SOBA Server: RTP Streaming Audio over Ethernet
Design Document ver. 1.0

Warriors:

Avraham Shinnar	as1619@columbia.edu
Benjamin Dweck	bjd2102@columbia.edu
Oliver Irwin	omi3@columbia.edu
Sean White	sw2061@columbia.edu

March 31, 2005

1 Overview

We will use the FPGA and onboard peripherals to input analog audio from local audio sources, convert the audio from analog to digital, encode the audio, and stream the audio over the Internet. We are also considering a direct data line between the audio codec peripheral and the ethernet peripheral. The primary components of the system include:

1. Analog audio input using the AK4565 Audio Codec. This codec takes direct analog audio and converts it into a serial bit stream. We will build a peripheral that controls the codec and converts the bit stream into bytes that can be encoded for packetization. These bytes will be written to the SRAM where we can manipulate them directly.

2. Audio Encoding. Initially, we will just use PCM encoding and if we have time, we'll experiment with non-linear mappings like u-law and a-law. We may also explore compression if we have time.

3. UDP and RTP packet construction. We will build packets two be handed to the Ethernet Controller. The packets will be designed to be primarily fixed data so we can quickly pass them on.

4. Internet streaming using the on-board ASIX AX88796L. Probably the most difficult part of our project, we will be sending packets over the Internet using this Ethernet Controller.

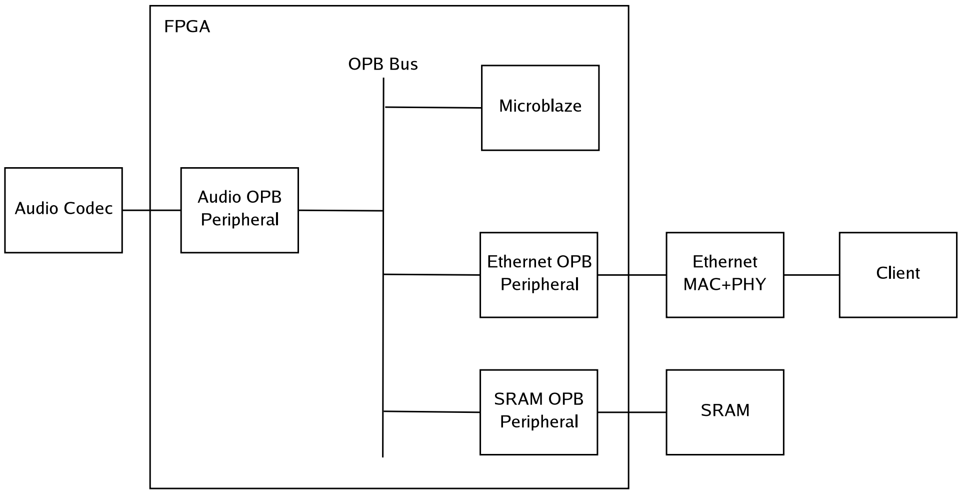

Figure 1: System Diagram

1.1 Design Process

We will start by getting the Ethernet peripheral working enough that it can make the Ethernet Controller send packets out. We will monitor these packets using a laptop running Ethereal or some similar software for listening to the Ethernet port.

Concurrently, we will start developing the audio peripheral to the point where we can initialize the audio and convert the serial bit stream to useful bytes. We may test control over the codec by using the pass-through mechanism. Validity of the data stream will be tested.

Next, we will focus on sending real RTP packets that can be received by a client and then developer full packet construction.

Finally, we will pass the audio data to the packet construction module. At this point, we should be able to stream audio.

We're expected to have 75% done April 14th and our target is to be finished by April 22nd.

2 Audio

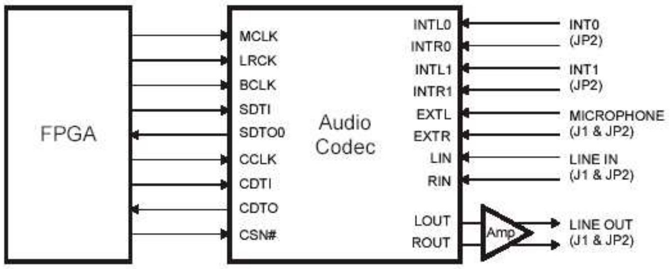

Figure 2: Audio Codec Diagram

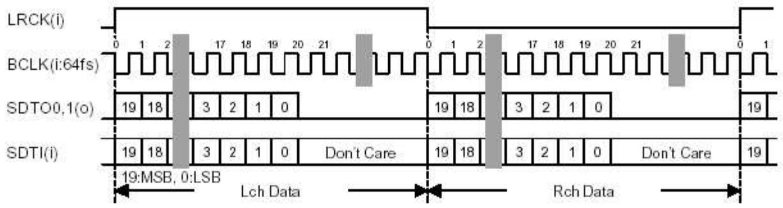

Figure 3: Audio Timing Diagram

The XSB has an AK4565 attack Audio Codec chip hardwired to input pins and jacks for rapid conversion of analog audio input to digital serial bitstreamed output. To make use of this chip we must design an OPB peripheral we will refer to as the Audio Codec Peripheral (ACP) on our FPGA that will communicate with the audio codec, retrieve serial data from the codec, and buffer that data so it can be sent out across the OPB.

The codec requires three clock signals:

- Master (12.5 MHz)
- Serial (1.5625 MHz)
- Frame (48.828 kHz)

which the ACP must supply to the codec.

The output bitstream will be fed into a memory module inside the ACP that acts as a shift register and will signal an interrupt once a full byte of data has been received. This will trigger a specific interrupt which will grab the data stored inside the shift register, reset the shift register, then bring the data through the OPB Bus to a place where the data can be encoded and placed as the payload into appropriate outgoing packets.

Each sample is 16 bits long. This way we could send two samples over the 32 bit OPB Bus.

2.1 Initialization

- initialization
- audio input and conversion
- can we mix using the codec?
- Can we multiplex?

3 SRAM

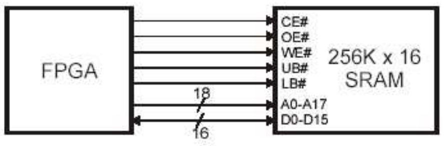

Figure 4: SRAM pins Diagram

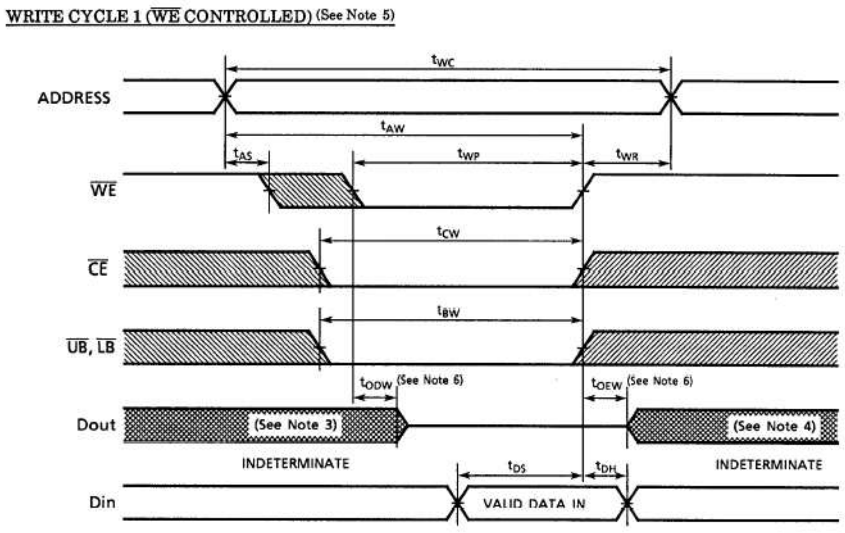

Figure 5: SRAM Timing Diagram

Our SRAM will be based on LAB6. We may not need it if we can squeeze our program into the existing memory. Our packet data size is relatively small so we probably don't need it for that.

4 Ethernet

The ethernet controller will first be initialized and then used to transmit packets. Once we have a packet constructed, we will pass it to our Ethernet Controller (ASIX AX88796L) via an OPB Bus peripheral. The peripheral will control the Ethernet Controller using the ISA Bus timing scheme found in figure 7. The NIC will be controlled based on writing to registers. Data will be sent by placing it in the on-chip SRAM.

We will update our timing scheme based on the experience of the JAYCAM group to avoid control issues. In particular, the CHIP select will be asserted prior to the Write Enable (not concurrent). We will use 4 cycles to write to the chip:

- All signals except WE are asserted
- WE is asserted
- wait cycle to allow DMA write
- wait cycle to disable data

Note that a 100 Mhz clock was used to ensure sufficient time for the set-up of the write-data (latching).

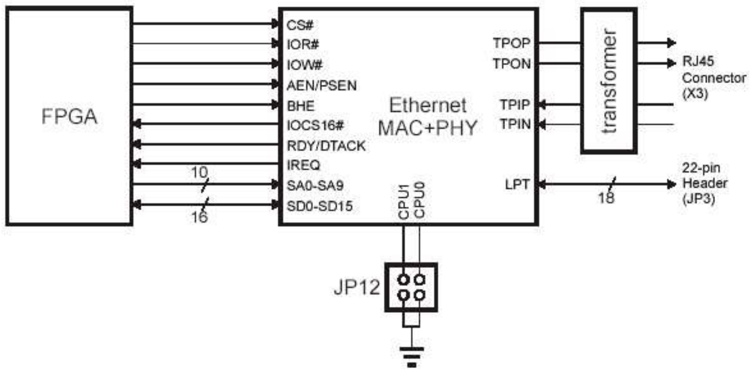

Figure 6: Ethernet codec Diagram

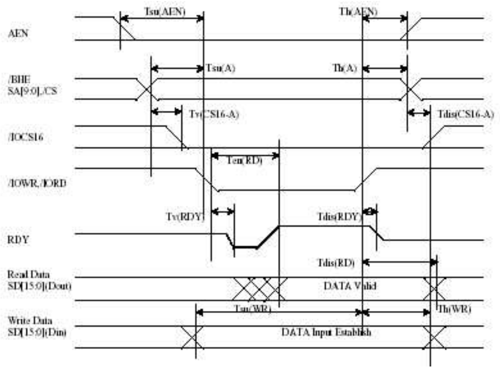

Figure 7: Ethernet Timing Diagram

4.1 Initialization

The following steps will be taken to initialize the Ethernet Controller:

1. Write 21h to Command Register to abort current DMA operations.
2. Wait 2 milliseconds (timout for inter-frame gap timer)
3. Write 01h to Data Control Register to enable 16-bit word transfers.
4. Write 00h to both Remote Byte Count Registers to zero out DMA counter.
5. Write 00h to Interrupt Mask Register to mask interrupts.
6. Write ffh to Interrupt Status Register to clear interrupt flags
7. Write 20h to Receive Configuration Register to put NIC in monitor mode
8. Write 02h to Transmit Configuration Register to put NIC in loop-back mode
9. Set RX Start and Stop, Boundary, and TX start page.
10. Reset interrupt mask and flags.
11. Write 22h to Command Register to start NIC
12. Write 00h to Transmit Configuration Register to set normal transmit operation

4.2 Transmission.

Once the controller is initialized, we can use it to send data. Sending data requires that we place the data in on-chip SRAM and then move the data into the transmission FIFO using a DMA operation. Note that writes to the Ethernet will be 16 bits.

1. Send the full packet data to the Ethernet's on-chip SRAM.
2. Write 22h to the Command Register - this activates the controller
3. Move data from on-chip SRAM to FIFO by starting Remote DMA write.
4. Write 26h to the Command Register to set the transmit bit and initiate transmission.

4.3 Packet Construction

There are generally three protocols involved in streaming media across the Internet: RTP, RTCP, and RTSP. RTP is a UDP based transport protocol for delivering media. RTCP is part of RTP and provides QoS and synchronization facilities. RTSP is a control protocol for setting up and directing media streams. A given system may use one or all of these protocols and our system will just implement RTP.

The audio data will be buffered in bram, in the middle of a packet template. After enough bits have been recieved, we will modify the packet wrapper to correctly reflects its payload and send it off. Note that all packets will be the same length.

Fields in a packet:

- **Headers (52 bytes)**
- MAC header (14 bytes)
  - Destination MAC address (6 bytes): fixed
  - Source MAC address (6 bytes): fixed
  - Length (2 bytes): fixed
- IP header (20 bytes)
  - Version (4 bits): fixed
  - IHL (4 bits): fixed
  - TOS (1 byte): fixed

  - Total Length (2 bytes): fixed
  - Identification (2 bytes): fixed
  - Flags (3 bits): fixed
  - Fragment Offset (13 bits): fixed
  - TTL (1 byte): fixed
  - Protocol (1 byte): fixed
  - Header Checksum (2 bytes): updated
  - Source IP (4 bytes): fixed
  - Destination IP (4 bytes): fixed

- UDP header (6 Bytes)
  - Source Port (2 bytes): fixed
  - Destination Port (2 bytes): fixed
  - Length (2 bytes): fixed
  - Checksum (2 bytes): updated
- RTP header (12 bytes)
  - Version (2 bits): fixed
  - Padding (1 bit): fixed
  - Extension (1 bit): fixed
  - CSRC count (4 bits): fixed
  - Marker bit (1 bit): fixed
  - Payload type (7 bits): fixed
  - Sequence number (16 bits): updated
  - Time stamp (32 bits): updated
  - SSRC (32 bits): fixed
  - CSRC list (0 bits): fixed
- **PCM payload (1460)**

As discussed, the PCM payload is put in place by the audio peripheral. We then set the sequence number and timestamp. Note that they are simply a fixed increment from their previous values. Finally, the IP and UDP checksums are calculated and set.

5 Microblaze

We will use standard Microblaze set-up based on our previous projects.

6 Audio Player

We will use MPlayer as our audio client. MPlayer has facilities for playing PCM audio streamed over RTP already so we don't need to write an additional client.It will also support alternative schemes like u-law PCM so we don't need to change players if we decide to use nonlinear encoding.