Tom and Jerry

A cartoon game

List of Authors (Michael Harris, Eric Yeh)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: mharris@oakland.edu, epyeh@oakland.edu

***Abstract*— Tom and Jerry is a simple game based off of the popular cartoon of the same name where the goal of the game is simply to have tom the cat capture jerry the mouse by overlapping the images. The purpose of the game is to help us understand the vga and the block memory.**

I. INTRODUCTION

For this project, we decided to do a simple game because it would be fun and simple. It was something that could show our understanding of the course by using VGA. It's not as complicated as other projects we attempted when we understood how the VGA hardware is working.

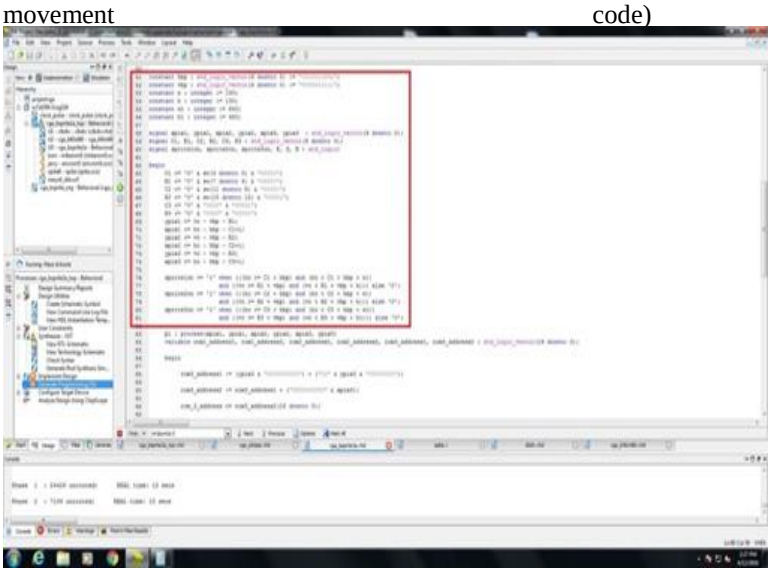
II. METHODOLOGY

Our project worked nicely during the demonstration although it was very time consuming when we build and troubleshoot it.

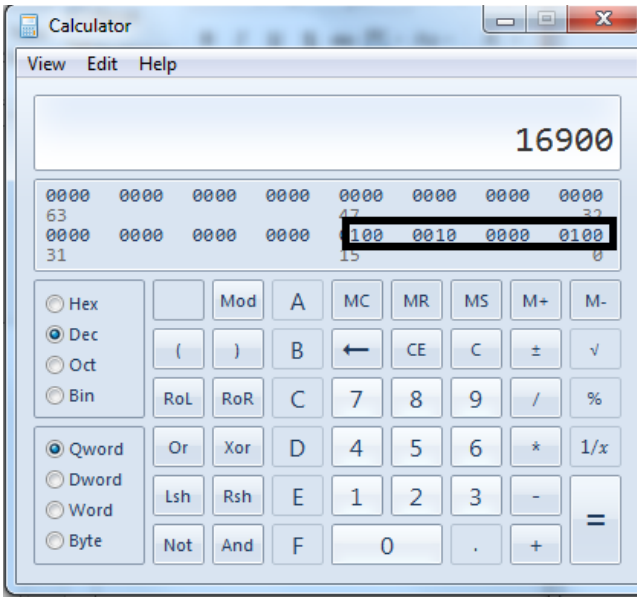
A. First Section

For the most part, the code we made along the guide lines of the code in the text book. Here, we have four components; a clock divider, a vga 640x480, a vga bsprite, and a two character ROM and a background ROM. 2 inputs, button(3), 8 switches, and a 50 mhz clock. The button acts as a reset. The clock keeps things on track and the switches control the characters. There are 5 outputs, 3 of which are the pixels where its red, green, and blue. And each color was originally 3 bits, 3 bits, and 2 bits respectively, but we found that it was more appropriate to make them all 4 bits. The last 2 outputs are horizontal and vertical sync's. inside the program, the clock goes into the character and the vga 640x480. The vga 640x480 sends 2 signals to the vga_bsprite that are horizontal and vertical counters, which are responsible to start the count from the top left of the screen and increase till it reach the bottom right of the screen. Also this will define the front porch and the back porch. After that we have the VGA screen defined so we can place the characters in their x and y locations. These locations can vary since their locations is dependent on two variables C and R that define x and y location of the character. C and R variables are cascaded with the switches so whenever you change the switches you are changing the x and y coordinates that changes the position of the characters.

(please see the RED box in the image below for character movement code)

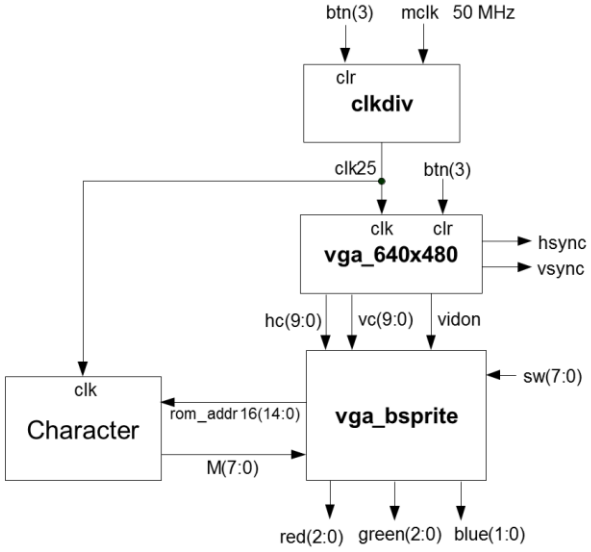


Most of the issues we have encountered were syntax errors but there were a few troublesome issues maintaining the character images as well as the background. For example, we would be able to "get" an image but the image details were so scrambled. We found out the error and corrected it through the ROM address. To do the background, we had to use a different bit size address. Since the character sprites were being overlapped by the background rather than the other way to take care of the address as well. To help solve this, we used the standard windows programing calculator found on in start menu using its programming/binary mode to take care of the 480x640 background image. Below is an example:



B. *Second Section*

The guide lines mentioned in section one looked like this. On a fundamental level, its really simple using only 4 blocks, 3 inputs, and 5 outputs and 5 signals



III. Experimental Setup

For the software used to make it, we used ISE software to make the character movements, And Matlab to generate a codded coefficient (.COE) file to generate a single port block memory to build the character images. To assist with the coe generator, we used a code from the textbook Digital Design Using Digilent FPGA Boards - VHDL/Active-HDL Edition
3rd Edition
Richard E. Haskell and Darrin M. Hanna

The Following Matlab Code will turn any image from any bit color into a 12 bit color regardless of size.

```
% Read the image
img = imread(imgfile);
h = size(img, 1); w = size(img, 2);
% Open the .coe file
s = fopen(outfile,'W');
% Print header
fprintf(s,'%s\n',''; VGA Memory Map ');
fprintf(s,'%s\n',''; .COE file with hex coefficients ');
fprintf(s,''; Height: %d, Width: %d\n\n', h, w);
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');
% Convert color channels to binary
R = dec2bin(img(:,:,1)',8);
G = dec2bin(img(:,:,2)',8);
B = dec2bin(img(:,:,3)',8);
% Stitch together the output words
out = bin2dec([ R(:,1:4) G(:,1:4) B(:,1:4) ]);
img2 = img;
for i=1:h-1
  sol = i*w-w+1; % Start of line
  eol = i*w;    % End of line
  % Print out words
  fprintf(s,'%03X,',out(sol:eol,:));
  fprintf(s,'\n');
  % Save new image
  img2(i,:,1) = bin2dec(R(sol:eol,1:4))'*2^4';
  img2(i,:,2) = bin2dec(G(sol:eol,1:4))'*2^4';
  img2(i,:,3) = bin2dec(B(sol:eol,1:4))'*2^4';
end
% Print out the last row
fprintf(s,'%02X,',out(h*w-w+1:end-1,:));
fprintf(s,'%02X;',out(end,:));
img2(h,:,1) = bin2dec(R(h*w-w+1:end,1:4))'*2^4';
img2(h,:,2) = bin2dec(G(h*w-w+1:end,1:4))'*2^4';
img2(h,:,3) = bin2dec(B(h*w-w+1:end,1:4))'*2^4';
% Close the .coe file
fclose(s);
```

Results

We had a very nice simple game is kind of a challenge to figure out the binary 1/0 that can get Tom closer to Jerry for the first player and Jerry away from Tom for the second Player. For the intent of the game, the terms for victory is usually determined by the players, but context wise as a cat and mouse game, it's when tom the cat "captures" jerry the mouse. This is usually achieved by overlapping the images of the cat over the mouse. Below are few pictures of the game.



Conclusions

As an ideally fully operational game, there are a lot of things that can be done to improve the function. Asides from improving the sprite art, there is actually putting in a story line, change the player interface from basic switches to using a joy stick or d pad, make the character movement more fluid. The story line would be of no issue due to the nature of the series we're using in which it's a simple cartoon. Although this is mostly superfluous in regards to the course itself. The original or main issues that we encountered were worked out as evident on how it was able to work in the end. What the project did to help us learn was establishing a more solid foundation for vga's and the nexus board. We Learned more about Single Port Block Memory and coded coefficient files and the art of creating a sprite image using vga with basic software which is paint and to use windows basic programs like calculator to figure out addresses size and figure out how the vga screen works (top left to bottom right). Remaining issues to be solved is to stretch the background toward the entire screen without overworking the fpga. It would be nice to use the SRAM on the nexus instead of the ROM fpga memory so we can make the characters move in a video it would be awesome.

References

Digital Design Using Digilent FPGA Boards - VHDL/Active-HDL Edition
3rd Edition
Richard E. Haskell and Darrin M. Hanna