Video-Conferencing System

Evan Broder and C. Christoher Post

Introductory Digital Systems Laboratory

November 2, 2007

Abstract

The goal of this project is to create a video/audio conferencing system. Video will be captured from a camera and then processed using JPEG compression. The resulting compressed signal is then framed for serial transmission along with an audio signal and sent to the decoder with a checksum to ensure data integrity. The decoder then extracts the separate signals, reverses the JPEG compression, displays the resulting image on the screen, and outputs the audio to a speaker.

Contents

List of Figures

1 Introduction

For this project, we will implement a real-time video conferencing system. In order to reduce the bandwidth needed for video transmission, we will perform JPEG compression on each video frame.

The target for the video system is a 320x240 image at 15 frames per second. Because the compression happens in several steps, it can be highly pipelined. In addition, JPEG operates independently on small blocks of the image, so it is highly parallelizable.

In order to maintain data integrity across the transmission medium, the video and audio data will be formed into packets with a checksum to detect malformed packets on the receiving end.

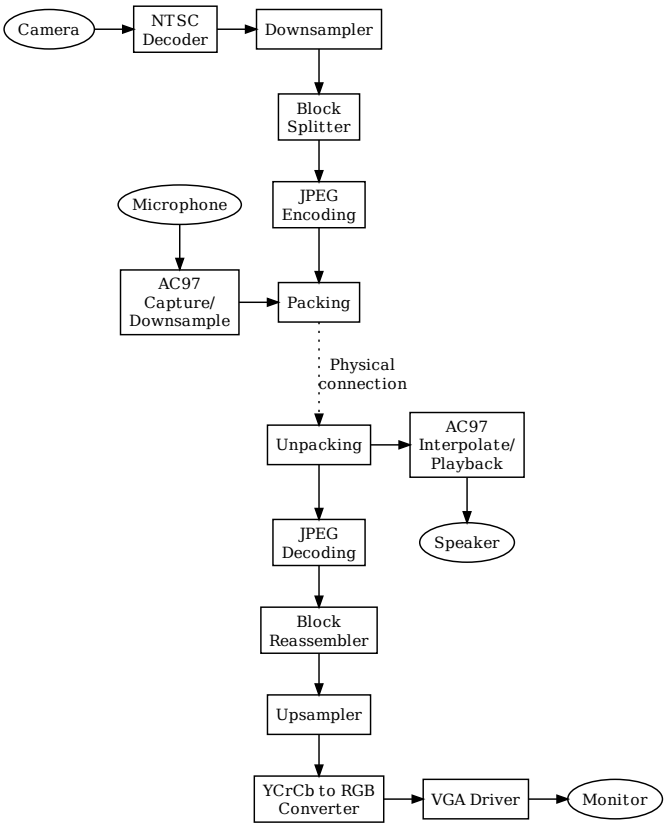

Figure 1: The top-level block diagram for the Video-Conferencing System

2 Modules

2.1 NTSC Decoder (Chris)

This module will take the output of the ADV7185 ADC on the labkit and decodes its packet sequence. The data from the ADV7185 will be interlaced, and since the transmission image

size is 320x240, there is no reason to deinterlace the incoming video. Therefore, this module will output images at 640x240. Additionally, this module will reduce the framerate of the video to the desired 15 frames per second.

2.2 Downsampler (Chris)

Our target resolution is 320x240. Since the incoming signal will be 640x240, we will, at the very least, need to downsample the entire image by a factor of two in the horizontal direction. Additionally, the human eye is less sensitive to changes in chrominance (color difference) than it is to changes in luminance (brightness). Therefore, the JPEG standard specifies that each of the two chroma channels is downsampled by an additional factor of 2 in each direction. The result of this operation is stored in BRAM where it accessed by the Block Splitter.

2.3 Block Splitter (Chris)

The rest of the data flow operates on 8x8 pixel blocks. This module reads data from BRAM one 8x8 block at a time and feeds it to the DCT.

2.4 DCT (Evan and Chris)

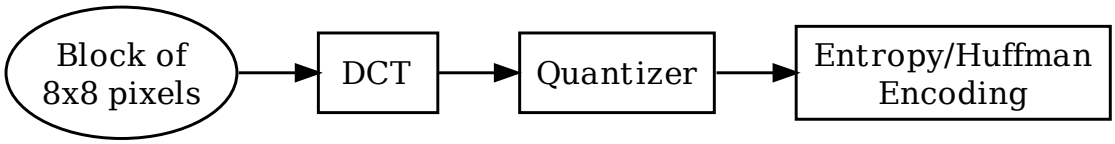

Figure 2: The JPEG compression algorithm is applied to each block of 8x8 pixels.

A DCT (Discrete Cosine Transform) translates spatial data into spatial frequency data. The human eye is most sensitive to changes in the DC ($f = 0$) and low-frequency AC component of the image and least sensitive to the high-frequency AC components. These low frequencies to which the eye is most sensitive are grouped in one area of the result matrix.

In JPEG compression, each 8x8 block of pixels is first converted from unsigned to signed integers, such that the range is centered around 0. Then, a two-dimensional (type-II) DCT is applied to each 8x8 block of pixels. The result is defined by

$$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

where u and v represent the spatial frequencies in the horizontal and vertical directions, respectively, $g_{x,y}$ is the pixel value at coordinates (x, y), G is the result matrix (still of size 8x8), and

$$\alpha(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

This result is passed to the Quantizer.

Since, from an algorithmic standpoint, this is the most significant and challenging module, we will be developing it cooperatively.

2.5 Quantizer (Evan)

If the brightness of a block is varying at a high spatial frequency, the human eye has a difficult time detecting the exact strength of that variation. Therefore, it is not necessary to retain as much information about those high-frequency components. The information is reduced by dividing each element of the frequency-domain matrix by a corresponding element of a constant quantization matrix and rounding the result. This quantization matrix is constant for all 8x8 blocks of a given channel; however, the matrix for the luminance channel may be different from the matrix for the chrominance channels.

Quantization is the lossy stage of the compression. Because many of the elements of the quantization matrix are relatively large (i.e. > 50 for 8-bit data), the elements of the result matrix become very small numbers, or even zero. This reduces the number of bits required to store the information. Because the quantization coefficients are higher in the lower-right corner (i.e. the higher-spacial frequencies), this region is most likely to go to zero, leaving the majority of the information in the top-left corner.

If there is enough space, it would very interesting from a pedagogical standpoint to be able to alter the quantization tables at run-time. However, this significantly increases the area required for this module and may therefore make it impractical.

2.6 Entropy/Huffman Encoder (Chris)

In this stage, the elements of the quantized matrix are first rearranged in a "zigzag" order (see Figure 3). After the DCT and Quantizer process a block, almost all of the information is in the top-left corner, and the zeros tend to be more concentrated in the lower-right corner. By applying this zigzag ordering, the new string of values typically ends in a long run of 0's. A Huffman encoding is then applied to this string, with a special EOB (End-Of-Block) character that ends the sequence prematurely when all remaining values are zero.

2.7 Audio Capture (Chris)

This module captures audio from the AC97 and reduces the sample frequency to a level suitable for a human voice (on the order of 16 kilohertz). It then passes the information to the Packer module. Depending on the timing parameters, it may be necessary to delay the audio output slightly in order to maintain sync with the video.

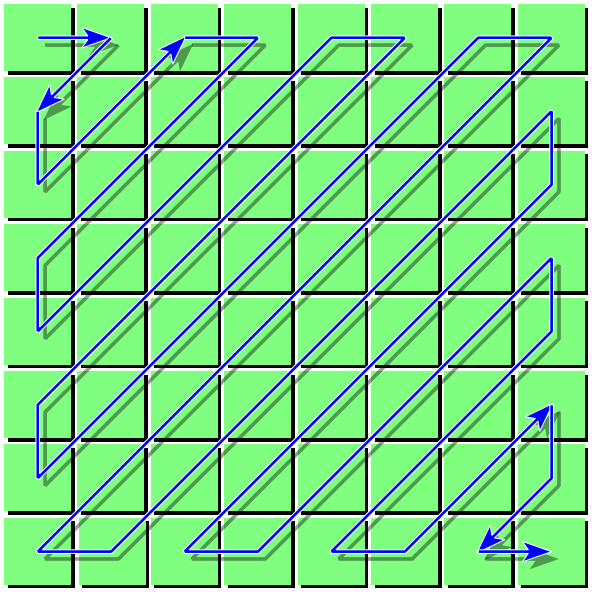

Figure 3: "Zigzag" ordering of elements in the block matrix [1].

2.8 Packer (Evan)

In this stage, the data from the video and audio encoding modules is wrapped in a simple packet format that specifies the type of packet (audio or video), the data length (since JPEG compression does not produce blocks of a fixed size), and a CRC value to ensure data integrity along the transmission medium. In addition, each packet will start with a fixed sequence to aid in finding the beginning of a packet. To prevent false starts, if this sequence occurs anywhere within the packet, it will be replaced by some alternative sequence. In addition, the module will interleave audio and video packets such that the time domain described by the data is approximately the same for each channel. The output of this module goes to whatever physical interconnect is used between inputs and outputs.

2.9 Unpacker (Evan)

This module stores the most recent valid audio and video packets in ZBT RAM. It receives incoming packets, extracts the data from the packets, checks the CRC against that data, and forwards the data along if it is valid. If it is not valid, it retrieves the most recent valid data from the ZBT cache and passes that instead.

2.10 Audio Playback (Chris)

New data from the Unpacker module is linearly interpolated back to 48 kilohertz and passed to the AC97 output.

2.11 Decoder

The serial stream from the Unpacker module is reassembled into a bitmap image. For each 8x8 block, this involves decoding the Huffman sequence, dequantizing the resulting matrix, and performing an inverse DCT on the result. The 8x8 blocks are then reassembled into a whole image, which is then upsampled to the original resolution.

Each of these steps is fundamentally related to a step in the encoding process. Therefore, each module will be developed by the same team member that developed its encoding counterpart. The one exception to this is that Evan will develop the Upsampler.

2.12 Color-Space Conversion and Display (Evan)

The decoded bitmap image is then converted from the YCrCb color-space to the RGB color-space and displayed on a VGA monitor.

3 Testing

For the modules in the encoding data path, their output is well-defined by their input. Therefore, we can create a series of test data and verify the result against a MATLAB script.

Additionally, if an encoding and decoding pair of modules are directly connected together, the output from the decoding module should always be the same as the input to the encoding module.

References

[1] "Jpeg zigzag.svg — wikimedia commons," 2007, accessed 1-November-2007. [Online]. Available: http://commons.wikimedia.org/w/index.php?title=Image:JPEG_ZigZag.svg&oldid=5483305