularly when using Nios II custom instructions that make the system much more flexible than using standard processors. The Avalon bus quickly helped us connect all the modules required in our system and made it just as easy to improve the system later when we realized that we lacked something.

While implementing the project, we learned the following things:

- SOPC Builder is a very flexible, powerful tool that allows even a software engineer to learn and design a hardware system quickly and efficiently on any Altera FPGA.

- We spent some time understanding the differences between the Nios and Nios II processors because we migrated some of the designs from a previous project.

- We found the Nios II IDE Project Wizard examples very useful because IDE generated the µC/OS-II and Dhrystone elements we used in this project.

- We found that the µC/OS-II source code is not copied to each project folder but instead uses the main copy in the Nios II IDE root folder, which made it difficult for us to switch projects between our optimized and regular RTOS. This setup is most likely due to the fact that most people treat the RTOS as a software entity that is not touched as part of the system design process.

- We were able to implement the accelerated RTOS using two custom instructions for the competition. It is unfortunate, however, that we did not have sufficient time to implement an event control block (ECB) custom instruction, which would have further improved the system performance (particularly for system synchronization and communication). In the future, we hope to integrate this custom instruction into the Nios II IDE, allowing developers to utilize the instruction with the Nios II Project Wizard. This modification will make optimization even more seamless. Additionally, we want to explore the C2H Compiler in the future to see if it can further ease the process of integrating modules for RTOS acceleration.

References

The references are for our group's relevant research publications.

[1] T F. Oliver, D L Maskell, "Accelerating an Embedded RTOS in a SOPC Platform," Proceedings of the annual technical conference of the IEEE Region 10 (TENCON), November 21 - 24, 2004.

[2] M Sindhwani, T Oliver, D L Maskell and T Srikanthan, "RTOS Acceleration Techniques - Review and Challenges," Proceedings of the Sixth Real-Time Linux Workshop, Singapore, pp. 123-128, November 2004.

[3] Z Jin, M Sindhwani and T Srikanthan, "RTOS Acceleration on Soft-core Processors Using Instruction Set Customization," 2004 IEEE International Conference on Field Programmable Technology (FPT 2004), Australia, pp. 371-374, December 2004.

[4] M Sindhwani and T Srikanthan, "Framework for Automated Application-Specific Optimization of Real-Time Operating Systems," Fifth International Conference on Information, Communications and Signal Processing (ICICS 2005), Thailand, pp. 1416-1420, December 2005.

Second Prize

Aerial Photographic System Using an Unmanned Aerial Vehicle

Institution: **Chungbuk National University**

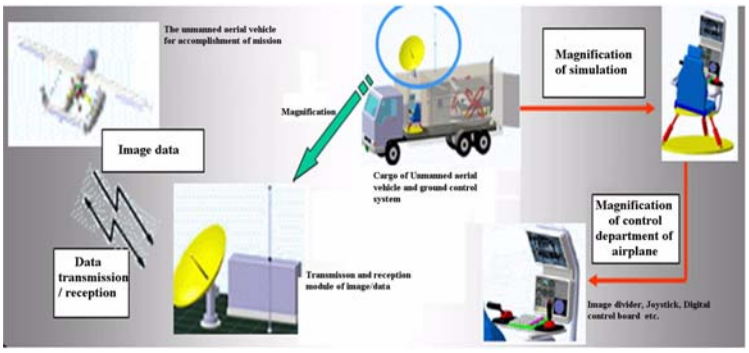
Participants: **Hyuk Joong Kwon, Woo Joong Kim, Jang Geun Kim, Sang Bae Park**

Instructor: **Professor Jung-Kwan Seo**

Design Introduction

While many software applications support JPEG image compression, few FPGA applications support it, and those that do are very expensive. Currently, software implementations of JPEG compression use many operations, and adding an algorithm slows performance. Our system implements JPEG compression using an FPGA, which improves the compression rate and operation speed. We implement JPEG compression using the Nios® II soft-core processor, which also performs aerial photography on an unmanned aerial vehicle (UAV). Figure 1 shows the overview of the project.

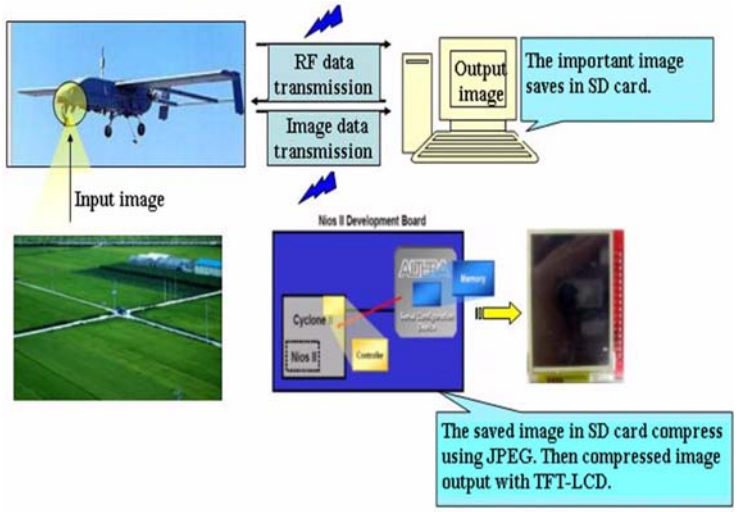
Figure 1. UAV Project Overview



Function Description

The aerial photography is saved onto a secure digital (SD) card using a personal computer. The Nios II processor running on the development board compresses the saved image. Then, the compressed image is output onto a thin film transistor liquid crystal display (TFT-LCD). See Figure 2.

Figure 2. Design Detail



Performance Parameters

The most important performance aspect of the system is tracking the images of a specific object saved in the SD card and keeping the image data. The saved image in the SD card is output onto a TFT-LCD via the Nios II development board. See Figures 3 and 4.

Figure 3. Object Tracking

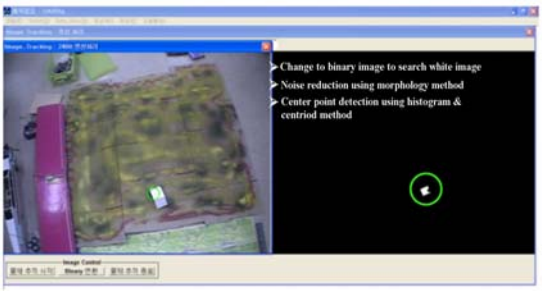
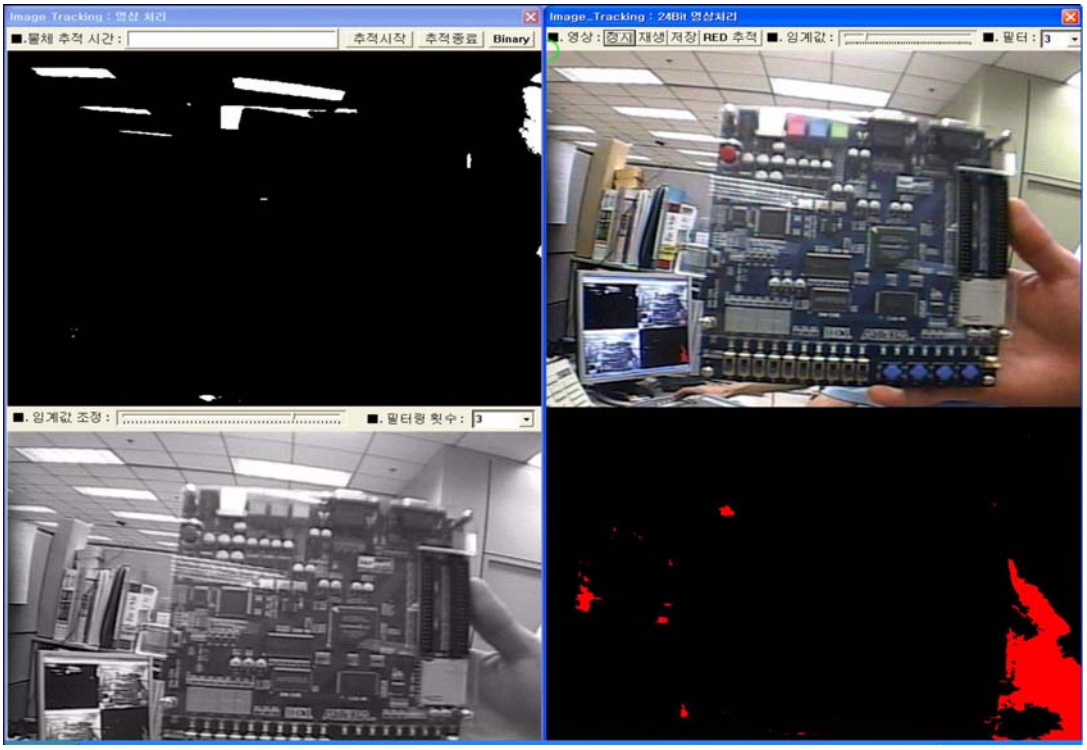


Figure 4. User Interface and External Pilot Image Processing



Design Architecture

This section describes the hardware design block diagram, software flow chart, and JPEG algorithm.

System Structure

Figure 5 shows the aircraft's internal system.

Figure 5. Aircraft Structure

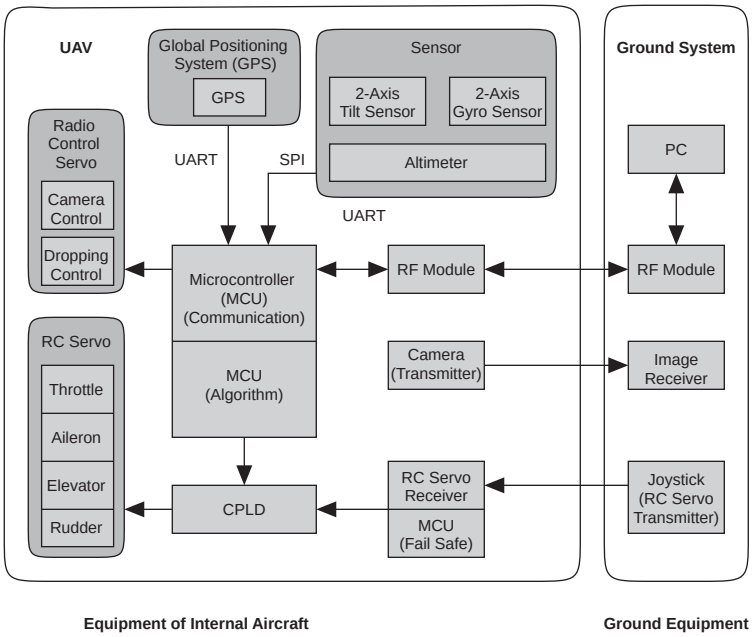
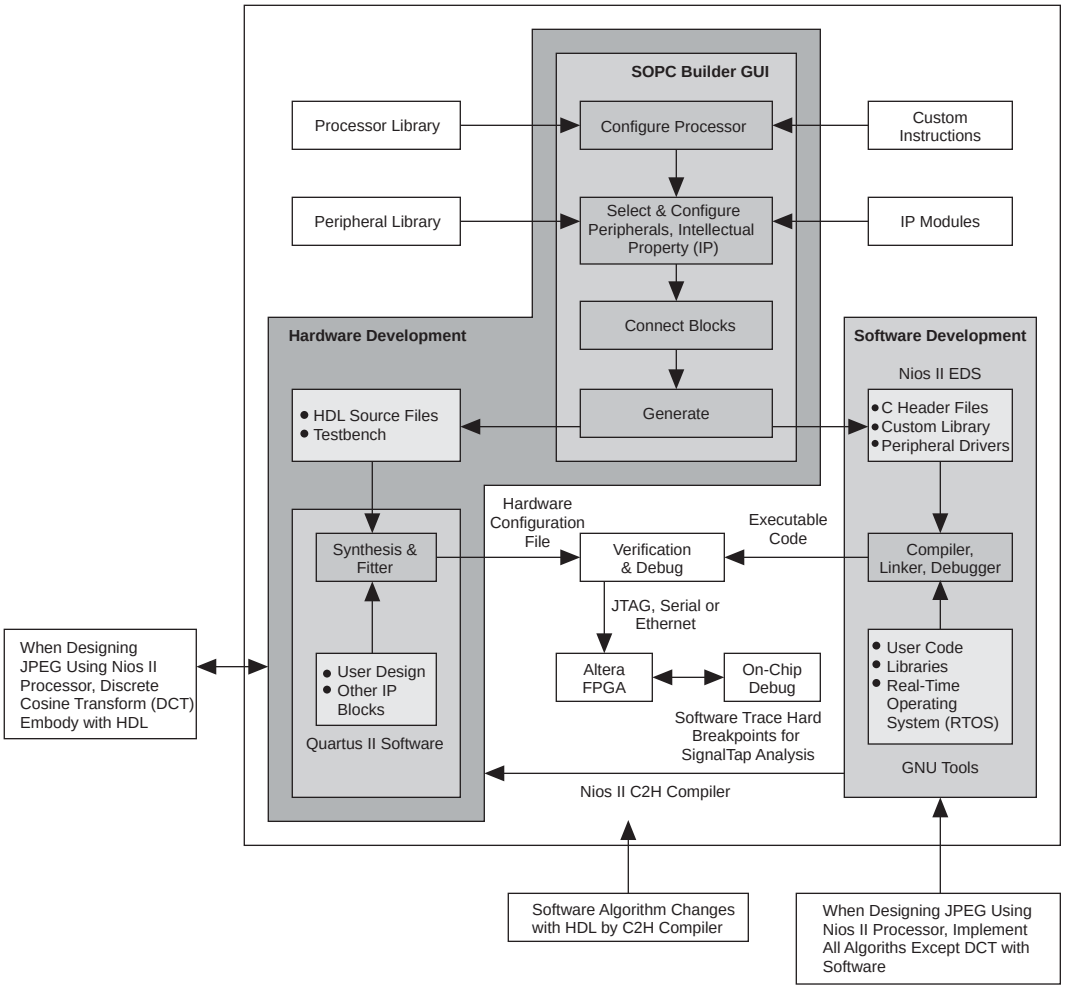


Figure 6 shows the Nios II system.

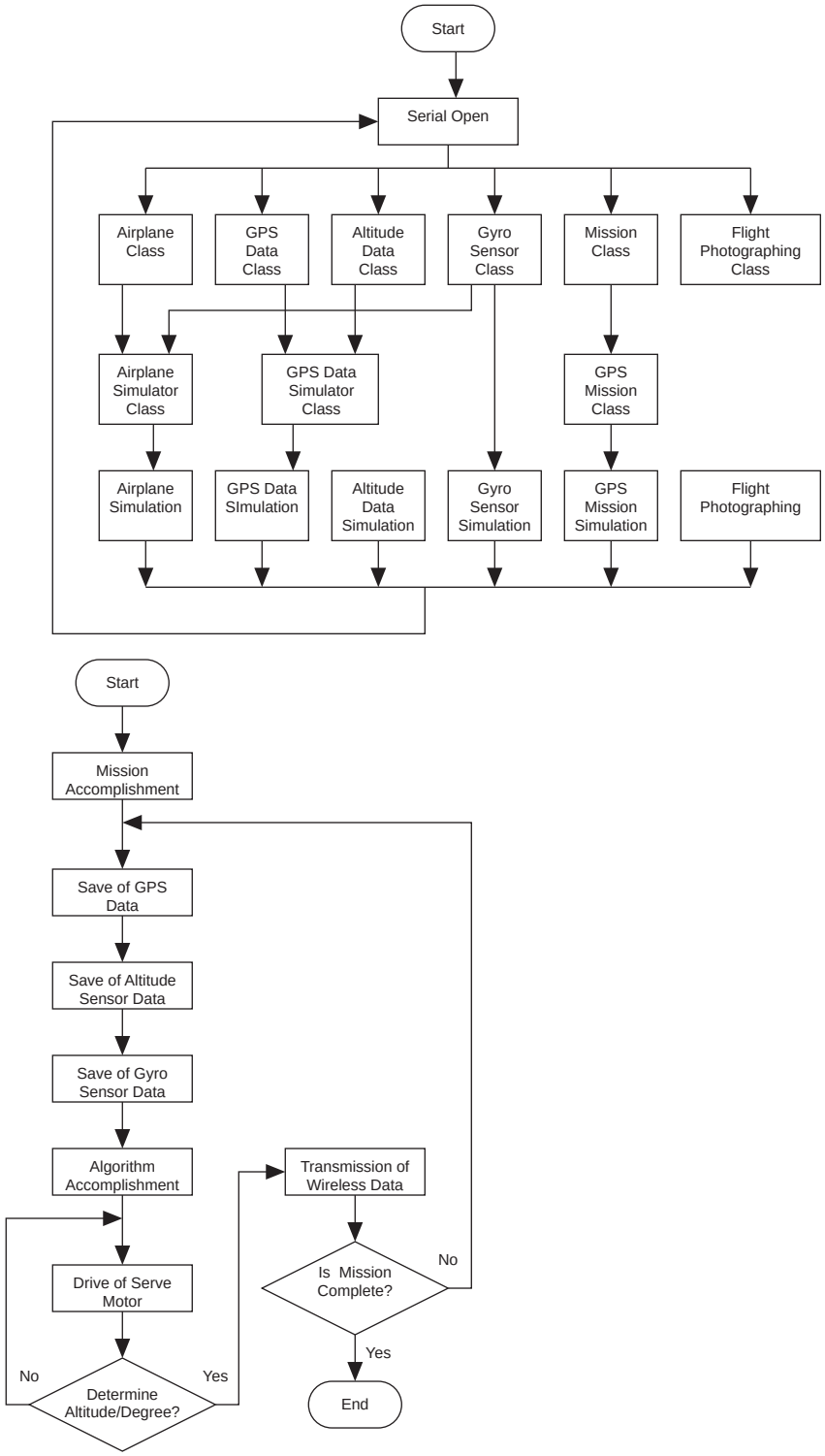
Figure 6. Nios II System Flow Chart



UAV Software Flow Chart

Figure 7 shows the aircraft's internal and external programs.

Figure 7. UAV Software Flow Chart



JPEG Software Flow Chart and Algorithm

JPEG compression has a basic method and an expanded method. The basic method uses 8 bits for each color in a pixel and consists of a sequential mode and Huffman encoding. The expanded method supports larger applications with 8 or 12 bits for each color in a pixel, sequential mode or progressive mode, Huffman encoding, and arithmetic code. The user can select which mode to use according to the application.

Figure 8 shows the JPEG encoding process. First, the converted input image is divided into 8 x 8 pixel blocks. A discrete cosine transform (DCT) operation is executed on each block to obtain the DCT coefficient. The DCT coefficient consists of direct current (DC) and alternating current (AC) components. Each component is independently quantized and the quantization table is individually created. The DC part of the DCT coefficient is encoded by the difference between the DC coefficient of the current block and the DC coefficient of the previous block. The AC components form an array by zig-zag scanning every block, after which the AC component is encoded. Figure 9 shows the DCT algorithm.

Figure 8. JPEG Encoding Flow Chart

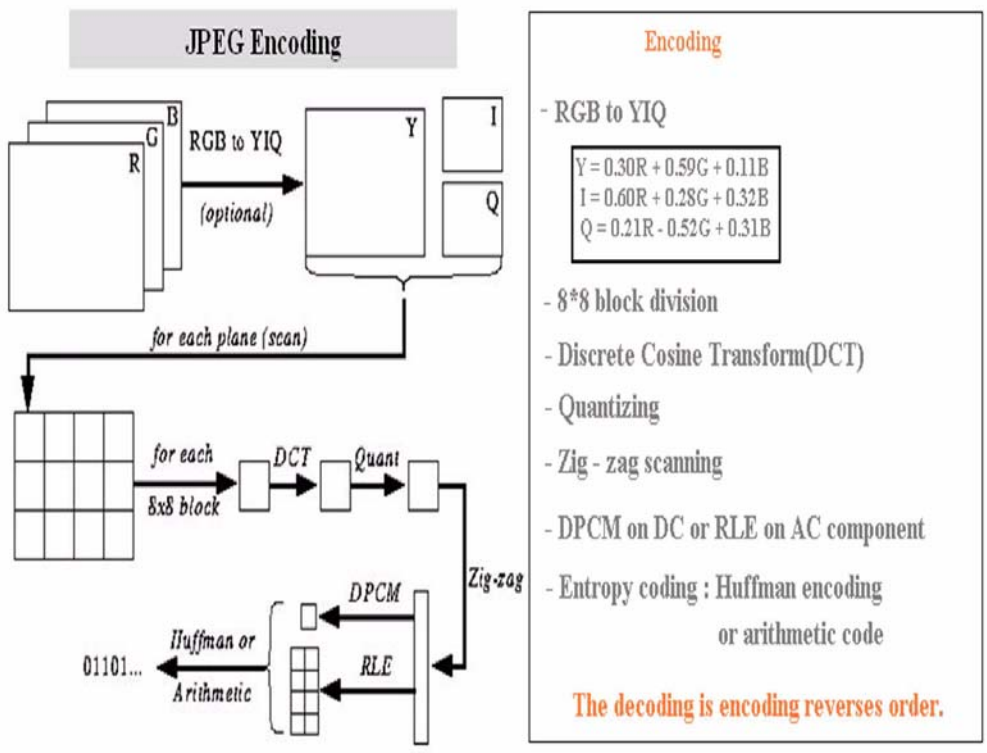
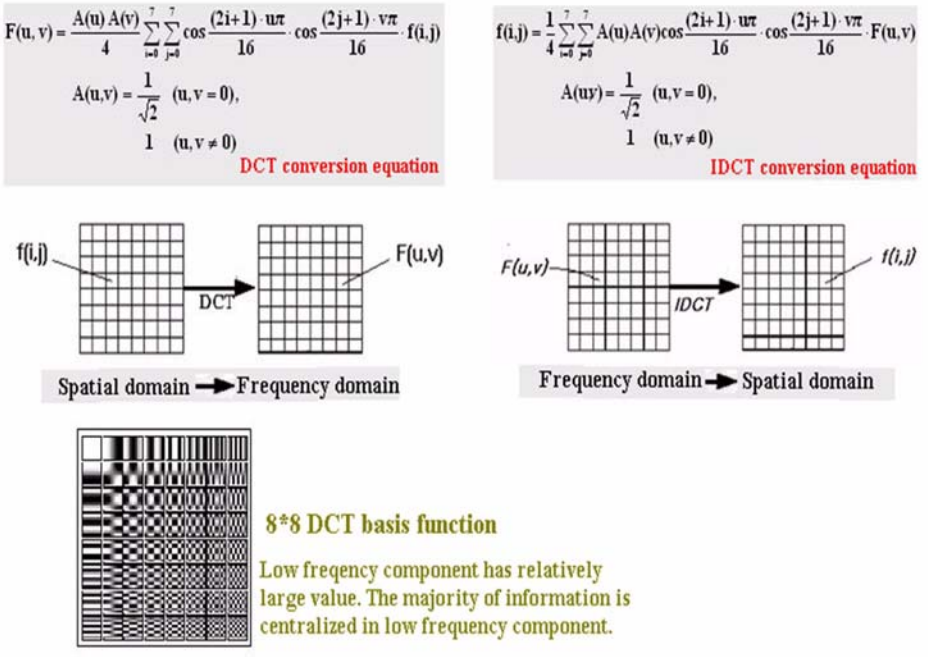


Figure 9. DCT Algorithm



Design Description

This section provides a detailed description of our design.

UAV Hardware Development Environment

Figure 10 shows a photograph of the aircraft and Table 1 shows additional details.

Figure 10. UAV Photograph

Table 1. UAV Details

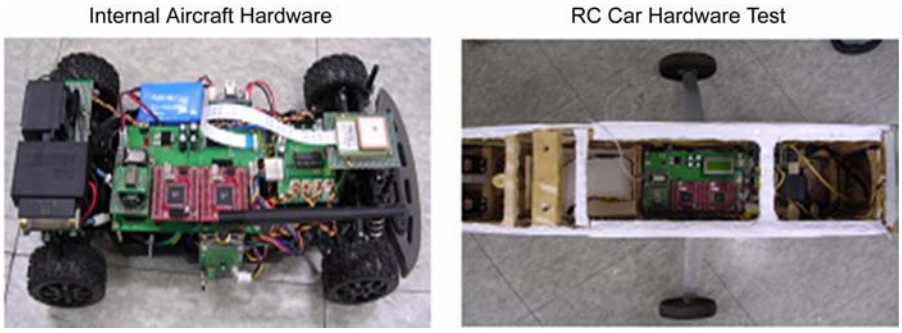
Plane Component	Description
Place Class	High Wing Plane
Body Material	Balsa and Plywood
Wing Span	2,040 mm
Fuselage	1,635 mm

Table 1. UAV Details

Plane Component	Description
Flying Weight	5,500 g

Figure 11 shows the internal hardware of the aircraft. We tested the hardware on a remote control (RC) car before using it on the aircraft.

Figure 11. Aircraft Internal Hardware



Nios II Development Environment

Figure 12 shows the system in SOPC Builder and Figure 13 shows the system generation result. Figure 14 shows the overall system schematic. We used the speed of the Nios II/f processor core to perform the compression quickly. Table 2 compares the features of the Nios II processor variants.

Figure 12. Configuration of SOPC Builder

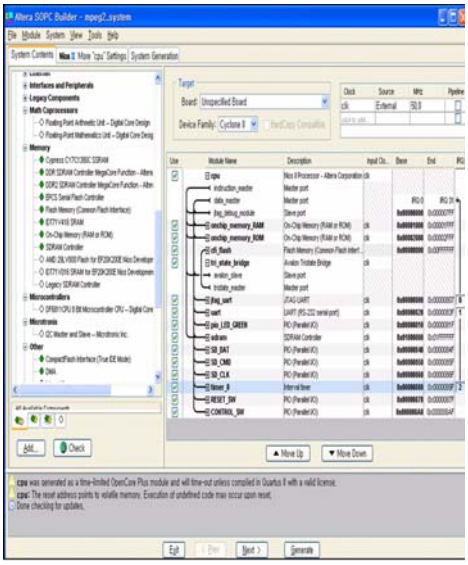


Figure 13. System Generation Result

Figure 14. Overall System Schematic

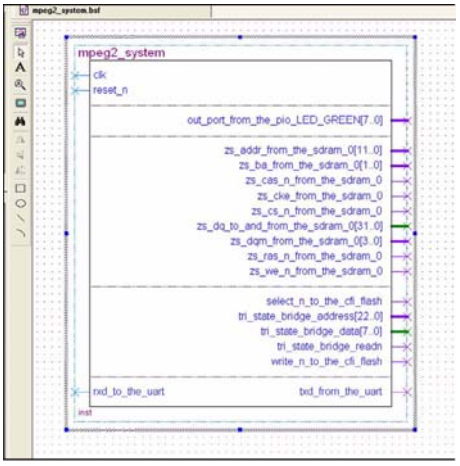


Table 2. Nios II Processor Features

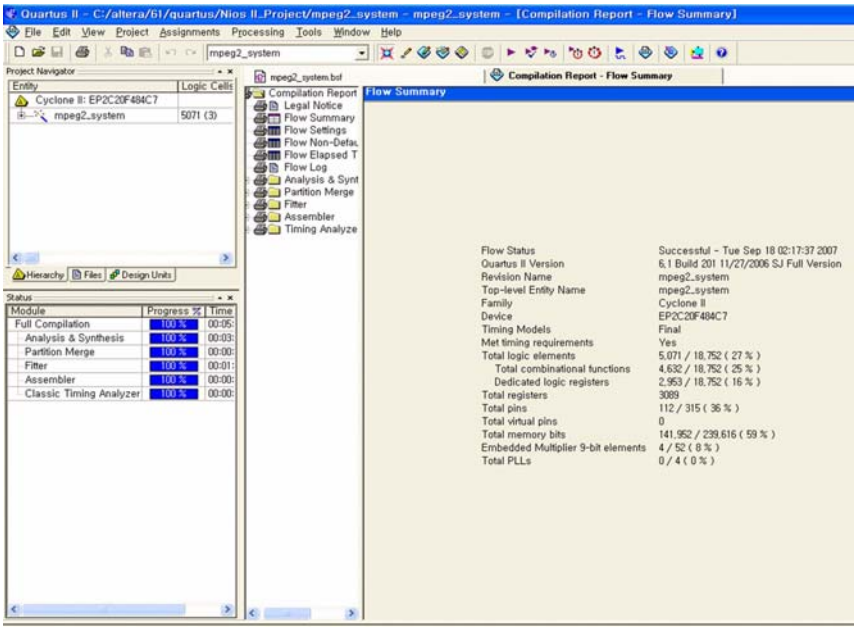
Feature		Core		
		Nios II /e	**Nios II/s**	**Nios II/f**
Objective		Minimal core size	Small core size	Fast execution speed
Performance	DMIPS/MHz	0.15	0.74	1.16
	Max. DMIPS	31	127	218
	Max. f_{MAX}	200 MHz	165 MHz	185 MHz
Area		< 700 logic elements (LEs) < 350 adaptive logic modules (ALMs)	< 1,400 LEs < 700 ALMs	< 1,800 LEs < 900 ALMs
Pipeline		1 stage	5 stages	6 stages

Table 2. Nios II Processor Features

Feature	Core		
	Nios II /e	Nios II/s	Nios II/f
External address space	2 Gbytes	2 Gbytes	2 Gbytes

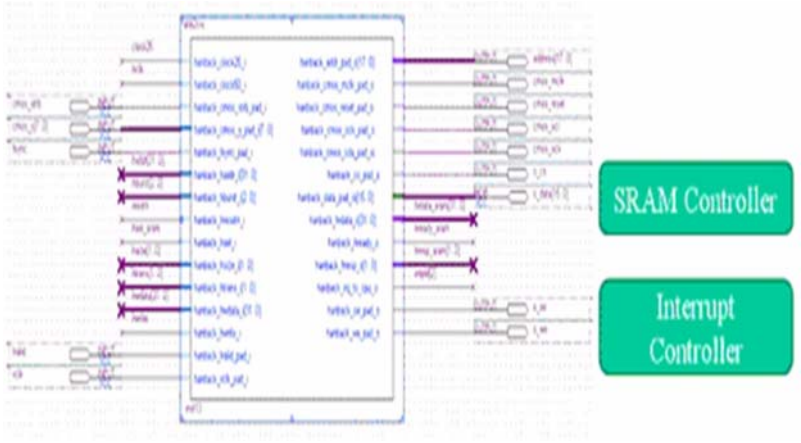
We compiled the generated system in the Quartus® II software. Figure 15 shows the compilation report. The system used 27% of the total available logic, 36% of the available pins, and 59% of the memory bits.

Figure 15. Quartus II Compilation Report



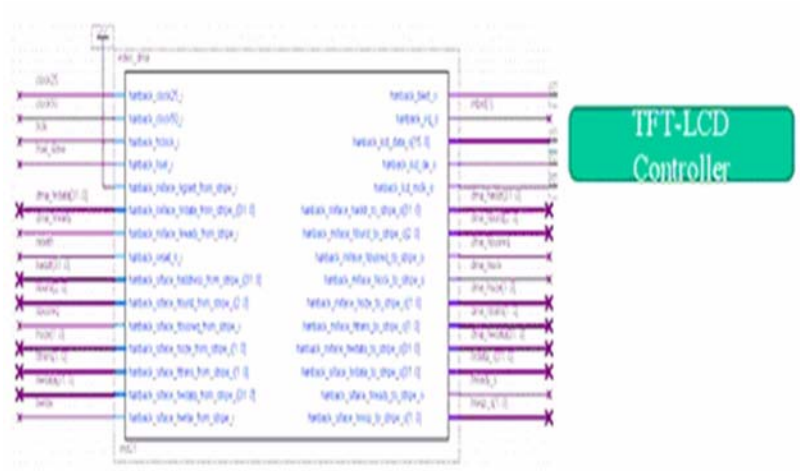
Figures 16 and 17 show the SRAM controller and TFT-LCD controller, respectively.

Figure 16. SRAM Controller



Input Ports	Output Ports
`fsync` = Synchronization of input image `vclk` = Data output clock `lvalid` = Line value	`ntpld[2]` = Interrupt generation `address[17..0]` = SRAM save address `S_data[15..0]` = SRAM save data `S_cs` = SRAM chip select `S_oe` = SRAM output enable `S_we` = SRAM write enable

Figure 17. TFT-LCD Controller

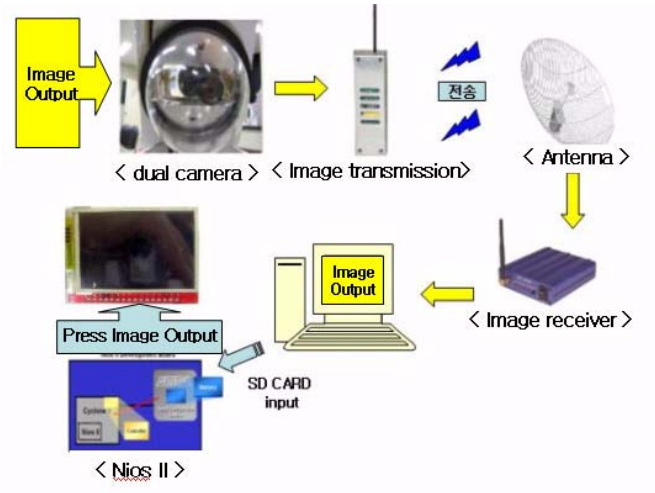


Output Ports
`bled_o` = Back light on/off `lcd_de` = LCD data enable `lcd_data[15..0]` = RGB[5:6:5] `lcd_mclk` = LCD clock

Image Processing Development Environment

Zoom cameras have many constraints when used in aircraft because of their size and weight. Using two camera lenses, we reduced this problem. Figure 18 shows the image processing using the dual camera lenses.

Figure 18. Image Display Processing



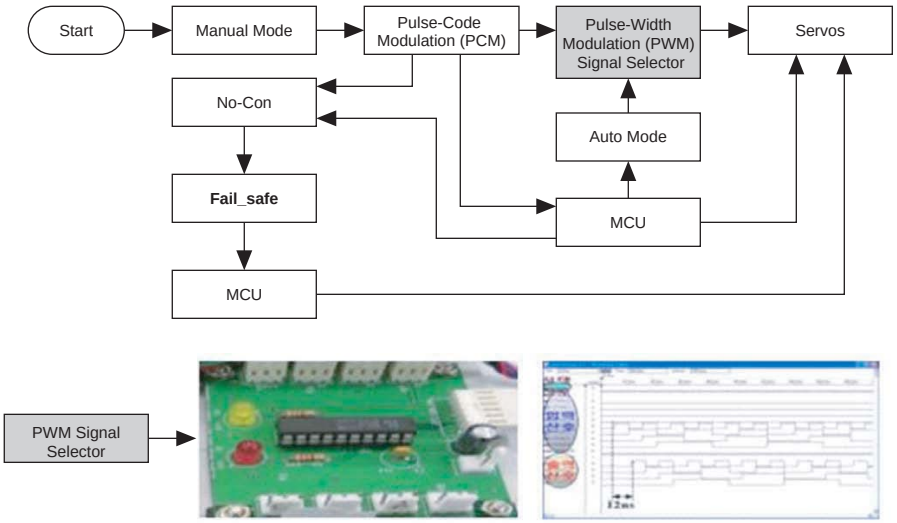
Design Features

This section describes the features of our design.

Flight Safety Test

To convert between automatic and manual piloting safely and rapidly, we built a switch using the GAL16V8 logic device. The device is simple to use. We implemented the conversion between automatic and manual piloting using channel five in the ratio controller. Two LEDs display whether the aircraft is on manual or automatic pilot, and can be seen easily. Figure 19 shows the setup of this switch using the logic device.

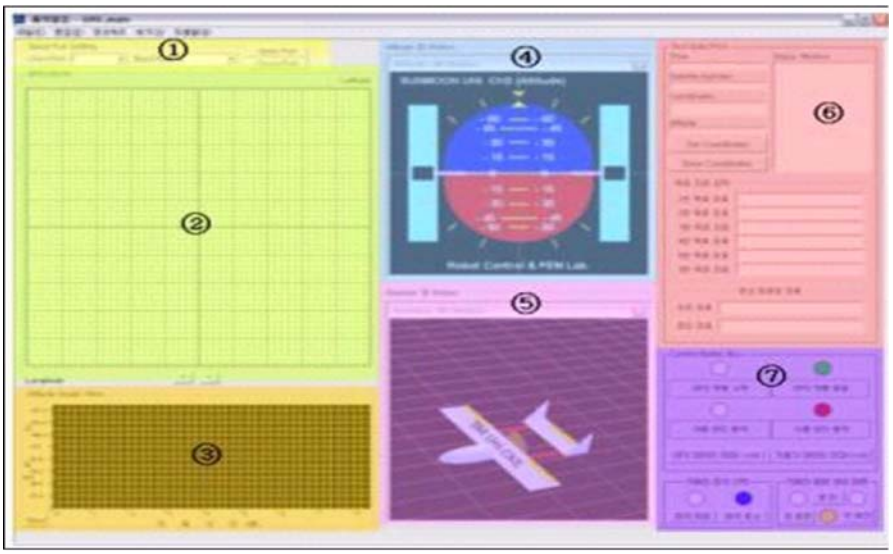
Figure 19. Automatic and Manual Pilot Switch Hardware and Flow



User Interface

A user interface provides communication with the aircraft (see Figure 20). See the following discussion for more information on the numbered areas in Figure 20.

Figure 20. Aircraft Communication User Interface



1. This part of the interface displays data from the hardware system. The application uses serial communication to connect to the hardware system and data is sent asynchronously.

2. This area represents the aircraft movement with a line. The user can judge any objective errors by enlarging or reducing the scale. Additionally, the user can determine whether the mission was executed properly by marking the position of the objective location.

3. This area displays the altitude using a line. The aircraft's altitude is shown in real time.

4. The 3-dimensional (3-D) motion area determines the aircraft's altitude and inclination using a 3-D program.

5. An OpenGL program shows the aircraft's inclination and direction.

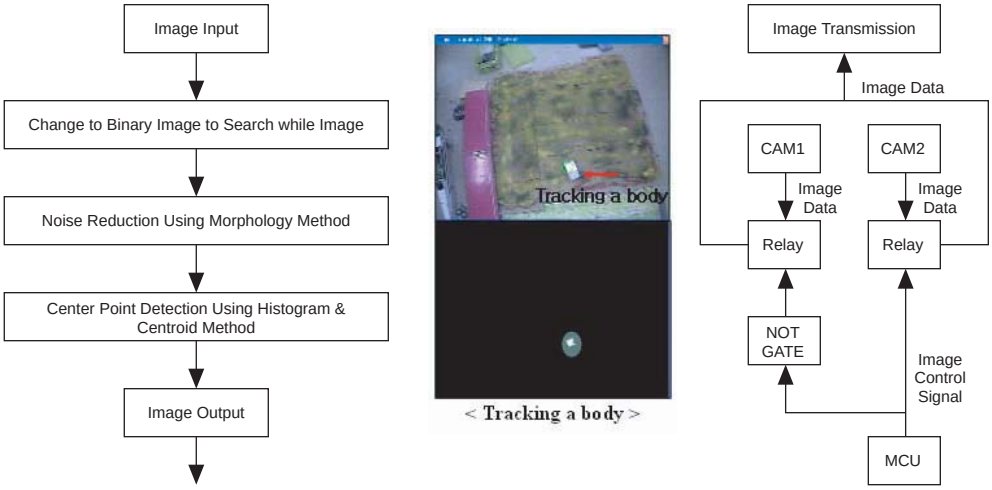
6. This area shows the GPS data received from the aircraft. The application also transmits the objective location to the aircraft. Additionally, it shows the tracking information obtained by the aircraft.

7. This area contains the aircraft control buttons for functions such as turning the GPS module on and off, the camera's position control, the field of view, etc.

Image Processing Algorithm and Camera Interface Hardware

Figure 21 shows the image processing flow as well as the camera's hardware interface.

Figure 21. Image Processing Algorithm and Camera Hardware Interface

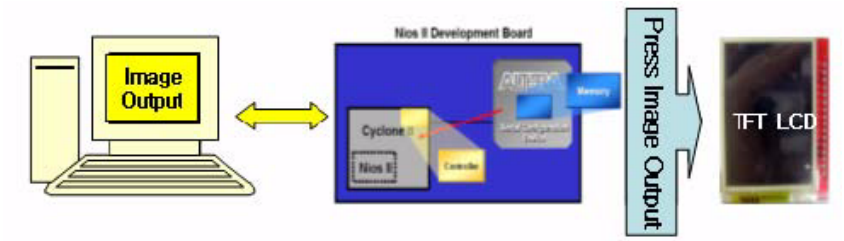


Nios II Processor Role

The Nios II processor compresses the aerial photography using JPEG compression. The camera collects a lot of image information to ensure that the mission has executed correctly. The image compression system codes the aerial photography and then decodes the collected images. Figure 22 shows how the Nios II processor fits into the system.

The system collects real-time image data and saves it to an SD card. The saved images are JTAG compressed by the Nios II processor running on the development board. Finally, the compressed image is output onto a TFT-LCD monitor.

Figure 22. Nios II Processor Role



Conclusion

We learned a variety of things while working on this project, such as:

- We used the JTAG module to perform many experiments with the Nios II processor and the development board.
- We could download the program to memory, and easily start and stop the program execution. The breakpoint and watchpoint features made it easy to debug any problems. It was useful to reference data by analyzing registers and memory.
- The Nios II Integrated Development Environment (IDE) example code and project templates made it easier for us to build the system.
- We used the µC/OS-II (real-time kernel) that was provided with the development kit.
- When we started the project, the on-line demonstrations, such as “Creating a Nios II System” on the Altera web site, were helpful.

The development board has built-in flash memory, but it is very small. We needed to change the scope of our project from using MPEG compression to using JPEG compression, which is relatively small in size. In the future, we will plan to use external memory, which will allow us to perform aerial photography using video compression.

Second Prize

Laser Direct Writing Digital Servo Controller Based on SOPC Technology

Institution: **Ultra-Precision Photoelectric Instrument Engineering Research Institute, Harbin Institute of Technology**

Participants: **Lei Yan, Tao Cheng, Ya Gao**

Instructor: **Wang Lei**

Design Introduction

The application of precision/ultra-precision processing technology has expanded from a few areas (such as national defense and aerospace) to different aspects of the national economy. As the semiconductor industry develops, it imposes increasing requirements on laser direct writing lithography, which is the core of very-large-scale integration (VLSI) manufacturing.

Laser direct writing lithography applies a flexible dose exposure on a substrate surface with a variable intensity laser beam. A key step of this process is high-precision scanning with a computer-controlled laser beam. During lithography, the substrate on the stage moves with the platform and is exposed with a flexible dose by controlling the laser beam intensity with an acousto-optic modulator. The stage's positioning accuracy and motion stability directly affect the performance of the lithography machine and the quality of the lithographic components. Therefore, a rapid, high-precision straight-line feed system is needed.

The voice coil motor (VCM) has high displacement resolution, zero-length feed drive chain (i.e., direct drive or zero drive), strong dynamic sensitivity, and good responsiveness. Therefore, the VCM has replaced the traditional positioning structure of the rotating servo motor plus ball screw shaft, becoming the motion servo core for micron or even submicron location of the laser direct writing stage.

The digital servo-based motion controller is the key to an ultra-precision positioning system. Digital servo means that digital technology is used for closed-loop control and system regulation. Additionally, the control and regulation are based on software, so that the pulse width modulator (PWM) control signal is exported directly. Alternatively, a digital-to-analog converter (DAC) chip generates the DC

drive current and the VCM is driven after power amplification of the motor driver. Software-based regulators allow a variety of controls, such as vector control, parametric adaptive control, sliding mode variable structure control, fuzzy control, neuron network control, etc. Software eases parameter self-optimization and fault self-diagnosis functions, and improves system control performance. It also overcomes the shortcomings of simulated closed-loop servo systems, including difficulty with weak signals, noise separation for weak signals, difficulty in improving control precision to above 0.1%, vulnerability to temperature, zero drift error of the location control, etc.

To curb noise interference (including gas film disturbances in movement of the laser direct writing stage) and to implement fast submicron location, this paper describes a VCM digital servo motion controller based on Altera's Nios® II embedded soft-core processor. The design uses the low-cost, high-performance Cyclone® II FPGA family and an integrated dual Nios II soft-core application system for various functions, including fast sampling and high-resolution decoding of signals from a displacement sensor (laser interferometer), motion state monitoring of a controlled object (static pressure air-bearing slider and VCM), motion characteristics spectrum analysis and digital filtering, PWM control signal or DAC DC voltage output based on the integration split packet identifier (PID) control algorithm, etc. With highly integrated system-on-a-programmable-chip (SOPC) technology, the design features flexible functions, reduced electromagnetic interference, improved processing speed and control reliability, reduced development costs, and easier system upgrades and maintenance.

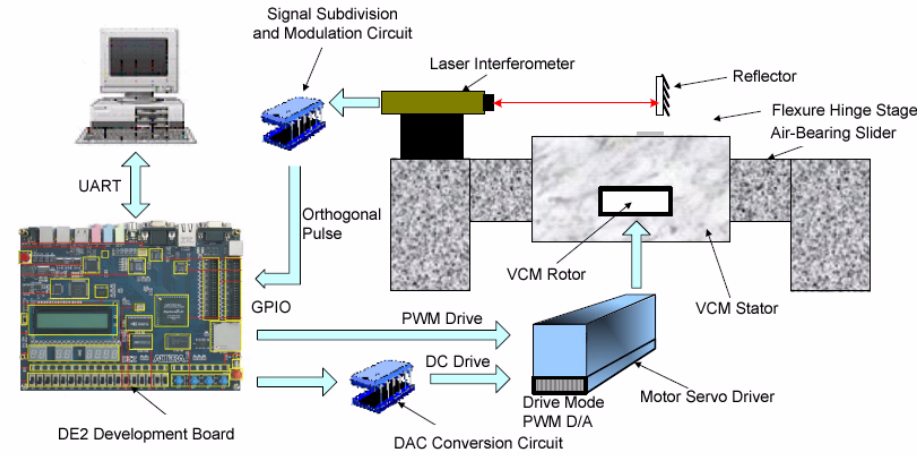
Function Description

In our laser direct writing motion control system, the dual Nios II-based digital servo motion control reduces noise caused by load disturbances of the motor rotation, gas film disturbances of the air-bearing slider, and mechanical resonance of the drive system. Our closed-loop servo system has four parts:

- *Controlled object*—The controlled object is the stage operating on a static-pressure, air-bearing slider, and is driven by the VCM.
- *Displacement sensor (feedback)*—The sensor is Renishaw's laser interferometer, which, based on principles of interferometric measurement, transforms the stage's displacement information into a corresponding orthogonal pulse signal and implements a logic-level conversion of the pulse signal through a signal modulation circuit.
- *Control unit*—The control unit is the Altera® Development and Education (DE2) development board that integrates dual Nios II embedded soft-core processors. It performs several tasks, including signal acquisition, digital filtering, a PID control algorithm, a DAC chip interface, a PWM control quantity output, and user interface based on the µC/OS-II real-time operating system (RTOS).
- *Actuator*—The actuator is a motor servo driver with two operating modes: DC voltage driving and PWM driving. The system can export the control algorithm result through a DAC interface module or custom PWM peripheral module, which controls the motor servo driver in different ways.

Figure 1 shows the structure of the digital servo system.

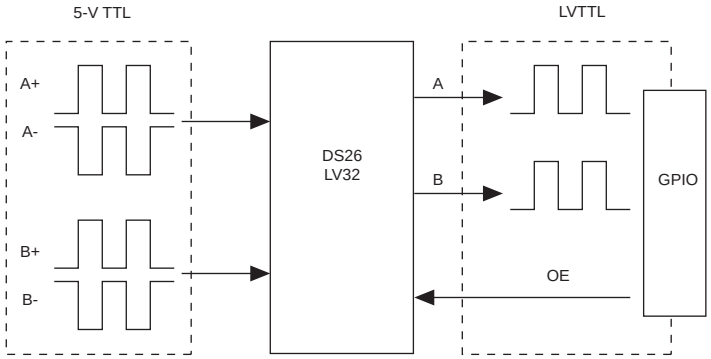
Figure 1. Digital Servo System Structure



Signal Modulation Circuit

The laser interferometer exports four types of differential signals, including A+, A-, B+, and B-, with an orthogonal phase relationship and a 5-V TTL logic level. We used the 10-MHz DS26LV32 differential receiver to convert the differential signals into single-ended signals and provide level conversion, acquiring LVTTL orthogonal pulses A and B for general-purpose I/O (GPIO) sampling of the control unit. It can also control the chip enable to prevent sampling errors. Figure 2 shows a schematic diagram of the signal modulation circuit.

Figure 2. Signal Modulation Circuit



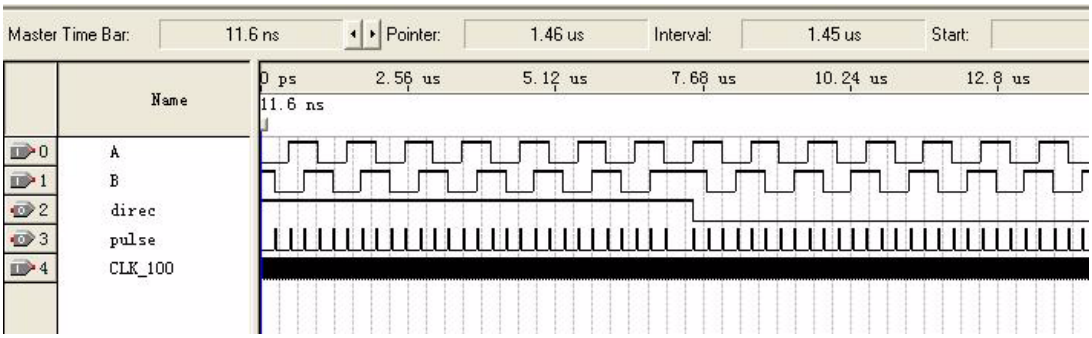
Pulse Signal Subdivision Decoding Unit

The pulse signal subdivision decoding unit consists of the subdivision sense module and the M/T decoding module. The subdivision sense module generates a standard-width spike pulse at the rising and falling edge of the orthogonal pulses A and B, performs logical OR operations, obtains a quadruple subdivision pulse (`pulse`), obtains a system resolution of 80 nm/pulse by calculating the laser interferometer resolution indexes, and calculates the motor's motion direction (`direc`) based on the phase relationship (`advance` or `lag`). The standard frequency signal `CLK_100` provides a latency base value to prevent post-processing counting errors due to narrow spike pulses. Figure 3 shows a schematic diagram of the top-level module.

Figure 3. Pulse Subdivision Sense Top-Level Module

Figure 4 shows the simulation result of the subdivision sense module functions. `CLK_100` is a 100-MHz standard frequency signal.

Figure 4. Pulse Subdivision Sense Module Simulation Result



The M/T decoding module processes quadruple-subdivision signals while acquiring the motor movement's relative position and speed, placing the motor under the control of a position loop or speed loop plus dual-position loop. The system can acquire the relative motor position (`position`) by adding and subtracting `pulse` by `direc`. The system synchronously presets gate signal `T1` by the tested signal to acquire the actual gate signal `T2`. It counts the standard frequency (`CLK_50`) and `pulse` simultaneously in `T2` to acquire count values N_C and N_P. The quadruple subdivision pulse frequency $f_P = N_P f_C / N_C$, and the motor speed can be acquired according to the motor displacement corresponding to each pulse of f_P. We used a synchronous gate to ensure integral counting multiples and implement an equal precision measurement within the measuring scope of the signals. Simultaneously counting the tested frequency and standard frequency allows the system to switch the cycle and frequency measurements automatically according to the motor's speed change and ensures a wide band for speed measurement. Figure 5 shows the schematic diagram of the top-level module.

Figure 5. M/T Decoding Top-Level Module

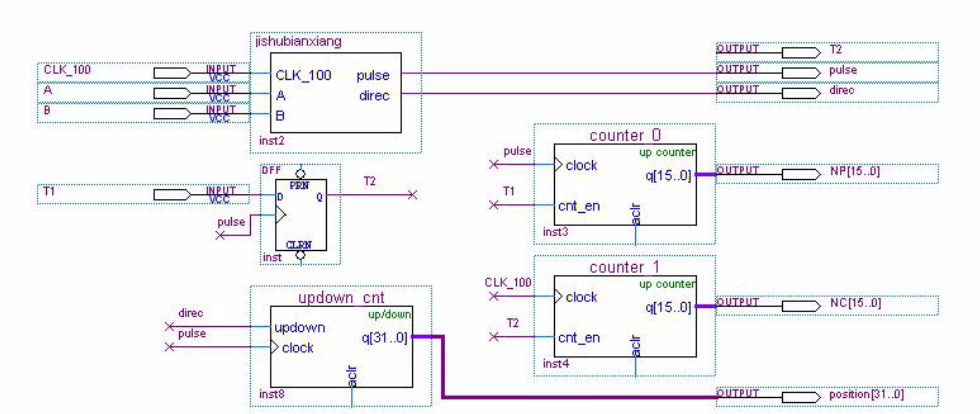
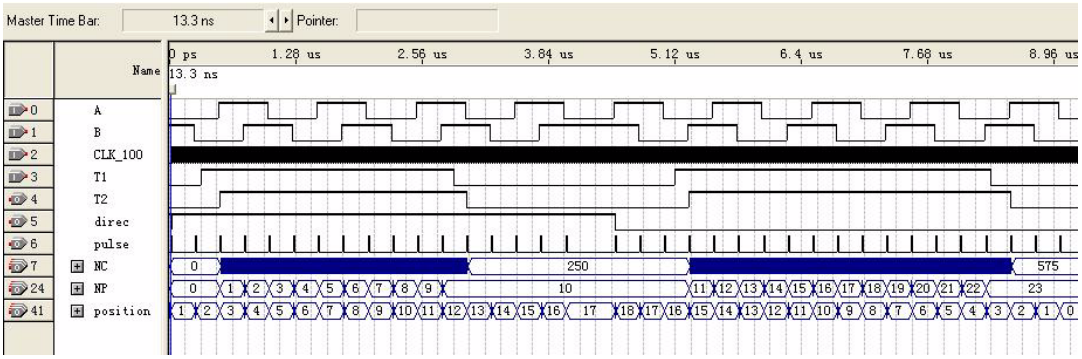


Figure 6 shows the M/T decoding module simulation result. `CLK_100` is the 100-MHz standard frequency signal, `T1` is the preset gate, `T2` is the synchronous gate, and `NC` and `NP` are the standard frequency and count result of quadruple subdivision frequency, respectively.

Figure 6. M/T Decoding Module Simulation Result



Data Processing Unit

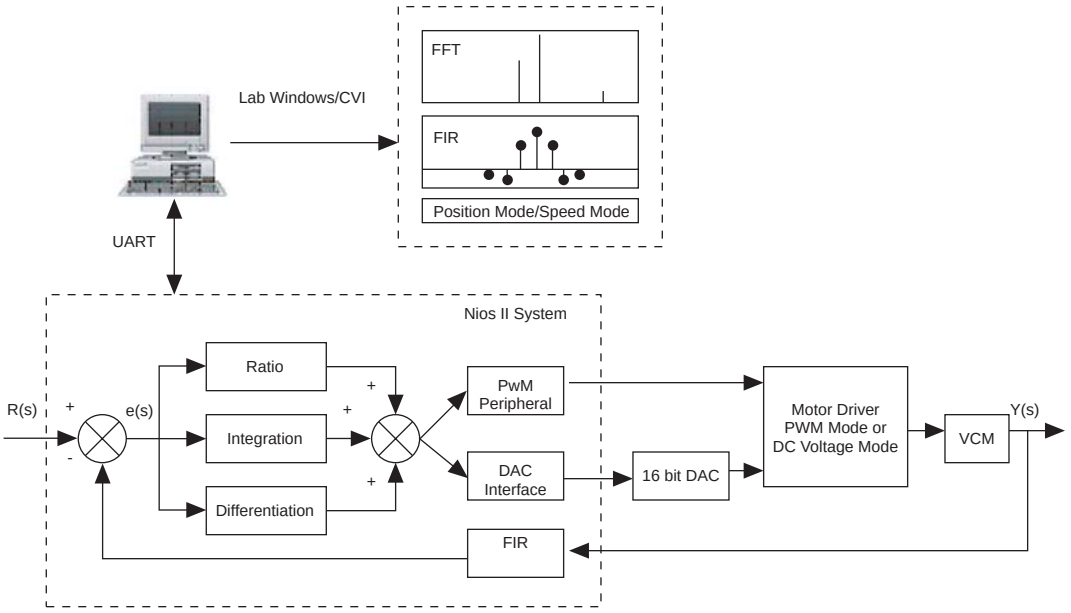
For the data processing unit, the core of the closed-loop control system consists of a Nios II hardware platform (including the Nios II embedded soft-core processor, memory, on-chip peripherals, custom peripherals, etc), as well as the supporting LabWinsows/CVI software on the PC side:

- *Nios II hardware platform*—We used a dual-core Nios II processor and updated parameters such as control quantity, PID, and finite impulse response (FIR) coefficients in real time via communications between the serial port and the high-level computer software. The system performs an efficient real-time FIR filter operation using a custom multiply-add instruction, collects sensor signals to integrate a split PID operation, and drives a custom PWM or DAC module to control the VCM servo driver operation. Figure 7 shows the data processing unit structure.

- *LabWindows/CVI platform*—This platform sets the PID control parameter to control the location and size. It uploads the follow-up signal position collected by the Nios II hardware platform through the serial port, performs a fast Fourier transform (FFT) operation, and draws a delay/frequency response curve. It sets the digital FIR filter parameters (such as filter type, order, quantification coefficient, and window function) according to the FFT analysis result.

Additionally, it sets excitation signals (e.g., sine, square wave, and step) and observes the follow-up curve position after filtering.

Figure 7. Data Processing Unit Structure



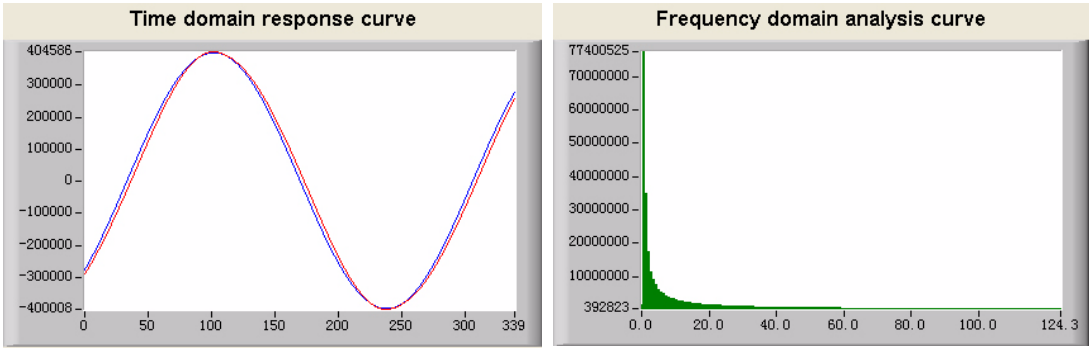
FFT

The discrete Fourier transform (DFT) of N* sample points is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \quad W_N = e^{-j\frac{2\pi}{N}}, \quad k = 0,1,\cdots,N-1$$

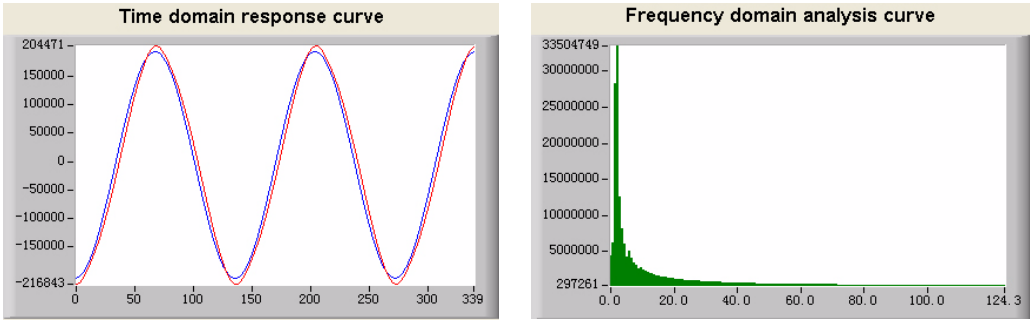
FFT, as an efficient implementation of DFT, delivers high operation efficiency and is suitable for real-time digital signal processing (DSP). Because the noise signal frequencies caused by gas film disturbances, mechanical resonance, etc. are relatively stable, we used post-processing and a CVI program for FFT operation on the Nios II-collected feedback signals. While ensuring the spectrum analysis result reliability, this method prevents real-time FFT operation with a Nios II soft-core processor and greatly reduces the system's operation load. We tested the system frequency response using sine wave excitation with a pulse amplitude of 400,000, a frequency of 1 Hz, a sampling frequency of fs = 250 Hz, and N = 340 sampling points. Figure 8 shows the position-time domain response curve and the acquired frequency analysis curve.

Figure 8. Time and Frequency Domain Response Curves with 1-Hz Sine Excitation



We tested the system's frequency response using sine excitation with a 200,000 pulse amplitude and 2-Hz frequency. The sampling frequency and the number of sampling points remain unchanged. See Figure 9.

Figure 9. Time and Frequency Domain Response Curves with 2-Hz Sine Excitation



According to the frequency response curve, noise interference exists at frequencies of 5 to 60 Hz and is especially significant around 5 Hz, which is quite similar to the gas film disturbance frequency in theoretical analysis. Other noise may come from mechanical resonance, electromagnetic interference, etc.

Digital FIR Filter

FIR filters are widely used in digital system design and can acquire a strict linear phase while ensuring amplitude characteristics and satisfying technical requirements. The system function H(z) of an N-order FIR filter is:

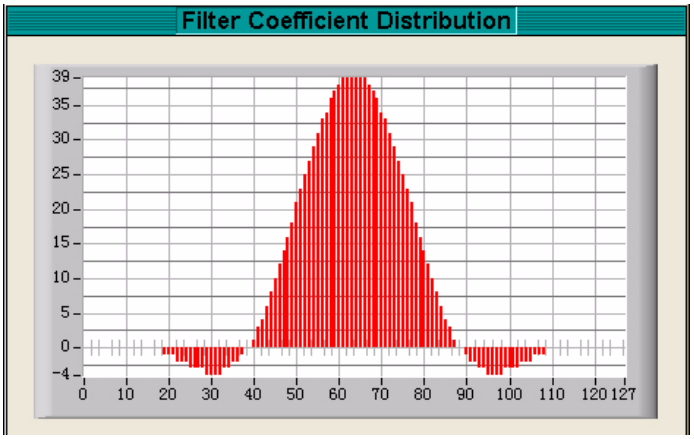
$$H(z) = \sum_{n=0}^{N-1} h(n) z^{-n}$$

In a digital system, the output and input functions $y[n]$ and $x[n]$ are:

$$y[n] = \sum_{k=0}^{N} a(k) x[n-k]$$

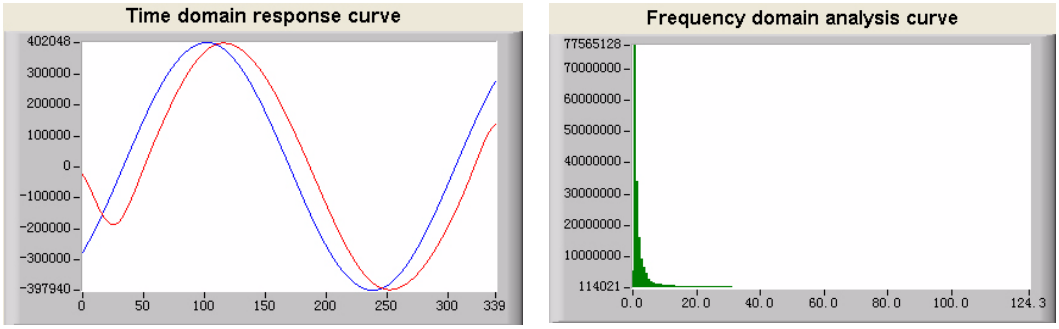
In LabWindows/CVI, we set a 128-order, low-pass FIR filter with a 2-Hz cutoff frequency, a 100-Hz sampling frequency, a Hanning window, and a 1,000 quantification coefficient. Figure 10 shows the coefficient distribution curve.

Figure 10. 2-Hz Low-Pass FIR Filter Coefficient Distribution Curve



We downloaded the filter coefficient torque to the Nios II hardware platform through the serial port. Input $x[n]$ and coefficient $a(k)$ are multiplied and added by a custom multiply-add instruction, implementing real-time digital filtering of the input signals. Figure 11 shows the time and frequency response curves for sine excitation with a 400,000 pulse amplitude and 1-Hz frequency.

Figure 11. Time and Frequency Domain Response Curves after Filtering for 1-Hz Sine Excitation



In LabWindows/CVI, we set a 100-order, low-pass FIR filter with a 5-Hz cutoff frequency, a 100-Hz sampling frequency, a Blackman window, and a quantification coefficient of 1,000. Figure 12 shows the coefficient distribution curve.

Figure 12. 5-Hz Low-Pass FIR Filter Coefficient Distribution Curve

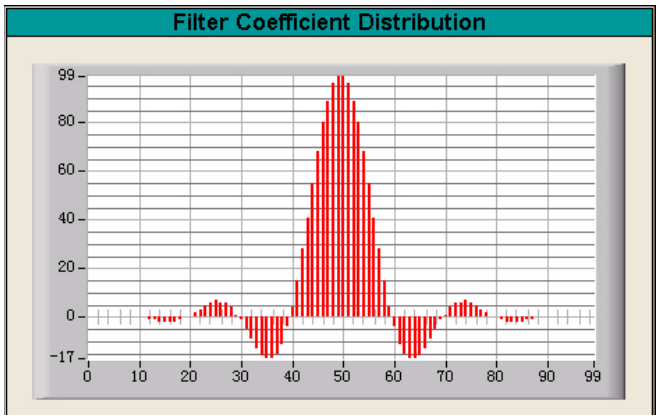
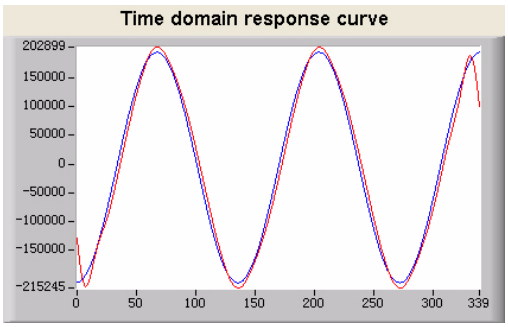
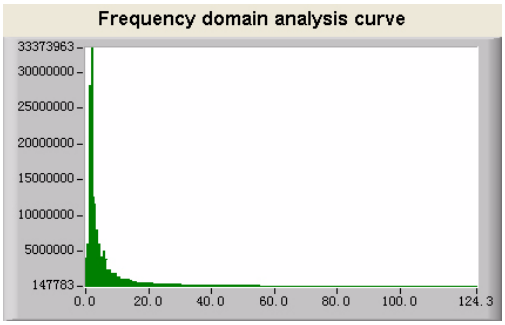


Figure 13 shows the time and frequency response curves for sine excitation with a 200,000 pulse amplitude and 2-Hz frequency.

Figure 13. Time and Frequency Domain Response Curves after Filtering for 2-Hz Sine Excitation





According to our analysis, noise in the feedback signals is curbed after we added the digital FIR filter. The position response curve becomes smooth but a linear phase shift (directly related to the filter order) occurs. In this project, we use a custom instruction for the filtering algorithm. The operation time for a single order is controlled to within 10 clock cycles (or 100 ns) and the total latency for a 128-order operation is about 13 μs, which is negligible compared to the 200-μs servo cycle. Therefore, the steady state error influence is within the specified scope. We can reduce the latency by increasing the system clock frequency or reducing filter orders. Additionally, we can reduce the phase delay error significantly by introducing phase compensation, further improving precision control.

Integrated Split PID Control Algorithm

In ordinary PID control, the proportional element reflects the control system's deviation signals proportionally and the controller performs control functions immediately when the deviation occurs to reduce the deviation. The integrating element mainly eliminates static error and improves the system's astatism. The derivative element reflects the variation rate of the deviation signals and introduces a valid early amendment signal to accelerate system operation and shorten regulation time. The main purpose of the integrating element is to eliminate static error and improve control precision. However, at the beginning, end, or substantial change of a setting, there is a large system output deviation for a short time, leading to an integrated PID operation, control quantity exceeding the control quantity limits corresponding to the maximum actuating range allowed by the actuator, significant system overshoot, or even oscillation.

To prevent this situation, we used an integrated split PID algorithm in this design. The integration function is cancelled when there is a large deviation between the controlled quantity and the set value to prevent system instability and increased overshoot caused by integration functions. When the controlled quantity is about the same as the set value, the system introduces an integration function to eliminate static error and improve precision control. See the following steps:

1. Set a threshold $\varepsilon > 0$ according to the actual situation.

2. When $|error(k)| > \varepsilon$, PD control prevents excessive overshoot and ensures quick system response.

3. When $|error(k)| \le \varepsilon$, PID control ensures the system's control precision.

The integration split control algorithm can be expressed as:

$$u(k) = K_P E(k) + \beta K_I \sum_{j=0}^{k} E(j)T + K_D[E(k) - E(k-1)]/T$$

Of which, K_P is the proportionality coefficient, K_I is the integration coefficient, K_D is the differential coefficient, T is the sampling time, and β is the integration switching coefficient. See the following equation:

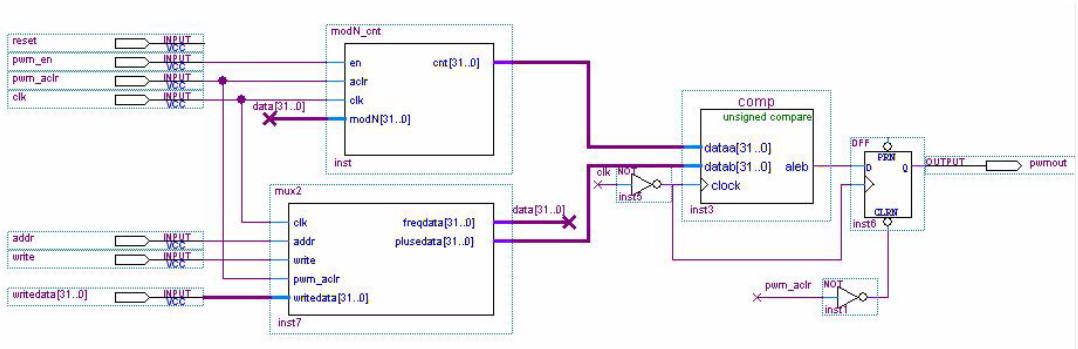
$$\beta = \begin{cases} 1 & |error(k)| \le \varepsilon \\ 0 & |error(k)| > \varepsilon \end{cases}$$

Custom Subordinate PWM Peripheral

To adapt to the motor servo driver's PWM operating mode and to convert PID operation results into control quantity signals, this design uses a customized Avalon® switch fabric-based PWM subordinate peripheral with an adjustable output frequency and duty ratio set according to the PWM waveform generation principle. The subordinate peripheral provides linear conversion from the operation result to the drive voltage, and the theoretical quantizer ratio is $1/2^{32}$.

The PWM module consists of a bus interface controller, a 2-bit frequency register, a 32-bit duty ratio register, and a comparator. Figure 14 shows the top-level module.

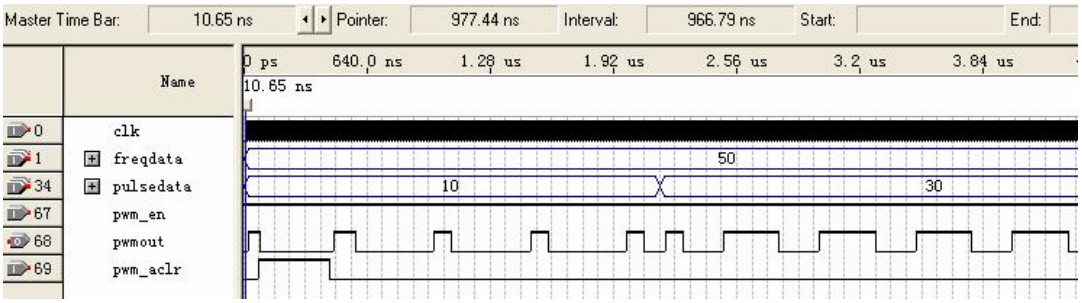
Figure 14. PWM Top-Level Module



Ignoring the Avalon bus control signals, the PWM module simulation result is shown in Figure 15. `clk` is the 100-MHz reference clock, the square wave output frequency (`freqdata`) is 50, the set duty ratio

(pulsedata) goes from 10 to 30, pwm_en is the module enable signal, pwm_aclr is the asynchronous clear signal, and pwm_out is the output. The simulation result shows that the module adjusts the duty ratio from 10/(50 + 1) to 30/(50 + 1).

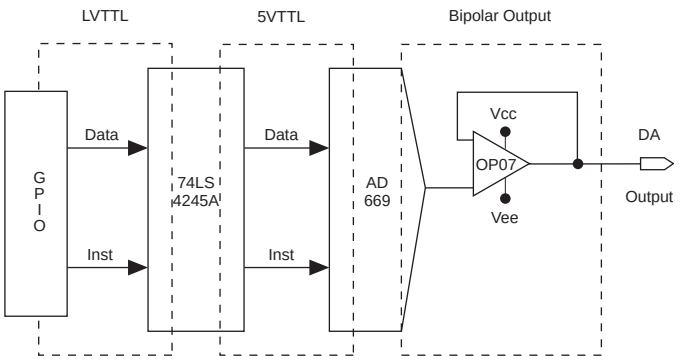
Figure 15. PWM Module Simulation Result

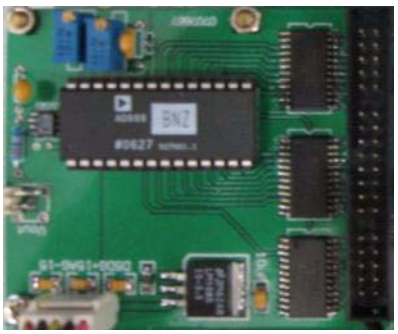


DAC Drive Unit

To adapt to the motor servo driver's DC voltage operating mode, we designed a AD669-based DAC drive circuit for functions such as level matching of the Nios II processing unit control signals, DAC control output, signal modulation, etc. The AD669 device is a high-performance, parallel DAC chip produced by Analog Devices, Inc. It features a bipolar voltage output (voltage between -10 and +10 V), 16-bit conversion precision, nonlinear error ≤0.003%, and an output setting time of ≤13 μs. The Nios II system writes the operation result into the DAC through the general-purpose I/O (GPIO) and controls the pin startup and conversion through the LDAC, DACS, etc. In the circuit, the 74LS4245A device performs level matching and the OP07 emitter provides impedance transformation with the circuit. Figure 16 shows the DAC drive unit structure and a photo of the circuit board.

Figure 16. DAC Drive Unit Structure and Photo of Circuit Board





Performance Parameters

The high-precision VCM control system should have quick response, small overshoot, and a stable rate. Therefore, besides performance benchmarks, we must consider the relationship between parameters when designing the control system. The system's main technical indicators are:

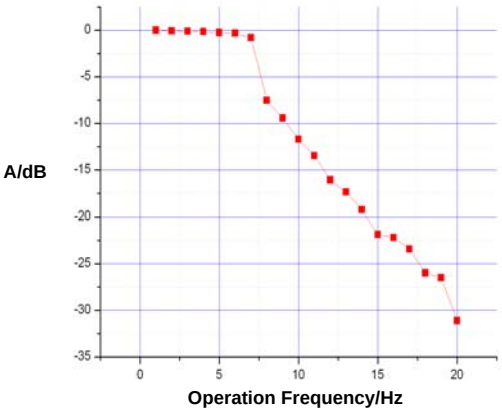
- Bandwidth: 10 Hz
- Maximum travel of VCM: 300 mm
- Minimum resolution: 80 nm

- Maximum tracking speed: 400 mm/second
- Steady state error: ≤3%

System Bandwidth

The system bandwidth reflects the control system's reproduction of useful signals and suppression of useless signals. The control system must have a quick response to curb the dynamic error after disturbance and fast attenuation to reduce interference, so we proposed a bandwidth requirement. Because the control system model is unknown, signal excitation is reproduced to acquire the actual system bandwidth. In this paper, we used a sine signal with a 300-mm amplitude (placed at the maximum travel of the system) on the controlled object and gradually increased the excitation signal's frequency to obtain the amplitude frequency characteristics curve. See Figure 17.

Figure 17. System Amplitude Frequency Characteristics Curve



According to the test result, the system responds quickly when the excitation signal frequency is around 8 Hz, but the position falls significantly as the frequency increases. Therefore, the system bandwidth for our project meets the 10-Hz design requirement.

Minimum Resolution and Maximum Tracking Speed

VCM features quick response and high tracking speed. The maximum tracking speed is mainly restricted by the output frequency of the laser interferometer; the system resolution and maximum tracking speed parameters are contradictory. According to performance parameters of the laser interferometer, the laser interferometer resolution is set as 80 nm and the tracking speed upper limit is 400 mm/second.

Steady State Error

We used step signals with different amplitudes for excitation to acquire the system position response and analyzed the test result to obtain the design's PID parameter. The system controls the steady state error to within 3% of the total amplitude.

Design Architecture

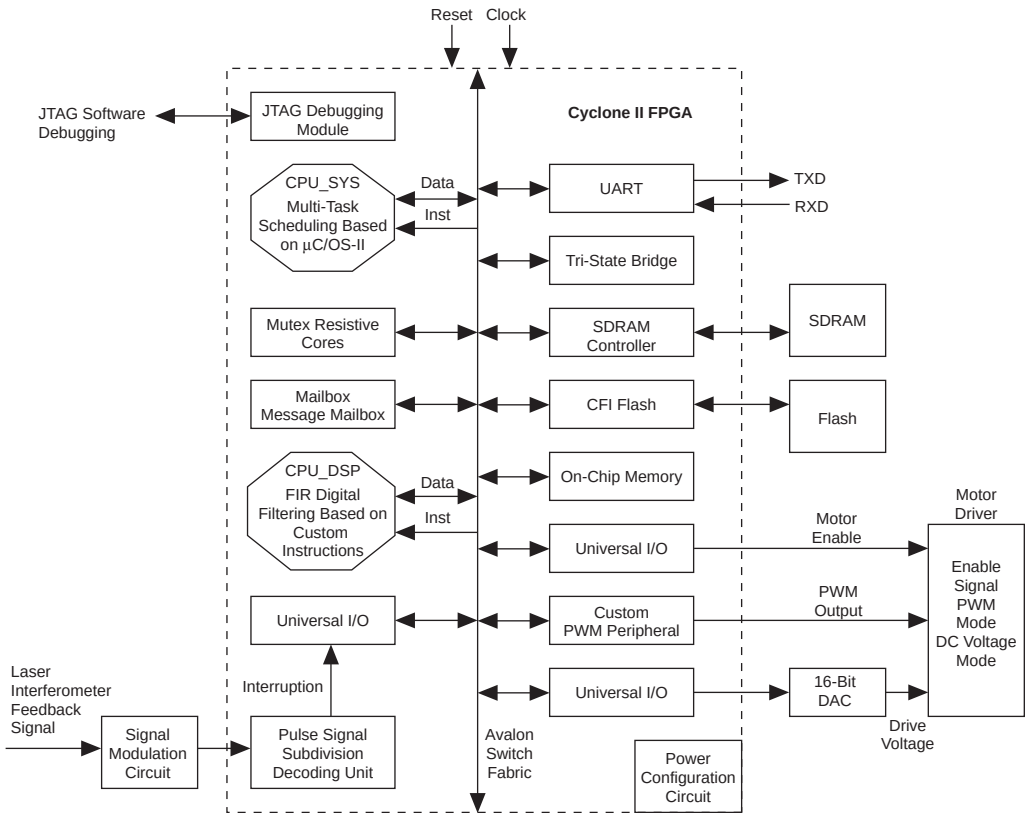
This section describes the design architecture.

System Hardware Design Diagram

The hardware structure of the SOPC-based laser direct writing digital servo controller comprises the DE2 development board, sensor signal modulation circuit, DAC drive circuit, etc. The core control unit (the DE2 development board) consists of a Cyclone EP2C35 FPGA, SDRAM, flash memory, UART interface, USB-Blaster download interface, clock, configuration circuit, power, etc. The EP2C35 FPGA integrates the dual Nios II processors, CPU_SYS and CPU_DSP, based on the Avalon switch fabric bus. To ensure correct access and modification of shared resources, we used Mutex resistive cores to restrict access rights. Data exchange between the two cores is implemented using MailBox (inbox and outbox).

To enable serial port communication, SDRAM, flash chip access, DAC control, etc, we configure several additional components, including an Avalon bus-based UART core, tri-state bus, SDRAM controller, CFI flash controller, M4K on-chip RAM, and universal I/O. Additionally, we customized the PWM subordinate peripheral to acquire programmable PWM pulse signals and allow the motor servo driver to operate in PWM drive mode. Figure 18 shows the system hardware block diagram.

Figure 18. System Hardware Block Diagram



System Software

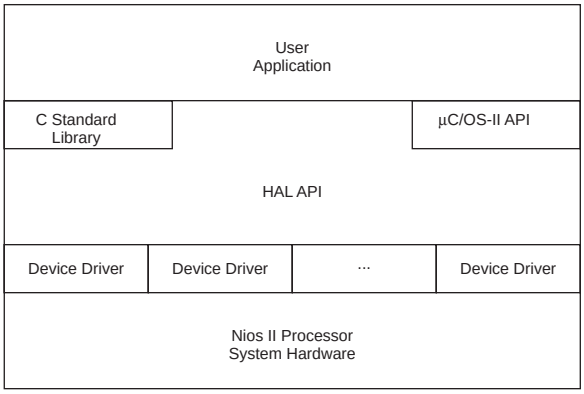
This system integrates the CPU_SYS and CPU_DSP dual Nios II soft-core processors based on the Avalon switch fabric bus. CPU_SYS is responsible for position and speed signal conversion, integration of the split PID operation, UART communication, PWM and DAC peripheral output control, etc., and

is dispatched and managed by the µC/OS-II embedded RTOS. CPU_DSP is responsible for acquiring the sensor decoding signals, providing digital FIR filtering, etc.

CPU_SYS Implementation

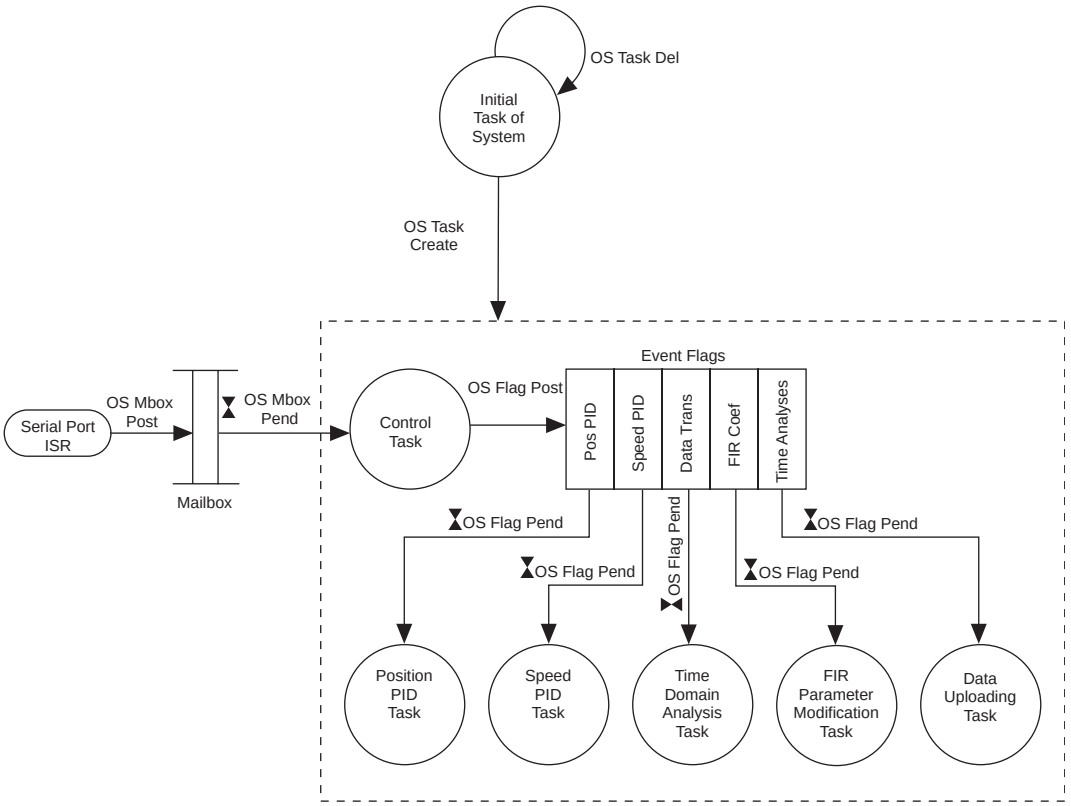
µC/OS-II is a portable, scalable, preemptive, real-time multi-tasking kernel compliant with the Radio Technical Commission for Aeronautics (RTCA) DO-178B level requirements adopted by the United States Federal Aviation Administration (FAA). It has high stability and security. Migration of µC/OS-II to the Nios II processor is based on the hardware access layer (HAL): the programs are not sensitive to low-level hardware changes and we can invoke the HAL API function. Figure 19 shows the µC/OS-II program structure.

Figure 19. µC/OS-II Program Structure



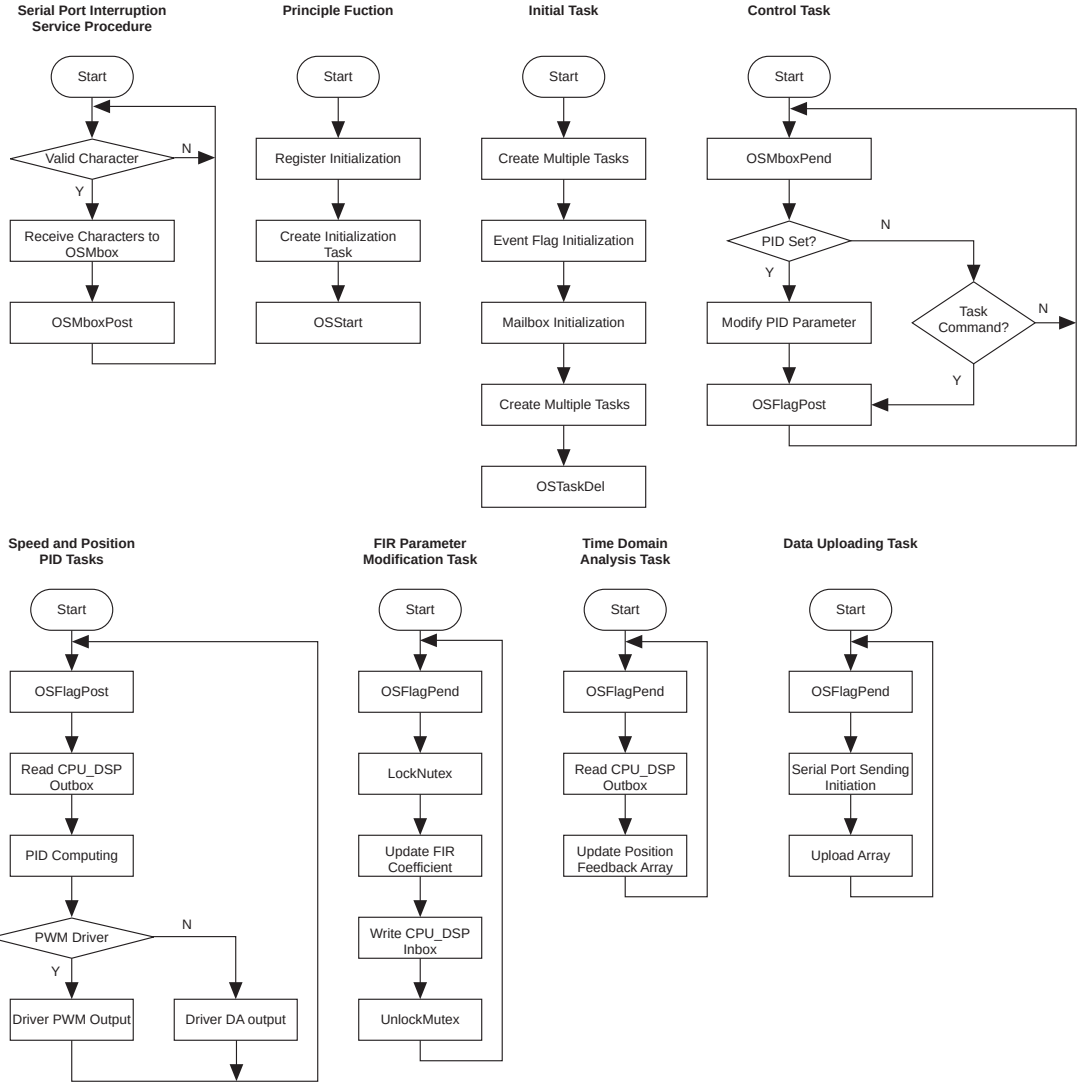
CPU_SYS, the central processing soft core, is based on the µC/OS-II multi-tasking environment. Figure 20 shows the state transition structure.

Figure 20. Task State Transition Structure



The task sequence, according to priority, is: system initial task, control task, position PID task, speed PID task, time domain analysis task, FIR parameter modification task, and data upload task. Figure 21 shows the serial port interruption and specific task flow charts.

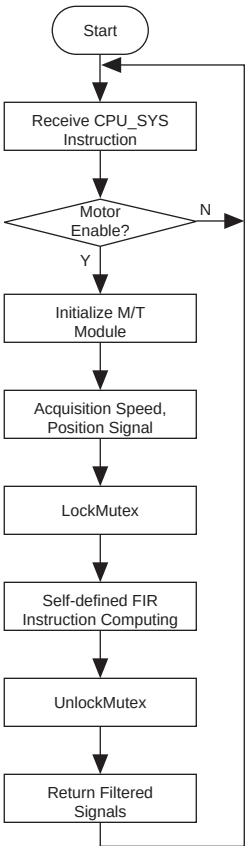
Figure 21. Serial Port Interruption Service and Task Flow Charts



CPU_DSP Implementation

CPU_SYS, the data processing soft core, initializes the M/T module, receives CPU_SYS commands and FIR filter parameters through the message mailbox, protects the shared memory using hardware resistive cores, and uses a custom multiply-add instruction for efficient digital FIR filter computing. Figure 22 shows the CPU_DSP flow chart.

Figure 22. CPU_DSP Flow Chart



Design Methodology

We used the following design methodology.

Nios II System Overall Resource Configuration

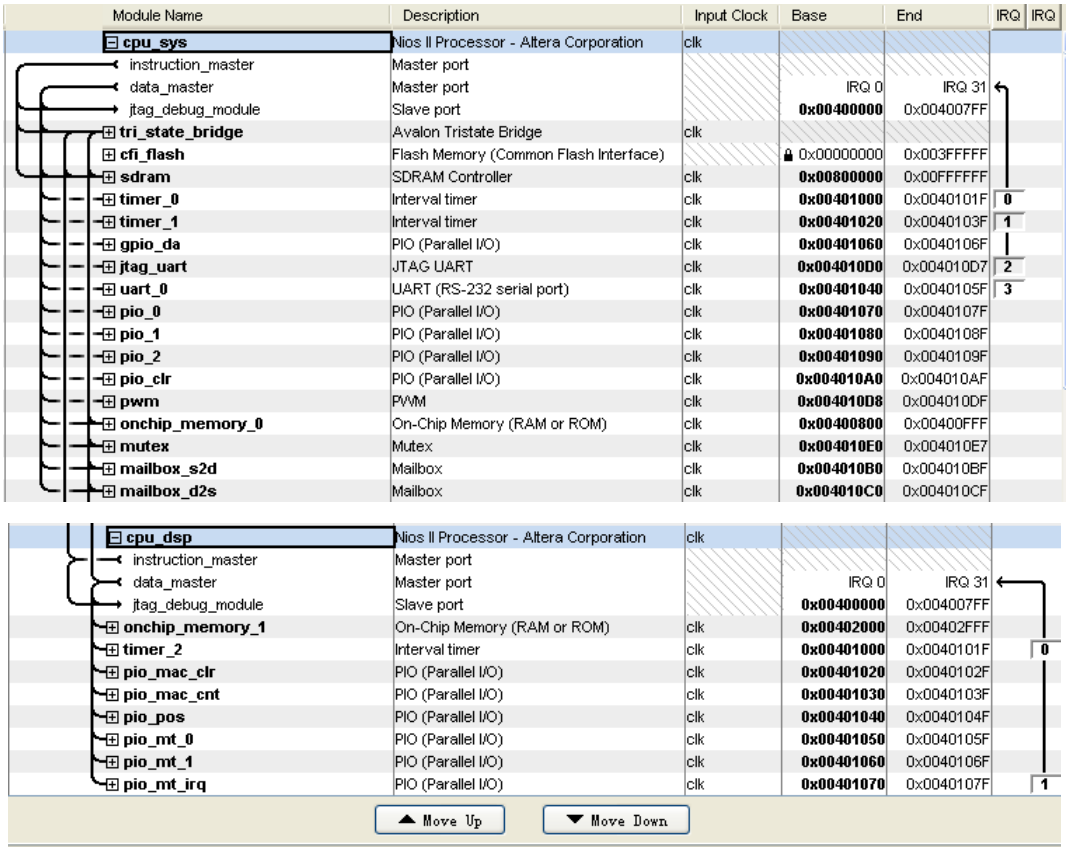
We implemented a multi-processor structure based on the Avalon bus: the dual cores share SDRAM and flash memory and start from different flash memory addresses by setting different reset addresses. We set multiple startup codes and programs in a single flash device based on the Nios II Integrated Development Environment (IDE) programming environment. By setting a break address, we can divide the SDRAM into two equal parts that operate in their respective areas without interfering with each other. The programmable I/O (PIO) ports are configured for the cores to set the signal subdivision decoding unit and read the position counting and M/T speed measurement results. The UART serial port communication module is configured for CPU_SYS to communicate with the high-level LabWinsows/CVI computer and control the motor state in real time. Considering memory's read/write speed, the system allocates a 4-Kbyte on-chip memory for CPU_DSP to implement fast storage position sampling and speed signals. The dual cores share another 4-Kbyte on-chip memory for FIR filter coefficients and to protect resources using Mutex resistive cores. The dual cores send messages through two mailboxes for task synchronization. Figure 23 shows the resource configuration structure.

Figure 23. Nios II System Resource Configuration Structure

Admin / Resource	CPU_DSP	CPU_SYS
PIO	5 Groups	6 Groups
SDRAM	4M	4M
FLASH	2M	2M
UART	UART_0	
PWM	1 Channel	
On-Chip Memory	4KB (Shared by Mutex)	
		4KB (Fast Signal Sampling)
Mailbox	Mailbox_d2s →	
	Mailbox_d2s ←	

We implemented this configuration in SOPC Builder and generated the Nios II system. Figure 24 shows the system structure.

Figure 24. Nios II System in SOPC Builder



Nios II Soft-Core Interaction

We added Mutex components to the design to prevent operation errors when the CPU_DSP and CPU_SYS cores modify the shared FIR parameters. Before accessing shared resources, the processor first tests whether Mutex is available; if it is, the resource access right is obtained in the operation. After using the Mutex-related shared resources, the processor releases the Mutex. Resource sharing between the dual cores is implemented with the hardware configuration shown previously.

The cores interact using MailBox; we configured `MailBox_d2s` and `MailBox_s2d` in this design. MailBox consists of two resistive cores and a shared memory. The resistive cores ensure single read-write operation for the shared memory, and the size of the shared memory is configurable. In the HAL, Altera provides the `altera_avalon_mailbox_pend()` and `altera_avalon_mailbox_post()` functions for using the mailbox. We implement message interaction and task synchronization between the cores using these functions.

Custom PWM Peripherals

To acquire the PWM drive signal, we designed custom PWM peripherals. See Figure 14 on page 182 for the hardware structure. The Nios II soft-core processor accesses the peripherals through the Avalon bus. The access interfaces include `reset` (bus reset signal), `clk` (bus clock), `avalon_chip_select` (bus chip select signal), `addrest` (1-bit address signal), `write` (write enable signal), and `writedata` (32-bit write data signal). The peripherals export drive signals through the `pwm_out` pin.

The write drive functions are `user_avalon_pwm_init()` (for initialization of the custom peripherals), `user_avalon_pwm_enable()` (for enabling the output), `user_avalon_pwm_disable()` (for disabling the output), and `user_avalon_pwm_data()` (for setting the duty ratio and frequency).

Custom Multiply-Add Instruction

For a Nios II embedded system, time-critical software algorithms can be executed faster by adding custom instructions. These instructions simplify complex standard instruction sequences into one instruction that is implemented in hardware, thereby optimizing the algorithm structure. In this design, we customize a multi-cycle instruction for the multiply-add operation, which is a key step in the digital FIR filter algorithm. At a 100-MHz clock frequency, it takes only 40 ns to complete one double 16-bit multiply-add operation, which is faster than the software algorithm. The latency for 128-order filtering is only 13 µs, improving the real-time system performance. Figure 25 shows the multiply-add instruction top-level structure.

Figure 25. Multiply-Add Instruction Top-Level Structure

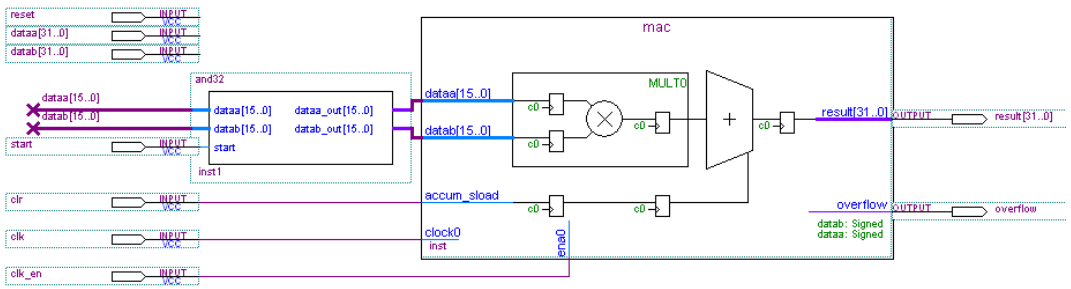
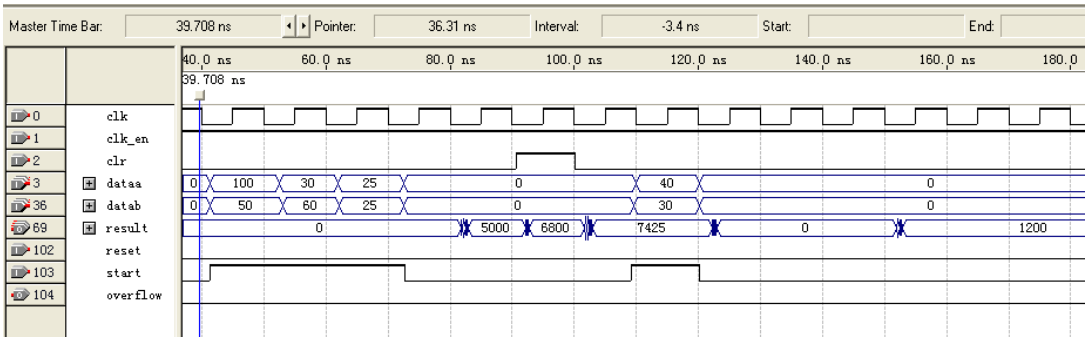


Figure 26 shows the simulation result of this module. In this figure, `clk` is 100 MHz, `dataa` and `datab` perform synchronous computation with `clk` under the control of the `start` signal, and the accumulator signal `overflow` indicates whether there is overflow in the result.

Figure 26. Multiply-Add Instruction Simulation Result



Design Features

Our design has the following features:

- The design implements a digital FIR filter with adjustable parameters using the Nios II processor to curb noise caused by gas film disturbances, mechanical resonance, etc. in the laser direct writing motion control. We customized a special hardware multiply-add instruction for the filtering algorithm to improve the algorithm's execution efficiency and shorten the software implementation time.
- The PWM subordinate peripheral waveform, based on the Avalon bus, is customized and the DAC drive unit outputs the DC voltage. With this implementation, the system can match two drive modes (PWM and DC voltage) of the motor servo driver, reflecting the flexibility of SOPC design.
- For the current VCM digital servo motion control, the MCU (or DSP) is always used for the control algorithm and the FPGA performs signal decoding. The signals are require a lot of resources, and are unstable and vulnerable to noise due to their unmatched speed with on-board signals. Using an FPGA with embedded dual Nios II processors as the main control kernel, this design implements the MCP and DSP functions at the same time, integrates the system to the greatest extent, greatly reduces electromagnetic interference, and improves system stability. Additionally, it simplifies software and hardware design and shortens the product development cycle.
- Migrating the reliable, real-time μC/OS-II into the Nios II processor has a significant effect in critical areas such as motor control.
- We used the LabWindow/CVI software to write the PC driver and operating interface, implement a friendly user interface, and complete off-line analysis of the VCM motion noise spectrum characteristics.
- With a larger FPGA, we could integrate several VCM digital servo controllers into a single chip to implement fast, high-precision monitoring of a multiple-degree-of-freedom motor system.

Conclusion

Altera's RISC-based Nios II soft-core processor had strong performance during system design and debugging. The VCM servo controller with an embedded Nios II soft-core processor based on SOPC technology features fast, flexible, scalable hardware design. SOPC Builder and the Nios II IDE provide a complete solution for software development. The design gives full play to the high-speed, high-integration features of the FPGA, allowing us to implement a complete motor servo control solution on a single chip.

In this design, the PIO reads the M/T decoding module result. We could greatly improve the CPU utilization if this module were re-created as a custom peripheral and connected to the Avalon bus. For the digital FIR filter design, considering the system servo cycle, our project implements a filtering operation of not more than 128 orders. If we used a faster clock, linear phase compensation, and a further optimized custom hardware structure, we could shorten the phase delay and further improve the system control precision. Using the serial port for the data transmission channel is not fast and affects the real-time system performance; using a high-speed bus such as USB can improve this area.

With development of the manufacturing industry, the technical requirements for motor servo control has increased. Our control system based on integrated SOPC technology has significant advantages in system volume, speed, signal integrity, etc., and will become the future trend of motor servo control systems.

Third Prize

Smart Bus Station Sign

Institution: **Oriental Institute of Technology**

Participants: **Jian Jinrong, Zhan Yilin, Lin Taida**

Instructor: **Xiao Ruxuan**

Design Introduction

In modern society, buses are a popular, necessary public vehicle. However, there are some problems and potential dangers with these vehicles. For example, if a person waves at the driver as the bus is approaching, the driver may be distracted and cause a dangerous situation. Or, if the bus driver stops the bus in a hurry to pick up a passenger, the vehicle following the bus may hit the bus. Alternatively, if a person does not pay attention while waiting for the bus, he/she could miss the bus and waste time. Research is needed to solve these issues. Our project proposes a smart bus station sign system that can be used on all roads, improving the convenience and safety of bus passengers.

Target Users

The smart bus station sign's main interface has a variety of buttons. When a user activates the bus calling function, the main interface presents the bus lines in real time, and a module on the bus receives a message from the host machine telling the bus that it has been called. Additionally, a voice function helps the elderly use the system.

Reasons to Use the Nios II Processor in the Design

We used the Nios® II processor for several reasons:

- The Nios II system core is flexible and convenient.
- With the Terasic Development and Education (DE1) board and the Altera® Quartus® II software, we could rapidly develop an embedded prototype system with the Nios II processor.

- Our instructor and seniors at our school have performed considerable research with the Nios II processor, and our school supports this system more than any other.

These advantages allowed us to create the system quickly.

Function Description

This design uses the DE1 board. The design is presented in a digital, modern method, and includes wave-free and bus-coming display functions. With these features, the new bus station sign lightens the burdens of passengers and the driver, leading to safer transportation. Highlights of the design are:

- Smart bus station sign (the module at the bus station)
  - Build a Nios II embedded system using SOPC Builder
  - Create a SD178A voice circuit and an nRF2401 RF module on the DE1 board
  - Design a SD178A controller using VHDL
  - Compile SD178A sub-drivers using the C language
  - Compile sub-drivers using the C language
  - Compile nRF2401 RF module sub-drivers using the C language
  - Compile the program for the main interfaces of the digital bus station sign using the C language
- Smart bus station sign (module on the bus)
  - Build a Nios embedded system using SOPC Builder
  - Create an nRF2401 RF module on the Taurus ACEX1K development board
  - Compile nRF2401 RF module sub-drivers using the C language
  - Compile the main program control settings of the bus sub-module using the C language
- Functions/objectives achieved
  - Build Nios and Nios II embedded systems
  - Display the main interface of smart bus station sign using a 320 x 240 LCD
  - Locate the bus correctly by touching the touch screen
  - Accurately send to the specified bus sub-module and support wave-free and bus-coming display functions
  - Support normal Chinese voice functions
  - Transmit the RF signal without obstacles

Performance Parameters

Table 1 shows the performance parameters for our design.

Table 1. System Performance

Parameter	Description
Flow status	Successful – Wed Sep 26 10:04:46 2007
Quartus II version	6.0 Build 178 04/27/2006 SJ Full Version
Revision name	Minimal_32_sram
Top-level entity name	Minimal_32_sram
Family	Cyclone® II FPGA Family
Device	EP2C20F484C7
Timing models	Final
Met timing requirements	No
Total logic elements	3,899/18,752 (21%)
Total pins	164/315 (52%)
Total memory bits	45,952/239,616 (70%)
Total phase-locked loops (PLLs)	0/4 (25%)

Design Architecture

The following sections describe the architecture of our design.

System Structure

The hardware has two parts: a module at the bus station and a module on the bus. Figure 1 shows the block diagram for the bus station module and Figure 2 shows the block diagram for the bus module.

Figure 1. Smart Bus Station Sign (at Bus Station) Design on DE1 Board

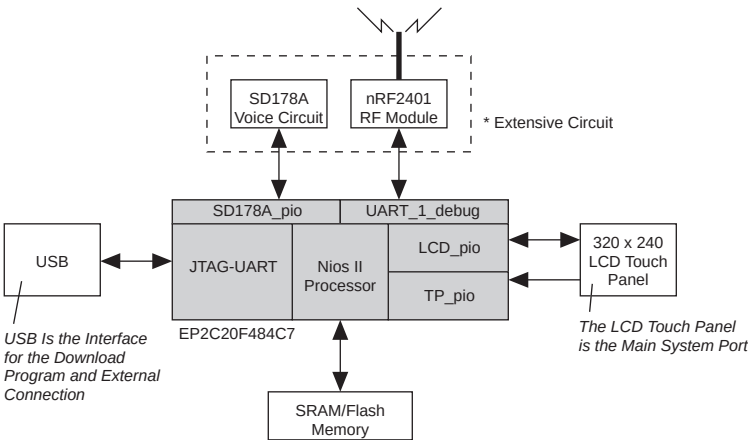
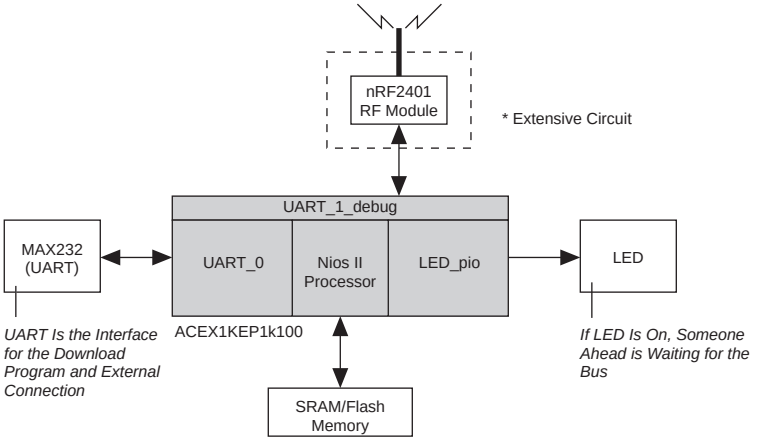


Figure 2. Smart Bus Station Sign (on Bus) Design on ACEX1K Board



Nios II Embedded System Design Diagram

Figure 3 shows the design schematic in the Quartus II software, including:

- Nios II CPU main system design
- Touch screen TOUCHCNTR design (compiled from VHDL code)
- SD178A voice circuit and program SD178A CNTR (compiled from VHDL code)

Figure 3. Design Schematic

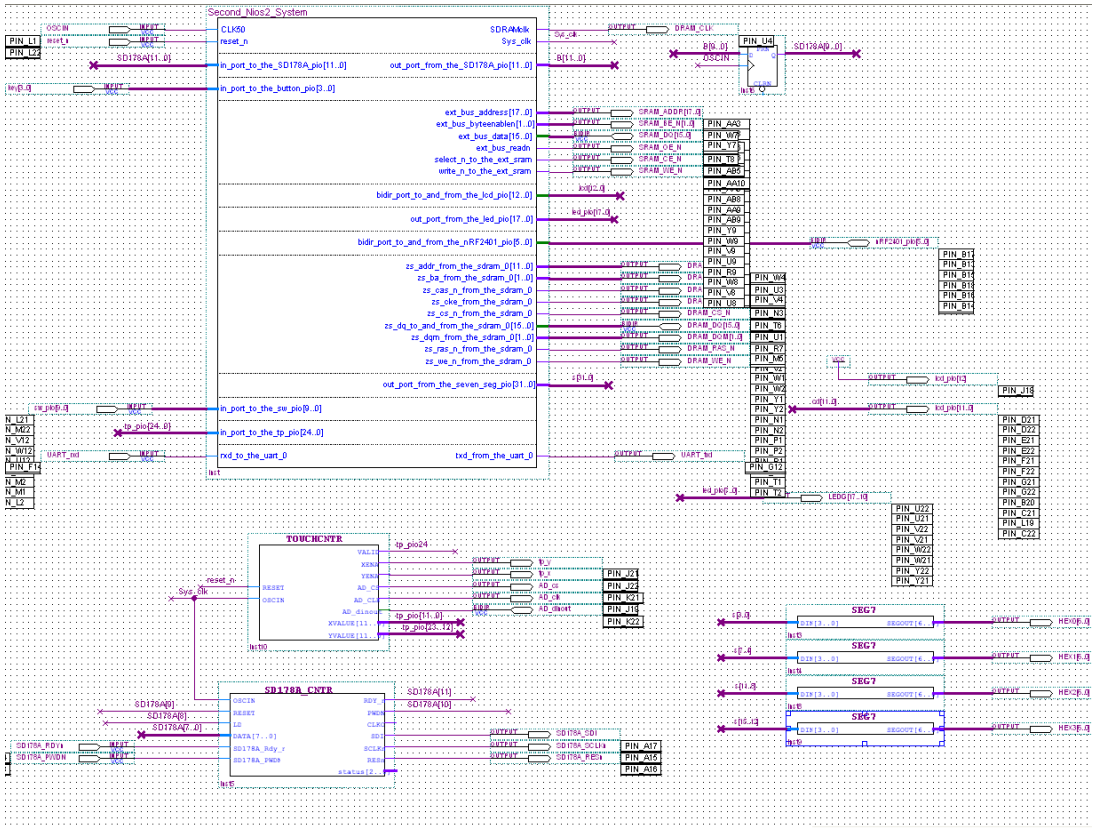
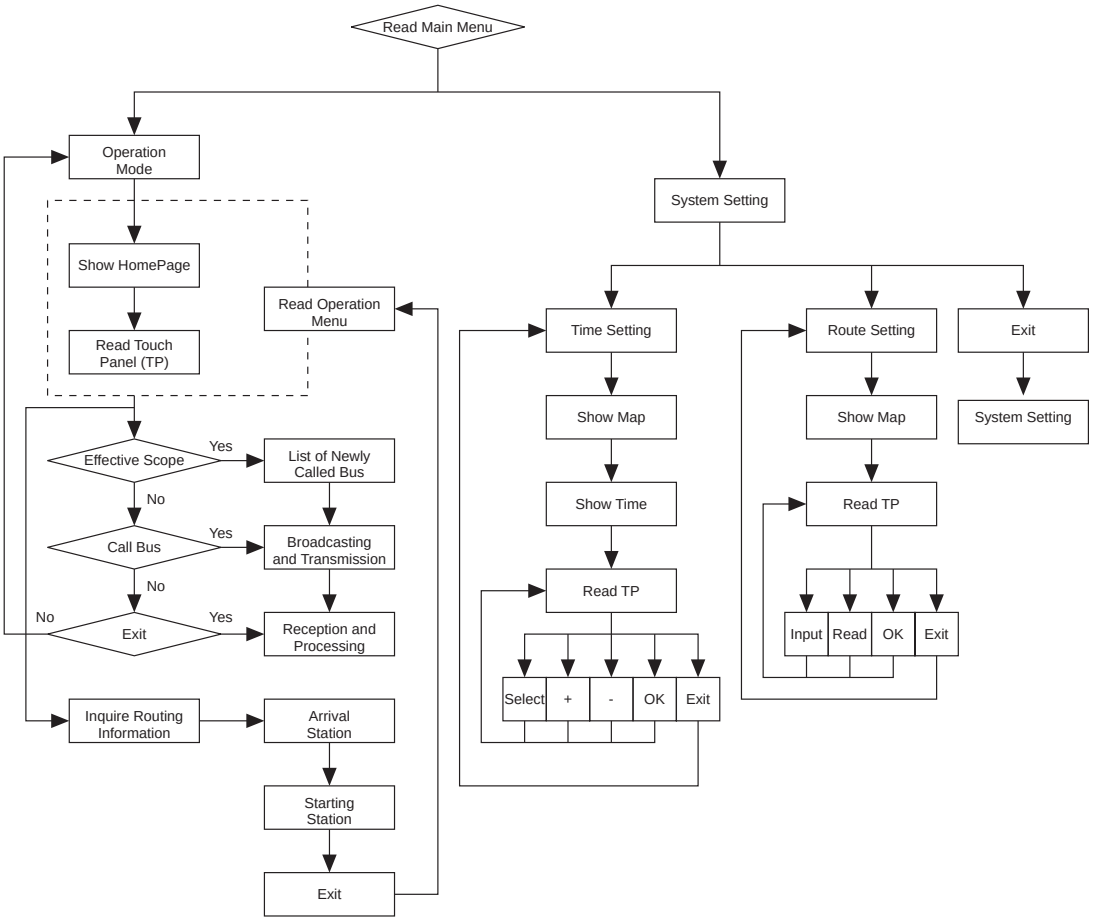


Figure 4 shows the system flow chart for the software on the module at the bus station.

Figure 4. Smart Bus Station Sign Flow Chart (Software at Bus Station)



Software Function User Interface and Flow Charts

Figure 5 shows the main program process and user interface.

Figure 5. Main Program Process and User Interface

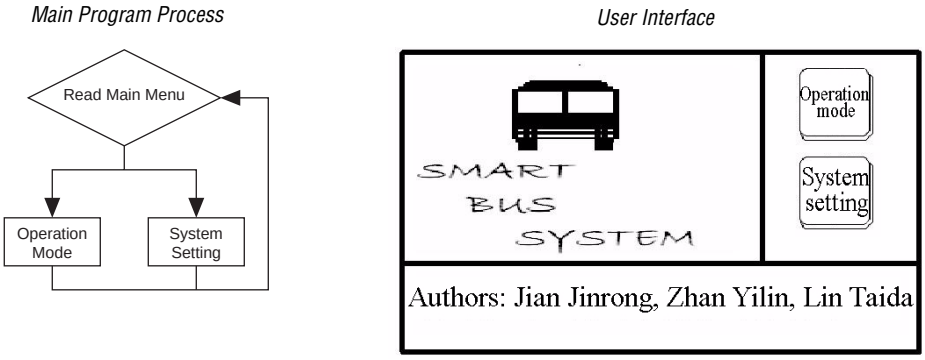


Figure 6 shows the operation mode process and user interface.

Figure 6. Operation Mode Process and User Interface

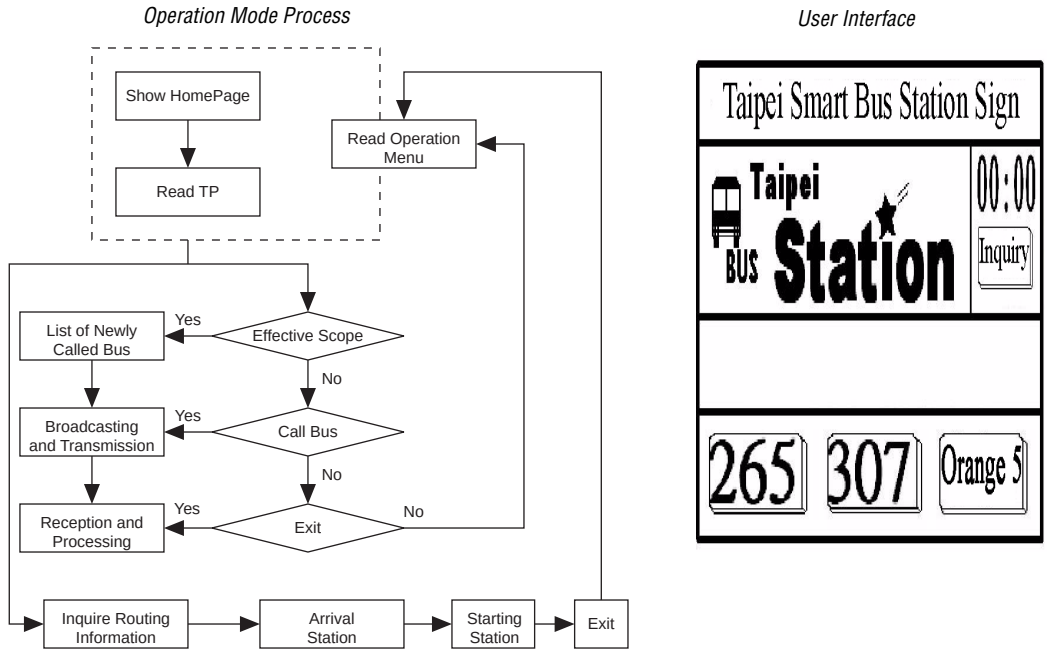


Figure 7 shows the system setting process and user interface.

Figure 7. System Setting Process and User Interface

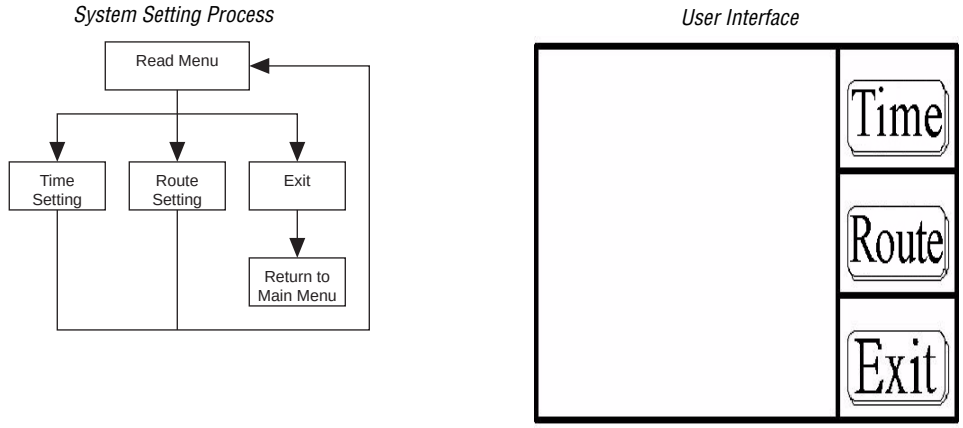


Figure 8 shows the time setting process and user interface.

Figure 8. Time Setting Process and User Interface

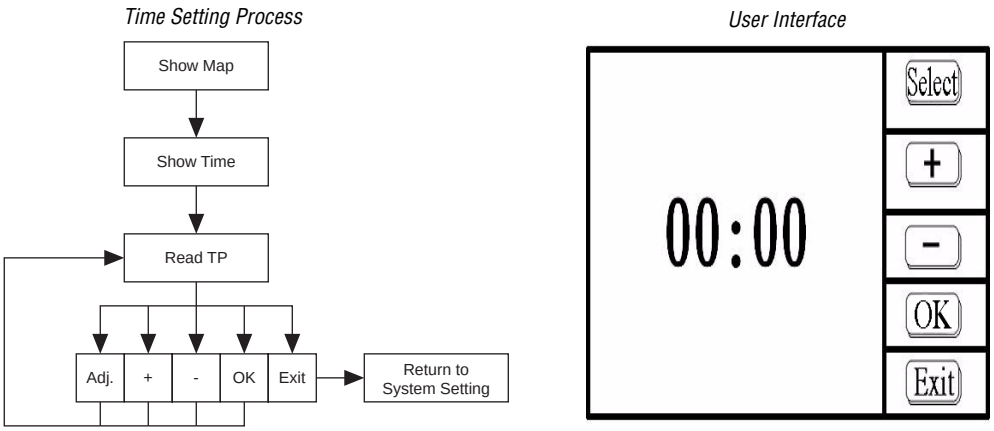
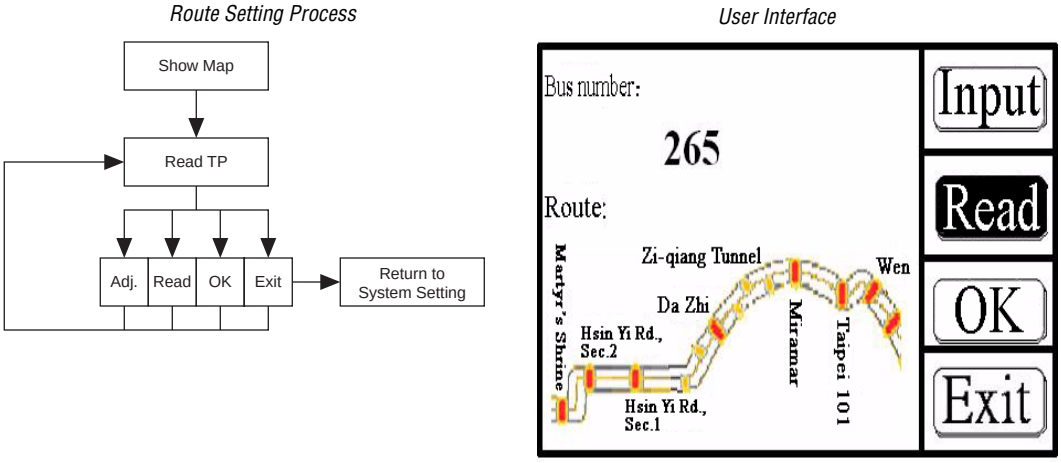


Figure 9 shows the route setting process and user interface.

Figure 9. Route Setting Process and User Interface



Design Methodology

Starting with the DE1 development board, we added a 320 x 240 touch-screen LCD, an RF radio transmission interface, and an SD178 Chinese voice circuit. We performed the following tasks:

- Designed the SD178A controller using VHDL
- Compiled the SD178A sub-drivers using the C language
- Compiled the sub-drivers using the C language.
- Compiled the nRF2401 RF module sub-drivers using the C language

We then used SOPC Builder in the Quartus II software to build the Nios II embedded system and we designed the embedded software in the Nios II Integrated Development Environment (IDE). We compiled the main program of smart bus station sign (at the bus station) using the C language.

Design Features

Our design has the following features:

- Smart bus station sign (the module at the bus station)
  - Build a Nios II embedded system using SOPC Builder
  - Create a SD178A voice circuit and an nRF2401 RF module on the DE1 board
  - Design a SD178A controller using VHDL
  - Compile SD178A sub-drivers using the C language
  - Compile sub-drivers using the C language
  - Compile nRF2401 RF module sub-drivers using the C language
  - Compile the program for the main interfaces of the digital bus station sign using the C language
- Smart bus station sign (module on the bus)
  - Build a Nios embedded system using SOPC Builder
  - Create an nRF2401 RF module on the ACEX1K development board
  - Compile nRF2401 RF module sub-drivers using the C language
  - Compile the main program control settings of the bus sub-module using the C language

Adaptability

We used the DE1 development board, touch-screen LCD, and Nios II processor to compile the smart bus station sign program with the C language in the Nios II IDE. Then, we downloaded the application to the development board and used the touch-screen LCD to input or output the images, thereby digitizing the bus station signs.

Creation

Our project revolutionizes traditional bus station signs and presents them in a modern, digital manner. The system helps users operate the interface, and differentiates between the Chinese, English, Taiwanese, etc. languages, better serving the passengers.

Conclusion

We used software and hardware integration to design the prototype system. Using the DE1 and ACEX1K development boards, we built Nios and Nios II embedded systems and compiled the main programs of host machine for the station sign and in-bus module as well as the SD178A voice circuit. The smart bus station sign prototype features convenience, safety, an excellent user interface, and Chinese voice controls. We want to incorporate these functions into real life, so we will continue to research more in this area.

Third Prize

FPGA-Based Smart Induction Motor Controller Design

Institution: **Electrical Engineering Department, Yuan Ze University**

Participants: **Zhong Zhaoming, Lin Minghong, Chen Yilong**

Instructor: **Lin Zhimin**

Design Introduction

From a control viewpoint, DC motors must be maintained frequently due to their brushes and rectifiers. Sometimes the motor is installed in a location that makes it difficult or impossible to maintain or repair the motor. Compared to the disadvantages of DC motors, AC motors have a variety of advantages. for example, AC motors feature small size, light weight, low rotary inertia, and low price. Generally speaking, induction motors are nonlinear and time-varying with a dynamic coupling system, so the controller design is complicated. When considering control problems, various control theories are often proposed, e.g., proportion-integral-derivative control, sliding mode control, adaptive control, etc. These methods aim to make the system's behavior comply with the design requirements for all system parameter variations and external interferences.

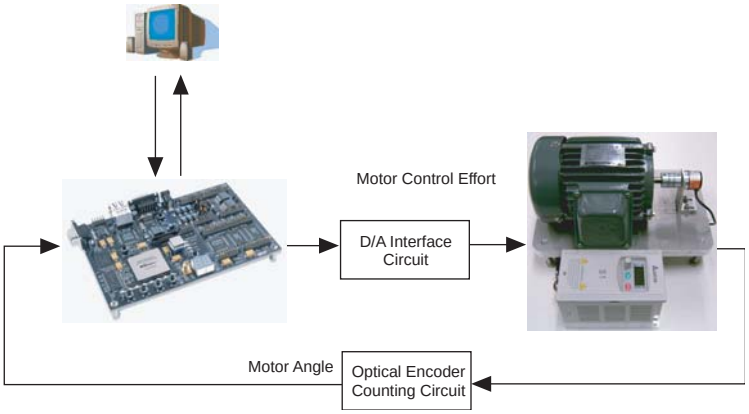
Most of these methods are based on knowledge of status equations for fully or partially controlled systems. However, in practice, the status equation is not easily obtained. Therefore, research for a smart control method with a self-learning capability for better control performance becomes an important subject. Our design uses a neural network (NN) for its amazing effect, which traditional controllers cannot achieve, when the system is involved in an uncertain, time-varying, or nonlinear status, etc.

The key for success using NN is the approximation characteristics. Currently, the methods commonly proposed are the back propagation algorithm, Lyapunov stability method, genetic algorithm (GA), etc. Although the back propagation algorithm is direct and straightforward to use, it is hard to ensure stability and robustness in a closed-loop system. A Lyapunov stability theory-based control structure can ensure system stability, but its computing process is complicated. GA can acquire the global optimization result, but its calculation is large and unsuitable for real-time control. Therefore, our team proposed using an adaptive, fuzzy neural network controller algorithm to control the induction motor.

The whole system can self-regulate parameters in real time based on the learning method deduced from Lyapunov stability theory and back propagation algorithm. The developed algorithm obtains the fastest parameter convergence rate, and is easy to perform and implement.

In actual use, digital controllers demonstrate higher stability, expectable output, and stronger anti-noise capability compared to analog controllers. In particular, the rapid growth of semiconductor technology in recent years makes single-component logic circuits the design trend. If an integrated circuit (IC) can be implemented based on digital circuit integration and control rules, the control system will certainly be less complicated and more reliable, providing smaller hardware, lower design cost, fast execution, and high flexibility. FPGAs are suitable for economic returns and research schedules. Based on these theories, we used Altera's Nios® II processor to implement the design control rules. When writing the program, we chose the Verilog HDL language for hardware and the Nios II processor for the control rule software. Meanwhile, we combined some hardware peripheral circuits to finish the design and construction of the entire experiment environment. Figure 1 describes the design of the proposed FPGA-based smart induction motor.

Figure 1. FPGA-Based Smart Induction Motor Design



The most important and difficult part of this design is implementing the smart control algorithm because the proposed control rules involve many calculations and complicated operations (such as positive/negative numbers and floating-point arithmetic) and Verilog HDL uses binary concepts. Although the complementary code and fixed-point methods are available, users unfamiliar with Verilog HDL grammar can spend a lot of time writing code and it is difficult to perform program maintenance. Instead, using the Nios II embedded processor for the design we wrote our code with the familiar C language without considering positive/negative numbers and floating-point arithmetic, and wrote code directly using the decimal system. Compiling the control rules with the Nios II processor saved a lot of time in parameter adjustment because the Nios II processor is faster to compile than hardware. Additionally, the Nios II processor offers a floating-point custom instruction; adding this instruction greatly shortens the time required for the hardware to process floating-point operations, enhancing system efficiency.

Function Description

Generally, the induction servomotor drive system can be simplified as

$$J\ddot{\theta} + B\dot{\theta} + T_l = T_e \tag{1}$$

Here, J is the rotary inertia, B is a damping coefficient, θ is the motor's rotation, and T_l is additional load interference. T_e is the electromagnetic torque and can be defined as below:

$$T_e = K_t i_{qs}^* \tag{2}$$

$$K_t = (3n_p/2)(L_m^2/L_r)i_{ds}^* \tag{3}$$

Where K_t is the torque constant, i_{qs}^* is the torque current command, i_{ds}^* is the outflow current command, n_p is the pole pair, L_m is the air gap magnetic flux, and L_r is the rotor inductance. In sum, the dynamic equation of induction motor can be rewritten as:

$$\ddot{\theta} = -\frac{B}{J}\dot{\theta} + \frac{K_t}{J}i_{qs}^* - \frac{1}{J}T_l \triangleq A_p\dot{\theta} + B_p u + D_p T_l \tag{4}$$

Here, A_p = -B/J, $B_p = K_t/J > 0$, D_p = -1/J, and $u(t) = i_{qs}^*(t)$ is the control command. The purpose of the entire induction motor control system is to design a control rule, allowing the motor angle to track the control command exactly. Herein, the tracking error is defined as:

$$e = \theta_c - \theta \tag{5}$$

First, provided that both system dynamic functions A_p and B_p of the induction motor and external interference T_l can be acquired, an ideal control rule can be obtained using the feedback control theory.

$$u^* = B_p^{-1}\left[-A_p\dot{\theta} - D_p T_l + \ddot{\theta}_c + k_1\dot{e} + k_2 e\right] \tag{6}$$

Insert equation (6) of the ideal control rule into system dynamic equation (4) and obtain:

$$\ddot{e} + k_1\dot{e} + k_2 e = 0 \tag{7}$$

If the appropriate selection of k_1 and k_2 allows equation (7) to be a Hurwitz multinomial (i.e., its roots are all on the left half plane), control objective $\lim_{t\to\infty} e = 0$ will be achieved. In actual use, however, the system dynamic function and external interference often cannot be acquired, i.e., ideal control rule (6) cannot come true.

To address the problem that the ideal controller cannot be realized owing to failed acquisition of the system dynamic function and external interference, we proposed an adaptive fuzzy neural network controller. Figure 2 shows the block diagram, including an NN controller and a compensation controller, and its arithmetic formula is:

$$u = u_{nn} + u_{cp} \tag{8}$$

The NN controller u_{nn} uses a fuzzy NN to learn ideal controller u^* and uses a compensation controller to overcome the learning error due to the neural controller. First, we define a sliding surface in packet identifier (PID) form as shown below:

$$s = \dot{e} + k_1 e + k_2 \int_0^t e(\tau) d\tau \tag{9}$$

Insert equation (8) into (4) and we get:

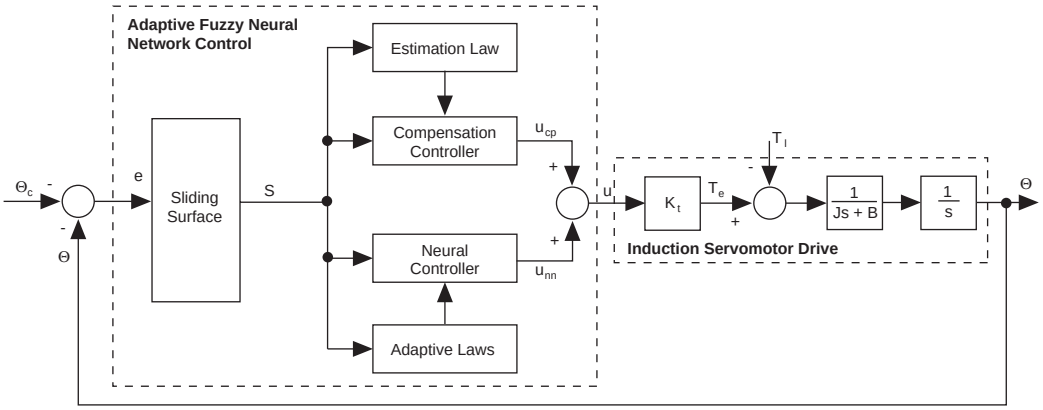
$$\ddot{\theta} = A_p \dot{\theta} + B_p (u_{nn} + u_{cp}) + D_p T_l \tag{10}$$

By subtracting equations (6) and (10), we use equation (9) and get the dynamic equation:

$$\dot{s} = B_p (u^* - u_{nn} - u_{cp}) \tag{11}$$

Figure 2 shows the adaptive fuzzy neural network control system block diagram.

Figure 2. Adaptive Fuzzy Neural Network Control System Block Diagram



According to the approximation theorem, we know that an optimal neural network controller is near to the ideal control (6), i.e.,

$$u^* = \mathbf{w}^{*T} \boldsymbol{\Theta} + \varepsilon \tag{12}$$

Where $\mathbf{w}^*$ is the parameter vector of fuzzy rule of optimal value, $\boldsymbol{\Theta}$ is the activation parameter vector of the fuzzy rules, ε is the approximate error for network learning, and we assume $|\varepsilon| \le E$. In actual use, the optimal network parameter $\mathbf{w}^*$ often cannot be acquired directly or it has multiple solutions, so the estimation machine estimates the entire optimal network parameter, i.e.,

$$u_{nn} = \hat{\mathbf{w}}^T \boldsymbol{\Theta} \tag{13}$$

Here, $\hat{\mathbf{w}}$ is estimation parameter vector of $\mathbf{w}^*$. Thus, the output learning error of network can be defined as:

$$\tilde{u} = u^* - u_{nn} = \tilde{\mathbf{w}}^T \mathbf{\Theta} + \varepsilon \tag{14}$$

Where $\tilde{\mathbf{w}} = \mathbf{w}^* - \hat{\mathbf{w}}$. Insert equation (14) into (11) and simplify (11) as:

$$\dot{s} = B_p(\tilde{\mathbf{w}}^T \mathbf{\Theta} + \varepsilon - u_{cp}) \tag{15}$$

To learn the needed controller parameter online and ensure the stability of the entire closed-loop system, this design deduces the required learning rules based on the Lyapunov stability theorem. Herein, Lyapunov functions are selected as shown below:

$$V = \frac{1}{2}s^2 + B_p(\frac{\tilde{\mathbf{w}}^T \tilde{\mathbf{w}}}{2\eta_1} + \frac{\tilde{E}^2}{2\eta_2}) \tag{16}$$

Where η_1 and η_2 are learning speeds and $\tilde{E} = E - \hat{E}$. Adjust equation (16) for time differential and insert equation (15) into it to obtain:

$$\dot{V} = s\dot{s} + B_p(\frac{\tilde{\mathbf{w}}^T \dot{\tilde{\mathbf{w}}}}{\eta_1} + \frac{\tilde{E}\dot{\tilde{E}}}{\eta_2})$$

$$= s[B_p(\tilde{\mathbf{w}}^T \mathbf{\Theta} + \varepsilon - u_{cp})] + B_p(\frac{\tilde{\mathbf{w}}^T \dot{\tilde{\mathbf{w}}}}{\eta_1} + \frac{\tilde{E}\dot{\tilde{E}}}{\eta_2})$$

$$= B_p \tilde{\mathbf{w}}^T(s\mathbf{\Theta} + \frac{\dot{\tilde{\mathbf{w}}}}{\eta_1}) + sB_p(\varepsilon - u_{cp}) + B_p(\frac{\tilde{E}\dot{\tilde{E}}}{\eta_2}) \tag{17}$$

The learning rule is chosen as shown below:

$$\dot{\hat{\mathbf{w}}} = -\dot{\tilde{\mathbf{w}}} = \eta_1 s\mathbf{\Theta} \tag{18}$$

The compensation controller is chosen as shown below:

$$u_{cp} = \hat{E}\,\mathrm{sgn}(s) \tag{19}$$

And

$$\dot{\hat{E}} = -\dot{\tilde{E}} = \eta_2 |s| \tag{20}$$

(17) can be simplified as:

$$\dot{V} = \varepsilon sB_p - E|s|B_p \le -(E - |\varepsilon|)|s|B_p \le 0 \tag{21}$$

Thus, the designed control system can ensure the system stability based on the Lyapunov stability theorem. To increase the network learning performance, our research introduces a steepest descent algorithm to adjust more parameters. First, an energy function is defined as:

$$E = \frac{1}{2}e^2 \tag{22}$$

Based on the steepest descent algorithm, the adjustment of fuzzy rule is:

$$\Delta w_k = -\eta_w \frac{\partial E}{\partial w_k} = \left[-\eta_w \frac{\partial E}{\partial y_o^4} \frac{\partial y_o^4}{\partial net_o^4} \right] \left[\frac{\partial net_o^4}{\partial w_k^4} \right] = -\eta_w \delta_o^4 x_k \tag{23}$$

Here $\delta_o^4 = \frac{\partial E}{\partial net_o^4}$. Comparing the coefficient of equation (18) with that of (23) and obtain:

$$\delta_o^4 = s \tag{24}$$

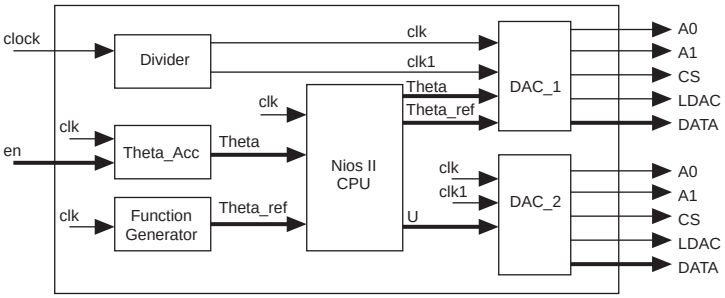
This is a Jacobian item of the entire system and the membership function parameter can be adjusted as:

$$\Delta m_{ij}^2 = -\eta_m \frac{\partial E}{\partial m_{ij}^2} = \left[-\eta_m \frac{\partial E}{\partial y_o^4} \frac{\partial y_o^4}{\partial net_o^4} \frac{\partial net_o^4}{\partial y_k^3} \frac{\partial y_k^3}{\partial net_k^3} \frac{\partial net_k^3}{\partial y_j^2} \frac{\partial y_j^2}{\partial net_j^2} \frac{\partial net_j^2}{\partial m_{ij}^2} \right] \tag{25}$$

$$\Delta \sigma_{ij}^2 = -\eta_\sigma \frac{\partial E}{\partial \sigma_{ij}^2} = \left[-\eta_\sigma \frac{\partial E}{\partial y_o^4} \frac{\partial y_o^4}{\partial net_o^4} \frac{\partial net_o^4}{\partial y_k^3} \frac{\partial y_k^3}{\partial net_k^3} \frac{\partial net_k^3}{\partial y_j^2} \frac{\partial y_j^2}{\partial net_j^2} \frac{\partial net_j^2}{\partial \sigma_{ij}^2} \right] \tag{26}$$

Figure 3 shows a block diagram of the FPGA-based induction motor smart control system. The hardware circuit includes a frequency divider (Divider), induction motor angle counting module (Theta-Acc), and two rows of digital-to-analog converter (DAC) modules (DAC_1, DAC_2). The software is the Nios II embedded processor (Nios II CPU). The following sections provide a detailed description of each module.

Figure 3. Hardware of FPGA Induction Motor Control System Block Diagram



Divider

Because the FPGA's input frequency is 50 MHz, it is divided into the required frequency. First, we designed two frequencies, clk and clk1. One frequency controls the DAC chip select (CS) signal, which updates the DAC data signal (LDAC) and controls the system to input/output data once for every 1

millisecond (ms). The other frequency controls the DAC data selection signal (A0, A1). Control using clk1 is slower than clk, ensuring that the data selection signal is not replaced until the output data arrives at the input latch and avoiding incorrect data output.

Theta_Acc

This module increases the motor angle (*en*) calculated by the optical encoder's 12-bit count circuit to 15 bits using an accumulator. The optical encoder's count circuit output is only 12 bits (0 to 4,095), so a 15-bit register (*Theta*) accumulates the optical encoder's output to make the motor's rotation angle larger than 4,095 for forward or backward rotation.

We designed a judgment condition to determine whether the current angle and the subsequent angle, which are 12 bits beyond the optical encoder (4,095) and smaller than 0, are forward or reverse rotation to calculate the correct rotation angle. A prerequisite is that the current and subsequent rotation angle shall not be larger than 2,047°. Then, the design accumulates the result with an accumulator and outputs it.

Function Generator Module

This module stores the sinusoidal function value in memory. It reads the memory content value using a look-up table to generate the sinusoidal function (Theta_ref) of the motor tracking command. The content value of the sinusoidal function memory has output-enabled amplitude of 2 V and a frequency of 1/2π. It increases as the sinusoidal function of 5/3π after 5.5 seconds.

Nios II CPU

The Nios II CPU writes the induction motor's proposed smart control rule and uses interrupts to control its calculation cycle as 1 ms. When an interrupt is generated during program execution, the design first inputs the motor angle and tracking command output with the induction motor's angle counting module hardware circuit and the function generator module into the Nios II CPU. Then, it calculates the motor's control effort using a CPU control rule. Because the calculated motor control effort ranges from -5 to +5 V and the induction motor's control voltage ranges from 0 to 10 V—in which 5 V means stop, >5 V means forward rotation, and <5 V means reverse rotation—5 V must be added in the displacement method to make the control voltage range 0 to 10 V. Finally, this control effort is output to the next module.

Digital-to-Analog Control Module (DAC_1, DAC_2)

This first group of digital-to-analog (D/A) modules outputs tracking commands and the motor angle, and the output voltage ranges from -5 to +5 V. Because the DAC IC can output two groups of signals (12 bits) but can only receive 8 bits of data, the tracking command and motor angle are output four times. First, the system separates the 12 -bit tracking commands and motor angle for output into 8 low-bit data and 4 high-bit data, and outputs the data to the DAC input latch with data selection signal (A1, A0), respectively. It uses the update signal (LDAC) to transfer the input latch data to the DAC latch to output the updated data. The output order is: output 8 low-bit, tracking command data and its 4 high-bit data. Then, output the 8 low-bit data of the motor angle and its 4 high-bit data.

The second group of D/A modules outputs the motor control. Because the DAC IC can output two groups of signals, the two groups of 12-bit motor control effort are also separated into 8 low-bit data and 4 high-bit data and output four times. The motor control input voltage of one group ranges from 0 to 10 V for controlling induction motor; the other group ranges from -5 to +5 V for connecting the oscillometer for observation.

Performance Parameters

Because we used an FPGA in this design, both the angle sampling cycle and induction motor control frequency can reach 1 kHz. Compared to a computer or single chip, which was used in the past, the FPGA remarkably boosts the control performance. Particularly, using the Nios II embedded processor to calculate the proposed smart control rule not only simplifies writing the program with Verilog HDL

but greatly shortens the development time. Additionally, because we can add peripheral devices quickly and easily, and many intellectual property (IP) cores are available, we can easily adjust the entire control system according to our needs and amend the control rule operation parameters and algorithm rules quickly.

Design Architecture

The design's peripheral circuit, as shown in Figure 4, contains an optical encoder count circuit and two groups of D/A signal circuits with adjustable output voltage. The entire induction motor positioning control experiment environment is shown in Figure 5 and the Nios II system circuit designed with the Quartus II software is shown in Figure 6. Figure 7 shows the software flow chart of the C language program flow written using the Nios II Integrated Development Environment (IDE).

Figure 4. Peripheral Hardware Printed Circuit Board

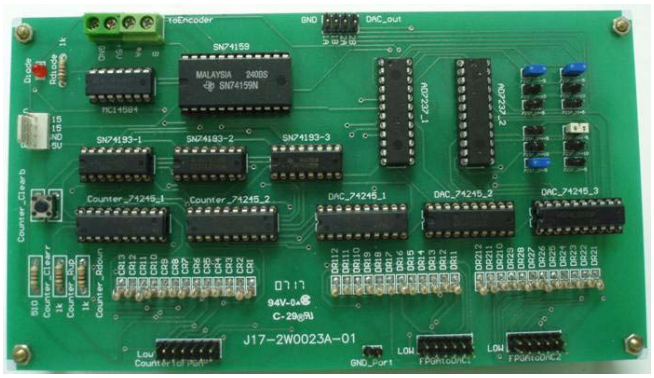


Figure 5. FPGA-Based Induction Motor Smart Control System Experiment Environment

Figure 6. FPGA-Based Induction Motor Smart Control System Circuit Design

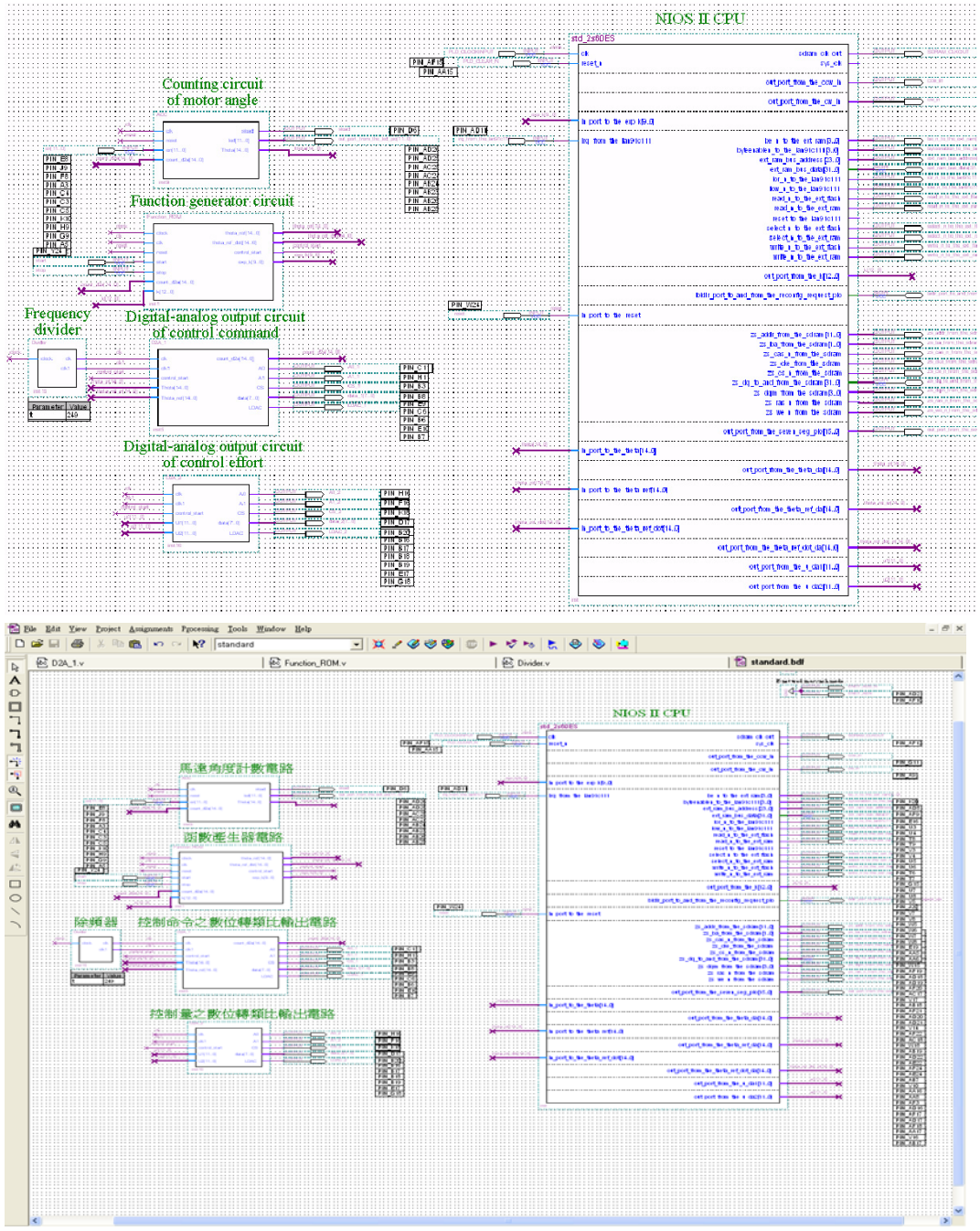
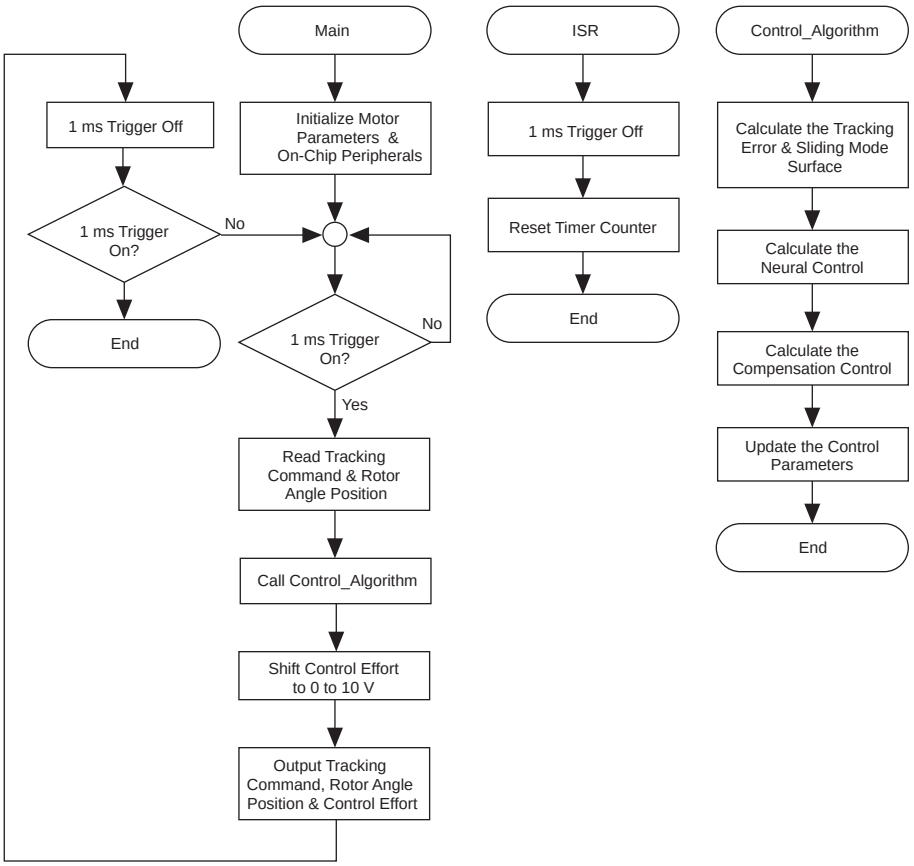


Figure 7. Software Flow Chart



Design Methodology

We used the methodology described in the following steps:

1. Design the peripheral hardware circuit of the induction motor system, including one group of the optical encoder count circuit and 2 D/A circuit groups. The optical encoder count circuit receives the rotation angle of the optical scale/encoder climate count induction motor, and the MC14584B IC postpones the optical encoder's phase A and phase B, causing fourfold resolution. The SN74159 IC uses the fourfold frequency signal to determine whether the induction motor rotates forward or backward and calculates its rotation angle with three 4-bit SN74193 ICs, obtaining the motor's actual angle. For the D/A design, we use the AD7237 IC, which has two channels capable of outputting 0 to 10 V and -5 to 5 V voltage signals, respectively. The 74245 IC prevents the current from back-flowing to the FPGA-based development board.

2. Use the Quartus II software to write Verilog HDL code (motor angle count module, D/A control module) corresponding to the optical encoder count circuit and D/A circuit. Perform simulation and actual hardware testing using the Quartus II software and peripheral hardware circuit, and use an oscillogram and embedded logic analyzer to verify the functionality.

3. Write a divider and control command function generator module. Use the divider to manage the system hardware control cycle as 1 ms. Pinpoint the command function value as 3 bits after the decimal point and stored it in the sinusoidal function memory.

4. Create a Nios II embedded processor for the induction motor control system using SOPC Builder. The system contain a 32-bit Nios II CPU, floating-point custom instruction, Nios II flash storage and SDRAM, Avalon® tri-state bridge, system ID peripheral, JTAG UART, timer for the Nios II processor, timer for interrupts, PLL providing the CPU and SRDAM clock, and an Avalon PIO for the motor angle, tracking command, and control effort. Combine the system with the hardware program to complete the hardware structure of the induction motor control system.

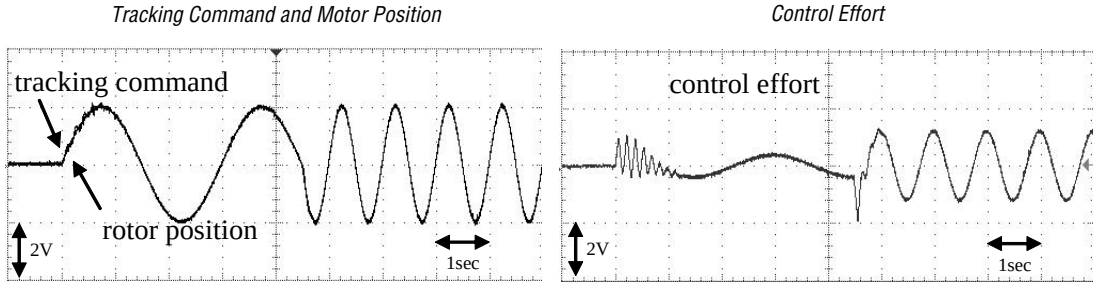
5. Use the Nios IDE to write a software program containing the peripheral device, smart control rule of the induction motor, control program, and main program. Design a 1-kHz interrupt program that executes the induction motor's smart control rule once every 1 ms to calculate the induction motor's output control effort.

6. Integrate a peripheral hardware circuit, Verilog HDL hardware program, Nios II software program, and induction motor to complete the FPGA-based induction motor smart control system.

7. After the induction motor is booted, its actual angle is first obtained by the optical encoder's count circuit and is increased by 15 bits through the FPGA hardware's motor angle count module. It is then sent to the Nios II CPU with motor angle tracking command generated by the control command function generator module. After the program is interrupted, the count circuit calculates the motor control effort using the induction motor's smart control rule and conveys it to the D/A module. It outputs to the external D/A chip to control the induction motor after it is converted into analog voltage.

8. Use a digital oscillometer to observe the control result and verify the performance of the entire induction motor smart control system. Figure 8 shows the experiment result, including the tracking command and motor position and induction motor control effort. The result shows that the method proposed provides a good response after it learns the control parameters, and it is not bad when the control command changes. Therefore, we conclude that our proposed method can effectively control the induction motor's rotation angle.

Figure 8. FPGA-Based Induction Motor Smart Control System Experiment Results



9. Subject to the contest, we have no way to demonstrate the result of the FPGA-based induction motor smart control system, so we display the hardware performance by controlling a brushless DC motor when exhibiting the project. In this case, it further reveals and proves the Nios II functionality and convenience as well as the adaptability of the smart control system. When the control objects are different, we do not need to change the hardware design or the controller parameters because of the on-board artificial intelligence.

Design Features

Integrated with an artificial intelligence smart control technology, we developed this design and applied it to various examples. Applying induction motor positioning control demonstrates the superiority of the design. This design has the following features:

- *Fast learning capability and robust control rules*—Because induction motor control is complicated and motor parameters easily vary with operation, traditional methods cannot provide effective controls. Therefore, we proposed an adaptive fuzzy neutral network controller. The entire control system includes an NN controller and a compensation controller, of which the former uses a fuzzy neural network to learn and is near to an ideal controller and the latter ensures the stability of the entire system. The controller can self-regulate its parameters in real time according to the learning method deduced by the Lyapunov stability theory, so that stability of the entire closed-loop circuit system ensures convergence. An induction motor control system integrated with the these two controllers achieves robustness and accurate positioning control of the induction motor control system. The adaptive fuzzy neural network controller has learning ability and adjusts its internal parameter value promptly according to external interference, achieving good control performance. Therefore, even for control systems requiring high reliability and real-time reaction, the designer does not need to worry about poor control caused by parameter changes or equipment failure due to long-term use.

- *Improved performance with FPGA*—In recent years, fueled by improving IC process technology, FPGA technology has become mature and is often applied to various hardware-enabled algorithms. Moreover, more and more systems on chip (SOC) have been used, so the proposed adaptive fuzzy neural network controller is implemented by an FPGA in this design. Compared with computers that were used in the past, FPGAs reduce the size of the control system, improve flexibility, lower cost, and boost the system's calculation and execution speed. In particular, by virtue of the FPGA's reprogrammability, designers can keep changing and planning devices to cater to users' needs.

- *Coded in Verilog HDL*—In the aspect of software writing, the hardware peripheral circuit is designed in Verilog HDL. We used the Nios II processor for the control rules, which simplifies the program and accelerates design and parameter adjustment.

- *Control system meets industry demands*—Using an adaptive fuzzy neural network controller in an FPGA, this design offer low cost, high performance, and high reliability for the induction motor control system. A slightly modified program can be used to control various motors. Featuring small size, high flexibility, low cost, fast processing speed, short production period, and modularized design of the hardware architecture, the system does not require much effort when applied to different control systems. Therefore, the design can be used in various highly efficient controls, including on-line and real-time learning systems (e.g., a machine arm in industrial and medical fields), household robot control systems, car back-up systems, automatic driving systems, and digital wheel chair systems. We believe that the design will contribute to many areas, including industrial and medical fields.

Conclusion

Before the contest we had a basic knowledge of Verilog HDL programming and the Quartus II software, but we were not very familiar with them. We did not know anything about the Nios II processor. During the contest we improved our hardware design ability and found that the powerful Nios II processor can not only accelerate hardware design, but also simplify and facilitate the design methodology. Thus, we could use many tools—such as an embedded processor, floating-point custom instruction, and memory—to plan the hardware quickly and easily. From the results, we were impressed by the FPGA performance. Moreover, the user-friendly Nios IDE allowed us to write, compile, and execute our programs easily. This project was not only helpful for our future research, but also strengthened our competitiveness for development in this field.

Third Prize

Intelligent Solar Tracking Control System Implemented on an FPGA

Institution: **Institute of Electrical Engineering, Yuan Ze University**

Participants: **Zhang Xinhong, Wu Zongxian, Yu Zhengda**

Instructor: **Professor Huang Yingzhe**

Design Introduction

In today's high-tech environment, energy has become the impetus for socio-economic development. Since the Industrial Revolution, humans have used fossil fuels as their primary energy source. However, the amount of fossil fuels on the earth is limited, and their use has caused unprecedented changes to the global ecological environment and climate. Gases from burning fossil fuels can build up in the atmosphere, becoming thicker and thicker to produce greenhouse effects such as rising global temperature and sea level. These effects will dramatically alter our living environment. Fortunately, humans are becoming more conscious of environmental protection, and are seeking new energy sources that cause less pollution and do not threaten the environment. As a free, nonpolluting, inexhaustible energy, solar energy is ideal for generating electricity. Currently, generating electricity by solar energy is inefficient, so our project focuses on how to improve its efficiency.

A solar panel receives the most sunlight when it is perpendicular to the sun's rays, but the sunlight direction changes regularly with changing seasons and weather. Currently, most solar panels are fixed, i.e., the solar array has a fixed orientation to the sky and does not turn to follow the sun. To increase the unit area illumination of sunlight on solar panels, we designed a solar tracking electricity generation system. The design mechanism holds the solar panel and allows the panel to perform an approximate 3-dimensional (3-D) hemispheroidal rotation to track the sun's movement during the day and improve the overall electricity generation. This system can achieve the maximum illumination and energy concentration and cut the cost of electricity by requiring fewer solar panels, therefore, it has great significance for research and development.

Solar Tracking Control System

Our high-performance solar tracking system has multiple functions and uses two motors as the drive source, conducting an approximate hemispheroidal 3-D rotation on the solar array (see Figure 1). The two drive motors are decoupled, i.e., the rotation angle of one motor does not influence that of the other motor, reducing control problems. Additionally, the tracker does not have the problems common to two-axis mechanical mechanisms (that one motor has to bear the weight of the other motor). This implementation minimizes the system's power consumption during operation and increases efficiency and the total amount of electricity generated.

Figure 1. Solar Tracking Array Architecture Scheme

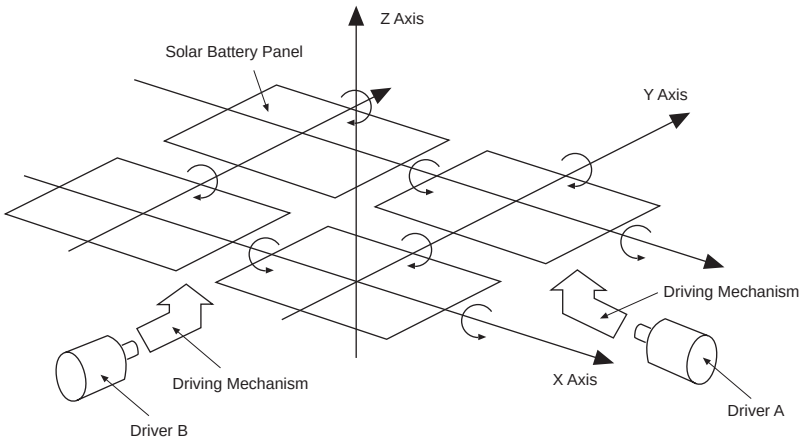


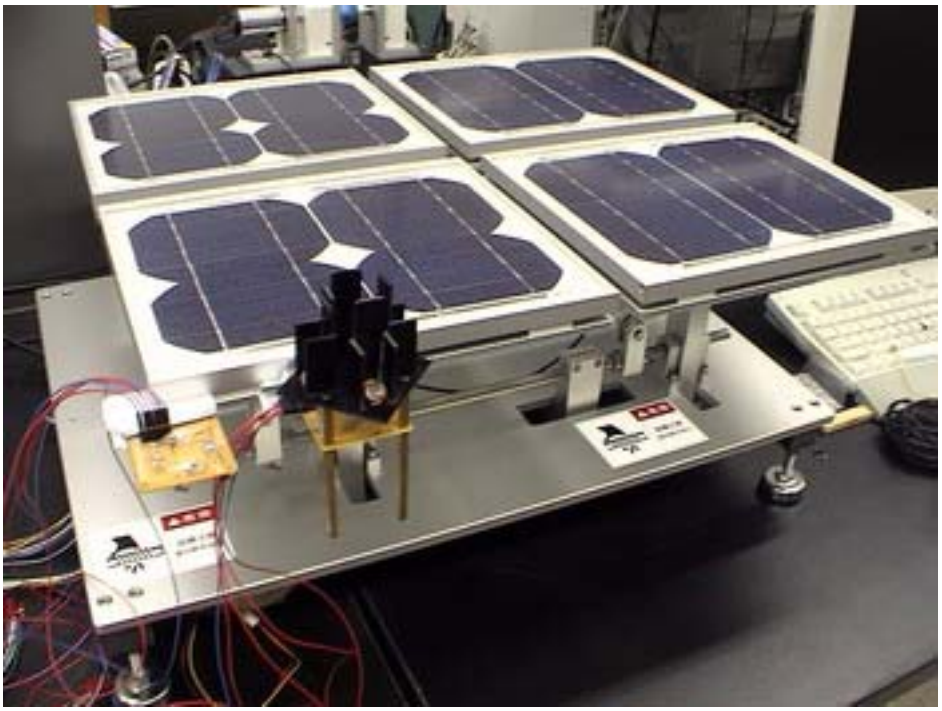
Figure 2 shows the solar tracking system we designed based on the considerations described previously. The mechanism must support the solar panel and allow the panel to conduct 3-D rotation within a certain amount of space. The array-type mechanism has two advantages:

- *High photoelectric conversion efficiency*—Because the flexible panel of the solar tracker array can conduct 3-D rotation, tracking the sun in real time, the system efficiently performs photoelectric conversion and production.

- *Simple, energy-saving controls*—The two rotational dimensions of the array solar tracker are controlled by two independent drive sources. The rotation angles are decoupled and neither one has to bear the weight of the other one. Additionally, the overall movement inertia is dramatically reduced.

We used the Altera® Nios® II processor to perform solar tracking. The design combines a Nios II processor with a two-axis motor tracking controller to integrate peripherals such as microprocessor, memory, and I/O into one Altera FPGA based on system-on-a-programmable-chip (SOPC) concepts. This integration accelerates development while maintaining design flexibility, reduces the circuit board costs with a single-chip solution, and simplifies product testing.

Figure 2. Complete Solar Tracking Control Platform



Function Description

Our design includes three modes: balance positioning, automatic mode, and manual mode.

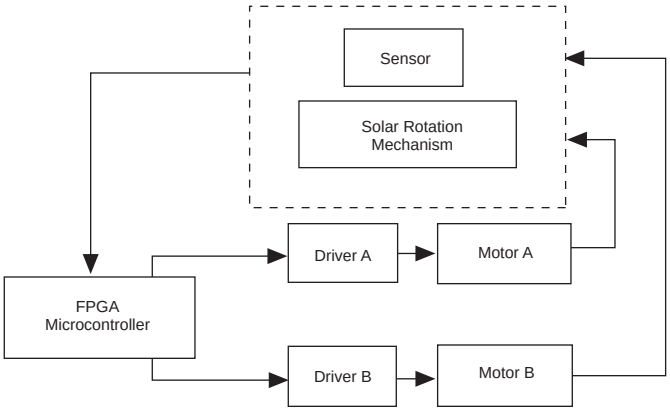
- *Balance positioning*—When setting the solar platform default, we used a mercury switch for balance positioning. The switch sets the four boundaries of the platform and prevents the solar panels's four tilting boundaries from hitting the mechanism platform and damaging it or the motor.

- *Automatic mode*—In this mode, the system receives sunlight onto the cadmium sulphide (CdS) photovoltaic cells and the CdS acts as the main solar tracking sensor. The sensor feeds back to the FPGA controller through an analog-to-digital (A/D) device. The Nios II processor is the main control core and adjusts the two-axis motor so that the platform is in the location for optimal, efficient electricity generation.

- *Manual mode*—If the system has a fault or needs to be maintained, we can switch the system to manual mode. In this mode, we can adjust the system's position to check it or perform repairs.

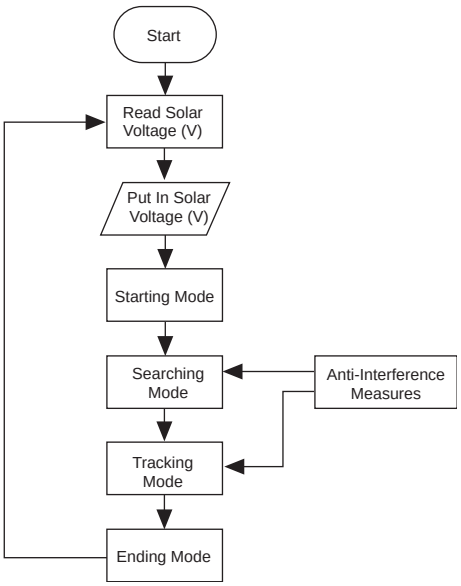
Figure 3 shows the block diagram of the solar tracking system.

Figure 3. Solar Tracker Control Block Diagram



We implemented the system's logic low design using the Nios II processor control circuit. Figure 4 shows the tracking control flow chart. The system starts when we turn on the tracking control circuit's power supply switch. The tracking control circuit performs system tracking, energy saving, and system protection, as well as a designed control mode and external anti-interference measures. External interference includes weather influences, such as wind, sand, rain, snow, hail, salt damage (i.e., salt erosion on the mechanism), etc.

Figure 4. Tracking Control Flow Chart



Performance Parameters

We used the following modules to build our solar tracking system:

- *Balance sensor module*—At the initial system reset, the four switches of the balance sensor are all powered-on at horizontal balance status.
- *A/D module*—We used the AD0804 A/D converter, which has a 100-µs conversion time and 8-bit resolution.

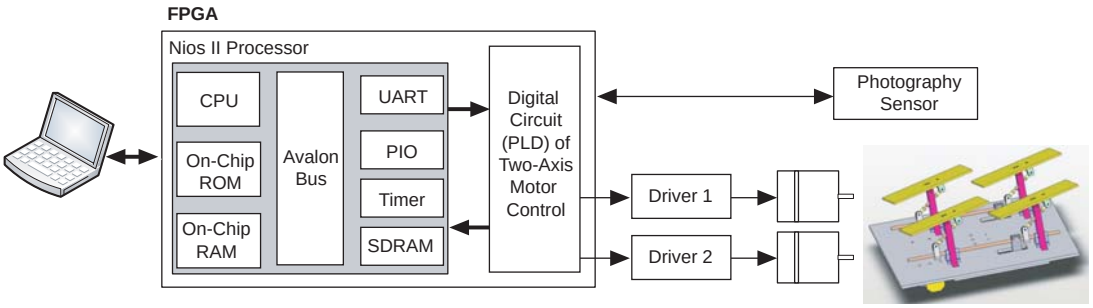
- *Motor control module*—We used a 100-µs circle step motor to control the motor's speed.
- *CPU*—We created a tailor-made 32-bit RISC-based CPU that has the capacity to control the solar system.

For precision, the fuzzy control time should not be more than 0.1 second. In addition to the reset balance (positioning level of 0 degree = 262,144), the balance sensor can also bound the X and Y axes.

Design Architecture

As shown in Figure 5, the Nios II processor is the control center and integrates our two-axis control chip. The system determines which data is fed back to the FPGA using a photography sensor. It conducts the tracking control rule operation to calculate the angle required by the motor and adjusts motor's current angle. It also moves the solar panel to achieve optimal power.

Figure 5. System Architecture



For the hardware design, we first used a balance sensor to set the system's zero point. Then, we designed a tracking sensor to determine the orientation of the solar light source. The signals fed back by the sensor form the basis of the controller input. The control design outputs the signals to control the two-axis step motor and the solar tracking control system. The following sections introduce the hardware.

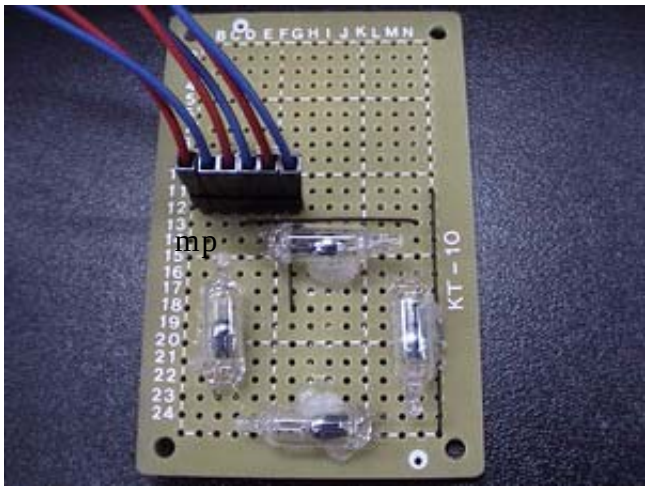
Balance Sensor

For the initial reset balance, we used a mercury switch (also called a tilt switch), which is a kind of circuit switch. Its main body is a mini container that is connected with electrodes and contains a drop of mercury, and usually the container is a vacuum or infused with inert gas (see Figure 6). Because of gravity, the mercury bead flows towards the lower position in the container. If it contacts the two electrodes simultaneously, the circuit closes and the power-on switch opens. According to our design principle, we set four switches (east, west, south, and north) in the mechanism design and fixed the mechanism. If the four switches are all powered-on, the mechanism balances. Figure 6 also shows the balance sensor stereogram.

Figure 6. Mercury Switches and Stereogram

Mercury Switches

Balance Sensor Stereogram



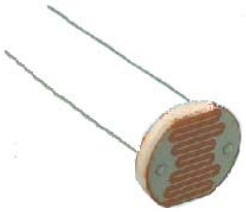
Sensor Design

One of our key modules is the sensor. Because the sensor tracks the solar light source's orientation, selecting the right tracking sensor is very important. CdS sensors (see Figure 7) are cheap, reliable, and photo-sensitive. In our design, the CdS sensor provides the following advantages:

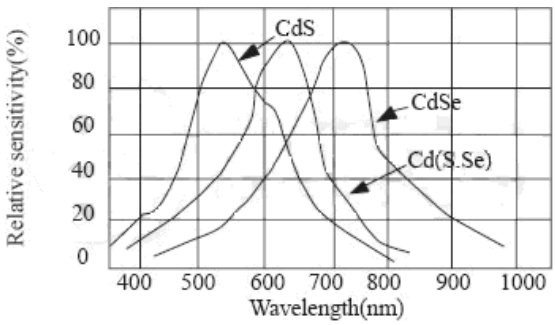
- Without polarity (ohmic structure), the CdS sensor is easy to use.
- CdS sensors have a photo-variable resistor in which the internal impedance changes with the intensity of light energy.
- When the ambient light brightens, the CdS sensor's internal impedance reduces.
- The CdS sensor's photo sensitivity (i.e., spectral characteristics) is 0.4 to 0.8 mm, which is close to the wavelength scope of visible solar light (0.38 to 0.76 mm), as shown in Figure 7.

Figure 7. CdS Stereogram and Sensitivity Scope

CdS Stereogram



CdS Sensitivity Scope



Tracking Sensor Design

The tracking sensor is composed of four similar CdS sensors, which are located at the east, west, south, and north to detect the light source intensity in the four orientations. The CdS sensor forms a 45° angle with the light source. At the CdS sensor positions, brackets isolate the light from other orientations to achieve a wide-angle search and quickly determine the sun's position (see Figure 8). The four sensors are divided into two groups, east/west and north/south. In the east/west group, the east and west CdS sensors compare the intensity of received light in the east and west. If the light source intensity received by the sensors are different, the system obtains signals from the sensors' output voltage in the two orientations. The system then determines which sensor received more intensive light based on the sensor output voltage value interpreted by voltage type A/D converter (ADC) and ADC0804 device. The system drives the step motor towards the orientation of this sensor. If the output values of the two sensors are equal, the output difference is zero and the motor's drive voltage is zero, which means the system has tracked the current position of the sun. The north/south sensors track the position of the sun similarly. Figure 9 shows the sensor stereogram.

Figure 8. Tracking Sensor Internal Design

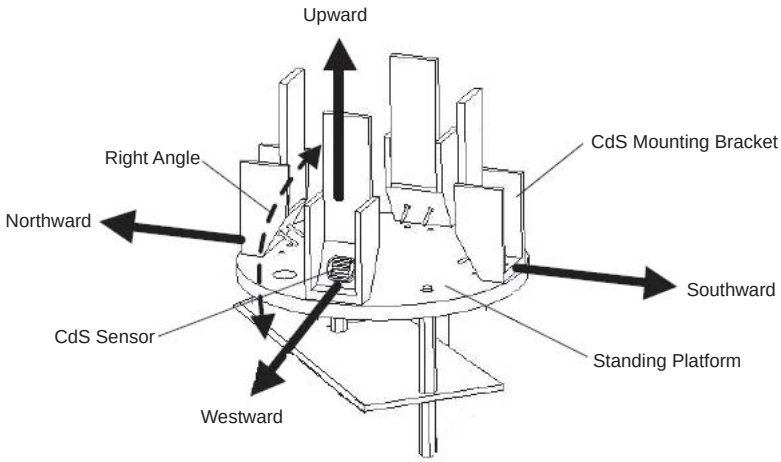
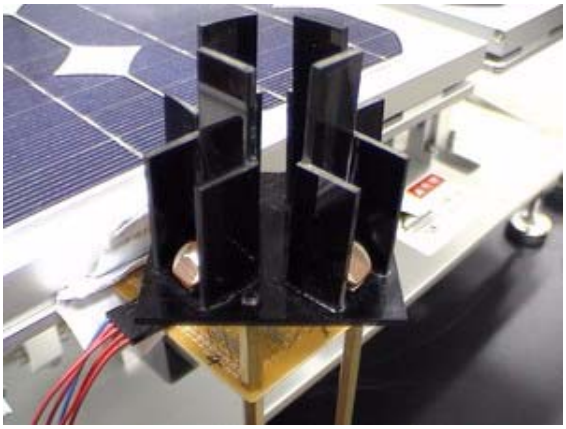


Figure 9. Tracking Sensor Stereogram



ADC

Generally, measured continuous signals such as voltage or current are analog signals. An ADC converts analog signals to digital signals. Digital signals can minimize noise interference during signal

transmission and debug noise interference with encoding technology. Additionally, digital signals are easy to store.

The ADC0804 ADC is a 20-pin device with 8-bit resolution and a single channel operating with a 5-V single power supply. Its analog input voltage scope is 0 V to 5 V. The power consumption is 15 mW and the conversion time is 100 µs. Because the resolution is 8 bits, there is a 256-step quantization. If the reference voltage is 5 V, each step is 5/256 = 0.01953 V. 00000000 (00H) represents 0.00 V and 11111111 (FFH) represents 4.9805 V. The unadjusted error of the ADC0804 device is 1 least significant bit (LSB), i.e., 0.01953 V, including the full-scale error, offset error, and nonlinear error.

Figure 10 shows the ADC0804 pins. `D0` through `D7` is the 8-bit output port. When both $\overline{CS}$ and $\overline{RD}$ are low, digital data is sent to the output port. When $\overline{CS}$ or $\overline{RD}$ is high, `D0` through `D7` float. $\overline{WR}$ is the control signal for initiating conversion. When $\overline{CS}$ and $\overline{WR}$ are low, the ADC0804 device performs deletion; when $\overline{WR}$ goes high, the device performs conversion. `CLK IN` is the time sequence input with a frequency scope of 100 to 800 KHz. $\overline{INTR}$ is high during conversion and changes to low when conversion ends. `Vin(+)` and `Vin(-)` are differential analog signal inputs, usually single-ended inputs, and `Vin(-)` is grounded. The ADC0804 device has two ground ends, `A GND` and `D GND`. `Vref/2` is half of the reference voltage input value if the overhead connection, `2Vref`, is equal to the power supply voltage, V_{CC}. The ADC0804 device is embedded with a Schmitt trigger as shown in Figure 11. If resistance and capacity are added to `CLK R` and `CLK IN`, the time sequence, which is required by operating the ADC, is generated with the following frequency:

$$f_{CLK} \approx \frac{1}{1.1RC}(Hz) \quad (1)$$

In the equation, the time sequence signal is decided by R and C and the signal does not need to be added with `CLK IN`. Figure 12 shows the ADC0804 circuit diagram. The input analog signal is controlled by variable resistance `VR2` and input from `Vin(+)` end with `Vin(-)` being short. `2Vref` is provided by `R1`, `R2` and `VR1`; `C1` and `R3` control the time sequence of the circuit. $\overline{CS}$ and $\overline{RD}$ are grounded to create the chip enable. $\overline{WR}$ and $\overline{INTR}$ receive the `SW1` switch to emulate control signals.

Figure 10. ADC0804 Pin Function Diagram

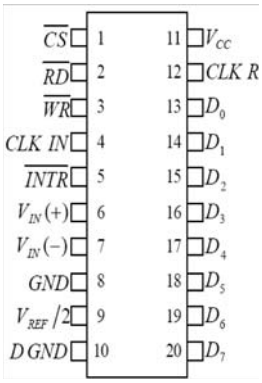


Figure 11. ADC0804 Internal Time Sequence Generation Circuit

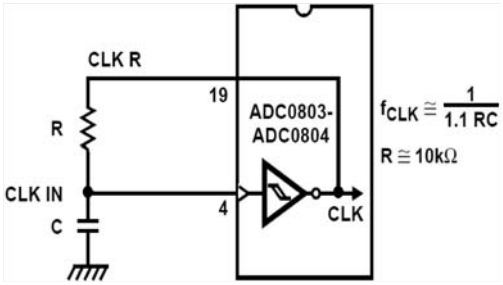


Figure 12. ADC0804 Circuit Diagram

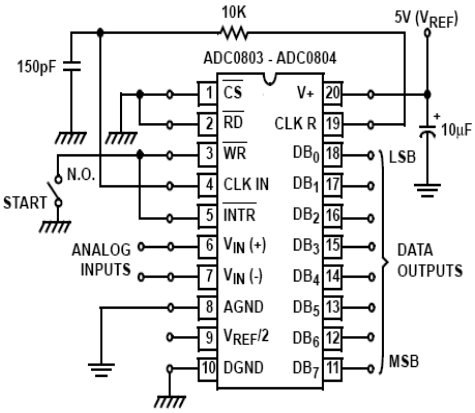


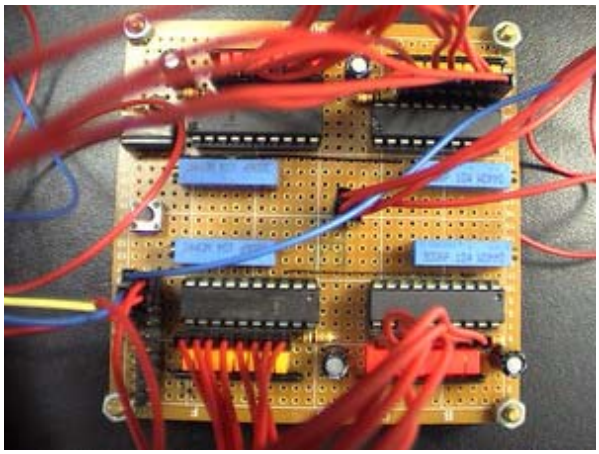
Table 1 shows the ADC specifications.

Table 1. ADC Specifications

Operating voltage	+5 V DC
Analog voltage input scope	$0 \le V_{in} \le +5$ V DC
Resolution	1/256
Conversion output value	0 to 255
Conversion frequency	f_{ck} = 1/(1.1 x R x C)
Conversion error	± 1 LSB
Reference voltage	+ 2.5 V DC

Figure 13 shows the completed ADC circuit stereogram.

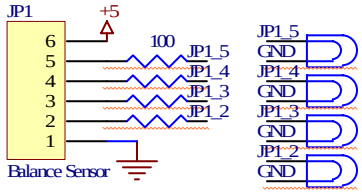
Figure 13. Complete ADC Circuit Stereogram



Sensor Production

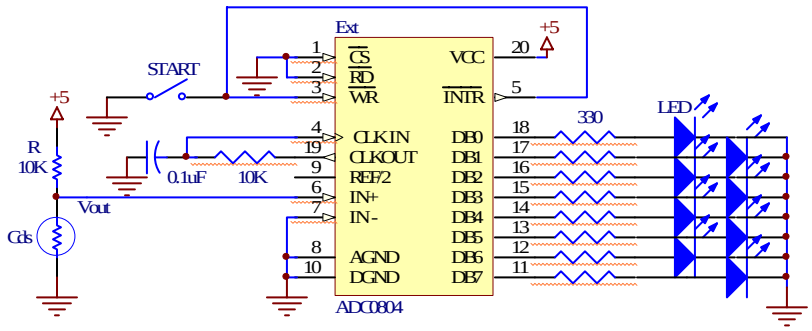
Figure 14 shows the balance sensor circuit diagram for the default mercury switch.

Figure 14. Balance Sensor Circuit Diagram



The CdS sensor's output signals generated by the solar light source are the input signals for the ADC chip's sixth pin. They are converted into analog signals and 8-bit output signals via the eleventh through eighteenth pin. Then, the signals are sent to the FPGA as input signals. Figure 15 shows the complete circuit diagram.

Figure 15. ADC Circuit Diagram



Design Methodology

Based on our experience, we know that if we make our solar panel perpendicular to sunlight, the illumination is strongest and electricity efficiency is highest. When we cannot adjust the actual position with accurate instruments, we measure, recognize, and determine the position with our eyes. Fuzziness might not be bad; sometimes we must accept information using a fuzzy method because we cannot know everything fully. General objects under control require a large number of complex mathematical formulas. To master controls with fuzzy features, scientists developed fuzzy theory. In application, the theory gives high importance to human experience and master degree for the properties of things, but does not advocate resolving problems with complex mathematical analysis and modes. In the following sections, we introduce fuzzy theory and describe how we use it in our Nios II control system.

Fuzzy Logic Control Design

Since professor L. A. Zadeh of the University of California at Berkeley proposed the concept of fuzzy sets in the academic journal, *Information and Control* in 1965, fuzzy theory has boomed. The theory emphasizes that most knowledge can be expressed with language, i.e., we can fuzzify all knowledge fields. Implementing the theory provides a wider application scope and error tolerance and is suitable for nonlinear systems in real life.

Early in 1974, professor E. H. Mamdani of Queen Mary, University of London, successfully applied fuzzy control to the automatic operation of a steamer. In 1980, a Danish company, F.L. Smidth, used fuzzy controls on a cement kiln. The corporation transformed the operation statuses of the cement kiln and their solutions into language-type control rules and controlled the cement kiln by computers. Additionally, the Sendai Municipal Subway of Japan, which was launched in July 1987, successfully used fuzzy controls to automatically operate trains. Some consumer electronics also use the theory.

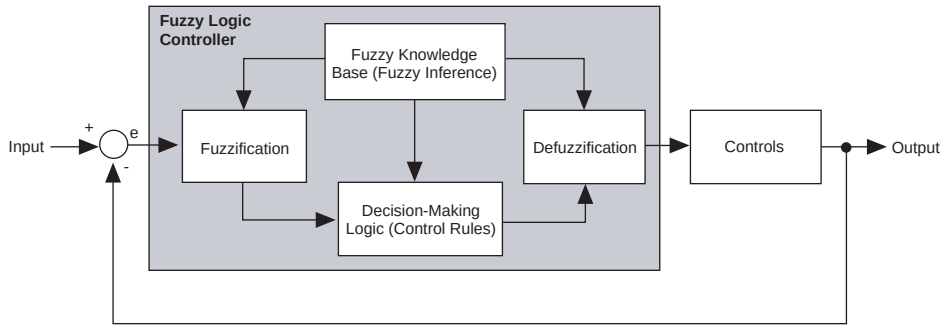
Fuzzy theory is a science closely related to our lives. Because it describes things with language, it is easy to accept. In real life, most descriptions are fuzzy. For example, when we say “sweet fruit” or “drive fast,” sweet and fast are not accurate values but simply a description of the degree. However, people can easily understand the meaning from the description. During nearly 40 years of development, achievements in fuzzy theory have been recognized and the application of the theory has extended to all scientific fields, including:

- *Control engineering*—Intelligent controls, vehicle electronics, Sendai Municipal subway, etc.
- *Image identification*—Image processing, voice identification, signal processing, etc.
- *Consumer electronics*—Washing machines, refrigerators, coolers, etc.
- *Other*—Data management, teaching appraisals, financial management, etc.

Fuzzy Logic Controller Structure

Figure 16 shows the fuzzy logic controller (FLC) structure. The FLC is composed of a fuzzification interface, knowledge base, inference engine, and defuzzification interface.

Figure 16. Fuzzy Logic Controller Basic Structure



Fuzzification Interface

The input of a common controller is a specific numeric value, but the knowledge base for fuzzy control is expressed with language. The system must turn numeric values into language and corresponding domains to allow the fuzzy inference engine to inference. This transformation is called fuzzification.

Knowledge Base

Knowledge base is the inference basis for fuzzy control. It defines all relevant language control rules and parameters. The knowledge base (including the database and rules base) is the core of a fuzzy control system.

Database

Fuzzy control rules have two types of presentations. The first is a state evaluation type that evaluates the system state at time (t) and calculates the fuzzy control action at certain point in time using the control rule and database input variables. The second is an object evaluation type that predicts current and future control actions. It determines whether the control object is achieved and then decides whether to output a control command. The database is usually built based on the following sources:

- *Working and expert experience*—The fuzzy control rule is based on information obtained by a controlled system. Experience rules are the most important part of fuzzy control.
- *System self-learning*—The system has an off-line learning method and builds a rules base with the help of other algorithms; however, this method requires a system model to be established already.
- *Predication evaluation*—This is a traditional method. It uses mass tests to verify the result and updates the rules base with the latest results. This method takes a long time.

Rules Base

The rules base contains many rules presented as language. The following rule is presented as a simple conditional statement:

Rt IF x is At THEN y is Bt (2)

where:

- t is the statement number.
- IF is the statement antecedent proposing the conditions to determine whether the statement is true.
- THEN is the statement consequence representing the inference result according to the conditions.

- Antecedent x is the input variable of the fuzzy system and is used to measure the system state.
- Consequent y is the output variable of the fuzzy system and is used to control the system.

The actual variable number can be increased or reduced according to the status of controlled system. A_t and B_t are the fuzzy concepts presented by language. For example, the definitions of tall, short, fast, and slow are different due to human subjective opinions and are difficult to present with data; instead they are defined using membership functions.

Fuzzy Inference Engine

As the most important part of fuzzy control, the fuzzy inference engine performs the actual decision-making process. The basic theory of the fuzzy inference engine is an approximate inference. The engine has two key inference methods: generalized modus pones (GMP) and generalized modus tollens (GMT). GMT is object-oriented inverse fuzzy theory, but GMP is forwarding linking inference modus. In GMP, when data is input, the output can be inferred according to rules; therefore, GMP is applicable for a fuzzy control inference mechanism. Its operation includes the following three calculations:

- Perform an AND operation for all the propositions of the antecedent of the triggered rule to obtain the antecedent fit.
- Perform an AND operation for all the propositions of the consequent corresponding to the antecedent fit of the triggered rule to determine how strongly true the rule is.
- Perform an OR operation for all consequents of all triggered rules.

Defuzzification

The reverse of fuzzification, defuzzification transforms the fuzzy inference engine's output values into equivalent assured values, making the assured value comply with the input signals of the controlled system. This process gives output control signals to the controlled system.

Solar Energy Controller Production

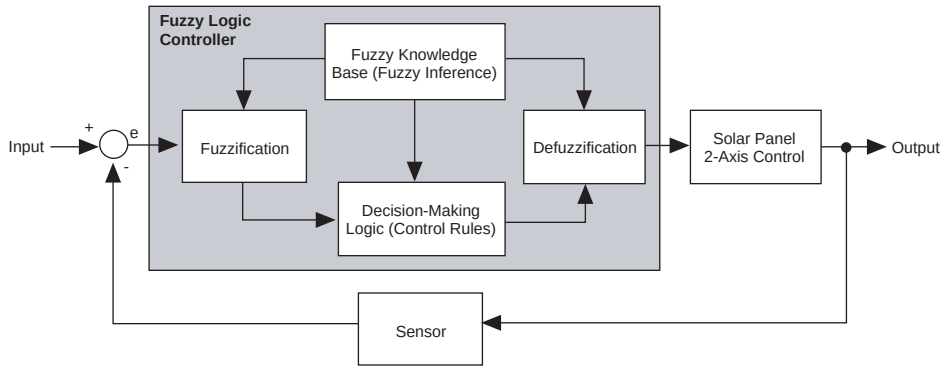
Although the solar tracking system's two drive motors can independently rotate without the problem of coupling, they inevitably have nonlinear phenomena in the moment of inertia (this is a common problem for 3-D rotation mechanisms). Therefore, the motors require a closed loop control. Although nonlinear phenomena exist in the moment of inertia control, it is not necessary for the solar tracking system to rotate very quickly due to the speed of the sun's movement. Therefore, we can use fuzzy control rules to control the motor operation while ensuring the system control mechanism's adjustability and fast response time.

We use fuzzy control theory as the control basis of motor driver. When implementing the hardware control circuit, we used a hardware description language such as VHDL and Verilog HDL to load the control program into the Nios II processor, which is the control center. Then, we created the sensor, decoder, and other devices to form a complete control loop, ensuring optimal electricity efficiency of the system.

Fuzzy Logic Controller Implementation

Our controller designed takes the measured value of the light strength received by the sensor as the feedback and implements control using many rounds of modifications. Figure 17 shows the basic fuzzy control system structure. The CdS sensor resistance changes with the light strength. It is converted by the ADC to obtain a partial pressure voltage. Fuzzy control takes the errors of the two groups in the vertical (southern and northern) and horizontal (eastern and western) axis as the fuzzy control input.

Figure 17. Solar Energy Fuzzy Control System Structure



The whole fuzzy controller can be designed in five steps.

Step 1: Definitions

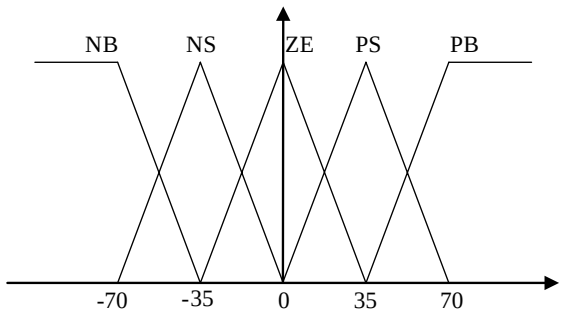
We define the input variables, output variables, and linguistic variables. Linguistic variables include the choice of input and output variables. In this study, for an axis we choose *e* as the input variable and *e* as the output variable, and define five triangle membership functions for one chosen linguistic variable.

- The input variable is Error e = b_pio - a_pio (3)
- The output variable is the number of seconds between the rotation and reversion adjustment of the step motor.
- For the domain of the output and input variables we used normalized discrete domain to set domain scope as [-75, 75].
- For the linguistic items, we defined five linguistic items for each linguistic variable, i.e., *e* ={NB, NS, ZE, PS, PB}. The linguistic items are defined as:
  - NB: Negative Big
  - NS: Negative Small
  - ZE: Zero
  - PS: Positive Small
  - PB: Positive Big

Step 2: Build Membership Input Functions

Based on the defined linguistic variables, we built the input membership functions. Figure 18 shows the membership function of errors in a horizontal or vertical orientation.

Figure 18. Fuzzy Control System Membership Function



Step 3: Set Up Fuzzy Rules Base

The setup of the fuzzy rules base is crucial because all states must be operated based on the rules defined in the rules base. To set up a fuzzy rules base smoothly, we adopted five fuzzy control rules expressed with IF THEN statements.

- *Rule 1*—If e is PB, then U_f is PB.
- *Rule 2*—If e is PS, then U_f is PS.
- *Rule 3*—If e is ZE, then U_f is ZE.
- *Rule 4*—If e is NB, then U_f is NB.
- *Rule 5*—If e is NS, then U_f is NS.

Step 4: Define Fuzzy Inference Engine

There are many fuzzy inference methods and different results are inferred with different methods. In this project we use the center of gravity method proposed by Mamdani as the defuzzification tool because the method is easy and reliable.

Step 5: Defuzzify Result

We defuzzify the result inferred by the fuzzy inference engine to convert the information into precise numbers. There are many methods for defuzzification. In this project we use center of gravity for defuzzification to obtain actual operation. The formula is shown in equation 4.

If the inference result is a fuzzy single value, the weighted mean method is most widely applicable. For n rules, w_i is the initiating strength of number i fuzzy rule, r_i is the inference result of number i fuzzy rule, and $\hat{U}_f$ is the output operation, the formula is:

$$\hat{U}_f = \frac{\sum_{i=1}^{n} w_i r_i}{\sum_{i=1}^{n} w_i} \tag{4}$$

Here we use the five fuzzy rules as follows:

$$\hat{U}_f = \frac{\sum_{i=1}^{5} w_i r_i}{\sum_{i=1}^{5} w_i} = w_1 r_1 + w_2 r_2 + w_3 r_3 + w_4 r_4 + w_5 r_5 \quad (5)$$

This defuzzification method can also be easily implemented with a digital circuit.

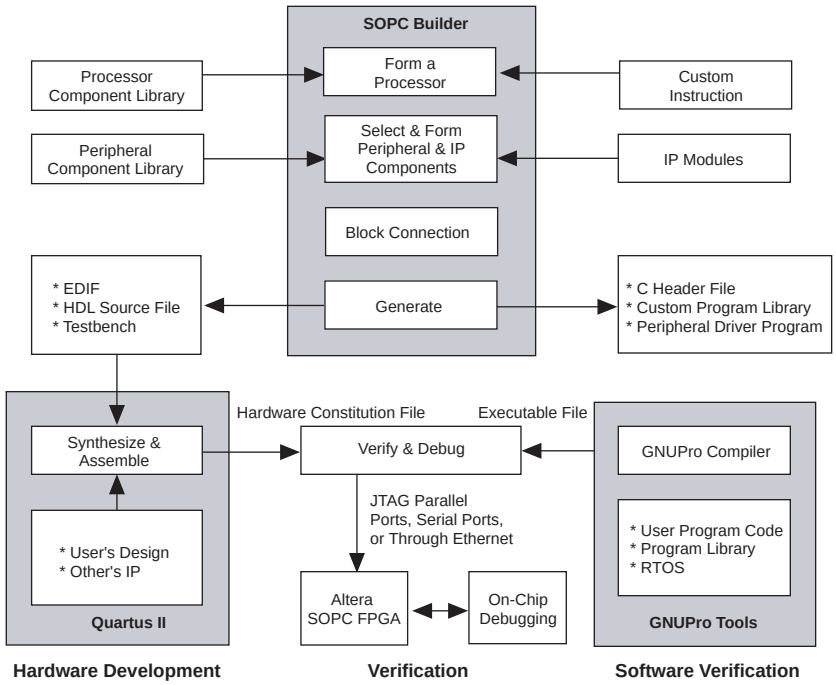
FPGA Program Design

As FPGAs have evolved in recent years, a single FPGA can accommodate more and more logic circuits. Building of a whole digital electronic circuit on a single chip provides a big edge in speed and power consumption, making system-on-a-programmable-chip (SOPC) designs gradually become the design tendency. As an emerging systematic design technology, SOPC design can incorporate the hardware system (including processor, memory, peripheral interface circuit, and user logic circuit) and software design on a single programmable chip.

SOPC System Design

Figure 19 shows the SOPC system development flow. First we define the system, including the processor, memory interface, peripherals, arbitrator, custom instructions, etc. Next, we generate the system with SOPC Builder. Then we perform the hardware and software design. During hardware design, we used the Quartus® II software to compile the logic circuit of the HDL programs and EDIF files. During software design, we used the GNUPro software development tool and software resources such as header files, library, monitors, and peripheral drivers to generate and edit application code. We debugged programs using Debug/Profile. To ensure the correctness of the hardware and software design, we used the ModelSim software for simulation. When we found an error, we went back to system generation so that SOPC Builder can modify and generate the system until it is right. Finally, we downloaded the hardware and software design into development board and prototyping kit for circuit verification.

Figure 19. SOPC System Development Flow



Hardware Design Solution

With Altera FPGAs, we could use the soft-core Nios II processor or the ARM ARM922T hard core. Altera provides the soft-core Nios II processor, which is a 16- or 32-bit RISC-based configurable embedded processor. For peripherals, Altera provides on-chip ROM, on-chip RAM, memory interfaces (such as SDRAM, SSRAM, and DMA controllers), series I/O (such as UART and Ethernet), parallel I/O (such as an input/output/two-way port, or PCI interface) as well as timers (such as a simple timer, frequency timer, and watchdog timer). For intellectual property (IP), Altera provides a PCI 32/33 bridge and Ethernet MAC. For the bus, Altera provides the Avalon® bus.

For the EDA hardware development tool, we used Altera's Quartus II software for place and route, design entry, compilation, and programming of the FPGA. The Quartus II software provides design entry methods such as VHDL, Verilog HDL, Altera Hardware Description Language (AHDL), and block/schematic entry. We used the LeonardoSpectrum software for circuit synthesis and the ModelSim® software for system simulation.

Software Design Solution

Altera provides the following relevant software development tools:

- Compiler
- Assembler
- Linker
- Debugger
- Monitor

- Libraries
- Utilities

Nios II Embedded Processor

Embedded systems were first designed for industrial computers. With the boom of information products and digital consumer electronics (CE), embedded systems became more popular. Embedded systems are used in a variety of applications, from information products, CE, and network products to portable devices. Because the ARM processor does not include FPGA-based IP, Altera embeds the ARM922T microprocessor hardware circuit into the FPGA to improve the system design integrity. In this project, integrating an FPGA control chip, it is convenient and feasible to use the Nios II processor to develop an SOPC embedded system. Using the SOPC Builder Integrated Development Environment (IDE) of Altera as an example, we can plan CPU types using the ARM core and Nios II IP (embedded into the CPU as VHDL or Verilog HDL) as the priorities. The main advantage of the SOPC Builder IDE is that it integrates and records the required circuits and peripherals to an FPGA to create a small circuit and maintainable hardware and software while accelerating product development and keeping a scalable design.

In this project we used the Altera Development and Education (DE1) board and implemented the Nios II embedded processor in the Cyclone II EP2C20F484C7 FPGA on the board. Figure 20 and Table 2 show the development board specifications. The resulting processor is a low-cost, high-performance FPGA and its system performance can be configured by customers as required, including:

- Three types of Nios II processors: fast (Nios II/f), standard (Nios II/s), and economical (Nios II/e). They are 32-bit instruction set structural systems.
- Complete peripheral hardware settings such as a timer, bridge, and counter, which SOPC Builder uses to integrate a complete microprocessor structure.
- Avalon switch structure, which can simultaneously process multiple units to improve system bandwidth with minimum FPGA resources.

Figure 20. DE1 Development Board

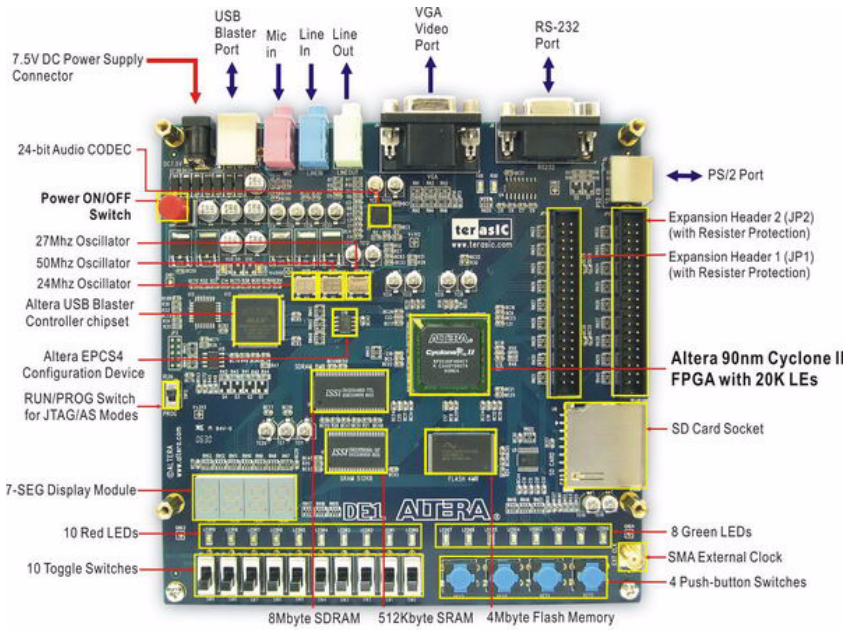


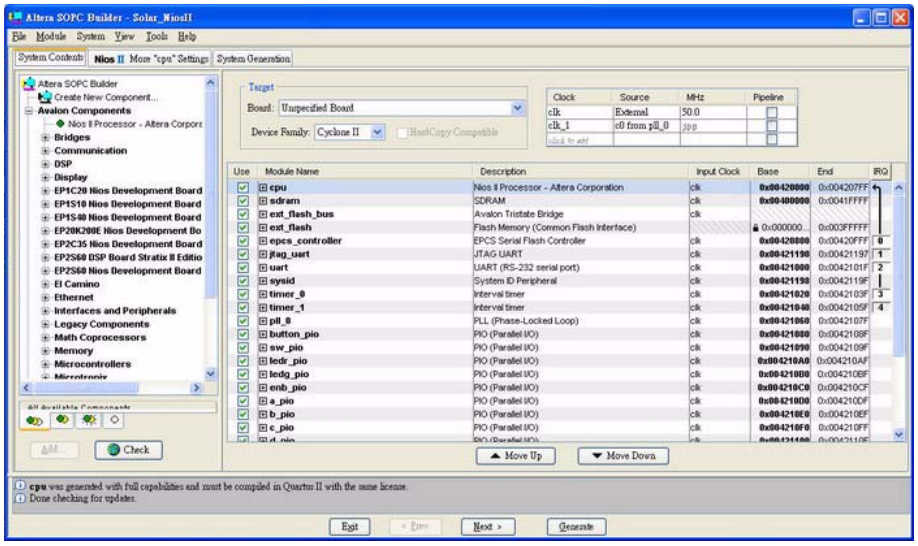
Table 2. DE1 Development Board Specification

Specification	Value
Total logic elements (LEs)	18,752
M4K RAM blocks	52
Total RAM bits	239,616
Embedded multipliers	26
PLLs	4
Maximum user I/O pins	315
FineLine BGA package	484

To design a Nios II system, we need to design the software and hardware. We used the Quartus II software and SOPC Builder to design the hardware. We used the Block Editor to generate the upper Block Design File (**.bdf**) and used the Text Editor to create VHDL or Verilog HDL hardware files. We also used the embedded MegaWizard® Plug-in Manager to generate low-order VHDL design files.

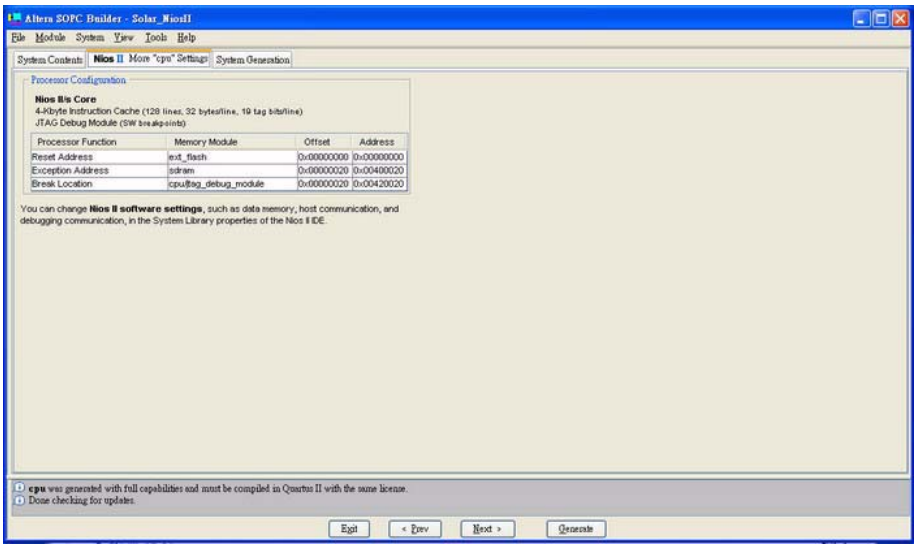
In the hardware design, we used SOPC Builder to establish Nios II system modules including the CPU, memory, and peripheral circuits. We selected the needed modules and set their parameters to designate the base address, interrupt request (IRQ), and system frequency as shown in Figure 21.

Figure 21. SOPC Builder System Content



Next, we designated the locations of the memory, boot chip, and interrupt vector table for the design and other system module settings as shown in Figure 22.

Figure 22. Nios II System Settings



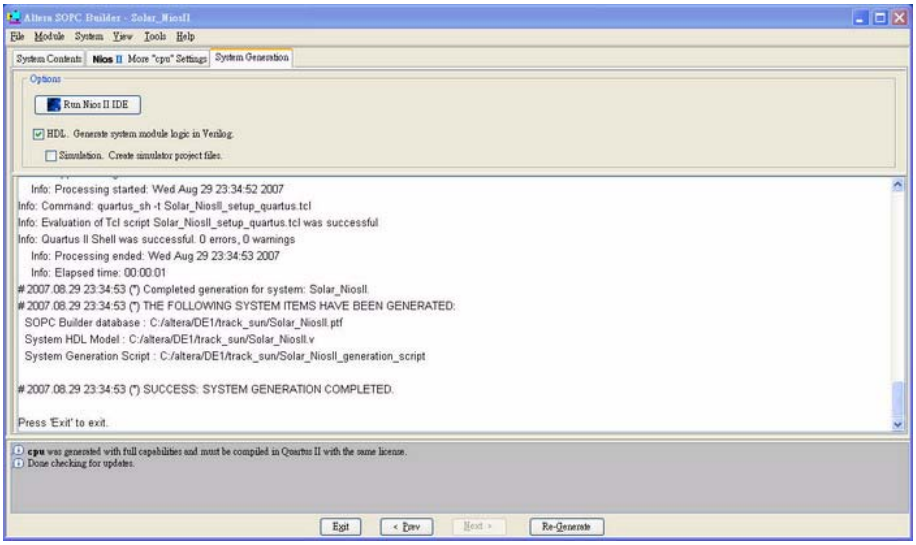
After completing the system design, we turned on the HDL option and turned on the chip model to be configured if the ModelSim simulation software has been installed. Then we clicked **Generate** to begin generation. This step performs the following actions:

- Generates the HDL file.
- Generates the simulation items and source files.

- Generates the C and Assembly language headers and source files.
- Compiles the system library.

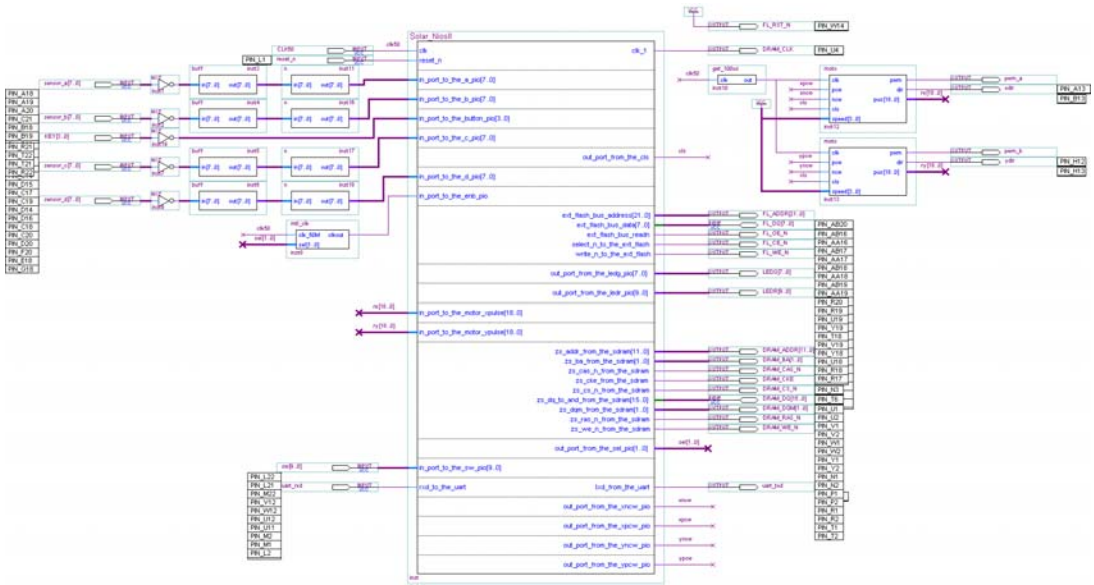
When system generation completes, we clicked **Exit** to go back to the Symbol Editor. See Figure 23.

Figure 23. SOPC Builder System Generation



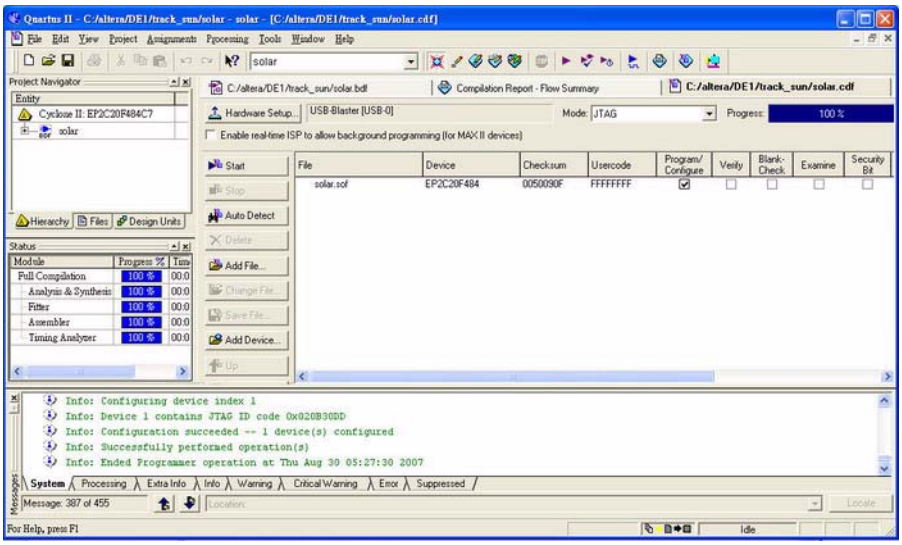
SOPC Builder generates a symbol of system module. Then, we added the circuit symbol into the BDF. See Figure 24. We added the input, output, and bidirectional pins, and then named each pin and other basic device symbols.

Figure 24. Quartus II BDF



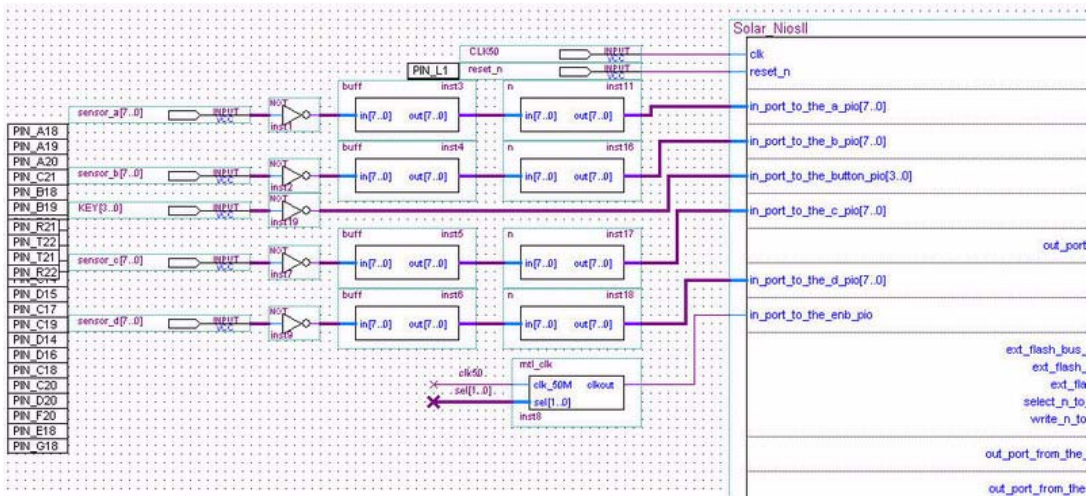
Finally, we downloaded the designed circuit to the development board using the Quartus II Programmer as shown in Figure 25.

Figure 25. Quartus II Programmer



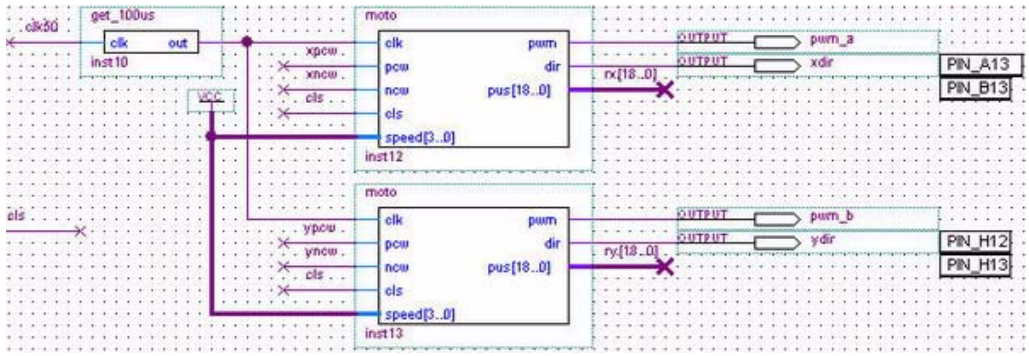
We input the decoding circuit for the CdS sensor using Verilog HDL. The signal is sent to the Nios II CPU, which controls it. See Figure 26.

Figure 26. Input Decoder Circuit using the CdS Sensor



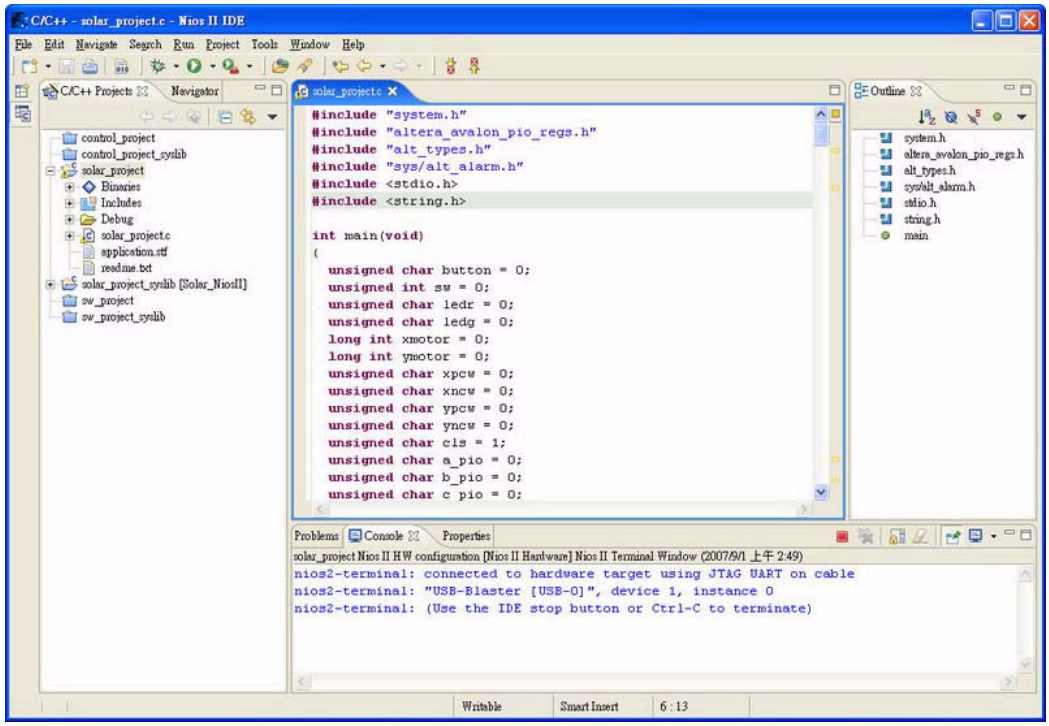
After determining the angle that the solar panels should reach, the Nios II CPU gives a signal to the motor to drive it. As shown in Figure 27, there are two motor modules controlling the system's X and Y axis.

Figure 27. Step Motor Module



After the required system hardware is generated, we can implement the controller using hardware or software. We used the Nios II 32-bit CPU to accelerate our fuzzy control rules. The Nios II IDE is shown in Figure 28.

Figure 28. Nios II IDE



Design Features

The Nios II processor helped us implement the design in the following ways:

- The major difference between our design and traditional, single-chip designs (such as the 8051 or PIC device) is that we added a fuzzy control rule to the circuit. Traditional chips cannot write VHDL. If we used traditional devices, we might need external logic circuits to implement the fuzzy controller, increasing the controller design volume and cost burdens. Alternatively, if we

used a microprocessor (MCU) plus an FPGA (excluding the Nios II processor), we would need to use a two-device assembly.

- For complex logic circuits, we can create a design using an FPGA and the Nios II processor with the Nios II IDE. The advantages of this method are that we can use the C language to write fuzzy algorithms and incorporate them into the Nios II CPU and we can compile the VHDL code into the FPGA to control the step motor. This implementation allows us to process algorithm operations and I/O control in parallel, improve integrated efficiency, and quickly implement and verify our hardware circuits.

Finally, we took the solar tracking control system for an outdoor test. For each 24-hour day, we used the step motor, FPGA development board, and control and sensing circuit to track the sunlight for about 30 seconds/hour. In one day, the solar panel is charged for about 8 hours, and in for the rest of the time it does not consume power (i.e., there is no standby mode to consume power). Therefore, we can calculate the energy data as shown in Table 3.

Table 3. Collected 24-Hour Solar Energy Radiation (Cloudy)

Test Method Measured Data	Fixed Solar Current Collection System	Smart Solar Current Collection System
Average 24-hour accumulated electricity generation of the solar panel P_S (J).	276,480	345,600
24-hour accumulated current consumed by step motor and control and sensing circuits P_C (J).	0	9,216
Average 24-hour accumulated net electricity generation $P_{total} = P_S - P_C$ (J).	275,480	336,384

Comparing the total net electricity generation of the fixed elevation angle control and smart solar tracking control, we found that smart system is superior to the fixed system.

Outdoor Measurement

In the study, we moved the solar platform to the top of the school building to test the fixed and smart systems (the results are shown in Table 3). Outdoor fixed and smart solar current collection system simulations are shown in Figures 29 and 30, respectively.

Figure 29. Outdoor Fixed Solar Current Collection System Test

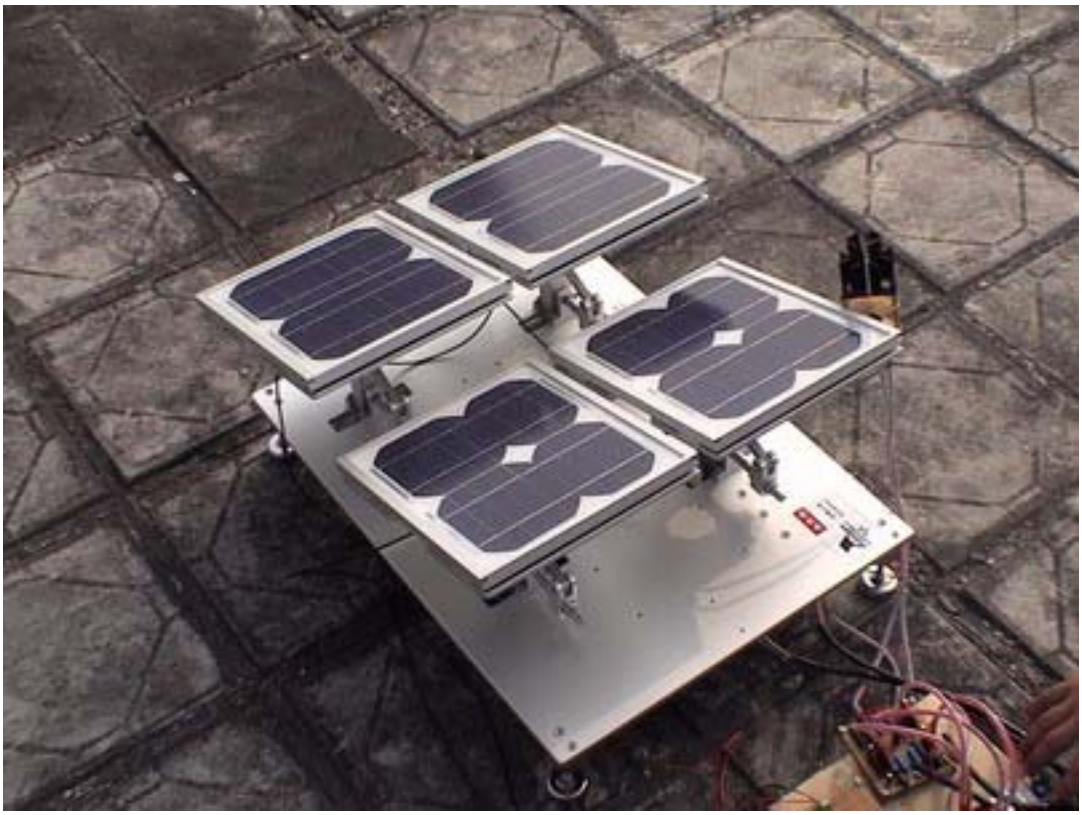
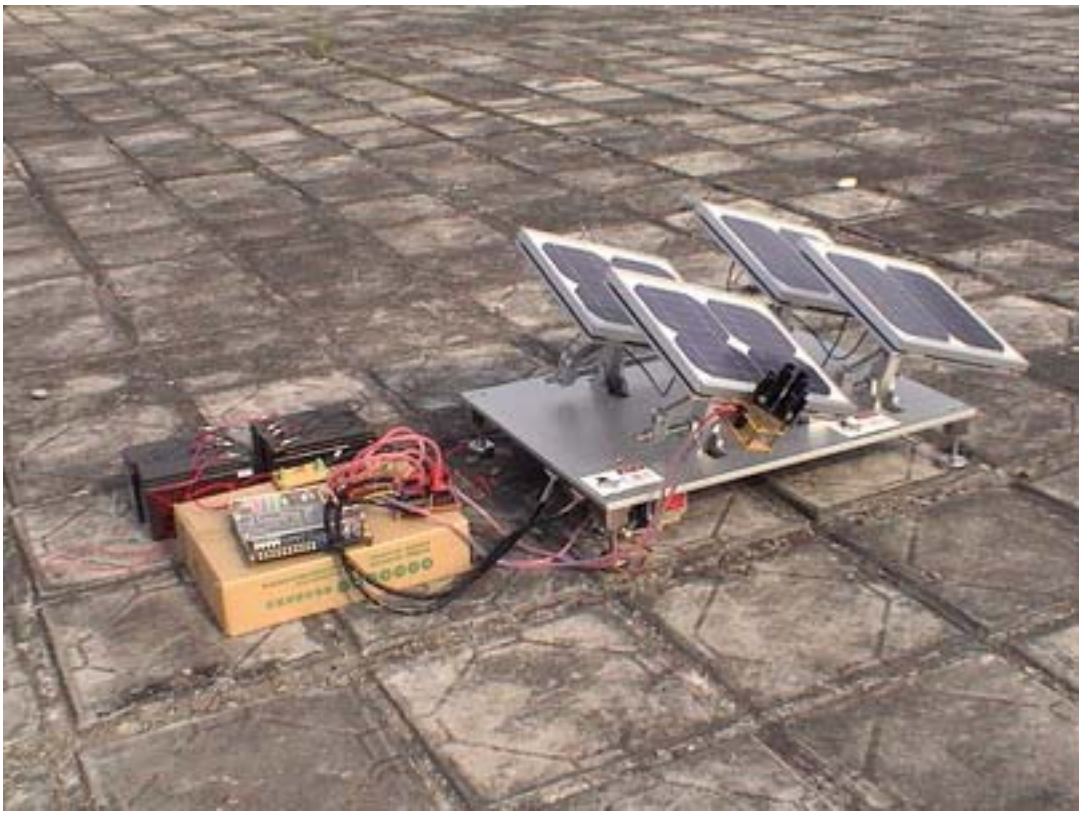


Figure 30. Outdoor Smart Solar Current Collection System Test



Indoor Measurement with Searchlight Simulating Sunlight

In this test, we used a searchlight as a simulated sunlight source, established a fixed and smart simulated sun running orbit, and used Visual Basic (VB) to transmit the measured voltage to notebook (NB) computer to measure the actual voltage through the RS-232 port. Figures 31 and 33 show the fixed and intellectual solar current collection system simulation, respectively. They show the different electricity generation efficiencies at the same angle. Figures 32 and 34 show the indoor fixed and smart solar current collection system voltages, respectively.

Figure 31. Indoor Fixed Solar Current Collection System Test

Figure 32. Indoor Fixed Solar Current Collection System Voltage

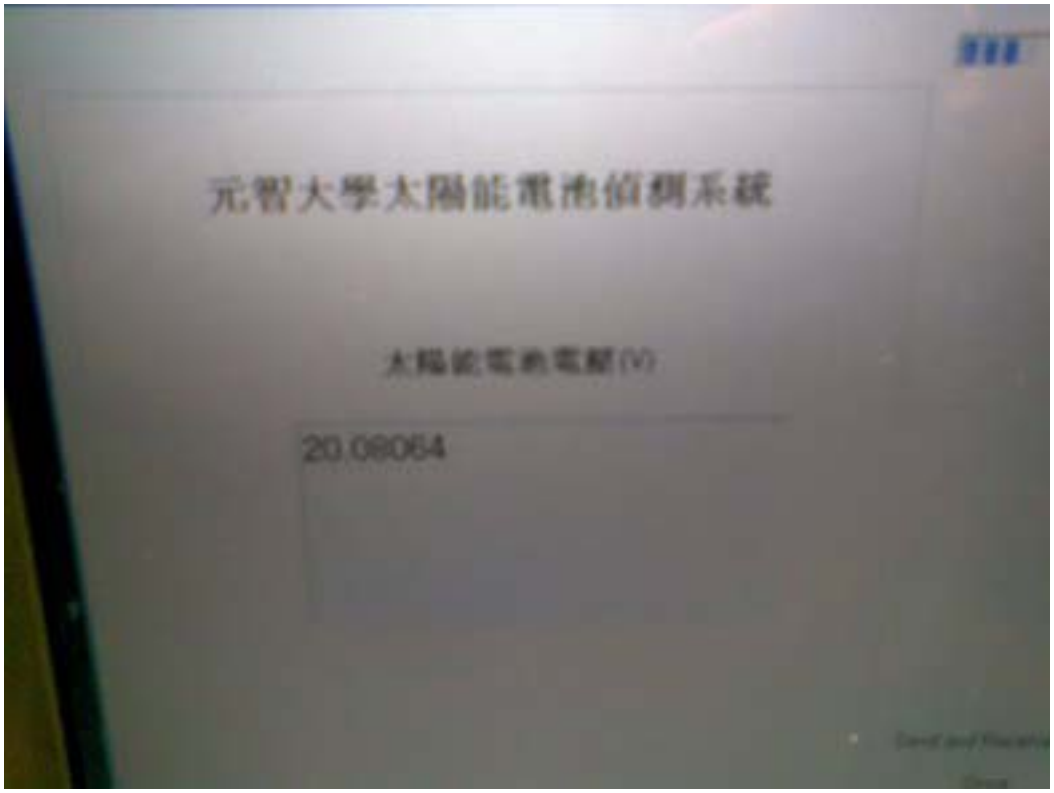


Figure 33. Indoor Smart Solar Current Collection System Test

Figure 34. Indoor Smart Solar Current Collection System Voltage

From Figures 32 and 34, we can see the voltage of the fixed solar current collection system is less than that of the smart solar current system. Therefore, the smart system is superior to the fixed system.

From the experimental results, we reached the following conclusions:

- By using the CdS sensor, the step motor, and Nios II processor in the FPGA to write and create different sunlight processing solutions (such as cloud intervention processing and reset solutions), we could establish a smart elevation angle control system to track a solar light source and improve the electricity generation efficiency of the solar batteries.
- The data in Table 3 shows that the smart elevation control system is a solar battery electricity generation system deserving wide promotion.
- Used with an FPGA, Altera's Nios II processor simplifies research and development and product testing and reduces circuit board costs, making it far superior to commonly used single devices.

Conclusion

We had used Altera FPGAs before we participated in the contest, so we had heard about the first-generation Nios processor. However, we only knew the Nios II processor from Altera's web site and collateral. This Nios II design contest gave us a good opportunity to enhance our ability in Nios II processor design and helped us thoroughly understand the Nios II processor's brand-new design concepts and features.

In this design, we used Altera's SOPC-based FPGAs. SOPC design represents a new system design technology, and SOPC Builder and the Nios II processor helped us see the powerful design technologies of software and hardware systems. Most traditional circuit designs are composed of hardware components building on a printed circuit board (PCB). If errors are found or the system needs to be improved or upgraded, the PCB must be redesigned. Adjusting and modifying the PCB is very inconvenient and increased the design cost and development period. In this contest, we accomplished our goals before the deadline. From concept design to system implementation, we only needed to model on the PC because Altera provides complete tools—including SOPC Builder, the Nios II IDE, and Quartus II development environment—to accelerate the software and hardware development. The Nios II processor is greatly improved over the Nios processor, and is more efficient, compact, and stable. Finally, integrating Nios II development, test environment, and C language compiler provides great convenience for users.

Third Prize

Nios II Processor-Based Fingerprint Identification System

Institution: College of Communication Engineering, Chongqing University

Participants: Ji Wang, Liang Wu, Yong Liu

Instructor: He Wei

Design Introduction

With the boom of information technology represented by computers since the 1960s, computer technology has begun to be used in the fingerprint identification field, bringing new thoughts, implementation methods, and processing approaches for automated fingerprint identification. Authorities, institutions, and universities have begun implementing fingerprint analysis and processing using computers. A computerized system that performs automated fingerprint identification is called an Automated Fingerprint Identification System (AFIS).

People often need to be identified in society. The common ID authentication methods such as keys, password, certificates, and IC cards provide identification using objects, which indirectly identify the object holder. These objects are not very accurate and have significant security risks, including counterfeited certificates and tokens, and decrypted or stolen passwords. With the development of image processing and pattern identification technologies, emerging identification technology based on biometric characteristics has become the focus of research and applications due to its unique reliability, stability, and convenience. As the earliest and most mature biometric identification technology in the pattern identification field, fingerprint identification technology integrates sensors, biometric technology, electronic technology, digital image processing, and pattern identification. Many automatic fingerprint identification systems are used worldwide, but fingerprint identification technology is not yet mature. China is behind in fingerprint collection and algorithm study, so the research of fingerprint identification algorithms and systems will play a significant role in theory and practice.

Systems with fingerprint minutia identification have wide applications in the fields of security, jurisdiction, military, finance and economy, information service, etc. Identification is needed in access control systems and other similar applications. Embedded fingerprint minutia identification systems

fully utilize biometric minutiae and are applicable for determining attendance in schools and enterprises, identifying residents in residential quarters, and so on.

The study and practice of police AFIS for two or three decades has laid a good foundation for civil AFIS. Specifically, the existing civil AFIS is easy to use, accurate, reliable, and affordable, allowing fingerprint identification products to enter our daily life. By replacing a personal identification code and password, fingerprint identification technology can guard against unauthorized access and illegal use of ATMs, cell phones, intelligent cards, desktop PCs, work stations, and computer networks. It can facilitate identification in telephone- or Internet-based financial deals and replace keys, certificates, stamps, and IC card readers in buildings or worksites.

As microelectronics technology advances, programmable logical controllers are becoming more diversified, faster, and more powerful. Today, many FPGAs support embedded soft-core processors to facilitate FPGA-based hardware development. For example, the Altera® Nios® II RISC CPU soft-core processor features pipelining and single instruction flow, can be embedded in an FPGA, and can leverage custom logic to build a FPGA-based on-chip system. Compared to an embedded hard core, a soft core is more flexible. Additionally, the faster FPGA exactly meets the fingerprint identification system's speed requirements.

We chose to use the Nios II soft-core processor in our design for the following reasons:

- The Nios II soft-core processor can cut costs through large-scale system integration and FPGA/CPU optimization.
- The Nios II processor is more flexible, provides shorter design cycles, and can prolong the product lifecycle with upgrades.
- Custom instructions and logic can accelerate complex arithmetic operations and logic.
- The Nios II C-to-Hardware Acceleration (C2H) Compiler allows designs to operate more than 40 times faster than software that is not accelerated.

Function Description

We designed the fingerprint identification system based on Altera's Nios II processor and FPGAs. This system can collect real-time fingerprint image signals, extract finger minutiae, and match minutiae in a database to perform fingerprint identification. The whole system design includes fingerprint image collection, fingerprint image preprocessing, minutia extraction, minutia matching, and a database.

Fingerprint Image Signal Collection

The fingerprint collector serves as the fingerprint collection module. The module's fingerprint sensor is Veridicom's third-generation product, the FPS200 sensor (with 256 x 300 array numbers and 500-DPI resolution). The sensor uses Veridicom's ImageSeek function and high-speed image transmission technology to obtain quality images of all fingerprint types. The fingerprint collector design performs the following functions:

- *Capture stage*—Each column of the capacitor array is connected to two sampling/hold circuits to capture one row of fingerprint images. The capture procedure has two stages: capacitors in the chosen row are charged to U_{DD} and store the charging voltage in the first sampling/hold circuit; next, the capacitors discharge to the current source at a speed proportional to the discharged current. After a discharge period, the second sampling/hold circuit stores the capacitors' final discharge voltage. By measuring the difference (ΔU) between the charging and discharging voltages, we obtain the capacity (C) of each capacitor unit.

- *Analog-to-digital (A/D) conversion*—The analog signals representing the unit capacities of the row go through A/D conversion to generate the digital fingerprint image information for the row. The system puts the information in a register and captures the fingerprint images of the next row.

- *Fingerprint signal transmission*—FPS200 contains three bus interface circuits: USB, serial peripheral interface (SPI), and microcontroller (MCU) interfaces. This design uses the SPI interface to transmit the digital information in the registers to the Development and Education (DE2) board's SDRAM.

Fingerprint Image Preprocessing

During fingerprint image preprocessing, the fingerprint image is enhanced. Accurate fingerprint identification relies on the identification of the fingerprint ridge texture and minutiae. However, due to skin condition, collection conditions, devices, the working and living environment of the fingerprinted person, etc., the raw fingerprint images collected by the fingerprint sensor usually contain noise and degrade dramatically. Therefore, the raw fingerprint images must be preprocessed after being collected. Fingerprint image preprocessing procedures include image normalization, orientation and frequency extraction, filtration, binarization, ridge thinning, etc.

Normalization

Image normalization reduces the diversification degree of the grayscale along the ridges and valleys without changing the raw image's structure or texture information. This process gives the image preset means and variances, and facilitates pattern capture and fingerprint frequency. Nevertheless, normalization can also enhance some hash in the image background. Equation 2.1 is for normalization and equation 2.2 is an improved version based on equation 2.1 for the convenience of hardware implementation.

$$G(i,j) = \begin{cases} M_0 + \sqrt{\dfrac{VAR_0(I(i,j) - M)^2}{VAR}} & I(i,j) > M \\ M_0 - \sqrt{\dfrac{VAR_0(I(i,j) - M)^2}{VAR}} & \text{Others} \end{cases} \quad (2.1)$$

$$G(i,j) = \begin{cases} M_0 + \lambda \times VAR \times |I(i,j) - M| & I(i,j) > M \\ M_0 - \lambda \times VAR \times |I(i,j) - M| & \text{Others} \end{cases} \quad (2.2)$$

Fingerprint Orientation Extraction

The basic concept for pattern extraction is to:

- Calculate certain statistics (like grayscale difference and gradients) of each point (or block) in all orientations of the raw fingerprint grayscale image.

- Decide the orientation of the point (block) according to the difference of these statistics in all orientations to obtain the fingerprint pattern.

The algorithm process is as follows:

1. Suppose f(i,j) is the grayscale value of fingerprint pixel point (i,j).

2. Divide the image into W x W-sized sub-blocks without overlapping. It is better for W to contain the size of a ridge and a valley. Here we use 10 for W.

3. Use the Sobel operator to compute the gradients of image's pixel points (i,j) in X and Y orientations, $G_X(u,v)$ and $G_Y(u,v)$. Figure 1 shows the Sobel operator template coefficient.

Figure 1. Sobel Operator Template Coefficient

X-Axis Orientation

1	2	1
0	0	0
-1	-2	-1

Y-Axis Orientation

1	0	-1
2	0	-2
1	0	-1

4. Calculate the partial orientation Θ(I,j) of the W x W sub-block focus on (I,j).

5. Smooth the resulting fingerprint pattern.

Fingerprint Frequency Extraction

In partial non-singular areas of the fingerprint image, the changing pixel point values of the ridges and valleys constitute an approximate two-dimensional (2-D) sine wave along the orientation vertical to the ridges (i.e., the gradient orientation of the ridges). The fingerprint frequency algorithm procedures are as follows:

1. Divide the normalized fingerprint image into W x W-sized sub-blocks without overlapping. Here we use 16 for W.

2. For each image sub-block focusing on (i,j), use the fingerprint orientation Θ(i,j) of (i,j) for the minor axis and draw a rectangular orientation window of L x W (32 x 16).

3. For each rectangular orientation window, calculate the grayscale discrete signals X[0], X[0],…X [L - 1] vertical to Θ(i,j) (i.e., the gradient orientation of the fingerprint) according to the following method: because X[k] constitutes an approximate 2-D sine wave, calculate the mean distance of the sine crest L_0 to obtain sine wave frequency $f = 1/L_0$.

Generally the frequency scope of 500-DPI fingerprint images is [1/25, 1/3].

Valid Mask Extraction

Valid masks are an important part of fingerprint image preprocessing. Valid mask extraction distinguishes the foreground of the finger image from its background. If we do not extract the valid image masks, we will waste time processing background areas and obtain many false minutiae. Therefore, it is necessary to extract valid masks of the fingerprint to eliminate the background and reduce false minutiae points. A valid mask in the fingerprint image sub-block must satisfy the following conditions:

1. The difference between the maximum crest (G_{MAX}) and minimum trough (G_{MIN}) in the 2-D sine wave along the fingerprint gradient orientation of the sub-block's central point should be more than the threshold (*T*).

2. The frequency of the sub-block's 2-D sine wave should be within a certain scope. Here we use [1/25, 1/3].

3. The amplitude variance of the sub-block's 2-D sine wave should be more than the threshold (*D*).

4. In the sub-blocks, the fingerprint grayscale along the fingerprint orientation and along the orientation vertical to the fingerprint orientation should have distinct differences.

Gabor Directional Filtration

To greatly improve the fingerprint image quality, our design uses an orientation filter enhancement method. The Gabor filter formula is:

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_x}\exp\left(-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_x^2}\right)\right)\exp(-2\pi ifx)$$

Removing the odd component simplifies the formula. Turn to the fingerprint orientation. The design takes each pixel point in the fingerprint image as the central point and W x W as the window to obtain the Gabor enhancement coefficient of each pixel point in the block along the fingerprint's texture.

Binarization and Noise Reduction

The binarization process generates a monochrome fingerprint image from a grayscale image. To implement binarization, we use an adaptive, local threshold scheme, i.e., we adjust the threshold by the global grayscale of a block in the image. The procedures is:

1. Divide the fingerprint image in the valid mask into W x W blocks (we use 8 for W in this design).
2. Calculate the grayscale mean of each sub-block.
3. Take the grayscale mean as the threshold to implement binarization in the sub-block.
4. The binarization process may introduce noise; therefore, after binarization the fingerprint image should be filtered and have noise removed to delete the holes, notches, and other salient features caused by binarization.

Thinning

Ridge thinning transforms distinct but diversely sized binary fingerprint images into a single-pixel central point and thread image. Our design uses the Hilditch algorithm for this operaton. With this method, the whole image must be scanned several times. During each scan, pixel points satisfying given conditions are marked. After scanning, the marked pixel points are deleted and the next scan begins. When no pixel points are marked during scanning, the ridge thinning process finishes.

This functional algorithm has been emulated successfully using the C language. We used different schemes to implement modules according to their operation time. If we directly implemented the normalization, frequency and orientation extraction, and orientation filtration of the fingerprint image in the Nios II processor, the operation time would be unacceptable. Therefore, our system accelerates the hardware logic and instructions with the C2H Compiler. Because binarization and image thinning consume less time, we can implement them directly in the Nios II processor. Additionally, the whole algorithm involves many multiplication, evolution, rotation, and floating-point operations, which greatly slow the processing speed. Therefore, our design uses custom instructions in the Nios II processor to add the custom functions directly into the arithmetic logic units of the Nios II CPU, implementing these complex, time-consuming operations in hardware.

Fingerprint Minutia Extraction

Minutiae extraction involves preprocessing the image to obtain a quality image, and then finding and specifying the minutiae. After a raw fingerprint image goes through orientation filtration, binarization, and thinning, it becomes a thinned image. We then determine the endpoint and bifurcation according to the crossing of each point on the thinned image and extract the useful information of the two minutia points, such as coordinate position, type, and orientation. Fingerprint minutiae fall into many types. From the perspective of probability, 2-bifurcation and ending are the most common.

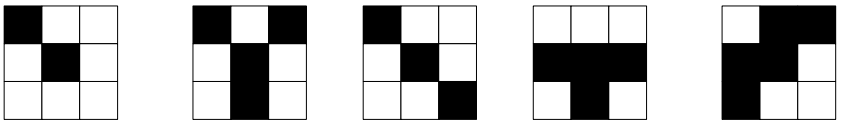
Minutia Point Extraction

For thinned images, the pixel point grayscale value can only be 0 or 1. We set 0 as the background point grayscale and 1 as the foreground point grayscale of the ridge. The crossing number C_N of eight fields of any point P on the thinned image (see Figure 2) is defined as:

$$C_N = \frac{1}{2} \sum_{k=1}^{8} |P_{k+1} - P_k| \text{, here } P_9 = P_1.$$

If $C_N(P) = 1$, point P is an endpoint. If $C_N(P) = 3$, point P is bifurcation. Otherwise, point P is a consecutive point or an isolated point and cannot be calculated into the minutia set.

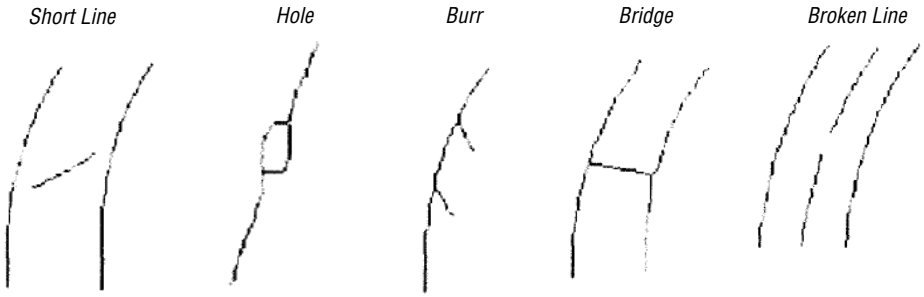
Figure 2. Minutiae Point Block Diagram



False Minutiae Removal

The thinned fingerprint images that are not refined include the false minutiae shown in Figure 3.

Figure 3. Basic False Minutia Structure Types



We remove these different false minutia points using different algorithms:

1. Remove the edge effect of the image.
2. Delete the false minutia point caused by the obscure part of the image.
3. Delete the false minutia point caused by broken ridges.
4. Delete the false minutiae point caused by holes.
5. Delete the false minutiae point caused by burrs.
6. Delete the false minutiae point caused by bridges.

These algorithms have been emulated successfully using the C language. Because the algorithms are comparatively easy, do not have time-consuming operations, and the processing speed meets real-time requirements, we implemented them directly in the Nios II processor.

Minutia Matching

Fingerprint minutia matching is the core of the fingerprint identification algorithm and is an important research subject. The accuracy and speed of minutia matching have a huge influence on the whole algorithm. Minutia matching matches the extracted minutia information set of two given fingerprint images to determine whether they are identical. In our design, we:

- Use a typical point pattern matching algorithm based on a minutia point coordinate mode.
- Utilize the triangle structure composed of three neighboring minutia points to locate a datum mark and to find conversion parameters.
- Conduct matching in the polar coordinate system after coordinate translation.
- Introduce multiple judgement conditions and a variable limit-box matching algorithm to improve the identification rate.

We implemented the algorithm as follows:

1. For any minutia point Q_i in input point set Q and any minutia point P_j in template point set P, search the two minutia points (Q_{i1},Q_{i2}) and (P_{j1},P_{j2}) that are nearest to Q_i and P_j, respectively, in Q and P. Thus, points (P_j,P_{j1},P_{j2}) and (Q_i,Q_{i1},Q_{i2}) form two triangles $\Delta P_j,P_{j1},P_{j2}$ and $\Delta Q_i,Q_{i1},Q_{i2}$.

2. Determine whether the two triangles are identical and have the same rotation orientation. If they do, take the point as a polar point to create polar coordinates, and then match the minutia points using a variable, limit-box method.

3. We believe there is a match only if the two conditions are met:

$$(N_m \geq T_m),\left(\eta = \frac{2 \times N_m}{N + M} \times 100 \quad \geq \quad _s\right)$$

   where N_m is the number of matching point pairs of the two minutia sets, T_m and T_s and are the thresholds under the two conditions, and N and M are the minutia numbers of the two minutia point sets.

These algorithms have also been verified successfully in the C language. For our simple fingerprint authentication service system, we implemented these functions in the Nios II processor.

Database

Due to the limitations of time and conditions, our system implements a simple fingerprint authentication service system. For this system, we loaded the fingerprint database into the flash memory on the DE2 development board. The system stores and accesses data using a linked list.

Liquid Crystal Display (LCD)

The system's LCD module is a complete hardware-control module designed with a hardware description language (HDL). When the CPU is not accessing the SRAM, the image information in the SRAM is displayed on the LCD. Fingerprints are displayed on the left side of the LCD and the names of the participating team and team members, as well as the operation button prompts, are displayed on the right.

The CPU does not need to control the LCD module. Using 1% of the FPGA logic resources, the function can satisfy the real-time system requirements and demonstrate the advantages of combining hardware and software using system-on-a-programmable-chip (SOPC) technology.

Performance Parameters

This section describes the design's performance parameters.

Actual System Performance Parameters

The actual system has the following parameters:

- Power supply: DC, 9 V
- Operating environment
  - Operating temperature: -5° to 45°
  - Relative humidity: 8% to 95%
- Input signal
  - Method: Input by our fingerprint collector with the FPS200 fingerprint sensor
  - Sensor array number: 256 x 300
  - Sensor resolution: 500 DPI
  - Signal transmission mode: SPI
- Display
  - LCD: 16 x 2
  - Display: 320 x 240
- Fingerprint recognition rate
  - Number of tested users: 5
  - Test times: 20 times/person
  - Recognition rate: 99%
  - False rejection rate: 1%
- Consumed time
  - Fingerprint collection time: 2.3207 seconds
  - Fingerprint image preprocessing time: 24.1294 seconds
  - Normalization time: 0.8326 seconds
  - Orientation extraction time: 1.3586 seconds

- Frequency extraction time: 2.8262 seconds
- Gabor filtration time: 12.5535 seconds
- Binarization time: 0.4280 seconds
- Thinning time: 6.1305 seconds
- Minutia extraction time: 1.4634 seconds
- Minutia matching time: subject to the number of fingerprints in the database

■ Fingerprint database storage size—The flash storage device on the DE2 board is only 4 Mbytes. Of the 4, we use 1 Mbyte to store other information, so the fingerprint database is only 3 Mbytes and can store a maximum of 48 fingerprints. If we expand the non-volatile memory, e.g., with another flash device, we can theoretically expand the fingerprint database to any size.

Nios II Processor Functions in the Design

As the core of this system, the Nios II soft-core processor has the following functions:

■ The Nios II processor contains two types of peripherals: standard and custom. It is easy to add standard peripherals. Additionally, Altera and some third-party corporations are providing more and more intellectual property (IP) cores for standard peripherals. Custom peripherals greatly accelerate system operation and save CPU resources. The design's fingerprint image preprocessing normalization module improves the processing speed over 20 times.

■ Custom instructions are a unique feature of the Nios II processor. They can dramatically improve the system performance. The system's Gabor filter takes full advantage of the custom instruction feature and improves the processing speed over 30 times.

■ The C2H hardware acceleration is efficient and easy for designers to use to accelerate system development. Additionally, it highly improves system performance. In this design, we only apply C2H hardware acceleration to some orientation extraction algorithms, which improves the processing speed more than 6 times.

Hardware Resource Usage and Performance Improvement

Table 1 shows the hardware resource usage and Table 2 shows the performance improvement.

Table 1. Resources Used

Resource	Logic Elements (LEs)		Memory Bits	
	Gate	%	Bits	%
Normalization (custom module)	1,394	5	24	<1
Gabor filter (custom instruction)	1,832	6	4,096	1
LCD	470	1	0	0
Orientation extraction (C2H)	5,314	16	512	<1
Cos, exp, and fpoint (custom instruction)	6,937	21	16,386	4
Total	15,947	49	21,018	5

Table 2. Performance Improvement

Operation	Software Operation Time (s)	Hardware Improvement	Operation Time after Improvement (s)
Normalization	16.9491	Hardware module	0.8326
Gabor filter	458.0213	Custom instruction	12.5535
Orientation extraction	9.490	C2H	1.3586
Frequency extraction	31.7303	Floating point, multiplication shift, and cos table look-up	2.8262

Design Architecture

This design considers two schemes. The first scheme is a simple fingerprint authentication service system (see Figure 4) that integrates a database server and web server into the embedded fingerprint identification system to provide a simple web-based network service. The second scheme is a complex fingerprint authentication service system (see Figure 5) that targets mass users who have strict system requirements. We began with the simple fingerprint authentication service system, and if we had enough time in the final stage, we would have expanded the system into a complex fingerprint authentication service system. However, due to time limitations and other constraints, we only implemented the simple fingerprint authentication service system.

Figure 4. Simple Fingerprint Authentication Service System

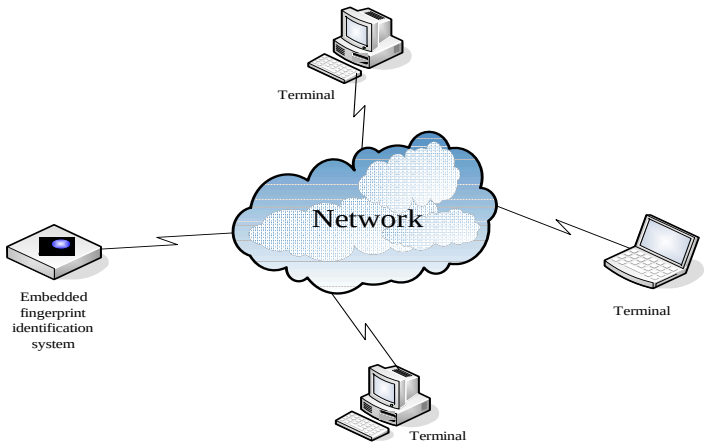
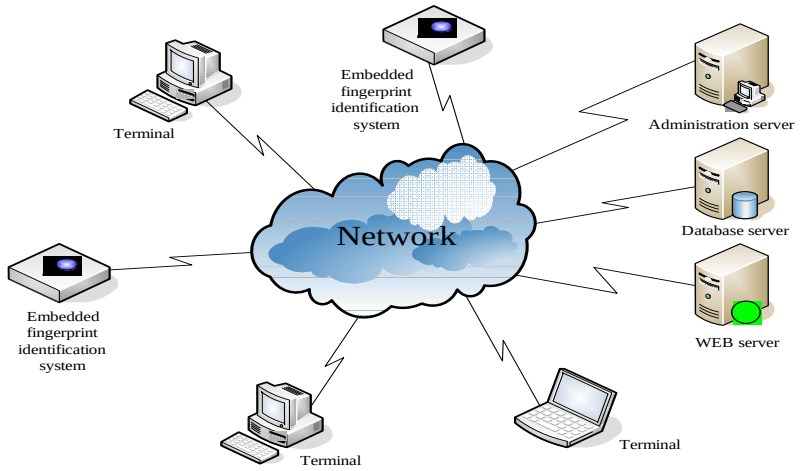


Figure 5. Complex Fingerprint Authentication Service System



Figures 6 through 8 show various block diagrams of the system.

Figure 6. Hardware System Block Diagram

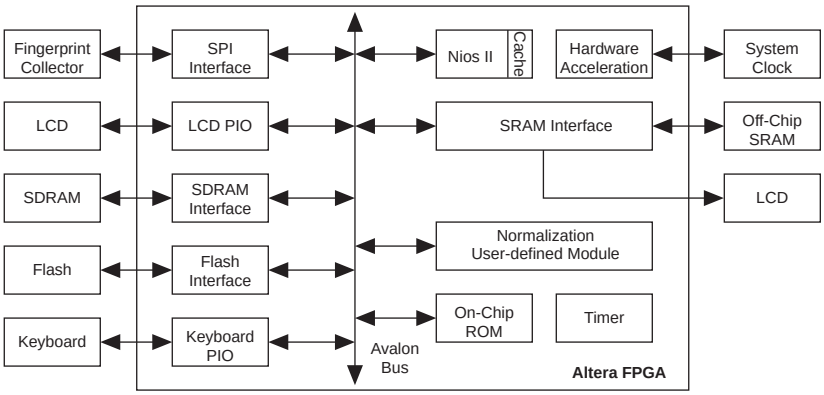


Figure 7. High-Level Block Diagram

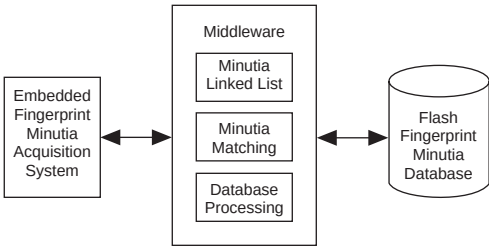
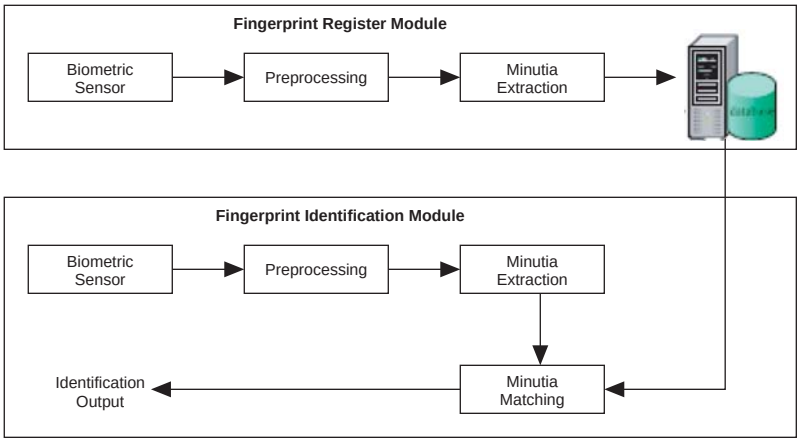


Figure 8. Structural Block Diagram



Figures 9 and 10 show the system flow charts.

Figure 9. Fingerprint Identification System Flow Chart

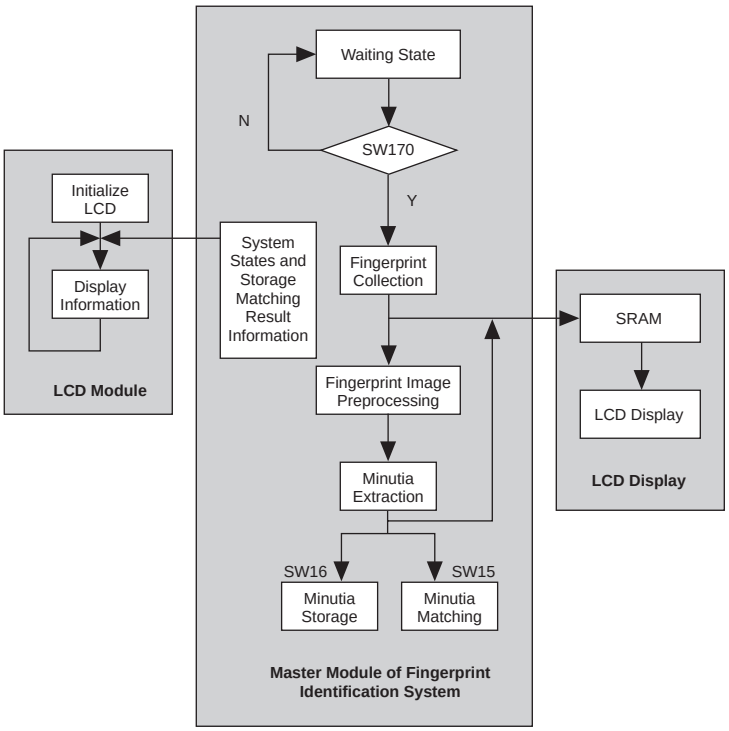
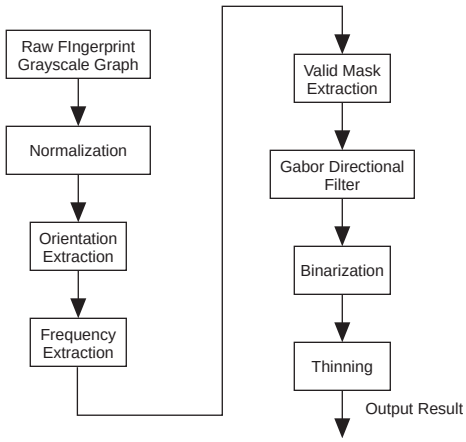


Figure 10. Fingerprint Minutia Preprocessing Algorithm Flow Chart



Figures 11 and 12 show the system in SOPC Builder.

Figure 11. SOPC Builder

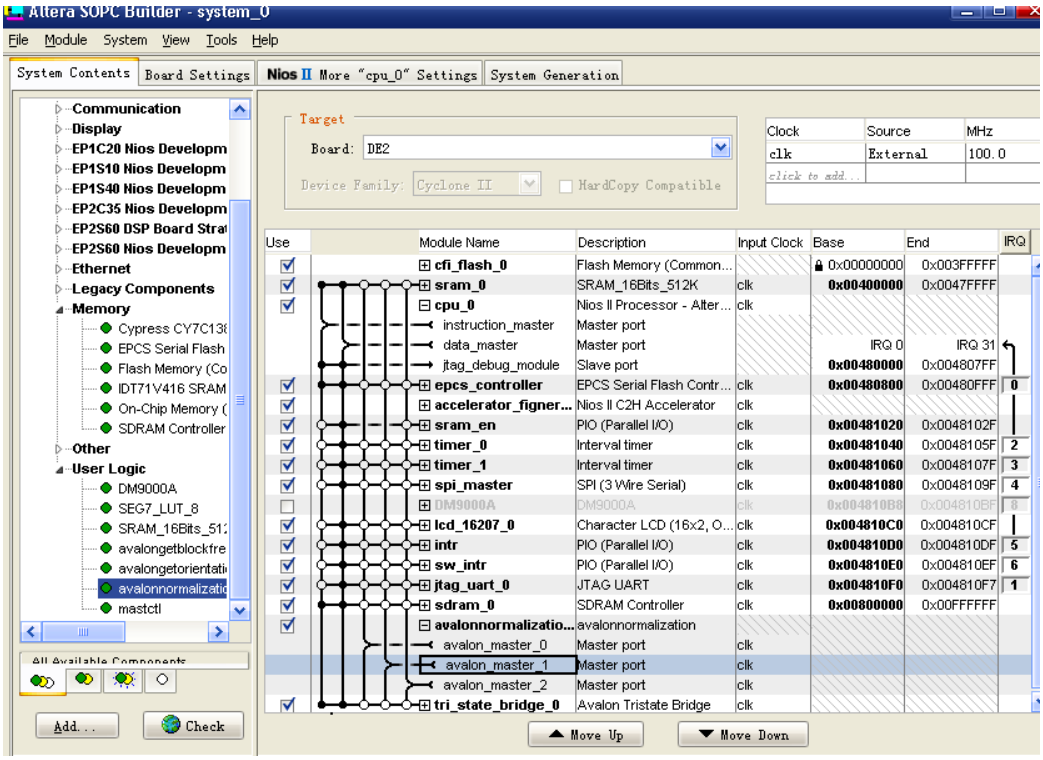


Figure 12. Custom Instructions

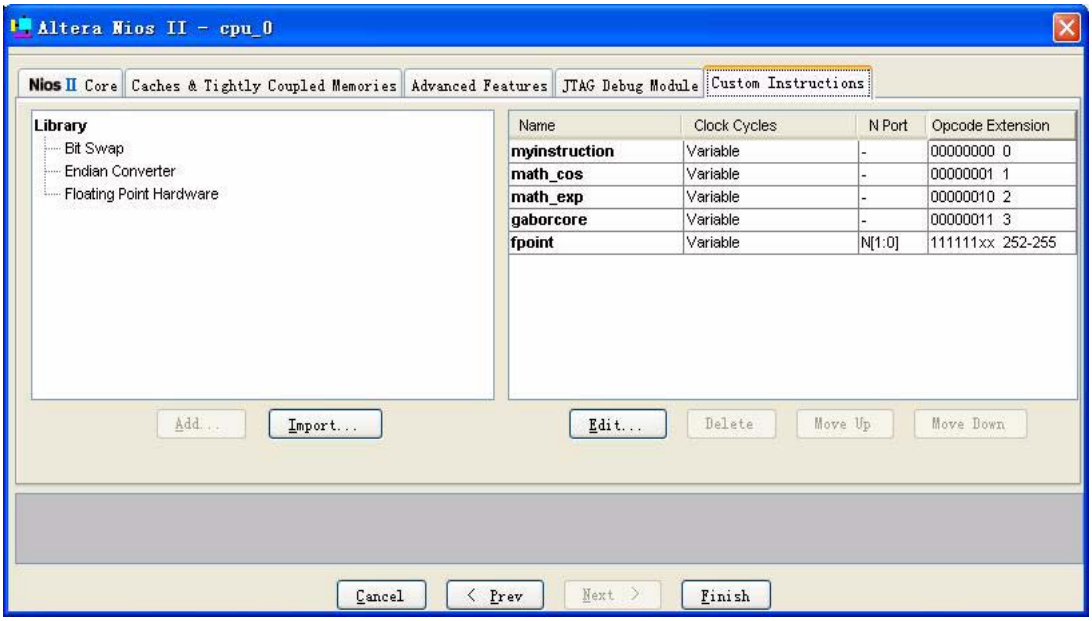
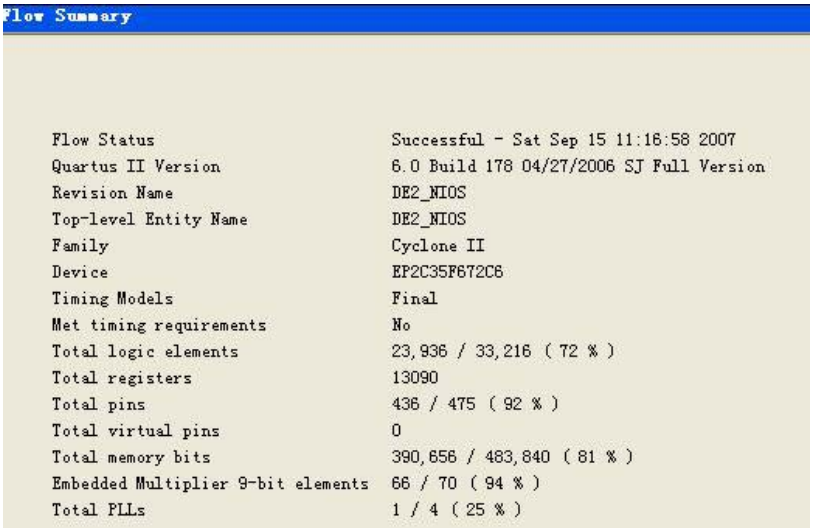


Figure 13 shows the hardware resource usage.

Figure 13. Hardware Resource Usage



Design Methodology

The system's hardware is based on the DE2 development board. The hardware design fully utilizes the board's existing interfaces, e.g., we used the LCD to facilitate the user interface. By combining the LCD and fingerprint collector, the system displays information promptly. The system software development is based on the Nios II embedded soft-core processor and leverages custom instructions, modules, and the C2H Compiler, which accelerates some software in hardware.

System Hardware Design

The following sections describe the hardware design.

Fingerprint Image Collection

Each column of the capacitor array is connected with two sampling/hold circuits to capture only one row of fingerprint images one time. The capture procedures include two stages:

- Capacitors in the chosen row are charged to U_{DD} and store the charge voltage in the first sampling/hold circuit.
- The capacitors discharge to the current source at a speed proportional to the discharged current.

After a discharge period, the second sampling/hold circuit stores the final capacitor discharge voltage. By measuring the difference (ΔU) between the charging and discharging voltages, we can obtain the capacity (C) of each capacitor. The analog signals representing the unit capacities of the row go through A/D conversion to generate the digital fingerprint image information for the row. The system places the information in a register and captures the next row's fingerprint images.

We added an SPI bus module and button interrupt module with SOPC Builder. The SPI module has a 12-MHz clock. A fingerprint image is 256 x 300, therefore, a fingerprint image can be transmitted within 0.1 second. The button interrupt has a double-edge trigger and the button is a slide switch. On power-up, the initialized fingerprint collects the module register values. When the button is interrupted, the fingerprint collection chip transforms the collected signals into digital information and stores them in the register. The CPU sends the digital image data in the fingerprint collector register to the specified memory space in the DE2 board's SDRAM via the SPI bus. After all fingerprint image data is collected, it is normalized and assigned in SRAM. The initial SRAM address is 0 and the data occupies the 0x12c00 address space. In the final stage, only the normalized fingerprint images are processed.

Custom Instructions

In this design, custom instructions target complex mathematical operations and some time-consuming algorithms during fingerprint image preprocessing. Because the embedded Nios II system consumes a lot of time performing complex mathematical operations, the system instead uses hardware to implement the operations and uses custom instructions in the CPU. Figure 12 on page 260 shows three custom instructions we added during CPU customization that accelerate the math_cos, floating operation fPoint, and exponential function math_exp trigonometric functions.

The Gabor directional filter consumes the most time in this design. Because it is very complex, there would not be enough FPGA logic resources if we only used custom modules. Our design implements some of the time-consuming algorithms through custom instructions. The main formula of Gabor directional filtration is:

$$G(x, y) = \sum_{u=-W/2}^{W/2} \sum_{v=-W/2}^{W/2} g'(u, v) N(x - u, y - v)$$

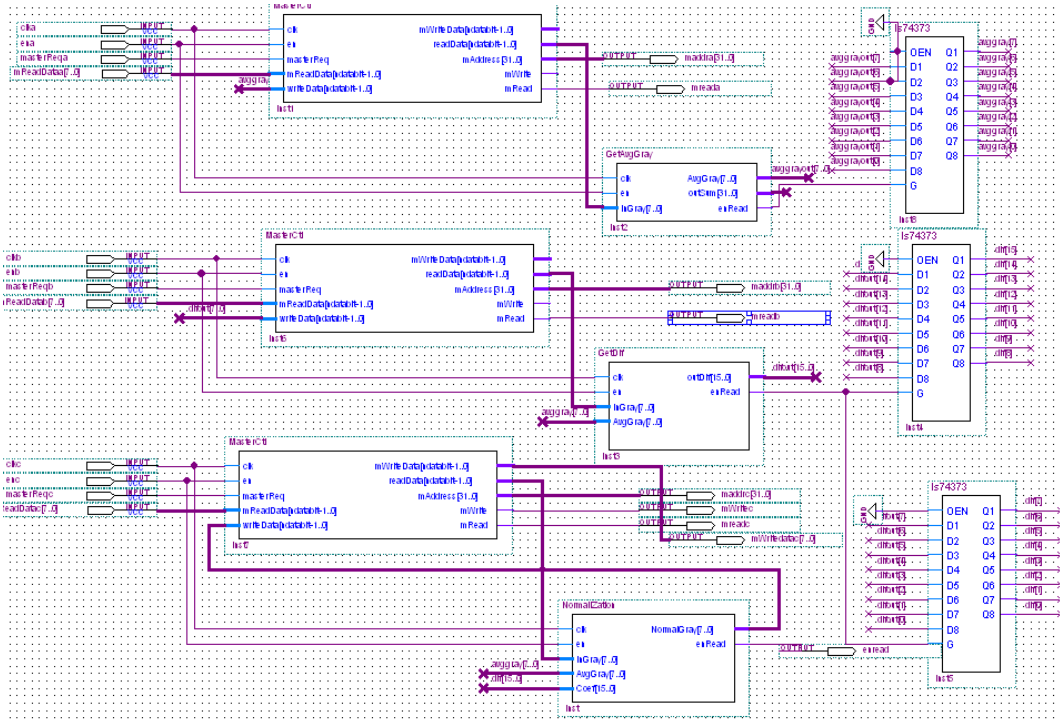
$$g'(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x'^2 + y'^2}{2\sigma^2}\right) \cos(2\pi f x')$$

N(x - u, y - v) is the grayscale value of the normalized fingerprint image. We use custom instructions to implement g'(x, y) x N(x - u, y - v), and implemented the accumulation summarization in software. After the CPU is generated, these functions generate the corresponding function interfaces. We invoke the programs in the same way as common function sub-programs.

Custom Modules

Our design contains many image processing algorithms, which require a lot of processing time. To shorten the processing time as much as possible, we convert normalization (the first part of image preprocessing) into hardware, which does not affect software processing. Figure 14 shows the normalization preprocessing hardware schematic.

Figure 14. Hardware Normalization Schematic



Because normalization compares the images and then normalizes them, we read the image using different methods, that is, we obtain the information using three main control modules and the Avalon® bus interface, and then generate the normalized image. Figure 15 shows the image processing result.

Figure 15. Image Preprocessing Before and After Normalization

C2H Compiler

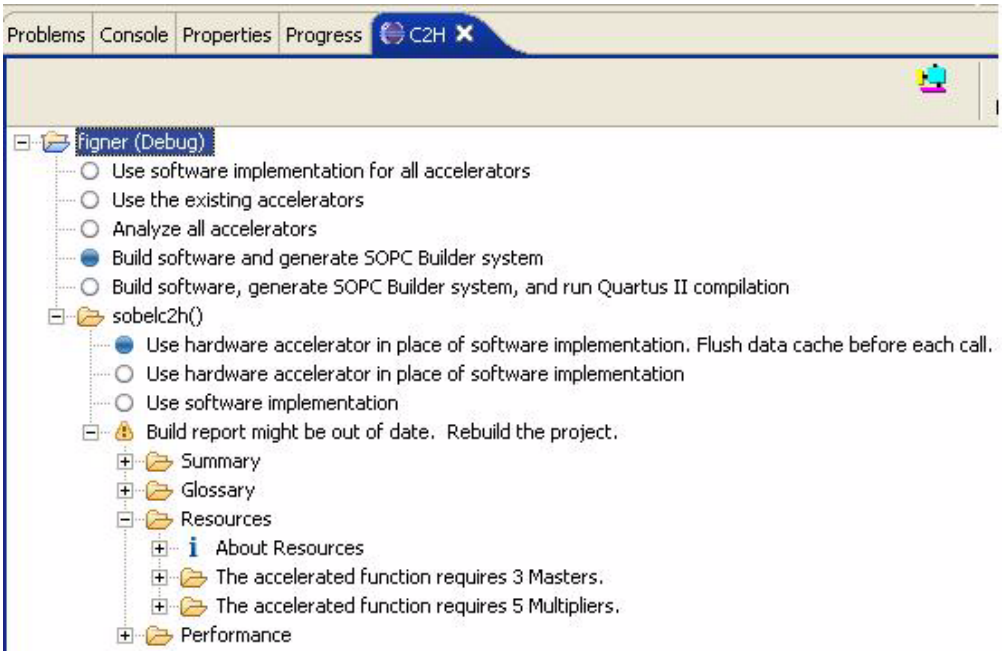
In this design, we used the C2H Compiler for integer algorithms with many loops and single operations. Because most of the algorithms use floating-point numbers, we translate them into integers before using the C2H Compiler to convert them into hardware. We used the C2H Compiler to optimize the Sobel operator for orientation extraction, which has 4 loops and integer accumulation algorithms. Figure 16 shows the Sobel operator C2H function.

Figure 16. Sobel Operator C2H Function

```
void sobelc2h(int iWindow,Img *ImgSrc,int *pDst,int *pModelCente
{
  int iRow,iCol,i,j;
  int iWidth,iHeight;//iSize;
  UINT8 *pSrcAddr,*pSrc;
  int iResult;
  UINT8 uGray;
  iWidth = ImgSrc->size.iWidth;
  iHeight = ImgSrc->size.iHeight;
  pSrc = ImgSrc->pImg;

  //iSize = iWidth*iHeight;
  //针对每隔像素均进行求取沿x或y方向上的梯度值
  for (iRow = iWindow;iRow < iHeight - iWindow;iRow++)
  {
    for (iCol = iWindow;iCol < iWidth - iWindow;iCol++)
    {
      pSrcAddr = pSrc + iRow * iWidth + iCol;
      //Sobel Core
      iResult = 0;
      for (i = -iWindow;i<=iWindow;i++)
      {
        for (j=-iWindow;j<=iWindow;j++)
        {
          uGray = *(pSrcAddr + i*iWidth + j);
          iResult += *(pModelCenter + i*iWinWidth + j) * uGray;
        }
      }

      *(pDst + iRow * iWidth + iCol) = iResult;
    }
  }
```

Figure 17 shows the C2H Compiler settings in SOPC Builder.

Figure 17. C2H Compiler Settings



After using C2H hardware acceleration, although the algorithm uses 5,314 logic resources, 512 memory bits, one M4K block, 22 DSP elements. and 11 18 x 18 DSP blocks (including 3 masters and 5 multipliers), the Sobel operator's operation speed increases 35 times and the speed of the whole orientation extraction process increases 6 times.

C2H hardware acceleration satisfies the system's speed requirements and saves the time in hardware module design by occupying some hardware resources. A system can use up to five C2H hardware accelerations, therefore if there are enough hardware resources, multiple C2H accelerations can be used.

LCD

The LCD module is a separate hardware module and is only connected to the SRAM. When the CPU is not accessing the SRAM, the LCD module displays fingerprint image data in the SRAM. During the fingerprint collection stage, the image is dumped to the SRAM by memory; then, the CPU releases SRAM control and the LCD module accesses SRAM and displays the image. When fingerprint preprocessing and minutia point extraction are complete, the CPU controls SRAM again, covering the original fingerprint image with the processed fingerprint minutia image and releasing the SRAM control to display the fingerprint minutia image.

In our system, when the CPU writes the fingerprint information data, it conflicts with the LCD reading the SRAM. To resolve this problem, we added an enabled programmable I/O (PIO) in SOPC Builder that allows the LCD to read the SRAM when the CPU is not accessing it. This design ensures that the CPU can access the SRAM at any time and the LCD can provide a stable display. Having the LCD read the SRAM relieves the CPU through the designed hardware module without CPU interference. When users implement mass and single-domain peripheral interaction functions during SOPC system design, they should use custom master peripherals because they can operate the storage area directly while interacting with off-chip devices; the CPU does not need to use a special process for storage area. These modules can be added as necessary if there are enough logic resources. This flexibility is one advantage of SOPC system design over system-on-chip (SOC) design.

System Software Design

The most important part of the software design is the control function, image preprocessing, minutia point extraction, and minutia matching.

Control Function

The control function mainly controls interrupts. It informs the system of specific operating instructions, including collecting the fingerprint, matching it, and storing it using button interrupts.

- *Collect fingerprint*—SPI communicates with the CPU to notify it when fingerprint collection ends, at which point, service function `fps_Irq_In()` is interrupted. When the collection finishes, the CPU automatically processes the fingerprints, extracts the minutia points using `FingerImgEnhance(pImgSrc,pImgSrc)`, and stores collected fingerprint information into the current finger information linked list `CurrentMinutia` (custom linked list structure) in the SDRAM.

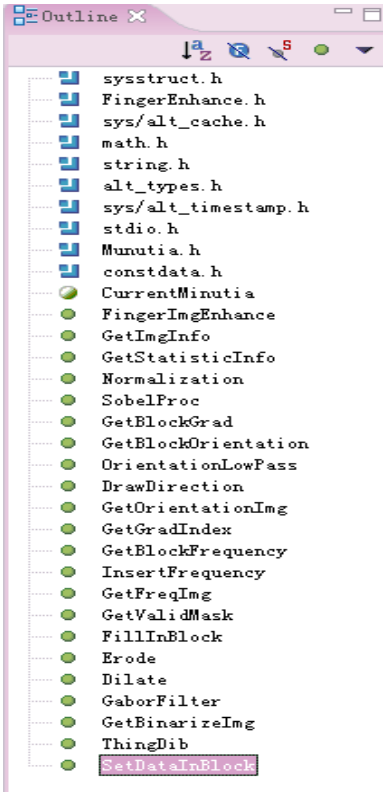
- *Match fingerprint*—If the system determines that there is a match, it performs `matchFigMinutia(&CurrentMinutia)`.

- *Store fingerprint*—To store the information, the system uses the `wOneFigMinutiaToFlash()` function. The current fingerprint information is stored in the flash device's fingerprint linked list.

Image Preprocessing

Image preprocessing enhances the fingerprint image, and includes the functions shown in Figure 18.

Figure 18. Image Preprocessing Functions



The functions are described as follows:

- *Normalization*—The system performs normalization in hardware. After the image is collected, the hardware normalizes the image. The resulting image is put into the 0 address of the SRAM. The software can read the image directly without interface problems.

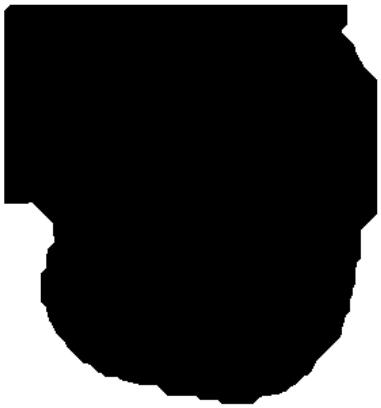
- *Orientation extraction*—The `GetBlockOrientation` function calculates the fingerprint image block orientation and invokes `Soborcore`. C2H hardware acceleration implements hardware instructions to calculate the gradients of each pixel point in the x and y orientations. Then `GetGradIndex()` is invoked, and orientation extraction is performed. Figure 19 shows the pattern.

Figure 19. Orientation Extraction Pattern

- *Frequency extraction*—The `GetBlockFrequency` function calculates the fingerprint image block frequency. The changes of fingerprint ridges and valleys constitute a 2-D sine wave. To obtain the frequency, the system invokes `GetImgInfo(ImgSrc)` as described previously. It gets the fingerprint information and frequency interpolating function `InsertFrequency(pOut,pOut,iWindow,ImgSrc.size)`, and invokes the orientation filter, `OrientationLowPass(pOut,pOut,iWindow)`, to smooth the image.

- *Valid mask extraction*—A full mask fingerprint image cannot usually be generated during collection. This function, `GetValidMask`, generates the foreground mask (the fingerprint image, which is the part of the finger touching the sensor) and background mask (hash). Valid mask extraction distinguishes the foreground from the background. It invokes `GetGradIndex(*(pOrient + iOffSetAddr))` to get the orientation. The `if(abs(xDelta - yDelta) > QNios)` function sets the threshold to compare image contrast. A valid mask dramatically changes the image as shown in Figure 20.

Figure 20. Valid Mask



- *Gabor filter*—To improve the fingerprint image quality, we use an orientation filter enhancement method. In noise removal and maintaining the fingerprint ridge structure, like a bandpass filter feature, we optimized an important operation with custom instructions and implement the formula

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_x}\exp\left(-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_x^2}\right)\right)\exp(-2\pi ifx)$$

  with software. Figure 21 shows the effect.

Figure 21. Gabor Filter Effect

- *Binarization and noise removal*—The `GetBinarizeImg()` function performs binarization and removes noise. During binarization, the Gabor-filtered grayscale image of a specific fingerprint generates another monochrome fingerprint image. The adopted adaptive local threshold scheme adjusts the threshold by the global grayscale of a block in the image. Implementing binarization in software is comparatively easy, but the loops consume a lot of processing time. Figure 22 shows the result.

Figure 22. Binarization and Noise Removal Result

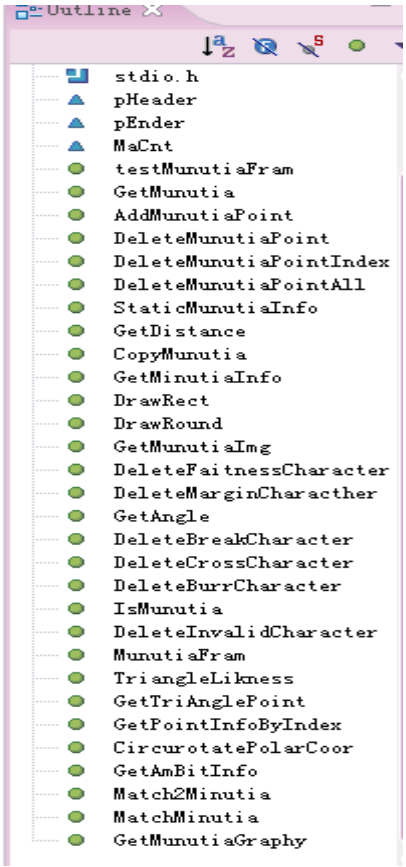
- *Thinning*—The `ThingDib(Img ImgDst, Img ImgSrc)` function extracts minutia points by thinning the binarization image to a single-pixel fringe central point and thread image. See Figure 23.

Figure 23. Thinning Result

Minutia Point Extraction

After a raw fingerprint image goes through orientation filtering, binarization, and thinning, it becomes a thinned image. We then determine the endpoints and bifurcation according to the crossing of each point on the thinned image, and extract the useful information of the two minutia points such as coordinate position, type, and orientation. Figure 24 shows the minutia point extraction functions.

Figure 24. Minutia Point Extraction Functions



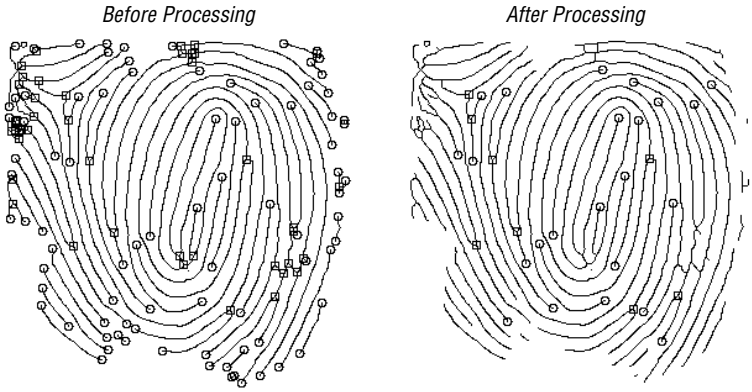
The main function, `GetMunutia(UINT8 *pDst,float *pOrient,UINT8 *pSrc)`, extracts minutia points by:

- Getting all possible minutia points: `GetMinutiaInfo(&CurrentMinutia,pOrient,ImgSrc)`
- Deleting false minutia points: `DeleteInvalidCharacter(&CurrentMinutia, &pBreakMunutia,&pBurrMunutia,&pCrossMunutia)`, which includes:
  - Deleting the false minutiae caused by image obscurity: `DeleteFaitnessCharacter(&CurrentMinutia,2,3,3,10,size)`
  - Deleting the false minutia points caused by broken lines: `DeleteBreakCharacter(&pBreakMunutia,5,20)`
  - Deleting burrs: `DeleteBurrCharacter(&pBurrMunutia,6,12)`
  - Deleting holes and bridges: `DeleteCrossCharacter(&pCrossMunutia,6,15)`
  - Deleting margin minutia points: `DeleteMarginCharacther(&CurrentMinutia,ImgSrc)`
- Output the image: `GetMunutiaImg(ImgDst,&CurrentMinutia)`

- Operations related to the linked list:
  - Add minutiae to the information linked list: `void AddMunutiaPoint(Minutia *pMinutia, CPoint point, float fOrient, enum MinutiaPointType pointType)`
  - Delete the designated minutia points: `void DeleteMunutiaPoint(Minutia *pMinutia,MinutiaInfo* pPoint)`
  - Delete the minutia points of the designated index: `void DeleteMunutiaPointIndex(Minutia *pMinutia,int index)`
  - Take statistics of Ne and Nb: `void StaticMunutiaInfo(Minutia *pMinutia,CSize size, int iWindow)`
  - Calculate the distance of two minutia points: `int GetDistance(CPoint cpPoint1,CPoint cpPoint2)`
  - Copy the template: `void CopyMunutia(Minutia *pFromMunutia, Minutia *pToMunutia)`
  - Get minutia points: `int GetMinutiaInfo(Minutia* pMunutia, float* pOrient, Img ImgSrc //thinned image)`
  - Draw the rectangle: `void DrawRect(Img ImgSrc, CPoint cpPoint, int iWindow)`, etc.

Figure 25 shows the effect after processing.

Figure 25. Before and After Minutia Point Extraction



The fingerprint minutia data structure includes the minutia point structure and fingerprint minutia set.

Minutia Point Structure

A complete description of minutia points includes the coordinate point, orientation, and type of minutia on the image. We designed the following data structure to describe the minutia point.

Minutia point information in the rectangular coordinate system:

```
typedef strut MunutiaPointInfo_st{
  int x; //horizontal coordinate value
  int y; //Vertical coordinate value
  float index; //orientation
  enum MunutiaPointType{END,CROSS} type; //minutia point type
}MunutiaPointInfo, *pMunutiaPoint;
```

Minutia point information in the polar coordinate system:

```
typedef strut MunutiaPolarInfo_st{
  int iRadius; //polar radius
  float fTheta; //polar angle
  float index; //orientation
  enum MunutiaPointType{END,CROSS} type; //minutia point type
}MunutiaPolarInfo,*pMunutiaPolar;
```

Fingerprint Minutia Set

The minutia set is the set of all minutia points of a fingerprint. Because the design uses a deletion operation to remove false minutia points, we use a two-way linked list design with the following data structure:

```
typedef struct Munutia_st{
  MunutiaPointInfo Info; //information of current minutia point under rectangular coordinate system
  MunutiaPloarInfo polarInfo;//information of current minutia point under polar coordinate system
  struct Munutia_st *pPrev; //direct to previous minutia point information
  struct Munutia_st *pNext; //direct to next minutia point information
}Munutia,*pMunutia;
```

Minutia Matching

In this design, we:

- Use typical point mode matching algorithm based on a minutia point coordinate mode.
- Use a triangle structure composed of three neighboring minutia points to locate a datum point and evaluate the conversion parameter.
- Conduct matching in the polar coordinate system after coordinate translation.
- Introduce multiple judgement conditions and a variable limit box matching algorithm to improve the recognition rate.

The main content is the matching function `matchFigMinutia(Minutia *CurrMa)`. If the current fingerprint matches, the fingerprint minutia set and the matching nodes are returned. Minutia matching functions include:

- Determine whether the triangles are similar: `float TriangleLikness(MinutiaPointInfo* PPointInfo, MinutiaPointInfo* QPointInfo, int iDelta /*= 5*/)`
- Get two coordinate points near to a datum: `mark bool GetTriAnglePoint(char* pSrc, MinutiaPointInfo* pointInfo,CSize size)`
- Rotate polar coordinate: `void CircurotatePolarCoor(float rotateAngle)`
- Get limit box size: `AmBitInfo GetAmBitInfo(int iRadius)`

- Match two minutia points: `int Match2Minutia(MinutiaPointPoolInfo PInfo, MinutiaPointPoolInfo QInfo)`
- Match current minutia set with template minutia set `pQMunutia` and roll back matching node number: `int MatchMinutia(Minutia *pQMunutia)`
- Display minutia point on graph `psr` and the displayed grayscale value is the position of the minutia point in the linked list: `void GetMunutiaGraphy(char* pSrc, CSize size)`

Design Features

Our design has the following main features:

- *Highly integrated SOPC technology and embedded custom modules*—If we implemented the image preprocessing, including image normalization, frequency extraction, orientation extraction, and filtering, directly in the embedded Nios II processor, it would take too much processing time. Therefore, the system uses an FPGA hardware algorithm to implement preprocessing. Binarization and thinning of the image consume less time and are directly implemented in the Nios II processor.
- *Custom instructions*—The fingerprint identification system involves many floating-point, multiplication, evolution, and rotation operations. If we implemented them in the embedded Nios II system, it would consume too many resources. Therefore, this design uses custom instructions to add the custom functions directly into the arithmetic logic units of the Nios II CPU, allowing these complex, time-consuming operations to operate in hardware. As a result, the data processing time increases dramatically and the design's real-time requirements are met. Our system uses five custom logic instructions.
- *C2H Compiler*—This system uses hardware acceleration with the C2H Compiler to accelerate software sub-programs that need high performance for image preprocessing. This acceleration dramatically improves the system's general performance. This system uses one C2H module.
- *Two fingerprint identification system architecture schemes*—According to practical production and actual requirements, we developed two feasible fingerprint identification system architecture schemes: complex and simple. Based on database technology, FPGA technology, and embedded technology, we propose a valid implementation. This system implements the simple fingerprint authentication service system.
- *Improved minutia matching algorithm*—In this design, we used a triangle structure composed of three neighboring minutia points to locate the datum point and evaluate conversion parameters, conducted polar coordinate system matching after coordinate translation, and introduced multiple judgement conditions and a variable limit box matching algorithm to improve the recognition rate.
- *Floating-point shift*—Because many multiplication algorithms in floating-point operations consume a lot of processing time, we shift floating-point numbers to integers, thereby improving the system's algorithm speed three to four times.
- *Convenient product design and development*—The FPGA-based Nios II soft-core processor and its SOPC solution have tailor-made, reconfigurable advantages over hard-core processors. It is easy to develop the software and hardware products with the Nios II processor. Additionally, the Nios II processor facilitates collaborative system design and permits system maintenance and upgrades. By using an FPGA with abundant resources, we can implement a multi-CPU system and simplify the hardware circuit using a design control module, satisfying the system's functional requirements and cutting the design cost. Moreover, we can upgrade the system hardware by loading an updated soft-core system without affecting the peripheral circuit.

Conclusion

With this design contest, we learned about the power and convenience of the Nios II processor, while improving our engineering ability. The advantages of the Nios II processor are as follows:

- Few resources are used. One Nios II CPU only occupies thousands of LEs.
- Constructing an SOPC system is flexible and convenient. We can create the desired system quickly and customize the peripheral modules required by the system, significantly reducing the difficulty of the hardware design and development period.
- FPGAs are flexibly designed for DSP. Users can design completely arithmetic circuits according to their needs. An FPGA can perform operation functions more efficiently at lower frequencies than CPU co-processors. For example, because this design uses a 100-MHz system clock, some algorithm operations are much faster than a 2-GHz Pentium 4 processor.
- The Nios II C2H Compiler and custom instructions help users implement algorithms and logic controls. With the C2H Compiler, users can input pointer parameters. However, floating-point numbers cannot be processed internally and users cannot modify the acceleration logic circuits generated by the compiler. Therefore, users have to first quantify the floating-point numbers and then transform them into floats after output during digital signal processing. This method is not as efficient as using the CPU's internal floating-point operation acceleration instructions. Custom instructions allow users to design hardware logic, but their input and output parameters are not as flexible as those of the C2H Compiler because common users seldom add single or double operand instructions but often process a lot of data. When we experienced this kind of problem, we had to replace the C2H Compiler and custom instruction implementation with a custom module and interrupt (or query), which destroys the system software flow to some extent. If the C2H Compiler and custom instruction advantages could be combined to create a universal function interface and allow an internal hardware implementation of a flexible design, the Nios II development platform would benefit users more.
- The FPGA-based flexible embedded design supports interfaces per the user's specific requirements. It can represent users' design ideas better than a hard-core processor design. However, FPGAs require users to plan and design the SOPC system comprehensively; in particular, the final adaptation of the hardware netlist in the Quartus® II software consumes too much time.

The following areas of the design could be improved:

- Due to the limitations of the hardware system and the small-sized volatile and non-volatile memory, the fingerprint database is small. If we had enough time to design a hardware system according to the practical requirements, abandon unnecessary hardware such as the Ethernet chip and video collection chip, and expand the memory, we could have improved the real-time performance and addressing properties of the design while lowering the development cost. Additionally, if required, we could add a network to the system so that users identify a fingerprint using a remote log-in, making the system more convenient for use in a civil access control system.
- As the scheme became clear with the progress of the design, we improved the real-time processing operations, hardware acceleration, and algorithms for the most time-consuming area of the design: fingerprint image preprocessing. As a result, fingerprint image preprocessing consumes 30 seconds instead of the original 10 minutes. Additionally, because the algorithms are too complex, we could only convert some time-consuming modules into hardware and instructions for image normalization, orientation and frequency extraction, valid mask extraction, Gabor filtering, thinning, binarization, image expansion and corrosion, etc. If we had more time for optimization, we could design the system to complete minutia extraction in 1 second.

- We evaluated the design effect for processing time and fingerprint matching accuracy. To save time, we appropriately adjusted the accuracy requirements for some areas, and then chose the proper parameters after weighing.

- We used the Cyclone® II EP2C35 FPGA in this design, which has sufficient logic units but was deficient in supporting the required hardware arithmetic logic units. If the FPGA has fewer hardware arithmetic logic units than required by the algorithms, a system signal processing bottleneck will occur. Similarly, the CPU clock is hard to improve because of the FPGA features. In the future, we plan to leverage the abundant hardware arithmetic units in the Stratix FPGAs to accelerate operations and improve operation accuracy and speed.

During this contest, we deeply studied SOPC technology, mastered its elementary design concepts, and achieved our target. We thank our instructor, colleagues in the lab, and all the other participants. The joint effort paved the way for the success of our system.

Appendix: C2H Compiler Usage

Our system is a Nios II fingerprint identification system that performs image processing. We used the C2H Compiler for integer algorithms with many loops and simple operations. Because most algorithms in the system use floating-point numbers, we translated them into integers before using the C2H Compiler to optimize them into hardware.

The part of the design we optimized with the C2H Compiler is the Sobel operator, which we used for orientation extraction. This part included 4 loops and an integer accumulation algorithm. Addresses are stored as pointers. With the optimization, the algorithm speed of the Sobel operator increased 35 times.

Algorithm Description

We used the Sobel operator to obtain the fingerprint orientation. The Sobel operator calculates the gradients $G_x(U,v)$ and $G_y(U,v)$ and of each pixel point (i,j) in the X and Y directions. Figure 26 shows the Sobel operator's template coefficient.

Figure 26. Sobel Operator Template Coefficient

X Axis Direction

1	2	1
0	0	0
-1	-2	-1

Y Axis Direction

1	0	-1
2	0	-2
1	0	-1

Figure 27 shows the Sobel operator C2H function.

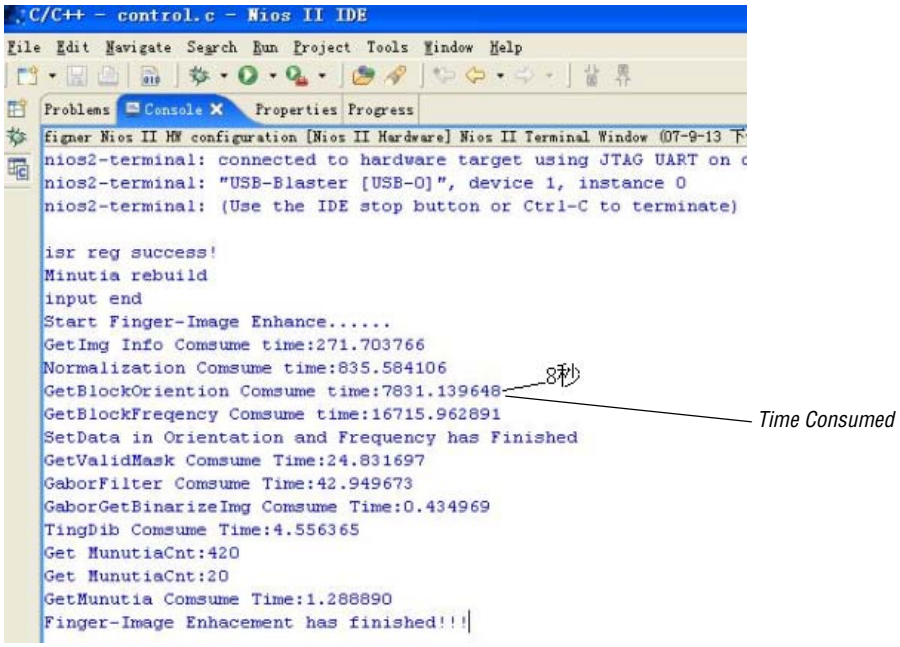
Figure 27. Sobel Operator C2H Compiler Function

```
void sobelc2h(int iWindow,Img *ImgSrc,int *pDst,int *pModelCente
{
  int iRow,iCol,i,j;
  int iWidth,iHeight;//iSize;
  UINT8 *pSrcAddr,*pSrc;
  int iResult;
  UINT8 uGray;
  iWidth = ImgSrc->size.iWidth;
  iHeight = ImgSrc->size.iHeight;
  pSrc = ImgSrc->pImg;

  //iSize = iWidth*iHeight;
  //针对每隔像素均进行求取沿X或Y方向上的梯度值
  for (iRow = iWindow;iRow < iHeight - iWindow;iRow++)
  {
    for (iCol = iWindow;iCol < iWidth - iWindow;iCol++)
    {
      pSrcAddr = pSrc + iRow * iWidth + iCol;
      //Sobel Core
      iResult = 0;
      for (i = -iWindow;i<=iWindow;i++)
      {
        for (j=-iWindow;j<=iWindow;j++)
        {
          uGray = *(pSrcAddr + i*iWidth + j);
          iResult += *(pModelCenter + i*iWinWidth + j) * uGray;
        }
      }

      *(pDst + iRow * iWidth + iCol) = iResult;
    }
  }
```

Performance Comparison

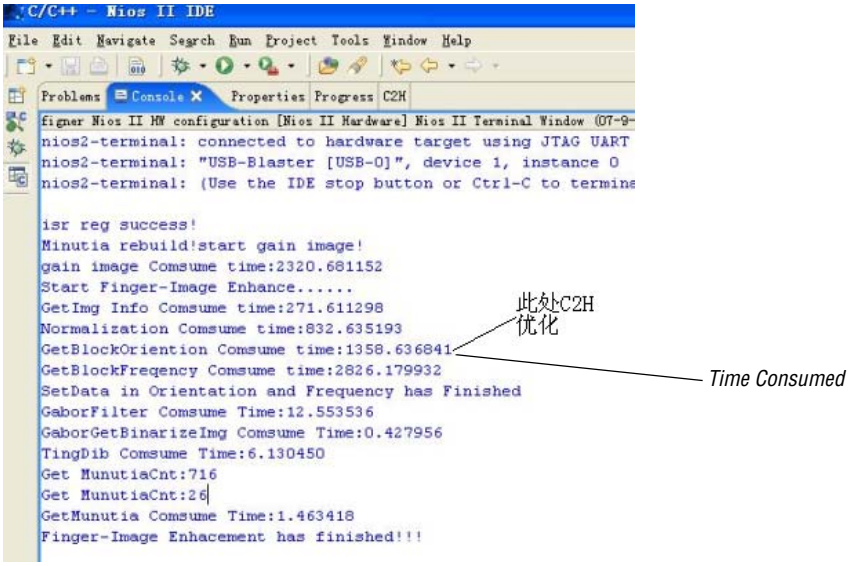
The Sobel operator's algorithm speed increased 35 times, which is only one step in orientation processing. The whole orientation extraction speed increased 6 times, and the effect was remarkable. Figure 28 shows the speed with a software-only implementation.

Figure 28. Consumed Time with Software-Only Implementation



After C2H optimization, the consumed time is only 1 second (see Figure 29).

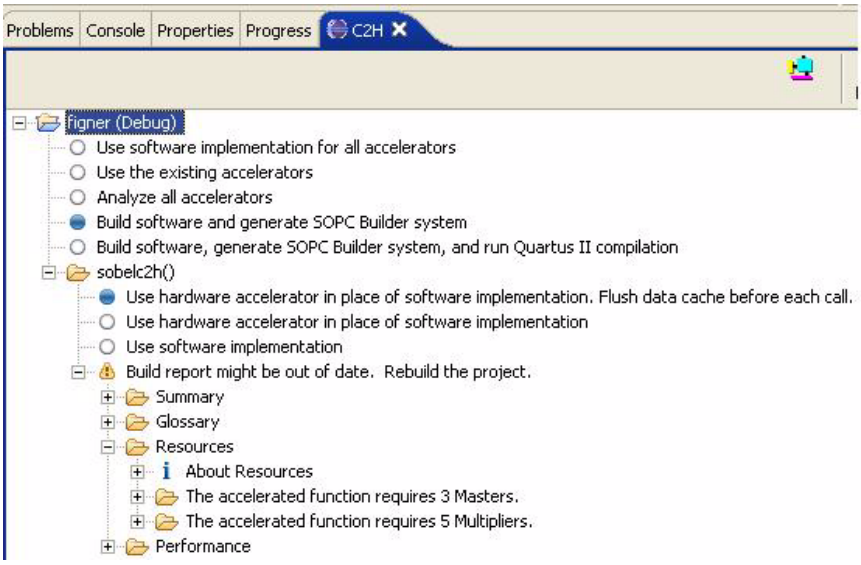
Figure 29. Consumed Time After Optimization



The processor's clock frequency is 100 MHz.

Figure 30 shows other logic that we accelerated and optimized.

Figure 30. Other Accelerated and Optimized Logic



Master Port

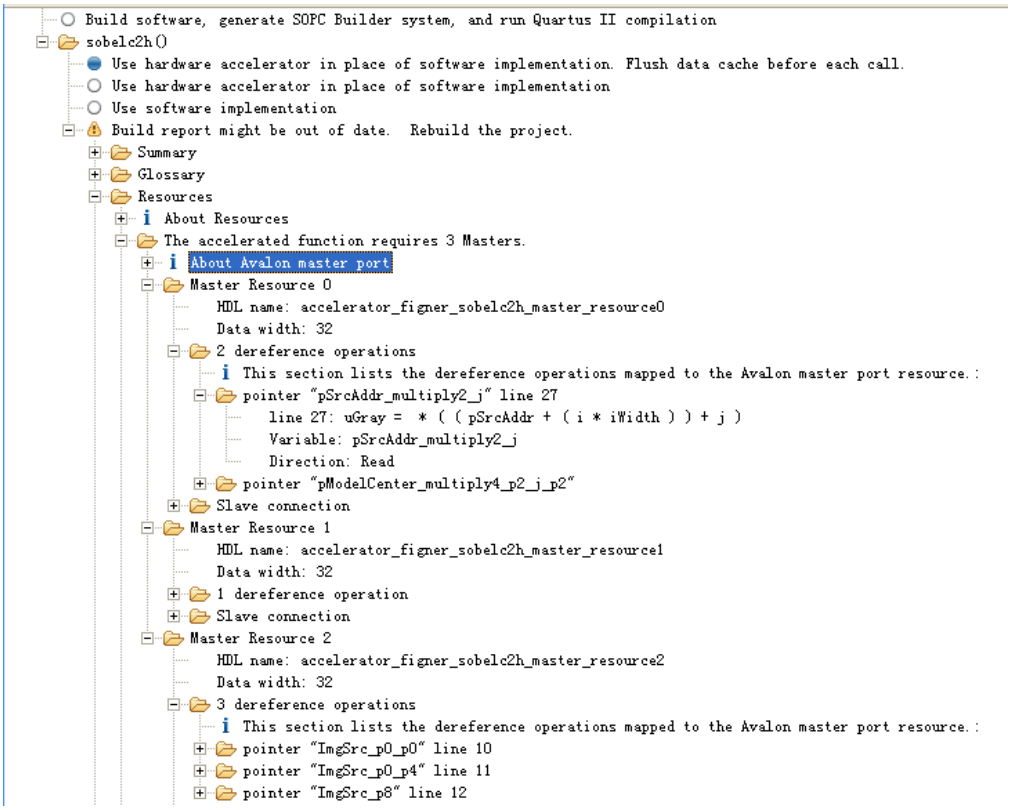
As with the algorithm, the SDRAM information is read in software through pointers. Because the C2H Compiler uses the master port to read and write SRAM and SDRAM and there are three different software pointer operations, we used three master ports for optimization. The optimization had

remarkable advantages and was very convenient to compile. The C2H Compiler freed us from developing complicated custom instructions or modules, compiling the master port operation, creating the hard-to-control working sequence of three master ports, and combining the operation with the algorithm. It also eliminated the need to implement a difficult algorithm sequence control for loading three master ports in a hardware description language. Figures 31 and 32 show the master port hardware and software, respectively.

Figure 31. Master Port Hardware

```
void sobelc2h(int iWindow,Img *ImgSrc,int *pDst,int *pModelCente
{
  int iRow,iCol,i,j;
  int iWidth,iHeight;//iSize;
  UINT8 *pSrcAddr,*pSrc;
  int iResult;
  UINT8 uGray;
  iWidth = ImgSrc->size.iWidth;
  iHeight = ImgSrc->size.iHeight;
  pSrc = ImgSrc->pImg;

  //iSize = iWidth*iHeight;
  //针对每隔像素均进行求取沿x或y方向上的梯度值
  for (iRow = iWindow;iRow < iHeight - iWindow;iRow++)
  {
    for (iCol = iWindow;iCol < iWidth - iWindow;iCol++)
    {
      pSrcAddr = pSrc + iRow * iWidth + iCol;
      //Sobel Core
      iResult = 0;
      for (i = -iWindow;i<=iWindow;i++)
      {
        for (j=-iWindow;j<=iWindow;j++)
        {
          uGray = *(pSrcAddr + i*iWidth + j);
          iResult += *(pModelCenter + i*iWinWidth + j) * uGray;
        }
      }

      *(pDst + iRow * iWidth + iCol) = iResult;
    }
  }
```

Figure 32. Master Port Software



Hardware Multiplier

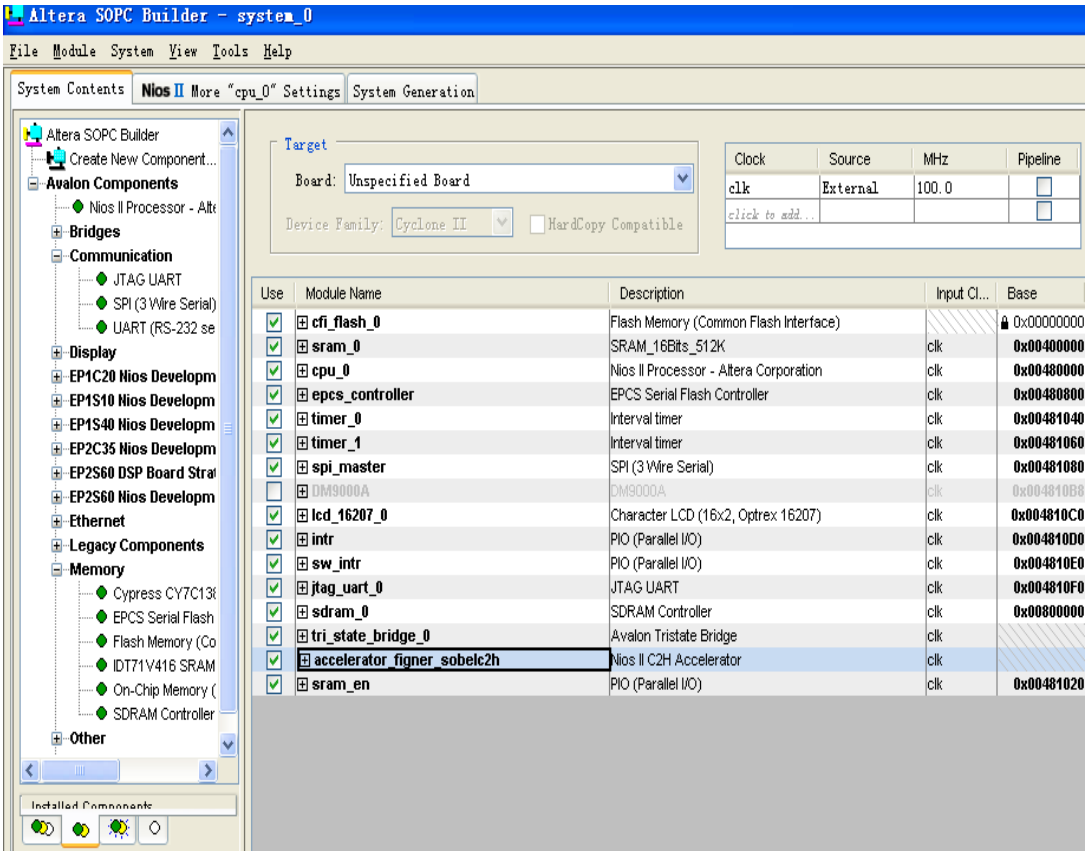
There are five multiplication operations in software that are automatically transformed into five hardware multipliers during C2H optimization. This implementation greatly shortens the processing time.

Loop

The effect of the loop optimization was remarkable. After being transformed into hardware, for, loop, while, etc. are just conditions. The loop algorithms are all transformed into hardware, increasing their speed. Therefore, using C2H optimization for image processing provides significant advantages.

Figure 33 shows the hardware in SOPC Builder.

Figure 33. Hardware In SOPC Builder



Accelerated with the C2H Compiler, the algorithm occupies 5,314 gates of logic resources, 512 memory bits, one M4K block, 22 DSP elements, and 11 18 x 18 DSP blocks (including three master ports and five multipliers). The operation speed of the Sobel operator increased 35 times and the speed of the orientation extraction process increased 6 times.

Conclusion

C2H hardware acceleration satisfied the system's time requirements and saved development time in the hardware module in exchange for using some hardware resources. A system can employ C2H hardware acceleration five times or more provided there are enough hardware resources.

During the design process, we tried to optimize the Gabor operator algorithm with the C2H Compiler because the algorithm contains so many pointer operations and complicated loop operations. Upon optimization, however, it required five master ports and 13 multipliers without giving a significant speed increase. Therefore, we gave up using the C2H Compiler for optimization and instead we optimized one operation in the loop using hardware. To optimize another area (and improve our skills), we used custom instructions instead of the Sobel operator.

To conclude, the C2H Compiler offers a sound optimization effect for algorithms that have many complicated loops, pointer operations, and master ports provided that the C2H conditions are met, that is, the operation does not have floating-point operations and the designer transfers arrays into pointers. The C2H Compiler is an excellent optimization tool. We believe it will become better and more flexible in the future.

Third Prize

Fingerprint Identification System Based on the Nios II Processor

Institution: **Huazhong University of Science and Technology**

Participants: **Linchuan Li, Yao Zhang, Chengdong Ge**

Instructor: **Xiao Kan**

Design Introduction

With the advent of fast-growing digital, information, and network technologies and the desire for a more convenient lifestyle in recent years, users have higher security expectations for electronic systems. Additionally, e-Business, ATM, access control, and intelligent cards all require a safe and easy-to-use identification technology. The traditional identification method, user ID plus password, cannot satisfy users' needs due to various defects such as forgotten passwords, hacker attacks, and theft. Fortunately, identification technology based on biometric characteristics of the human body offers an efficient solution.

This technology uses human physiological features and behaviors to identify a user's ID, and it is more secure, reliable, and human-oriented. Common biometric characteristics used for identification include fingerprint, palm print, iris, face image, voice, handwriting, DNA, etc. Considering the accuracy, durability, convenience, and cost, fingerprint identification technology features a high benefit/cost ratio, security, maturity, and widespread applications. Statistically, fingerprint identification products account for over 90% of the total biometric identification systems in China.

As microelectronics technology advances, programmable logical controllers are becoming more diversified, faster, and more powerful. Today, many FPGA devices support embedded soft-core processors to facilitate the development of FPGA-based hardware. For example, Altera's Nios® II processor, a RISC CPU soft core, features pipelining and a single instruction flow. Designers can embed it in an FPGA and leverage custom logic to build a FPGA-based system. Compared to an embedded hard core, a soft core is more flexible. Additionally, the fast FPGA meets the speed requirements of a fingerprint identification system.

Based on a separated identification system in verification mode, this design authenticates or registers users after they state their ID (i.e., by inputting an ID) and input their fingerprint at a terminal. It also allows a host to manage multiple terminals via the network, and allows an administrator to administrate the system.

Target Users

By integrating other functional service modules, the system can serve as a public service system where the fingerprint becomes the key ID authentication tool. Additionally, the system can be used as an HR management tool or security protection product. The design could be used in the following applications:

- *e-Business*—Credit card consumption, e-buying network
- *Banking*—ATM
- *Enterprises and institutions*—HR management
- *Security administration*—Access control system
- *Qualification*—Examinations

The system's host/slave network mode efficiently implements separated authentication and centralized management, making it suitable for partial authentication. If we improved the communication efficiency and security, it could be extended into a larger system.

Nios II Advantages

Most traditional fingerprint identification technologies depend on a PC or digital signal processor. However, image processing on PCs is expensive, has low speed, and requires a large storage space. Digital signal processors are not flexible enough due to the function and parameter limitations. FPGAs are advantageous for use in fingerprint identification because they feature high processing speed, flexibility, low cost, and embedded system portability. As a high-performance and configurable soft core, the Nios II processor has unique features. Designers can use the C language with short development cycles and portable code. Combined with custom hardware logic, they can conduct complex parallel image processing without losing the advantages of using an FPGA. For our system, it is easy to integrate the relevant components and we can easily adjust the relevant image processing parameters. As a result, the system satisfies various performance indicators and users in different conditions.

Function Description

This section describes the functionality of our design.

Scalable Authentication Network

With a host plus terminals mode and a bus-type local area network (LAN), the system can be centrally managed and extended. By adding terminals into the Terminal Management tab, the host administrator can conveniently add new terminals.

Excellent User Interface

We use an LCD and keyboard to facilitate operation.

Fingerprint Collection

The fingerprint collector collects the user's fingerprint and a driver accesses the serial peripheral interface (SPI) to get data. The program has an automatic finger detection function. Three sets of parameters are allocated to account for the skin moisture level when fingerprints are collected. The

system picks best image of the three. Additionally, the fingerprint collector stays in sleep mode and is only activated during fingerprint collection, minimizing power consumption.

ID Verification

The terminal collector collects fingerprint signals, processes images, and extracts fingerprint minutiae information. After registration, the host acquires the fingerprint and corresponding ID information from the terminal and stores them in a fingerprint database. Upon login, the host returns the corresponding fingerprint information based on the ID, and the slave compares and displays the corresponding login information.

Information Management

The host is a powerful PC that operates a set of management programs, including:

- *User account management*—View or modify registered users.
- *Terminal management*—Add new terminals or modify the terminal priority.
- *Log review*—Review the system access log.
- *Password change*—Change the administrator's login password.

Performance Parameters

Performance parameters include the fingerprint image processing speed and accuracy.

Fingerprint Image Processing Speed

Table 1 shows the time used by the main processes and the total time required for fingerprint image processing before and after hardware acceleration (for the configuration and hardware acceleration principles, see "Design Methodology" on page 288). We use a 256 x 300 8-bit grayscale image as the object that is processed. Using hardware acceleration for image processing greatly improves the processing speed.

Table 1. Image Processing Speed

Operation	Time Required before Hardware Acceleration (Seconds)	Time Required after Hardware Acceleration (Seconds)
Image filtering	36.40	4.77
Ridge thinning	13.54	2.67
Total	54.93	11.57

Fingerprint Identification Accuracy

Because the system's fingerprint image processing and comparison algorithm are designed for a special fingerprint collector, we do not use a universal fingerprint database in the test. Instead, we chose about 40 fingerprints of 10 people at random to test the system's fingerprint identification accuracy.

The statistics show that the system's false accept rate (FAR), i.e., the probability of mistaking non-identical fingerprints as identical fingerprints, is less than 5%. The false reject rate (FRR), i.e., the probability of mistaking identical fingerprints as non-identical fingerprints, is less than 20%.

Identification accuracy is greatly influenced by the skin's cleanliness and moisture, these results are the system's comprehensive fingerprint identification performance and are not from a separate algorithm performance test.

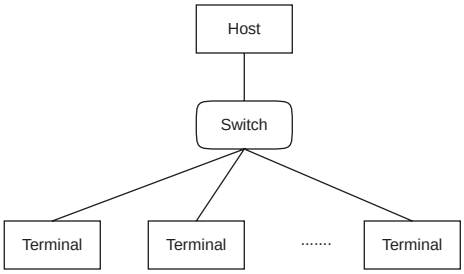
Design Architecture

This section describes the design architecture, including the network topology, modules, hardware, and software design.

Network Topology

Figure 1 shows the network topology. In the system, a switch is the central node that connects the terminals and the host.

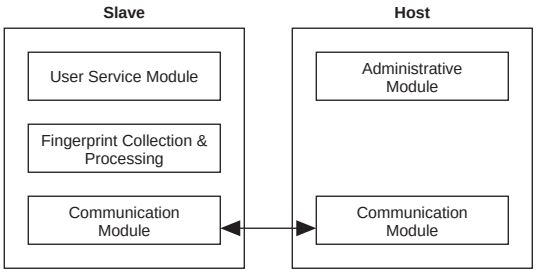
Figure 1. Network Topology



Module Division

Figure 2 shows the modules in the design.

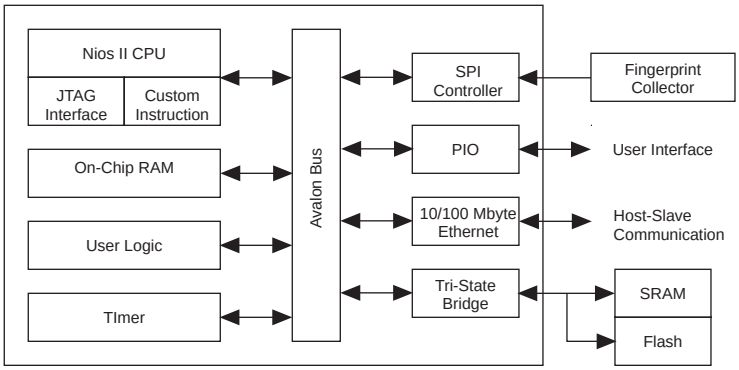
Figure 2. Modules



Hardware Design

Figure 3 shows the hardware design.

Figure 3. Hardware Design



Software Design

Figures 4 through 7 show the flow charts for the system.

Figure 4. Slave Software Flow Chart

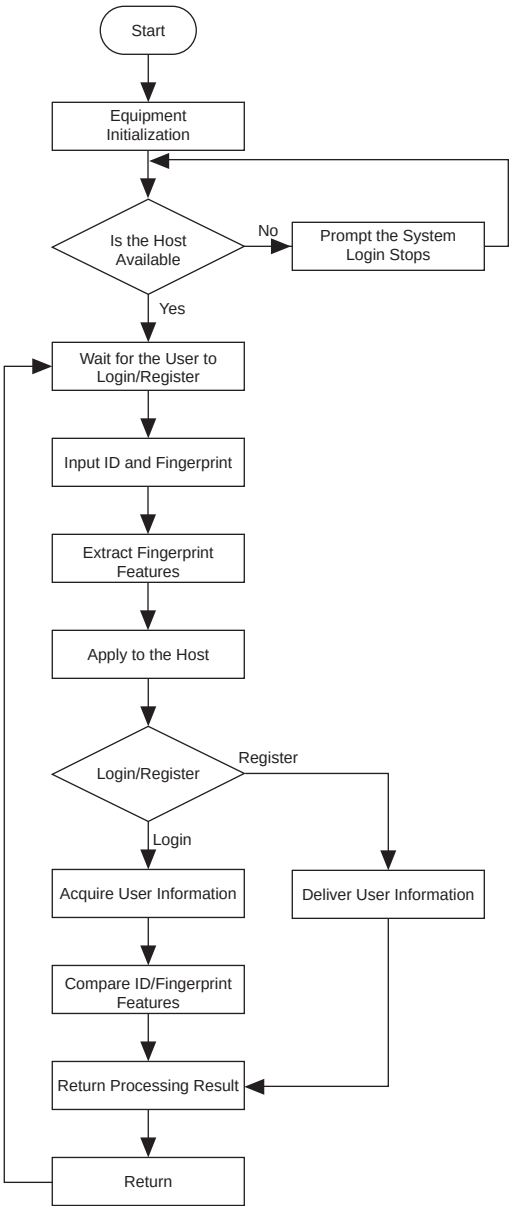


Figure 5. PC Host Flow Chart

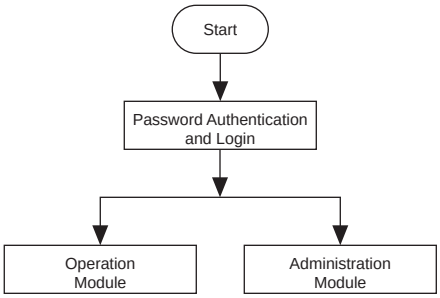


Figure 6. Operation Module Flow Chart

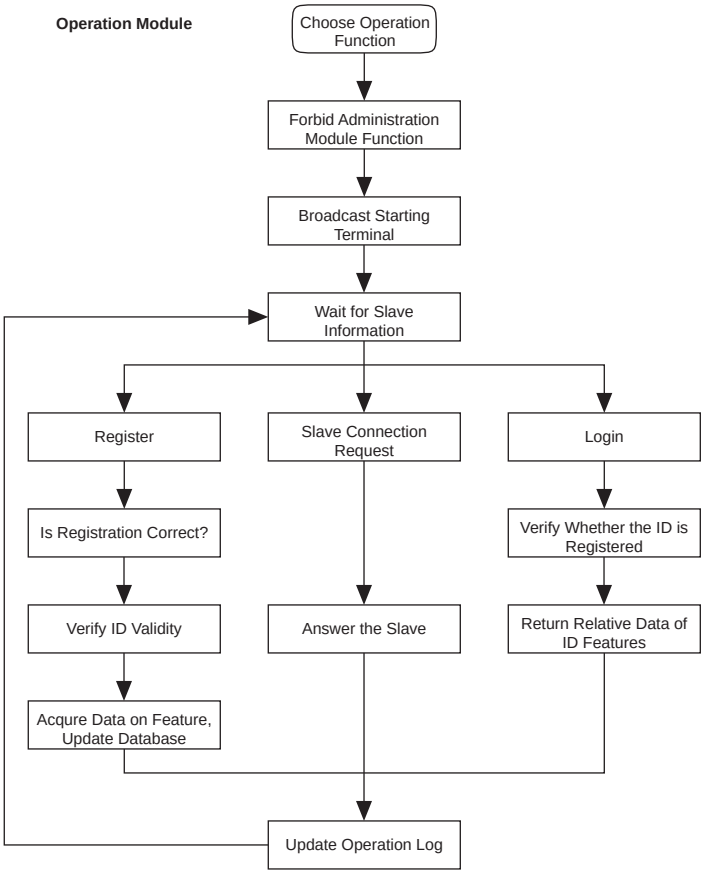
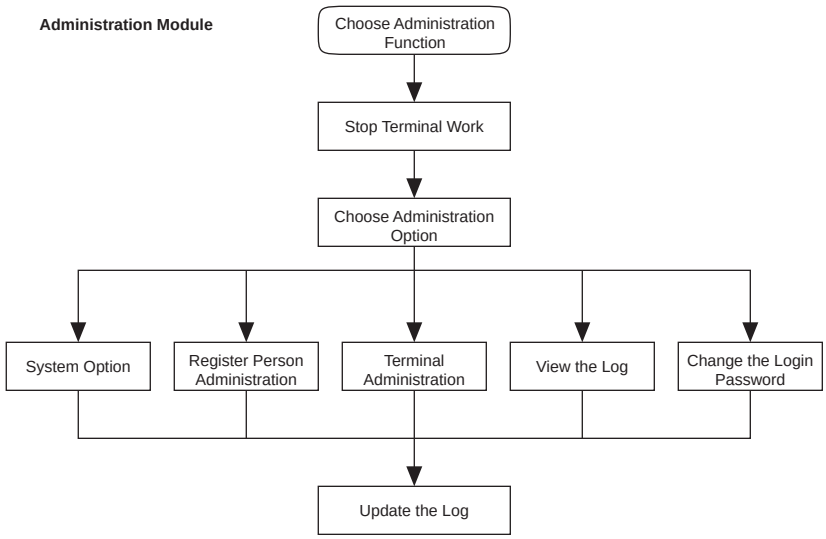


Figure 7. Administration Module Flow Chart



Design Methodology

This section describes our design methodology.

System Development Flow

Most of design's functions are implemented using the Development and Education (DE2) board and a peripheral fingerprint collection circuit. We implemented the system's functional modules, assembled the modules, and debugged the system as described below:

- Referring to examples and test documents provided with the DE2 development board, we implemented the Nios II processor, performed C language programming in the Nios II Integrated Development Environment (IDE), performed on-line program debugging, and loaded the program onto the board.
- Using SOPC Builder, we created access to the peripheral memory, RS-232 serial port, DM9000A network interface, and programmable I/O (PIO) interface on the DE2 board.
- We implemented access to the 4 x 4 keyboard using the expanded PIO interface on the DE2 board.
- We used SPI on the DE2 board to read the fingerprint collection data.
- We used custom peripherals and instruction hardware acceleration on the DE2 board to complete the fingerprint processing algorithm.
- We purchased a fingerprint collection chip, which we used to design and implement the fingerprint collection circuit, automating the finger detection function.
- We developed the administrator program for the PC, and debugged the system.

Hardware

The following sections describe the method we used to create the hardware system.

Nios II Soft Core Configuration

We configured the Nios II processor as described below:

- The Cyclone® II EP2C35 FPGA on the DE2 board is the control chip, and we designed all functions based on it. The hardware represents highly-integrated system-on-a-programmable-chip (SOPC) design ideas and principles.
- We used the 50-MHz Nios II/f fast CPU to support JTAG level 3.
- We used a 12.5-MHz SPI core to transfer fingerprint data.
- We added the DM9000A Ethernet control chip and implemented an Ethernet physical layer.
- We added the JTAG debug module to facilitate on-line system debugging.
- We used a 38,400 bps serial port communication module. With a UART, we can transfer data between the Nios II system and the PC before fully implementing network communication and monitor the fingerprint image processing procedures on the PC.
- SRAM, SDRAM, and flash memory are used to implement the program operation memory, data distribution space, and program writing space, respectively.
- The design has a tri-state bridge connection between memories.
- Two timers implement the system's delay requirements and test the time that the Nios II system requires to process the fingerprint data.
- Keys, nixie tubes, an LED, and a 16 x 2-character LCD interface facilitate the user interface.

Using SOPC Builder, we conveniently built a tailor-made, configurable system that met our requirements.

Making the Fingerprint Collector

We designed the fingerprint collector as described below:

- *Functional design*—We used the fps200 fingerprint collection chip to collect the original fingerprint image data. The circuit has an automatic finger detection function and only instructs the CPU to accept data when a finger is detected. The image data is transferred to the Nios II processor via the SPI.
- *Schematic diagrams*—After studying the fingerprint collection chip data sheet, we designed the schematic diagram to implement the SPI and preserve the microcontroller (MCU) and USB interfaces.
- *PCB schematics*—We used a two-side PCB for the final circuit board.
- *Circuit board welding and debugging*—Our experiments verified the functions of the fingerprint collector, data collection, and data transfer. We adjusted the parameters related to fingerprint collection as needed.

Fingerprint Image Processing Hardware Acceleration

When the Nios II CPU is configured as fast and the fpoint operation instruction is added, the image processing algorithm using C requires 50 seconds (see "Performance Parameters" on page 283 for more details). This speed is acceptable for a real-time processing system.

Two methods can improve the image processing efficiency: implement a digital signal processing (DSP) module on the FPGA or use hardware acceleration. DSP can process images more efficiently without depending on the CPU operation speed, but it is difficult to implement. In contrast, it is easy to use hardware acceleration for image processing. Therefore, we decided to use hardware acceleration, which greatly improved the algorithm performance efficiency.

The fingerprint image processing algorithm has unique features: it requires a large number of repeat operations and identical pixel processing procedures. Therefore, if we improved the performance efficiency of the most fundamental operations, we could improve the performance efficiency as a whole. We divided the fingerprint processing procedures into pattern finding, image filtering, binarization, ridge thinning, minutiae location, etc. (see "Software" on page 290 for more details), and calculated the time required for each procedure. We determined that two procedures should be accelerated: image filtering (36.4 seconds) and ridge thinning (13.5 seconds).

To perform image filtering, we take the data of 52 pixels around the target pixel, multiply the data with corresponding filtering coefficients, and accumulate the results to use as the new value of the target point. In the procedure, the filtering coefficient is taken from a coefficient array with relevant directions. The complete software procedure requires 52 multiplication accumulations. We designed a custom instruction `CI_multi_accumilate`, to complete one multiplication accumulation in three clock cycles. After the acceleration, the image filtering procedure uses only 4.77 seconds.

To perform ridge thinning, we take the data of the target pixel and 15 pixels around the target pixel and compare them with 16 templates to decide whether the target pixel should be removed. After scanning the whole image many times, the fingerprint ridge is thinned into a single pixel width. The complete software procedure requires 16 comparisons. We designed two custom peripherals, `prematch` and `user_delete`, to complete the 16 comparisons in six clock cycles. After acceleration, the ridge thinning procedure uses only 2.67 seconds.

Ethernet Implementation

In the system, a switch is the central node that connects the terminals and the host. Because there is not a large volume of data and no complex routing in the communication between the terminals and the host, the system does not have a physical connection with the public network. However, an embedded real-time operating system (RTOS) must implement a good IP protocol. It is unnecessary for our slave system to use an operating system for dispatching; therefore, we did not use the TCP/IP protocol to build the network. Instead, we used a MAC-to-MAC method for physical addressing, that is, the system directly sends an original data packet that adds source and target MAC addresses as headers.

Software

The software design includes the Nios II program and PC program. The PC program is the fingerprint authentication administrator program that fulfills functions such as responding to login/register requests and system administration.

The Nios II program includes the initialization, fingerprint collection, image processing, host-slave communication, user interface, etc. modules. By organically combining the modules, we created a simple, client fingerprint authentication program as described below:

- *System initialization*—After the system is powered-on, all modules must be initialized. In particular, the network adapter must be initialized to connect the Nios II processor to the PC. Initialization makes the fingerprint collector go into a low-power mode and it does not wake up until a fingerprint needs to be collected.

- *Fingerprint collection*—Based on a driver, the fingerprint collector obtains images through the SPI. The procedure must perform automatic finger detection and skin moisture adaptation.

- *Fingerprint processing*—This module provides fingerprint data processing and comparison. It includes two sub-modules: fingerprint image processing and fingerprint comparison.

- *Host-slave communication*—This module implements communication between the Nios II slave and the PC. Its primary transmitting function is to request commands such as register and log-in requests, fingerprint minutiae information, ID information, etc. Its primary receiving function is to obtain the PC's reply, the Nios II control information sent by the PC, the fingerprint template information that the PC returns to the Nios II processor, etc.

- *User interface module*—This module consists of a 4 x 4 keyboard, 16 x 2 LCD, LED, and nixie tube. The LED indicates the current working status of the Nios II system as well as information such as success, failure, and timeout. The nixie tube displays the input ID information, and the LCD displays real-time system information such as the system status and operation prompt.

The LAN transfers original data packets that use source and target MAC addresses as headers; therefore, we use the winpcap (windows packet capture) protocol on the LAN. winpcap is a free public network access system for Windows platforms, and it gives win32 applications the ability to access the network infrastructure. It provides the following functions:

- Captures the original datagram, including the datagram that hosts send/receive and exchanges them on a shared network.

- Filters special datagrams according to user-defined rules before they are sent to the application.

- Sends the original datagram over the network.

- Collects the statistical information in network communication. Our tests and final design demonstrate that winpcap conveniently receives/sends data packets between the PC and Nios II processor and also satisfies our functional requirements.

Fingerprint Image Processing Module

Figure 8 graphically shows the fingerprint processing procedure (see references [1] and [2] for more details).

Figure 8. Fingerprint Processing Procedure

The procedure includes the following elements:

- *Pattern finding*—This process includes two steps: first, it calculates the direction of single points based on the grayscale value of the points around the target point. For simplification, we divide 180° into 8 directions. Next, it obtains a 5 x 5 block pattern using a statistical method and marks the blocks without clear direction about the background. Pattern finding lays a foundation for the direction-based image filtering.

- *Image filtering*—We designed filtering coefficient templates for 8 directions in advance, took 52 pixels of data around the target pixel, multiplied the data with the corresponding filtering coefficient, and accumulated the result, which is used as the new value of the target point. Filtering enhances the image continuity along ridges and improves the image contrast perpendicular to the ridges to segment neighboring ridges.

- *Binarization*—After filtering, the image ridges are distinct. Therefore, we only need to use fixed threshold binarization, i.e., to segment the image into black and white images by taking a fixed gray value as the standard. After binarization, the image can be completely thinned.

- *Ridge thinning*—We use an OPTA (or parallel thinning) method to corrode the ridges gradually until the ridges are thinned into a single pixel width. Ridge thinning facilitates minutiae location.

- *Locating minutiae*—First, we scanned the pattern to locate the central point of the fingerprint. Then, we located the tip points and the split points in the thinned ridge image. To find the points, we assumed that the tip points are black and surrounded by only one black point and spit points are surrounded by three black points.

Fingerprint Comparison

Through many tests, we improved the central point-based fingerprint comparison method into three steps: coarse matching, coordinate transformation, and precise matching.

- *Coarse matching*—The system makes a coarse match for two fingerprint images by taking their central point as the original point and ±30° as the range for the angle of rotation. Then, we ascertain the most matched angle of rotation and some minutiae pairs.

- *Coordinate transformation*—Based on obtained minutiae pairs, we calculate the precise coordinate translation and rotation parameters of the two images.

- *Precise matching*—Based on the obtained coordinate translation and rotation parameters, we precisely match the minutiae a second time and evaluate the degree of matching. The system gives three points to point pairs matched in type (tip or splitting) and location, and gives two points to those matched in location only. If the total score is more than the threshold value, there is a match. If any step fails, it is not a match.

The central point-based fingerprint comparison method efficiently resists the interference caused by image translation. That is, the combination of the final precise matching with the point-to-point comparison method partially resists interference caused by central point location errors. Considering that the minutiae type (tip or splitting) is usually incorrect, we regard cases where the fingerprints have unmatching types but matched location as matching, improving the comparison accuracy.

Design Features

Our design has the following features:

- *Bus topology between hosts and slaves*—Because organizations or institutions may identify fingerprints in different locations and it is impossible to configure a system at each location, we use hosts and slaves to fulfill the task. Slaves collect and process fingerprints and transfer the processed results to the hosts over the network. The hosts provide management and storage. In this way, hosts and slaves have clear duties to leverage their own advantages.

- *Custom instructions to acceleration of key algorithms in hardware*—Extracting fingerprint minutiae is a complex DSP function, and it greatly slows the system speed if software is used for extraction. Fortunately, Altera's SOPC Builder software allows users to create and deploy their own Nios II system as well as add custom instructions. Therefore, the system uses a hardware description language to implement the minutiae extraction algorithm. We used custom instructions to define the IP algorithm as a special instruction that directly invokes processing, implements hardware acceleration, and greatly improves system speed.

- *Ethernet transmission*—We used Ethernet to transmit requests from the slave to the host for processing. Additionally, we used a competitive terminal access mechanism that provides high efficiency when the system load is light. Ethernet ensures the application of the system and allows terminals to be added conveniently in the future.

- *Easy hardware and software upgrades*—It is impossible for an application to fully meet the requirements of each user in a huge user base. However, Altera's SOPC system design solution

can solve the problem, allowing different users to modify the hardware and software easily based on the original system. They can develop new products and improve competitiveness by utilizing FPGA programmability. In the system, we can put the fingerprint minutiae extraction module at terminals or hosts based on the number of user terminals, balancing the communication traffic and host load. In this way, we dramatically improved the adaptability of the system.

- *Benefits in cost, power consumption, portability, and integration*—The FPGA-based system design integrates processor, peripherals, memory, and I/O interface into a single FPGA, reducing the system's costs, complexity, and power consumption. Additionally, because the Nios II processor is superior to hard cores in terms of cost, the system has higher integration and is more cost-effective.

Conclusion

The two-month contest was a process of SOPC development, learning, and application, as well as learning about Nios II embedded processor design. By using the development and design technology, we now understand the technology more. We learned the following things during the contest:

- *Easy-to-use design environment*—It was a challenge for us to gain basic knowledge about software and hardware development and conduct systematic design, implementation, and verification within two months. However, SOPC Builder and the Quartus® II software provided a visual, fast methodology and the Nios II IDE, which helped us achieve our goal in a short time with a fast learning curve.

- *Software/hardware co-design and verification*—Parallel hardware/software design is a crucial task for embedded system design. During the design, the major challenge is synchronizing and integrating software and hardware design. During the contest, we used SOPC Builder to conduct comparatively independent software and hardware implementation with comprehensive system design and the reasonable division of software and hardware. Independent functional verification shortens the development cycle.

- *Innovative Nios II system hardware acceleration*—Compared with traditional DSP solutions, custom instructions or peripherals can be better integrated into the Nios II system, ensuring a fast processing speed and flexible controls. Compared with an ASIC solution, the Nios II processor is more cost-effective because it can provide the same function using internal FPGA resources without adding other devices.

- *Improving Nios II stability*—The issue may not be apparent for small system designs but can occur during complex system designs. For example, random software operation errors occurred during same hardware deployment. The system handled the fault after we recompiled the wiring. The error was often due to an infrastructure I/O driver. In our opinion, the Avalon® bus is not a fixed connection wire but is a set of connection wires generated in the FPGA after Quartus II compilation, so the random wiring leads to bus instability. We did not verify our guesswork, but the problem was a big challenge.

- *Problem solving*—Although many problems are simple now, they seemed insurmountable barriers when we first met them and did not have enough information. We had to consider and collect various data, dare to practice, be persistent, and consequently find a way out of desperation. We made arduous efforts to discover solutions and tackle problems, but we gained knowledge and the methodologies to solve problems in the process, which will aid us in our future study and life.

References

Lingli, Liu, *Preprocessing and Minutiae Extraction of Fingerprint Image*, Master degree thesis of Hunan University, December 2005.

Chunlei, Li, *Research on Fingerprint Identification Algorithm and FPGA-based Hardware Realization*, Master degree thesis of Shandong University, April 2005.

Medical Applications

This section includes projects that can be used as medical applications, including:

MRI Spinal Segmentation Based on the Nios II Processor 299

Nios II Processor-Based Self-Adaptive QRS Detection System 319

Portable Telemedicine Monitoring Equipment 333

FPGA-Based Clinical Diagnostic System using Pipelined Architectures in the Nios II Soft-Core Processor 373

First Prize

MRI Spinal Segmentation Based on the Nios II Processor

Institution: **Information Science Institute, College of Computer and Information Technology, Beijing Jiaotong University**

Participants: **Weiming Li, Ruiqiong Shi, Bo Li**

Instructor: **Xiaoming Ding**

Design Introduction

As fast-developing medical imaging technology promotes modern medicine, computed tomography (CT), magnetic resonance imaging (MRI), and positron emission tomography (PET) are becoming widely used in clinical diagnosis and analysis. These systems have developed from tools for non-invasive examination and anatomical structure visualization to operation planning and simulation, operation navigation, and radiotherapeutic planning and focus change tracking, as well as segmenting the anatomical structure from medical images and forming shapes.

Magnetic resonance (MR) spinal image segmentation research plays a critical role in medical imaging-related computer-aided recognition and neuropathology clinical research. If the acantha cannot be segmented and recognized accurately and clearly, computer technology will provide little to medical clinical research. Additionally, it is not enough to complete this task by hand. Traditional acantha fracture assessment involves manually marking the acantha with six points (on the four corners and the centers of the top and bottom edge), and then measuring the height of the acantha on the front, middle, and back, which is a time-consuming task. According to some documents, using a mouse to locate a single patient's vertebra takes more than 15 minutes, and locating the entire acantha takes an extremely long time. In these circumstances, there is an urgent need of a clinical method that automatically segments the MR spinal image. Locating vertebra and intervertebral disks automatically via computer is important for diagnosis during clinical treatment.

Our design implements a feasible, robust algorithm running on the Nios® II processor platform. The design locates intervertebral disks automatically and quantitatively marks acantha nuclear magnetic resonance (NMR) sagittal views using an MRI spine segmentation algorithm in our lab. Fully using the

Altera® FPGA and Nios II resources makes the system small and portable. This algorithm improves the automatic assessment of vertebra fractures derived from osteoporosis, and enables quantitative analysis of the intervertebral disk to facilitate image alignment with other imaging (e.g., CT scans) and image-guided vertebra operation.

Application Scope and Targeted Users

This design is applicable to medical institutions possessing NMR instruments. Because the Nios II design is easy to use, it is also applicable to the average person, who can learn the patient's situation and remind him or her about acantha diseases in daily life. Additionally, the design's network transmission function facilitates telemedicine.

Nios II Design Advantages

The Nios II processor provides several advantages for our design, including:

- *Innovative system-on-a-programmable-chip (SOPC) design concept*—We can adjust the Nios II soft-core system's performance according to the application to satisfy specific user demands. Compared to a fixed processor, the Nios II soft core system achieves higher performance at lower clock rates. The abundant intellectual property (IP) core library helps users create designs and effectively improve the system's computation capability. Moreover, its user logic functions and custom instructions highlight Nios II technologies, allowing optimized, accelerated computation, increasing processing speeds, and facilitating algorithm commercialization.

- *Development environment*—The Nios II Integrated Development Environment (IDE) integrates the µC/OS-II real-time operating system (RTOS), which has been ported for the Nios II processor. We can use the operating system (OS) directly for functional design and system expansion.

- *DSP Builder*—DSP Builder provides a variety of functional modules and IP cores. With DSP Builder, we can perform algorithm-level system development in the Simulink software. Then, we can design the algorithm as a custom user instruction, and integrate it into the Nios II system using SOPC Builder and the Quartus® II software. We access the DSP algorithm by activating the custom instruction via software.

- *C-to-Hardware Acceleration (C2H) Compiler support*—The Nios II C2H Compiler can automatically convert a C language program that requires high performance into a hardware accelerator, and integrate it into the FPGA-based Nios II subsystem. In this way, the program shares the Nios II processor data computation and memory access and helps the processor perform other tasks better. Because the Avalon® interconnection architecture has no limit on the number of hosts and slaves in a system, the Nios II C2H Compiler can generate multiple memory self-governing hardware accelerators according to the target code conversion requirement. This process helps embedded system developers improve efficiency and implement successful designs.

Function Description

MRI spinal segmentation research is an integral step to providing qualitative and quantitative analysis of body tissue with MRI devices. The introduction of computer-aided MRI spinal segmentation technology will make clinical diagnosis more accurate and timely, lower medical expenses, and alleviate doctors' pressure. Therefore, this technology has a bright future.

The design's hardware module consists of the Development and Education (DE1) development platform, an MRI LCD panel, and the network interface panel for the MR device and PC. The system's functional modules include the MRI image preprocessing module, spinal cord extraction module, acantha detection segmentation module, LCD image display and human-machine interaction module, MRI image data access module, and the network transmission module.

- *MRI image preprocessing, spinal cord extraction, and acantha detection segmentation modules*—These modules are part of the design's algorithm and core functionality. Because of the Nios II

processor's computational capabilities, we implemented most algorithms using C programs. The C2H Compiler accelerates the most time-consuming part of the algorithm to shorten the development cycle.

- *LCD image display and human-machine interaction module*—We implemented the image display and human-machine interaction using an LCD display and mouse, which are connected to the Nios II processor as an Avalon slave using a compiled IP core. In the design, we used µC/OS-II in the IDE to deploy the system's task management and algorithms. We adopted the µC/GUI corresponding to the TCB8000C LCD controller chip to enable the system's human-machine interaction.

- *MRI image data accessing module*—Generally the images obtained directly from the imaging device comply with the standard medical image format, i.e., dicom format. Images in this format are not commonly used by ordinary users, and must be read using special software or medical instruments. To allow the images to be read conveniently in most circumstances, we convert them from dicom to bitmap (bmp) before the processing. The images downloaded from the network to the hardware platform's storage devices are in bmp format, and are uploaded to the PC after processing. The images require a large storage space. Therefore, we use a secure digital (SD) card to store the data and migrate the µC/FS file system corresponding to the card to enhance the system's expandability and data management capability.

- *Network transmission module*—We implemented data interaction using an Ethernet interface. The Ethernet interface allows the system to obtain images from the MR device. We developed the network interface panel using the extensive DE1 interface and used the DM9000A chip as the core. Using Ethernet makes the system more scalable.

With the Nios II processor features and SOPC design concepts, our design became a process of optimizing an algorithm, improving the design, and accelerating system operation. We implemented the design in two steps:

- Established the Nios II system to run the algorithm in software and implement the fundamental system functions.

- Accelerated the algorithm operation using a custom instruction and peripherals.

Performance Parameters

Spinal NMR imaging plays a critical role in diagnosing spinal diseases. For example, it is more effective than other imaging methods in describing degenerating intervertebral disks and can assess the curative effect of acantha operation. Analyzing and processing spinal NMR images will make the diagnosis more accurate, and save time and cost. Therefore, for this system, the most important thing is to segment and recognize the acantha accurately and clearly from the image. Additionally, we need to accelerate the calculation speed by optimizing code, employing C2H tools, and creating custom user peripherals.

Figure 1 shows the system's resource usage after compilation.

Figure 1. Resources Used

Flow Status	Successful - Sat Oct 06 15:37:37 2007
Quartus II Version	6.1 Build 201 11/27/2006 SJ Full Version
Revision Name	DE1_SD_Card_Audio
Top-level Entity Name	DE1_SD_Card_Audio
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	No
Total logic elements	8,593 / 18,752 (46 %)
Total combinational functions	7,603 / 18,752 (41 %)
Dedicated logic registers	4,769 / 18,752 (25 %)
Total registers	4886
Total pins	273 / 315 (87 %)
Total virtual pins	0
Total memory bits	79,360 / 239,616 (33 %)
Embedded Multiplier 9-bit elements	28 / 52 (54 %)
Total PLLs	2 / 4 (50 %)

After code optimization, the time required for all algorithm steps to operate is:

- Image preprocessing: 4,128.77 ms
- Spinal cord extraction: 24,208.14 ms
- Disk detection: 6.03 ms
- Image storage on the SD card: 15,881.59 ms
- Image reading from the SD card: 5,730.02 ms

The algorithm segmentation accuracy can be seen in the final processing image that is generated after the system operation. It proves that the system can mark the acantha by the number on the platform precisely. The medical instrument performance can be verified through long-term clinical experiments, during which we can improve our algorithm.

Figure 2 shows the processing effect of a design. Figure 3 shows the image that the system transmits to the computer via the network.

Figure 2. Design Processing

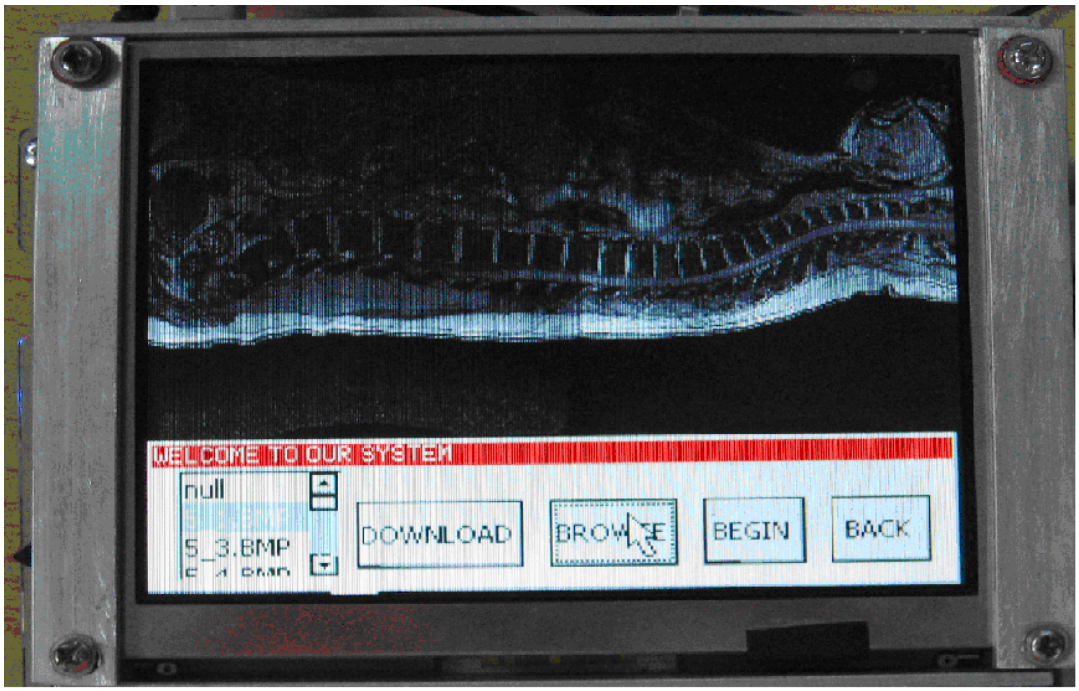
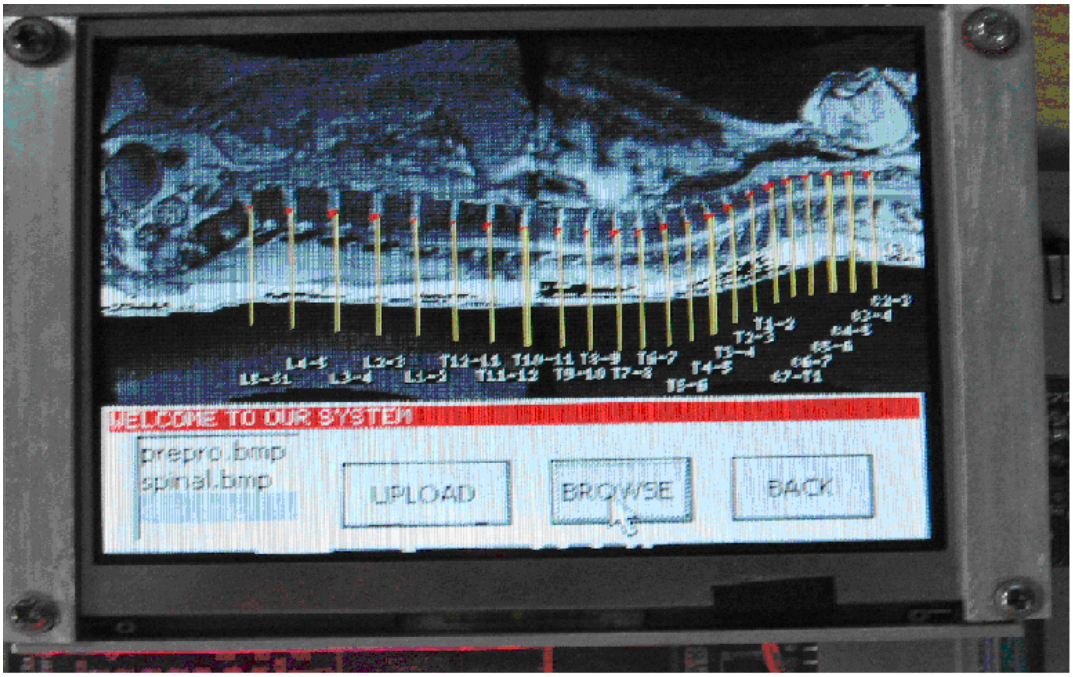


Figure 3. Image Transmitted to Computer



Design Architecture

Figure 4 shows the hardware block diagram of the MRI spinal segmentation system.

Figure 4. MRI Spinal Segmentation System Block Diagram

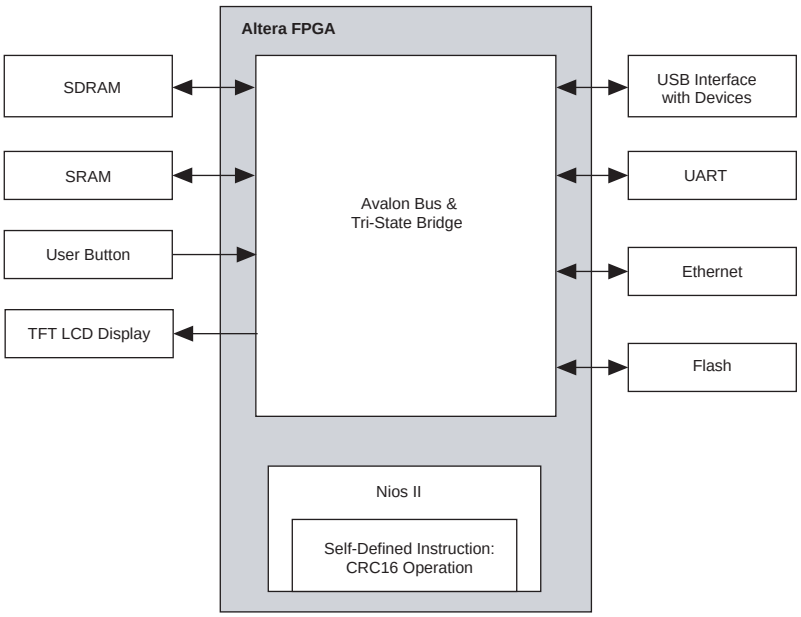
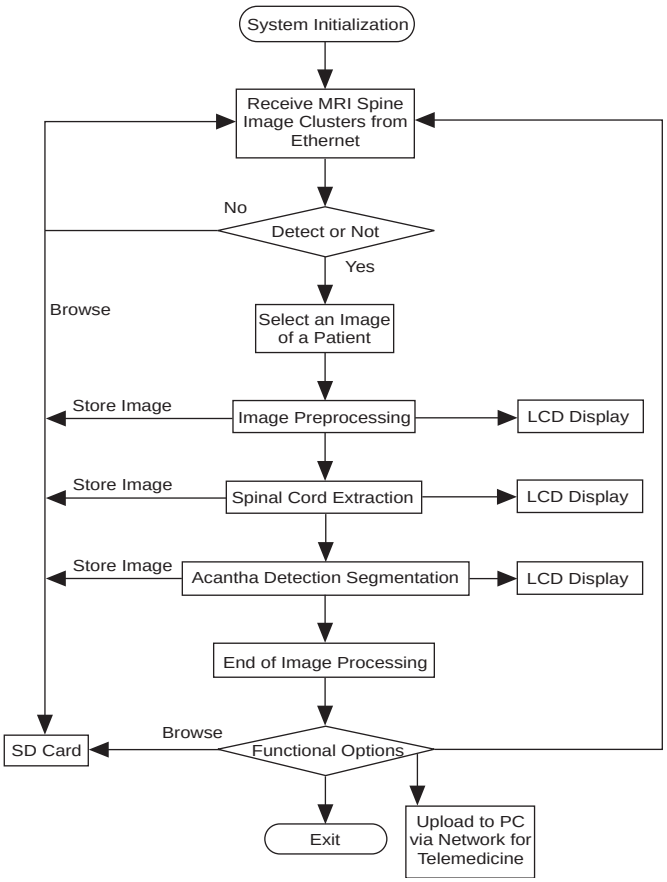


Figure 5 shows the software flow chart when the system initialization receives MRI spine image clusters from the Ethernet.

Figure 5. Software Flow Chart



Design Methodology

This section describes the system hardware and implementation method.

System Hardware

Figure 6 shows the design in SOPC Builder.

Figure 6. SOPC Builder

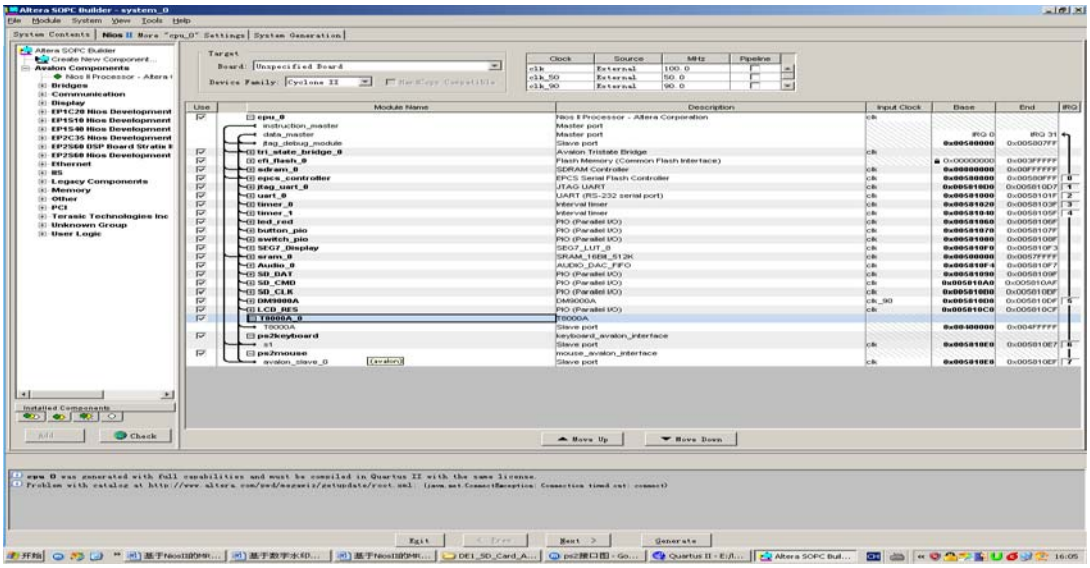
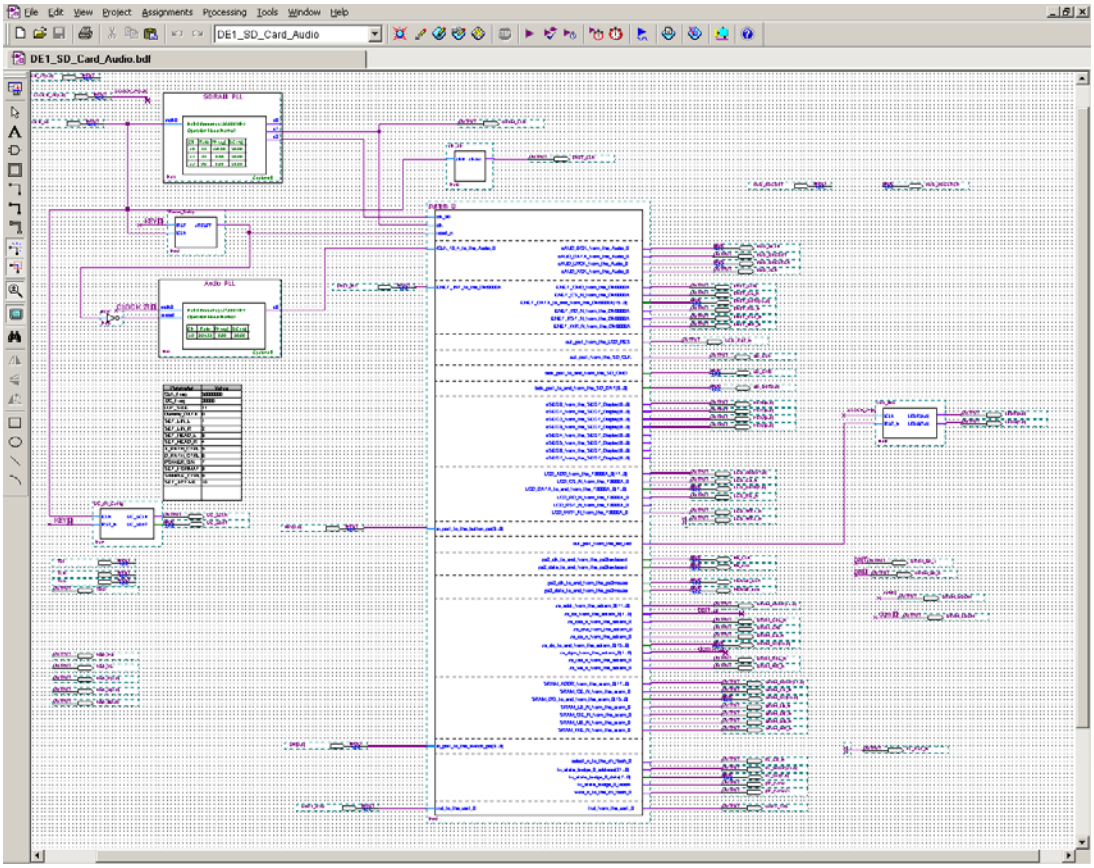


Figure 7 shows the Quartus II schematic diagram.

Figure 7. Quartus II Schematic Diagram



Implementation Method

This section describes the implementation for the various design modules.

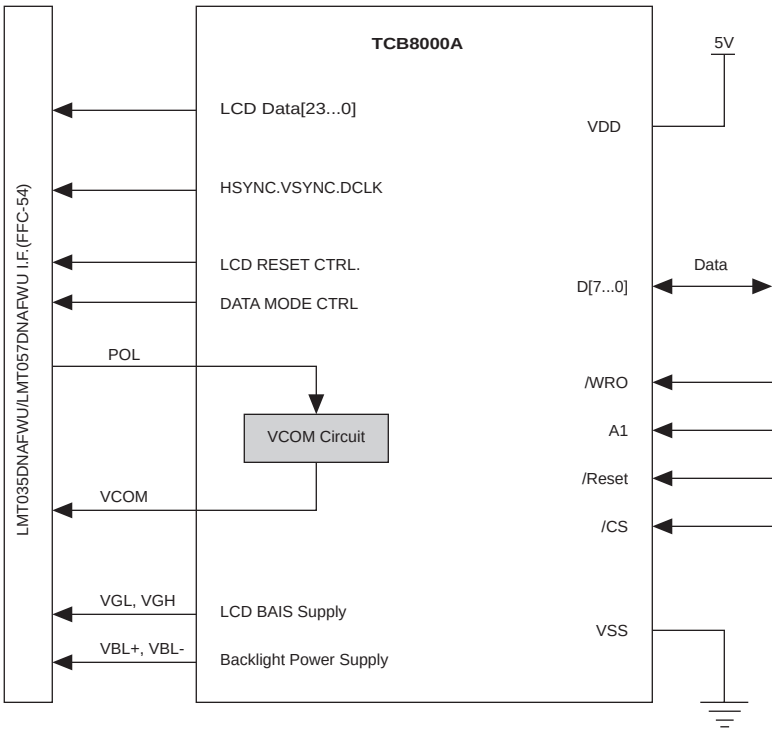
LCD Image Display and Human-Machine Interaction Module

The following sections describe the LCD and mouse hardware implementation as well as migration using µC/GUI.

LCD and Mouse Hardware

We used the TCB8000C device as the LCD controller chip to control a 5.7-inch TFT65000 LCD display. Figure 8 shows the controller and microcontroller (MCU) interfaces. The controller is connected to the Avalon bus as a slave, and the Nios II processor directly accesses the LCD controller. Therefore, the driver can be compiled easily and driving the LCD after GUI migration is simplified.

Figure 8. LCD Controller



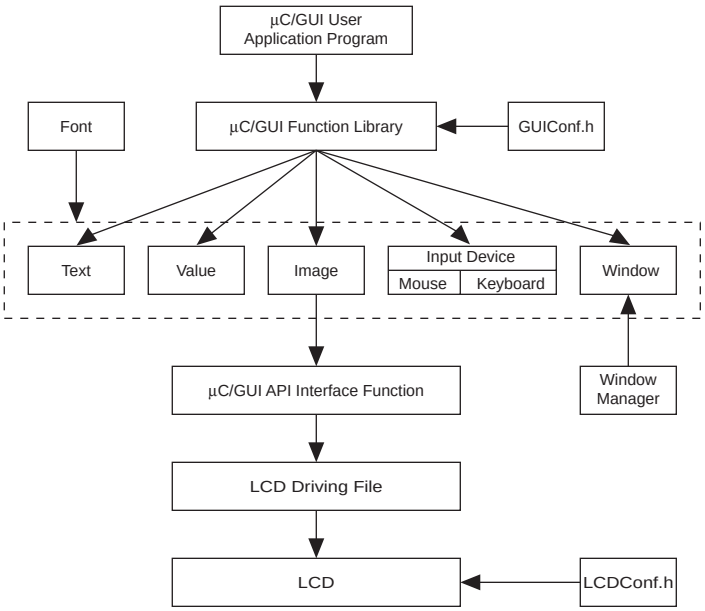
The system employs a PS/2 interface mouse, which has a two-way synchronous serial protocol, i.e., a pulse on the clock line is followed by one-byte data to the data line. For connection with the Nios II processor, we used Verilog HDL to compile the Avalon slave implementing the PS/2 mouse transmission protocol. The Verilog HDL programs include:

- **mouse_avalon_interface.v** is the Avalon bus slave interface.
- **mouse_register_file.v** and **mouse_avalon_interface.v** convert the data and transmit the PS/2 mouse protocol, compile the corresponding driver in the GUI, and implement the GUI mouse operation.

µC/GUI Migration

µC/GUI is an excellent graphic software for embedded systems. It features open source code, portability, clipping, stability, and reliability. µC/GUI provides rich interface elements, such as buttons, edit boxes, sliders, etc., and also supports an efficient window deployment mechanism for providing connectivity between interfaces and applications. The human-machine interaction interface of the small multi-functional digital photo frame system is developed using this tool. Figure 9 shows the µC/GUI software system structure.

Figure 9. µC/GUI Software System Structure



The µC/GUI functions library provides a GUI interface for user programs, including text, value, 2-dimensional (2-D) view, input device, and various window objects. The input device can be a keyboard, mouse, or touch screen. The 2-D view contains pictures, beelines, polygons, circles, ellipses, arcs, etc. Window objects include buttons, edit boxes, progress bars, check boxes, etc.

The user can configure the µC/GUI functions library with the **GUIConf.h** file, including whether to use a memory or window management device, whether to support an OS or touch screen, selecting the dynamic memory size, etc. Additionally, the **GUIConf.h** file defines various hardware-related attributes, such as LCD size, color, and interface functions. The LCD driver interprets µC/GUI functions into the liquid crystal interface function defined in the **GUIConf.h** file, and is not dependent on the hardware connection. Using a driver, the µC/GUI-LCD hardware interface converts the hardware interface function into an LCD read/write function defined in the **GUIConf.h** file.

Migrating µC/GUI involves:

- *Migrating the configuration file*—First we migrated the **GUIConf.h** file and then the **LCDConf.h** file. We configured the relevant configuration file parameters according to the digital album system's display module requirements.

- *Migrating the LCD driver*—µC/GUI provides drivers for different LCD controllers. For example, the KS0713, SED1335, and T6963 controllers have corresponding LCD drivers. However, our system's display module is the TOPWAY TCB8000A LCD controller with a TFT 65,000 color LCD screen, and µC/GUI does not provide a driver for it. Furthermore, unlike the LCD controllers for which µC/GUI provides drivers, the TCB8000A controller has an independent screen control instruction system. These factors made it difficult to migrate the controller for µC/GUI.

During migration, we first used the TCB8000A instruction system to let µC/GUI provide the most application program interface (API) functions for the upper application function. Then, for API functions that the TCB8000A controller does not support in hardware, we repaired the drivers using software. Last, by testing a large quantity of controls, we adjusted the TCB8000A display instruction parameters used in the LCD drivers, optimizing the LCD driver performance and seamlessly migrating µC/GUI for the TCB8000A controller.

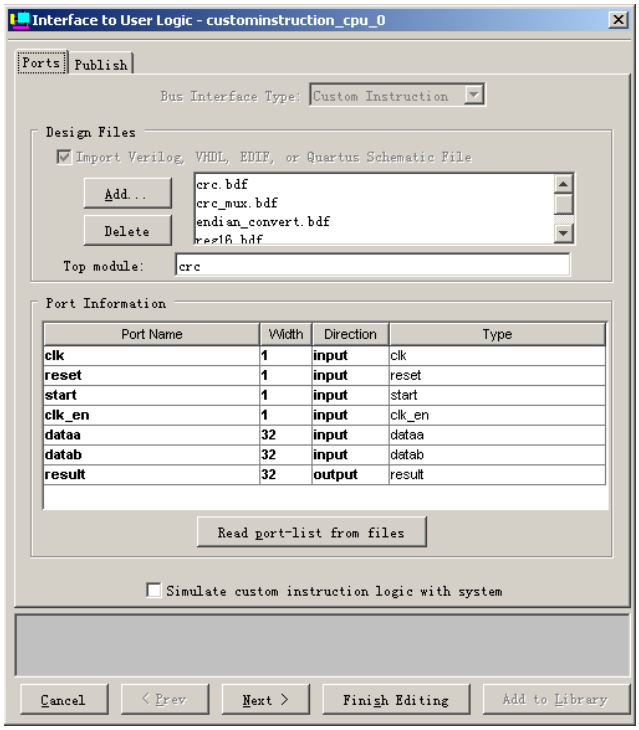
MRI Image Data Access Module

SD card technology was co-developed by Panasonic, Toshiba, and SANDISK. As a totally open standard (or system), SD cards are used in MP3 players, digital video cameras, digital cameras, e-books, audio visual (AV) appliances, etc., and particularly in ultra-slim digital cameras. SD cards are the same as multimedia cards (MMC) in shape, but are a little thicker than MMCs and have more capacity. Additionally, SD cards are compatible with MMC interface specifications. The nine-pin SD card can change the transmission mode from serial to parallel, improving the transmission speed. It reads/writes faster than MMCs in a more secure mode. To make the system more applicable and compatible, we decide to use an SD card as the primary storage media to store photos, music, material, etc.

µC/OS-II is contained in the Nios II software IDE, and users can apply it easily in their own software engineering. To facilitate concurrent task processing and CPU sharing, we used µC/OS-II with a file system. At the beginning, we selected the zlg file system, but its speed was unsatisfactory during testing. Reading/writing 1 Mbyte data into the SD card required 37 and 57 seconds, respectively. When we analyzed the situation we found that the DE1 board's one-line SD card (i.e., it only has one data line) reading functionality is significantly limited in speed. We added data lines, changing the read/write mode to four lines, which makes it more compliant with the system demands.

Testing showed that the speeds were improved to 17 and 27 seconds, respectively. However, these improvements were not quadruple the original speed as we expected. After performing an Internet search, we found that zlg/fs was low performance and consumed too much time. Therefore, we decided to use Micrium's µC/FS, which has excellent compatibility with µC/OS-II and high performance. After several weeks, we successfully migrated µC/FS version 1.34 to the DE2 platform to establish the file system for the four-line SD card. According to our comparison test, the read/write speeds were greatly increased, requiring only 3.6 and 11 seconds, respectively for 1 Mbyte of data. Therefore, this implementation essentially satisfied our speed requirement for accessing data files. Writing required more time because when data is written into the SD card, each block calculated a 16-bit cyclic redundancy code (CRC), which takes some delivery time. To improve write speeds, we accelerated the CRC16 computation with a Nios II custom instruction. With this method, we reduced the time it took to write an MR image (about 1.5 Mbytes) to the SD card from 21.7 to 15.8 seconds. With a higher speed SD card, the write times will be even faster. Figure 10 shows the custom instruction in SOPC Builder.

Figure 10. CRC 16 Custom Instruction in SOPC Builder



We used the following signals:

- CLK—Host to card clock signal.
- CMD—Bidirectional command/response signal.
- DAT0 through DAT3—4 bidirectional data signals.
- VDD, VSS1, and VSS2—Power and ground signals.

Figure 11 shows the data packet formats for one-line and four-line SD card read/writes. Figures 12 and 13 show the single block read and write, respectively.

Figure 11. Data Packet Format

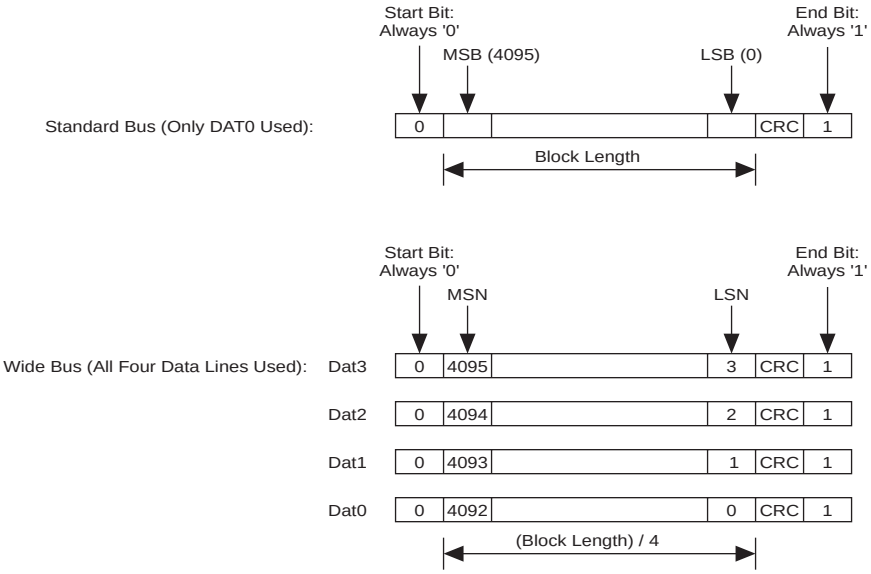


Table 1. Timing Diagram Symbols

Abbreviation	Definition
S	Start bit (= 0)
T	Transmitter bit (host = 1, card = 0)
P	Single-cycle pull-up (=1)
E	End bit (= 1)
Z	High impedance state (-> - 1)
D	Data bits
X	Don't care data bits (from card)
+	Repetition
CRC	Cyclic redundancy check bits (7 bits)
	Card active
	Host active

Figure 12. Single Block Read Timing

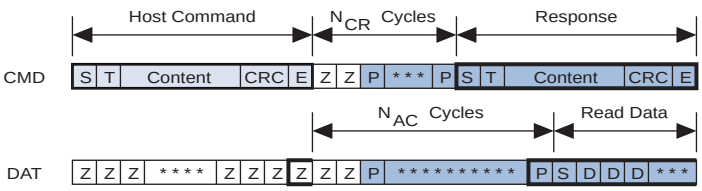
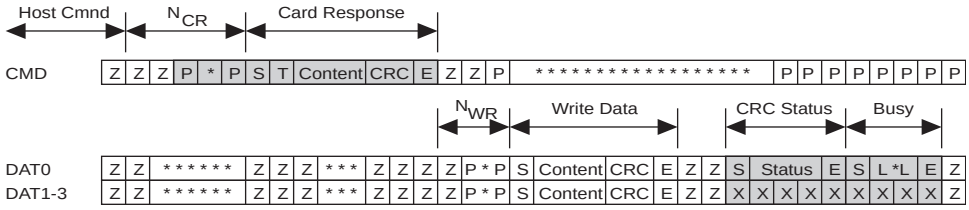


Figure 13. Single Block Write Timing



Network Interface

Because the DE1board does not have a network interface, we designed a network interface board that connects to the DE1 interface to enhance the system's scalability and add network capabilities. Because the board's I/O interface is limited and we also needed to connect the LCD display through I/O, we used a DM9000A network chip that has fewer I/O. Additionally, the DE1 board transmits data at a fixed time while the PC receives data upon inquiry. To use the network, a µC/OS-II driver implements data transmission and data is received with interrupts on the second network layer.

Image Processing Algorithm Implementation

Typically, bodily acantha contain 24 pre-sacral acanthae, including 7 cervical acanthae, 12 thoracic acanthae, and 5 lumbar acanthae. Therefore, in our acantha sagittal views, 23 intervertebral disks are visible in typical images, specifically, C2-3, C3-4, C4-5, C5-6, C6-7, C7-T1, T1-2, T2-3, T3-4, T4-5, T5-6, T6-7, T7-8, T8-9, T9-10, T10-11, T11-12, T12-L1, L1-2, L2-3, L3-4, L4-5, and L5-S1.

The image processing algorithm quantitatively marks the visible intervertebral disks, as well as predicting and marking unclear intervertebral disks. We implemented the algorithm using a C program compiled under µC/OS-II and using µC/GUI and µC/FS, we implemented the MRI acantha image segmentation system with good human-machine interaction and facilitated the system operation.

The algorithm has three steps:

- Image preprocessing
- Spinal cord extraction
- Disk detection

Image Preprocessing

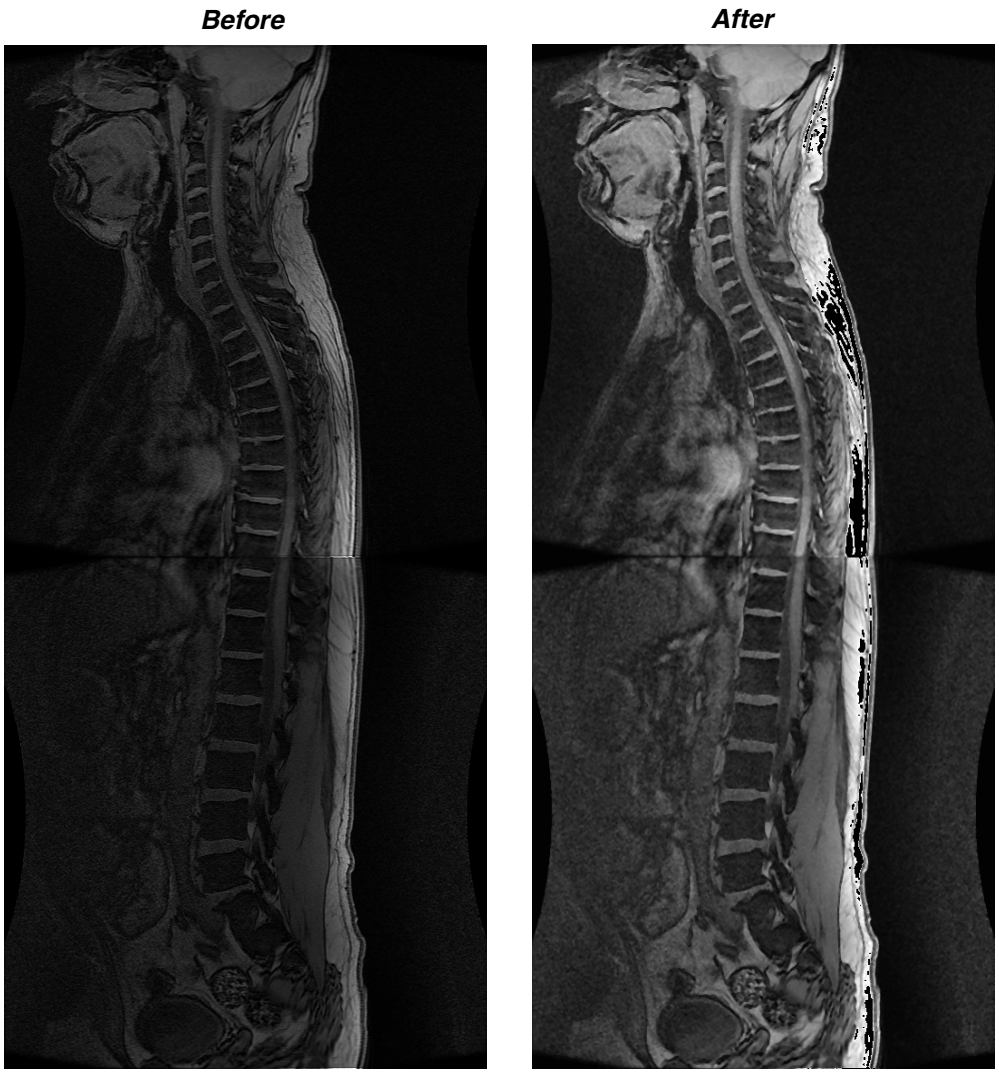
Because the original NMR images have low contrast with unclear visual effect, we first improved the image quality and the visibility of the intervertebral disks. Figure 14 shows the contrast before and after preprocessing.

The image median filtering is time-consuming, but we were able to speed it up using the C2H Compiler to accelerate the algorithm. Table 2 compares the image processing time requirements.

Table 2. Image Processing Time Requirements

Requirement	Before C2H Acceleration	After C2H Acceleration
Time for median filtering	10,077.35 ms	2,811.41 ms
Total time for image preprocessing	11,394.71 ms	4,128.77 ms
Logic resources used	5,446/18,752 (29%)	8,593/18,752 (46%)

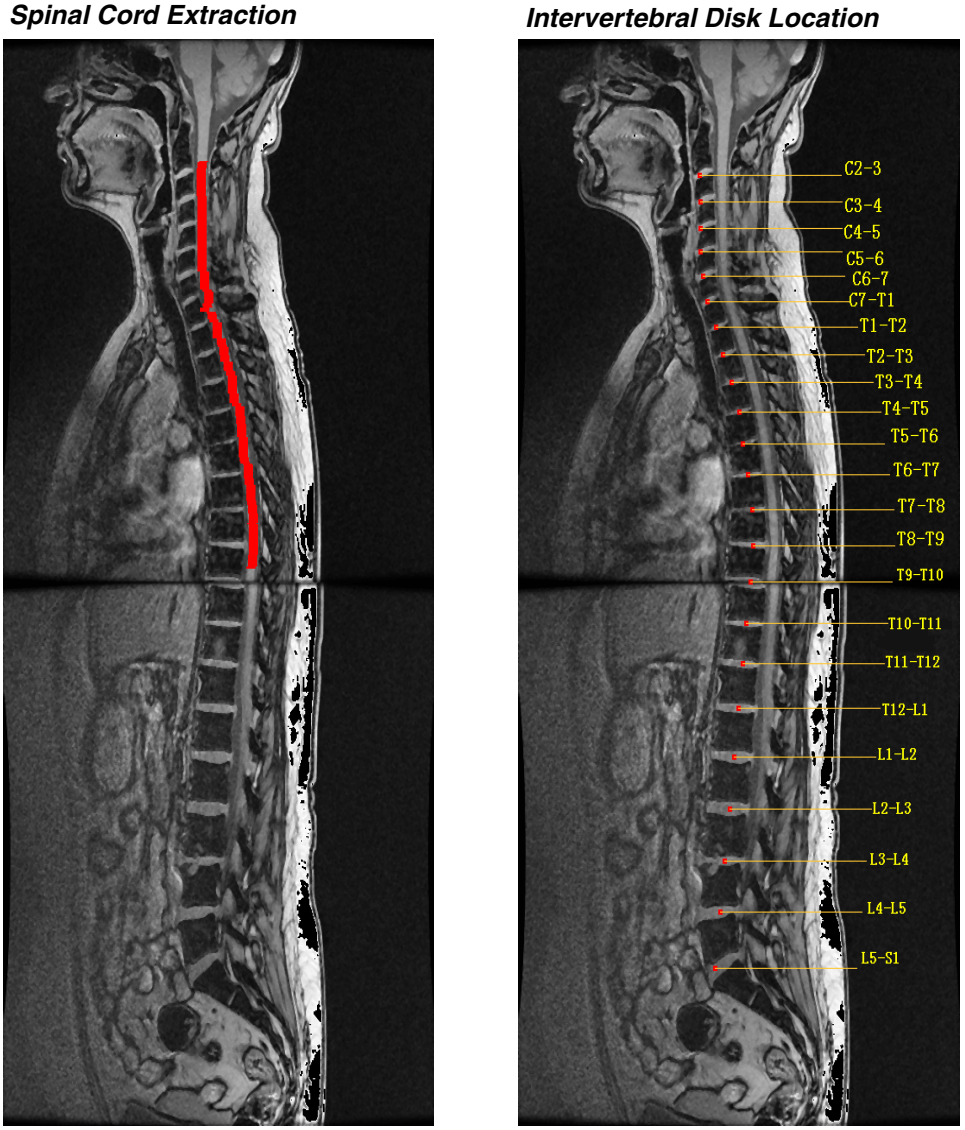
Figure 14. Sample Image Before and After Preprocessing



Spinal Cord Extraction

The spinal cord is conspicuous in the spinal NMR image, and provides directional information for the location of the intervertebral disks. We can extract the spinal cord using fuzzy match in statistical mode recognition. In our design, we only extract the spinal cord for the upper body because this curve is relatively complicated. We can then predict the lower body curve accordingly. The red line in the left image in Figure 15 is the spinal cord extracted from the upper body. Compared with other parts of the algorithm, this part is time-consuming. Given the current algorithm structure, we cannot use C2H acceleration. We expect to further optimize the algorithm, so we did not use other methods to accelerate the algorithm. In the future we will improve the system functions in terms of the algorithm structure and acceleration.

Figure 15. Spinal Cord and Disk Location



Disk Detection

In the right image in Figure 15, the red points represent the located disks, which the system marks with yellow lines.

SOPC Concepts Used in the Design

Using SOPC concepts, we completed the design successfully. In the system design, SOPC concepts are embodied in the following aspects:

- *System reconfigurability*—As a soft-core processor, the Nios II processor can be clipped. Therefore, the system we design has huge potential for scalability. For example, due to time limitations, we used C programs (with limited speed) to implement some algorithms that are actually suitable for implementation in the FPGA hardware. But, we can upgrade the system later without changing the hardware platform. Additionally, because of resolution limitations, we can

compile different LCD controllers for use with other LCD displays. These types of changes are advantages of the system reconfigurability provided by SOPC-based designs.

- *Modular system design*—System design is a process of labor division and teamwork. A typical embedded processor platform is designed according to the specified processor, and software debugging can only be conducted upon completion of the hardware. But designing with the Nios II processor is different: as long as we use an FPGA the supports the Nios II processor, we can later debug the program on other FPGA-based platforms without any differences. This method allows the designer to conduct the hardware and software design synchronously. In actual designs, designers can initiate market expansion and product research and development simultaneously, shortening the product's time-to-market and bringing significant benefits.

- *Diversified implementation modes*—In a SOPC-based system design based, there are various implementation modes. For example, to accelerate an algorithm, you can use a custom instruction, custom peripheral, or C2H acceleration and compare them to find the best method.

Design Features

The Nios II-based MRI spinal image segmentation system features fast computation, small size, and simple operation. It can be integrated easily with the original MRI devices to form a new system, and facilitate consultation using the segmented images.

- With the introduction of Nios II-based µC/OS-II, this operating system has been widely applied in many fields worldwide, such as mobile phones, routers, hubs, aerocrafts, medical instruments, etc. µC/OS-II is suited to a small control system, and features high efficiency, small size, excellent real-time performance, good scalability, etc. The operating system has been integrated into the Nios II IDE, avoiding the need for additional migration. Using the system we found that the Nios II-based µC/OS-II and tasks are very stable. We completed all system software development using the Nios II-integrated µC/OS-II RTOS.

- With µC/GUI, the system gains good human-machine interaction. All system functions can be accessed using a mouse. This feature is very helpful for promoting the system.

- In the system design, a large quantity of MRI images are stored in a high-capacity SD card. Taking advantage of the Nios II processor, we easily added in the four-line SD card controller to improve its read speed. We also migrated the µC/FS file system for the SD card to facilitate the file access operations.

- The µC/OS-II-based network interface enables strong scalability. Additionally, network support allows image updating and supports telemedicine.

- The Nios II processor includes several CPUs, and users can create a perfect solution by using processor, peripherals, storage, and I/O interfaces according to the system demands. In this way, designers build a reasonable performance combination and save system development cost, enhancing cost competitiveness.

- The Nios II C2H Compiler can automatically convert a C language program that requires high performance into a hardware accelerator and integrate it into the FPGA-based Nios II subsystem, which improves system operation speeds.

Conclusion

We completed our MRI spinal segmentation system based on the Nios II processor design for this contest, and implemented the system functions as scheduled. However, we still need to improve some areas. During the contest, we learned many new things about the Nios II processor and it was the first time we tried to implement an algorithm in the Nios II processor for medical imaging. Our experience demonstrated the Nios II processor's strong processing capability and reliable operation. Additionally,

we became experienced with using μC/OS-II, μC/GUI, and μC/FS with the Nios II processor and solved many debugging problems with teamwork.

Nios II system performance can be adjusted according to the application requirements to satisfy specific user demands, which gives Nios II systems strong advantages over fixed processors. User logic function and custom instructions are highlights of the Nios II processor, and provide a variety of methods to implement systems. SOPC concepts bring creative design thought and enlighten students' creativity.

Thanks to Altera for providing us a good opportunity to combine theory with practice. The contest verified the feasibility of our algorithm in hardware and promoted our theoretical research.

Second Prize

Nios II Processor-Based Self-Adaptive QRS Detection System

Institution: **Indian Institute of Technology, Kharagpur**

Participants: **Sai Prashanth, Prashant Agrawal**

Instructor: **Professor Agit Pal**

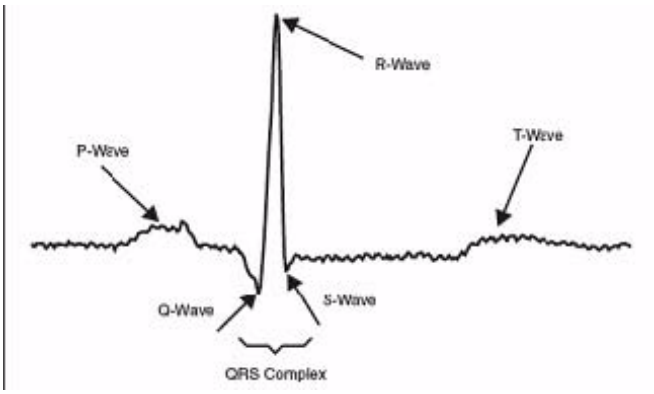
Design Introduction

For our project, we designed and implemented a cardiac arrhythmia electrocardiogram (ECG) monitoring system that can adaptively modify and change the components of its processing chain to carry out the best treatment on electrocardiogram signals. We investigated and provided a solution to a fundamental problem in the area of biomedical signal processing: accurate QRS complex detection for varying environmental and patient conditions. We implemented the ECG monitoring system in an Altera® Cyclone® II FPGA.

Background

The QRS complex is the most striking waveform within the ECG. Because it reflects the electrical activity within the heart during the ventricular contraction, the time of its occurrence and its shape provide a wealth of information about the current state of the heart. Due to its characteristic shape (see Figure 1), it is the basis for automated determination of the heart rate, an entry point for classification schemes of the cardiac cycle, and often used in ECG data compression algorithms. Therefore, QRS detection provides the fundamentals for almost all ECG analysis algorithms.

Figure 1. QRS Complex within the ECG signal

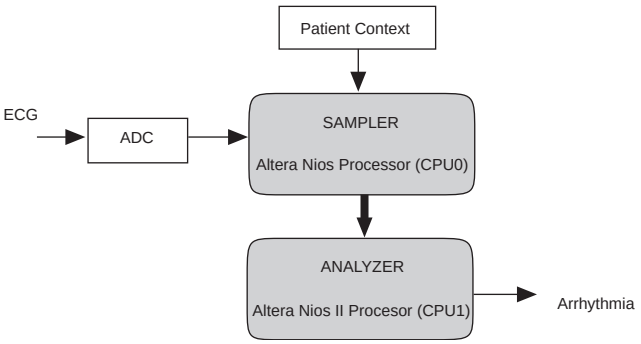


Software QRS detection has been a research topic for more than 30 years. However, experience gathered over the years shows that the proposed strategies for ECG analysis [2], and particularly QRS complex detection based on signal processing techniques, have reached an asymptotic detection performance. This situation exists is because most algorithms operate optimally in a given set of contexts (environmental and/or patient-related), and produce increasing error rates when the contexts are not matched. Therefore, choosing the QRS detection algorithm best suited to the current context is an essential step in the development of a real-time ECG analysis system. Our implementation adopts the real-time piloting system proposed by F. Portet and G. Carrault, in "Piloting Real-Time QRS Detection Algorithms in Variable Contexts" (see "References" on page 332), however, we adapted the system for optimal performance using the Altera Nios® II processor.

Project Outline

In medical monitoring, reducing false alarms and missed detections is crucial, and its importance cannot be overemphasized. Our novel adaptive, algorithm-bank-based solution reduces the number of errors by performing a periodic sampling of the input ECG signal and making a dynamic decision to find the most appropriate algorithm for QRS detection under the current context. Figure 2 shows the design overview of our ECG monitoring system. The gray area represents sub-units within the scope of this project, and is implemented using the Altera Cyclone II FPGA.

Figure 2. ECG Medical Monitoring System



Our ECG monitoring system has two distinct components: an analyzer that performs the actual QRS complex detection and medical diagnosis, and a sampler that performs acquisition, context analysis, and piloting of the analyzer. When a change occurs in the input data, the sampler should react to adapt itself to the new context. Otherwise, the analyzer transmits erroneous data and causes false alarms and low-quality diagnosis.

FPGA Design Significance

The trend in embedded system design is towards implementing the entire functional system on a single chip. The advent of high-density FPGAs with high-capacity RAM blocks and support for soft-core processors such as Altera's Nios II processor have enabled designers to implement a complete system on a chip. We use FPGAs, and in particular, the Altera Nios II soft-core processor to take advantage of the following benefits:

- Altera Cyclone II FPGA systems are portable, cost effective, and consume considerably less power compared to PCs. This fact is important when the application is designed for battery-operated devices. We can implement a complete system easily on a single chip because complex integrated circuits (ICs) with millions of gates are now available.
- SOPC Builder can trim months from a design cycle by simplifying and accelerating the design process. It integrates complex system components such as intellectual property (IP) blocks, memories, and interfaces to off-chip devices including application-specific standard products (ASSPs) and ASICs onto Altera high-density FPGAs.
- The Altera Nios II processor supports hardware/software co-design in which the time-critical blocks are written in HDL and implemented as hardware units, while the remaining application logic is written in C. The challenge is to find a good trade-off between the two. Both the processor and the custom hardware must be optimally designed such that neither is idle or under-utilized.
- FPGAs provide the best of both worlds: a microcontroller or RISC processor can efficiently perform control and decision making operations while the FPGA can perform digital signal processing (DSP) operations and other computationally intensive tasks.
- The Altera Nios II processor supports multi-core processing, which enables off loading and time-sharing critical mutually independent operations between two processor cores to offer real-time response in crucial situations. The synchronization between the processors is easily facilitated by the Avalon® bridge fabric.

Nios II-Based Design

We decided to use the Nios II processor after analyzing the various requirements for a real-time ECG medical monitoring system. A handheld, battery-operated medical monitoring system requires that the design be optimized for performance and energy efficiency. Altera offers easy customization of both these features. A basic system requires application programs, running on a customizable processor that can implement custom digital hardware for computationally intensive operations such as fast discrete cosine transform (DCT) functions, matrix inverse calculations, etc. Using a soft-core processor, we can implement and customize various interfaces, including serial and parallel. The Altera development board, user-friendly Quartus® II software, SOPC Builder, Nios II Integrated Development Environment (IDE), and associated documentation enable even a beginner to feel at ease with developing an SOPC design. We can perform hardware design and simulation using the Quartus II software and use SOPC Builder to create the system from readily available, easy-to-use components. With the Nios II IDE, we easily created application software for the Nios II processor with the intuitive click-to-run IDE interface. The development board's rich features and customization, SOPC Builder's built-in support for interfaces (such as serial, parallel, and USB), and the easy programming interface provided by the Nios II hardware application layer (HAL) make the Nios II processor and an FPGA the ideal platform for implementing our ECG medical monitoring system.

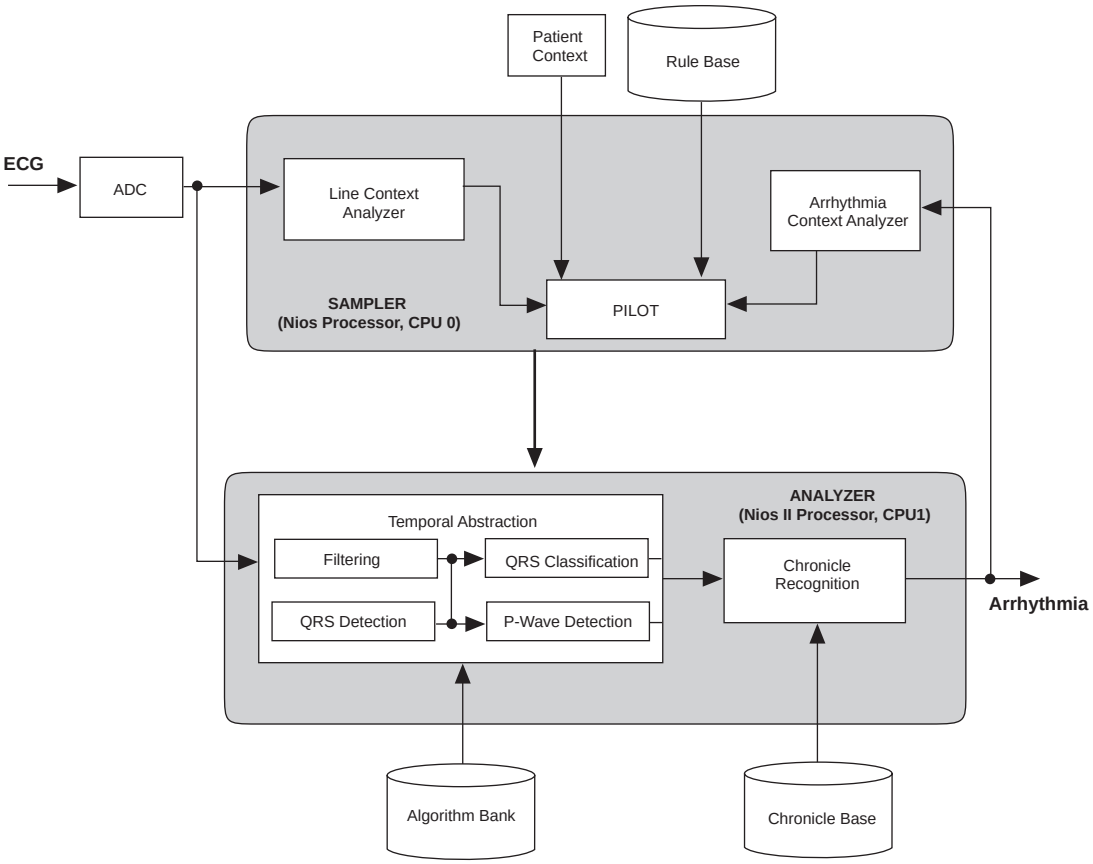
Application Scope and Target Users

Our design is customized for optimal real-time response, which is critical in a medical setting such as an electrocardiogram monitoring system. The design is implemented on a Cyclone II FPGA, and is very power efficient, which makes it suitable for handheld, battery-operated devices like the Holter ECG monitoring systems. It can also be used as a stand-alone clinical system for accurate patient heartbeat monitoring in hospitals and ambulances.

Functional Description

Our ECG monitoring system is devoted to cardiac arrhythmia recognition. Arrhythmia can be diagnosed from the morphology of the P and QRS waves and their temporal relationships. Our system computes a diagnosis from an abstracted representation. Figure 2 shows a high-level overview of the design. The analog electrocardiogram signals are first digitized using an external analog-to-digital converter (ADC) and are fed into to the system through the serial port. The system is composed of two on-line main modules: a sampler and an analyzer, each of which is implemented using a separate Nios II processor. Figure 3 shows a detailed overview of the module.

Figure 3. ECG Medical Monitoring System



At a high level, the sampler module continuously samples the input ECG signals to analyze the line context, and combines this information with the arrhythmia context of the higher-level patient context information. The sampler is governed by a pilot, which uses a set of statistically obtained piloting rules to determine the context. When a change of context is detected, it triggers the analyzer module to switch the algorithm used for the temporal abstraction, i.e., the QRS detection algorithm. The analyzer module consists of the signal processing algorithms that detect and classify the ECG events from the ECG signal. The chronic recognition module analyzes the vents flow and computes the diagnosis. The ECG monitoring system is piloted in three ways: the pilot activates and deactivates the temporal abstraction tasks, chooses and tunes the signal processing algorithms, and selects the level of detail that the arrhythmia recognition needs.

Arrhythmia Recognition Piloting

An arrhythmia can be diagnosed according to several ECG features. In our system, all features are constantly extracted and sent to the arrhythmia recognition, but in some contexts, a reduced number of features can be sufficient to recognize an arrhythmia. Thus, the arrhythmia recognition piloting involves choosing the chronicle abstraction level to recognize by selecting corresponding chronicle models, according to the current diagnosis hypotheses.

Temporal Abstraction Piloting

Temporal abstraction is composed of four linked tasks that extract four main features:

- Filtering separates the actual ECG signal from the noisy part of the signal
- QRS detection identifies QRS occurrence dates
- QRS classification labels QRS morphologies
- P wave detection identifies P wave occurrence dates

Depending on the context chosen by the arrhythmia recognition piloting system, a subset of the temporal abstraction piloting unit is activated. To be more efficient and to base the recognition on reliable information, the architecture enables the activation and deactivation of the temporal abstraction tasks according to the needs and to specific contexts.

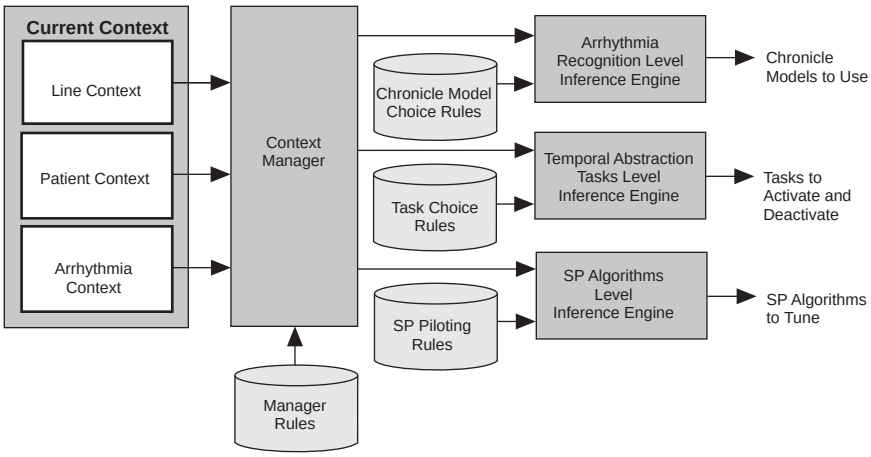
SP Algorithm Piloting

The temporal abstraction tasks are performed by shortest path (SP) algorithms. In our system, a unique SP algorithm is devoted to a particular task. However, related literature describes several possible algorithms, whose performance vary according to the context, to achieve these tasks. The preliminary study, described in [2], showed that the performance of various QRS detection algorithms change with the current context (line noise and QRS morphology). The new extended algorithm base contains several SP algorithms for each task. Therefore, the pilot must choose the algorithm best suited for the current context and then tune its parameters.

Pilot

Figure 4 shows the pilot architecture. It has three inference engines that deduce the actions to perform on the system for the three piloting levels and a context manager that deduces the information needed by the engines from the current context. The context manager instantiates and updates useful variables from the raw information transmitted by the context analyzers. Its knowledge is represented by expert rules stored as rules of thumb in the manager rule base. The system is piloted at three levels: the arrhythmia recognition level, the temporal abstraction tasks level, and the SP algorithms level. From the information transmitted by the context manager, the engines infer the actions to perform on the system. Their piloting rules are mainly defined by an expert and are grouped: chronicle model choice rules, task choice rules, and SP algorithm choice rules. The chronicle recognition adapts the abstraction level to the context. The temporal abstraction tasks are activated according to the needs and to technical constraints. The SP algorithm choice rules determine the algorithm best-suited to the task according to the temporal abstraction tasks and tune it.

Figure 4. Pilot Architecture



We used four real-time QRS detectors:

- *pan*—The Pan and Tompkins [3]
- *gritzali*—The Gritzali's detector [4]
- *df2*—The Okada's detector modified by [5]
- *af2*—A derivative QRS detector modified by [5]

We obtained the QRS detection piloting rules by performing a statistical analysis. We inferred the following rules:

IF <*L* and *bw* and *SNR* >= -5 dB>
THEN <*choose Gritzali's QRS detector*>
IF <(*L* or *F*) and *no noise*>
THEN <*choose Gritzali's QRS detector*>
IF <(*F* or *P*) and *bw* and *SNR* >= 0 dB>
THEN <*choose Gritzali's QRS detector*>
IF <*em* and ((*N* or *A* or *P* or *R*) and *SNR* = -15 dB)>
THEN <*choose df2 QRS detector*>
IF <*em* and (*SNR* = -5 dB and *P*)>
THEN <*choose df2 QRS detector*>
IF <*default*>
THEN <*choose PAN's QRS detector*>

The first rule means that if the line context has the value bw noise at -5 db and the arrhythmia context informs that it has mainly QRS of form L, then the Gritzali's detector is chosen.

Performance Parameters

For this system, accuracy and identification speed are the most important performance parameters; therefore, we focused on these areas. Using the Altera Quartus II design platform, we were able to speed up the design without lowering the design complexity. A single identification, including complex preprocessing, context checking, accelerated C-to-hardware acceleration (C2H) preprocessing, and hardware QRS complex detection, should be performed in real time to operate on the data streaming in. According to the MIT-BIH database (from the Harvard-MIT Division of Health Sciences and

Technology) benchmark experiment data, the threshold gives a very low 10.6percent error rate, which is considerably lower than the 14.3 percent error rate obtained when no sampler is used.

SOPC Builder allows the user to configure additional aspects of the microprocessor to improve computation speed, at the expense of using more system memory and logic elements (LEs). Specifically, the user can control the core type (Nios II/s, Nios II/f, etc.), pipelining, hardware multiply and divide, and cache allocation. Pipelining allows multiple instructions to be fed into each stage of the microprocessor execution cycle in parallel, enabling maximum execution performance of the navigation system software. Larger caches provide more memory data storage, which makes code execute faster. A large cache is particularly useful for the monitoring system software, which uses an incremental iterative process (i.e., values in a discrete wavelet transform (DWT) matrix are updated in a scanned incremental manner) to determine the DWT. However, larger caches also use more FPGA LEs and memory, and the designer can inadvertently create a system that does not fit into the target Cyclone device. Ultimately, we selected the cache size and pipelining based on trial and error, with the goal of maximizing the cache size while still fitting the design into the Cyclone FPGA.

Performance is assessed by the number of errors (Ne), which reflects both false alarms and missed beats. For each test, FN (the number of false negatives or missed QRSs) and FP (false positives or false alarms) are computed to obtain Ne = FP + FN. The error rate is Er = Ne / NQRS, where NQRS is the total number of actual QRSs. The study leads to 16,000 Ne values, and for this amount of data, we performed a principal components analysis (PCA) to analyze the detector results graphically. To test the piloting rules, five ECGs were generated from the MIT-BIH database. Each ECG lasted from 20 to 30 minutes for a total of about 2 hours. Three to four different contexts are introduced in each test ECG to assess the system performances in the specific contexts as well as around the context transitions. Parts of the original ECGs were corrupted with the three real clinical noise types defined previously (bw, ma, and em). In each context, the pilot chooses the best algorithm with the aid of the piloting rules. In this study, the algorithm thresholds are optimal in the sense that Ne is minimum. See Table 1.

Table 1. QRS Detection Results for Different Detectors and Pilot

ECG Score	Ne 1	Ne 2	Ne 3	Ne 4	Ne 5	Total	
						Ne	Er (%)
Pan	*20	*91	*240	*312	*367	1,030	14,3
Gritzali	20	*160	388	360	*295	1,223	17
df2	307	278	*174	*160	*302	1,221	17
Pilot	20	88	185	167	304	764	10,6

C2H Compiler

The Nios II C2H Compiler can automatically integrate high-performance C programs into the hardware accelerator, which is then integrated into the FPGA-based Nios II subsystem. The C2H Compiler supports standard ANSI C code, accelerates multiple application programs, and improves operational efficiency, including access to local and external memory and peripherals. We used SOPC Builder to generate a broadband Avalon interconnected architecture, which processes the external memory and peripherals, such as pointer dispersal and array access. The Nios II C2H Compiler accelerates implementation of memory interfaces, and generates hardware accelerator logic and the correct Avalon host and slave interface to match the memory delay. It shares the data computing and memory access functions with the Nios II processor, and lets the processor perform other tasks. Because the Avalon architecture does not limit the number of hosts and slaves in a system, the Nios II C2H Compiler can generate multiple hardware accelerators according to the target code's transfer requirements. The C2H Compiler helps embedded system developers improve design efficiency. In our system, the signal preprocessing function is implemented in software. Because we have high-speed identification requirements and the C software code takes a long time to perform the task, we optimized the ECG signal preprocessing module with the Nios II C2H Compiler to accelerate processing. We tested the implementation speed. With this optimization, the design uses extra logic resources: 65% instead of 20% without optimization.

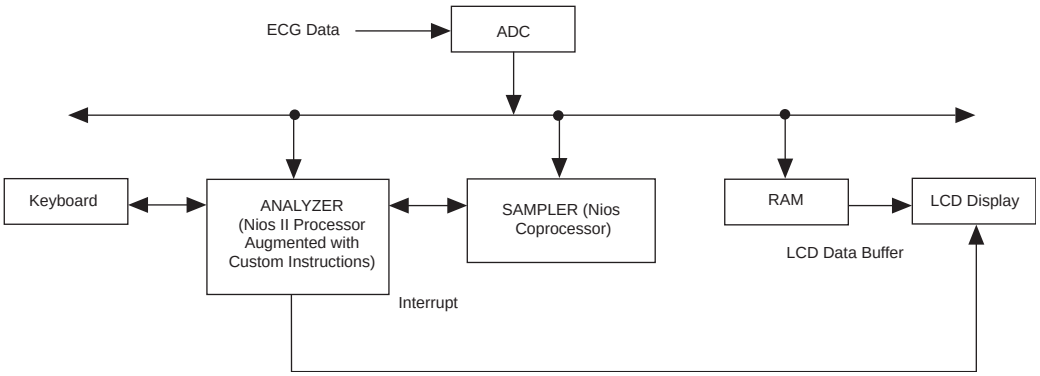
Nios II Processor

The Nios II processor's excellent performance facilitated our design. We chose the Nios II/f CPU because of our high-speed processing requirements. We combined the processor, peripherals, memory, and I/O interface with the Nios II processor and FPGA design. Because the Nios II processor is configurable, we could modify the system performance requirements at any time. Furthermore, we were able to improve the module performance with Nios II custom instructions.

Design Architecture

The Figure 5 shows an abstract hardware design block diagram of the ECG monitoring system.

Figure 5. ECG Hardware Block Diagram



The hardware system consists of two Nios II processors, which implement the sampler and analyzer modules, a SRAM, an external keyboard for user input, and an LCD display for graphically displaying the ECG signal. The input analog ECG signals are converted to digital format using an external ADC. The Avalon tristate bridge provides seamless communication between the various components of the system. Figure 6 shows the general software flow chart of the QRS detection algorithms. All QRS detection algorithms used in our context follow this methodology for identifying the QRS complex.

Figure 7 shows the user interaction state transition diagram and is important to understand the system operation. The user can interact with the ECG monitoring system via the keyboard and select the required operation mode. Some important operation modes are acquire signal from patient, analyze data, retrieve data, and transmit data.

Figure 6. Software QRS Detection Algorithm Flow Chart

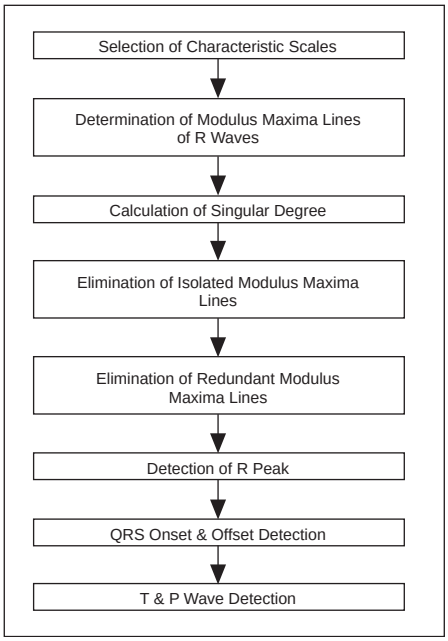
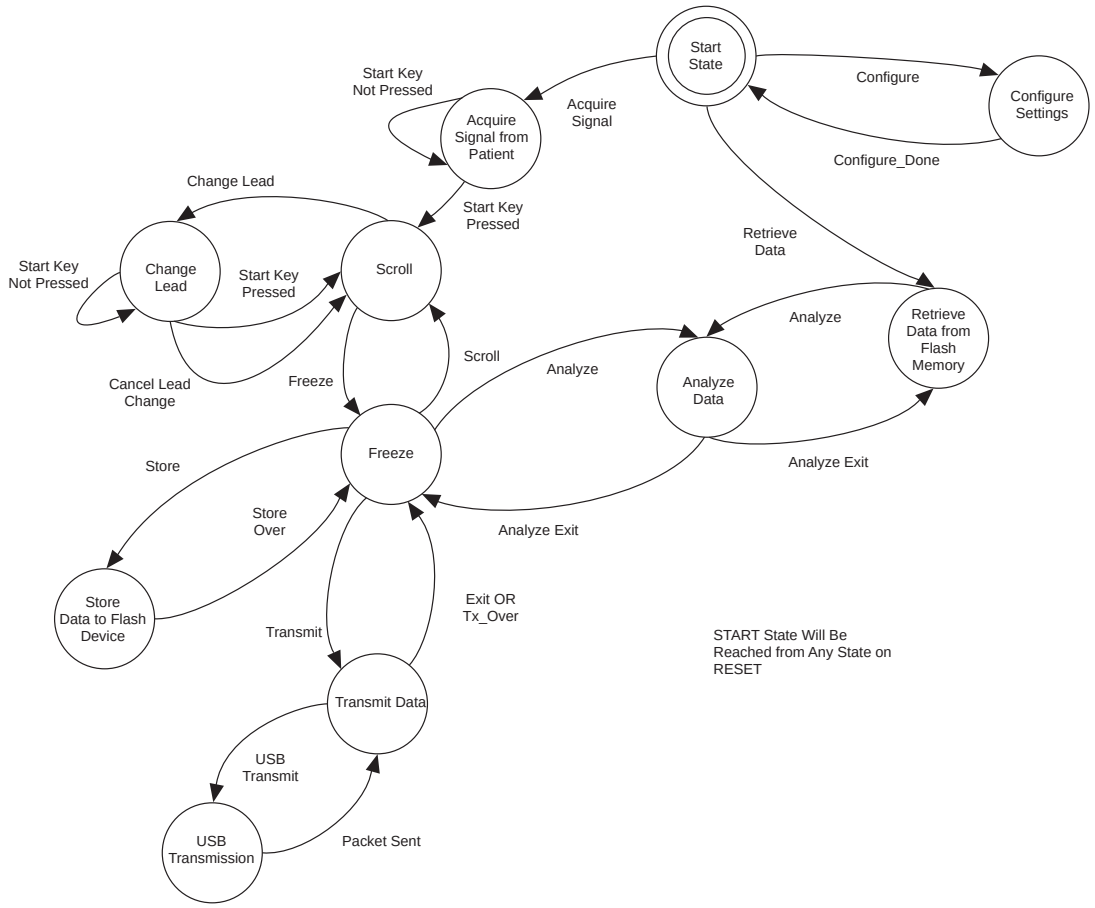


Figure 7. User Interaction State Transition Diagram



Design Description

The design's implementation steps are as follows:

1. Research and determine a set of complementary QRS detection algorithms that work effectively in a mutually exclusive set of contexts, and develop software algorithms to support them. Research Altera FPGAs using the Quartus II software, SOPC Builder, and Nios II IDE. Use the web editions of the software and documentation available from the Altera web site.

2. Create a Quartus II project, selecting appropriate Nios II dual-core processors in SOPC Builder, as shown in Figure 3 on page 322. Compile and debug the Quartus II design and review the compilation report. Test the project, including testing the processor response, error rate, and miniboard hardware (UART communications, etc.), with the simple hello world Nios II program.

3. Create an algorithm bank consisting of four different QRS complex detection algorithms implemented in the Nios II processor. Test the performance in the Cyclone device with the appropriate test input.

4. Update the SOPC Builder processor configuration to determine the fastest possible configuration that fits in the device's memory and available LEs. Optimize the processor until the desired performance requirement is met.

5. Create interrupt-based interfaces using the Nios II IDE to control the appropriate input and output to integrate the ECG medical monitoring system with other systems. Test these I/O interfaces.

6. Test the ECG medical monitoring system performance using standard available electrocardiogram signals from the MIT-BIH database and determine the error rates of the QRS complex detection.

Design Environment

We used the Altera Cyclone II development board for the initial code debugging and then used the development and education (DE2) development platform for final implementation. The DE2 board has a variety of integrated peripheral interfaces that were convenient to use in our design.

Software and Hardware Design

Figure 8 shows the SOPC Builder configuration.

Figure 8. SOPC Builder Configuration

The two important modules in the ECG monitoring system are the sampler module and the analyzer module. We implemented them using separate Nios II processors to facilitate real-time response to the streaming in electrocardiogram signal. The sampler module is implemented using **cpu0**, and it incorporates the line context analyzer and the pilot. The line context analyzer analyzes the quality of the incoming ECG signal and determines the decibel noise level. It also has an arrhythmia context analyzer, which contains information about the QRS morphologies that occurred in the past. Using this information, in addition to the high-level patient related context information, the pilot then uses statistically determined piloting rules to decide which context is most suitable for the analyzer to use. It makes a decision dynamically and interrupts the analyzer module to change its processing cycle.

The analyzer module consists of the temporal abstraction unit, which is composed of the signal processing algorithms and the chronicle recognition unit. Depending on the interrupt received from the sampler module, the analyzer uses the appropriate QRS detection algorithm for processing the ECG signal, as outlined in “Functional Description” on page 322. Upon appropriate processing, the morphologies are then passed to the chronicle recognition unit to determine any arrhythmia, which can then be subject to medical diagnosis. Figure 9 shows the block diagram for system implementation using the Quartus II software.

Applying SOPC Concepts

Altera introduced system-on-a-programmable-chip (SOPC) technology and its related development platform, the Quartus II software. SOPC is the FPGA version of system-on-chip (SOC). Compared to ASIC SOC, SOPC has many unique features. Our design uses SOPC concepts in the following ways:

- *Modular system design*—At the beginning of the system design, we partitioned the system into a line context analyzer (preprocessing) and processing. The system is divided and simplified, which makes it easier to implement. According to the module interfaces, we can accurately evaluate the design's application scope and future development at the initial design stage. We can perform market exploration and product research and development simultaneously in the practical product design, which shortens the time-to-market and accelerates enterprise development.

- *System integration*—An embedded system shows its features with its size, power consumption, and integrity. Except for the expanded 1-Mbyte SRAM front-end collection module, we could implement all system functions on the development board. It is very difficult to implement such a highly integrated design without lowering the design target or using a different FPGA.

- *Various modes*—We can diversify the implementation. For example, the front-end module uses an IP core, preprocessing is implemented with software, and key steps are fulfilled using the C2H Compiler to transfer operations to hardware. The trademark checking module uses hardware, software, or hardware/software with custom peripherals and instructions. Using SOPC concepts and excellent design tools enabled us to use these various modes.

- *Final system can be upgraded*—The design can be flexibly configured and updated during the design process.

Figure 9 shows the Quartus II system implementation block diagram.

Figure 9. Quartus II System Implementation Block Diagram

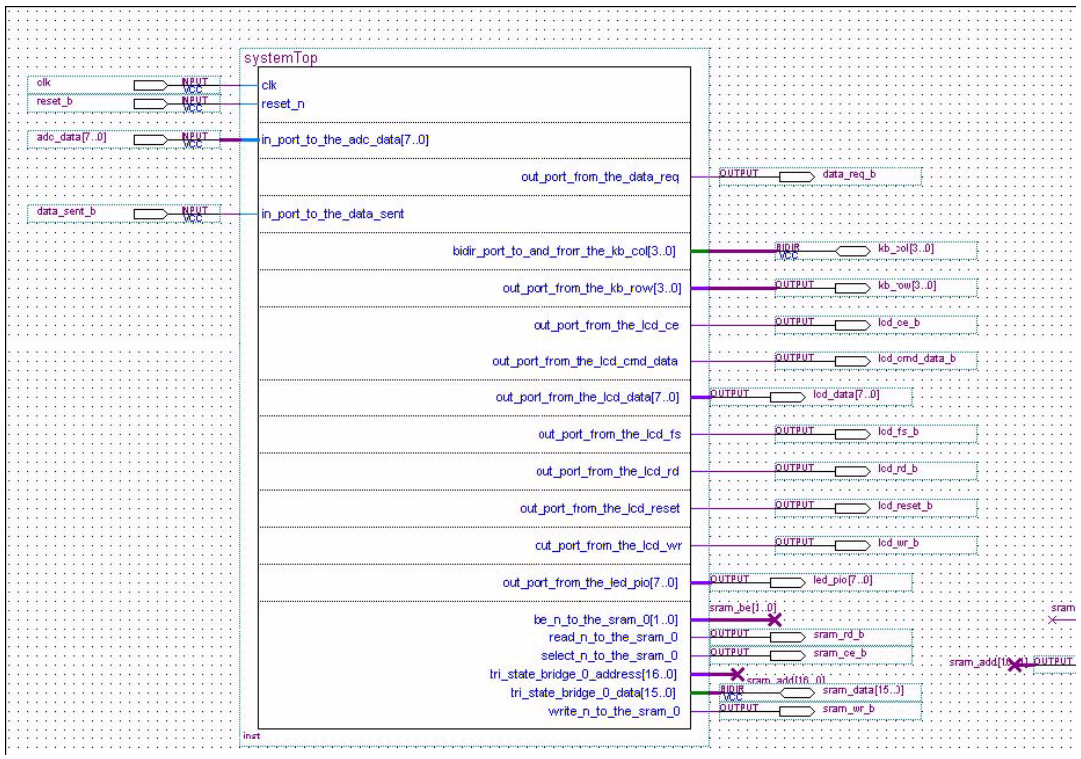


Table 2 shows the percentage of execution time spent in each of the sub-functions that constitute the QRS detection algorithm. The `dwt_ecg` function, which performs the DWT computation, consumes the most time. The computationally intensive portions of the QRS complex detection algorithm are implemented using custom instructions. The most time-consuming function computes the DWT, which

is implemented using custom instructions. Prior to this, loop unrolling was performed on the initial code to reduce the number of iterations required to perform a single DWT calculation.

Table 2. Execution Time for Various Functions

Function	% of Total Time
dwt_ecg	69.2
detect_mm_R	9.99
detect_r	0.13
detect_qrs	0.16
detect_t	13.7
detect_p	6.7

Design Features

Our ECG medical monitoring system has the following features:

- It is a standalone system for detecting QRS complexes in an electrocardiogram for further medical diagnosis without using a PC for recognition.
- Performs accurate QRS complex detection under varying conditions, where any single algorithm would fail to function effectively. These conditions include environmental disturbances (such as noise due to electrical interference, muscular activity, or loss of contact) and patient characteristics (i.e., varying heart beat classifications).
- Custom instructions are optimized for area and energy using Nios II architectural features, such as an extended custom instruction architecture (for resource sharing) and internal registers (to reduce memory access latency). These features reduce the device cost.
- Run-time electrocardiogram signal acquisition, processing, and morphology recognition makes the system suitable for practical use in Holter ECG systems, clinical use, etc. Implementation is optimized for minimal latency by exporting computationally intensive parts of QRS complex detection algorithms to custom instructions, and using the periodic sampling subunit as a co-processor. This technique enables the Nios II processor to exhibit real-time performance, which is critical in biomedical signal processing applications such as heart beat monitoring.
- The algorithm software is also optimized (using techniques such as loop unrolling) with the C2H Compiler.
- Overall energy-efficient system design enables use of the design in hand-held, battery-operated devices, such as Holter systems.
- SOPC design plays a central role in all design features, and enables easy optimization for minimal latency, area, and energy consumption. Several Nios II architectural and support features ease the process of system design and development.
- Displaying the ECG signal on a liquid crystal display (LCD) aids a specialist in deducing graphical conclusions from the morphologies.
- The design uses a variety of features and components available for Nios II-based development, such as PIO, UART, and RS- 232 communication. In the future, we would like to implement USB communication as well so that we can provide a standalone ECG monitoring system that can automatically log data in an auxiliary storage device for archiving.

- The system has low cost and high performance. The single chip ECG monitoring system is small, easy-to-carry, and cost effective, which satisfies the needs of most engineering technicians. Compared to the expensive medical monitoring systems currently on the market, this system provides excellent performance at a lower cost.

- The system is portable: the single FPGA and Nios II processor can implement ECG signal collection, analysis, storage, control, and transfer, which allows the ECG monitoring system to migrate from large a desktop to small handsets. Additionally, it provides portable terminals suitable for working outdoors.

Conclusion

The Altera Nios II design contest enabled us to develop a better understanding of the Nios II processor. Using it, we were able to design our system easily, including dual-core embedded processors, on-chip and off-chip memory, and high-speed I/O ports. Altera development tools let us develop our own multi-functional custom instructions quickly. Additionally, we could modify the CPU hardware at any time for multi-purpose development using SOPC Builder. We hope to use the Nios II IDE debug function in future to shorten the software development time significantly. Altera's ability to develop and update the Nios II processor and functions was extremely important. For example, using custom instructions we could accelerate the hardware computation speed, which improved our system's efficiency. We thank Altera for having the contest and acknowledge their support when we had design problems. On the whole, using SOPC concepts allowed us to create a more flexible, dynamically reconfigurable, and computationally intensive implementation.

References

[1] F. Portet and G. Carrault, "Piloting Real-Time QRS Detection Algorithms in Variable Contexts," *IFMBE Proceedings*, Volume 11, Prague/Czech Republic 2005.

[2] B.U. Kohler, C. Hennig and R. Orglmeister, "The Principles of Software QRS Detection," *Engineering in Medicine and Biology Magazine, IEEE*, Volume 21, Issue 1, pp 42-57, Jan./Feb. 2002.

[3] Pan, J. and Tompkins, W.J. "A real-time QRS detection algorithm"

[4] Gritzali, F. "Towards a generalized scheme for QRS detection in ECG waveforms"

[5] Friesen, G.M., Jannett, T.C., Jadallah, M.A., Yates, S.L., Quint, S.R., and Nagle, H.T. "A comparison of the noise sensitivity of nine QRS detection algorithms"

Second Prize

Portable Telemedicine Monitoring Equipment

Institution: **HuaQiao University**

Participants: **Huafeng Hong, Qianjiang, Yongjie Li**

Instructor: **Ling Chaodong**

Design Introduction

For our design, we wanted to provide a specialized in-home medical monitoring system. The following sections provide background information about health issues and telemedicine.

Background

Our project focuses on several issues, including:

- *Medical*—In medicine today, the focus has shifted from disease treatment to prevention and health care. People care more about their health, and while disease prevention and health care have become an indispensable part of their lives, daily care for current physical conditions can eliminate problems and pain that could result from untreated conditions.

- *Social*—With increasing attention on health and technological progress both home and abroad, home health care engineering (HHCE) is an emerging discipline. It advocates the concepts of medical treatment at home, self health care, and remote diagnosis, and combines technology with medical treatment. While addressing the trends of an aging society, soaring medical expenses, and increasing health demands in the 21st century, HHCE enables medical resource sharing and improves medical care in remote areas, making it well received by society.

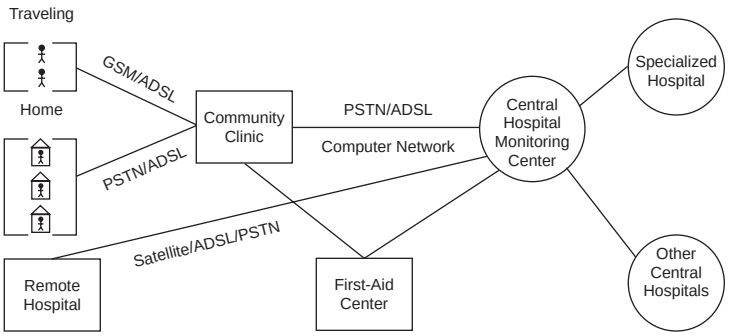
- *Technological*—Modern science and technology provides a technological basis for these designs. Embedded technology provides a leap forward and enables a diverse array of electronic products. Meanwhile, advancing network communications allows networked devices to share all kinds of

information easily. With improving manufacturing processes, chips are becoming more integrated and the resulting products are more portable and simplified.

Telemedicine

Telemedicine, which integrates network and medical technology, generally comprises remote diagnosis, expert consultation, information service, online checkups, remote communication, etc. Based on computers and network communication, it implements remote transfer, storage, query, comparison, display, and sharing of medical data, video, and audio information. See Figure 1.

Figure 1. Telemedicine Networking Structure



GSM = Global System for Mobile Communications
ADSL = Asymmetric digital subscriber line
PSTN = Public switched telephone network

Telemedicine has the following benefits:

- When used for home health care at the proper location, telemedicine can greatly reduce the time and cost of transporting patients.
- Medical centers, receiving photos, can perform management and home medical service assignments.
- Doctors can share medical records and diagnosis photos without geographical barriers, contributing to clinical research development.
- Medical staff in remote areas can receive better medical education.

Design Considerations

Our design accounts for the following considerations:

- The existing medical, social, and technological background shows that medical monitoring is moving towards personalized, portable, multi-functional systems. Systems and equipment are needed to meet this trend, which is the starting point of this design.
- Telemedicine will expand the network of existing HHCE equipment to every corner of the world. Benefits of telemedicine prove that medical equipment will play a greater role in a networked environment.
- The research in China is still in its infancy; for example, the remote network simply stores and transfers medical data in the database rather than truly combining the network with medical equipment. In other countries, although many funds have been invested in research, medical data

is still collected by expensive equipment, and data acquisition and network diagnosis are completed based on a PC and the Internet. Our design will provide a breakthrough in this respect.

- Many design solutions exist today. Systems based on embedded processors such as monolithic, digital signal processors (DSPs), ARM processors, and the Nios® II processor are good solutions. We decided to use the Nios II soft-core processor for the following reasons:

  - Altera's flexible, efficient system-on-a-programmable-chip (SOPC) solution integrates the Nios II processor, memory, I/O interface, and other functional modules onto a single FPGA to form a programmable system-on-chip. It boasts a flexible design, many available intellectual property (IP) cores, as well as clipping, expansion, and upgrading.

  - As an embedded soft-core processor, the Nios II processor features flexibility, high performance, low cost, and a long life cycle. Additionally, it comes with technical documentation and examples. Combined with an FPGA, you can develop anything that you can imagine, which is the key benefit of the Nios II processor and other soft-core CPUs. The Nios II processor supports µC/OS-II, µClinux, and many other real-time operating systems (RTOS), a light-weight TCP/IP (LwIP) stack, and zip file system, allowing users to add custom instructions and custom hardware accelerators, and to migrate customized peripherals and interface logic seamlessly. These features facilitate user designs while improving performance.

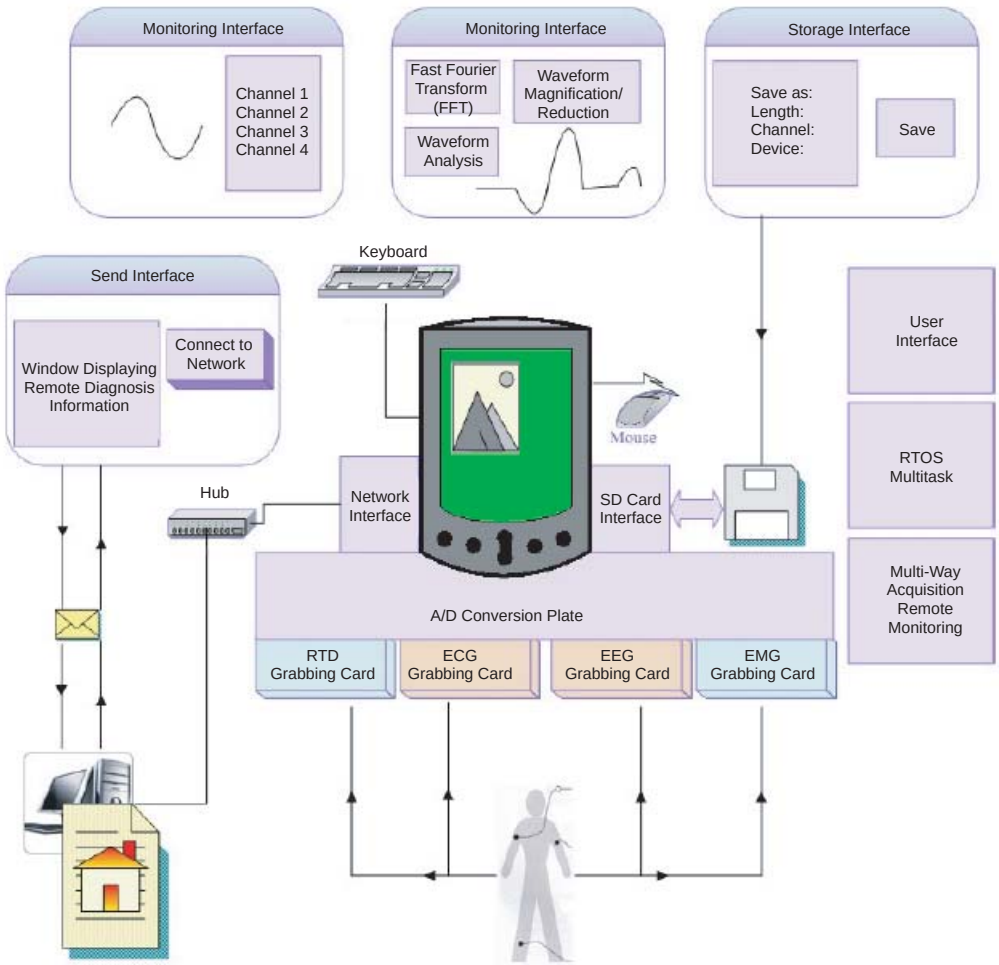
  - Altera is at the forefront of FPGA embedded system development. As soft-core embedded technology evolves, we will have a competitive edge in this field by mastering it early.

Based on these considerations, we decided to focus our design on user terminals that provide convenient, appropriate, operable, and Internet-enabled home telemedicine monitoring equipment for the aging population, young people, and children whose lives rely on technology (e.g., those suffering from accidents, disabilities, and congenital diseases), chronic disease patients, terminal cancer or AIDS patients, and special people (e.g., newborn babies or pregnant women).

Function Description

The design offers an effective, convenient medical monitoring solution for home, community, and home-care doctors. Designed mainly for user terminals, the monitoring equipment allows individuals to easily check and analyze their health conditions by themselves and obtain physical information (e.g., biomedical signals such as ECG, EEG, EMG, respiration, temperature, etc). The equipment displays these signals in graphics or waveforms so that individuals know intuitively whether their health indicators are normal. Additionally, the caretaker can make preliminary pathological diagnosis using the equipment's analysis function. The system stores the physical information for subsequent data analysis and processing. With the development of telemedicine, the system can connect patients to medical service (e.g., a hospital, private practitioner, or monitoring center) and deliver the physical information in real time to a remote database or doctor through the network. This feature helps manage medical information databases and provides remote monitoring and diagnosis, allowing individuals to enjoy timely and effective diagnosis without leaving home. See Figure 2.

Figure 2. Functional Diagram



The design in Figure 2 has the following functionality:

- *Multi-way acquisition of biomedical signals*—Simulated biomedical signals are collected modularly, e.g., a medical sensor and signal filtering/amplification modulation circuit and separate regulating cards collect different parameters. Because the physical signals frequency bands are below 2 K, we use an analog-to-digital (A/D) conversion chip with a 40 K sampling rate A/D conversion, and reserve a data port for card access.

- *Real-time display of physical information (graphics and data)*—The design uses serial input for data acquisition, saving I/O interface resources and eliminating the synchronization problems caused by parallel input. The collected data is transferred to SDRAM cache by direct memory access (DMA), reducing the CPU load. A display cache is created in SDRAM, while DMA technology transfers data to the liquid crystal display (LCD) for viewing. We designed the A/D acquisition and LCD interface control IP ourselves.

- *User-friendly operating interface and diversified processing and analysis functions*—We used a 320 x 240 thin-film transistor (TFT) LCD; migrating µC/GUI makes the interface more friendly and attractive. We provide four functional areas: monitoring, analysis, storage, and detection, and

multiple sub-functions simplify the operations. We compiled algorithms for detection, analysis, and processing to the signal characteristics, ensuring high accuracy.

- *Multiple functional interfaces (e.g. for network, compact flash (CF) or secure digital (SD) card) to facilitate data storage and transfer*—We used an SD card as the storage device and implemented an SD mode. We used the FAT16 file system for data access. A PS/2 interface enables interaction with the monitoring equipment.

- *Scalable interface and software upgrades*—For hardware, we provided a USB port, serial port, Integrated Development Environment (IDE) interface, and drivers for platform updates. The design adopts a RTOS to support application installation and upgrading.

- *Embedded web server allows the monitoring equipment to access and receive data via Ethernet*—We used the DM9000A network interface chip that has chip control IP that allows us to access the network easily. In the protocol layer, Altera provides a LwIP software component that comprises all protocols required by the network. We used a socket application programming interface (API) to write web server programs that made network communication easy. The design allocates an IP block for the monitoring equipment or uses DHCP. DHCP allows a remote PC to access the monitoring equipment through the Internet to obtain real-time data and parameters and send diagnosis information to local monitoring equipment through the web page's input area, implementing remote monitoring. Additionally, data can be saved to a remote database for management.

Performance Parameters

The following sections provide the design's resource usage and performance parameters.

Resource Usage

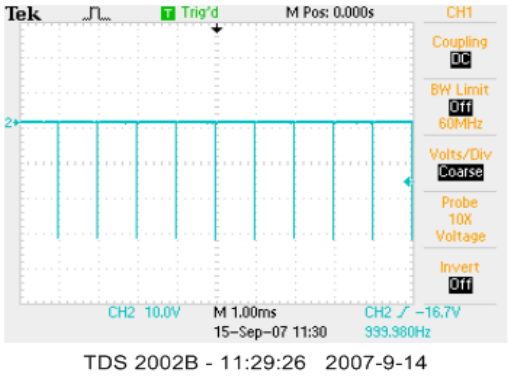
Figure 3 shows the design's system resource utilization given by the Quartus® II software. The system has 109.90-MHz f_{MAX} performance.

Figure 3. System Resource Utilization

Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	No
Total logic elements	8,481 / 33,216 (26 %)
Total registers	5569
Total pins	429 / 475 (90 %)
Total virtual pins	0
Total memory bits	166,784 / 483,840 (34 %)
Embedded Multiplier 9-bit elements	4 / 70 (6 %)
Total PLLs	2 / 4 (50 %)

The one-way A/D sampling controller uses 203 logic elements (LEs) and 8,192 memory bits (corresponding to a 512 bytes x 2 cache). It has a sampling frequency in the range of 0 to 1.667 MHz because the TLC549 sampling controller's maximum frequency is 40 kHz and the system's sampling frequency must be 0 to 40 kHz. If the sampling frequency set in software is 1,000 Hz, the output sampling frequency is 999.98 Hz. Figure 4 shows the A/D chip sampling rate.

Figure 4. A/D Chip Sampling Rate



The system software uses 3,046 Kbytes for storage.

Design Performance Parameters

The following sections describe the performance parameters of the design.

Front-End Acquisition Board

Table 1 shows the pre-amplifier channel bandwidth test. The pre-amplifier gain is stable between frequency bands of 1 to 1 kHz, i.e., the channel bandwidth is ≥ 1 kHz.

Table 1. Pre-Amplifier Gain Changes with Frequency

f (Hz)	1	5	10	20	50	100	200	500	1,000
G (Vpp = 10 mV)	12.1	12.4	12.3	12.4	12.0	12.1	12.1	12.1	12.1

Table 2 shows the amplifying power of the amplifier for different signals. At 20- and 50-Hz frequencies, the amplifier gains are stable when the input signal amplitude changes.

Table 2. Pre-Amplifier Gain Changes with Input Signal

Vpp (mV)		40	60	80	100	120	150	300	400	800
G	f = 20 Hz	11.9	12.3	12.4	12.6	12.5	12.7	12.5	13.0	13.3
	f = 50 Hz	11.9	12.3	12.4	12.9	12.7	12.9	12.4	13.1	13.3

Table 3 shows the trapper's trap feature test. The attenuation degree is increased by compromising trap impedance, i.e., a proper point is adjusted to obtain the attenuation and trap impedance.

Table 3. Change of 50-Hz Trapper with Frequency

f (Hz)	1	10	20	40	45	47	48	49	50	51	52	53	55	60	80
G (Vpp = 50 mV)	6.4	6.4	6.8	4.5	2.8	2.0	1.5	1.0	0.6	0.9	1.3	1.6	2.4	4.3	6.1

SD Card Parameter Test

The test uses 100 16-bit data for reading/writing text files, which takes 40 ms. Assuming that the front-end data sampling is 2 kHz and 2,000 data points are collected every second, the storage time would be (2,000/100) x 40 ms = 800 ms. The test result shows that the collected data is not lost. See Table 4.

Table 4. SD Card File Operation Functions

Operation	Function
File system start/exit	FS_EXIT(), FS_INIT
Open/close file	FS_FCLOSE(), FS_FOPEN()
Read/write file data	FS_FREAD(), FS_FWRITE()
Locate file	FS_FSEEK(), FS_FTELL()
Remove file/file directory	FS_REMOVE()
Create/close file directory	FS_MKDIR(), FS-_CLOSEDIR()
Open/read/locate directory	FS_OPENDIR(), FS_READDIR(), FS_REWINDDIR()

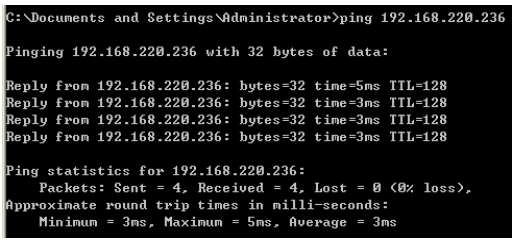
Network Speed Test

We set the monitoring equipment's IP address to 192.168.220.236 in the test using the Linkwan.com web site test tool. Figure 5 shows the test result.

Figure 5. IP Address Test

We tested the operation of the built-in ping command as shown in Figure 6.

Figure 6. Ping Command Test



ECG Signal Detection Result

Table 5 shows the R-wave detection results and Table 6 shows the data compression result. The data source is the MIT/BIT ECG database, with 250-Hz sampling, 8-bit quantification, and four signal groups collected for detection. The ECG detection algorithm's average R-wave false detection rate is

0.58%, the data compression ratio is as high as 13.75 times, and the correlation coefficient (CC) reaches 98.9%. The indicators are generally at a high level.

Table 5. R-wave Detection Result

Signal	Total Heart Rate	False Accept Rate	Undetected	False Detected Heart Rate	False Detection Rate (%)
Sddb-30	1,545	4	1	5	0.32
Sddb-32	2,013	6	7	13	0.66
Sddb-35	3,326	15	14	29	0.87
Sddb-37	2,111	4	6	10	0.47

Table 6. Data Compression Result

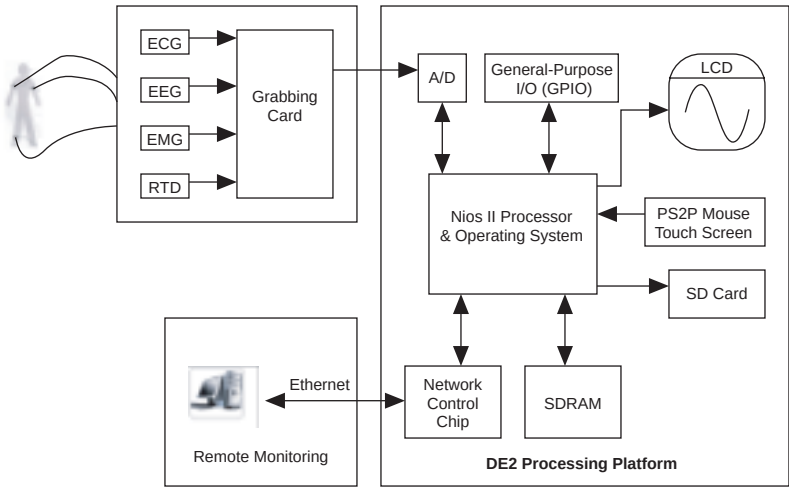
Signal	CR Compression Ratio (%)	PRD (%)	Correlation Coefficient (%)
Sddb-30	19	7.3	99.7
Sddb-32	11	16	98.8
Sddb-35	12	23	97.9
Sddb-37	13	13	99.2

Design Architecture

The monitoring equipment mainly consists of three modules (see Figure 7):

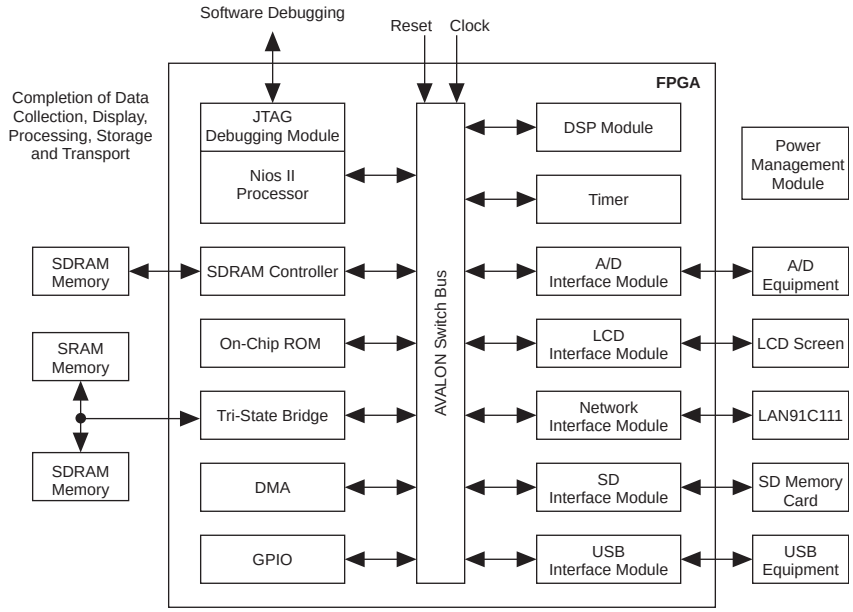
- Front-end collection and modulation
- Signal processing, storage, and transfer platform
- Remote monitoring

Figure 7. System Structure



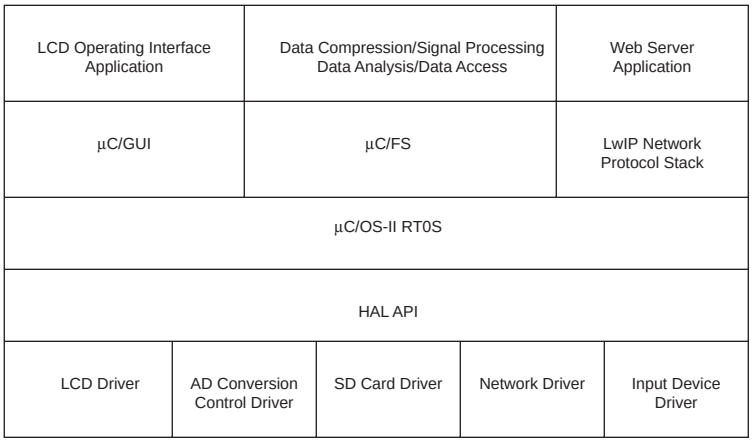
The hardware platform is the Development and Education (DE2) board, which contains the Altera® Cyclone® II EP2C35F672C6 FPGA. The hardware integrates the Nios II soft-core processor, memory, IP functions, and I/O ports on a single FPGA via SOPC technology. Peripheral hardware, including the data acquisition module, network, LCD screen, touch screen/keyboard, USB/SD memory, etc., are extended with a scalable I/O interface to facilitate system upgrades. Figure 8 shows the hardware platform.

Figure 8. SOPC Hardware Platform



Because our hardware platform is designed using SOPC concepts, the Nios II system is written using a hardware abstraction layer (HAL) driver, including the A/D conversion control IP core, LCD control core, network control IP, SD card control core, PS2 IP, etc. For multi-tasking, we used µC/OS-II for system dispatching. Additionally, we migrated µC/GUI and µC/FS as our graphical interface and file system, respectively. The LwIP protocol stack is a part of the software layer, and performs TCP/IP network communication. Other applications include a web server, operating interface, signal processing, fast Fourier transform (FFT) algorithm, and data compression. Figure 9 shows the software layer structure.

Figure 9. Software Layer Structure



Design Methodology

This section describes our design methodology.

Biomedical Signal Regulation Card Design

The biomedical signals are collected modularly, including the medical sensor, signal filtering amplifying modulation circuit, and A/D sampling circuit. The modulation circuit selects different filters and the amplified circuit according to the spectrum and scope of different biomedical signals. Using ECG as an example, the signal is amplified via the pre-amplification block, including a right leg driver to suppress common mode interference, a shield wire driver to eliminate lead wire interference, and the tenfold set gains.

We designed the pre-amplification block using the Analog Devices AD620 medical amplifier. The AD620 device is based on a modification of the classic three operational amplifier approach and is integrated using a co-phase differential amplifier in parallel. The AD620 device has a wide power supply range (± 2.3 V to ± 18 V), small size, and low power (it uses only a 1.3 mA maximum supply current), making it a good fit for low-voltage, low-power applications. Other advantages include a high common-mode rejection ratio, sound temperature stability, amplified bandwidth, and low noise. The amplified signal is further magnified using filtering and a 50-Hz trap filter. The post gain is set as 1 to 100. Because the maximum ECG signal is several mV and the A/D conversion input signal is over 1 V, the total gain is set as 1 to 1,000. Filtering uses a voltage-controlled voltage source second-order high (low) pass filter to eliminate signals interference such as myoelectricity beyond 0.05 to 100 Hz, as well as other high-order industrial frequency harmonics. Additionally, we used an active twin-T band-stop filtering circuit to curb the 50-Hz industrial frequency interference.

The A/D sampling chip is the Texas Instruments (TI) 8-bit serial TLC549 device. It uses a serial peripheral interface (SPI) to provide collection control and data transmission using three wires. It provides an on-chip system clock that typically operates at 4 MHz, as well as a software/hardware controlled circuit with a conversion time of less than 17 µs and a sampling rate of 40 kilosamples per second (KSPS). With a differential voltage reference, the TLC549 device can measure a minimum value of 1,000 mv/256, i.e., 8-bit resolution can be obtained without amplifying the 0- to 1-V signal. Figure 10 shows the ECG signal regulating card structure and Figure 11 shows the circuit diagram.

Figure 10. ECG Signal Regulating Card Structure

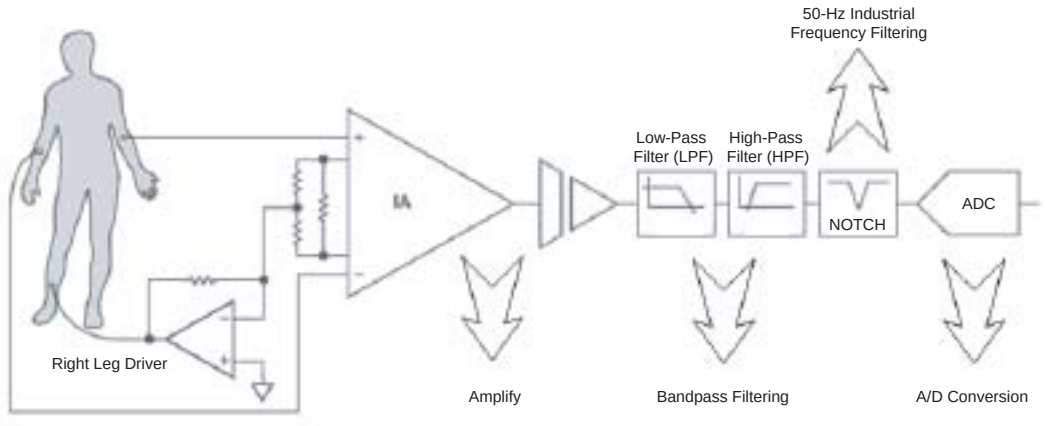
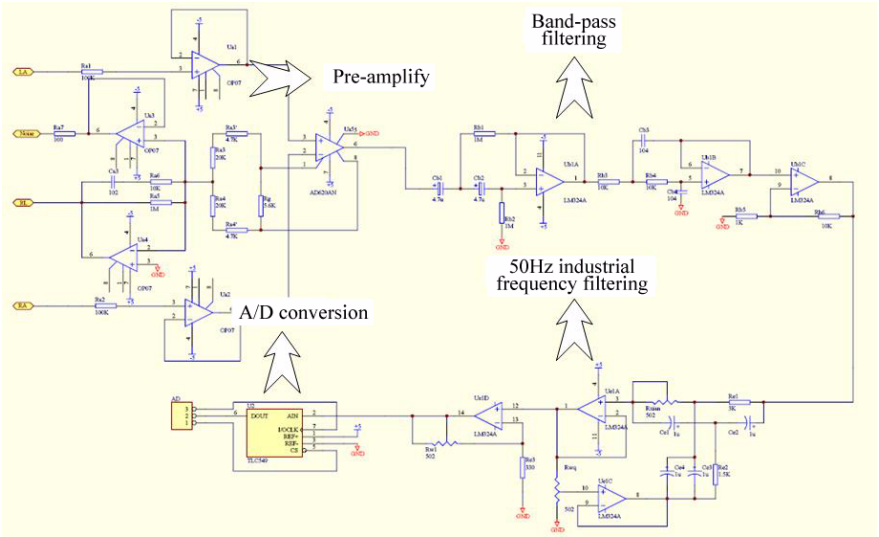


Figure 11. ECG Collection Circuit Diagram



SOPC Hardware Platform Design

We designed the hardware platform based on the Nios II processor. For our work, we first implemented the IP design, such as the A/D conversion control, LCD control, and data storage/transfer using a custom peripheral.

A/D Conversion Control IP Design

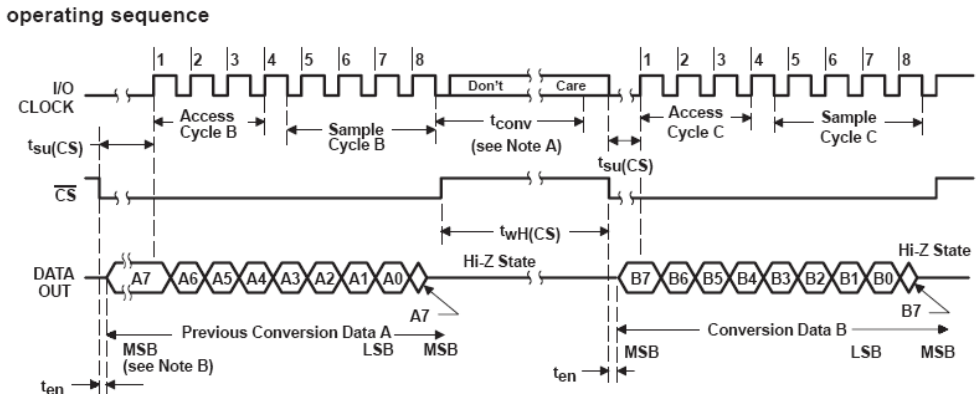
The A/D module design prevents the front-end signals from distortion and loss and deal synchronizes the data because of the multi-channel acquisition.

A/D Sequence Control Module

The system's A/D conversion chip is the TI TLC549 (TLC548) device, which is a low-cost, high-performance, 8-bit A/D converter. It implements A/D conversion using an 8-bit switched-capacitor successive-approximation approach. With a conversion speed of less than 17 μs, the TLC549 device can easily connect to various microprocessors using a three-wire serial interface to form various low-cost

test and control application systems. With a differential voltage reference, the TLC549 device can measure the minimum value of 1,000 mv/256, i.e., we can obtain 8-bit resolution without amplifying the 0- to 1-V signal. Sequence control is generated according to the sequence diagram shown in Figure 12.

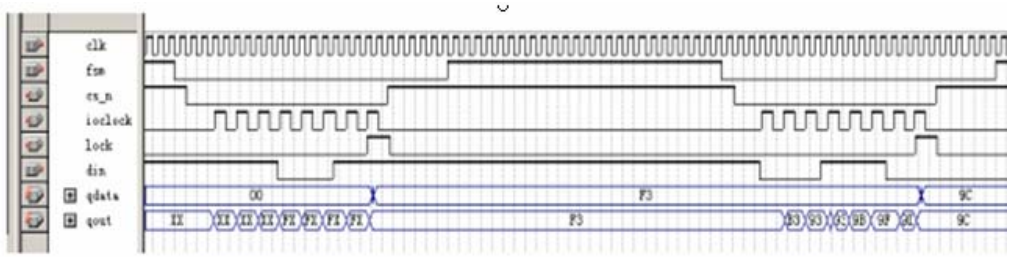
Figure 12. TLC549 Sequence Diagram



According to the sequence, the following tasks are performed when eight external clock signals are input at the TLC549 device's I/O clock: read the previous A/D conversion result, sample and reserve the input analog signal converted currently, and initiate A/D conversion.

To implement the TLC549 analog controller in the FPGA, we designed the simulation with a Verilog HDL control state machine as shown in Figure 13:

Figure 13. State Machine Sequence

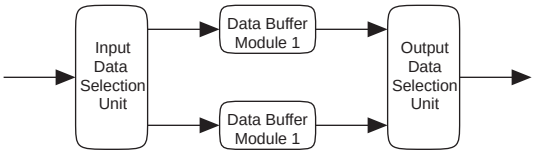


`din` is the serial input of data collected, and the clock is obtained through frequency division coefficient. `fsm` is the sampling control clock to adjust sampling speed as required.

Double-Buffered Operating Technology

Because A/D sampling is short, it is impractical to query or read data with interrupts. Therefore, the buffer design must reduce the interruption time by temporarily storing the converted data for N times in the buffer memory. To collect data continuously and correctly implement a seamless buffer, we use a ping-pong operation structure with double-buffer storage that takes advantage of the FPGA's design flexibility. The ping-pong operation is a handling technique for the data stream (see Figure 14). The data buffer module can be any storage unit and in this design we use dual-port RAM (DPRAM).

Figure 14. Ping-Pong Operation

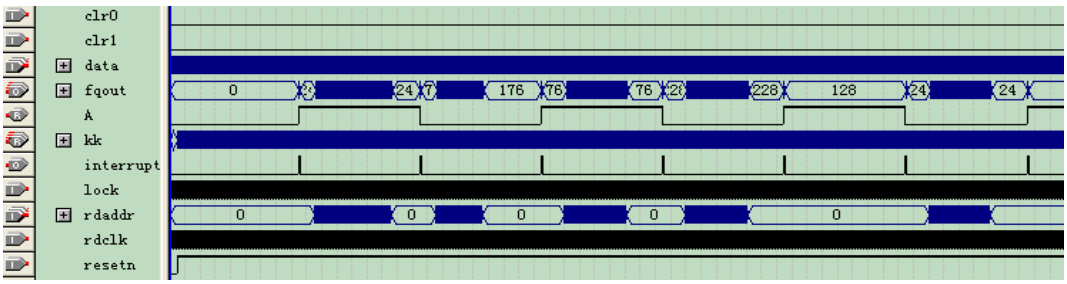


During ping-pong operation, the input data streams are distributed to two data buffer areas through the input data selection unit. The data buffer module can be any storage module, and the common storage units include DPRAM, single-port RAM (SPRAM), FIFO, etc. In the first buffer period, the input data stream is cached into data buffer module 1. In the second buffer period, the input data stream is cached into data buffer module 2 using the input data selection unit switch while the first period data in data buffer module 1 is output through the output port and the output data selection unit. In the third buffer period, the input data stream is cached into data buffer module 1 with the input data selection unit while the second period data in data buffer module 2 is switched by the output data selection unit and output via the output port for operation. The process repeats as required.

The ping-pong operation's unique feature is the collaborative switching of the input and output data selection units according to a meter, which sends the buffered data streams to the data stream processing unit for operation without pausing. The ping-pong operation module is an independent function and the input and output data streams are continuous at the sides of the module; therefore, the design can process data streams in the form of a pipeline. Ping-pong operation is usually applied in a pipeline algorithm for seamless data buffering and processing.

This design implements a data cache by alternatively storing an A/D sampling sequence controller into two 512-byte DPRAM blocks. When DPRAM1 is full, the data is stored in DPRAM2 with one interrupt so that the system has enough time to move the data out of DPRAM1 when the controller writes data into DPRAM2. Figure 15 shows the DPRAM buffer system timing diagram.

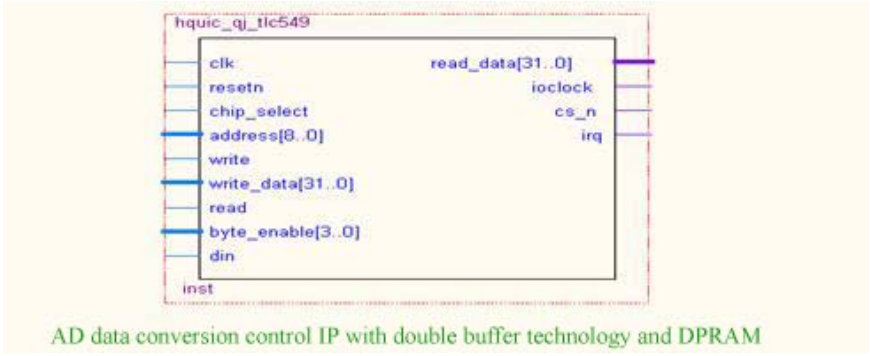
Figure 15. DPRAM Buffer System Timing Diagram



IP Design

The final IP core in the A/D sampling module includes an A/D conversion sequence controller, a double-buffer ping-pong operation module, a control register such as a sampling clock frequency division controller, and a bus control signal. Figure 16 shows the A/D conversion control IP core structure, which can connect directly to the Avalon® bus and can be added to the system if necessary. Four cores are added to the system.

Figure 16. AD Conversion Control IP



Each IP core has an independent double buffer and control register, which work in parallel. Control registers include a sampling clock controller (`FSAMPLE`), A/D-enabled control (`EN_AD`), etc. The sampling speed is controlled by setting the value of the sampling clock controller. For example, if we want to perform 10-kHz sampling for the analog signal with a 5-MHz control clock, the core just writes 10,000 in `FSAMPLE` and then a 1 in `EN_AD` to initiate the A/D conversion.

DMA Transmission

Our design uses the DAM core to move data blocks from DPRAM to SDRAM in the A/D conversion IP core. This process needs to write control instructions into the DMA control register to initiate the data transmission process. The status, read address, write address, length, and control registers require initialization. The DAM operation is initiated using the system's DAM subprogram. In the HAL API, the party with an auto-incremental address opens a receiving or sending channel and configuration address; the fixed-address party sets the `alt_dma_rxchan_ioctl()` parameters (using `ALT_DMA_RX_ONLY_ON` or `ALT_DMA_TX_ONLY_ON`) and the configuration address. For data transmission from DPRAM to the SDRAM, that is, when both source and destination are in auto-incremental address mode, we use the following DMA data transmission code:

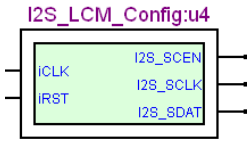
```
tx = alt_dma_txchan_open("/dev/dma_0");
   dma_res = alt_dma_txchan_send(tx, ad_buf, 32, NULL, NULL);
   // ad_buf is the source address

rx = alt_dma_rxchan_open("/dev/dma_0");
   dma_res = alt_dma_rxchan_prepare(rx, ad_buf, 32, dma_done, NULL);
   // ad_buf is the destination address, dma_done() is the call back function
   // employed upon the completion of DMA.
```

LCD IP Design

Our design requires a display device to show the collected signals and data waveforms in a format that is easy to understand. We used an LCD screen with the Terasic TRDB_LCM expansion board. The board has a Toppoly TD036THEA1 compact LCD module, can process an 8-bit (RGB or YUV) digital signal, and supports TSC and PAL sequences. It has a three-wire register control for display and function selection as well as embedded contrast, brightness, and rectification modules. It supports band color filtering 960 x 240 (TH mode, three primaries (red, green, and blue) virtualization, and YUV input). The expansion board is connected to the DE2 board's `GPIO_0` expansion port.

We use progressive scanning, and the LCD clock is 25.175 MHz. The design uses a three-wire LCM to configure the IP. The IP core's main function is to compile the state machine according to the control sequence in the data sheet and deliver configuration data. Figure 17 shows the TRDB_LCM block.

Figure 17. I2S_LCM Block



The LCD module does not have a display controller, so we designed it independently with Verilog HDL. The controller supports multiple color modes, including 18, 16, and 8 bpp, and self-defined mode. The image memory uses an on-chip FIFO buffer, which is adjustable according to design needs. A 256-color look-up table also adopts on-chip RAM. The image information can be read automatically from memory with DAM using the Avalon bus's main module transmission port. Figures 18 and 19 show the system.

Figure 18. LCD IP Core Structure

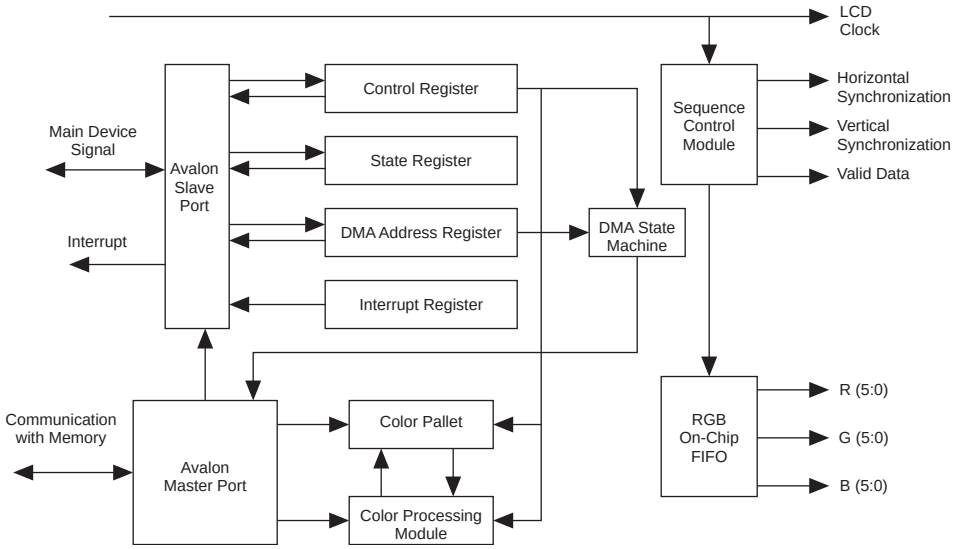
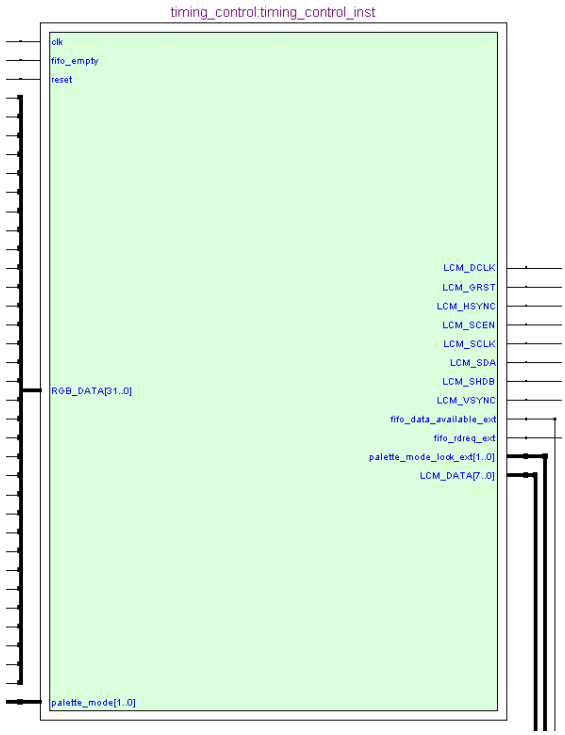


Figure 19. LCD IP Core RTL



The LCD core has four modules: the interface module, memory module, color conversion module, and sequence module.

The interface module operates the controller and reads the state. It contains a control register, state register, DMA address register, and interrupt register. See Table 7.

Table 7. LCD Controller Registers

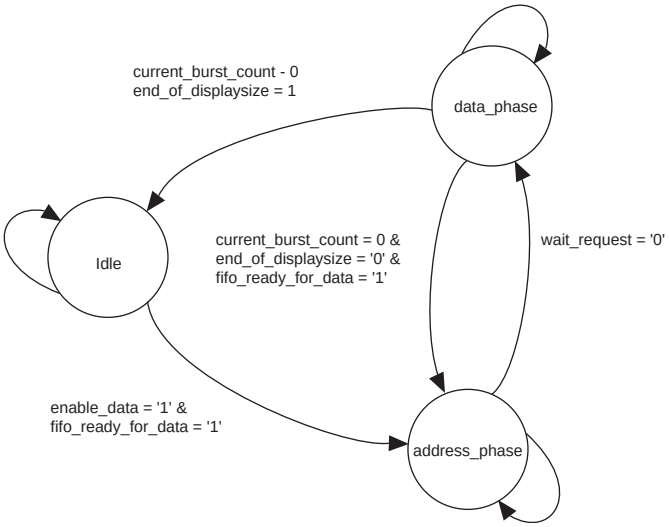
A1-A0	Register	Read/write	Description/Register Bit					
			31…5	4	3	2	1	0
0	Control register	Read & write		EIRQ		PMODE		EDMA
1	State register	Read & write	Status Inquiry					
2	DMA address register	Read & write	Write the start address of DMA transmission					
3	Interrupt register	Read	Clear interrupt					

The control register's EDMA initiates the DMA, PMODE selects the color pallet mode (18, 16, and 8 bpp, and self-defined), and EIRQ enables interrupts. The state register queries the interrupt state. The DMA address register sets and queries the DMA start address. The interrupt register clears interrupts.

The memory module reads the SDRAM's `FRAMEBUFFER` data independently into an on-chip FIFO in DMA mode using a state machine that reads/writes the Avalon master port. Based on the sequence, the state machine has three states: idle, address, and data. In idle state, it waits for the DMA start-up signal and initializes the module transmission and DMA start address number. When the DMA enable signal is initiated and the on-chip FIFO buffer is idle, it jumps to the address state. In address state, the wait signal on the wait bus is cancelled to enter a data read state. In data read state, the read signal begins to

read the SDRAM address data, and the module counter reduces by 1 for each consecutive data block. It returns to address state after reading a data block and adds the address automatically. It returns to idle state after reading a frame of data and waits for the transmission of the next frame. See Figure 20.

Figure 20. DMA State Machine

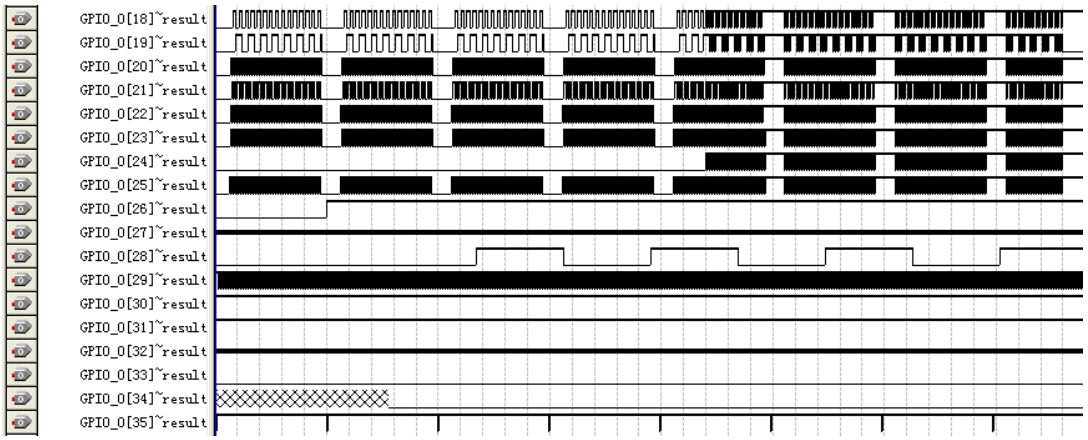


The color conversion module converts the read data according to four color modes. The 8 bpp and self-defined modes require a color look-up table because they have insufficient colors. The self-defined mode can preset the color pallet's address manually to define the color output.

The sequence module is compiled strictly according to the sequence of the LCD. The LCD clock is 25 MHz. The FIFO data output is initiated by controlling the data enable signal, and is displayed with a progressive scan. Meanwhile, the design must check whether there is data in the FIFO buffer before the data effective signal arrives to decide whether to read and transmit data. The color pallet mode is set and locked during frame transmission to avoid errors. Different read time periods are determined according to the bpp mode: BPP_18 must be read every time, BPP_16 is read every two times, and BPP_8 is read every five times.

We verified that the core outputs data and synchronous signals are stable, and sets display mode and RGB data bits through the register. Figure 21 shows the LCM analog sequence signal output to the DE2 board's `GPIO_0` expansion port.

Figure 21. Analog Sequence of LCM Signal Output to GPIO_0 of DE2 Expansion Port



SD Card Interface Design

One of the design's functions is to store the monitoring data. We use the DE2 board's SD card interface for large-volume data storage. The highly integrated SD card flash memory has serial and random access capabilities. It allows access through the specified serial interface with optimized speed and reliable data transmission, and we can stack several externally connected cards together. The interface completely complies with the SD card system standard defined by the SD card system specification, which is the latest consumer standard. The SD card clock is generated by the internal clock generator, and the interface driver unit synchronizes the external clock's `DAT` and `CMD` signals with the internal clock. The SD card has two communication protocols: SD and SPI. Comparing the two protocols, the biggest advantage of SD is that it has fast reads/writes, up to a theoretical 25 Mbytes/second for a single data line. The SD card interface in our system uses a single data line, `DATA0`. Three parallel I/O (PIO) IP blocks are used in the SOPC design as `SDATA`, `SCLK`, and `SCMD` SD card single data lines.

We compiled the SD card protocol in software and migrated the file system to save FPGA resources without any impact on the read/write speed. See "Software Platform Design" on page 356 for details.

HAL Network Driver-Based Design

The Nios II system takes HAL as a BSP to provide a unified peripheral interface in the embedded system. The HAL device driver abstraction, the main service provided by the HAL system library, is highly integrated into the SOPC design, allowing later software development to facilitate development and updates without hardware impact. The DM9000A-based HAL device driver development has two steps: designing the DM9000A read/write driver and migrating the DM9000A driver in HAL-based driver mode.

DM9000A Read/Write Driver

The DM9000A device is an integrated 10/100 Mbyte adaptive Ethernet control chip on the DE2 board. It has low cost and fast speeds, and features a common processor interface, 10/100 Mbyte adaptive, and 16 Kbit static access memory. Its simple design allows easy development of software drivers for different systems.

The DM9000A device cannot directly access the in-chip registers, but it can read/write using the data and index ports, which are controlled by the `CMD` pin. When `CMD` is high, it is a data port, and when `CMD` is low, it is a control port. The process to read/write any register is as follows:

1. Enable the DM9000A device by setting `AEN` and `SA7` low, and `SA8` and `SA9` high (this step is generally done in hardware without setting it in software).

2. Set the `CMD` pin low using software.

3. Input the register location to be read/written on the index port.

4. Set CMD pin high.

5. Input/output the register data to be read/written at the material port. See Figures 22 and 23.

Figure 22. DM9000 Read Process

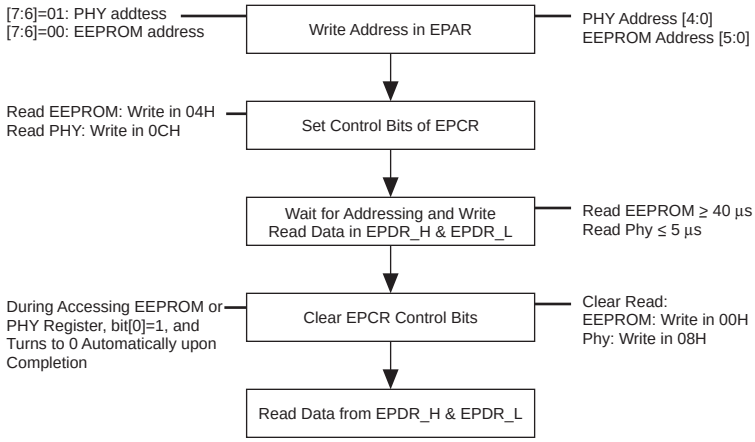
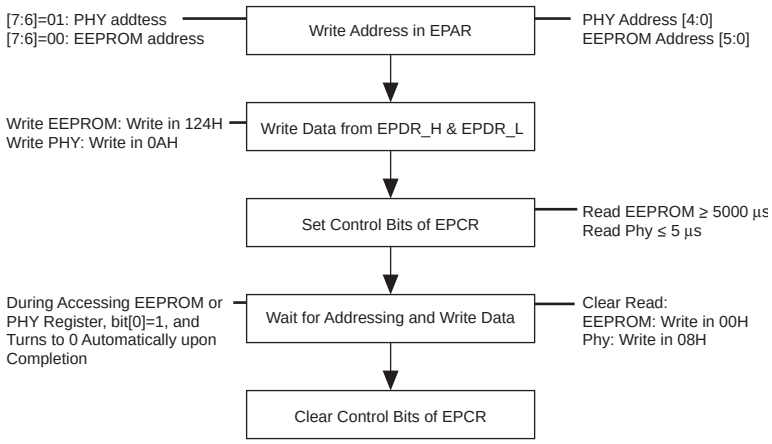


Figure 23. DM9000 Write Process



Develop DM9000A HAL Network Device Driver

Because the DM9000A device provides a complete bus interface, we need the Avalon bus and DM9000A interface logic in SOPC Builder. The DM9000A device communicate with the Nios II processor as an Avalon slave. Creating a HAL device driver includes creating device instances and registering the character device.

By referring to the HAL device driver development documentation and focusing on the lightweight IP (LwIP) driver structure, we defined the following structure DM9000A `alt_dev` structure:

```
typedef struct
{
   alt_lwip_dev_list   lwip_dev_list;
   int base_addr;
   int irq;
```

```
   u_char hwaddr[6];
   int index_offset;
   int data_offset;
   int dm9k_tx_space;
   int dm9k_linked;
   sys_sem_t arp_semaphore;
   sys_sem_t tx_semaphore;
} alt_avalon_dm9k_if;
```

When the Nios II processor starts running, the device is initialized in `alt_sys_init()`, allowing the program to identify the driver.

```
#define ALTERA_AVALON_DM9K_INSTANCE(name, dev) \       /*instantiate device*/
alt_avalon_dm9k_if dev = \
{\
   {\
      ALT_LLIST_ENTRY,\
      {\
         0,\
         name##_NAME,\
         alt_avalon_dm9k_init, \
         alt_avalon_dm9k_rx,\
      },\
   },\
   name##_BASE, \
   name##_IRQ,\
   { 0x00, 0x90, 0x00, 0xAE, 0x00, 0x01}, \
   0, 1, 2\
}

#define ALTERA_AVALON_DM9K_INIT(dev)alt_lwip_dev_reg(dev) //initialize
   //devices, register in HAL.
```

SOPC Builder

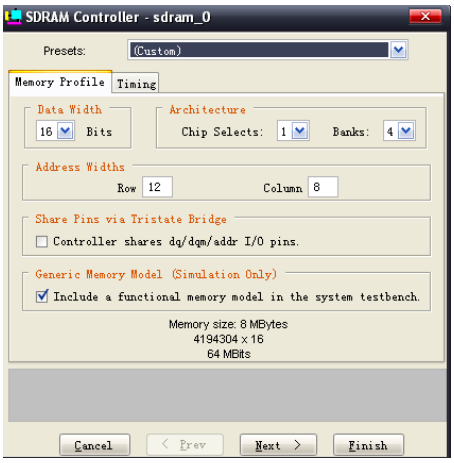
The Quartus II software's SOPC Builder integrates a hardware system in an FPGA, including writing the CPU, memory, interface IP blocks, timer, and Avalon bus in a hardware description language, and presenting it in the form of an IP block. Using the DE2 development board and considering the design functionality, we created the SOPC system.See Figures 24 and 25.

Figure 24. Nios II CPU Customization

Figure 25. 4-Mbyte Flash and 8-Mbyte SDRAM Controller Customization

Flash Control IP Customized Interface

SDRAM Control IP Customized Interface



The custom peripheral includes the A/D conversion control IP block, LCD control IP block, DM9000A bridge IP block, PS2 protocol resolution IP block, etc. See Figure 26.

Figure 26. Custom Peripheral Interface

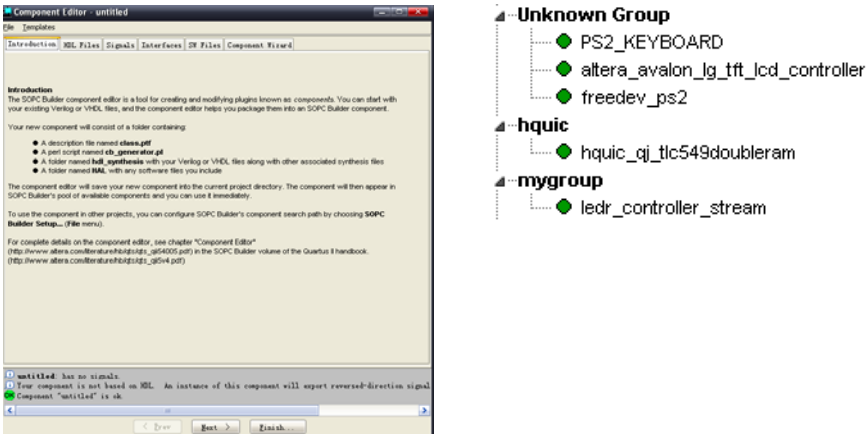


Figure 27 shows the system in SOPC Builder. The system clock is 100 MHz and peripheral clock is 50 MHz. The clock is derived from an external phase-locked loop (PLL) that generates a double frequency clock. Designing our embedded system with SOPC Builder minimized our development time.

Figure 27. SOPC Builder Interface

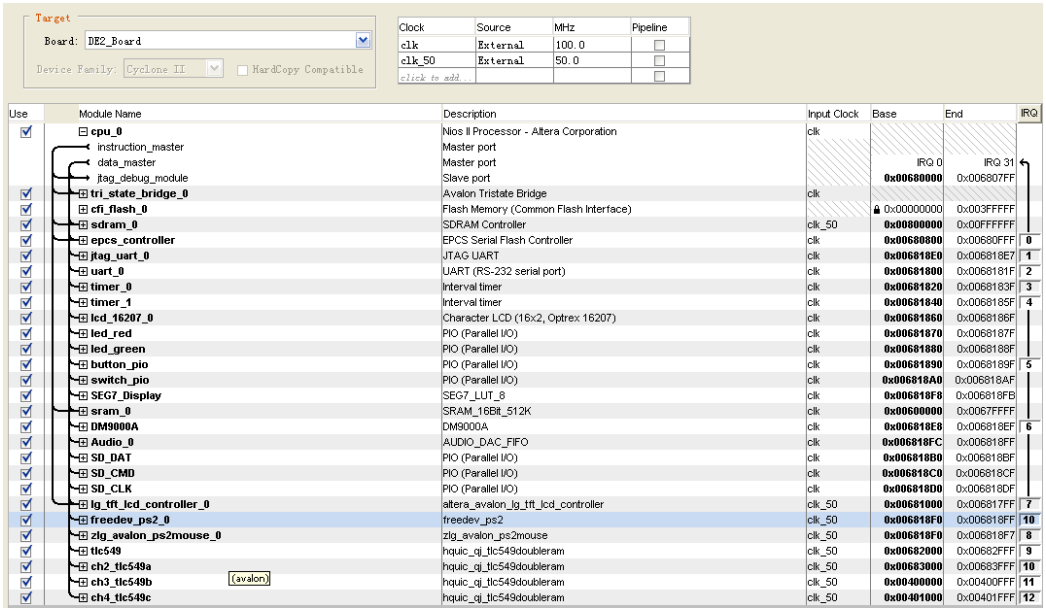


Figure 28 shows the SOPC Builder-generated schematic diagrams in the Quartus II software.

Figure 28. SOPC Builder-Generated Modules

We added a PLL in the Quartus II software to distribute pins, and we generated the hardware SRAM Object File (**.sof**) after compilation. This step completed our project's hardware platform, and we next entered the software phase.

Software Platform Design

This section describes the software development for our design.

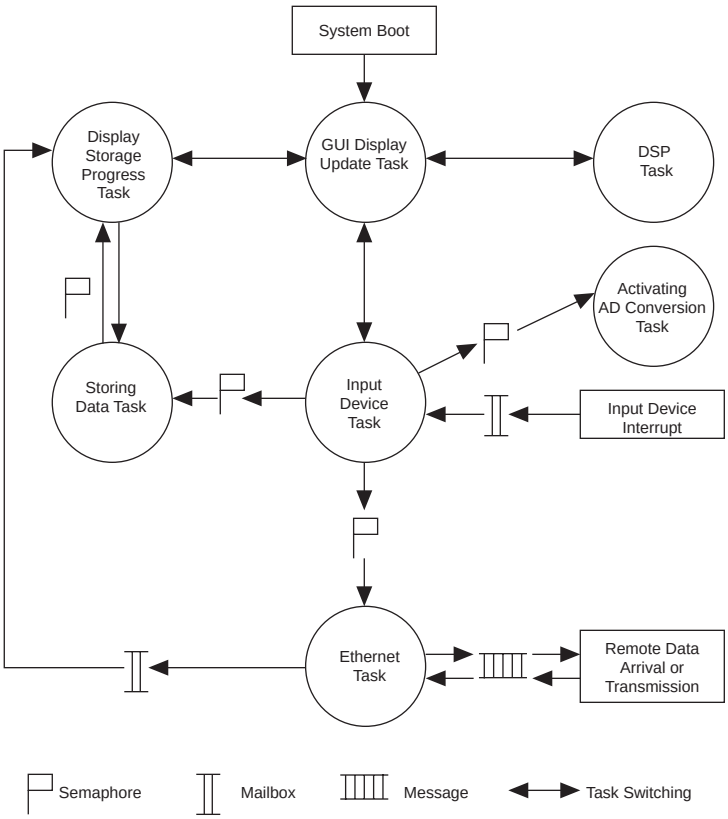
µc/OS-II Multi-Tasking Design

Because our system involves many tasks, such as collection, display, networking, storage, etc., we used the real-time µC/OS-II operating system to manage the entire system, resulting in smooth operation of the hardware modules and application program. µC/OS-II has already been migrated to the Nios II Integrated Development Environment (IDE), so we only needed to select it in the IDE. We divided the main tasks as:

- Display
- Collection
- Data storage
- Network tasks
- Signal processing
- Input device

Display is the highest priority task. All main tasks have other relative subtasks. Figure 29 shows the relationship of the main tasks.

Figure 29. System Task Relationships



The GUI display update task is the highest priority. An input device interrupt sends messages in the interrupt program. Then, the boot input device task and input device task judge messages. The system sends a semaphore to different operations, including the data storage, network, and A/D conversion tasks. The storing task stores data in the SD card and then the file system's internal task begins. The Ethernet task performs web server functions; it receives and sends messages to a remote PC and executes HTTP internal tasks to finish sending and reading web page data. The signal processing task implements data acquisition detection and analysis as well as data compression.

µC/GUI Migration

µC/GUI, a graphic support software for embedded applications, provides an application that has one graphic LCD with an effective graphical user interface (GUI) independent of processor and LCD controller. It can operate in a single-task or multi-task environment and works with any size physical or virtual display that uses an LCD controller or CPU. The modular design consists of layers of varied modules. One layer, called the LCD driver, contains all accesses to the LCD. Additionally, µC/GUI works for all CPUs because it is compiled purely in the ANSI C language.

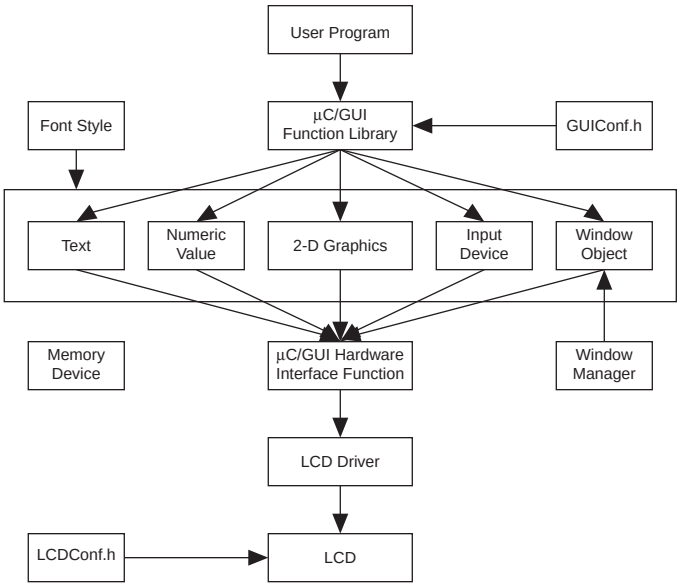
In past projects, we wrote data into the LCD display memory (with starting address 0x00f00000) to display graphics on the LCD. But this method had many problems—such as unstable graphics display, monotone color, only simple images, and difficult design—because the algorithm between the display memory data address and LCD display is complicated. In this project, we wanted to solve those issues by migrating µC/GUI to the Nios II processor.

Successful migration means that the public GUI source code can operate on the hardware platform we design, i.e., it acts according to our commands. Once we migrate µC/GUI, we can display images and graphics directly on the LCD by invoking functions from within application programs, such as an API

drawing function. We can create fascinating, innovative images and graphics, implement a multi-tasking LCD, e.g., multiple windows, controls, anti-aliasing, etc., and be free from basic tasks such as the graphic data location in the LCD display memory. Undoubtedly, this method will greatly facilitate our future LCD development.

Figure 30 shows the µC/GUI software system.

Figure 30. µC/GUI Software System



Modifying the **GUIconf.h** and **LCDconf.h** files is an important migration step (see Figure 31). Some of the files we modify are described below.

- *GUICONF.h*—In this file, we configure the GUI migration options for different operating systems. In our design, we configure migration to µC/OS-II and allow multi-tasking to invoke µC/GUI functions.

- *LCDconf.h*—In this file, we define various attributes related to hardware, such as the LCD size, color, and interface function. The LCD driver interprets µC/GUI functions into the LCD interface function defined in the **LCDConf.h** file, which is not applicable to the hardware connection. Using the driver, the µC/GUI and LCD hardware interface converts the hardware interface function into an LCD read/write function as defined in the **LCDConf.h** file.

- *LCDDDummy.c (LCD drivers)*—These two functions, `LCD-L0-SetPixelIndex(int x, int y, int PixelIndex)` and `LCD-L0-GetPixelIndex(int x, int y)`, constitute the basic low-level LCD drive function and connect directly with the hardware. Some basic functions such as `LCD-L0-DrawHLine`, `LCD-L0-DrawVLine` and `LCD-L0-FillRect` are also defined in driver. They are invoked from `LCD-L0-SetPixelIndex(int x, int y, int PixelIndex)` and `LCD-L0-GetPixelIndex(int x, int y)`.

Figure 31. LCD Configuration

```
#ifndef LCD-BUSWIDTH
#define LCD-BUSWIDTH (32)—LCD data line is 32 digits.
#endif
#define LCD-READ-MEM (off) IORD-32DIRECT (0X01F40000, (Off<<2))-for operating hardware
#define LCD-WRITE/MEM (Off, data) IOWR-32DIRECT (0X01F40000, Off*4, data)--- for operating hardware
#define LCD_WRITE_REG0(data)
IOWR_altera_AVALON_LG_LCD_CONTROLLER_CR(LG_TFT_LCD_CONTROLLER_0_BASE,data)
#define LCD_WRITE_REG2(data)
IOWR_altera_AVALON_LG_LCD_CONTROLLER_NBAR(LG_TFT_LCD_CONTROLLER_0_BASE,data)
#define LCD_READ_REG1
IORD_altera_AVALON_LG_LCD_CONTROLLER_SR(LG_TFT_LCD_CONTROLLER_0_BASE)     #define LCD_READ_REG3
IORD_altera_AVALON_LG_LCD_CONTROLLER_ISR(LG_TFT_LCD_CONTROLLER_0_BASE)

----Four #defines above are operation over four registers of LCD.
Initialize LCD controller
#define LCD_INIT_CONTROLLER() LCD_WRITE_REG2(0X01F40000);\
   LCD_WRITE_REG0(000001);
#endif
```

SD Card-Based File System Migration

Figure 32 shows the µC/FS structure, which, like µC/GUI, is public source code. Figure 33 shows the card-based layers.

Figure 32. µC/FS Structure

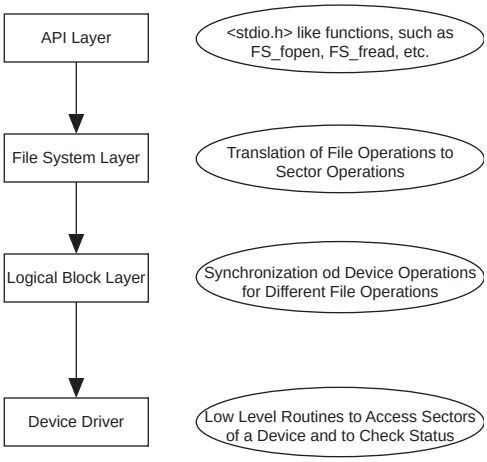
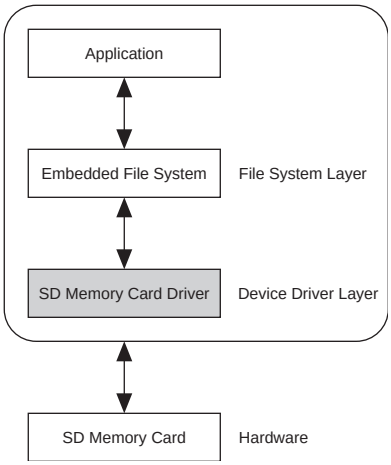


Figure 33. SD Card-Based FS Layer



Some key points when migrating the file system (FS) are:

- Because of µC/OS-II, we reduced unnecessary operating system (OS) source code in the FS to conserve storage space.
- We modified the **fs-port.h** file to implement a data type that the Nios II processor can recognize. For the **fs-conf.h** file, we modified the relevant FS configuration, such as the number of opened files, names of supported devices, etc.
- We determined the lowest level FS device driving function and added the SD card device-driven function.

In the SD card protocol, the host sends `CMD` first and then the card sends `RES`. If there is data to be transferred, it is transferred on the `DATA` line. Except for copyright protection commands, the SD protocol has 34 total commands. For users, configuring SD card means that the system reads/writes the register. The main registers include `CID`, `CSD`, and `OCR`. We used the C language to compile an SD driver in the Nios II IDE, i.e., one that performs initialization and data read/write functions.

In µC/FS, a low-level device driver invokes functions directly as shown in the following code and our SD card driver is implemented in the four functions listed.

```
const FS__device_type FS__SDdevice_driver=
{
"SD",
_GENDEV_DevStatus,
_GENDEV_DevRead,
_GENDEV_DevWrite,
_GENDEV_DevIoCtl,
};
```

We defined the card initialization process as `SD-card-init()` in µC/FS; `GENDEV-DevStatus( )` invokes the process to perform the following functions:

1. Reset the card and its control module, and keep card frequency at no more than 25 MHz during the reset process. The card can be reset using the `CMD` method (`CMD0.CMD52`).
2. Determine whether the input card is an SD card or MMC using `CMD55`.

3. Obtain the card ID (`CID`) by transmitting `CMD2`. The `CID` is integrated in the card and each card has only one `CID`. Once `CID` is obtained, the card can enter certification status.

4. The relative card address (RCA) is the unique symbol that controller uses to access the card. Dynamic distribution is available using `CMD3`.

5. Set the read/write block size. According to the FAT and FS settings, we can set a read/write block as 512 bytes.

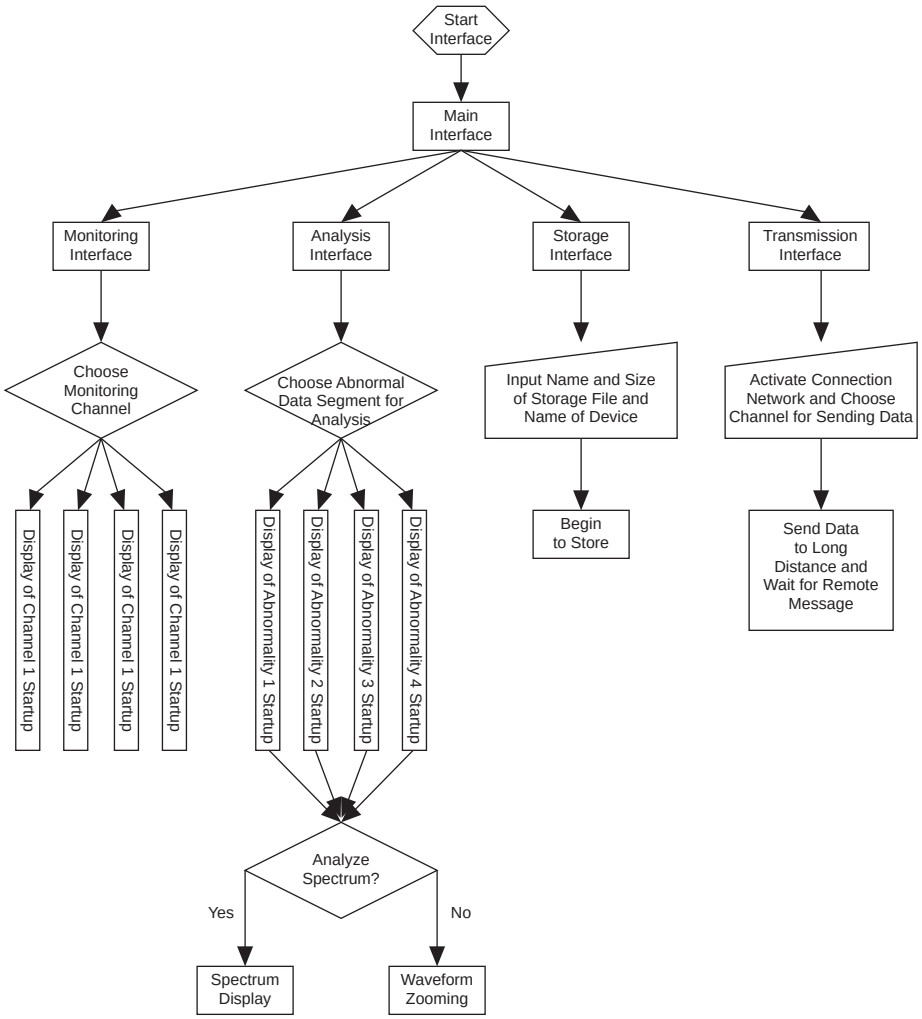
We enable the card reading/writing using the data block and every read/write is an integral multiple of the block. `CMD17`/`CMD18` and `CMD24`/`CMD25` read/write one or many data blocks over the card, respectively. In µC/FS, we implemented reading/writing in `SD-read-Iba(Unit, Sector, pBuffer)` and `SD-write-Iba(Unit, Sector, pBuffer)`.

Data always has an attached cyclical redundancy check (CRC) code. In µC/FS, CRC codes are implemented with the `GetCRC16()` function.

Operating Interface Design

All main monitor operations are merged onto the LCD; therefore, a user-friendly operating interface is key to the design. When we migrate the GUI, developing GUI-based operating interfaces will become faster and more efficient. Figure 34 shows the operating interface process.

Figure 34. Operating Interface Process



Web Server Design

The network communications design is based on the TCP/IP protocol. The key to successful communication is to embed the protocol into the system and migrate the network interface control chip driver to implement communication at the physical layer. At the application layer, we can implement communication by writing different applications according to the required services.

The LwIP protocol stack, is already integrated into the Nios II processor, accelerating the network development. Therefore, we only needed to focus on designing the network interface driver and developing the application.

LwIP Overview

LwIP was originally written for embedded system by Adam Dunkels of the Swedish Institute of Computer Science. It can be migrated to an OS or operated independently. LwIP has the following features:

- Supports IP forwarding with multiple network interfaces.
- Supports the Internet control message protocol (ICMP).
- Includes an experimental user datagram protocol (UDP).
- Includes congestion control, round-trip time (RTT) estimation, TCP of fast recovery, and fast retransmission.
- Provides a dedicated raw API for improving application performance.
- Includes an optional Berkeley interface, an API (in case of multi-threading).
- Supports the point-to-point protocol (PPP) in the latest version.
- Has increased IP fragment support in the latest version.
- Supports the DHCP protocol and dynamic IP address allocation.

To adapt to different operating systems, LwIP adds an OS-encapsulated layer between LwIP and the OS instead of using system calls and data structures relating to a certain OS. The layer provides a unified interface for OS service (timing, process synchronization, and messaging), uses semaphone for process synchronization, and mbox for messaging. The following code shows the OS encapsulated layer's main functions:

```
void sys_init(void)//system initialization
sys_thread_t sys_thread_new(void (* function)(void *arg), void *arg,int prio
   )//create a new process
sys_mbox_t sys_mbox_new(void)//create a new mailbox
void  sys_mbox_free(sys_mbox_t mbox)//release and delete a mailbox
void  sys_mbox_post(sys_mbox_t mbox, void *data) //send a message to the mailbox
void sys_mbox_fetch(sys_mbox_t mbox, void **msg)//wait for a message in the mailbox
sys_sem_t sys_sem_new(u8_t count)//create a new semaphore
void sys_sem_free(sys_sem_t sem)//release and delete a semaphore
void sys_sem_signal(sys_sem_t sem)//send a semaphore
void sys_sem_wait(sys_sem_t sem)//wait for a semaphore
void sys_timeout(u32_t msecs, sys_timeout_handler h, void *arg)//set a timeout event
void sys_untimeout(sys_timeout_handler h, void *arg)//delete a timeout event
```

The Nios II processor includes LwIP, including the source code and corresponding design environment (EDS). Using LwIP with the Nios II processor is based on the μC/OS-II multi-threading environment; therefore, we must implement μC/OS-II before using LwIP. The Nios II variant of LwIP is based on HAL, which includes a socket API function.

Connection between DM9000A Driver and LwIP

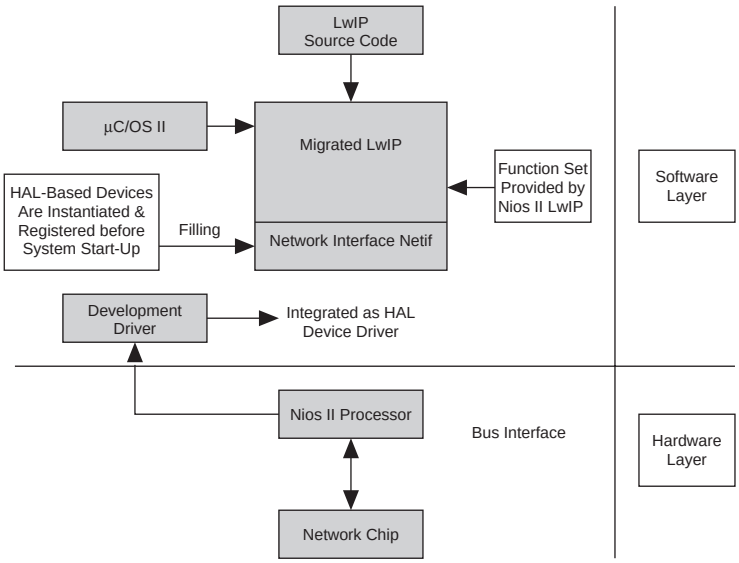
LwIP functions are invoked using the `netif` structure (see Figure 35).

Figure 35. netif Structure

```
r;
ik;

调用来传递包给协议栈. */
pbuf *p, struct netif *inp);

]来发送包给接口层. * /
t netif *netif, struct pbuf *p,
ıddr);

> 模块调用传递包给接口层 */
truct netif *netif, struct pbuf *p);

设置来表示设备状态. */

/** 硬件地址长度 */
 unsigned char hwaddr_len;

/** 硬件地址 */
 unsigned char hwaddr[NETIF_MAX_HWADDR_LEN];

 /** 最大传输数 */
 u16_t mtu;
 /** flags (see NETIF_FLAG_ above) */
 u8_t flags;

/** 连接类型 */
 u8_t link_type;

/** 描述名 */
 char name[2];

/** 接口号 */
 u8_t num;
};
```

After the DM9000A device driver is encapsulated into the HAL library, the initialization function registers functions related to the device driver in the `netif` structure. This process allows LwIP to recognize the DM9000A driver while invoking the network interface layer after system start-up, invoking the chip at the physical layer. See Figure 36.

Figure 36. Hierarchy Chart for Migration of Drivers to LwIP

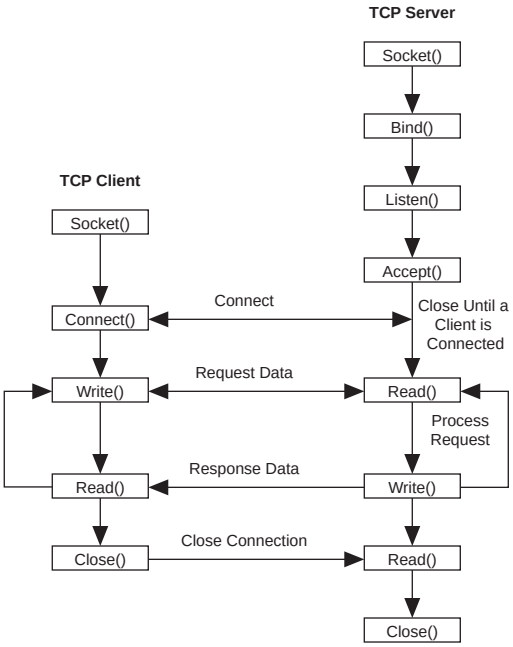


HTTP Services Application Development

After successfully migrating LwIP, we can use the socket API to design applications. The socket is made up of interfaces that forms the middleware abstraction layer for communication between the application layer and TCP/IP protocol suite. In design mode, a socket is a facade that hides the complex TCP/IP protocol suite: the user only deals with a simple set of interfaces while the socket organizes the data to adapt to the specified protocol.

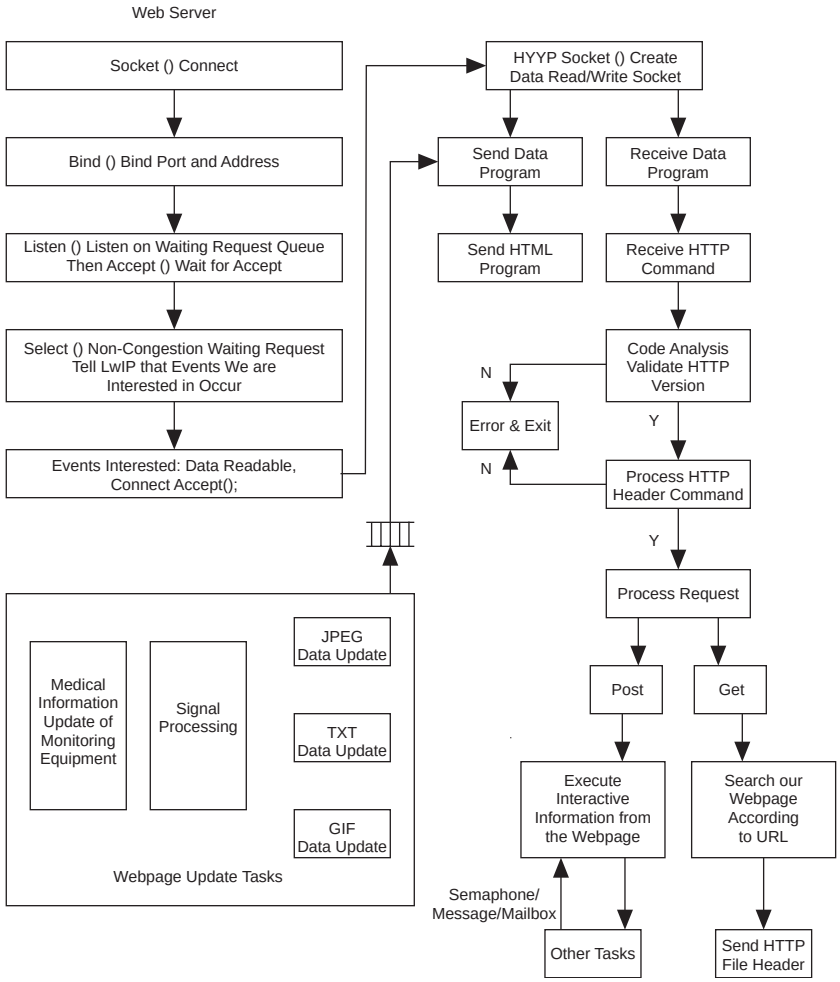
For network communication with a socket, the system initializes the socket, binds it to a port, listens on the port, invokes congestion acceptance, and waits for a client connection. If a socket is initiated and the server successfully connects to at least one client, the client/server connection is established. The client sends a data request while the server accepts and processes the request and sends response data to the client. The client reads the data, closes the connection, and the interaction ends. See Figure 37.

Figure 37. Client/Server Socket Communication Process



To allow remote PCs to obtain the monitoring equipment data and communicate with the equipment via a web page, we designed a simplified web server that provides services for web browser requests. We wrote our web page in HTML; therefore, the request-response transfer is based on HTTP. The client runs programs on the browser, connects to the server, and sends a request. The server responds with a status line (including the message's protocol version), a success or error code, and a message that consists of server information, entity information, and other possible content. The web uses client/server mode, so our design establishes socket connection. The monitoring equipment waits for a remote connection, analyzes HTTP upon connection, and starts HTTP tasks, including analyzing the HTTP request, executing requests, sending responses, closing the HTTP communication, etc. Figure 38 shows the HTTP application software process.

Figure 38. HTTP Application Software Process



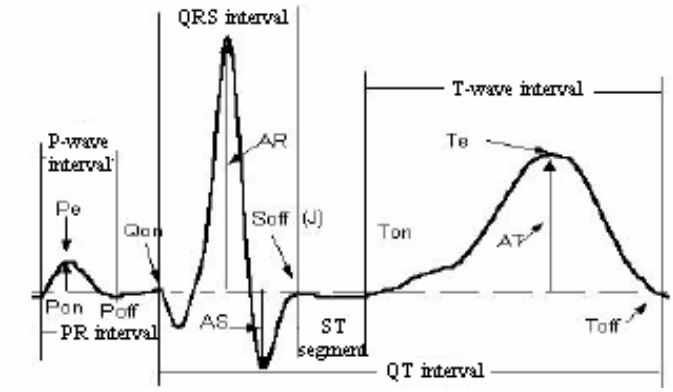
ECG Signal Detection and Compression Algorithm

During real-time acquisition and signal display, the monitoring equipment can automatically detect and store abnormal signals, allowing the user to analyze the situation and make an initial diagnosis. The following sections describe the detection, analysis, and compression of ECG signals.

Waveform Detection

ECG signal pre-processing is followed by detection of ECG characteristics, which is essential because the characteristic waveform directly reflects the heart's health status. A normal ECG waveform consists of a group of characteristic waves and its transition period. Each cardiac cycle includes a P-wave, PR interval, QRS combination waves, ST segment, T-wave, and QT interval as shown in Figure 39.

Figure 39. ECG Waveform



Detection of ECG characteristic points is the basis and key to automatic ECG detection and diagnosis. With this method the system must find various parameters, including the starting/ending points and voltages of each waveform in the ECG, the vertex and voltage of each characteristic waveform, the ST segment, etc. Then, the system analyzes each characteristic segment and performs a diagnosis of the monitored patient's heart status.

QRS waves are different from other ECG signals, accounting for a large proportion of energy. They are distributed in medium-high frequency compared to the other ECG signals, with a peak value between 10 and 20 Hz and distinct amplitude characteristics. Therefore, QRS waves are always located first when detecting ECG characteristic waveforms, and analysis of other waveforms are based on the R peak value, i.e., QRS detection is the premise for detecting all waveforms. The primary methods for waveform detection currently include:

- *Difference threshold*—Determine the QRS wave's negative edge by combining the first/second difference of the filtered signal application with the threshold. The system then locates the QRS wave vertex using a window and threshold.

- *Template matching*—Separate the QRS wave into a series of templates (segment or peak); the characteristic parameter of each template is indicated by a series of characteristic factors. The QRS wave is confirmed if the detection signal symbol sequence characteristics comply with those of the QRS template sequence. This method prevents recognition errors in the difference threshold method for a QRS wave with a lot of waveform variation and few parameter changes; however, the analysis is slow.

- *Wavelet analysis*—Wavelet transformation, a time-frequency local analysis method that has "micro" capability in areas with high signal frequencies, is particularly suitable for detecting the characteristic points of ECG signals. Singular points of a transient signal always contain important information. Detecting the location of the signal's singular points and determining the singularity is a concern and is a key part of a wavelet transformation application.

Our design uses a discrete dyadic wavelet transformation, which can be calculated using the following formula:

$$s_{2^j}f(n) = \sum_{k \in n} h_{0k} s_{2_{j-1}} f(n - 2^{j-1}k) \quad (1)$$

$$sw_{2^j}f(n) = \sum_{k \in n} h_{1k} s_{2_{j-1}} f(n - 2^{j-1}k) \quad (2)$$

QRS wave detection is a primary concern in ECG waveform detection as well as an important reference for arrhythmia diagnosis. The other ECG details are not analyzed until the QRS wave is confirmed. Based on further analysis of the detection principle for the singularity of wavelet transformation, comprehensive analysis, and comparison of the detection methods already discussed, we chose a quadratic differential wavelet to detect the ECG waveform. We designed the wavelet filter by taking the second derivative of a Gaussian function (i.e., Marr wavelet) as the generating function.

We created a Marr wavelet decomposition filter based on a scaling function and a wavelet function according to a two-scale equation. Table 8 shows the filter coefficients.

Table 8. Filter Coefficients

k	-5	-4	-3	-2	-1	0	1	2	3	4	5
h_{0k}	0.0032	-0.0132	0.0393	0.0450	0.2864	0.4317	0.2864	0.0450	0.0393	-0.0132	0.0032
h_{1k}	0.0039	0.0062	-0.0226	-0.1120	-0.2309	0.7118	-0.2309	-0.1120	-0.0226	0.0062	0.0039

The process for the whole R-wave detection algorithm is summarized as follows:

1. The discrete dyadic wavelet transformation is performed on ECG digital signal f(n) with a Marr filter coefficient according to the Mallat algorithm presented in the discrete dyadic wavelet transformation formula, and $s_{2_j}(n)$ and $w_{2_j}(n)$ (for j = 0, 1, 2, 3, and 4) are acquired.

2. Based on a signal segment with a clear waveform, the system determines the initial threshold R_{th}^{j} of the modulus maxima at different scales.

3. In the wavelet transformation $w_{2^4}(n)$ with scale a = 2^4, find all modulus maxima with a higher threshold, and obtain a set of locations n_k^4 (for k = 0, 1, 2, 3, ... N).

4. In the wavelet transformation with scale a = 2^3, find the modulus maximum in the neighborhood of n_k^4 (for k = 1, 2, ... N) with a threshold greater than ε_{th}^3 and with the same symbol as the wavelet transformation at n_k^4, and define its position as n_k^3. If there is more than one modulus maxima near n_k^4 at the scale a = 2^3, choose the one with the largest amplitude value; however, if the largest modulus maximum is smaller than 1.2 times other modulus maxima, choose the maximum point nearest to n_k^4. If no modulus maximum is found in the n_k^4 neighborhood, define n_k^3, n_k^2, and n_k^1 as zero. Then acquire a set of locations n_k^3 (where k = 0, 1, 2, 3, ... N). According to our experience, we take 10 ms as the neighborhood scope, i.e., if the signal sampling frequency is 360 Hz, a neighborhood of 10 ms around n_k^4 will be $0.01 \times f_1 = 0.01 \times 360 = 3.6 \approx 4$.

5. Similar to the process in step 4 above, find the modulus maxima locations at scale a = 2^2 and a = 2^1, respectively and obtain two sets n_k^2 (for k = 0, 1, 2, 3, ... N) and n_k^1 (for k = 0,1, 2, 3, ... N).

6. According to the modulus maxima location set for a characteristic scale, the set of modulus maximum series $\{ n_k^1, n_k^2, n_k^3, n_k^4$ (for k = 0, 1, 2, 3, ... N)$\}$ is acquired; excluding the modulus maximum series with value = 0, the rest series set is $\{ n_k^1, n_k^2, n_k^3, n_k^4$ (for k = 0, 1, 2, 3, ... N) $\}$.

7. Determine whether the time interval of two adjacent modulus maxima at scale a = 2^1 is larger than 1.7 times the average RR interval. If yes, halve the threshold in this time period and search for the R peak again.

8. Amend the modulus maxima according to refractory period and L.E. index to remove some pseudo R peaks. After steps 7 and 8, the set of rest modulus maximum series is { n_k^1, n_k^2, n_k^3, n_k^4 (for k = 0, 1, 2, 3, ... N) }.

9. Determine the location of the R wave in the original signal according to the set n_k^1 (for k = 0, 1, 2, 3, ... N).

Compression Processing

The ECG data compression methods include a direct method, transformation, and parameter extraction. The first two methods apply in the case of waveform restructuring. Compared to the direct method, the discrete cosine transformation (DCT) we used in this design features high quality, noise reduction, simple restructuring, and a smooth waveform. The main reason DCTs are used widely in ECG data compression today is the low compression ratio (typically 3 times) during application. Signals can be processed by segments according to the ECG characteristics to improve the compression effect.

To implement ECG data compression, we first adopted DCT compression data sequences and divided them into shorter sequences by frames. Considering the waveform completeness and data processing timeliness, the system takes the sampling data sequence of each cardiac cycle waveform as a data sequence frame. In actual data processing, the central point of two adjacent R peaks is the preferred frames break point. Because there are many proven QRS wave detection methods available, it makes sense to choose the QRS waveform as high frequency. After extracting the high frequency, the system uses linear difference (usually inserting 2 to 4 points) to connect the low-frequency waveforms at both ends into one segment through a smooth migration.

As the distribution of signal energy in the DCT domain features low frequency, high amplitude and high frequency, low amplitude while maintaining a certain fidelity of the restructured waveform, only M times the DCT components with low frequency need to be kept: C(0), C(1), ... C(M-1). We can then set a threshold E_{th} and require that the total energy of the M times components accounts for >= E_{th} of the total energy, i.e., E_f(M-1) x 100 >= E_{th}. The threshold E_{th} is determined according to the required fidelity of the restructured waveform. C(0), representing the direct current video component in the time domain waveform, only decides the horizontal baseline value of the restructured waveform, and can be rejected. Each DCT component that is kept should be converted to 8-bit integer data (7-bit integer data plus1-bit symbol data). Then we set the transformation scale factor of low frequency as 1 (which can be rejected) while keeping the ratio (two bytes) between the high frequency and low frequency scale factors.

Compared to other data compression methods, segmented DCT compression provides a high data compression ratio, high-fidelity waveform restructuring, and significant noise reduction.

System Integration and Effect

Because the system involves many software and hardware modules, we used software/hardware codesign. The following points are worth noting:

- We used IP design for hardware, all drivers are based on the HAL layer, the system clock is unified as 100 MHz, and the peripheral clock is 50 MHz.
- The OS tasks are prioritized according to different module weights.
- For the file system and GUI in the IDE, we defined the header file path using compiler options.
- We simulated the software algorithm in the MATLAB software before completing the implementation.

Figures 40 through 42 illustrate the system operation after software/hardware integration.

Figure 40. Demo

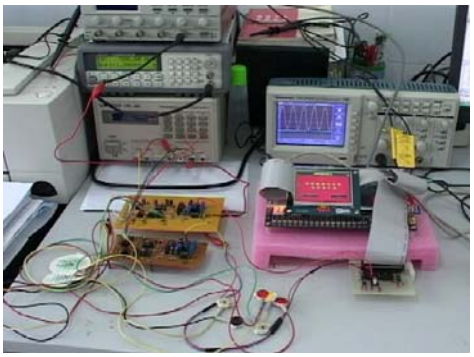
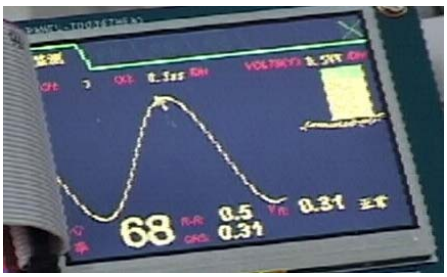
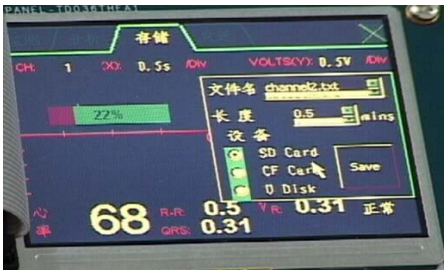


Figure 41. Remote Web Page Logon

Figure 42. Human-Machine Interface Operation





Design Features

Our design has the following features:

- *Conception*—While HHCE is becoming part of our lives, real home medical monitoring equipment is not available. HHCE is not effectively implemented because the existing portable monitoring equipment provides functions that are too simple or too expensive. Seeing this opportunity and feasibility, we chose to do the design. The Nios II processor gives our product superiority over its competitors in terms of size and price.

- *Function*—The product is modern with competitive features, such as the simultaneous measurement of multiple parameters, high-capacity data storage, network communication, more signal processing algorithms to support more functions, etc. Meanwhile, its diversified functions prove the flexibility and high performance of the Nios II system.

- *Hardware design*—SOPC techniques make our design simple and clear. We can complete each module independently and integrate them easily to form a system. We can embed several main devices (including the A/D converter and LCD IP blocks) for data transfer to reduce the CPU load while updating hardware modules rapidly. We can easily connect an IP block to the bus at the touch of a button, which was impossible for previous chip-based SOC systems. All IP blocks use the Avalon bus architecture and a unified synchronous clock.

- *Software algorithm*—Our design is complex, including many modules and complicated algorithms. These difficulties are also features of our design. Algorithms such as waveform detection, data compression, and spectrum conversion are not fully applied in existing monitoring equipment. With these algorithms, our product will be more specialized, providing users more medical diagnosis methods and allowing them to enjoy medical treatment at home.

- *Upgrades*—According to the requirements of different hospitals, communities, and homes, the multi-function, portable, medical monitoring system can be configured or upgraded quickly by selecting different front-end data acquisition modules and corresponding data processing CF cards without replacing the whole system platform.

- *Remote monitoring*—The networking function enables more comprehensive monitoring. With the Internet, users can communicate remotely at any time. Doctors can monitor patients remotely through the network to determine the monitored patient's health status while updating software or managing a database in real time. The Nios II protocol stack eased network development.

Conclusion

With this project, we gained a broader understanding of SOPC concepts and learned how to perform embedded development with the Nios II CPU. Altera's SOPC solution provides a powerful design platform that allows us to develop hardware, drivers, and applications to develop systems rapidly and efficiently.

The two-month effort we put into the contest brought us the final design as well as more experience. We found the "real sense" of the design process, including IP cores, pre-simulation, post-simulation, and application optimization. Embedded development requires developers to have a systematic view and patience; program debugging is time-consuming but beneficial.

We found that design tools such as the embedded logic analyzer, SOPC Builder, C-to-Hardware Acceleration (C2H) Compiler, IDE, and DSP Builder accelerated our development. Although tasks differ, we learned how things relate and felt the power of teamwork.

After participating in the Nios II design contest several times, we are impressed with the Nios II processor's flexibility and transparency, which distinguishes it from other embedded systems. We can

see the full transparency from the bottom layer implementation to the application layer design, supporting better system design than any other embedded system.

Until now, we have had many problems to solve, for example, how to accelerate an interface switch, how to truly connect databases, how to improve the system's DSP performance, and how to implement a more accurate biomedical signal waveform detection algorithm. These areas need our continuous efforts.

Finally, we would like to express our appreciation to all our teachers and classmates who helped us with the design, as well as our sincere gratitude to the host and judges of this contest. We will move ahead on our road of SOPC development.

Third Prize

FPGA-Based Clinical Diagnostic System using Pipelined Architectures in the Nios II Soft-Core Processor

Institution: **Jadavpur University, Calcutta**

Participants: **Shubhajit Roy Chowdhury**

Instructor: **Professor Hiranmay Saha**

Design Introduction

Clinical decision making is a complicated process. It is based on medical knowledge derived from medical books and literature and on data obtained from various clinical trials and diagnostic tests; however, it is also dependent on experience, judgment, and reasoning, which are functions of the human brain. In many situations, human decision making is not available, and in these situations, instruments play a major role in helping reduce human suffering.

In third-world countries, very few doctors are available in rural areas. For example, in India, 75% of qualified consulting doctors are in urban areas and 23% are in semi-urban areas, which leaves only 2% in rural areas where, unfortunately, nearly 78% of Indians reside. This imbalance has created a patient-doctor ratio of more than 10,000 patients for each doctor in rural India. Therefore, equipment that can predict imminent health hazards and can red-alert patients to contact a doctor for necessary care is urgently needed. Each doctor must handle a large number of patients; therefore, it would be useful for a doctor to be able to track patient data, especially because data and document preservation, such as investigation reports, is poor in rural areas. It would be useful to have a system that can be used effectively for a variety of chronic disease conditions such as renal dystrophy and diabetes mellitus. These types of diagnostic decision making can be performed with fuzzy logic. Initiated by Zadeh in 1965, fuzzy logic and fuzzy set theory are being used more and more in medical expert system applications.

The current research focuses on an FPGA-based smart processing system that can predict the patient's physiological state given the patient's past physiological data. The scheme can provide an alarm to the

relevant personnel, who would contact a physician at a remote site before the patient reaches a critical state. The physician would take the necessary actions to provide medical support to the patient. The smart processing system consists of blocks for fuzzification, inferencing, and defuzzification of patient data. It can handle patients' peripheral health screening, and help caregivers focus on the few critical patients who really need a physician's clinical assistance.

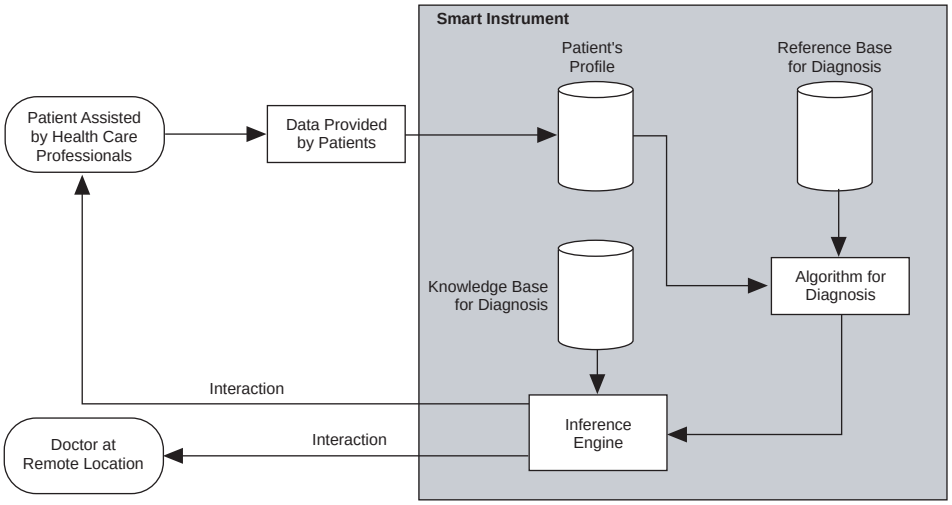
To reduce the combinational logic blocks required to implement the system, we implemented the division process required for normalizing membership functions using intelligent multiplication techniques. To make the computing system fast, we used pipelined data processing architectures. An FPGA implementation is useful in developing countries because of the low investment required compared to ASIC prototyping costs. Additionally, FPGAs are reprogrammable, which allows design improvements. This feature is important because it supports new structures, e.g., upgrading the current smart diagnostic system, supporting other diseases, mapping to other fields of human expertise, etc.

With the Nios® II soft-core processor, we can overcome design issues such as limited peripheral resources, difficult I/O configuration, complex hardware design, and software programming. This design also meets time sequence and function requirements, optimally uses the processor's resources, and greatly improves the overall system efficiency. Because the system has external memory and I/O, memory access is frequent. With the Nios II processor's user-defined peripherals, user-defined logic, and direct memory access (DMA), the design can easily access memory and move data when using SDRAM, SRAM, and flash memory. In our design, the patient profile is stored in flash memory. By combining the requirements of both software and hardware in a coordinated development process, Altera's SOPC solution is the best choice: it can fully showcase the advantages of an FPGA's logic control and data processing capabilities. This design approach allows for flexible system configuration, provides simple, convenient development, supports various processing modes, and offers powerful data processing capacity at low cost.

Function Description

The designed system has a pipelined smart processing unit that can predict the patient's future pathophysiological state based on past pathophysiological data. Figure 1 shows the functional architecture of a diagnostic system that includes a smart agent that we plan to implement.

Figure 1. Smart Agent Based Diagnostic System Functional Architecture



In Figure 1, the smart agent is represented by a fuzzy system. At least three entities are required in this concept of diagnostics:

- The healthcare personnel provide data by measuring the patients' health parameters.
- The physician interacts with the smart instrument and confirms or denies the diagnosis made by the smart instrument.
- The smart instrument performs the diagnosis at regular times and predicts future states of the patient using fuzzy logic.

Based on previously fed data, the smart instrument can give an early signal of deterioration in the patient's health status and indicate an imminent emergency situation. Initially the data provided by the patients under the assistance of health care professionals is stored in a patients' profile. The data from the patients' profile is subjected to a diagnostic process using a knowledge base for diagnosis. The diagnosis process is based on fuzzification of patient data. The inference engine makes a prediction about the future physiological state of the patient based on the fuzzified data. Based on the prediction, the smart system gives an indication about the possible next physiological state of the patient.

The smart processor we developed can fuzzify and defuzzify patient data. The patient data cannot always be trusted because it relies on the quality and accuracy of measuring units and the technician's skill. Moreover, based on a single bit of data, it would be highly difficult to make an accurate decision about the future pathophysiological state of the patient, particularly in a chronic case. Therefore, we fuzzified the patient data to transform periodic measures into likelihoods that the pathophysiological parameter of the patient is high, low, or moderate compared to a reference value set.

As an example study, this project analyzes patient renal data and predicts the patient's future physiological state. The system calculates the patient's body mass index (BMI) using the patient's height (in feet) and weight (in kilograms). Because doctors are more interested in knowing whether the pathophysiological risk parameters of a patient is high, moderate, or low as well as the patient's physiological parameter trends, it is more useful to represent the patient's pathophysiological risk parameters as linguistic variables instead of ordinary variables. Then, we can use fuzzy logic to build a model that predicts the fuzzy set (low, moderate, or high) in which the patient's particular risk parameter (e.g., B.M.I, glucose, urea, creatinine, and blood pressure) lies to be referenced at the next reading of that patient data. For this purpose, we used triangular and trapezoidal fuzzy operators. A typical triangular function takes the form:

$$A(x; a, m, b) = \max\{\min[(x - a)/(m - a), (b - x)/(b - m)], 0\}$$

Similarly, a typical trapezoidal function takes the form:

$$A(x; a, m, n, b) = \max\{\min[(x - a)/(m - a), 1, (b - x)/(b - n)], 0\}$$

We determined the membership function in accordance with the ranges and tolerance limits set up by the World Health Organization. Figure 2 shows the plot of the membership functions defined above.

Figure 2. Plots of the Membership Functions

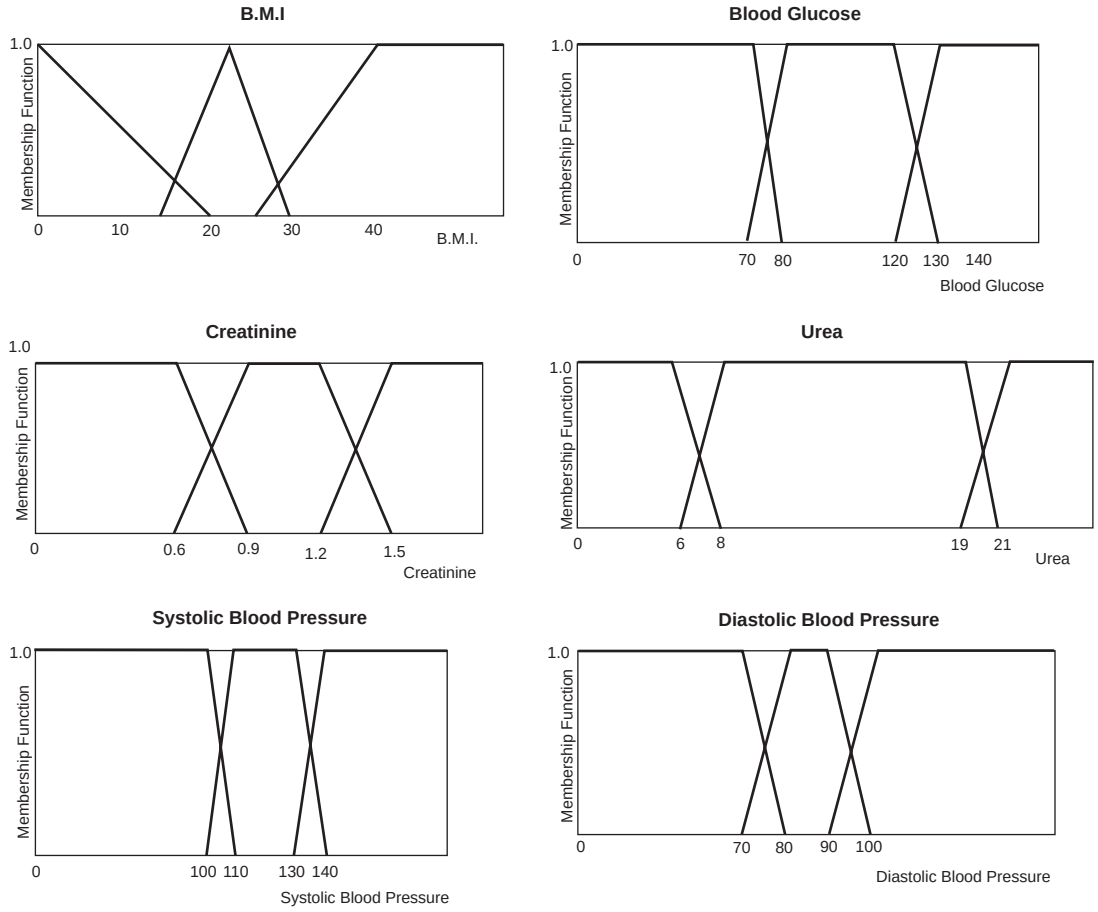


Figure 2 depicts the cognitive frames used for fuzzy modeling patients' BMI, blood glucose, blood urea, blood creatinine, systolic, and diastolic blood pressure data. It is obvious that all low, moderate, and high risk parameter ranges (modeled here as fuzzy sets) of the patients fall in the same universe of risk parameter values.

Inferencing involves deciding whether the patient is in a normal condition, is heading towards a moderately critical condition, or is in a severely critical condition. Inferencing is done by taking the diagnostic algorithm's next possible state output at different points in time. Typical rules for inferencing take the following forms:

- *R1*—If (BMI is high) and (glucose is high) and (urea is high) and (creatinine is high) and (systolic blood pressure is high) and (diastolic blood pressure is high) then the (patient's renal condition is severe).
- *R2*—If (BMI is high) or (glucose is high) or (urea is high) or (creatinine is high) or (systolic blood pressure is high) or (diastolic blood pressure is high) then the (patient's renal condition is moderately critical).
- *R3*—If (glucose is high at time Ti) and (glucose is low at time Tj) and (Ti ≠ Tj) then the (patient should go for glycosylated hemoglobin).

- *R4*—If (BMI is moderate) or (glucose is moderate) or (urea is moderate) or (creatinine is moderate) or (systolic blood pressure is moderate) or (diastolic blood pressure is moderate) then the (patient's renal condition is normal).

and so on.

Defuzzification involves taking a crisp action based on the inference drawn, and can be implemented by illuminating an LED or by outputting data. In our scheme, LEDs indicate that the patient's state is approaching criticality.

The system has two rule bases. The knowledge base contains rules for inferencing based on the patient data set currently received and the already stored patient data. The reference base contains the reference values for fuzzifying the patient data. Based on these values, the system computes the membership function values of low, moderate, and high for the patients'pathophysiological parameters at different points in time. The diagnostic algorithm uses these values to compute the possibility that the next pathophysiological data will be low, moderate, or high.

The diagnosis algorithm computes the time-weighted mean of the membership functions of the patient's pathophysiological data. The possibility that the next pathophysiological data will be low, moderate, or high is computed as:

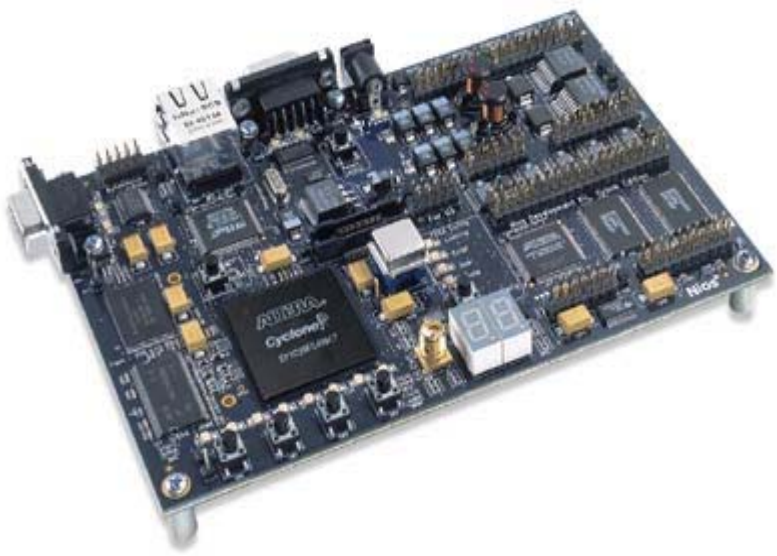
$$P_R(x) = \frac{\sum_{i=1}^{n} i\mu(x)}{\sum_{i=1}^{n} i}$$

where the summation is done from i = 1 to n, and the value of n is the sequence number of the time instant at which the current pathological data of the patient is taken and R ∈ {low, moderate, high}. $\mu(x)$ is $\mu l(x)$ $\mu m(x)$, or $\mu h(x)$ acordingly as the membership function refers to a low, moderate, or high fuzzy set, respectively. To predict the fuzzy set in which the next state input of a certain pathophysiological parameter will lie, the value of $P(x)$ is considered for which $P(x) \geq P_R(x)$.

We implemented the smart data processing system on an Altera® Cyclone® EP1C6Q240C8 FPGA. We could also have implemented the system using software; however, this solution would require a powerful computer to run the software at reasonable speed and accuracy. A powerful computer is too costly and would require a steady supply of electricity in rural sectors. The cost would be an impediment to adoption of the smart diagnostic system in the rural health care centers in third-world countries. Additionally, a software-only solution could take longer to process if we constructed more complex systems covering many different infected parts of the human body, However, a few milliseconds delay cannot be considered important for medical diagnosis. The main reason for a hardware-based implementation is the need for an inexpensive, portable diagnostic system. The main disadvantages of an ASIC-based solution is the high development cost and the low reconfigurability. An FPGA solution ensures that new changes in the proposed diagnostic algorithm can be mapped onto the hardware without having to make costly changes.

Figure 3 shows the UP3 board on which the FPGA is mounted.

Figure 3. UP3 Board



We generated an SRAM Object File (**.sof**), which is a bitstream pattern, using the system's VHDL model and downloaded it to the FPGA via the JTAG interface and ByteBlaster II cable. The configuration data is stored in the FPGA's SRAM.

The input is sent to the FPGA using push-button switches. We need 12 push-button switches but the UP3 board does not have that many. Therefore, we developed our own printed circuit board (PCB) containing the required push-button switches and connected it to the UP3 board. The FPGA receives a 0 input when the user presses a switch. The binary data entered using the switches are converted into real numbers for computation using conversion weights stored in a look-up table (LUT) implemented on the SDRAM.

Using these parameter values, the system computes the corresponding membership function values μ_l, μ_m, and μ_h. These values refer to whether the pathological parameter value is in the low, moderate, or high fuzzy set, respectively. The values are stored in the CMOS flash memory using the Nios II soft-core processor. Based on these values, the system computes the possibility that the values of the different physiological parameters are low, moderate, or high. The maximum of these three possibilities at any instant suggests the patient's next possible physiological state.

The output therapeutic decision is displayed on UP3 board's 7-segment display. The 7-segment display indicates whether the pathological parameters will be low, moderate, or high values. Because, there are four 7-segment displays in the final system and the board only has port available, we used a 4-bit output called SCAN (0 to 3). The SCAN's bit lines are connected to the cathodes ofthe LED 7 segment display, which selects the 7-segment LED in time-shared mode. The display codes are stored in a LUT.

The user can reset the system at any time using a push-button switch. Two LEDs connected in common anode mode indicate whether the patient's condition is moderately critical (MC) or severely critical (SC). The system has a battery back-up that provides a continuous power supply to overcome the FPGA volatility.

The FPGA system implementation is very attractive because FPGAs are reconfigurable and becoming more economical and faster as time goes on. We tested the FPGA implementation with a patient to compare the decision result of the physician vs the smart agent.

To test the system, we analyzed data from a 5-foot tall, 42-year-old patient. Tables 1 through 6 show the results. In the tables, AS refers to the actual physiological state of the patient. In the actual experiment, the patient's weight (Wt), glucose, urea, creatinine, systolic, and diastolic blood pressure data taken at 10-day intervals are input to the system at time Ti (where i = 0,1,…, 9). Initially, the height of the patient is also given. Based on the height and weight data, the system computes the patient's BMI at different times. Using these parameters, the system, computes the corresponding membership function values μ_l, μ_m, and μ_h.

In the AS clumn, M indicates a moderate risk, H* indicates high risk that still falls within the tolerance limits of moderate value, and H indicates strictly high risk.

Table 1. BMI Data Results

Time	Weight	BMI	μ_l	μ_m	μ_h	P_l	P_m	P_h	AS	PNS
T1	64.1	27.97	0.00	0.29	0.14	0.00	0.29	0.14	M	M
T2	66.2	28.31	0.00	0.24	0.16	0.00	0.26	0.15	H*	M
T3	66.8	28.57	0.00	0.20	0.18	0.00	0.23	0.17	H*	M
T4	67.5	28.87	0.00	0.16	0.21	0.00	0.20	0.18	H*	M
T5	66.9	28.61	0.00	0.19	0.19	0.00	0.18	0.17	H*	M
T6	67.8	29.00	0.00	0.14	0.21	0.00	0.14	0.18	H*	H
T7	68.2	29.17	0.00	0.12	0.23	0.00	0.11	0.20	H*	H
T8	69.5	29.73	0.00	0.04	0.27	0.00	0.09	0.22	H*	H
T9	70.5	30.15	0.00	0.00	0.29	0.00	0.07	0.24	H	H
T10	70.6	30.62	0.00	0.00	0.33	0.00	0.06	0.25	H	H

Table 2. Glucose Data Results

Time	Glucose	μ_l	μ_m	μ_h	P_l	P_m	P_h	AS	PNS
T1	120	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T2	125	0.00	0.50	0.50	0.00	0.67	0.33	H*	M
T3	128	0.00	0.20	0.80	0.00	0.70	0.30	H*	M
T4	127	0.00	0.30	0.70	0.00	0.54	0.46	H*	M
T5	128	0.00	0.20	0.80	0.00	0.33	0.57	H*	H
T6	128	0.00	0.20	0.80	0.00	0.29	0.71	H*	H
T7	128	0.00	0.20	0.80	0.00	0.26	0.74	H*	H
T8	129	0.00	0.10	0.90	0.00	0.23	0.77	H*	H
T9	129	0.00	0.10	0.90	0.00	0.20	0.80	H*	H
T10	131	0.00	0.00	1.00	0.00	0.16	0.84	H	H

Table 3. Urea Data Results

Time	Urea	μ_l	μ_m	μ_h	P_l	P_m	P_h	AS	PNS
T1	17.0	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T2	17.5	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T3	18.7	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T4	19.1	0.00	0.95	0.05	0.00	0.98	0.02	H*	M
T5	20.7	0.00	0.15	0.85	0.00	0.70	0.30	H*	M
T6	20.6	0.00	0.20	0.80	0.00	0.56	0.44	H*	M
T7	20.8	0.00	0.10	0.90	0.00	0.44	0.56	H*	H
T8	20.9	0.00	0.05	0.95	0.00	0.36	0.64	H*	H
T9	20.9	0.00	0.05	0.95	0.00	0.29	0.71	H*	H
T10	21.0	0.00	0.00	1.00	0.00	0.24	0.76	H	H

Table 4. Creatinine Data Results

Time	Creatinine	μ_l	μ_m	μ_h	P_l	P_m	P_h	AS	PNS
T1	1.0	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T2	1.1	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T3	1.2	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T4	1.3	0.00	0.67	0.33	0.00	0.87	0.13	H*	M
T5	1.4	0.00	0.33	0.67	0.00	0.69	0.31	H*	M
T6	1.4	0.00	0.33	0.67	0.00	0.59	0.41	H*	M
T7	1.4	0.00	0.33	0.67	0.00	0.52	0.41	H*	M
T8	1.4	0.00	0.33	0.67	0.00	0.48	0.52	H*	H
T9	1.8	0.00	0.00	1.00	0.00	0.38	0.62	H	H
T10	2.4	0.00	0.00	1.00	0.00	0.31	0.69	H	H

Table 5. Systolic Blood Pressure (SBP) Data Results

Time	SBP	μ_l	μ_m	μ_h	P_l	P_m	P_h	AS	PNS
T1	128	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T2	131	0.00	0.90	0.10	0.00	0.93	0.07	H*	M
T3	132	0.00	0.80	0.10	0.00	0.87	0.13	H*	M
T4	136	0.00	0.40	0.20	0.00	0.68	0.13	H*	M
T5	137	0.00	0.30	0.70	0.00	0.55	0.45	H*	M
T6	138	0.00	0.20	0.80	0.00	0.45	0.55	H*	H
T7	139	0.00	0.10	0.90	0.00	0.36	0.64	H*	H
T8	140	0.00	0.00	1.00	0.00	0.28	0.72	H	H
T9	140	0.00	0.00	1.00	0.00	0.23	0.77	H	H
T10	143	0.00	0.00	1.00	0.00	0.18	0.82	H	H

Table 6. Diastolic Blood Pressure (DBP) Data Results

Time	DBP	μ_l	μ_m	μ_h	P_l	P_m	P_h	AS	PNS
T1	87	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T2	88	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T3	90	0.00	1.00	0.00	0.00	1.00	0.00	M	M
T4	94	0.00	0.60	0.40	0.00	0.84	0.16	H*	M
T5	96	0.00	0.40	0.60	0.00	0.69	0.31	H*	M
T6	98	0.00	0.20	0.80	0.00	0.55	0.45	H*	M
T7	98	0.00	0.20	0.80	0.00	0.46	0.54	H*	H
T8	97	0.00	0.30	0.70	0.00	0.42	0.58	H*	H
T9	100	0.00	0.00	1.00	0.00	0.34	0.66	H	H
T10	101	0.00	0.00	1.00	0.00	0.28	0.72	H	H

Based on these membership function values, the possibilities the values of the different pathophysiological risk parameters will be low, moderate or high has been computed by the system (see Table 7). The AS and PNS subscripts are the first letter of the corresponding risk parameter. For example, AS_B refers to the patient's state based on BMI data.

Table 7. Smart Agent Decision Results

Time	AS_B	PNS_B	AS_G	PNS_G	AS_U	PNS_U	AS_C	PNS_C	AS_D	PNS_D	AS_S	PNS_S	SC
T1	M	M	M	M	M	M	M	M	M	M	M	M	0
T2	H*	M	M	M	M	M	M	M	M	M	H*	M	0
T3	H*	M	H*	M	M	M	M	M	H*	M	H*	M	0
T4	H*	M	H*	H	H*	M	H*	M	H*	M	H*	M	0
T5	H*	H	H*	H	H*	M	H*	M	H*	M	H*	M	0
T6	H*	H	H*	H	H*	M	H*	M	H*	M	H*	H	0
T7	H*	H	H*	H	H*	H	H*	H	H*	H	H*	H	1
T8	H*	H	H*	H	H*	H	H*	H	H*	H	H	H	1
T9	H	H	H*	H	H*	H	H	H	H	H	H	H	1
T10	H	H	H	H	H	H	H	H	H	H	H	H	1

Although the system gives a crisp decision regarding the patient's future pathophysiological state, the point is that the system predicts a critical state at time T7, which is well before a clinically overt criticality occurs at time T9. The system can thus be deployed in a variety of telediagnostic environments, in which health care professionals provide support services without a doctor present.

Performance Parameters

Our system has the following performance parameters:

- *Power supply*—DC voltage 5 V and the operating current is 175 mA.
- *Operating temperature*—5^{o} to 45^{o} C and relative humidity or 8% to 95%.
- *Data input*—Data is input via push-button switches. The input data is converted into real numbers for the ease of computation.
- *Data output*—The output data is displayed on the 7-segment display. A patient's critical state is signaled by a glowing LED.
- *Storage*—The patient data is stored in a TC58FVB160AFT CMOS flash memory device.

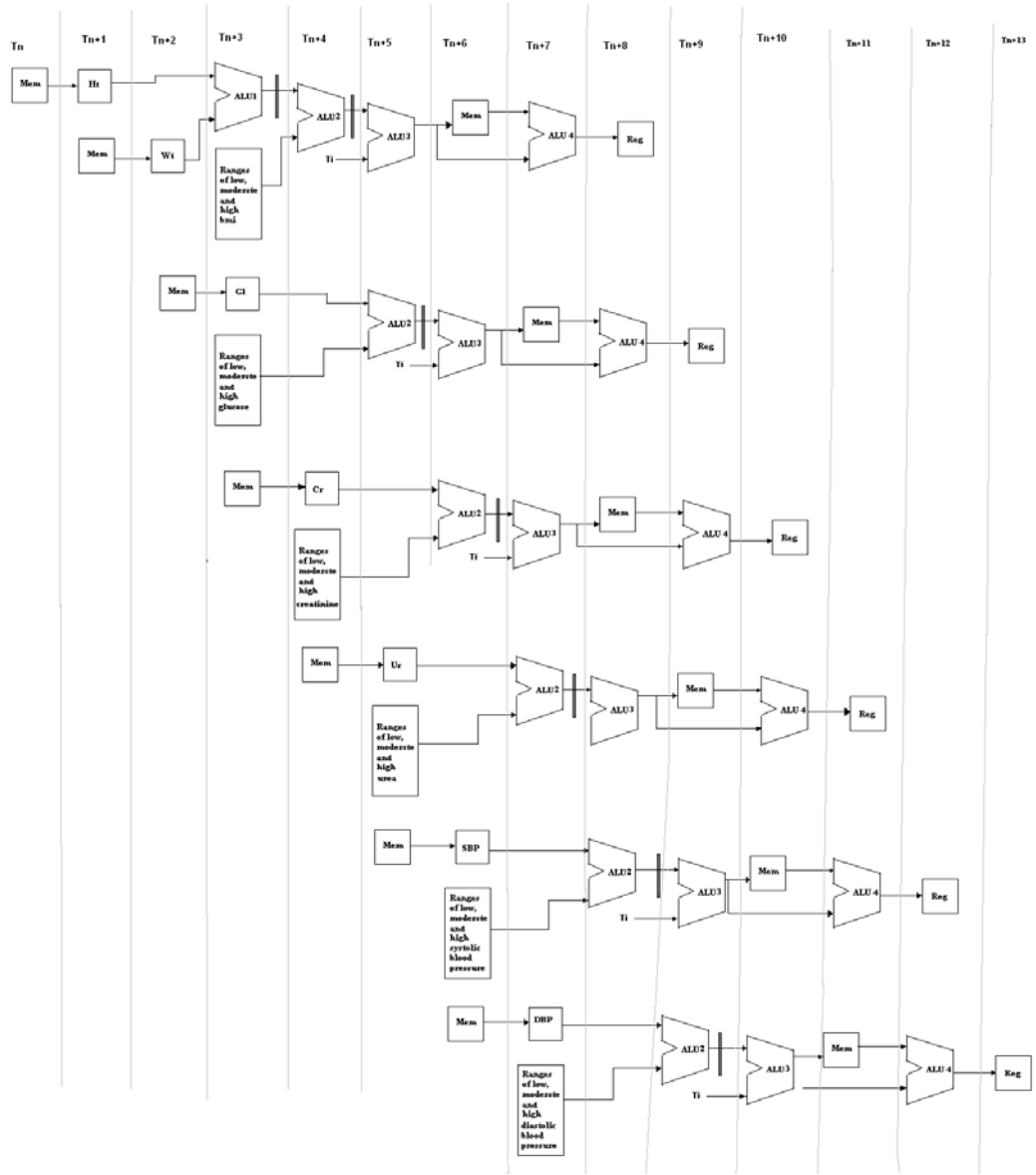
We used the Nios II processor to manage the FPGA's internal resources, define the time sequence requirements for data processing, and handle display and control of the system. Additionally, we needed to access multiple peripherals frequently from the main system, and the Nios II processor helped us to improve the system's overall operational efficiency. The Nios II functions are descibed below:

- It is the main processor, and controls the whole system logic.
- It handles the transfer of patient data between FPGA and CMOS flash memory.
- It handles all instructions to the FPGA's internal logic through user-defined programmable I/O (PIO) peripherals.

Design Architecture

For fast computation, we implemented a finite state machine with pipelined data processing on the FPGA. Figure 4 shows the pipelined architecture.

Figure 4. Pipelined Architecture Implemented in FPGA



The system has four arithmetic logic units (ALUs):

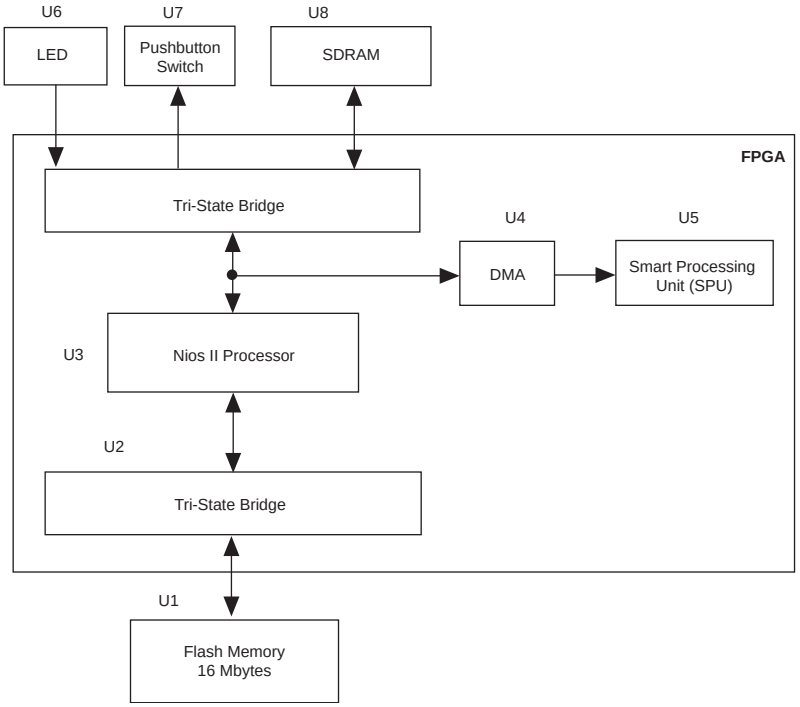
- ALU1 computes the BMI using height and weight data.
- ALU2 computes the membership function values using the instantaneous values of the pathophysiological parameters.

- ALU3 computes the probability that low, moderate, or high pathophysiological parameters will occur.
- ALU4 decides whether the next pathophysiological parameter reading is low, moderate, or high.

With a pipelined architecture, the system can compute the decision for all pathophysiological parameters in 14 clock cycles vs. the 44 clock cycles required in an unpipelined architecture. This difference increases performance about 3.14 times.

Figure 5 shows the system architecture including the Nios II processor.

Figure 5. Medical Diagnostic System Architecture Co-Design



The design's main modules are:

- *U1*—2-Mbyte flash memory containing the patient data.
- *U2*—Tri-state bridge.
- *U3*—Nios II processor.
- *U4*—DMA controller configured to feed the smart processing unit (U5) with patient data.
- *U5*—Smart processing unit based on the pipelined architecture discussed previously.
- *U6*—Four 7-segment displays and two LEDs for displaying the output results.
- *U7*—11 push-button switches for entering data.
- *U8*—SDRAM LUT that stores the display codes and conversion weights.

The hardware/software co-design involves the following steps:

1. Develop the system algorithm.
2. Implement the hardware peripherals using hardware description languages.
3. Verify the hardware peripherals' functionality using the ModelSim software.
4. Synthesize the peripherals using the Leonardo Spectrum synthesis tool.
5. Perform layout and timing analysis of the hardware peripherals using the Quartus II software.
6. Implement the algorithm in the Nios II processor.
7. Use the Nios II Integrated Development Environment (IDE) to connect the Nios II processor with the hardware peripherals.
8. Build and load the Quartus® II project onto the FPGA.

Figure 6 shows the data processing software flow chart.

Figure 6. Software Algorithm Flow Chart

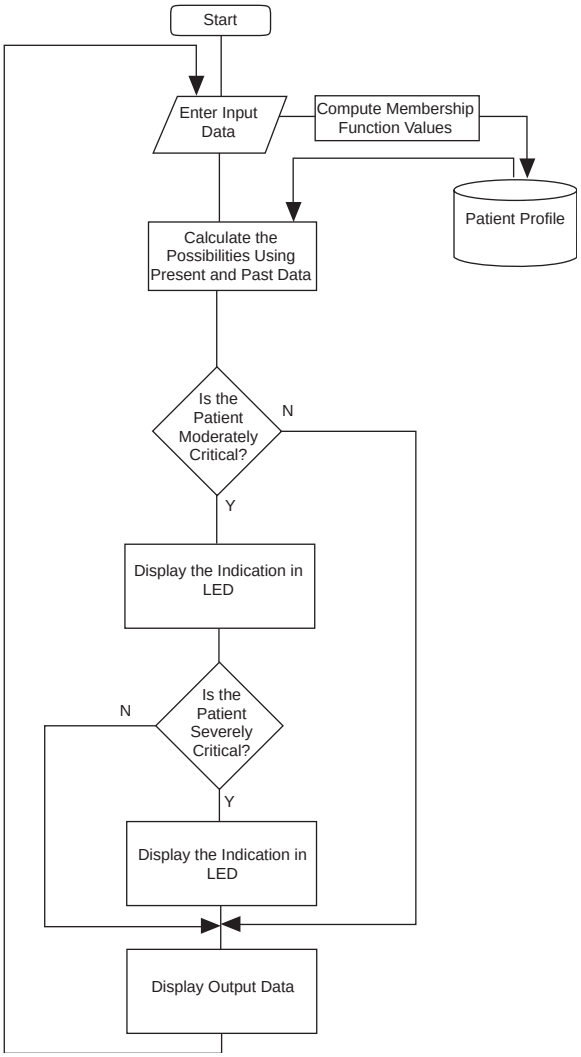
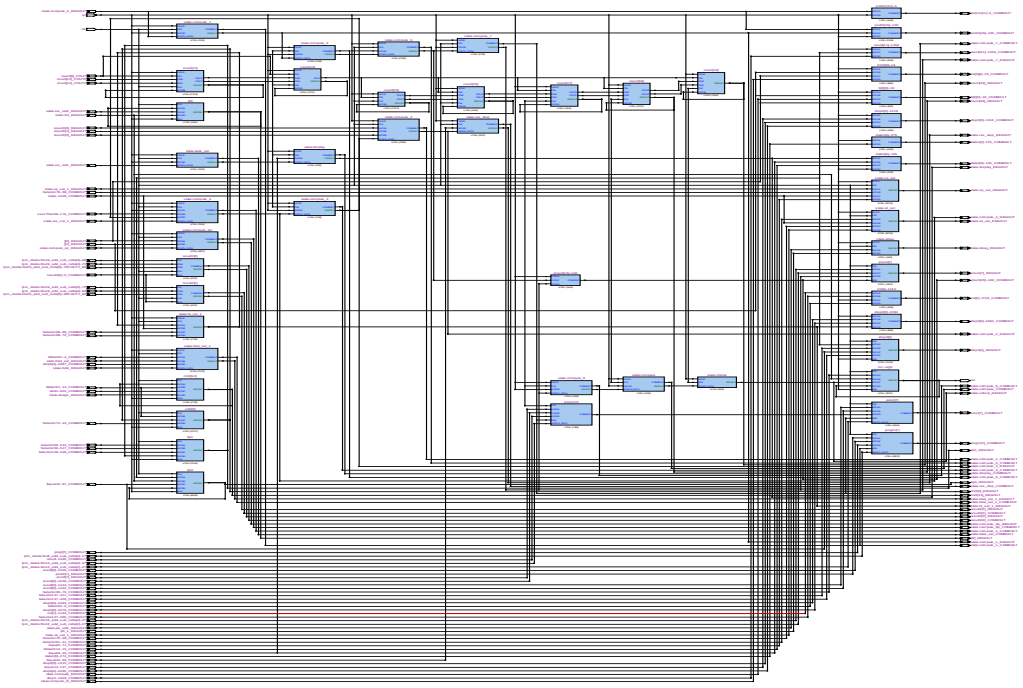


Figure 7 shows the smart agent's schematic.

Figure 7. FPGA-Based Smart Agent Schematic



Design Description

Using Altera's UP3 development board, we were able to design most of the system functions, and test and simulate all functions. Additionally, we designed a few circuit boards for design and test. We performed our system design and testing using the following steps:

1. Used SOPC Builder to access peripheral storage on the UP3 development board.

2. Referred to UP3 board examples and testing documents, and learned about the Nios II architecture, C language software programming in the IDE, and online programming and debugging for the device.

3. Implemented user-defined peripherals and logic on UP3 development board.

4. Implemented debouncing of data entered via the push-button switches.

Hardware Implementation

Our hardware design task was to combine the UP3 board testing methods and our circuits, and then design the necessary hardware modules to develop a medical diagnostic system that implemented the required functions on the Nios II processor. Using the PROTEL 99SE tool, we partitioned the design using the UP3 development board and our own circuit modules. We designed all functional system modules based on the FPGA, which represents a system-on-a-programmable-chip (SOPC) design concepts in hardware. Because the Nios II processor is already available, we simply needed to configure peripherals such as flash devices. Other system peripherals include a power management unit, LED interface, and push-button switches.

To make it easy to understand the system hardware design process, we split the design description into the following sections:

- *Schematic diagram design*—Because we completed most of the functional testing and simulation on the UP3 development board and self-designed circuits, the schematic diagram mainly refers to designs that combine these elements into the schematic most representative of the circuit system. With Altera's SOPC solution, we integrated all processors and functional control units into the FPGA, further simplifying the design structure.

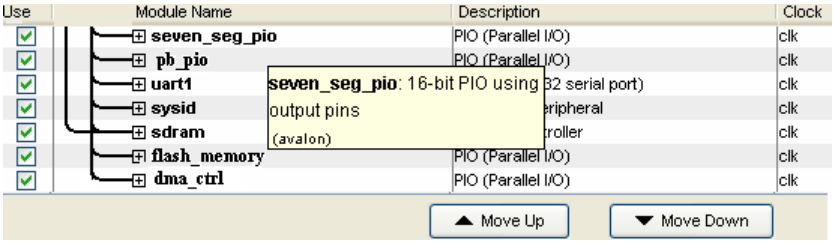
- *Schematic diagram functional verification*—Although most of the functional testing was completed on the test board, we needed to functionally verify the hardware integration. The schematic diagram functional verification primarily demonstrates the proof of concept. This process confirmed our complete circuit design.

- *Component purchase*—We purchased the necessary components, such as the 7-segment displays and push-button switches, while designing the schematic diagram and verifying it. All component packages were clearly marked on the schematic, which made PCB development easier.

- *Implementing the LUTs*— Our design requires two LUTs. One LUT stores the conversion weights that convert the binary data entered using the push-button switches to integers and real numbers for data computation purposes. The other LUT stores the display codes for displaying data in the 7-segment displays.

We added peripherals using SOPC Builder (see Figure 8).

Figure 8. Adding Peripherals Using SOPC Builder



Software Implementation

Our software design task was to migrate the VHDL and C language programs of the previously described functions (on the UP3 development board and self-designed circuits) to the Cyclone EP1C6Q240C8 FPGA with a Nios II soft-core processor. We based the software module design on highly integrated hardware modules so that we could complete core modification and perform upgrades. We used the Altera Quartus II software, SOPC Builder, and Nios II IDE to build the Nios II processor and to develop the system control program, highlighting the SOPC solution's highly integrated and programmable concepts. The design made it possible to implement, add, and remove multiple peripherals easily, access user-defined peripherals, and design user-defined instructions. Generally, the Nios II processor controls the system. However, we implemented most of the software modules based on cooperation between the Nios II processor and the FPGA logic.

We used the VHDL and C languages for the software design. We wrote the logic control and data processing program in VHDL, and used C language routines for the control program of the main and sub Nios II processors. Based on the functional tasks, the system software is divided into system initialization, data acquisition, data display, and patient state prediction. The implementation method and steps are as follows:

- *System initialization*—The system is initialized by the Nios II processor via the tri-state bus, which includes user buttons, 7-segment displays, and the flash memory.

- *Data acquisition*—The patient data is acquired through push-button switches. The system receives a binary 0 input when each switch is pressed. The binary data entered through the push-button switch array is converted into real numbers for computation using conversion weights stored in a LUT.

- *Data display*— The output therapeutic decision is displayed on 7-segment displays. The displays indicate the possibilities of low, moderate,and high values of the different pathological parameters at the patient's next physiological state. Because there are four 7-segment displays for output in the final system and there is only one port available for display, we used a 4-bit output called SCAN (0 to 3). The display codes corresponding to the 7-segment display are stored in ROM.

- *Patient state prediction*—The patient state is predicted by the smart agent implemented on the FPGA and interfaced with the Nios II processor. The smart agent's operation logic is discussed in "Function Description" on page 374.

Design Features

Using Altera's SOPC solution, we learned a new way to solve system design problems. By creating an SOPC design, we learned its advantages and disadvantages. Because SOPC designs use multiple IP modules to optimize the hardware design, we could simplify the system revision and debug process. This approach also allowed us to design software and hardware modules simultaneously. In this design contest, we acquired hands-on experience and were able to use some excellent hardware development tools. The Quartus II software and SOPC Builder made it easy for us to modify and change hardware, depending on the application. We also think that this design approach is economical—you do not need to buy additional hardware, you just change the Nios II configuration. We can easily build new functions by adding related hardware based on the changed Nios II processor.

The main features of our design are:

- *Portability*—Because the requirements of the medical diagnostic system software may change from time to time, we can design the system such that it can port to different hardware configurations. The Nios II processor provides this flexibility.

- *Power consumption*—The system is applicable for rural health care centers where power sources may not be available; therefore, the system should consume very low power. The whole system consumes as little as 60.00 mW.

- *Ease of operation*—The system is designed and developed to be operable easily so that it can be handled by health care professionals. Moreover, it can give an indication about the future pathophysiological state of a patient in the absence of the physician.

- *Integration*—The Nios II processor makes it possible to integrate the FPGA with the peripherals.

Conclusion

During Altera's 2007 Nios II processor competition, our design group divided the design tasks into system integration, hardware development, and software design.

- *System intregration*—The convenience of the Nios II IDE and the SOPC Builder tools gave us the flexibility to implement the design quickly on a prototype machine, which accelerated the development process. In this competition, we learned the process of consumer-electronics product development. The SOPC design approach reduces the cost of manpower and material resources during development. Therefore, we believe that this design approach will become popular in the future. Although we did not add many components, this competition made us appreciate the potential system integration capabilities. Additionally, we hope that Altera can provide a variety of demonstration board programs that will help interested students quickly grasp the FPGA design development process.

- *Hardware development*—In this competition, we used a top-down design approach and planned the complete hardware design in the beginning. Therefore, we needed to establish data stream rules at the start of the planning process. These rules eliminated problems during the design stage, allowing us to complete the project on time. Teamwork was an integral part of this contest. Although the Quartus II tool was easy and flexible to use, there were design issues that required experience; for example, using different frequencies while accessing the SDRAM. This competition gave us an important opportunity to learn about teamwork and problem-solving, understand system development, and resolve challenging design questions.

- *Software design*—We developed the necessary software interface, focusing on implementing the smart agent on the FPGA and interfacing peripherals with the Nios II processor. We used the SOPC Builder C++ tool to create the software design for the Window's interface. We also used it as a verification tool and performed Nios II communication debugging for the phase test. We hope to learn more about SOPC design in the future!

Appendix:
Nios II Embedded Processor Family

Today's embedded design engineers face a tough challenge: finding a processor with the perfect mix of features, cost, performance, while managing processor availability and obsolescence issues. Altera's Nios® II processors deliver the perfect fit every time with fully customizable features and performance, low product and implementation costs, ease of use and adaptability, and obsolescence-proof design.

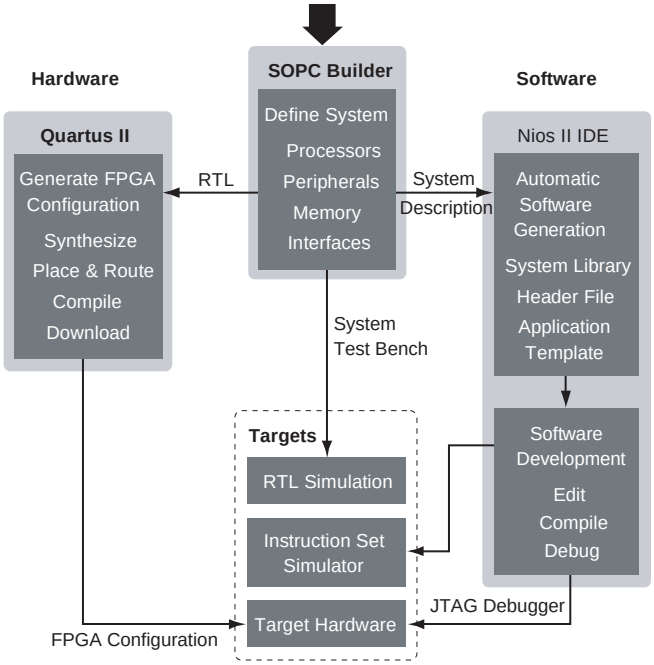
The Nios II family of 32-bit RISC embedded processors delivers up to 300 MIPS of performance and can consume as little as US $0.23 of FPGA logic. Because the processors are soft-core and flexible, you can choose from an unlimited combination of system configurations to meet your performance, feature, and cost targets. Designing with Nios II processors helps you send products to market faster, extend your product's life cycle, and avoid processor obsolescence.

Increase Productivity With Powerful Development Tools

With today's short design cycles, anything that can increase productivity has the potential to pay big dividends, win market share, and establish a competitive advantage. Altera's powerful development tools allow you to rapidly prototype, develop, and deploy embedded systems, reducing time-to-market while extending time-in-market.

Altera's comprehensive hardware and software tools help you create powerful Nios II processor systems in minutes. Figure 1 shows the complete Nios II embedded processor design flow. From concept (at top), through hardware and software implementation, to debug, Altera offers all the tools you need to get your product to market fast.

Figure 1. Nios II Embedded Processor Development Flow

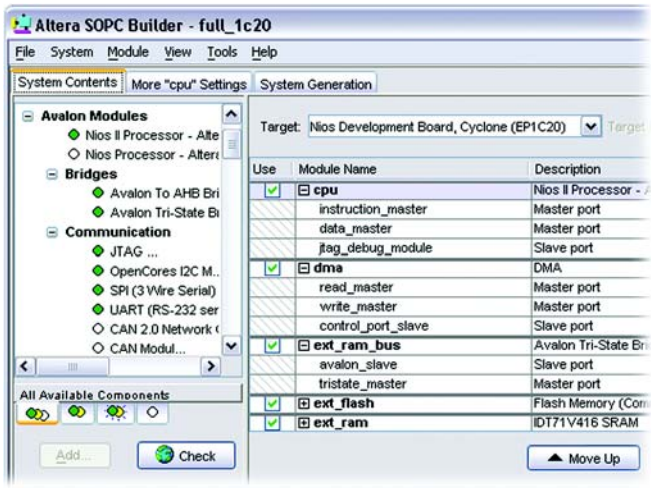


Hardware Development Tools

Altera provides a complete set of tools for your hardware design, including the SOPC Builder system development tool, Quartus® II design software, ModelSim®-Altera software, and SignalTap® II embedded logic analyzer.

Hardware design for creating Nios II processor-based systems uses the SOPC Builder system development tool to specify, configure, and generate systems. Launching from within the Quartus II design software, SOPC Builder provides an intuitive wizard-driven graphical user interface (GUI) so you can create, configure, and generate system-on-a-programmable-chip (SOPC) designs. It minimizes the time spent integrating components into a coherent system. Figure 2 shows a view of the intuitive SOPC Builder GUI.

Figure 2. SOPC Builder GUI



Use the SOPC Builder system design tool to choose from a menu of peripherals, communication interfaces, memory, and I/O to suit your application needs. This tool is tightly integrated with the Nios II Embedded Design Suite and automatically generates a complete custom board support package, including all required software drivers.

Quartus II Design Software

Altera's Quartus II design software technology leadership gives you unmatched levels of performance and ease-of-use. Using Quartus II software, you can easily design, optimize, and verify your Nios II designs in an Altera device.

When you are ready to simulate your design, SOPC Builder generates both VHDL and Verilog HDL simulation models. You can easily simulate Nios II processor-based systems using an automatically generated simulation environment created by SOPC Builder and the Nios IIIntegrated Development Environment (IDE). A full Quartus II software subscription includes ModelSim-Altera software that can also be used to simulate your Nios II designs.

SignalTap II Embedded Logic Analyzer

The ultimate testbench for engineers who want to see the active processes within their design is running at speed under real-world system conditions. The challenge is in accessing nodes buried within the FPGA architecture. The SignalTap II embedded logic analyzer eliminates this challenge by providing access to nearly any node within your FPGA design through a standard Joint Test Action Group (JTAG) port to view design nodes in system and at system speeds.

Software Development Tools

The Nios II Embedded Design Suite (EDS) is a collection of tools, utilities, libraries, and drivers used to develop embedded software for the Nios II processor. The Nios II EDS includes:

- Integrated development environment
- JTAG debugger
- Instruction set simulator
- Flash programmer

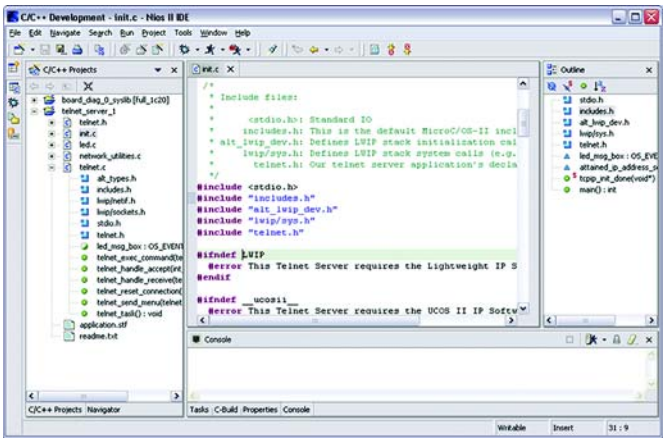
- Design examples (in the form of software templates)
- Software libraries and embedded software components
  - Hardware abstraction layer (HAL)
  - µC/OS-II real-time operating system 1
  - TCP/IP protocol stack: NicheStack TCP/IP Network Stack-Nios II Edition 1
  - Newlib ANSI-C standard library
  - Simple file system
  - Other Altera command-line tools and utilities
- Embedded software acceleration tool: Nios II C-to-Hardware Acceleration (C2H) Compiler [1]

(1) Fully functional evaluation version is included with the Nios II EDS and can be licensed separately.

Nios II Integrated Development Environment

Based on the open, extensible Eclipse IDE project and the Eclipse C/C++ Development Tools project, the Nios II IDE is the primary software development tool for the Nios II family of embedded processors. You can complete all software development tasks within the Nios II IDE, including editing, compiling, downloading, debugging, and flash programming. The Nios II IDE, shown in Figure 3, provides a consistent development platform that works for all Nios II processor systems. With a PC, an Altera device, and a JTAG download cable, you have everything you need to develop and debug Nios II processor-based systems.

Figure 3. Nios II IDE



JTAG Debugger

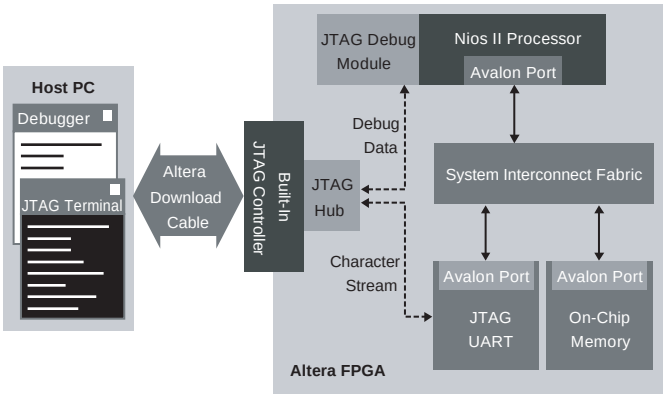
The Nios II architecture supports a JTAG debug module that provides on-chip emulation features to control the processor remotely from a host PC. The Nios II IDE communicates with the JTAG debug module on one or more Nios II processors so you can:

- Download programs to memory
- Start and stop program execution

- Set breakpoints and watchpoints
- Analyze registers and memory
- Collect real-time execution trace data

The debug module connects to the JTAG circuitry built into all Altera devices and connects to the host PC via a download cable (Figure 4). Additionally, debug support for the Nios II processor is available from several industry-standard providers.

Figure 4. Nios II JTAG Debug Module



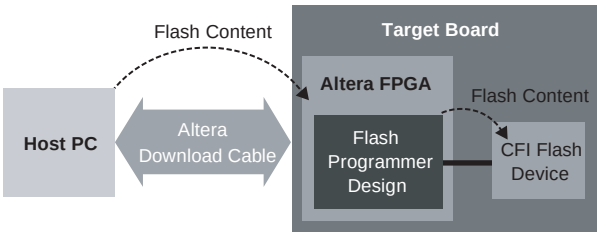
Instruction Set Simulator

The Nios II instruction set simulator (ISS) allows you to begin developing programs before the target hardware platform is ready. The IDE allows you to run programs on the ISS as easily as running them on a real hardware target.

Flash Programmer

Many designs that use Nios II processors also incorporate flash memory on the board. Any common flash memory interface (CFI)-compliant flash device connected to the FPGA can be programmed using the Nios II IDE flash programmer. The Nios II IDE flash programmer can also program any Altera serial configuration device connected to the FPGA, as shown in Figure 5. The Nios II IDE flash programmer is pre-configured to work with all of the boards available with the Nios II development kits, and can be easily ported to any custom hardware.

Figure 5. Transmitting Flash Content to the Flash Device



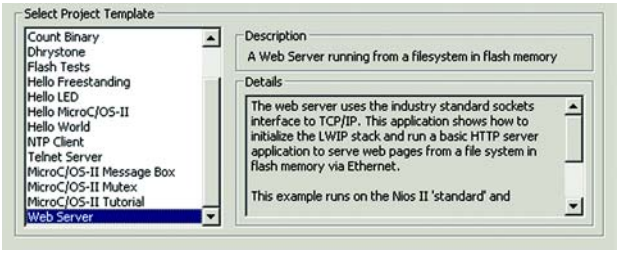
Software Templates

In addition to a project set-up wizard, the Nios II IDE provides software code examples, in the form of project templates, to help you bring up working systems as quickly as possible.

Each template is a collection of software files and project settings. You can add your own source code to the project by placing the code in the project directory or importing the files into the project.

Figure 6 shows some of the available software project templates.

Figure 6. Software Project Templates



System Software

The Nios II IDE lets you customize systems quickly using system software. With system software (also referred to as "software components"), you have an easy way to painlessly configure your system for specific target hardware.

- *Hardware Abstraction Layer*—The hardware abstraction layer (HAL) system library is a lightweight run-time environment that provides a simple device driver interface for programs to communicate with underlying hardware. As SOPC Builder and the Nios II IDE are tightly integrated, the HAL system library can be automatically generated to serve as a board support package for Nios II processor-based designs.

- µ*C/OS-II*—A complete, portable, ROM-able, preemptive real-time kernel, µC/OS-II from Micrium ships with all Nios II development kits and includes full source code, reference manual, and free developers' license. When you are ready to migrate your design to your board, you can purchase a shippers' license. A shippers' license entitles you to a license for three developers to create an unlimited number of designs for one year on µC/OS-II and a perpetual license to support designs created during the subscription period (to fix bugs and make minor modifications).

- *TCP/IP Stack*—NicheStack TCP/IP Network Stack, Nios II Edition is a software suite of networking protocols designed from the ground up to provide an optimal solution for designing network-connected embedded devices with the Nios II processor (refer to Table x). The stack has a small footprint, is portable, and delivers high performance without compromising compliance to request for comment (RFC) standards. NicheStack supports a wide variety of physical interfaces, and can be configured as a standard client machine, an IP router, or a multi-homed server. The suite also contains a comprehensive device networking package, FTP, Telnet, IGMPv1, and DNS and DHCP client components. The NicheStack TCP/IP Network Stack, Nios II Edition is distributed by Altera as full ANSI C source code and includes an evaluation license. If you wish to design with this software suite, you must purchase a license from Altera. The Nios II Edition NicheStack TCP/IP Network Stack has the following highlights:

  - Zero data copy for ultra fast performance
  - Standard sockets interface
  - Raw socket support
  - Non-blocking versions of all functions
  - Versatile MSS and window options

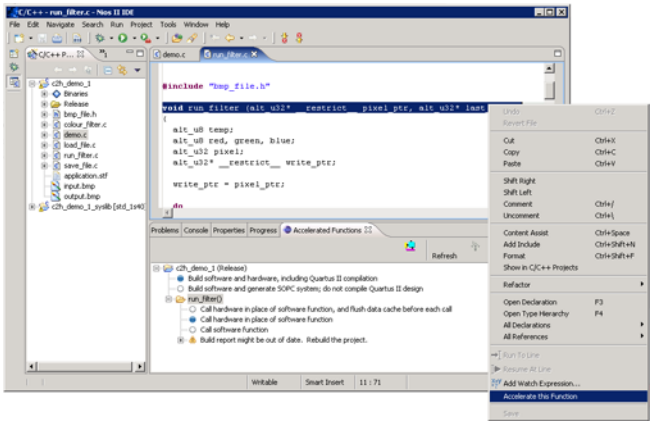
- Connections limited only by memory availability
- Optional optimized assembly language checksum routines
- “Predictive” header processing for speed
- Nagle algorithm (slow start)
- VJ smoothed round trip timing
- Delayed ACKs
- BSD style "keepalive" option
- Complete debugging and optimization module

Nios II C-to-Hardware Acceleration (C2H) Compiler

Altera also offers the Nios II C-to-Hardware Acceleration Compiler (C2H Compiler), a productivity tool that gives embedded developers push-button acceleration of performance-critical C-language software algorithms. These algorithms are automatically converted into hardware accelerators in the FPGA that act as coprocessors with a latency-aware, pipelined connection to the processor's memory map. With this tool, designers have an easy way to boost performance using a known programming language and familiar tools, improving productivity and speeding time-to-market.

The C2H Compiler is tightly integrated into the Nios II development environment (Figure 7), leveraging Altera's proven SOPC Builder tool and the system interconnect fabric to automate the conversion of ANSI C source code to hardware (register transfer language), integrate the resulting hardware accelerator into the system's memory map, and schedule memory transactions with latency-aware pipelining. It enables developers to quickly prototype functions in software running on the processor, then easily convert the software into a hardware-accelerated implementation.

Figure 7. C2H Compiler



Protect Your Software Investment from Processor Obsolescence

Component obsolescence impacts virtually every industry, especially those with products that have long lifecycles such as automotive, industrial, military, aerospace, and medical. The biggest investment in

any embedded system is the application software; changes to the system hardware can threaten years of investment in software development.

All silicon is eventually discontinued, and FPGAs are not immune to that. However, by designing with a soft processor your software investment is protected in many ways. Nios II processor designers have a perpetual license to create and deploy Nios II processor-based designs in Altera FPGAs, so even if the underlying FPGA device changes, the investment in application software is preserved.

- *No need to port application software*—Because the processor is soft, it is implemented as a hardware project that can easily be migrated to another Altera FPGA device while maintaining the same application code.
- *No need to requalify a new processor*—Because the processor hasn't changed, you do not need to spend the time and effort requalifying a new device.
- *Minimal lost opportunity cost*—By keeping the same design tools, design flow, hardware design, and application software, your lost opportunity cost is minimized.
- *Board redesign*—In the worst-case scenario where your current FPGA is not available, you may need to redesign your board. However, design risk is significantly reduced because the entire hardware design is intact and only the device and pin connections change.
- *Phasing in new product*—Phasing in a new product is always a challenge. However, if you design your system with an Altera FPGA, upgrading your system simply involves a flash update that can be performed remotely, as opposed to a hardware replacement. This minimizes customer down time and field service costs.

Scale System Performance

Altera's Nios II processors let you take full advantage of the inherent parallelism of FPGAs to achieve high levels of system performance. Multiple processors can execute code simultaneously while hardware accelerators can offload compute-intensive algorithms at the same time. Upgrade your embedded system's performance at any stage of the product life cycle without the need to redesign the board or develop hand-optimized assembly code.

Three Processor Cores

Choose from three code-compatible, 32-bit, soft-core processors: one optimized for maximum system performance, one optimized for minimum logic usage, and one that strikes a balance between the two. These cores can easily be configured with features such as multipliers, user-specified cache memories, custom instructions, hardware debug logic, and more to adapt to your specific performance needs. Table 1 shows the features of each Nios II processor family member; Tables 2 and 3 provide additional performance details.

Table 1. Nios II Processor Family Members (Part 1 of 2)

Feature	Nios II /f (Fast)	Nios II /s (Standard)	Nios II /e (Economy)
Feature	Nios II/f (Fast)	Nios II/s (Standard)	Nios II/e (Economy)
Description	Optimized for maximum performance	Faster than the fastest and smaller than the smallest first-generation Nios CPU	Optimized for minimum logic usage
Pipeline	6 Stage	5 Stage	1 Stage
Multiplier	1 Cycle *(1)*	3 Cycle *(1)*	Emulated in software
Branch Prediction	Dynamic	Static	None

Table 1. Nios II Processor Family Members (Part 2 of 2)

Feature	Nios II /f (Fast)	Nios II /s (Standard)	Nios II /e (Economy)
Instruction Cache	Configurable	Configurable	None
Data Cache	Configurable	None	None
Custom Instructions	Up to 256	Up to 256	Up to 256

Note to Table 1:
(1) Using DSP Blocks in Stratix® or Stratix II FPGAs.

Table 2. Maximum Clock Frequency (f_{MAX}) for Nios II Processor System (Note 1)

Device Family	Device Used	Nios II /f	Nios II /s	Nios II /e
Stratix III	EP3SL70F484C2	266	200	322
HardCopy® Stratix II	HC230F1020C	202	202	321
Stratix II	EP2S60F1020C3	222	171	285
HardCopy Stratix	EP1S80F1020C5_HC	147	131	176
Stratix	EP1S80F1020C5	148	128	172
Cyclone III	EP3C40F324C6	163	136	190
Cyclone II	EP2C20F484C6	142	111	193
Cyclone	EP1C20F400C6	134	121	173

Note to Table 2:
(1) These results were generated using seed swapping and synthesis/fitting settings in the Quartus II software.

Table 3. DMIPS for Nios II Processor System (Dhrystone Benchmark v2.1) (Note 1)

Device Family	Device Used	Nios II /f	Nios II /s	Nios II /e
Stratix III	EP3SL70F484C2	300	128	50
HardCopy® Stratix II	HC230F1020C	228	129	49
Stratix II	EP2S60F1020C3	251	110	44
HardCopy Stratix	EP1S80F1020C5_HC	166	84	27
Stratix	EP1S80F1020C5	168	82	27
Cyclone III	EP3C40F324C6	165	68	17
Cyclone II	EP2C20F484C6	144	55	18
Cyclone	EP1C20F400C6	130	53	17

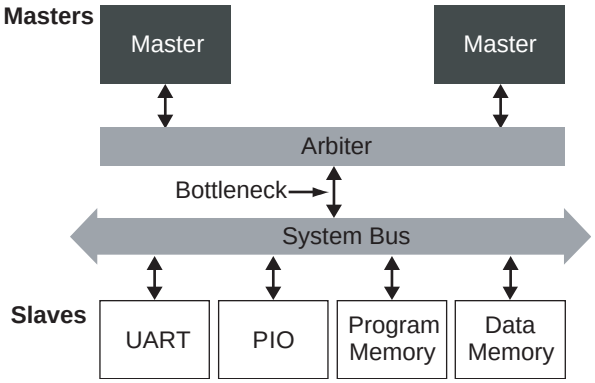
Note to Table 3:
(1) These results were generated using seed swapping and synthesis/fitting settings in the Quartus II software.

High-Bandwidth System Interconnect

Altera's SOPC Builder system design software lets you generate high-throughput systems that take advantage of the inherent parallelism of FPGAs. The system interconnect fabric is fully switched, in that dedicated connections between master and slave components allow multiple simultaneous transactions without the arbitration stalls found in traditional bus architectures.

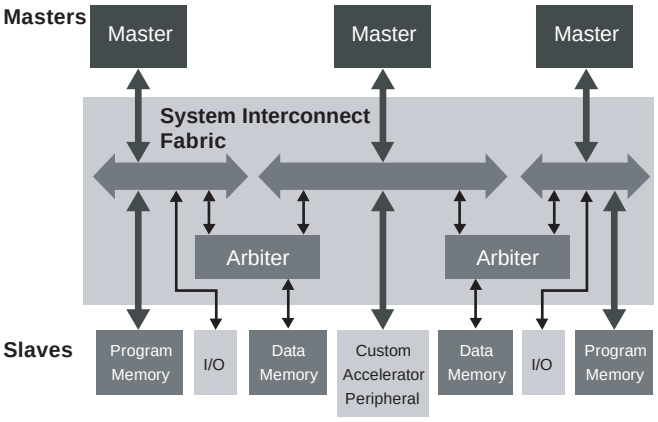
In traditional bus architectures (Figure 8), a single arbiter controls communication between the bus masters and slaves. Each bus master requests control of the bus, and the arbiter then grants bus access to a single master. If multiple masters attempt to access the bus at once, the arbiter allocates bus resources to a master based on a fixed set of arbitration rules. This can lead to a bandwidth bottleneck as only one master can access the system bus and its resources at a time.

Figure 8. Traditional Bus Architecture



The system interconnect fabric's simultaneous multi-master architecture increases your system's bandwidth by eliminating this bottleneck (Figure 9). Using Altera's system interconnect fabric, each bus master gets its own dedicated interconnect, meaning that bus masters only contend for shared slaves, not for the bus itself. Each time a component is added or the peripheral access priorities change, SOPC Builder generates a newly optimized system interconnect fabric with a minimum of FPGA resource use.

Figure 9. System Interconnect Fabric Architecture



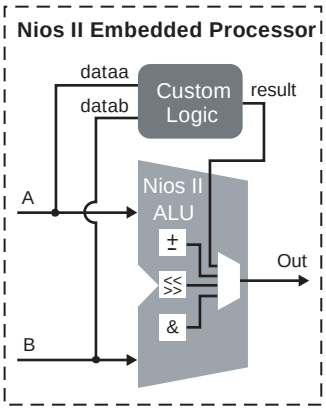
The system interconnect fabric supports a wide range of system architectures, including single- and multiple-master systems, and allows seamless data transfers between peripherals with performance-optimized data paths. Your design's off-chip processors and peripherals are equally well supported by the system interconnect fabric.

Custom Instructions

Custom instructions allow developers using Nios II processors to increase system performance by extending the CPU instruction set to accelerate time-critical software. Using custom instructions, you can optimize system performance in a way not possible with traditional off-the-shelf processors.

The Nios II family of processors supports up to 256 custom instructions to accelerate logic or mathematically complex algorithms normally handled in software. For example, a block of logic that performs a cyclic redundancy code calculation on a 64-Kbyte buffer operates 27 times faster as a custom instruction than when performed by software (Figure 10). Nios II processors support fixed and variable cycle operations, include a wizard for importing user logic as a custom instruction, and automatically create software macros for use in developers' code.

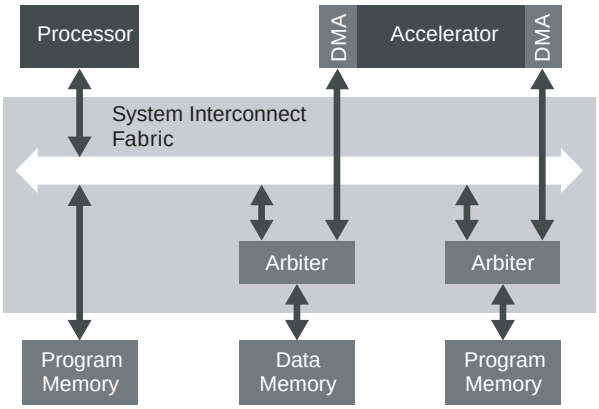
Figure 10. Nios II Custom Instructions



Hardware Accelerators

Automatically accelerate your software by converting C language subroutines into hardware accelerators that boost performance without increasing clock frequency and power consumption. Simply "right-click to accelerate" performance-critical functions using the Nios II C-to-Hardware (C2H) Acceleration Compiler and eliminate the time and expense of manually generating Verilog HDL or VHDL accelerators. See Figure 11.

Figure 11. Nios II Hardware Accelerator



Multiprocessor Systems

Use multiple processors to scale your system's performance or to partition software applications into smaller, simpler tasks that are easier to write, debug, and maintain. The Nios II Embedded Design Suite (EDS) and tools from industry-leading embedded software providers support developing and debugging multiprocessor applications. Nios II processors, combined with high-density devices such as Stratix FPGAs and HardCopy structured ASICs, are ideal platforms for creating high-performance multiprocessor systems.

Configurable Caches and Tightly Coupled Memory

Adjust the size of the processor instruction or data cache to meet the performance needs of your application. For fast access to frequently used routines, add up to four tightly coupled memories that provide cache-like access without the penalty of cache misses.

Customize Your Processor

Rather than being limited to a pre-fab processor, with Nios II processors, you choose the exact peripherals, memory, and interface features you need-customizing the processor to your specifications. In addition, you can easily integrate your own proprietary functions to give your design a unique competitive advantage.

Nios II development kits include a library of commonly used peripherals and interfaces. See Table 4. For a complete list of SOPC Builder-ready intellectual property (IP) and peripherals, visit www.altera.com/SOPCBuilderReady.

Table 4. Nios II Peripherals

Peripheral	Description
JTAG UART	Communicates serial character streams between a host PC and an SOPC Builder system using the JTAG circuitry built into Altera FPGAs
CompactFlash Interface	Provides mass storage support
Interface to User Logic	Connects on-chip user logic or off-chip devices to an SOPC Builder-generated system
UART	Provides common serial interface with variable baud rate, parity, stop and data bits, and optional flow control signals
Interval Timer	Provides a 32-bit timer; can be also be used as periodic pulse generator or system watchdog timer
Parallel I/O (PIO)	Provides 1- to 32-bit parallel I/O (input, output, and edge-capture) ports
Serial Peripheral Interface (SPI)	Implements an industry-standard serial peripheral interface with either master or slave protocol
DMA Controller	Performs bulk data transfers by offloading memory tasks from the CPU
SDRAM Controller	Provides a simple Avalon® interface to off-chip SDRAM and supports 8-, 16-, 32-, and 64-bit data
Memory Interfaces	Includes: ● On-chip ROM and RAM ● SDRAM, SSRAM, SRAM, and flash ● Altera serial configuration devices
Ethernet Port	Includes: ● 10/100 Mbps SMSC LAN91C111single-chip Ethernet controller interface ● Software support provided by the lightweight IP TCP/IP stack ● Included with the Nios II development kits

Using the interface-to-user-logic wizard in the SOPC Builder software, you can also create your own custom peripherals and integrate them into Nios II processor systems. With SOPC Builder and Altera FPGAs, you can assemble embedded processor configurations not available in off-the-shelf processors, letting you create exactly what you need, every time.

Adapt to Changing Requirements

Add hardware features at any time in the product lifecycle without the need to redesign your board. You can boost performance and add features, even late in the development stage, to help you adapt to changing requirements and reduce development time, insulating your product from the competition. To

deploy bug fixes, feature enhancements, and service upgrades for systems in the field, on site or remotely, simply download a new firmware image on the target system.

Reduce Your Total System Cost

Your total investment in an embedded application is more than the cost of the embedded processor. By implementing your embedded system in an FPGA, you can reduce system cost several ways:

- *System Integration Reduces BOM Cost*—Reduce chip count by implementing functions currently handled by discrete components. Integrating functions into a single FPGA reduces board size, cost, density, and power consumption.
- *Royalty-Free Perpetual Use License*—Purchase once, use perpetually, and pay no royalties for Nios II processors implemented in Altera FPGAs and HardCopy structured ASICs.
- *Low Cost Embedded FPGA*—Combine the Nios II /e core with a low-cost Cyclone® series FPGA and consume as little as US $.23 of logic, which leaves plenty of room to add your own custom design or integrate functions performed by external devices. By using a soft-core processor, you can always target the latest, lowest cost device with minimal design risk.
- *High-Volume Migration Path to ASIC*—Once your FPGA design is finalized and moves into high-volume production, you can quickly migrate it to an Altera HardCopy II structured ASIC solution, dramatically reducing device cost. Altera also offers an ASIC license for the Nios II processor, peripherals, and interconnect logic for designs that need to migrate to cell-based ASIC designs.
- *Exact-Fit Peripheral Feature Set*—Avoid paying for unnecessary peripherals, features, or power consumption. Create a custom-tailored system that combines one or more CPUs with the exact set of peripherals, memory, and I/O interfaces you need, and a power consumption you can live with.
- *Custom Coprocessor*—Avoid migrating to an expensive discrete processor for incremental performance enhancements. Instead, offload your computing to a low-cost coprocessor inside your FPGA. Using a combination of one or more Nios II processors, custom instructions, and hardware accelerators, your FPGA co-processor can provide significant performance boost to your existing processor.
- *Low-Cost Development Tools*—You can download full featured evaluation versions of the Nios II processor, Nios II Embedded Design Suite, and SOPC Builder system design tool today for free enabling you to go from concept to complete system running in the lab in a matter of hours.
- *Driver Development Costs*—Choose from a library of available drivers for your embedded needs and drastically reduce the need for custom driver development and accelerate time-to-market.
- *Reduce the Cost of Hardware Upgrades and Maintenance*—The ability to update firmware over Ethernet is a common feature in today's embedded systems. In classic embedded systems (comprised of a discrete microprocessor and the devices it controlled), the word “firmware” applied to the update of the software image that the microprocessor was running. Add FPGAs to the embedded mix and the remote update possibilities increase. This is especially true of systems that contain a Nios II soft processor, because you can upgrade both the Nios II processor (as part of the FPGA image) and the software that it runs in one remote configuration session.

Speed Development with Nios II Development Kits

Altera and its partners offer development kits that give you everything you need to start designing the perfect processor for your system today: from documentation to download cables, from boards to design software. One example kit is shown in Figure 12. To find out more, visit Altera's development kits web site at www.altera.com/devkits.

Figure 12. Nios II Development Kit, Cyclone Edition

Accelerate Your Learning Curve

Get started immediately by downloading the complete set of Nios II design tools and intellectual property (IP) cores or accelerate your design efforts with one of our low-cost development kits. Reuse one of the many pre-built embedded reference designs as a template for your own application and quickly come up to speed on how to use Altera tools through several in-depth tutorials, online training modules, or instructor-led training classes.

There are several ways to learn more about Altera's embedded solutions, all of which begin by navigating to the Altera embedded web site (www.altera.com/embedded) where you can:

- View online demonstrations
- Read in-depth technical documentation
- Download an evaluation version of the Nios II processors and Nios II IDE
- Check out the latest Nios II development kits
- Register to attend on-line or instructor-led training

When you are ready for the next step, simply order a development kit or contact a sales office. Visit www.altera.com today for details.

You can also visit the Nios design forum site (www.niosforum.org), where Nios and Nios II users around the world share ideas, design examples, and other information.

Appendix: Nios II Embedded Processor Design Contest Winners

The following table lists the 2007 contest winners.

Nios II Design Contest Winners

Design	University/College	Award	Location
MRI Spinal Segmentation Based on the Nios II Processor	Information Science Institute, College of Computer and Information Technology, Beijing Jiaotong University	First	China
Portable Telemedicine Monitoring Equipment	HuaQiao University	Second	China
Laser Direct Writing Digital Servo Controller Based on SOPC Technology	Ultra-Precision Photoelectric Instrument Engineering Research Institute, Harbin Institute of Technology	Second	China
Nios II Processor-Based Fingerprint Identification System	College of Communication Engineering, Chongqing University	Third and C2H Award	China
Nios II-Based Intellectual Property Camera Design	Xidian University	Third	China
An Internet-Based Smart Terminal	Shanghai Jiao Tong University	Third	China
Fingerprint Identification System Based on the Nios II Processor	Huazhong University of Science and Technology	Third	China
Multi-Functional Digital Albums Based on the Nios II Processor	Information Science Institute, Beijing Jiaotong University	Third	China
SOPC-Based Cordless Phone	National Institute of Technology, Tiruchy	First	India
Nios II Processor-Based Self-Adaptive QRS Detection System	Indian Institute of Technology, Kharagpur	Second	India
FPGA-Based Clinical Diagnostic System using Pipelined Architectures in the Nios II Soft-Core Processor	Jadavpur University, Calcutta	Third	India
Police Vehicle Support System with Wireless Auto-Tracking Camera	Inha University, Korea Aerospace University, Hongik University	First and C2H Award	Korea
Aerial Photographic System Using an Unmanned Aerial Vehicle	Chungbuk National University	Second	Korea
Auto Audio Equalizer Using Digital Signal Analysis	Hanyang University	Third	Korea

Nios II Design Contest Winners

Design	University/College	Award	Location
RTOS Acceleration Using Instruction Set Customization	Centre for High Performance Embedded System (CHiPES), Nanyang Technological University (NTU)	Star Award	Singapore
Smart Self-Controlled Vehicle for Motion Image Tracking	Department of Information Engineering, I-Shou University	First	Taiwan
H.264 VBS-BMA-Based Hardware Infrastructure Implementation on an FPGA	Ching Yun University/ Department of Electronic Engineering	Second	Taiwan
Smart Bus Station Sign	Oriental Institute of Technology	Third	Taiwan
FPGA-Based Smart Induction Motor Controller Design	Electrical Engineering Department, Yuan Ze University	Third	Taiwan
Intelligent Solar Tracking Control System Implemented on an FPGA	Institute of Electrical Engineering, Yuan Ze University	Third	Taiwan

The following pages show photos of some of the award presentations and design contest winners.

Prize Won: Second Prize, Taiwan
University: Ching Yun University
Project: H.264 VBS-BMA-Based Hardware Infrastructure Implementation on an FPGA

Prize Won: First Prize, Taiwan
University: I-Shou University
Project: Smart Self-Controlled Vehicle for Motion Image Tracking

Bob Xu, University Program Manager of Altera presented the Third Prize Award to Beijing Jiaotong University and Huazhong University of Science and Technology.

The winning teams in China showcased their projects at SOPC World 2007 in Wuhan, China.

Prize Won: First Prize, India
University: Jadavpur University, Calcutta
Project: SOPC-Based Cordless Phone

Eric Law, FAE Director of Altera Asia Pacific, presented the award to the Best Instructor from the National Institute of Technology, Tiruchy

Michael Lee, Altera FAE Manager, presented the prize to the Best Instructor of the First Prize winner and the winning team from Inha/Hongik/ Korea Aerospace University.