FPGA Passport

6.111 Project Proposal

Diana Wofk and Lorenzo Vigano

1 November 2016

1 Overview

The modern smartphone has made taking pictures extremely commonplace and easy to do. And with the invention of social media networks like Facebook, Snapchat, and Instagram, it is very simple to share photos of yourself in amazing places doing incredible things. The majority of everyday life, however, is less amazing and more mundane. Our motivation for this project was to turn the mundane into the amazing.

The FPGA Passport will transport its user across the world by editing their face/body into exciting locations. Our project will assume that you are standing in front of a green screen. The FPGA will continuously take in data from a camera pointing at you and display the camera input on a monitor. When you decide you like how you look, you will be able to press a "shutter" button to take a picture of what is currently being displayed on the monitor. Afterwards, the green screen background will be removed, and you will have the option to choose from a number of stored landmark backgrounds. Using chroma-keying, we will overlay your face on top of your chosen background.

You will then have the option of adding text and graphics to the image. Do you fancy yourself to be a king or queen? Try on the crown graphic and maybe even add some crazy facial hair! Have you always wanted to traverse through a rain-forest? We have the perfect adventure-hat for you to try on. By estimating where your face is, the FPGA Passport application will be able to place a crown on your head or gear you up for a tropical adventure. And if the placement is not quite right? Hat too low? Mustache too high? You will have the option of fine tuning the location of these overlayed graphics via button inputs. The composited image along with the optional text and graphics will be shown on the monitor.

When you are satisfied with the image, you will be able to press a "save" button that will send the image data to a PC in uncompressed bitmap format. You will then be able to view your "selfie" as a bitmap image on the PC, so that you, too, can brag about your travels to all of your friends.

2 Design

2.1 Goals & Scope

2.1.1 Baseline Goals

(1) Background images stored on external SD card; the FPGA is able to read these images out of SD card memory and display them on the monitor

(2) Camera input to the Nexys4DDR

  (a) Camera pointing at user standing in front of green screen

  (b) Live camera feed displayed on monitor via VGA output

(3) User poses and then presses a button to freeze the pose

  (a) Latest video input frame remains stored in a FIFO buffer

  (b) Processing begins: green screen removed and replaced by user-selected background (user selects background via input switches)

  (c) Composited image shown on monitor

  (d) User presses a button to proceed with image editing (see expected goals)

(4) After editing the image, the user presses a button to initiate image data transfer to PC; image is transferred in uncompressed bitmap format

2.1.2 Expected Goals

(1) User will be able to toggle a switch to add text to the composited image

  (a) Default text will be "Greetings from ___" – these will be pre-programmed and will correspond to user-selected backgrounds; for example, if there is an image of the Eiffel Tower in the background, the blank will say "Paris"

  (b) Text will be white on an semi-transparent (alpha blended) black bar, similar to what one may see in Snapchat pictures

  (c) User will be able to move the text bar up and down

(2) User will be able to toggle a switch to add graphics to the composited image

  (a) Example graphics

    i. Crown

    ii. Mustache

    iii. Adventure Hat

    iv. Sunglasses

(b) Graphics will be approximately fitted to the users face

(c) Fine tuning available through the use of buttons

(3) User will be able to toggle a switch to add filter effects to the output image

(a) Sepia filter will result in a vintage effect

(b) Soft glow filter will result in a dreamy effect

2.1.3 Stretch Goals

(1) User will be able to add custom text to the composited image

- Custom text (up to a max number of characters in length) will be sent as a series of character bytes from the PC to the FPGA
- Corresponding text will be generated and overlayed on the image

(2) User will be able to select an additional cartoon filter that will apply a cartoon-style effect to the output image

(3) User will be able to enhance the composited image

- User will be able to change the saturation and brightness of the image
- Enhancements will be made visible on the monitor in real-time

2.2 Block Diagram

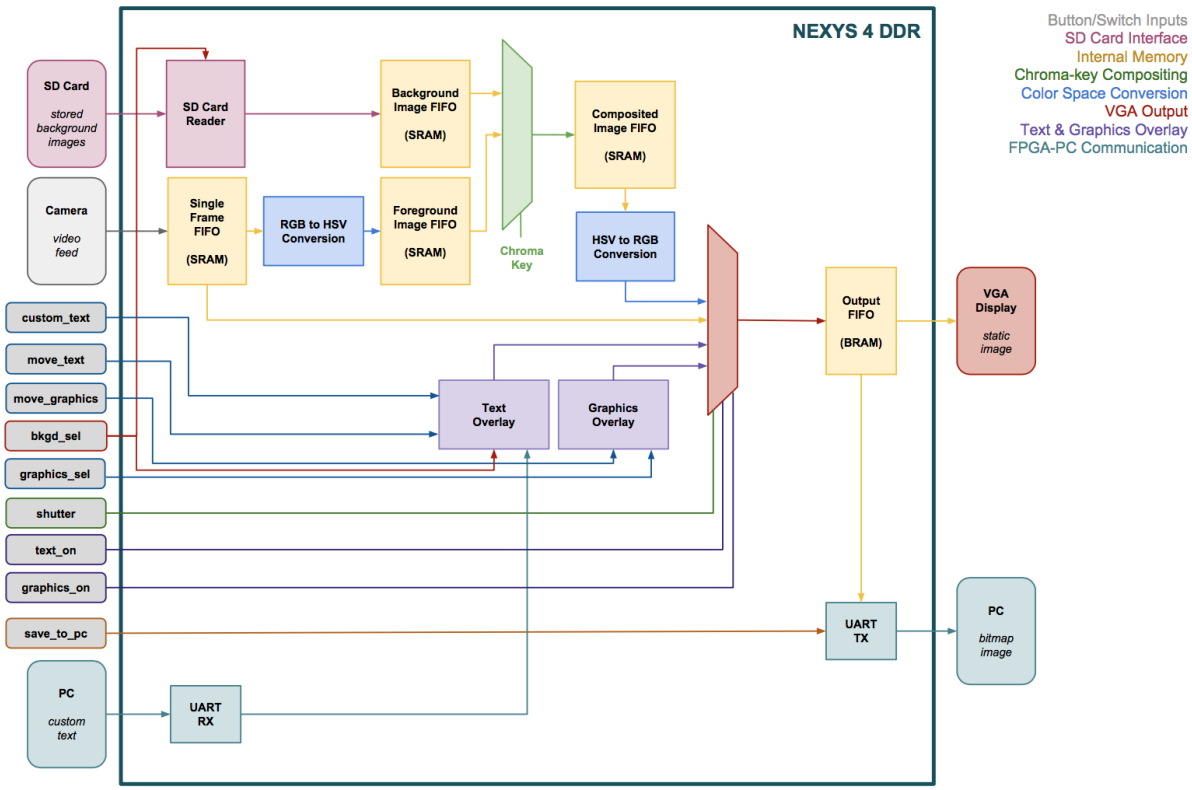

Figure 1: High Level Overview

3 Implementation and testing

3.1 SD Card Interface

The Nexys4DDR board has a micro SD card connector. Due to the limited amount of memory on the FPGA itself, we decided that it would be best to store background images on an external SD card. The `sd_card_reader` module will interface with the micro SD card connector; it will read out background image data from the SD card memory and pass the image data into a buffer. The user will be able to select which background image is loaded into the buffer via switch inputs.

Testing of this interface will be most easily done after the VGA is implemented. We will simply load a picture on the the SD card and attempt to display it on the screen. If we can do this, then we are capable of reading off of it and incorporating its data.

3.2 Camera Input

This module will utilize code provided by the TAs and will interface with the digital cameras available in lab. Depending on what color space the input is in (e.g. RGB), we will convert the pixels to the color space in which it may be easier to carry out chroma key detection. Color spaces are elaborated on in section 3.4.

3.3 Internal Memory

The Nexys4DDR board has 128 MiB DDR2 SDRAM memory as well as 4,860 Kbits of fast BRAM. This generous amount of internal memory will be very important to us as we plan to use five FIFO buffers within our system.

The `memory_controller` module will handle all memory block instantiations as well as read/write operations to the memory.

Testing of the memory will be critical. It is imperative that we are able to accurately write and read from memory. We will test memory by loading in various random data and then reading from it if it works then we were able to implement it correctly. Timing will be another thing to test, especially with the finicky DRAM which will be need to written to in bursts.

3.4 Color Spaces & Conversion

There are several ways to represent color; commonly used color spaces include:

- RGB - in which red (R), green (G), and blue (B) light are added together to produce various colors
- YCrCb - in which colors are represented as brightness and two color difference components; Y is the luma (brightness) component, Cb is the blue-difference (B-Y) component, and Cr is the red-difference (R-Y) component
- HSV - in which colors are represented by hue (H), saturation (S), and value (V)

The HSV color space is ideal for chroma-keying as it would allow for easier thresholding (the hue could be set to the chroma key, which in our case is the color green, and the threshold for keying could then be adjusted using saturation and value). This would result in more accurate compositing, since slight deviations in the brightness of the green screen backdrop (e.g. due to lighting conditions, shadows, etc) would not compromise the quality of the composited image.

Most graphics, however, use RGB. In order for chroma-key compositing to be done in the HSV color space, preprocessing of background images will thus require converting the image pixels' RGB values into HSV values. This can easily be done in MATLAB before saving the background images on the external SD card.

Since the output after chroma-key compositing will be in the HSV color space, it will need to be converted into the RGB space before being sent to the VGA display. The `hsv_to_rgb` module will be responsible for carrying out this conversion.

Conversion to the HSV color space involves several mathematical operations, including division, which will require the usage of a divider module. The conversion implementation may need to be pipelined for optimization.

Testing of the color space conversion modules will be based on several test conversions done in MATLAB. We can use MATLAB's color space conversion functions to generate test cases for our Verilog implementations. We will test the conversion modules by ensuring that specific input values get converted to the expected values. We can use LED outputs to aid us in verifying the functionality of this module on single pixel values before applying it to image data.

3.5 Chroma-Key Compositing

Chroma key compositing is a post-production technique for layering two image or video streams together based on color hues. This technique is often used to remove a background from the subject of a photo or video and layer the foreground onto a filmed background footage or a static image. In our design, a video stream will be paused and the static video frame will serve as one image stream. A static image will serve as the second image stream. We will use a green backdrop as our chroma-key.

This module will be the cornerstone of our project. By using chroma key detection, we will remove a green screen from a stored video frame. We will multiplex the pixels from the stored foreground with the pixels from the user-selected background image using a green chroma key selector. The module will output the composited image.

We will test the module incrementally. First, we will determine that the chroma key detection module is working correctly and that our threshold values are appropriately chosen. By displaying on the monitor what the module thinks is green, we can make sure that it is correctly detecting the green (we will use an image that has very clear green portions). Once this is working, we will be able to test the mixer module that will replace the green pixels with the corresponding pixels from the background image. If this works, we can then go about using camera input data instead of a dummy image.

3.6 Text Generator Subsystem

The module will be used to create sprites of letters but will not directly overlay them onto the image. The Text Generator will be a FSM with around 30 states (alphabet plus a few special characters). The generator will be called by the Text Overlay Module. The letters will be of fixed size and theme.

This module will also be easiest to test after the VGA and monitor are set up. We will need to make sure that the right letters are output with the correct input.

We will also text that letter proportions are correct.

3.7 Graphics Generator Subsystem

This module is similar to the Text Generator module however it will have fewer states (target is 4: crown, safari hat, mustache, and sunglasses). It, too, will not be called on its own but rather the Graphics Overlay will call it. It will take in which graphic it is to create and output the corresponding sprite.

Similar to above, we will test the module after the VGA and monitor are set up. We will make sure that the right graphics are output with the correct input. We will also test that proportions are appropriate, especially when fitting and resizing.

3.8 Text & Graphics Overlay

These two modules will take on/off inputs from the switches. By toggling specific switches, the user will be able to incorporate the text and/or graphics features into their photo. Both modules will be FSM's that will choose specific text or graphics based on the inputs given to them. For the text overlay, additional inputs received from the background select will help to determine which text phrase to select. Text overlay will the instantiate the Text Generator module accordingly. As a stretch goal, we will also generate custom text based of input from the PC. The graphic overlay module will also be quite similar, however it will receive more inputs from addition switches. The different switches will select between a number of different graphics to overlay onto the photo. The graphic overlay will the instantiate the specific chosen graphic, and it will also estimate where best to place the object (i.e. crown should be near top of head). The placement of the object will be adjustable by use of buttons.

Testing for these modules will be done in conjunction with their respective generator modules. Using the monitor we will make sure that they behave correctly with the given commands (i.e. moving image). And later when we have the green screen and chroma key working, we will test the graphic placement by comparing what our module calculates to be appropriate destination locations to what we decide they should be. We will then iterate upon this process until it provides output close enough to what we want.

3.9 VGA Output

The Nexys4 DDR board uses 14 FPGA signals to create a VGA port with 4 bits per color and the two standard sync signals (HS – Horizontal Sync and VS – Vertical Sync). 12 bit RGB output will allow us to display up to 4096 different colors. In our design, the VGA monitor will be driven in 640x480 mode. Correct VGA output timing will be determined from the Nexys4DDR documentation and a demo project provided by Digilent.

3.10 FPGA-PC Communication

In order to provide additional user interaction with the system, FPGA-PC communication will be implemented. A communication channel between the FPGA board and the PC will allow the static frame currently being displayed on the monitor to be transmitted from the FPGA to the PC as a series of RGB values; on the PC end, the image will be reconstructed and saved in uncompressed bitmap format. Such a communication channel will also allow the user to send custom text from the PC to the FPGA as a series of character bytes; the Text Generator subsystem will then process these received bytes and display the corresponding text.

Since the Nexys4DDR board contains a shared UART/JTAG USB port, we have decided to use UART for serial FPGA-PC communication. UART transmitter (TX) and receiver (RX) modules will be written in Verilog for FPGA-side communication, and Python or MATLAB programs will be written for PC-side communication.

In our design, communication between the FPGA and the PC will occur unidirectionally: custom text characters will be sent from the PC to the FPGA before the text is added to the chroma-key composited image, and image data will be sent from the FPGA to the PC after all text and graphics have been (optionally) added and the user has initiated data transfer. Implementation of UART is a baseline goal for our project and will be verified through successful FPGA-to-PC image data transfer. PC-to-FPGA text transfer is part of the custom text generation stretch goal.

4 List of Modules

- `sd_card_reader` (baseline)
- `camera_input` (baseline)
- `memory_controller` (baseline)
- `rgb_to_hsv` (baseline)
- `hsv_to_rgb` (baseline)
- `chroma_key_detector` (baseline)
- `chroma_key_mixer` (baseline)
- `image_enhancer` (stretch)
- `char_generator` (expected)
- `text_generator` (expected+stretch)
- `graphics_sizer` (expected)

- `graphics_generator` (expected)
- `output_image_mixer` (baseline)
- `effects_filter` (expected+stretch)
- `vga_output` (baseline)
- `uart_tx` (baseline)
- `uart_rx` (stretch)

5 Timeline and Responsibilities

5.1 Timeline

We plan to complete the specified desired functionalities in the following timeframe:

(1) **Week of October 31**

- Camera input and VGA output; SD card interface.
  In the first week, we will set up the camera input to the Nexys4 and then send that data to a memory buffer.Then the VGA output module will read from the buffer to show the stored data. We also plan to to get the "shutter" button functionality working. Since usage of memory buffers, the camera input, and VGA output are all vital to our project, we will collaborate heavily on this to have it functioning as soon as possible.

(2) **Week of November 7**

- RGB to HSV conversion; Chroma key compositing.
  Our goal in the second week will be to get chroma key detection working. This will involve us completing the integration of all the memory buffers along with the chroma key detection/compositing and color space conversion modules. It's imperative that we get this working as soon as possible so we can fine tune it and start working on the other modules.

(3) **Week of November 14**

- HSV to RGB conversion; Text generation; Filter effects.
  The HSV to RGB conversion module will hopefully be easier to design after gaining experience working with the RGB and HSV color spaces. The text generation goals of this week will hopefully be the easier part of this week. To generated curved edges multiplication and pipelining will need to be considered and incorporated. Filter effects will be a bit of experimenting but we believe that will will be able to figure it out this week as well.

(4) **Week of November 21**

- Graphics generation; Output image mixing.

  Due to similarities between the Text Generator and the Graphics Generator, we hope that we will be able to focus the majority of our efforts this week on the image mixing, which is the main aspect of our final output.

(5) **Week of November 28**

- UART data transfer; Integration.

  Integration will be the most debugging intensive step of our project. While we will have been extensively testing our modules individually up to this point, there will inevitably be more bugs when integrating the modules in one system. We plan to implement the UART portion relatively quickly and spend the majority of this week in the integration/debugging stage. If integration goes well, Thanksgiving break will give us a chance to work on a few of our stretch goals.

(6) **Week of December 5**

- Finishing touches and optimization.

  This week will also act as a bit of a buffer in case other stages take longer than we initially budgeted for (however, we will try our best NOT to fall behind! "How do you fall behind? One day at a time.").

(7) **Week of December 12**

- Checkoff. Write-up due on December 14.

5.2 Responsibilities

Subsystem	Designer(s)
Camera Input	Both
SD Card Interface	Diana
Memory Controller	Both
Color Space Conversion	Diana
Chroma Key Detection	Both
Chroma Key Mixing	Lorenzo
Character Generation	Lorenzo
Text Generation	Lorenzo
Graphics Sizing	Lorenzo
Graphics Generation	Lorenzo
Image Mixing & VGA Ouput	Diana
Filter Effects	Diana
UART Data Transfer	Diana
Image Enhancement (stretch)	Both

6 Resources

Resources needed for this project include the Nexys4DDR FPGA board, a camera, a compatible micro SD card, and a green screen.