Final Project Proposal Prannay Budhraja and Mary Knapp

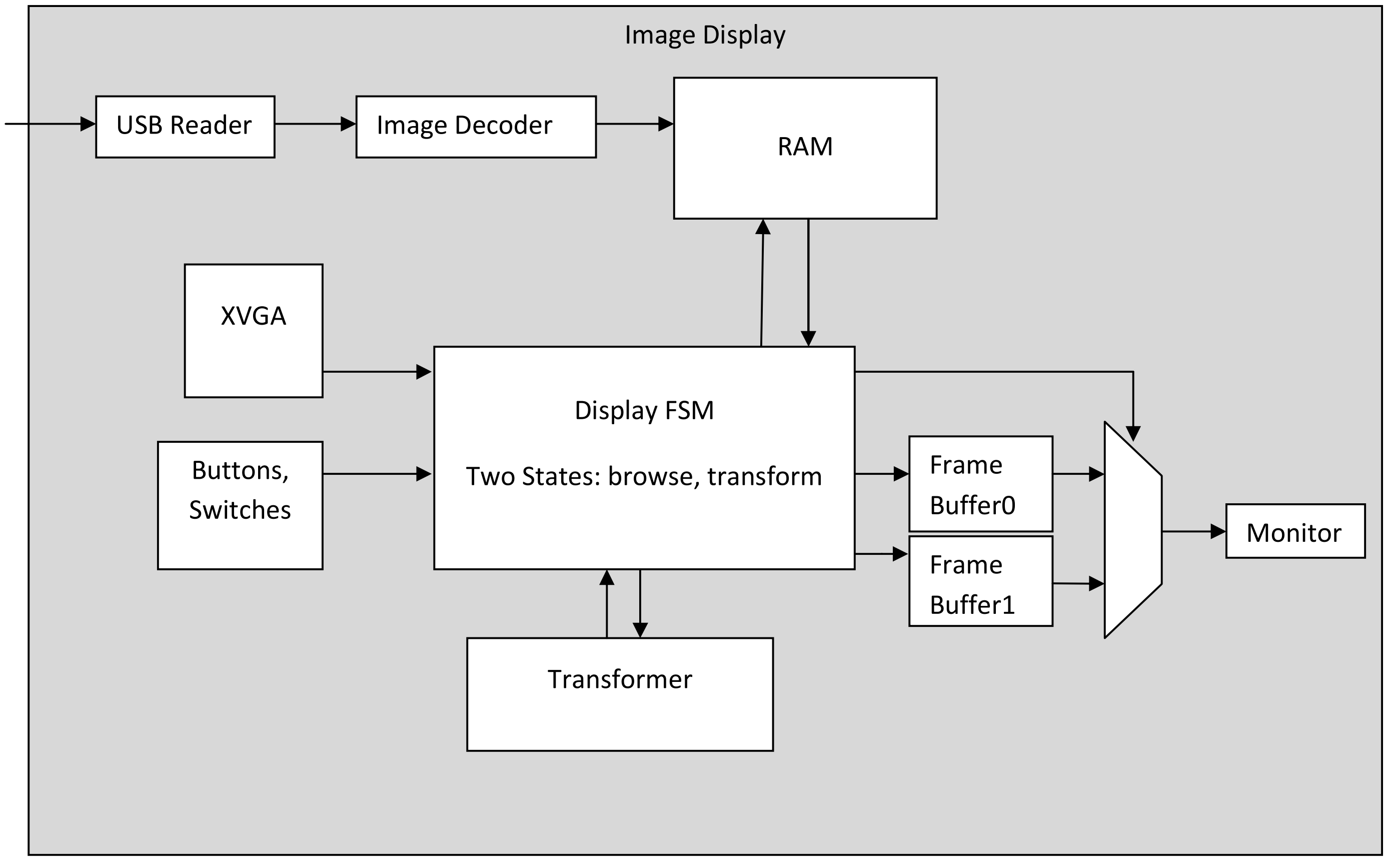
Image Browser and Editor

Introduction

We propose to create a system that allows a user to display and manipulate digital images stored in a portable FLASH memory device. The actual image data will be read and preprocessed by a computer and then made available to the FPGA via a USB cable. These images will be displayed on XGA monitor as a "film strip" that the user can scroll left or right to find a specific image. The user will then be able to select that image for scaling and/or rotation. The user will interact with the system via the buttons and switches on the labkit. If this baseline functionality is achieved, gesture recognition capability may be added through the use of a video camera.

Block Diagram

The draft block diagram below illustrates the flow of information between the modules that will be used to implement this interface.



Preprocessing
To successfully read and store images on the labkit, the images must be sufficiently small to fit on the labkit's memory and in a format that can be sent to the monitor (RGB). The FPGA has 5MB(40mbit) of available memory, and we need to store at least images in RAM at a time. To meet these requirements, we are converting all images to .bmp format in 256 colors. These images will then be processed into RGB values using a MATLAB script (credit: Gim Hom). The output of the script is a list of RGB values that can be sent to the FPGA via a USB cable and USB adapter. The XGA display requires 24 bits of RGB, so the image will be processed onboard the FPGA to convert it from 8-bit RGB (256 color) to 24-bit RGB using a set of LUTs generated by the MATLAB script. So the largest image size (1024x768 @ 8bit/pixel) requires 6.29mb for pixels and (256*24) bits for the LUT. Four of those will take approximately 25.2mbits which can fit in the available BRAM on the FPGA.

Modules
We propose a modular design, separating key indivisible functions into separate modules. The following is a description of each module.

USB Reader
The USB adapter has a connector for a USB cable and plugs into the labkit breadboard. The adapter has 24 pins with 8 for data. Data from the adapter will be sent to the in/out user pins on the labkit. From there, the data will go through a processing module before being stored in the labkit's BRAM. The incoming data will likely need to be sent through a synchronization module before it can be read into memory.

Image Decoder/Processor
This module will determine where in memory to write incoming data from the USB adapter. It will take the 8-bit input from the USB adapter, determine the appropriate memory address and then output the data to RAM. Each image will actually consist of 4 parts: an array of 8-bit pixels and three (RGB) look-up tables which will be used to convert the 8-bit pixel color to a 24-bit RGB value that can be displayed by the monitor. Choosing the location for each image and its associated LUTs is important for the browse function (described below) since slices of each image will be called and displayed sequentially. To test this module, image files and LUT files from MATLAB will be loaded directly into ISE (bypassing transfer through USB cable/adapter). The data in these files will then be run through the logic which converts the image back to 24-bit RGB and sent to the monitor. These tests may use the BRAM module for simplicity. In order to test our ability to write to and read from the ZBT memory, we will attempt to write color bar patterns into memory and then read them out to the display.

Display FSM
The way in which images will be displayed and modified will be determined by this module. It will have two states: browse and transform. Since this is the central module for this system, both members of the team will contribute to its design. Mary will be responsible for output to the monitor for all states as well as browsing and selection functionality. Prannay will be responsible for the implementation of scaling and rotation in the modify state.

This module will take input from the labkit input devices, the memories (BRAM), and the xvga module. Its output will be a pixel value per clock cycle that will be written to the current frame buffer.

User input will determine the state of this module. In browse mode, the user will be able to scroll through a “filmstrip” of images. As the user scrolls, images will appear from one side of the screen, move across the screen until the entire image is visible, then disappear at the other side of the screen. Pieces of several images will be visible at any given time. The images will be called from RAM and copied to the frame buffer. In transform mode, only the selected image will be visible and will fill the majority of the screen. A selected image is ready to be scaled or rotated. User input from the switches will determine which action is performed.

Frame Buffers

Two frame buffers will be used to display images on the monitor. One will contain the frame currently being displayed on the monitor and the other will contain the next frame, which will be in the process of being created. This allows 1/60th of a second for each frame to be generated. Since we will be using the 65MHz video clock (as in Lab 5), there will be over one million clock cycles between each frame for calculations.

Transformer

This module receives an array of pixels for the image and two parameters for scaling and rotation. The scaling parameter is signed to represent zoom-in or zoom-out actions, whereas the rotation parameter is the unsigned angle in radians specifying the desired rotation about the image center. The module outputs the visible region of transformed image.

Scaling will be done through a bilinear anti-aliasing and interpolation. Rotation will have a flexible implementation that can be increased in complexity depending on performance characteristics. Three hierarchical algorithms are proposed.

1. Simple Rotation: This algorithm determines the output pixel by calculating all the input pixels with that rotation destination and picking the nearest one.
2. Bilinear Interpolation: This builds on top the simple rotation by interpolating those input pixels to get the output pixel.
3. Area Mapped Interpolation: This assigns weights to each pixel contributing to the interpolation based on the fraction of area of the output pixel it covers using sub-pixel area calculation.

Possible optimizations include pipelining the three separate R G B channels to compute in parallel and merging in the end. This decreases the size of each operand and increases throughput. To test this module sample input data will be used with preset parameters and the output will be compared against output generated by a MATLAB script for that input data. This allows testing of this module independently from the rest of the system.

The algorithms themselves can be first tested on a computer in Matlab and then implemented in Verilog. The user interface for the browse and transform modes will need to be directly tested on the FPGA using a monitor, the logic analyzer and the hex display.