Video Conference System
CSEE 4840 Design Document

Manish Sinha, Srikanth Vemula, William Greene
Department of Electrical Engineering
Columbia University
{ms3766,sv2271,wmg2110}@columbia.edu

1 ABSTRACT

In this document, we present the design details for our Video Conference System. We first cover the hardware architecture of the system by explaining the SRAM controller, SDRAM controller, Ethernet controller, VGA controller, and Video controller. We then cover the software architecture of the system by detailing the activities of the NIOS processor. Following the hardware and software architectures, we include a brief constraint analysis section to verify our design decisions with respect to our primary system constraint: maintaining realtime video streaming. Finally, we conclude with a roadmap detailing the three upcoming milestones and how we intend to reach them.

2 INTRODUCTION

The overall system will contain the following components: two cameras with composite output, two Altera DE2 boards, two LCD monitors with VGA interface, and one Ethernet switch. The figure below illustrates this setup:

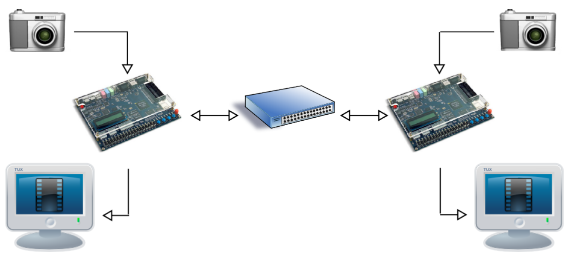

Figure 1: System layout [1] [2] [3]

The cameras will interface with the Altera DE2 boards using the CVBS protocol [4]. The Altera DE2 boards will interface with the LCD monitor using the VGA protocol. The Altera DE2 boards will communicate with the switch using the IP protocol. The underlying transport layer protocol will be UDP.

On the LCD monitor, the user will see a split-screen: the left half of the screen will contain the video coming in from the local camera; the right half of the screen will contain the incoming video produced by the user on the other end of the network. The decision to include the local video on the LCD screen was made to ease debugging.

3 HARDWARE ARCHITECTURE

The hardware architecture is as follows:

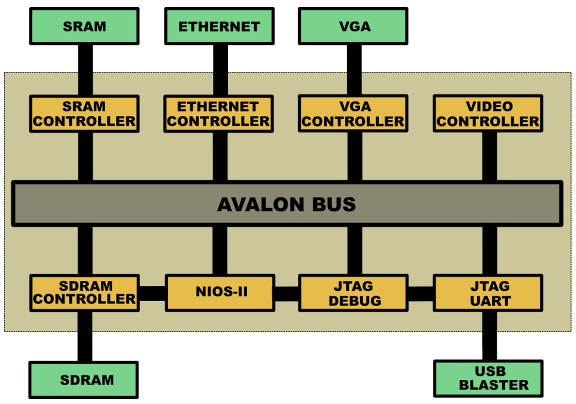

Figure 2: Block Diagram

3.1 SRAM CONTROLLER

The SRAM stores the video information that is received from the ADV718B Video Decoder. The output of the video decoder is first stored in a line buffer (not located in the SRAM). This line buffer is eventually transferred to the SRAM via the NIOS. The data that is received through the network is stored in another buffer called the data buffer which is also a part of the SRAM. The

SRAM controller provides the necessary address signals and control signals to write into or read from the two buffers in the SRAM when necessary. Each pixel needs 4 bits. Since each memory location can store 16 bits of data, we can store 4 pixels of data in each memory address. The two figures below detail the necessary signal timing we will need to adopt in order to interface with the SRAM.

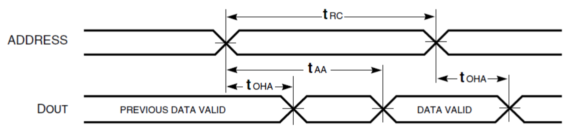

Figure 3: Timing considerations when SRAM is read [5].

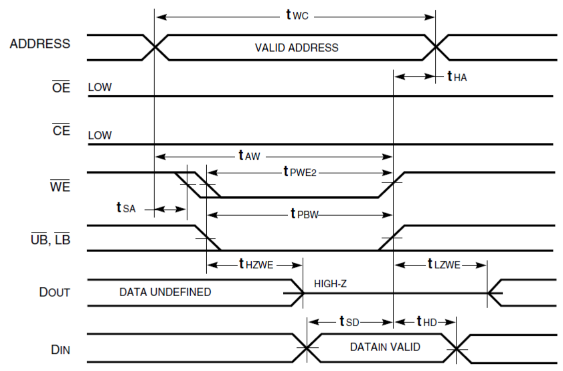

Figure 4: Timing considerations when SRAM is written [5].

3.2 SDRAM CONTROLLER

Since the memory available on the SRAM is not sufficient to store the buffers and the C code, we will use the SDRAM to store the C code. Using the SDRAM IP available in the SOPC builder, we will instantiate the SDRAM controller in the NIOS system that we plan to build. The port mapping has to be done in the top level entity. The SDRAM clock has to run faster than the NIOS system clock by 3 nanoseconds. This is ensured by having a phase locked loop component.

3.3 ETHERNET CONTROLLER

For the ethernet controller, we plan to use the existing verilog file provided in lab 2 in conjunction with SOPC builder. A driver, written in C, is also provided in lab 2 which we will make use of because we will use UDP as well.

3.4 VGA CONTROLLER

The VGA controller reads the data from both the video and the data buffer and displays both on the VGA. We will build on the VGA controller that we used in lab 3. The controller also provides the necessary sync signals and ensures that the video from the two buffers is displayed properly on the VGA. The VGA will be divided into two halves. One half will be used for displaying the video buffer that is the data received from the camera and the other half will be for displaying the data received from the network.

In order to maintain a good dynamic range with respect to luminance, the brightness of each pixel should be scaled with respect to maximum amplitude. Therefore, for every n-bit representation of a decoded camera pixel, an m-bit pixel will be provided to the VGA controller such that the maximum value of the n-bit pixel

$$2^n \tag{1}$$

will correspond to a similarly maximal m-bit pixel

$$2^m \tag{2}$$

and all other values will be scaled in a convenient manner down to corresponding values of 0 luminance. Such an approach may be necessary to maintain good performance, though this will be subject to experimentation. We will try to use mathematical techniques such as interpolation. If the mathematical operations can not be accelerated enough, then numerical approximations will be used.

One technique we may utilize is a “bit staggering” approach by which we inflate a 4-bit pixel to an 8-bit pixel by alternately inserting, in order, a bit taken from the 4-bit number and a ‘0’ bit so that an 8-bit output value is generated. As an example, take “1101”. This would correspond to a staggered 8-bit value of “10100010”.

The pseudo-code algorithm for the VGA Controller is on the following page.

```
process clock and clock'rising_edge;
signal pixel_cnt, line_cnt;
-- main loop
while (true) {
  -- wait until the frame buffer is filled
  if ready (local_frame_buffer) {
    if (line_cnt < 480) {
      -- read line number lin_cnt from the frame buffer
      line = local_frame_buffer[line_cnt++];
      -- iterate through pixels in the line
      for (col_cnt = 0; col_cnt < PIXELS_PER_LINE; col_cnt++) {
        -- pass each pixel to the vga controller after properly formatting the
        -- pixel to the appropriate vga controller format (e.g. 24 bit)
        vga_display_pixel(format_vga_pixel[col_cnt], col_cnt, line_cnt);
      }
    }
  }
}
```

Figure 5: VGA controller pseudo-code.

3.5 VIDEO CONTROLLER

We will use the Analog Devices ADV7181 Video Decoder to facilitate the decoding of the composite video signal from the camera. We will sample pixel data output from the ADV7181. Using a VHDL implementation, we will tie in the appropriate pins and control signals such that a hardware specified procedure will properly monitor the ADV7181 and buffer the output data in a manner suitable to construct a frame buffer in SRAM. Because the ADV7181 is highly configurable, it will be necessary to describe the use of the device generally. In the course of implementing the project, we will investigate particular features in further depth. A key responsibility of the video controller is to provide signals for other processes to ensure that they are properly synchronized with the data that is being sampled from the ADV7181.

The ADV7181 provides three pins specifically suited toward processing the encoded video signal: these are the SYNC pins ([6] p. 38). The horizontal sync, or HS, pin will signal that a horizontal synchronization has been read and implies that the VHDL procedure should prepare to receive a burst of pixel data, referred to as active video. An active video burst, which is essentially an encoded horizontal row of pixel data, is always preceded by a SAV sequence and followed by an EAV sequence. SAV and EAV consist of the following values issued over 4 clock pulses: FF 00 00 XY. An active video burst should run for the appropriate number of clock cycles given the input and output format. For an NTSC horizontal row, we expect the standard 640 pixels. If the ADV7181 pixel output is set to 16-bit, then luminance and chrominance data will be sent in parallel, each on the appropriate 8 pin port ([6] p. 58). Depending on the port mapping, the pixel output may be a luminance (Y) or chrominance (Cr Cb) component. It will be critical to appropriately identify the output ordering which will be configured to generalize and simplify our implementation. The vertical synchronization and field pins will be used to identify the beginning of a new frame. Based on the NTSC standard, a frame will consist of two interleaved fields. Further specification will be required to determine exactly how the ADV7181 will group fields and whether configuration of related options may simplify characterization. We will expect that 640 pixels will be transmitted between HS events and 525 rows will be transmitted between VS events. Ideally, the chip will be configured such that the FIELD event will represent a new frame (in other words, that 2 VS events have been received).

Because full composite video will exceed realistic network bandwidth limitations, we will use the simple technique of sample decimation to achieve suitably compressed video streams. Only a limited subset of the information present in the composite signal will be sampled for a given frame. Only luminance information will be sampled. Chrominance, or color, information will be ignored. The ADV7181 provides filter shaping for luminance such that a monochrome output will not carry artifacts of the chrominance encoding ([6] p. 26).

Estimation of memory requirements and data path timing will be made based on the assumption that a frame will comprise 320 x 480 x 4-bit pixels or 614400 bits (75 Kb). A line buffer for 320 pixels (or 160 bytes) will be provided for in the FPGA. This buffer will be alternately filled and read into off-chip SRAM. During an HS event, the line buffer data will be moved into SRAM, allowing for the next line to be read from the ADV7181, overwriting the buffer. An appropriate semaphore mechanism must be implemented.

General algorithm for reading Y output from the ADV7181 is on the following page.

```
process clock and clock'rising_edge;
signal sample_cnt, pixel_cnt;
-- main loop
while (true) {
-- if all sync pins indicate
-- a new frame is about to be encoded
if FIELD && VS && HS {
  if ready (frame buffer) {
    if read_signal(SAV);
      read_line = true; sample_cnt = 0; pixel_cnt = 0;
    if read_signal(EAV);
      read_line = false;
    if read_line {
      if (sample_cnt++ mod decimate_by == 0) {
        -- downsample the pixel luminance
        pixel = sample_and decimate_Y;
        line_buffer[pixel_cnt++] = pixel << 4;
      } else {
        skip frame;
      }
    }
  }
}
```

Figure 6: Video controller pseudo-code.

4 SOFTWARE ARCHITECTURE

4.1 NIOS

In our system, the NIOS processor does not work on any algorithmically involved procedures. Rather, it merely acts as a memory shuffler responsible for moving data from the line buffer to the SRAM. It will periodically poll a flag in the VGA controller to check whether the line buffer has been filled or not. Once the flag is active, it will move the line buffer (which is 16 bits large) to the SRAM (where one word is 16 bits).

The other task the NIOS needs to accomplish is construct the outgoing IP packets. Again, lab 2 will serve as a guide for this phase.

4.2 JTAG Modules

These modules will be used primarily for debugging purposes. The JTAG Debug & UART modules enable debugging statements like "printf()" to print in the console of the NIOS IDE.

5 CONSTRAINT ANALYSIS

5.1 Video

When streaming video over a network, bandwidth is a key concern. In our setup, both hosts need to be capable of transmitting *and* receiving a stream all while displaying the local video on the screen. Not only should the system be able to perform realtime video streaming, but the user should also be able to view the video in a watchable quality. There are three key variables that we have control over to influence both the bandwidth usage and the user perception: frame rate, frame size, and pixel depth.

5.1.1 Frame Rate

The rate at which the system draws a new frame on the LCD monitor is defined as the frame rate. If the frame rate is too slow, the user will not be able to establish that motion is taking place. Rather, he or she might conclude that they are watching a slideshow. Nominally, we would set the frame rate to a human's perceptual limit—29.97 frames per second. However, as it will soon become clear below, this is a far too ambitious goal because of bandwidth constraints. A framerate target range of 7.5 to 15 frames per second is a more feasable goal.

5.1.2 Frame Size

The size of the frame, or image, drawn on the LCD monitor will also affect the watchable quality by the user and the bandwidth requirements of the system. We have elected to draw both the local camera output in addition to the incoming streaming video from the other camera. The camera we will use outputs at a NTSC standard of 640 pixels along the horizontal and 480 pixels along the vertical. Because the LCD screen we plan to use has a 640x480 resolution, we plan to downsample the video coming in from the camera by half along the horizontal. Thus, we will be able to fit two 320x480 videos on one 640x480 LCD screen.

5.1.3 Pixel Depth

The Analog Devices ADV7123 chip on the Altera DE2 uses a 10-bit analog to digital converter. This makes the chip capable of outputting 1024 different colors for a given pixel. However, as noted earlier, we are only reading the black and white component of the input signal. This leaves us with the ability to output 1024 different shades of gray. However, does the user need 10-bits of precision when fewer could suffice? As a small experiment, we converted a color image to different pixel depths to give us an idea of what the best pixel depth is without significantly degrading picture quality:

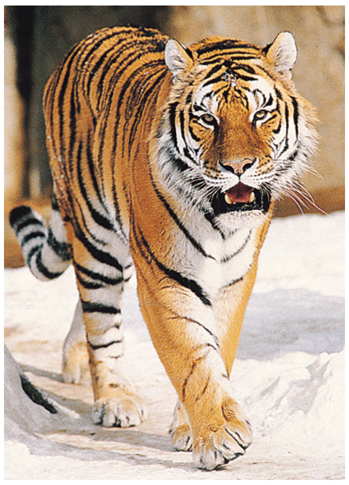
Figure 7: Original 557x400 8-bit Color GIF [7]

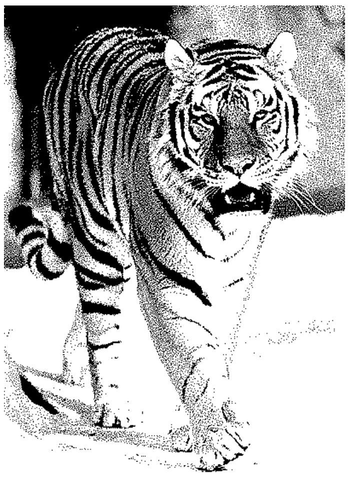
Figure 8: Monochrome (1-bit)

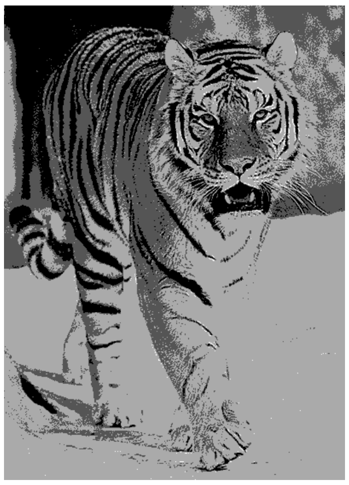
Figure 9: Black & White, 2-bit Pixel Depth

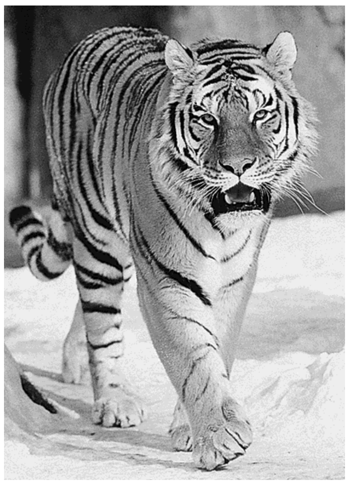
Figure 10: Black & White, 4-bit Pixel Depth

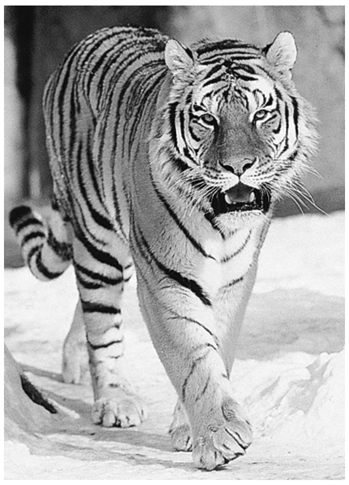
Figure 11: Black & White, 6-bit Pixel Depth

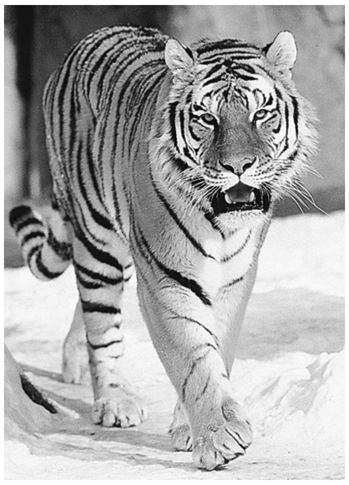
Figure 12: Black & White, 8-bit Pixel Depth

The difference between the original image and the monochrome image is stark. The difference is still noticable with the 2-bit pixel depth image as well. However, the 4-bit pixel depth gives decent image quality. In fact, the 6-bit and 8-bit images give diminishing returns compared to the 4-bit image. Leaving aside frame size and frame rate, choosing a 4-bit pixel depth for the output appears to be a reasonable choice.

Having reviewed the three critical variables, we can now proceed to some bandwidth calculations:

Frame Size	Frame Rate (fps)	Pixel Depth	Bandwidth·2 (Mbit)
640 x 480	30	10	184.32
320 x 480	30	10	92.16
320 x 480	15	10	46.08
320 x 480	15	4	18.432
320 x 480	7.5	4	9.216

Figure 13: Bandwidth Table

The bandwidth column has been scaled by two because in our system, both users will be transmitting and receiving video at the same time.

5.2 Timing

The core unit of information we wish to manage is the frame. The memory requirements of the frame will be variable so that flexibility is provided in managing time constraints of the system. However, the ability to drop frames will very likely serve to allow the system to recover from any minor delays or failures while still maintaining roughly the desired throughput.

The timing analysis is largely fixed by the well-defined characteristics of the NTSC composite video standard. First, we know that a new frame will be filmed by the camera at a rate of 30/1.001 fps. Given an estimated frame size of 75 kB, bidirectional network throughput at 30 fps will require 2 x 6144002 bits x 30 fps, or approximately 37 Mbps. This rate seems reasonable with respect to the bandwidth of the ethernet line. However, in providing for simultaneous display of local and remote frames, it seems likely that this frame rate will not be sustained.

A key property of the system will be its ability to monitor its internal frame buffers. In general, a new local frame should be displayed and transmitted simultaneously, but only after the last remote frame has been displayed. Thus the rate at which remote frames are delivered via ethernet will serve as a governor over the rate at which local frames are transmitted. A nominal balance should be maintained.

A new horizontal sync will occur roughly every $6 \cdot 10^{-5}$ s. It is important to consider the times that will be required to output the pixel data from the ADV7181 and the time required to move that data into the frame buffer in SRAM. The ADV7181 will never outpace itself, but our choice of design technique will depend on this memory throughput. The key consideration is whether a row of pixel data can be moved during the nominal horizontal sync period of the ADV7181, which for NTSC composite video, is approximately 250 cycles at 27MHz. Assuming that the SRAM controller will run at 25MHz and is capable of moving 1 word per clock in burst—a row of pixel data (320 pixels at 4 bits/pixel) containing 80 words of data—then given the roughly equivalent clock speeds, the SRAM transfer should be completed well before the nominal horizontal synchronization period. Thus, it is not a design requirement to utilize multiple line buffers since a single line buffer should be read to SRAM well before it is overwritten by the next burst of line data.

6 MILESTONES

6.1 Milestone 1: March 30

At this milestone, we hope to have the video controller complete. This means that the (1) reading and downsampling from the DAC phase, (2) store in line buffer phase, and (3) shuffle buffer to SRAM phase must all be complete.

6.2 Milestone 2: April 13

At this milestone, we hope to be able to display the video on the screen. This will entail reading data from the SRAM and properly scaling the data for the VGA controller.

6.3 Milestone 3: April 30

At the last and final milestone, we hope to be able to complete the network transmission of the video stream. Major functionality of the Video Conference System should be complete at this stage.

7 REFERENCES

[1] Network Switch, `http://openclipart.org/people/rgtaylor_csc/rgtaylor_csc_net_switch.svg`

[2] Altera DE2, `http://faculty.lasierra.edu/~ehwang/digitaldesign/webpages/de2.jpg`

[3] Computer Icon, `http://www.gnome-look.org/content/show.php/SudUbuntu+?content=93010`

[4] Maxim, *Video Basics*, `http://www.maxim-ic.com/appnotes.cfm/an_pk/734`

[5] ISSI, *256K x 16 HIGH SPEED ASYNCHRONOUS CMOS STATIC RAM WITH*

3.3V SUPPLY, http://www1.cs.columbia.edu/~sedwards/classes/2007/4840/ISSI-IS61LV25616-SRAM.pdf

[6] Analog Devices, *CMOS, 240 MHz Triple 10-Bit High Speed Video DAC*, http://www1.cs.columbia.edu/~sedwards/classes/2009/4840/Analog-Devices-ADV7123-video-DAC.pdf

[7] Tiger, http://static.howstuffworks.com/gif/willow/tiger-info0.gif

8 APPENDIX

8.1 Imagemagick

To construct the images, we used the following command to generate the monochrome image:

```
convert tiger.gif -monochrome tiger.png
```

To generate the various black & white images, we first converted the color GIF image to a black & white image via:

```
convert tiger.gif -colorspace Gray tiger_bw.png
```

Then we did the following (substitute 'X' with the desired pixel depth):

```
convert tiger_bw.png -depth X tiger_bw_X.png
```