ECE 554 FPGA

Display of BMPs & Fonts on VGA

Description

- This exercise will use the following peripherals of the FPGA board:

1. VGA Output
2. PLL

- It will also introduce a perl program that allows you to display images from any 24-bit color BMPs

- You will extend the base project to include your 552 processor and animate some images and text.

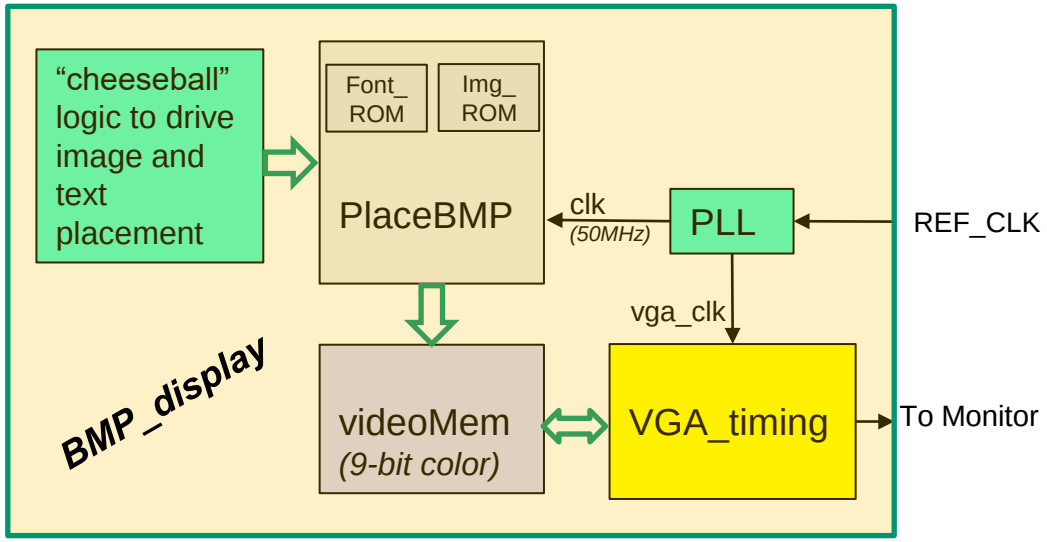
Compile and download base project.

- A .zip file called: BMP_display.zip is available.
- Download, unzip, and compile in Quartus
- Download to a DE1-SOC board with a VGA monitor connected and see what it does
- Study the code. See if you can figure out the basic sub-components and how they work. Particularly **PlaceBMP.sv** and how it is connected/driven by top level.

Project Block Diagram

The project drives a 640x480 VGA display from the contents of a video memory that stores 9-bits per pixel (3R,3G,3B).

The video memory is written by a block called ***PlaceBMP***. This block contains ROMs that have image data stored in a 9-bit format that can be loaded into the ROMs using $readmemh.



The ***PlaceBMP*** is also capable of displaying characters of a fixed 16pt New Courier font. The characters available are: 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ =>,()
The images, text, and the corresponding locations on the screen to display them are driven by some cheesy logic that drives the inputs of ***PlaceBMP***. One of your jobs in completing this exercise will be to replace that cheesy logic by making ***PlaceBMP*** a memory mapped component and driving it with your 552 uProc. The definitions, address locations, and bit mappings of the memory mapped registers are up to you. I used 3 registers, one for XLOC, one for YLOC, and one that contained a concatenation of the control signals used in the original PlaceBMP. **NOTE:** Due to memory constraints you will have to reduce your instr_mem to 4k and your data_mem to 2k to fit into the FPGA

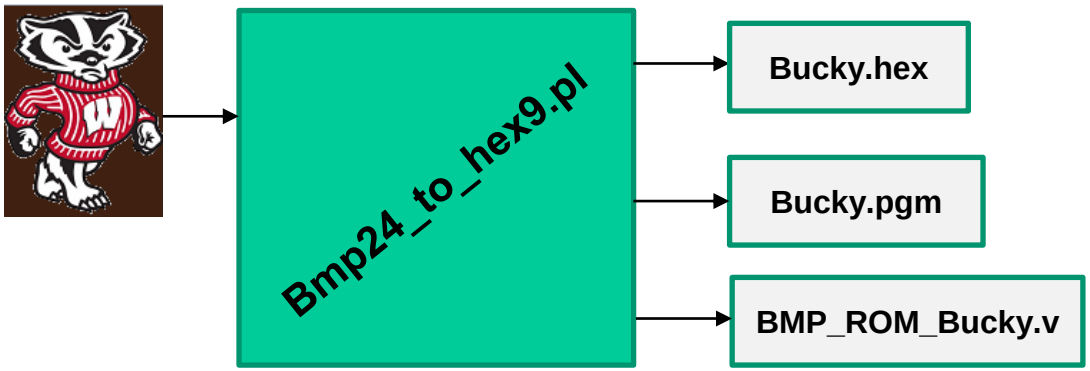
Requirements:

- Create a new image to display and add it to ***PlaceBMP*** (see next slide)
- Create some memory mapped wrapper logic around PlaceBMP, or modify PlaceBMP directly so you can drive its controls from your 552 uProc.
  NOTE: *memory footprints of instr_mem & data_mem need to be reduced*
- Write ASM code to drive your memory mapped logic to make all 3 images (Bucky, Mario, and the new image you added) appear on the screen
- Also use the Font ROM to render the initials of all your team members on the screen
- Demo your project/code to either Tananun or Eric
- Submit the following files: modified **PlaceBMP.sv**, (or wrapper logic), .ASM code your processor ran. Brief 1 page document outlining team members contributions to the exercise.

24-bit Color BMP to 9-bit .hex Generation

Don't use too large of a BMP, remember our screen resolution is only 640x480 so a 100x150 image covers a good portion of it.

The .pgm (portable grep map) file can be used to "sanity check" the image. There are online .pgm viewers. You are just checking the image is not skewed. If it is, adjust the dimensions of the incoming .bmp file. It likes even dimensions in the x.



Usage:
system_prompt> perl bmp24_to_hex9.pl Bucky.bmp

If you use the very specific color of RGB = 64,32,16 (a very dark brown) then that will be mapped to transparent. Can work nice for background of an image.

The program not only outputs the .hex of the image, but also a verilog file of a ROM that uses the .hex. Instantiate that ROM into ***PlaceBMP***. You will need to expand the "read mux" inside ***PlaceBMP*** to accommodate any new images you add.