Second Prize

Embedded Network MP3 Playing System

Institution: **Southern Taiwan University of Technology**

Participants: **Cai Suwei, Xiao Xingjie, Zhang Jiahao**

Instructor: **Dr. Wei Zhaohuang**

Design Introduction

We designed an embedded Netware MP3 player system that consolidates both software and hardware, and is based on system-on-a-programmable-chip (SOPC) design principles. The product finds use in the following applications.

Public Broadcasting System

Many public audio-broadcasting systems have transmitted audio signals that suffer from weak audio quality and complicated cabling, and are confined to individual or limited area broadcasts. In contrast, our system broadcasts MP3 audio via Ethernet to solve these problems. Our design not only improves the broadcasting quality, but also extends the life of the broadcasting device, reducing installation and maintenance costs for the entire system.

CD Audio Player

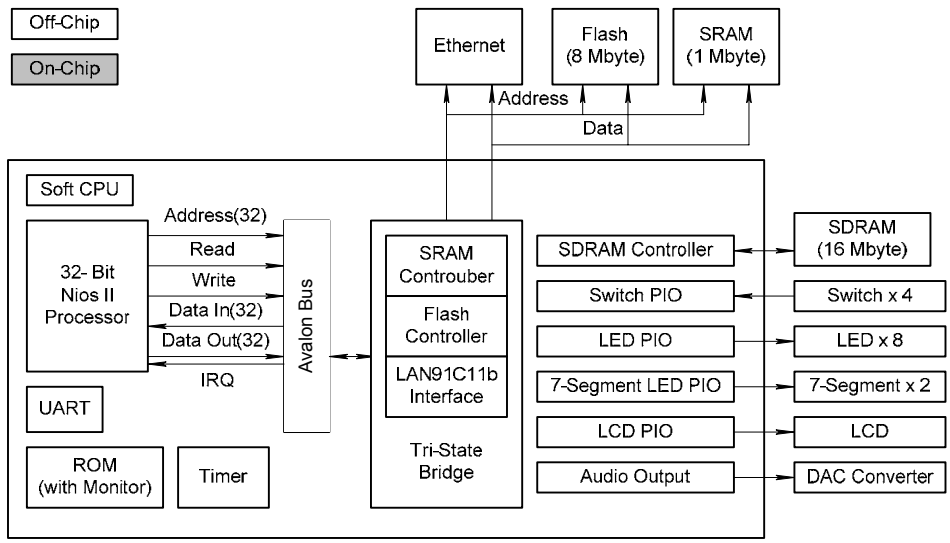
In music stores and supermarkets, customers are provided with preset audio CD players on which they may sample the music. Frequent customer usage can cause the CD players to malfunction. Additionally, the CD audio content must be updated manually. Using our design, you can download MP3 archived audio from a server into the system. This approach needs no mechanical operation and the MP3 server can update the audio data at any time.

Recently, the electronics industry has been shifting away from PC-centric devices to multi-functional Internet appliance (IA) applications, leading to a boom in rapid market development of multimedia and consumer electronics such as MP3, DVD players, TV game consoles, consumer electronics devices, and mobile phones. This trend complicates the system design and shortens the product development cycle. Therefore, using FPGAs has helped product designs become powerful, multi-featured, consistent, low

power, and highly integrated. Programmable logic devices greatly assist designers in planning and customizing the systems they want to build. FPGAs help keep costs down and reduce marketing time, important factors in the electronic industry. Also, programmable logic components play a key role in system integration.

Figure 1 shows the SOPC hardware platform used in our system design. The SOPC hardware features The Altera® 32-bit Nios® II processor, which is the intellectual property (IP) that can be embedded into many FPGA devices. This design approach enables the developer to build systems without worrying about development costs. Using the FPGA, we implemented a serial port, timer, boot ROM, Avalon® bus bridge, and PIO, which connects these interface devices.

Figure 1. Internal Architecture of SOPC Development Platform



The Nios II CPU can be optimized in three ways:

- For maximum system performance.
- For minimum logic use.
- For a mix of system performance and logic use.

Because the program machine codes for all these optimized CPUs are 100% compatible, designers can easily modify the CPU performance according to changes in system requirements. The 32-bit RISC embedded processor has been designed specifically for the FPGA architecture, and features a performance of greater than 200 MIPS (Dhrystones 1.1). Additionally, the development costs are low when you use an Altera FPGA, making it a good choice for consumer electronics applications that demand low price and mass production.

The Nios II processor continues to use the Avalon bus structure introduced by the first-generation Nios processor. This structure provides a set of pre-defined signal types, which can be used to connect more than 60 peripherals, including Ethernet, USB, and memory controllers. SOPC Builder and related tools, such as the Altera Quartus® II software, generate the Avalon bus structure logic automatically. The structure includes data channel reuse, address decoding, waiting period generation, dynamic bus size,

and interrupt assignments. Designers can use the SOPC Builder Import Guide to integrate their own IP modules with other peripherals into the Nios II project.

The Avalon bus structure provides flexible interconnection, simultaneously permitting multiple cores (CPU and accelerator) to communicate on dedicated channels while reading/writing data. This design scheme helps increase the volume of system data transmission. Therefore, the hardware accelerator often used in network communications to compute cyclic delay code can increase its performance two-fold, compared to software processing. For instance, using software, we need millions of clock cycles to process a 64-kilobyte data block. If we used Nios II custom instructions, this could be accomplished with hundreds of thousands of clock cycles. Using the hardware accelerator, it takes only tens of thousands of clock cycles.

Unlike the Nios processor, the Nios II processor does not restrict you to five custom instructions, and allow you to use unequal clock cycles. Furthermore, the Nios II processor supports a maximum of 256 user-defined instructions with fixed or variable frequency period operations. Designers can use these instructions to accelerate the program code and meet an application's strict timing requirements. Additionally, they can implement large and complex algorithms and call them as subroutines in the C language.

Altera provides the Quartus II software, a complete software development tool for the Nios II processor, which includes a Compiler, integrated development environment (IDE), JTAG debugger, and TCP/IP protocol stack:

- *Nios II IDE*—The Nios II IDE is an Eclipse project that opens with the original code and provides a complete C/C+ software development kit, including a compiler, project manager, building tools, debugger, and flash programmer complying with Common Flash Interface (CFI). The IDE supports connection with target hardware via the JTAG port, and supports a connection between the Nios II instruction set simulation and Mentor Graphics' ModelSim hardware simulation tools.

- *IP TCP/IP protocol stack*—A lightweight IP TCP/IP protocol stack provides a Berkeley socket application program interface (API), supporting IP, ICMP, UDP, TCP and RTT estimation, fast recovery, and fast re-transmission.

Function Description

The system comprises an MPEG-1 Layer III (MP3) archive server, MP3 decoder, and Ethernet connection. The design is implemented using software and hardware with an embedded SOPC platform, and uses the embedded uClinux as its real-time operating system (RTOS). The system design principles and detailed description are described in the following sections.

MPEG-1 Layer III Coder Architecture

The MP3 coding principle should be understood before creating the MP3 decoder. Its coding architecture is shown in Figure 2, which includes a psychoacoustics model, hybrid filter bank, and quantization/Huffman coding. Quantization and distortion-free code are mainly used for the rate and distortion control loop in the MP3 architecture.

Taking monophonic data as an example, one MP3 frame contains 1,152 sound samples (a frame equals two granules, a granule contains 576 sound samples); each sample is 16-bits of data. Using the filter bank analysis, the originally entered 16-bit PCM audio is transformed into 32 sub-band signals with the same bandwidth. Then, each sub-band signal is subdivided into 18 hypo-band signals using the modified discrete cosine transform (MDCT). Next, bit assignment and quantization coding are made for each sub-band signal according to the signal-to-mask ratio (SMR) provided by the psychoacoustic model II. At last, this coded data emerges in bit serial mode defined by MPEG-1.

MP3 Decoder Architecture

Figure 3 shows the MP3 decoding architecture. The MP3 bitstream uses a demultiplexer to perform the header and side information decoding. It then implements Huffman Decoding, a descaler, and a dequantizer using the header and side information. Next, it performs the inverse modified discrete cosine transform (IMDCT) using dynamic windows and outputs PCM data through the filter group.

Figure 2. MP3 Decoding Data Stream

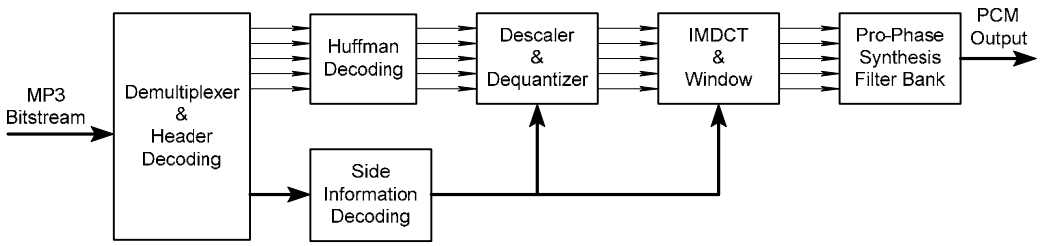
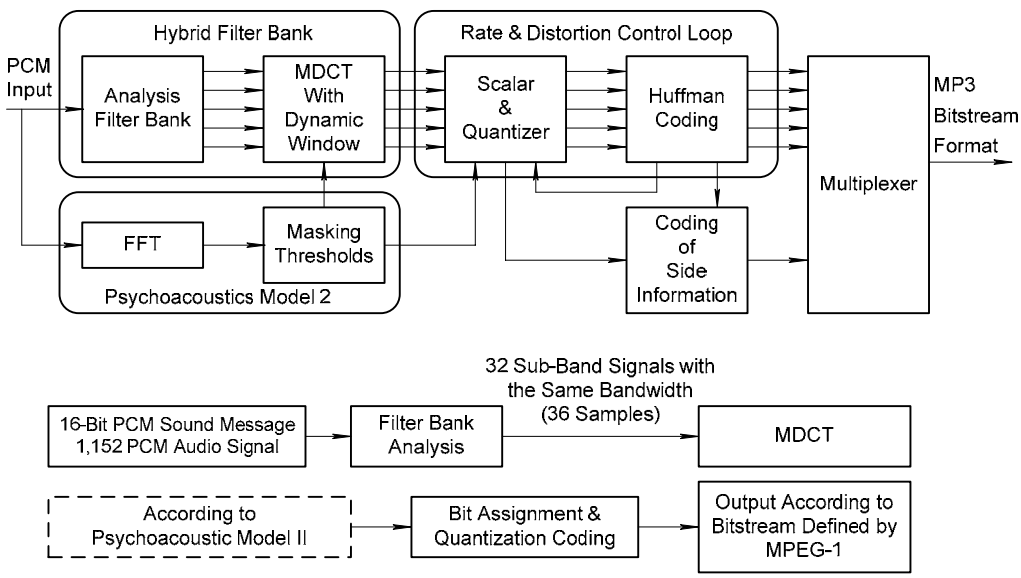


Figure 3. MPEG-1 Layer III Coding Architecture



MP3 Archive Description

Related information about archive decoding should be obtained before implementing the MP3 decoding. This data will be used in the corresponding encoding. You need to know the data's bitstream format, frame format, header file format, and side information format.

Bitstream Format

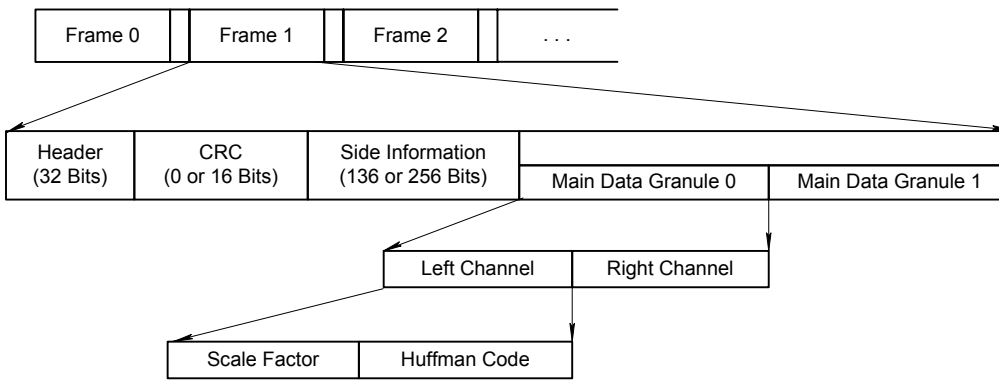
The MP3 bitstream is composed of many frames, with the slot size as its basic unit. The MP3 standard defines a slot size as 1-byte of data, and therefore, the whole bitstream is made up of slots of integer numbers. Every frame contains a great deal of decoding information. The starting point of each frame is a set of sync words, and the ending point is before the sync word of the next frame. All frame sizes are 1,152 samples, although their frame length may vary. The length varies because Huffman decoding is

used during MP3 decoding, which results in a non-identical decoding length and a variable frame length.

Frame Format

The MP3 frame is composed of four parts: the header, the cyclic redundancy code (CRC), side information, and the main data (see Figure 4). The header contains sync words and other important system information used to detect the new frame's starting point, such as layer, bit rate, sampling frequency, and number of channels. The CRC is used in error correction and is 16-bits of data. The side information includes information needed for main data decoding, and the main data section contains data that is needed during Huffman decoding and scale factor rebuilding.

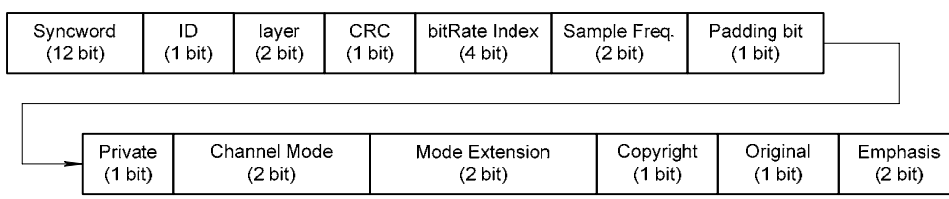
Figure 4. MP3 Frame Format



Header Format

The first 32 bits of a frame hold the header file information, which is divided into 13 fields, and which records important frame information (see Figure 5).

Figure 5. MP3 Header Format



A sync word identifies the start of the frame and detects the frame length, i.e., the distance between the sync word and the next group of sync words (also referred to as a frame). Sync words comprise 12-word groups of 1s. However, ID information is used to tell the decoding end which algorithm needs to be adopted to decode the current bitstream. The decoding end should use this algorithm to decode while the ID field is indicated using a single bit. When the ID is '1', it indicates the adopted algorithm to be the MPEG-1 audio standard (ISO/IEC11172-3); when the ID is '0', it indicates the algorithm to be the MPEG-2 audio standard (ISO/IEC 13818-3). The layer field is shown in two-bit format, and indicates which layer of the audio standard is used. The protection bit indicates whether or not there is an added 16-bit CRC in the bitstream.

The output bit rate index indicates the bit rate of the bitstream with 4 bits. Layer 3 ranges from 32 kbps to 320 kbps. The MPEG-1 audio defines three kinds of sampling rates: 48, 44.1, and 32 kHz. Two bits

are used to indicate the sampling frequency in the header files, while another two bits are used to support four channel modes and indicate the channel mode of the bitstream in the header files. When the mode is set at "01" it is in the joint stereo channel mode. Therefore, the joint stereo processing step must be added to the decoding. Additionally, the two flags—copyright and original—indicate whether the MP3 archive owns copyright or is original, respectively.

Side Information Format

Side information records the information needed in audio data decoding. It is 17 bytes long in a single channel frame and 32 bytes in dual channel or stereo channel (see Figure 6). The starting index of the main data in the side information indicates the starting address of the main data. The Part2_3_length field indicates the length of the scale-factor data. The scalefac_compress[gr] table shows how many bits need to be read each time. Because transform coding is used in MP3, it must transform according to the window size. In addition to the long window used for a stable signal and the short window used for an unstable signal, there are two relay windows used for when it changes from a long window to a short window or vice versa. We use window_switch_flag to indicate which tables are being used. The advantage of using four different windows is that frequency resolution can be added to maintain good audio quality. The block_type field indicates which kind of window each granule uses. There are 32 Huffman tables in all, so table_select in the side information indicates which table is used to perform the Huffman decode.

Figure 6. MP3 Side information format

Single-Channel Mode 17 Bytes

Main_data_begin (9 Bits)	Private_bit (5 Bits)	Scfsi[ch][scfsi_band] (4 Bits)	Gr0 Side Information (59 Bits)	Gr1 Side Information (59 Bits)

Dual-Channel Mode 32 Bytes

Main_data_begin (9 Bits)	Private_bit (5 Bits)	Scfsi[ch][scfsi_band] (4 x 2 + 8 Bits)	Gr0 Side Information (59 x 2 = 118 Bits)	Gr1 Side Information (59 x 2 = 118 Bits)

MP3 Decode Operation

The MP3 decoding operation includes the Huffman decoder, dequantizer, reordering, IMDCT, and synthesis filter bank, which are described in the following sections.

Huffman Decoder & Dequantizer

To obtain high compression rates during MP3 decoding, the spectrum coefficient transformed via the MDCT is divided into 3 regions—the big_value region, the count1 region, and the rzero region from low frequency to high frequency values. Data needs to be recovered from each region according to different Huffman code tables during decompression.

Before entering data into the synthesis filter bank, the value after the Huffman decoding should be dequantized.

Long window:

$$xr_i = sign(is_i) * \left|is_i\right|^{\frac{4}{3}} * 2^{\frac{1}{4}(global\_gain[gr]-210)} * 2^{-(scalefac\_multiplier*(scalefac\_l[gr][ch][sfb][window]+preflag[gr]*pretab[sfb]))}$$

Short window:

$$xr_i = sign(is_i) * |is_i|^{\frac{4}{3}} * 2^{\frac{1}{4}(global\_gain[gr]-210-8*subblock\_gain[window][gr])} * 2^{-(scalefac\_multiplier*scalefac\_s[gr][ch][sfb][window])}$$

Where is_i indicates audio cable undequantized, and xr_i represents that of dequantized; sign() takes the positive signal; global_gain and preflag are obtained from side information.

Reordering

The MDCT adopts two block modes: the short window and the long window, which have different spectrum types, after the MDCT transform. For greater efficiency in Huffman decoding, the short-window spectrum is deployed prior to the Huffman decoding. This reordering step recovers the original sorting sequence.

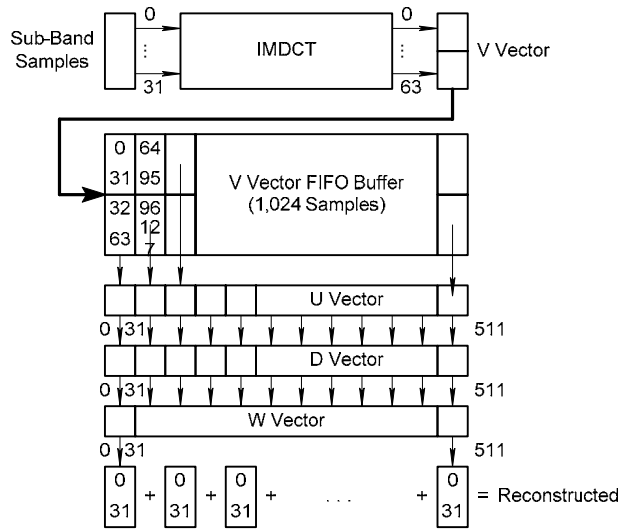
IMDCT

The IMDCT operation is performed on the data frame as the standard unit. There are 576 points in the data of a frame, which are further divided into 32 sub-bands. Each sub-band has 18 points whose data are IMDCT-transformed into 36 points. Each frame thus transforms into 1,152 points, and this value is multiplied with appropriate window functions depending on the type of window. IMDCT definition is (2-1) mode where *N=36* in the long window and *N=12* in the short window.

$$X_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos(\frac{n}{2\pi}(2i+1+\frac{n}{2})(2k+1)) \quad for \;\; i=0 \;\; to \;\; n-1$$

After the IMDCT transform, the synthesis filter bank data enters into the polyphase filter to synthesize the audio signal in PCM format. The synthesis actions in the polyphase filter are remove, IMDCT, and matrix-multiply. IMDCT transforms 32 samples to get a 64 V vector and takes 512 samples to the synthesis filter (W window) to form the W vector (512 elements). The 512 elements are placed into 16 groups of 32 elements, whose summation of the vectors is the final rebuilt audio signal in PCM format (see Figure 7).

Figure 7. Synthesis Filter Bank Flow



Preamble	SFD	DA	SA	LN	Data	PAD	FCS
7	7	2/6	2/6	2	0~1500	0~46	4

- Preamble: Synchronization function
- SFD: 10101011, starting byte
- DA, SA: Destination and starting address
- LN: Data length
- Data: At most 1500 bytes
- PAD: Ensure one frame has at least 64 bytes
- FCS: Error checking code

Ethernet Protocol

Ethernet uses a carrier sense multiple access/collision detection (CSMA/CD) for data transfer. It must ensure that no online signal transfer is made before the data transfer. The data transfer needs to be halted in case of conflict, and retried after some random delay.

Data on Ethernet is transferred in frames. The format is shown in the following section. Each frame has 64 bytes of data as the minimum frame length, and 1,518 bytes as the maximum frame length. PAD is used to fill the packets to 64 bytes in case no data exists or the data is less than 64 bytes.

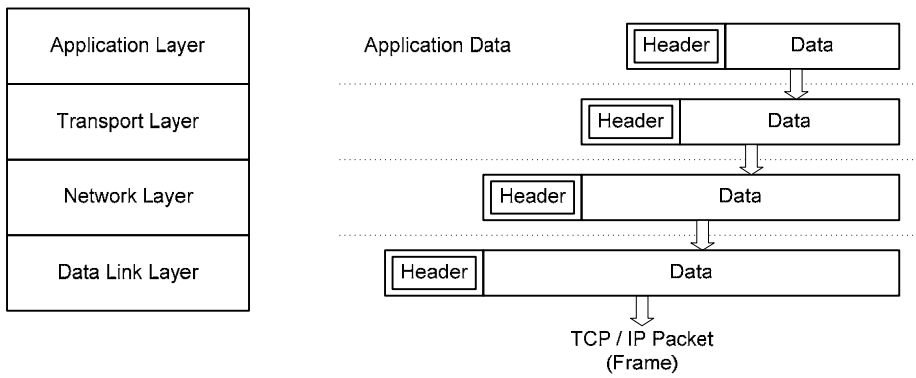
The Ethernet addressing mode is set by the DA/SA in the packet, whose value could be two bytes (for local supervision) or six bytes (for global addressing). The first byte determines transfer to the individual address or the group address. If all values are ‘1’, it is a multicast or a broadcast address.

Using these addressing modes, the MP3 broadcaster can transfer audio to specified receivers or multicast the data.

Data in the packet includes TCP/IP information. TCP/IP is the best protocol available today for use in Internet or Intranet applications. The protocol is briefly described as follows.

The network protocol is built layer by layer. Each layer is responsible for a certain network function. The TCP/IP four-layer network communications architecture includes the network application layer, transport layer, network layer, and data link layer. Programs implemented on the application layer are HTTP, telnet, e-mail FTP, etc; TCP and UDP are implemented on the transport layer; IP and ICMP are carried out on the network layer; the Ethernet driver and PPP protocols are implemented in the data link layer. Figure 8 shows the network protocol.

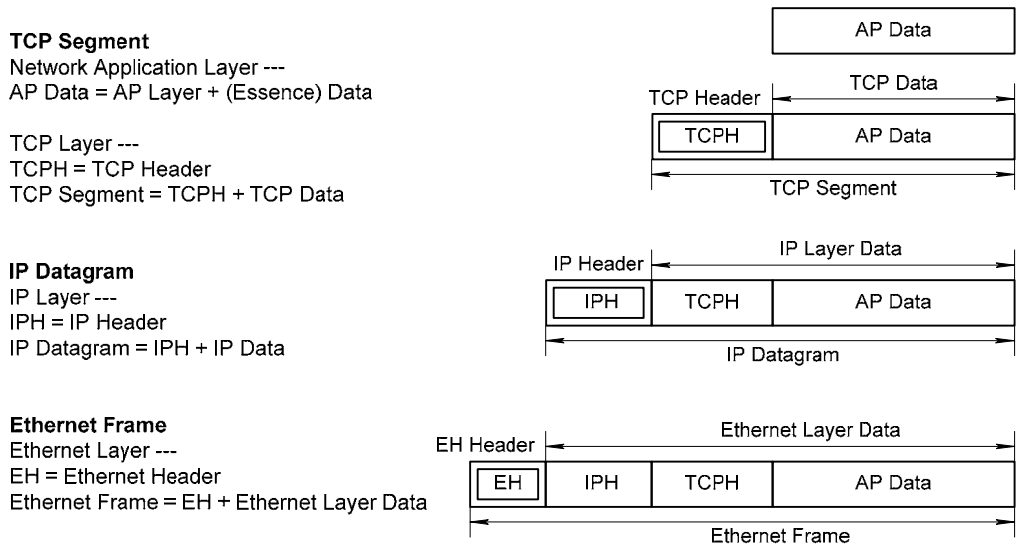
Figure 8. Network Protocol



Single channel mode transfers the network data with TCP/IP, whose operation for “additional Header” in each layer is shown as follows.

The data architecture of each layer of the TCP/IP network communication is shown below. AP Data is added in the header of each layer from the TCP segment to the Ethernet frame, and forms network packet data. Figure 9 shows the data architecture.

Figure 9. Data Architecture



Embedded Operating System Analysis & Selection

An efficient RTOS needs to be supported by the system to enable the embedded products' multifunction features and shorter development time. In the past, embedded systems have implemented fewer functions and therefore did not need operating systems. However, as more and more consumer electronics devices integrate multiple functions, many embedded systems are more complex.

Among the many available RTOS for embedded systems, the µC/OS and uClinux are the most popular, offering excellent performance and open source code. µC/OS is suitable for small control systems because of its high efficiency, small size, and strong scalability. For example, the smallest core of the µC/OS can be compiled into a 2-Kbyte space. The uClinux, adapted from the standard Linux OS, is designed according to the feature of the embedded processor. With built-in network protocols, uClinux supports many file systems.

Comparison between the µC/OS & uClinux Embedded Operating Systems

The embedded operating system is the control center of the embedded system software and hardware resources. It organizes and manages multiple user needs. Task scheduling, file system support and system migration are general tasks in embedded operating system applications. This section compares how the µC/OS and uClinux RTOSs handle these tasks.

Task Scheduling

Task scheduling arranges the use of system resources (memory, I/O devices, and CPU). Task scheduling—also referred to as CPU scheduling—allocates the CPU for tasks in ready status based on two basic modes: preemptive and non-preemptive scheduling. In non-preemptive scheduling, a task is implemented once it is scheduled, unless it gives up the CPU time and enters into wait status, in which case the CPU is reallocated to other tasks. Preemptive scheduling identifies the current task, and is preempted to ready status once a task with higher priority exists in ready status or the running program has used up the specified time slice, in which case the CPU will be allocated to other tasks.

Being an RTOS, μC/OS adopts preemptive real-time multi-task core, i.e., the core always runs the task with the highest priority in ready-status. μC/OS supports a maximum of 64 tasks, which correspond to a priority status of 0~63, 0 being the highest priority. Scheduled tasks can be divided into two parts: search and switch, of the task with the highest priority.

Search of the task with the highest priority is performed by setting up the task in ready status. All tasks in μC/OS have an independent stack. They also have a data structure called tcb (task control block) with a stack index. The task scheduling module first records the tcb address of the task in ready status with the highest priority, currently with the OStcbHighRdy variant, then invokes the os_task_sw() function to realize the task switch.

The uClinux task scheduling adopts the traditional Linux mode. The system starts the task in certain intervals, generates fast and periodic clock-timing interrupt, and decides when the program could possess its time slice by using the scheduling function (timer processing function). It then makes the related task switch by invoking the fork function with the parent task.

After the invoking the fork function in the uClinux system, the subtask substitutes the parent task to realize the implementation. This time, an executable file is generated, even if this task is a copy of the parent task. After the subtask has exited or executed, it awakes the parent task using a wakeup to continue the parent task implementation.

Because uClinux does not have a memory management unit (MMU), its access to the memory is direct, and the accessed addresses in all programs are real physical addresses. The operating system does not protect the memory space, so all tasks actually share the same running space. In this case, data protection needs to be made during the multi-task implementation, which may also result in the user's program taking up the system core space. All these problems should be addressed during the design stage.

From the above analysis, we deduced that μC/OS is best suited our real-time system requirements.

File System

The file system handles the access and management of file information. These operations include file creation, reading/writing, modifying, copying, and software programs that manage resources (directory table, storage media, etc.).

μC/OS is used for medium and small-sized embedded systems. Because the core of μC/OS is only 6 to 10 Kbytes after compilation, the system itself does not support the file system.

uClinux inherits the perfect Linux file system performance. It adopts the romfs file system, which requires less space than the general ext2 file system. The space savings is made up of two aspects. The core needs less code when supporting the romfs file system than supporting the ext2 file system. On the other hand, romfs file system is easier to implement, and requires less storage space when establishing the superblock file system. The romfs file system does not support dynamic erase saving. It adopts a virtual RAM mode to process data that needs to be dynamically saved by the system (RAM would use ext2 file system).

uClinux also inherits the advantages of the Linux network operating system, which supports the network file system and embeds TCP/IP protocol that simplifies development of the network embedded devices of uClinux RTOS.

Based on the file-system support, we decided that uClinux is better suited for complex embedded systems that need many file processes. In contrast, μC/OS is suitable for smaller control system applications.

Migration of the Operating System

The basic idea of migrating an embedded operating system is to run the OS on a certain microprocessor or microcontroller. µC/OS and uClinux are operating systems with open original code whose structural designs make it simple to separate processor-related segments. Therefore, they can migrate to the new processor easily.

The target processor has to meet the following requirements to migrate µC/OS:

- The C program compiler of the processor must generate re-settable machine code and open and close the interrupt routine using the C language.
- The processor supports interrupt and can generate interrupts periodically.
- The processor supports abundant RAM (several Kbytes) as the task stack under a multi-tasking environment.
- The processor features instructions to read or store the stack index and other CPU registers into the stack or memory.

When compared with µC/OS, migrating the uClinux RTOS is more complex. Generally speaking, the target processor should also support external ROM and RAM with abundant capacity in addition to the above-mentioned µC/OS requirements.

The migration of uClinux can be split into three hierarchies:

- *Migration of hierarchy*—If the structure of the processor to be migrated is not the same as any supported processor structure, then the relative processor structure archive in the linux/arch directory must be modified. Although most of the machine code of the uClinux core is independent of the processor and its architecture, the machine code in the lowest hierarchy is not the same in different systems because of their unique interrupt process program, memory map, task process invoking, and initialization process. These routines are in the linux/arch/directory. Because Linux supports various architectures, to handle a new architecture, its low-level program should be compiled by copying the architecture similarly.
- *Platform level migratio*n—If the processor to be migrated is the branch processor of an architecture that has been supported by uClinux, the corresponding directory must be established under a relative architecture directory and the corresponding machine code must be compiled. For example, the migration in this case must establish the linux/arch/nioscpu/platform/nios directory and compile the tracking program (to realize the user program to core function interface and some other functions), the interrupt control program, and the vector initialization program in the directory.
- *Board level migration*—If the processor is supported by uClinux, then board level migration will suffice. The board level migration needs a corresponding board directory established in the linux/arch/platform/ directory, and it should contain the corresponding starting machine code crt0_rom.s or crt0_ram.s and link description file rom.ld or ram.ld. The board level migration also includes driver compilation and environment variables setup.

Comparing µC/OS and uClinux, we can see that the two operating systems have their strengths and weaknesses. µC/OS takes up less space, implements with high efficiency, offers real-time performance, and migrates to a new processor relatively easily. uClinux takes up more space, implements with weak real-time performance, and migrates to the new processor in a relatively complex way. However, with its embedded TCP/IP protocol, uClinux supports various file systems, and benefits from abundant open Linux program resources. uClinux is stronger when used in some more complex applications.

Performance Parameters

The system features the Nios II /s CPU, peripheral compact flash (CF) card, Ethernet network chip, MP3 decoding chip, ROM, SRAM, SDRAM, and other peripherals. The hardware design is shown in Figure 10, and the hardware resource utilization in Table 1. Figures 11 and 12 show the finished MP3 player development platform.

Figure 10. Nios II CPU & Peripheral Component Design

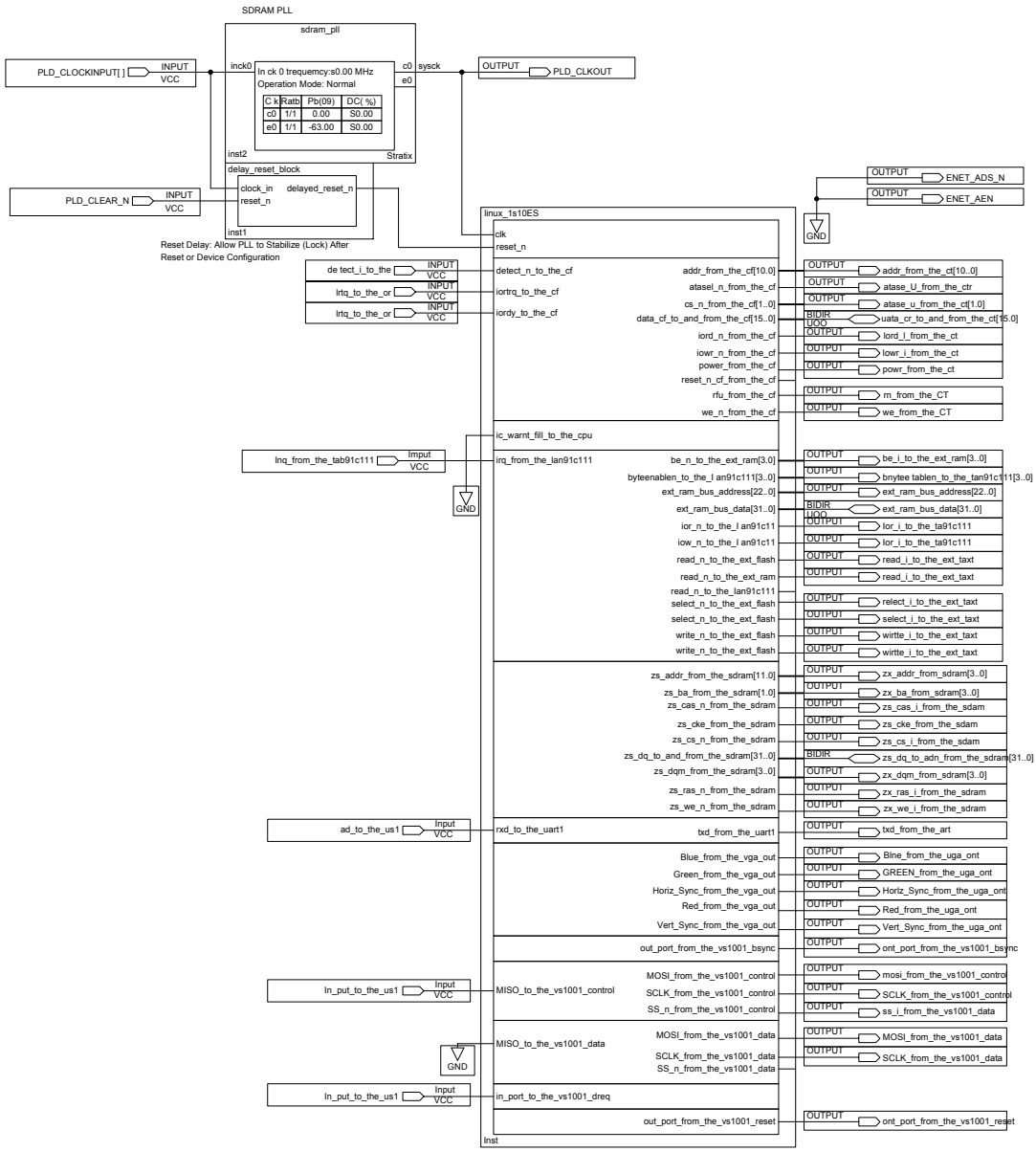


Table 1. MP3 Chip Used by Network Player Hardware & Performance

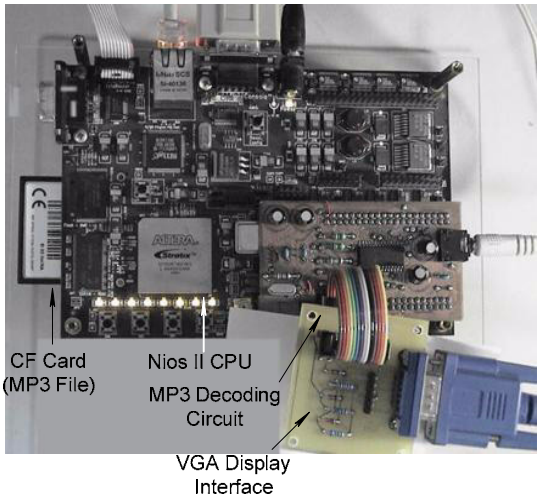
Item	Description			
Function	Nios II MP3 Player System			
Hardware Device	Altera EP1SF780C6ES FPGA			
	Available	Usage		
LEs	10,570	6,575 *		
9-Bit DSP Blocks	48	9		
On Chip Memory (Bits)	920,448	801,536		
		CPU	On Chip RAM	VGA
		45,824	524,288	230,400
I/O Pins	427	191		
Performance	61.63 MHz (fmax)			

*Using the Quartus II software version 5.0 group compiling and integration; Set options together: the least logic

As for software resources, the uClinux kernel and file system take up 1,784 Kbytes and 2,417 Kbytes of external flash ROM, respectively. The application takes up about 30-Kbytes space, and the MP3 files are stored on the CF card.

Figure 11. Outside View of MP3 Player LCD

Figure 12. MP3 Player Development Platform



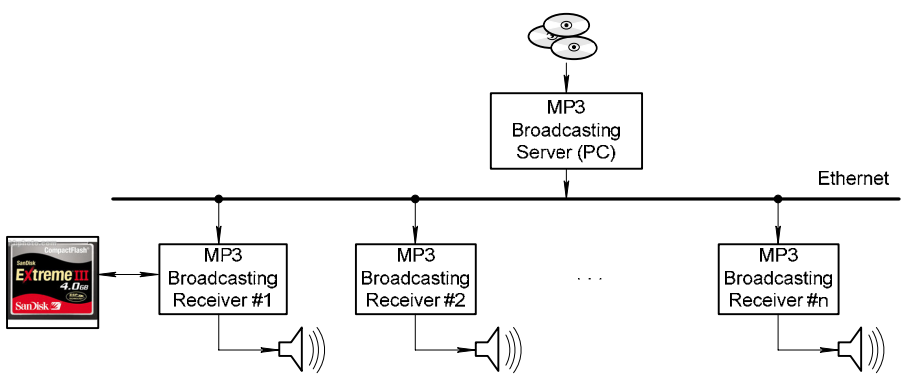
Design Architecture

This section describes the project's design architecture.

MP3 Broadcasting Network

This system architecture will be used for music audio/broadcasting or music audition (see Figure 13). MP3 audio is provided by the MP3 broadcasting server and is downloaded to each MP3 receiver from the Ethernet. The broadcasting server selects the audio actively, and the receiver plays or audits the downloaded music. In this system, the broadcasting server is based on a PC because of the large hard disk, which is easy to store, extend, and maintain. Additionally, the hard disk offers abundant operating system support. If the system is used for business purposes, the Linux/uClinux is preferable because of the server operating system and receiver operating system, considering system stability and costs.

Figure 13. Ethernet MP3 Broadcasting System

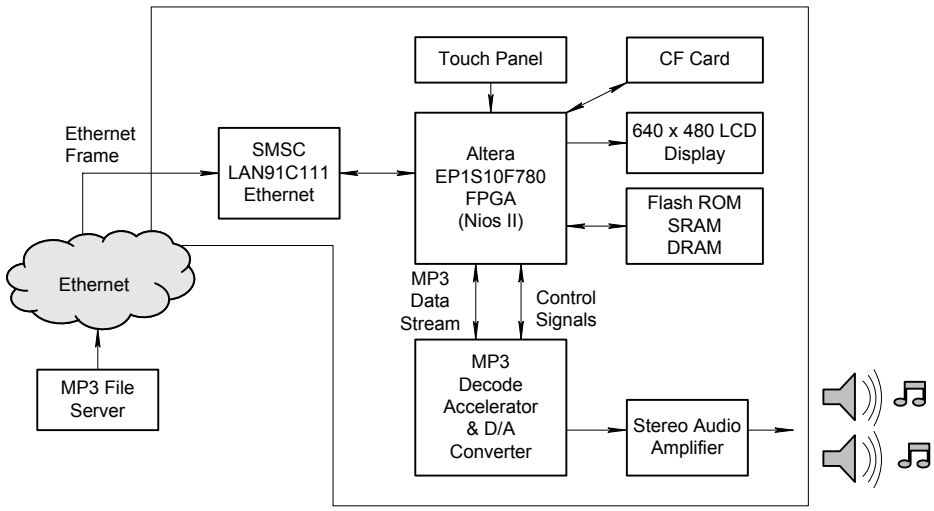


MP3 Broadcasting Receiver

Figure 14 shows the circuit function block of the MP3 broadcasting receiver. After receiving the Ethernet data, the MP3 data is first prepared for processing by the Nios II CPU, which also performs

MP3 decompression. Then, the audio is output via a dual-channel amplifier. The received MP3 data can also be stored in flash memory for playback.

Figure 14. Ethernet MP3 Receiver



MP3 Decoder

From the previous MP3 coding theory, we found that you needed a huge amount of data packet composing or decomposing, and a great deal of repeated numerical operations in both the coding and decoding processes. Additionally, in a real-time system, a large volume of data needs to be processed in every clock cycle. If the processor speed is low in an embedded system, then you cannot play MP3 audio. Our system has been designed to accelerate MP3 decoding with the help of a peripheral decoding circuit. Although the Nios II processor in an FPGA can perform the MP3 software decoding function, it cannot reach the real-time processing speed. Therefore, the MP3 data stream decoding is implemented using an additional decoding chip to reduce the burden on the CPU.

The system adopts a special chip to implement MP3 decoding. Other design considerations include:

- MP3 audio play needs to be transformed into simulation signals towards the end of the process. However, the FPGA currently cannot implement this simulation circuit; in contrast, the dedicated chip features a simulation circuit. Taking this design approach, you can save on components of the simulation circuit.

- Because MP3 audio players are very popular, you can easily get cheap, reliable, and low-power integrated circuits.

Figure 15 shows the block diagram of internal functions of the VS1001, an MP3 decoding chip developed by VLSI Solution Oy from Finland. It features a low-power DSP chip, which could be used to implement user programs to process special audio effects. It also operates as the MP3 decoder and has a 16-digit dual channel digital-to-analog converter (DAC) with no phase difference and a simulation earphone amplifying circuit. These features could significantly simplify the MP3 decoder production.

Figure 15. Block Diagram of Internal Functions in VS1001 MP3 Decoding Chip

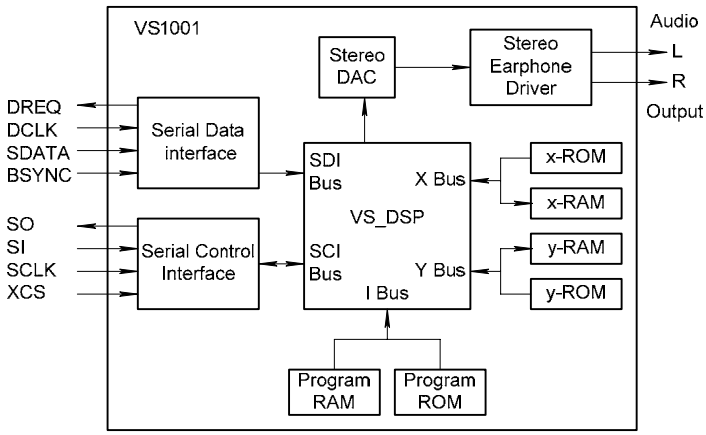
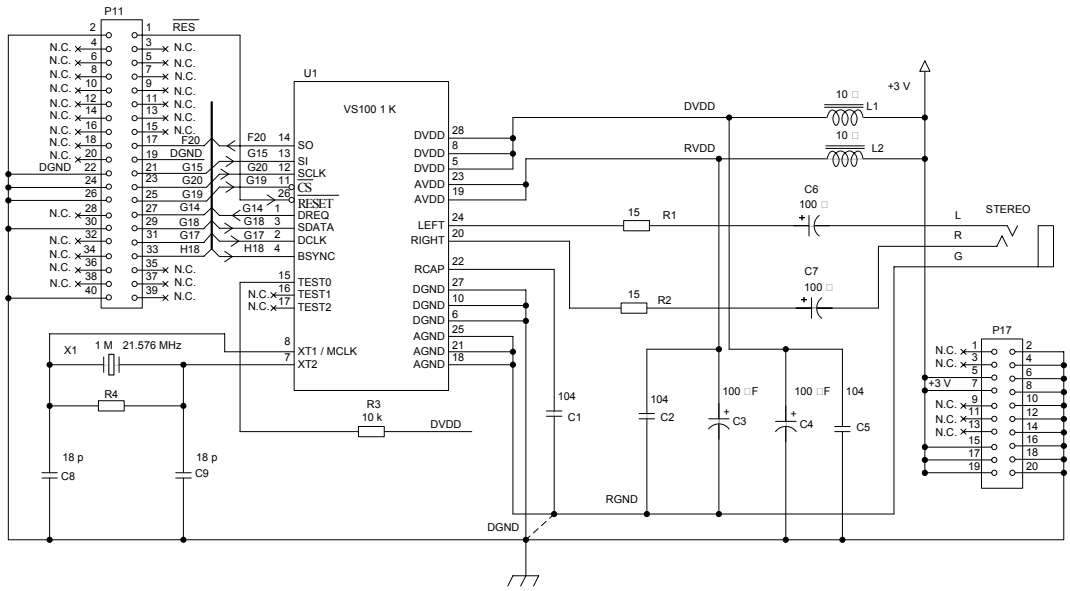


Figure 16 shows the circuit connections between this chip and the Nios II development platform. The 3.3-V working power is obtained from P17 (on the board), the controlled serial peripheral interface (SPI) signal and MP3 data's SPI signal are connected with to the development board at P11. The two SPI signals are controlled by the independent SPI interface in the FPGA (see Figure 16).

The VS1001 chip uses two SPI buses, the serial data interface (SDI) that transmits MP3 compressed data and serial control interface (SCI) that implements control instructions. By reading/writing the 16-bit register of the SCI interface, the following operations can be implemented:

- Operation mode control
- Loading user program
- Reading header data
- Reading status information
- Accessing decompressed digital data
- Feed-in entry data

Figure 16. MP3 Decoding Chip Circuit



The SCI instruction read/write waveforms are shown in Figures 17 and 18, respectively. The data is read or written to at the SCK rising edge.

Figure 17. Reading of SCI Word

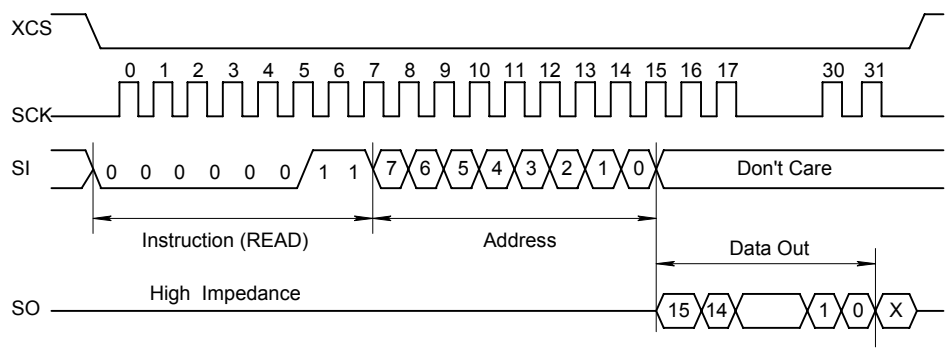


Figure 18. Writing of SCI Word

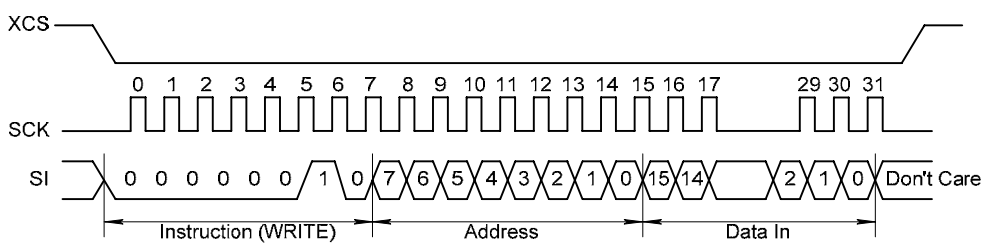


Figure 19 shows the MP3 decoding flow in the VS1001 device and its program steps are as follows:

1. MP3 data is entered through the SDI bus.The data is delivered to the bass/tenor enhancing circuit, which is controlled by the SM_BASS of the SCI register, after MP3 decoding.

2. If A1ADDR of the SCI register is not zero, the application code set by the user is implemented. The starting address is specified by A1ADDR.

3. The digital PCM audio data is then sent to the volume control unit. The output signal is temporarily stored in a FIFO buffer, and is then transformed into simulation audio by the DAC for output according to the sampling frequency. The FIFO buffer can save 512 stereo audio (2 x 16 bits).

Figure 19. VS1001 MP3 Decoding Flow

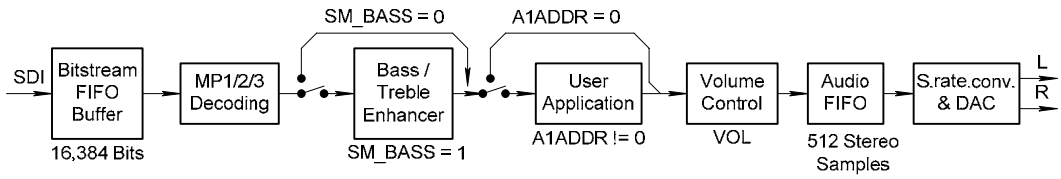


Figure 20 shows the connection of the microprocessor to the VS1001 chip. The basic settings are listed for the microprocessor interface as follows:

- SO and DREQ are inputs, and the rest of the signals are outputs.

- When the SPI control is idle, the PI clocks should be set to low power.

- If the microprocessor has no SPI signal interface, the SO, SI and SCK signals could be implemented by the general I/O; however, the microprocessor must have a fast enough operation speed.

Figure 20. VGA Display Control Circuit

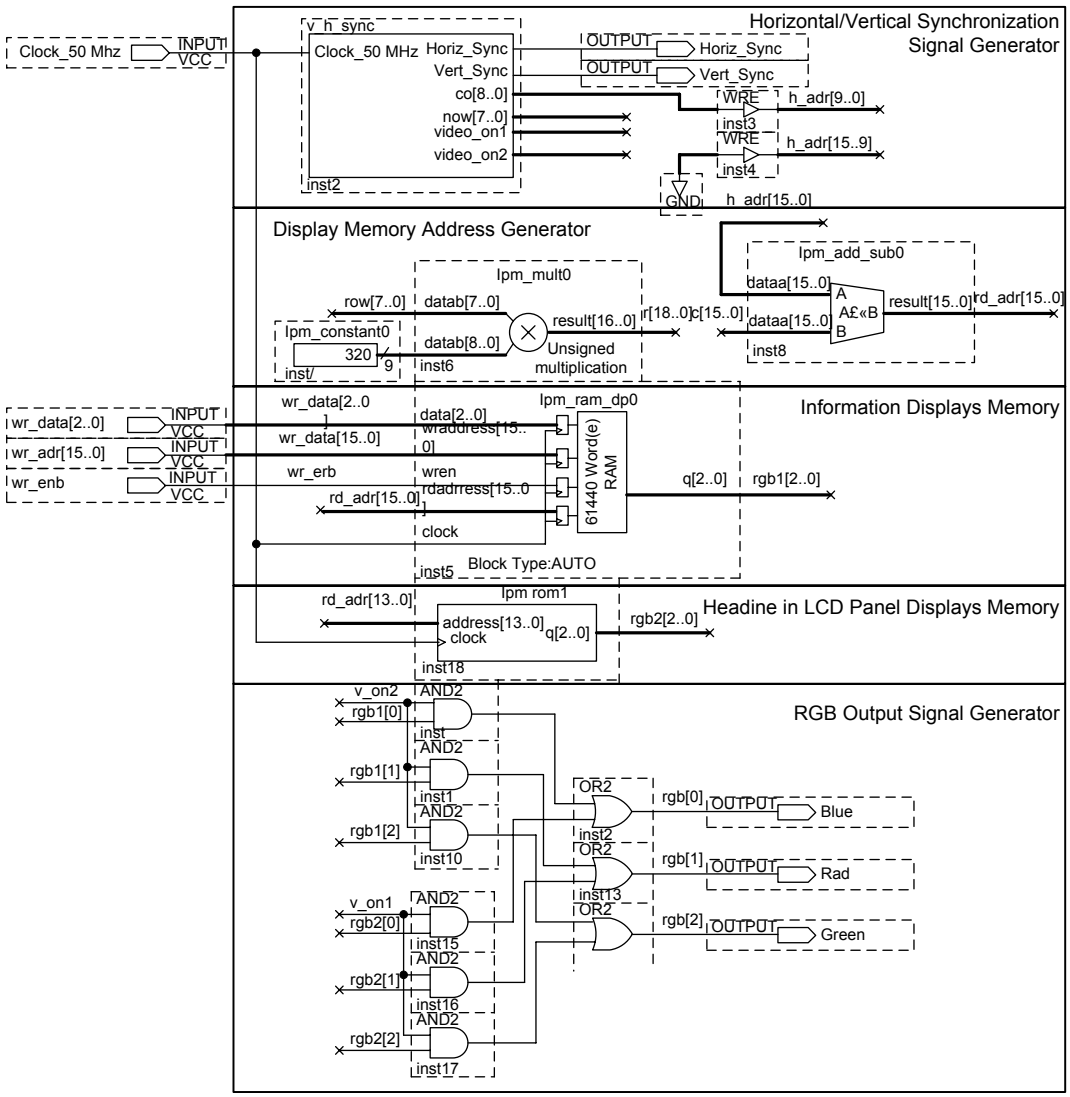
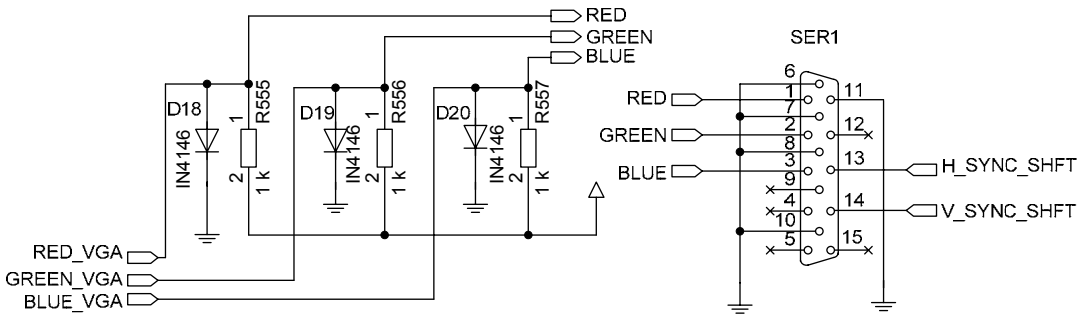


Figure 21 shows the VGA output interface circuit.

Figure 21. VGA Output Interface Circuit (from Altera UP3 Development Kit)



LCD Display Panel

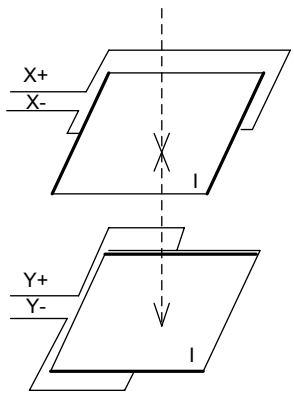
A dot-matrix LCD controller is added to the MP3 receiver for displaying the related MP3 information (such as MP3 file name, length, bit rate, receiving status, and play menu) on the 640 x 480-pixel LCD panel. The panel information operation is controlled by the touch panel.

Touch Panel

Touch panels are becoming more and more popular these days. They are mainly used in environments where there are space constraints and it is not convenient to use a normal keyboard. These applications include operating table supervision systems, self-service meal ordering systems, PDAs, cell phones, logistics management, and inventory management. To make it easier for the user, a four-wire touch panel, rather than an ordinary keypad, is used in the system. Its principle is described as follows.

The basic structure of the four-wire resistive element is very simple: it is made up of two resistive films (see Figure 22). On the X axis of the top resistive film, there are X+ and X- poles connected with the resistive film; and another two Y+ and Y- poles connected with the bottom resistive film. A voltage is applied interactively onto one side of the pole of the two-layer resistive film. When the two-layer resistive film is touched, a voltage value resulting from the touch of the resistive films can be measured via the pole on other side, which is not electrically connected, and hence the X and Y coordinate of the touch point can be obtained. The touch panel outputs X, Y coordinates in serial mode after processing by the interface integrated circuit. Its data format is shown in the tables following Figure 22.

Figure 22. Touch Panel Diagram



Description	Data Byte				
Pen Up	1	2	3	4	5
	BF	00000xxx	0xxxxxxx	00000yyy	0yyyyyyy

Description	Data Byte				
Pen Down	1	2	3	4	5
	FF	00000xxx	0xxxxxxx	00000yyy	0yyyyyyy

If the first serial data character is 0xFF, the following four characters are X, Y coordinates of the touching point of the touch panel; if it is 0xBF, the characters are the X, Y coordinates of the leaving point of the touch panel. We have mapped the system LCD screen into 40 horizontal characters and 30 vertical characters. Therefore, the program divides the read X, Y value with 40 and 30 to determine the touch display position of the character. Every time the panel is touched, the Nios II CPU issues a short beep to confirm the touch command.

Ethernet Chip

We used SMSC's LAN91C111 device as the Ethernet control chip in our MP3 receiver. The LAN91C111 device is a 128-pin TQFP, full-duplex network chip that can be connected with 8-, 16- or 32-bit microprocessors and can work in 10/100-Mbps mode. See the following table.

Item	Byte/Bit	7	6	5	4	3	2	1	0
Command code	1	C7	C6	C5	C4	C3	C2	C1	C0
X-High byte	2	0	0	0	0	0	X9	X8	X7
X-Low byte	3	0	X6	X5	X4	X3	X2	X1	X0
Y-High byte	4	0	0	0	0	0	Y9	Y8	Y7
Y-Low byte	5	0	Y6	Y5	Y4	Y3	Y2	Y1	Y0

Currently, the LAN91C111 chip used by our system network interface has two main function blocks: the MAC and PHY. The MAC is used mainly for digital data processing and the PHY for simulation data processing. Figure 23 shows the LAN91C111 simplified circuit diagram with the MAC and PHY function blocks. Figure 24 shows the block diagram of the LAN91C111 device's internal functions.

Figure 23. Basic Connection Block Diagram

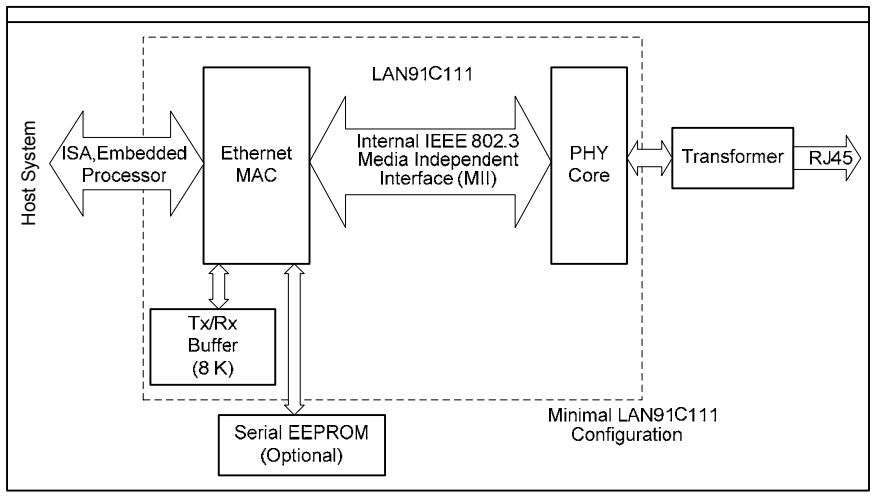
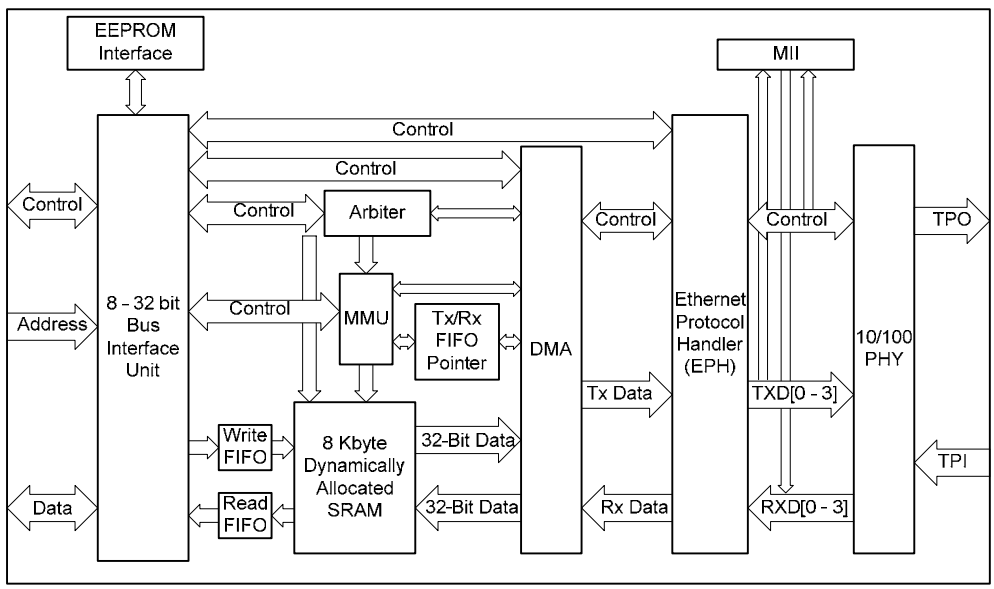


Figure 24. Basic Functional Block Diagram



The LAN91C111 device features an 8-Kbyte FIFO buffer, which is used to store the transmitted and received data packets. The FIFO buffer can be accessed externally in DMA mode. The device uses 9,346 (64 x 16-bit EEPROM) to store resource configuration (e.g., I/O address, boot ROM base address, and interrupt request source) and ID parameters. The chip control circuit comprises four register banks. The first 16 registers are used for control and status, registers 16 through 23 are used for DMA data access, and register 24 through 31 are used for chip reset. The transmitted or received data packets range from 60 to 1,514 bytes:

Item	Description	Size
Destination address	MAC address of target node	6 bytes
Source address	MAC address of sending node	6 bytes
Length	Packet length	2 bytes
Data	Data	46 ~ 1,500 bytes

The transmit and receive flow of data packets are shown in Figures 25 and 26, respectively. The packet transmission flow includes memory configuration, packet buffer writing, packet array arrangement, and transmission.

Figure 25. Data Packing Transmission Flow

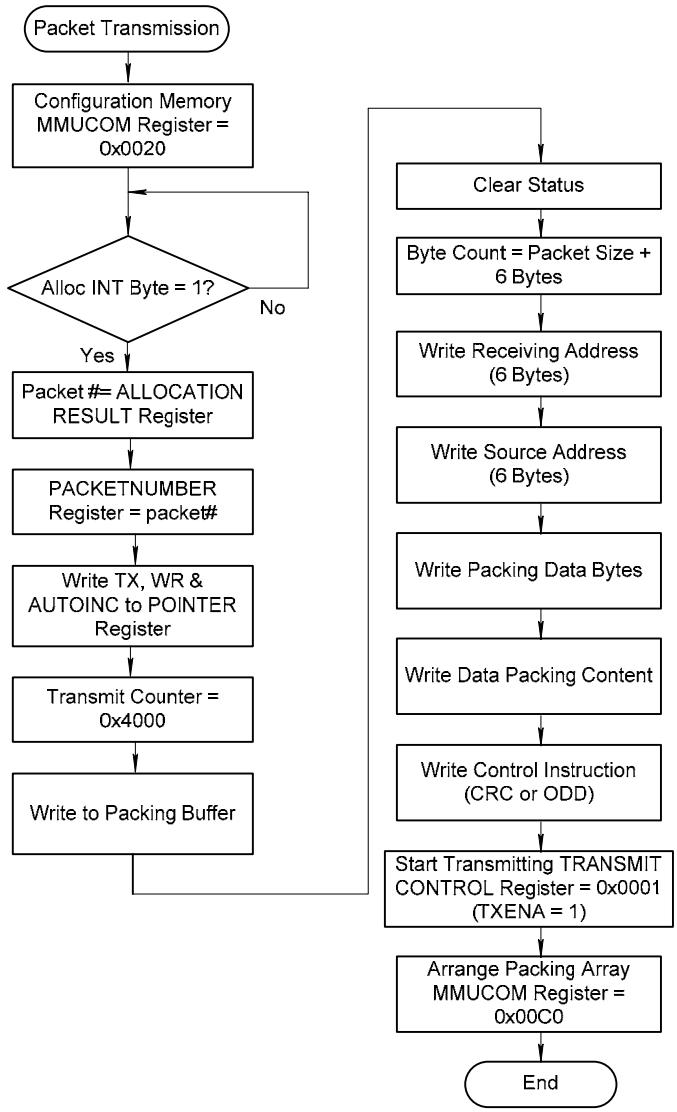
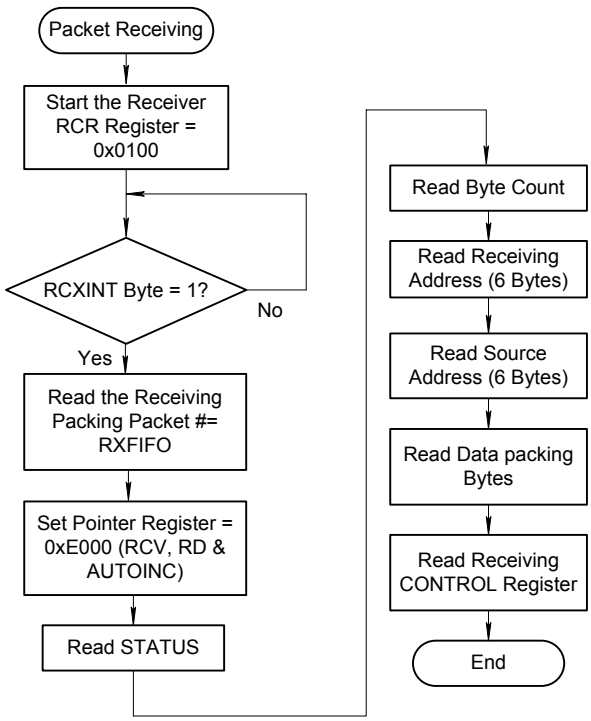


Figure 26. Data Packet Receiving Flow



Operating System

We designed the system software based on the uClinux RTOS that can be implemented on the Nios II processor. The main functions of each task are briefly described as follows (see Figure 27):

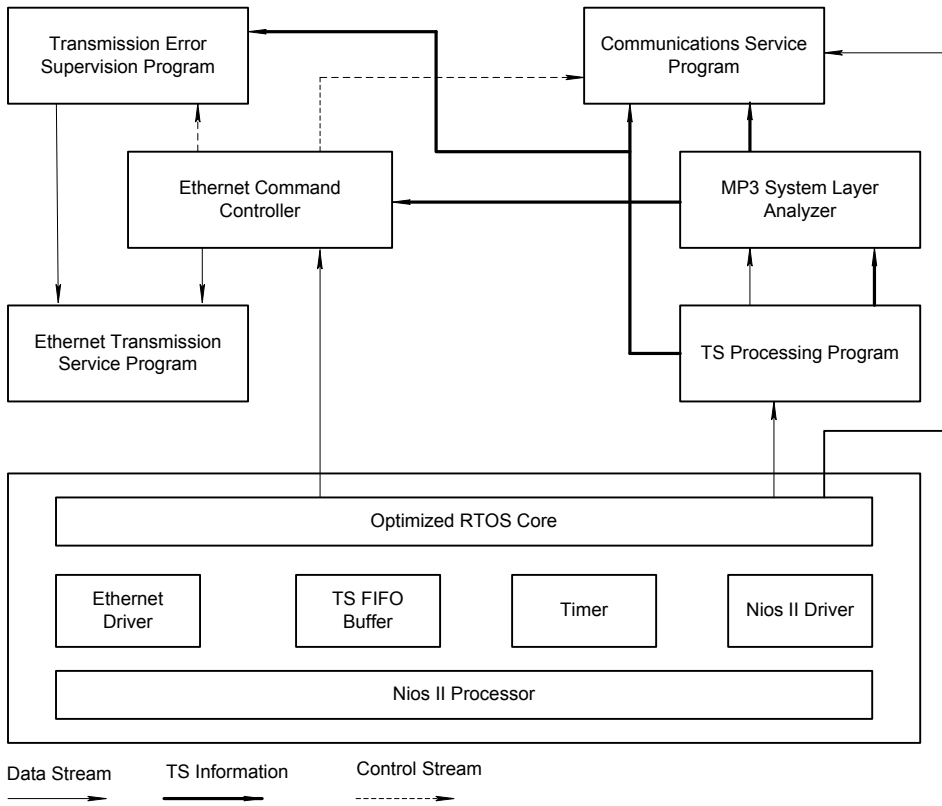
- *Transport stream (TS) processing program*—This program implements the TS input, output, and memory management functions. It is similar to the physical layer of the communications system, which implements transmission control of the hardware system. In general, this task stores the data read from the FIFO buffer in different segments of SRAM, and submits the index address. Meanwhile, the data stream to be output is read from SRAM and written to the output FIFO buffer according to the application requirements. In special cases, the input data volume is modified according to the input data stream rate and memory availability.

- *MP3 system layer analyzer*—The MP3 information that is entered into the data stream is analyzed according to the MP3 system layer standard. Various acquired parameters and audio information are sorted and stored. The data is reassembled according to the simple network management protocol (SNMP) data structure and is updated when necessary. The results are then submitted to the trigger program of the transmission error transaction.

- *Transmission error supervision program*—The program first completes the synchronization and analysis in succession according to different priorities, and then stores the analysis results data in a structured format to submit to the communications module. It implements a fault prompt and alarm via the pre-designed fault mode. Too many error alarms cause an information jam; therefore, it is helpful to be judicious in judging problems by combining it with the related errors into higher-level alarm information.

- *Communications service program*—The program completes the design of Ethernet transmission control according to TCP/IP protocol and SNMP protocol. Data output transmits the statistical information database and the analytical database to the controller side, based on the standard SNMP protocol. Meanwhile, control command communications are made via TCP or UDP protocols. Semantic analysis can be made for statistical information data entered by SNMP by means of additional analysis software. Nevertheless, the program needs to transmit local hardware timing information as the reference, or display analysis data on the console directly. After adding the web server function to the communications service, the analysis results can be displayed directly via a browser.

Figure 27. Embedded Operational System Software Structure



Design Methodology

The development of the system is divided into system planning, hardware circuit design, software program design, OS migration, and system integration test.

At the beginning of system planning, we decided to perform MP3 coding by combining software and hardware. During the testing phase, we realized that using only software programs will not work in real-time, as they consume large amounts of time. Also, if we used MP3 coding hardware circuits along with the FPGA, these circuits would occupy too many chip resources, and some circuits would operate poorly. Therefore, we decided to use an additional MP3 coding chip to save the process of implementing a simulation circuit.

The hardware circuit design includes the Nios II CPU, touch pad, compact flash (CF) card, memory, Ethernet interface, VGA display interface, and MP3 chip control interface. We developed the circuits in VHDL, and used the SOPC Builder tool for the CPU and the peripheral design. Finally, we used the Quartus II software to compile and compose our design.

The software program was written in the C language and compiled with the gcc compiler. For making easy modifications to the system during development, we performed the interface circuits tests of all using independent programs without involving the OS. This process saved us development time because we did not have to recompile the software/hardware combination of the design.

The uClinux RTOS was downloaded to the Nios II development board. The CF card access and Ethernet operated under the RTOS, so we did not have to write drivers.

After completing the actions and tests above, we took enough time to integrate and test the system, using the following steps:

1. Establish the kernel project. Plan and compile the Nios II CPU system that generates the PTF file for the establishment of the kernel (including selecting the development board whose kernel configuration is set to the Stratix® or Cyclone™ device).

2. Establish a file system project, including the basic instructions to be performed on the OS, as well as some application programs such as boa server, telnet, and ftp.

3. Download constructed kernel (vmlinux.bin) and file system, romfs.bin, to the flash ROM, and then download the SRAM Object File (.sof) or Programmer Object File (.pof) of the CPU. Once the software development kit (SDK) window changes to Nios II terminal mode, you can start the OS and check some messages, as well as control the OS after inputting the user account and password.

4. The development of the application program must establish the write program, build the makefile, and specify the option setting of the compilation in the integrated development environment (IDE). The program should copy the .exe file generated by compilation to the filesystem/bin subdirectory, rebuild the project to download to the flash ROM, and start the RTOS to test the program.

5. When testing a program, you must first start up the RTOS network connection and download the program execution file to the CF card via ftp or telnet.

When the tested program is in an endless loop, it can skip over the execution program if the user presses the Ctrl+C keys when testing with the SDK. However, if you press the Ctrl+C keys in the RTOS, the program only leaves terminal mode and does not end the program.

Design Features

This design project integrates the Nios II CPU, the MP3 encoder, Ethernet, the RTOS, and other software/hardware technologies on the SOPC, and completes the embedded network MP3 broadcast system. The system can download MP3 messages through Ethernet, which are then played directly on the MP3 receiver. This system is applicable for use in:

- Public places, replacing conventional loudspeaker broadcasts and providing good quality audio and voice messages.
- Music audio in shops, replacing CD audio players and providing the convenience of the latest MP3 technology.

The main function blocks in the system design include:

- Nios II CPU and peripheral circuit plan
- MP3 encoder interface circuit
- VGA display interface circuit
- Embedded Ethernet
- Embedded OS uClinux
- MP3 file server
- SOPC software/hardware system integration

Considering the flexible design of the system software/hardware, we adopted the SOPC design methodology to complete the system. If the system were to be implemented only using software programs, it would be extremely difficult to process the MP3 data in real-time, or it would require a higher performance processor, which is too costly and consumes more power. Many application programs of the system cannot be realized in hardware circuitry, and therefore are not flexible. Therefore, an FPGA that contains the Nios II soft processor is the best choice for this design.

Conclusion

The purpose of our design, the Embedded Network MP3 Playing System, is to make general public announcements at selling booths and public places. These messages are played in real-time, and could include popular music or electronic information (in MP3 file format) that can be transmitted to MP3 players by the main control room to the Internet or LAN.

The system core uses a 32-bit RISC Nios II embedded soft processor, which was released by Altera in 2004. The system OS is uClinux. The development tools, such as the SOPC Builder and the Quartus II software version 5.0 IDE, helped us to partition the software/hardware module design, compile, combine, program, and test, as well as to integrate the program into the Altera Stratix FPGA development board.

The Nios II processer has three types of optimal CPU variants: one that has high system performance, one that uses the fewest logic resources, and one that provides a balance between system performance and logic resources.

Although Altera's development tools were good, we faced a few problems during development.

- It was difficult to make an optimal choice because selecting the CPU, parameters of the peripheral components, and even circuit combinations were very complicated.
- Whether the RTOS was in use or not, the OS name was not matched during hardware development, which we needed to change manually.
- It was difficult to write this report because the sections were out of order and contained repeated information. Instead, we would like to suggest the following project report sections: Motivation and Purpose, System Architecture (including circuit diagram, program flow, and specifications for use), Design Principle, Design Description (detailed circuit diagram and program description), Test and Experiment Result, Conclusion, and Appendix.

The initial hardware design of the system was to select the Nios II /f CPU using SOPC Builder, and add a user logic interface, designed by us, to connect the VGA display. At the beginning of the design, we took more time to add a display with extra pixels, but considering the memory resources used by the system and the size restriction of some RAM and ROM, we used the Nios II /s CPU instead, compiled it in the Quartus II software, and implemented a full-screen display. Besides the LCD display, the system also supports the touch screen input by the UART interface, and allows further design modifications on the touch screen functions. The play of the MP3 decoder is controlled by two SPI interfaces. We spent a lot of time at the beginning of the project on this aspect, due to our unfamiliarity with the communication protocol and SPI timing, learning how the function base of the SPI is applied in the Nios II system, and verifying using an external MP3 decoder circuit. After completing a few modules, we still had some difficulties when integrating the uClinux RTOS, accessing the CF card, and performing Ethernet transmission. We overcame these problems in the end.

For this kind of project, no matter how the software and hardware cooperate, or what the setting and operation of the RTOS, each function that needs to be implemented successfully must be understood and developed with the correct debugging procedures. You cannot wade into this design blindly and use trial and error. That approach wastes a great deal of time and lowers your confidence. A special thanks to our instructors for providing such great help and important suggestions about the design and operation of this product. We thank all our college mates for their active participation and sincere devotion during this competition. Through this competition, we learned how to design an embedded system and perform system integration. We had a pleasant experience entering the competition, and we hope to do even better next time.

Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights.