ESC532 – Final Design Report: Real-Time Note Digitizer!

Konstantine N. J. Tsotsos ([redacted])
Sanae L. M. Rosen ([redacted])
Sean C. Bell ([redacted])
Team $A\backslash\backslash 350/\backslash\backslash 3$!

4 April 2011

Contents

List of Figures

List of Tables

1 Overview

1.1 Goals

The goal of this project is to create a tool to aid in digitizing handwritten notes. The project was motivated by instructors who display notes using overhead projectors. While students can see the final process, a compact digital representation for playback is generally not available. Merely recording a video of the notes is not satisfactory, as this includes unncessary visual components such as the instructor's hand. Our system provides a compact representation of notes being written that removes clutter and reduces file size. The system records only the notes written on a piece of paper, removing the user's hand and other extraneous data, and displays them via a VGA port.

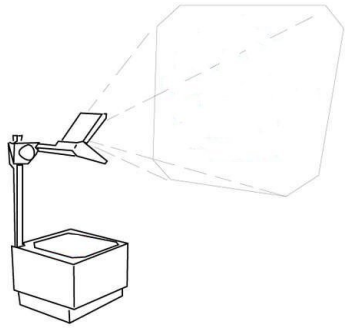

Figure 1: Projector[1]

The system is robust to movement of the paper and allows the user to write in multiple colours. To implement this, each pixel is classified in hardware as ink or not with a decision tree classifier. The classifier is designed to further exclude pixels having a skin colour. The output of the classifier is written to RAM, to be later read by the processor. The processor computes the time that each pixel has been classified as ink and uses this information to generate video output. The entire process is recorded for later playback.

[1]Image adapted from: http://www.instructables.com/id/How-to-Make-XXL-Street-Stencils-%26-Get-Away-WIth-It/step5/Scale-up-your-letters/

1.2 Block Diagram

A block diagram of the system is shown in Figure 2. The functionality of each block is described in later sections.

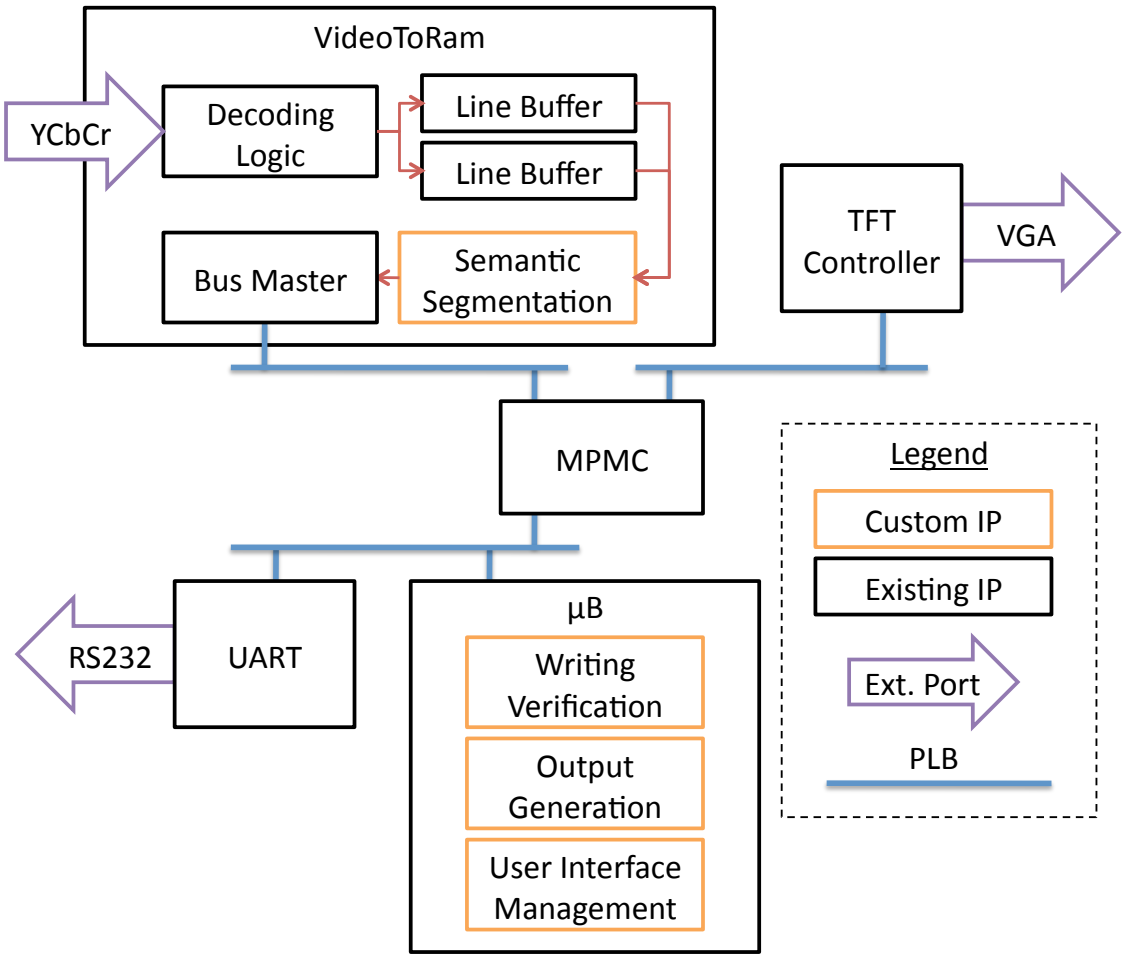

Figure 2: Block diagram of system showing internal and external connections

1.3 Brief Description of IP

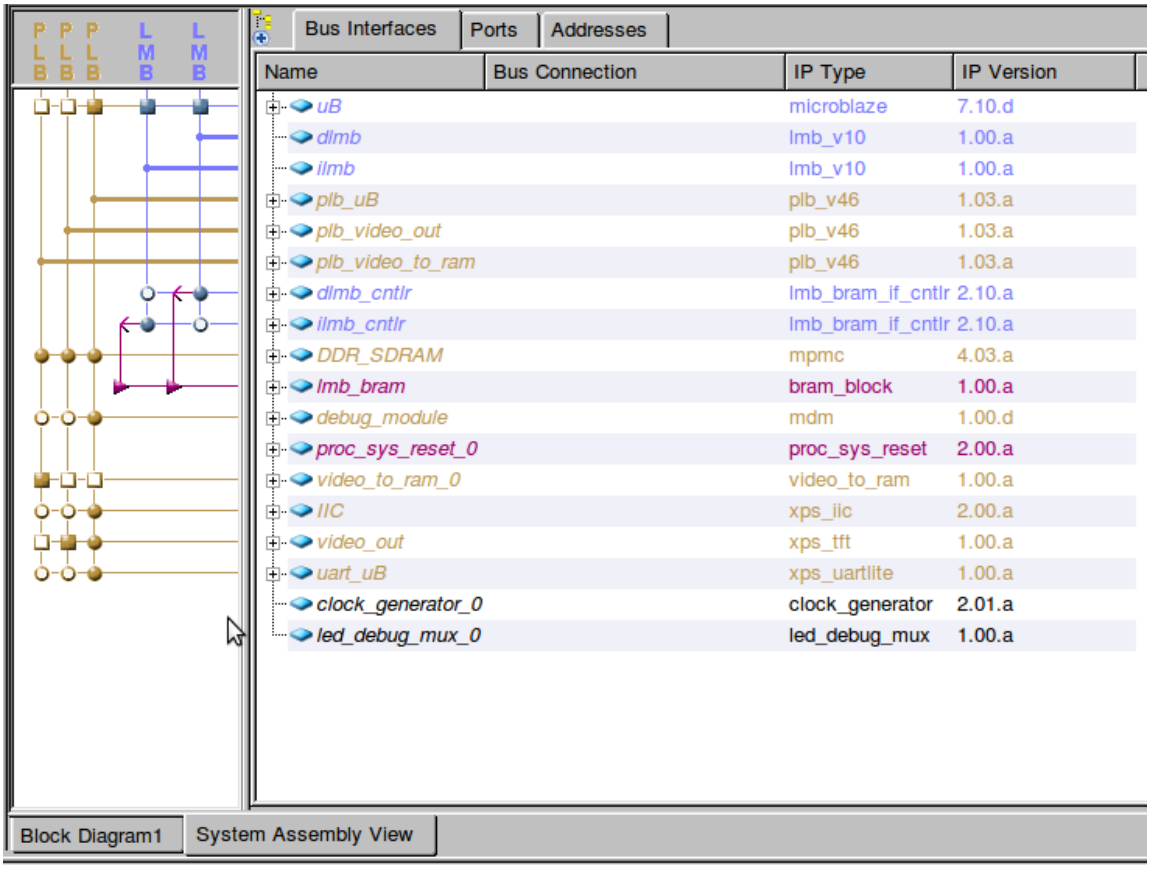

Figure 3: Overview of IP

1.3.1 Reference Design

As a basis for the video processing portion of our design, we used a reference design by Jeffrey Goeders posted on the 2011 discussion board on Blackboard for ECE532 [1]. In the reference design, the `video_to_ram` pcore reads in data from a video camera using the video decoder. It converts from YCbCr format to RGB format and sends pixels to RAM via the PLB bus. All control signals for timing is computed in `video_to_ram`. There is a second pcore (`video_out`) which reads pixels from RAM and generates signals necessary to drive the monitor via VGA. The video decoder is

[1] https://portal.utoronto.ca//E163B5E40E942455D203314EA00BDC27/courses/1/Winter-2011-ECE532H1-S-LEC0101/db/_2337128_1/Video_To_RAM_System.rar

configured using the processor via IIC. The code for configuring the video decoder is contained in `video_setup.c`.

We adapted the design for our project and modified modules contained in the `video_to_ram` pcore as well as the code in `video_setup.c`.

1.3.2 Processor Code

The primary source file is `video_setup.c`. Originally, this code was borrowed from the `Video_to_RAM` project by Jeffrey Goeders and only configured the video decoder. We added in additional functionality as follows:

- It receives data from the `video_to_ram` module, and based on the values given (either 1 or 0, for black or white), stores these in a buffer.
- When it determines that a pixel has been black for a period of time, it decides it corresponds to ink (and not to a transient shadow) and sets the pixel value accordingly.
- When it determines that a pixel has been white for a longer period of time, it decides that the ink has disappeared (and is not occluded by a hand) and removes it.
- It stores frames in a circular buffer, so that it is possible to replay a video of the notes being written.
- A menu allows the user to pause recording, replay the buffer, or change the colour of the ink.

1.3.3 Video to ram

This hardware module does the following things:

- It reads in data from the video camera (This functionality is based on the pcore `Video_To_RAM` by Jeffrey Goeders).
- This data is classified using the YCbCr colour data for each pixel. The hand-trained decision tree classifier first uses the Cr value to decide if a pixel belongs or does not belong to a hand. For the non-hand pixels, it then classifies them as ink or paper based on the Y value.
- It converts the decision made for each pixel to a RGB value (either all ones or all zeros, i.e. either black or white) and sends one bit per value per pixel to the processor.

1.3.4 Other IP

dlmb, ilmb, dlmb_cntlr, ilmb_cntlr: Used for communicating between the processor and the BRAM.

lmb_bram: The block RAM.

plb_uB: The master on this bus is the processor, which uses it to communicate with the `video_to_ram` module, the UART module and the SDRAM primarily. The iic and debug modules are also slaves on this bus.

plb_video_out: The TFT module, which communicates with the VGA, is the master on this bus, which uses it to communicate with the SDRAM. This is used to communicate between the processor and the tft module that communicates with the VGA.

plb_video_to_ram: This is used to communicate between the `video_to_ram` module and the SDRAM module, with the `video_to_ram` module being the master.

DDR_SDRAM: The main memory, where we store our data (frames, etc.)

debug_module: Used for debugging the MicroBlaze processor over JTAG.

proc_sys_reset_0: Allows external resetting via the reset button.

iic: Used to configure the video decoder.

video_out: Sends data to the VGA.

uart_uB: An `xps_uartlite` module. Allows the user to interact with the processor over USB.

clock_generator_0: Provides clocks.

led_debug_mux_0: Unused, put in as an initial test.

2 Outcome

2.1 Results

For this project, we were very successful in meeting all of our originally proposed goals, requirements, and acceptance criteria. Results are summarized in Tables 1, 2, and 3.

When designing the system, we realized that the acceptance criterion of 10 frames per second (Table 3) is undesirable. Since human writing speeds are slow compared to hardware, large amounts of memory and processing power would be required to keep track of the length of time that each

Features	**Status**
Record notes as users write them	Complete
Ignore user's hand and writing utensil	Complete
Optional features	**Status**
Allow user to erase previous text	Complete
Allow user to change colour of text	Complete
Record voice along with video	
Save data produced to a PC.	

Table 1: Status of initially proposed features.

Functional requirement	**Status**
Real time video input to FPGA board	Complete
Real time thesholding of pixel values to determine what is possible writing	Complete
Store past state of writing for several frames to determine what constitutes writing	Complete
Output buffer of stored writing to monitor	Complete
Functional requirements of optional features	**Status**
Use above real time thresholding of pixel values to detect white paper, indicating text has been erased (and is not merely occluded)	Complete
Change output colour of text based upon switches set by user	Complete
Record and store audio at a known sampling rate to synchronize with video	
Over USB, additionally save the buffer of stored writing to a PC.	

Table 2: Status of initially proposed functional requirements

Acceptance criteria	**Status**
System correctly differentiates between notes and non-notes in physical tests of the system	Complete
System is able to correctly recall position of notes on screen and display them	Complete
~~System is able to process a 320x200 frame at a rate of 10 frames/second~~ System is able to process a 640x480 frame at a rate of 2 frames/second *This improves performance. See discussion.*	Complete

Table 3: Status of initially proposed acceptance criteria

pixel has been classified as ink. Accordingly, reducing the frame rate allows the system to consider a larger period of time while still using a small amount of memory.

To reduce the frame rate, we implemented most of the core video processing logic, and in particular the history and replay logic, in software rather than hardware. This transition provided several benefits:

1. A larger time frame can be considered
2. More freedom in the way memory addresses are mapped
3. Easy to pause processing for menu interactions
4. Complete freedom in the way real-time results are debugged
5. Small compile time (5 seconds compared to 20 minutes) allowed rapid prototyping of features

The details of how the system functions is described in later sections.

2.2 Possible Further Improvements

A future extension to this project would be to allow users to adjust the decision tree classifier in software, allowing for training/calibration of the classifier at runtime instead of in a separate project. This could be implemented several ways, such as with fast simplex links, shared memory, or memory mapped device registers. The system is presently calibrated for a particular lighting environment and does not allow the user to change the calibration. This feature would greatly improve the robustness of the system.

Another possible extension would be a dynamic construction of the tree based on the luminance value (so that the user would only need to manually set thresholding to exclude their hand). An average brightness of the scene would be computed, and then the decision tree nodes would be assigned as a fraction of the average luminance. Once coded, some additional testing would be required to ensure that the classified pixels form a consistent fraction of the average luminance.

3 Description of the System Components

3.1 video_to_ram

`video_to_ram` contains all of the video processing logic in this project, including receiving YCbCr input from the video decoder, controlling timing of the video signals, classifying pixels as ink or not ink, and finally writing the classification results to RAM via the PLB.

3.1.1 Decision Tree Classifier

The decision tree for the project is shown in Figure 4. The text below describes our design process of obtaining the tree.

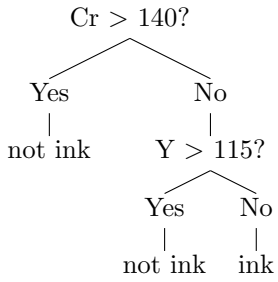

Figure 4: Final decision tree for system

We did some testing to figure out the optimal classification scheme. We'd originally planned to simply classify based on the Y (luminance) value, with all pixels below a given value being classified as ink, but we found it was hard to distinguish between a writing hand and ink. From a distance, even a thick black marker would not show up in the video camera as being black. Evidently, a more complex classification scheme was needed.

Based on our knowledge of how YCrCb works, we hypothesized that a decision tree might work well. We would first classify using the Y value, with high values being classified as paper, and low values as ink. The values in the middle would be further classified using the Cb value. Since we know, due to the use of blue and green screens, that these colours are considered to be easy to separate from human skin tones, we believed that it should be able to remove hands from the image due to their low Cb value. However, we did not have as good results as we expected.

We decided to systematically experiment with thresholding based on Y, Cb, and Cr separately in order to determine how best to go about constructing our decision tree. We created a new Xilinx project, a modified version of our main project, in order to do so. We first displayed the YCbCr values read by the camera directly to the VGA. The Y values were mapped to green, the Cb values to blue, and the Cr values to red. From this, we hypothesized that we could probably get better results by taking into account the Cb and Cr values. It was apparent that a writing hand, the paper, and the ink differed significantly in more than just the luminance value.

We then modified the code so we could display only one of the three values at a time, and so that the colour we choose to display could be selected by modifying and recompiling the software instead of the hardware, and ran tests on each individual value. The results of the tests are summarized in Appendix A.

Our final classifier first removes the user's hand, based on the Cr value (Figure 4). With careful adjustment of the thresholding value, it is possible to also remove shadows cast by the user's hand, as these often have a slight reddish tint. We then separate between ink and paper using the Y (luminance) value. We continue to use the testing project we created in order to more efficiently determine the thresholding value at each node of the decision tree, as we can then separate the effects of each node on the final display.

3.2 Processor

All of the functionality of the processor block is contained in `video_setup.c`, the main software component of the system. Algorithmically, the software behaves as follows:

1. Initialize of all data structures for video processing
2. Loop indefinitely:
    - Clip borders and fill in with background colour to deal with offsets in output. It was found that some visually unappealing noise appeared around the borders of the frame, likely due to errors in the YCbCr decoder.
    - Update pixel history and assign new output colours (section 3.2.3).
    - Update output to VGA controller (section 3.2.2).
    - Poll for user input to enable menu functionality (section 3.2.4).

3.2.1 Memory map

Figure 5 shows the addresses used by the processor for data storage. The decision tree classifier in the `video_to_ram` module writes to `0x40000000` - `0x40200000`.

3.2.2 VGA.h

The `VGA.h` header file is used to facilitate buffered access to the `xps-tft` module for outputting video. It was written by Andrew Shorten and taken from the project website[2]. The header defines the various structures and helper functions for screen buffering, modifying pixel rgb values, and handling double buffering. We use circular buffering instead of double buffering. The reasons for this improved system are described in section 3.2.5.

[2]http://www.eecg.toronto.edu/ pc/courses/432/2009/projects/tft-edk.txt

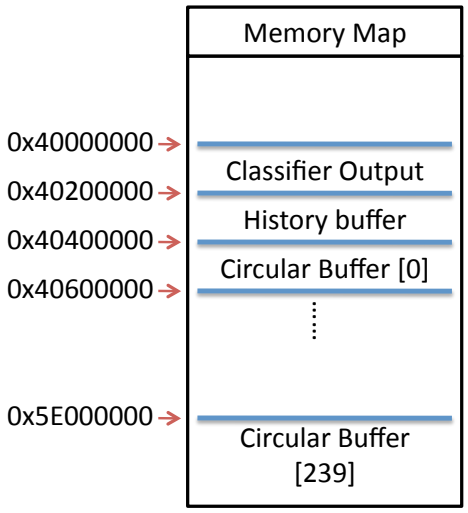

Figure 5: Memory map for SDRAM, accessed by both the processor and the `video_to_ram` module.

3.2.3 History

For each pixel, the processor reads the binary output from the decision classifier (starting at `0x40000000`) and appends it to the history buffer for each pixel (starting at `0x40200000`). Each pixel's history buffer is fixed to 32 bits, so the previous values are bit-shifted to make room (Equation 1).

$$\text{HISTORY-BUFFER}[x, y] = (\text{HISTORY-BUFFER}[x, y] << 1) \mid (\text{CLASSIFIER-OUTPUT}[x, y]) \quad (1)$$

Monitoring the recent history of a pixel is crucial in deciding whether or not it corresponds to actual writing that we want to record. The decision-tree classifier (section 3.1.1) is used to segment pixels that potentially correspond to ink on the page, but it is the history that verifies this through permanence. We make the assumption that any transients in the colour of a pixel will correspond to things that are not ink, such as the tip of a pen or a particularly dark shadow. If a pixel has remained dark for sufficiently long we decide that it is not transient and most likely corresponds to ink on the page.

In technical terms, we first set aside a screen buffer in memory to act as a 'history buffer'. This is identical to the structures used to buffer individual frames in memory except that instead of each word corresponding to the RGB values of a pixel it corresponds to the history of the colour of that pixel. The 32 bits correspond to the value of that pixel over the last 32 frames, with '0' meaning the pixel was dark in that frame. The algorithm for updating the history and verifying the content

of a pixel is as follows:

1. Load previous history word for the pixel
2. Bit-shift left and OR with the binary output of the classifier for the pixel in the current frame
3. Store the new history word in memory
4. *Switch to Background*: If the history equals `0xffffffff`, i.e. has been non-ink for 32 frames, then the background colour is written to the output monitor for display (Equation 2). This condition is met by pixels that were ink but have been erased, or by pixels that has never been classified as ink.

$$(\text{HISTORY-BUFFER}[x, y] = \texttt{0xffffffff}) \implies \text{OUTPUT}[x, y] = \text{BACKGROUND-COLOR} \tag{2}$$

5. *Switch to Foreground*: If the last 16 bits equals `0xffff`, i.e. has been ink for 16 frames, then the foreground colour is written to the output monitor for display (Equation 3). Additionally, we use a check to see if this pixel is being newly coloured (ie. it previously was the colour of the background). This supports writing in multiple colours as it causes previously coloured pixels to remain unchanged when the foreground colour is changed.

$$\begin{aligned} &((\text{HISTORY-BUFFER}[x, y] \ \&\ \texttt{0xffff}) = 0) \ \&\& \\ &\text{PREVIOUS-OUTPUT}[x, y] = \text{BACKGROUND-COLOR}) \\ &\implies \text{OUTPUT}[x, y] = \text{FOREGROUND-COLOR} \end{aligned} \tag{3}$$

3.2.4 Menu System

Users can interact with the system using the UartLite communication link to transmit data between the processor and the laptop over USB. The user menu system provides access to three major functionalities:

- **Change Foreground Colour**: Users can change the colour of the output foreground (the written text that is displayed) to one of Black, White, Red, Blue, Green, Yellow, Teal, or Purple. This foreground colour will be used for all subsequent writing on the page.
- **Playback recent history**: The history playback feature is discussed in section 3.2.5.
- **Reset**: The reset feature allows the user to reset all colours, buffers, and histories. This is a software equivalent of performing a manual reset of the board.

3.2.5 Replay

In order to implement the replay feature we needed to move away from the standard double-buffering method of storing and accessing image data in RAM. Data is still loaded into a single buffer location in memory from the camera but the location that the output frame is moved to (after history checking and colour assignment) is a circular buffer using the remainder of RAM. This allows us to store a large number of past frames (239 in total) that we can play back on demand to show the recent history of writing on the page. At each update of the circular buffer the start and end addresses are saved for later playback.

The play back functionality is accessed through the user menu. Starting from the first valid frame in the circular buffer (ie. the start) we iterate through the buffer to the last valid frame (end of the buffer). In between each iteration we sleep (spin in a loop) for a pre-specified amount of time in order to give time for the frame to be written to the `video_out` module as well as to meet a desired playback speed (chosen arbitrarily at compile time using the setting of a playback delay parameter).

3.3 Internal Communication and External Ports

3.3.1 Video input

We use the video decoder board to connect to a video camera, which connects directly to the `video_to_ram` module (without using a bus).

3.3.2 Internal Communication

The `video_to_ram` module, the processor and the `video_out` module all communicate by reading from and writing to memory. The `video_to_ram` module has its own bus, with which it writes to the SDRAM. A second bus is used for all processor communication: with it, the processor can access the SDRAM, the video_out module, and the UART.

3.3.3 Video output

A third bus is used to communicate between the `video_out` module and the SDRAM. The `video_out` module, a `xps_tft` module, is used to write to write to the VGA.

4 Description of Design Tree

As there are many files in our project folders, we have summarized only the key files below.

Design: main XPS project

- ↪ **pcores/video_to_ram_v1_00_a:** video-to-RAM module
  - ↪ **video_to_ram.v:** takes data from the camera, sends it to ycrcb2rgb, and then sends the classification decision to the processor
  - ↪ **ycrcb2rgb.v:** contains colour-conversion code, and classification code
- ↪ **sw:** software
  - ↪ **video_setup.c:** primary file for our C code (not just the video setup code)
  - ↪ **VGA.h:** data structures for dealing with frames

Calibrator: the XPS project used to calibrate the classification system. Nearly identical to the above project, with the following differences:

- ↪ **pcores/video_to_ram_v1_00_a:** video-to-RAM module
  - ↪ **ycrcb2rgb.v:** sends Y to G, Cr to R and Cb to B.
- ↪ **sw:** software
  - ↪ **video_setup.c:** various simple decision trees are set up here; select between them by commenting out the unwanted ones and recompiling

Video

Documentation: excluding individual reports

- ↪ **Group Report**
- ↪ **Slides**

Appendices

A Experimentation with Classification Schemes

The following tests were done only to determine the accuracy and usefulness of thresholding based on each of Y, Cr and Cb. Tests are displayed in chronological order, as the results varied with the height of the sun in the sky outside.

Cb Value	Notes
100	good - a bit blocky
125	doesn't work at all - entirely white
75	almost all black, the hand shows up to some extent
85	still has large black blotches on the paper
95	almost entirely black
125	almost entirely white
115	paper and hand show up as mostly white, but text shows up very blurry
100	hand shows up white, paper and ink both mostly black

Table 4: Split on Cb

There was about an hour break taken around the horizontal line. Note how the results at a given threshold value differ significantly before and after the line.

Cr Value	Notes
110	all black
200	all white
150	text somewhat black, paper somewhat white, unreliable
140	slightly worse result than 150
160	paper is black, text is black. Hand (and surrounding shadows!) is all white.
155	like 160, but better separation between hand and non-hand.

Table 5: Split on Cr

We also tried an experiment where we tested with a black background. Interestingly enough, the hand showed up as black, with a white outline around it. We did not, however, wind up using this phenomenon in the final design.

Choosing a Thresholding Algorithm

- If Cb < 100 (on a scale of 0 to 255), we can safely assume that we should classify the pixel as white. If Cb > 100, it should be classified as black. In between, no assumptions could be

Y Value	Notes
115	Too dark.Hand all black, some of page black.
100	adequate. Words a little faint, hand black.
80	hand invisible, text too faint.
120	words come out strong, maybe a little too strong in the corner (although corner text is blurry in the camera view too.
125	Seems good.

Table 6: Split on Y

made. These values could change substantially based on minor changes in lighting conditions (such as if the sun is lower in the sky, if the experiment is performed in a room with a window even if the camera is not in direct sunlight or near the window.) Evidently, using Cb is not particularly useful, given the small benefit given and the high precision of the thresholding values needed.

- If we split Cr at 160 then we can use the Cr value to distinguish between a hand and a non-hand. This seems to be accurate at about ± 10; if we can get this value precisely then shadows cast by a hand will also be classified as being red (as presumably some light reflects off the hand, giving the shadows a slight reddish hue).
- If Y is split at 125, we can more or less accurately determine between ink and non-ink. Some shadows cast by the hand will appear, but as long as the hand keeps writing their positions will change and they will not be classified as ink. This is accurate to roughly ± 15.

Based on the above experiments, we tried splitting based on Cr, then Cb, then Y. We also tried splitting based on Cr, then Y, and got much better results.

We did additional testing in the lab to determine which values to use with those lighting conditions, and we decided to split Cr at 140 and Y at 115.

B Previous design: all processing in hardware

Before we realized that attempting to process incoming video in realtime is counter-productive, we attempted to design a system that could perform *all* processing in realtime. As shown in Figure 6, this involved several hardware modules that would all read and write to different portions of memory. The first module, `VideoToRam` would be in charge of controlling the other modules. Each module would be on its own PLB bus to write to minimize bus conflicts, as all devices will be continuously writing in burst mode.

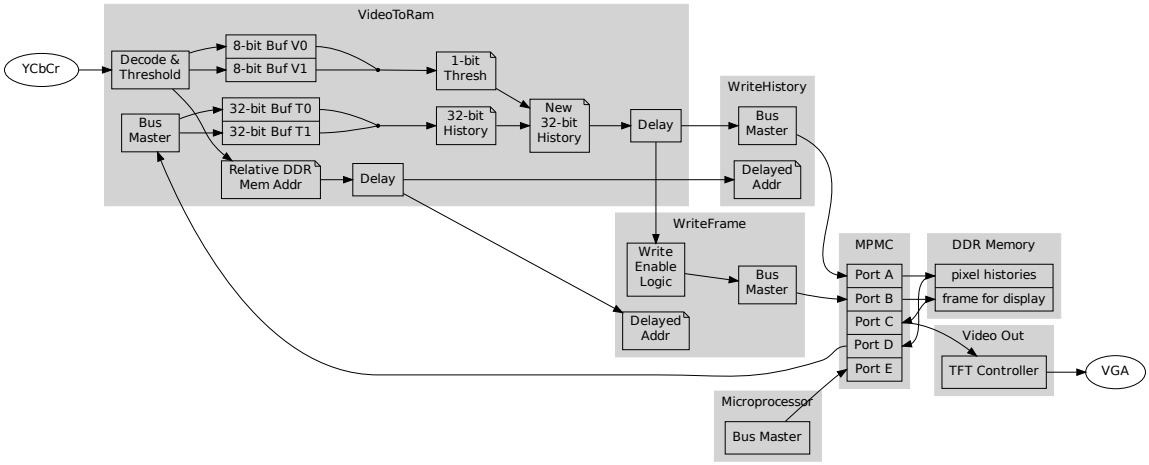

Figure 6: Design of system for processing entirely in hardware