Michael Chan, Siddhanta Shrestha

ECE 332: Lab Report 2

**Introduction**

The goal of this lab was to familiarize ourselves with programming the D5M camera using the Altera Monitor Program. The focus was primarily on learning the basis of image processing, achieved through the processing tasks set out for this lab. The objectives of this lab are:

* Understand the procedure involved in interfacing peripherals to an embedded system
* Understand the process of capturing image through a camera
* Learn to develop C programs for the ARM hard core processor using the Altera Monitor Program
* Understand basic image processing concepts
* Understand how images are displayed on a monitor

**Detailed Procedure**

The hardware setup for this lab was not as extensive as the previous lab. It required us to connect the DE1-SoC to an additional monitor through a VGA cable in order to output the display of the D5M camera module. The Qsys files were provided for us thus the hardware/connection setup for the board was not required. At the initial stage, the given C code allowed us to pause and start the camera’s display when pressing a button. We then added a timestamp with the number of pictures taken and the date and time the picture was taken. This would then be displayed on top of the picture. The time stamp was achieved by using the time.h library where we used the localtime() function. Upon completing this we then wanted to flip and mirror the image, the process for both being very similar. Flipping the image over the y-axis was achieved by traversing the image pixel by pixel and saving it into a temporary array then traversing that temporary array and saving to the original array. This functions very similar to a FILO stack where the first pixel in is the last pixel out. The same process allows us to mirror the image over the x-axis. The final two image processing tasks for this lab was turning the picture black and white and vice versa. The procedure for these two tasks were also similar since we simply needed to traverse the image pixel by pixel and dissect the individual RGB values of each pixel. If the sum of these values was above a certain threshold then we would turn it black/white if below then white/black.

**Hardware Changes**

There weren't any hardware changes to our system other than attaching the D5M Camera to the DE-SoC. All of the Qsys files and Verilog files were provided at the beginning of the lab. No change in pin assignments were made.

**Software Changes**

The initial source code that was provided allowed us to start and stop the display of an image on the monitor. We added many functions to this code to reduce the number of lines of code required. These functions include:

To achieve the timestamp we wrote a function that uses the imported time.h library that we imported where we used the function localtime() to get the current time in EST, save that into the time\_t variable t and then set a pointer to the address . Then we write that into the timetext so that we can show it on the display.

| // timestamp - Gets the current time in EST void timestamp(){  time\_t t = time(NULL); // sets up time for timestamp  struct tm \*tm = localtime(&t);  strftime(timetext, sizeof(timetext), "%c", tm); // string of timestamp } |
| --- |

Timestamp - Gets the current time in EST

Our printtext() function essentially gets the time stamp and count value and prints it to the screen exactly where we want to using the offset value. In our case we chose the bottom left for the screen. We chose to adjust the count to be displayed above the time and date for legibility reasons, however the process for displaying was the same (iterate through the display from left to right until it reaches the place where the values should be printed).

| // printtext - Prints time/count to VGA void printtext(char \*time\_ptr, char \*count\_ptr){  count++; // Keeps track of number of images captured  timestamp();  int offset = (55 << 7) + 5; // Max: Y=60, X=80 | The Offset is (000000 0000000)base2  while ( \*(time\_ptr)){  \*((char\*)0xC9000000+offset) = \*(time\_ptr);  ++time\_ptr;  ++offset;  }  snprintf(counttext, sizeof(counttext), "Count: %d", count);  offset = (53 << 7) + 5; // Count is printed above time and date  while ( \*(count\_ptr)){  \*((char\*)0xC9000000+offset) = \*(count\_ptr);  ++count\_ptr;  ++offset;  } } |
| --- |
|  |

Printtext - Prints time/count to VGA

The saveimage() function essentially iterates through pixel by pixel and saves it onto the 2D array.

| // saveimage - Saves image from VGA to input (temp array) void saveimage(volatile short \* Video\_Mem\_ptr){  int y,x;  for (y = 0; y < 240; y++) {  for (x = 0; x < 320; x++) {  input[y][x] = \*(Video\_Mem\_ptr + (y << 9) + x);  }  } } |
| --- |

Saveimage - Saves image from VGA to input (temp array)

Similar to the saveimage() but instead we are saving the 2D array to the address of the video\_Mem\_ptr pointer.

| // writeimage - writes image to screen from input (temp array) void writeimage(volatile short \* Video\_Mem\_ptr){  int y,x;  for (y = 0; y < 240; y++) {  for (x = 0; x < 320; x++) {  \*(Video\_Mem\_ptr + (y << 9) + x) = input[y][x];  }  } } |
| --- |

Writeimage - Writes image to screen from input (temp array)

Flip and mirror work very similar to one another, only difference is the x and y axis are essentially “flipped” where one is going over the x axis while the other is going over the y axis. The way we did this is similar to a FILO stack implementation where we save pixels on half the side we are working on then save the other half on top, then essentially “pop” the images so that the first pixels in are the last out.

| // flip - flips the image along Y-axis void flip(){  int i, j;  long rd2;  rd2 = rows/2;  for(i=0; i<rd2; i++){  for(j=0; j<cols; j++){  tempflip[rows-1-i][j] = input[i][j];  }  }  for(i=rd2; i<rows; i++){  for(j=0; j<cols; j++){  tempflip[rows-1-i][j] = input[i][j];  }  }  for(i=0; i<rows; i++){  for(j=0; j<cols; j++){  input[i][j] = tempflip[i][j];  }  }  } |
| --- |

Flip - Flips the image along Y-axis

| // mirror - mirrors the image along X-axis void mirror(){  int i, j;  long cl2;  cl2 = cols/2;  for(i=0; i<rows; i++){  for(j=0; j<cl2; j++){  tempflip[i][cols-1-j] = input[i][j];  }   }  for(i=0; i<rows; i++){  for(j=cl2; j<cols; j++){  tempflip[i][cols-1-j] = input[i][j];  }  }  for(i=0; i<rows; i++){  for(j=0; j<cols; j++){  input[i][j] = tempflip[i][j];  }  } } |
| --- |

Mirror - Mirrors the image along the X-axis

Again the black\_white() and invert() are essentially the same, the only difference being that the sum of the RGB values being above a certain threshold results in the pixels either turning black or white. We needed the R, G, B values of each pixel to understand how much the pixel comprises of each of those colors.

| // black\_white - Converts image to black and white void black\_white(volatile short \* Video\_Mem\_ptr){  int y,x,R,G,B,sum;  int threshold = 30;  short temp;   for (y = 0; y < 240; y++) {  for (x = 0; x < 320; x++) {  Video\_Mem\_ptr = FPGA\_ONCHIP\_BASE | (y <<10)| (x <<1);  temp = \*(short \*) Video\_Mem\_ptr;  R = (temp >> 11) & 0x1F;  G = (temp >> 5) & 0x3F;  B = temp & 0x1F;  sum = R + G + B;   if (sum >= threshold){  \*(short \*) Video\_Mem\_ptr = 65535;  }  else{  \*(short \*) Video\_Mem\_ptr = 0;  }  }  }  writeimage(Video\_Mem\_ptr); // write image } |
| --- |

Black\_white - Converts image to black and white

| // invert - Inverts image void invert(volatile short \* Video\_Mem\_ptr){  int y,x,R,G,B,sum;  int threshold = 30;  short temp;  for (y = 0; y < 240; y++) {  for (x = 0; x < 320; x++) {  Video\_Mem\_ptr = FPGA\_ONCHIP\_BASE | (y <<10)| (x <<1);  temp = \*(short \*) Video\_Mem\_ptr;  R = (temp >> 11) & 0x1F;  G = (temp >> 5) & 0x3F;  B = temp & 0x1F;  sum = R + G + B;   if (sum < threshold){  \*(short \*) Video\_Mem\_ptr = 65535;  }  else{  \*(short \*) Video\_Mem\_ptr = 0;  }  }  }  writeimage(Video\_Mem\_ptr); // write image } |
| --- |

Invert - Inverts the black and white of a image

**Problems Encountered and Solutions**

There were no serious problems we encountered for this lab. The difficulty for us was in learning how to output our task onto the display. We were both confused about using pointers since we haven’t had much experience working with them. This was easily fixed however by looking through the provided worksheets and working through some of the examples in the manual.

The first problem we came across started with the printing of Time and Date on the monitor. Anytime we printed the time, it would display the time in UTC. This problem occurs even when using the localtime() function from the time.h library. We solved this problem by establishing a new environment and setting the timezone at the beginning of our program using:

| setenv("TZ", "EST5EDT", 1); // Sets Timezone to EST |
| --- |

Another problem we found was determining how many bits to shift for inverting an image when it comes to the pixel\_address.. Many attempts resulted in a completely black or white screen. We noticed afterward that the number of bits that we needed to shift was not the same for Red, Green, and Blue. According to the manual, it was 5 bits for Red and blue, and 6 bits for Green.

**Results and Conclusions**

We were ultimately able to achieve all of our objectives for this lab since we were able to do all of the image processing tasks that were required for this lab. This was a really intriguing experience for us since we were able to learn about image processing as well as brush up on pointers and type casting which we haven’t had experience implementing in a project environment.