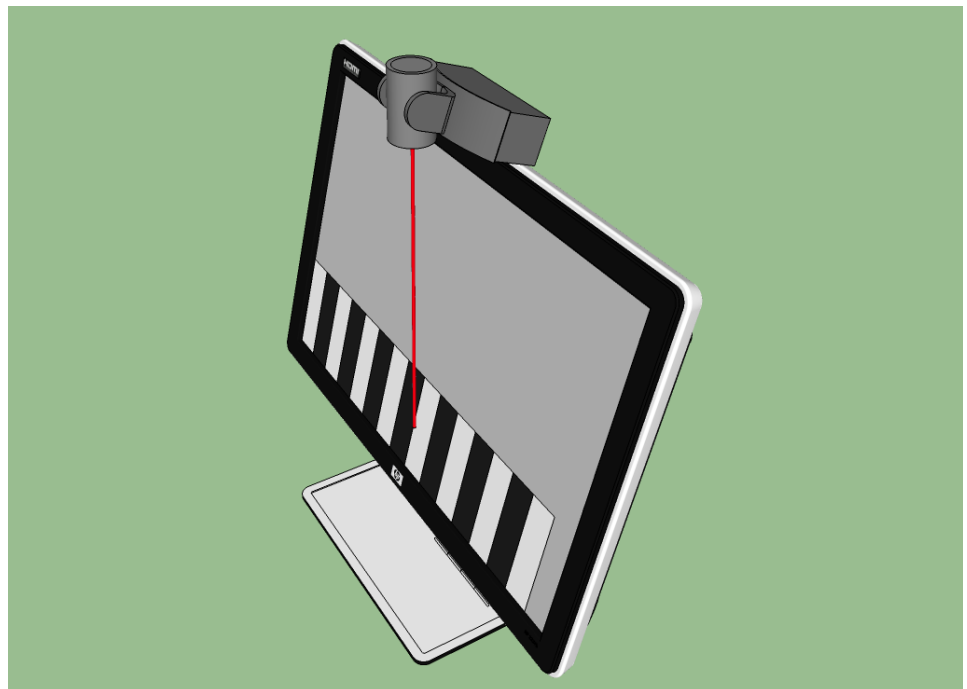


PIANO PLAYER WITH VIRTUAL TOUCH KEYBOARD



2/23/2014

Project Design

CSEE W4840 EMBEDDED SYSTEMS

Spring 2014 Prof. Stephen A. Edwards

By Daran Cai (dc2946@columbia.edu)

Linjun Kuang (lk2578@columbia.edu)

Wei Xia (wx2147@columbia.edu)

Wenyuan Zhao (wz2262@columbia.edu)

Piano player with Virtual touch keyboard

PROJECT DESIGN

ABSTRACT

This document describes our preliminary project design implementation based on our research so far. For our project, we are going to implement a piano player. Unlike traditional piano player software and applications, we will use a normal screen and a camera to realize a virtual touch screen and a virtual keyboard. We will use a Sockit board, an ARM Cortex- A9 processor, a camera (Logitech HD Portable 1080p Webcam C615 with Autofocus), a VGA display and a pair of speakers.

INTRODUCTION

The key point of our project is how to use a camera and a normal screen to realize the function of a touch screen. The basic idea is that camera is used to capture the image of fingers and the screen in real time, and then using computer vision algorithm to recognize the relative position of the finger on screen. Using

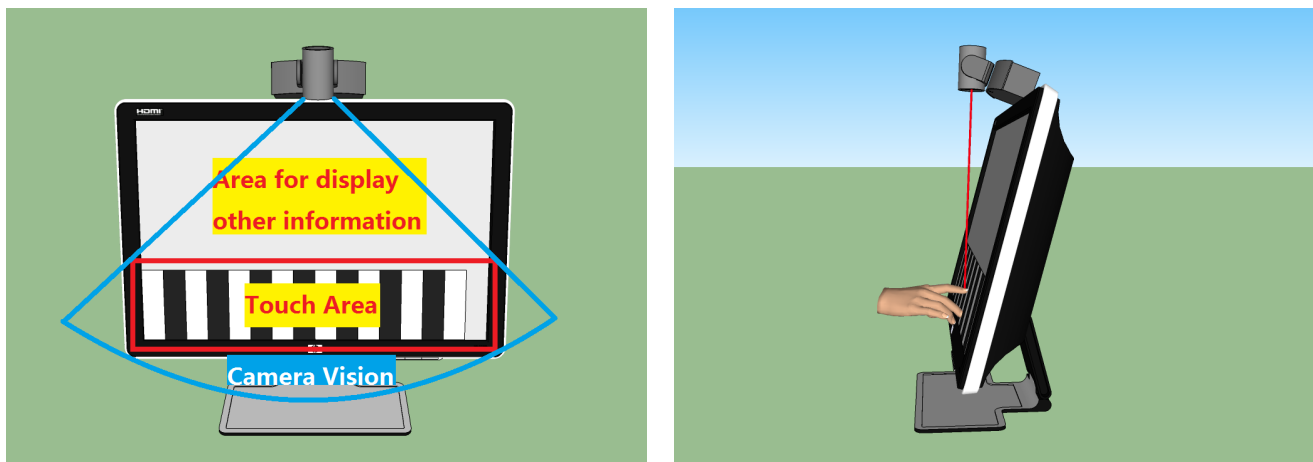


FIGURE 1: THE INTRODUCTION OF THE KEY CONCEPT

this relative position, we can achieve the function of the touch screen.

As for the user interface, we will use sprite graphics. There will be a keyboard on the bottom of the screen. When users press those virtual keys, the corresponding sounds will come out.

Except for the basic free mode, in which users can play whatever they like, we will provide other modes such as tutorial mode and player mode. If we can realize these modes, we can add more interesting mode.

We got the basic idea of our project from a project made by Cornell student, but we will do many innovations and improvements in our project. The description of this Cornell project is listed in reference part.

The implementation of image processing and user interface will be done using FPGA. Sound samples will be stored in the board, and called using software.

HIGH LEVEL BLOCK DIAGRAM

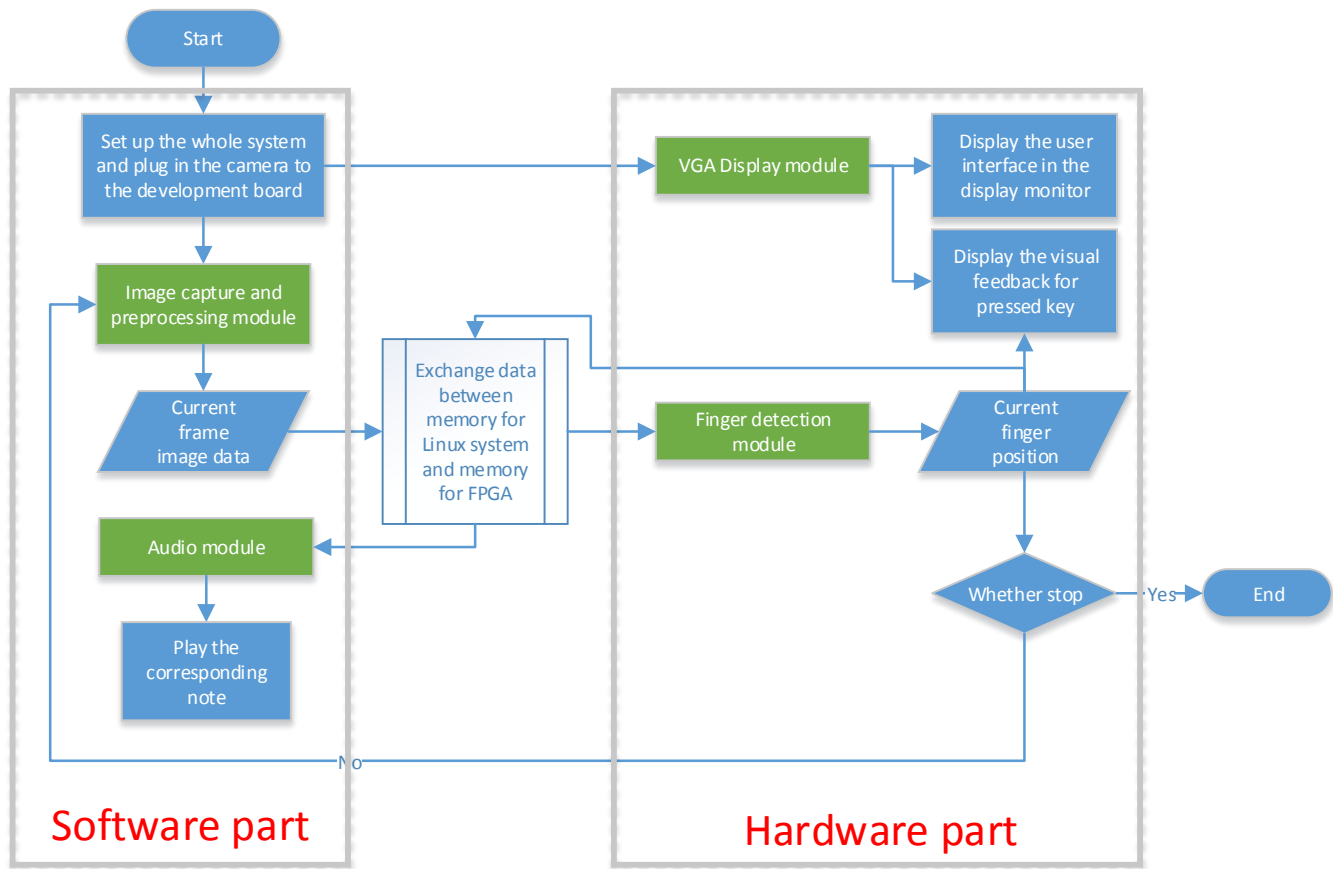


FIGURE 2: HIGH LEVEL BLOCK DIAGRAM

We will introduce those four main modules in the detailed implementation section.

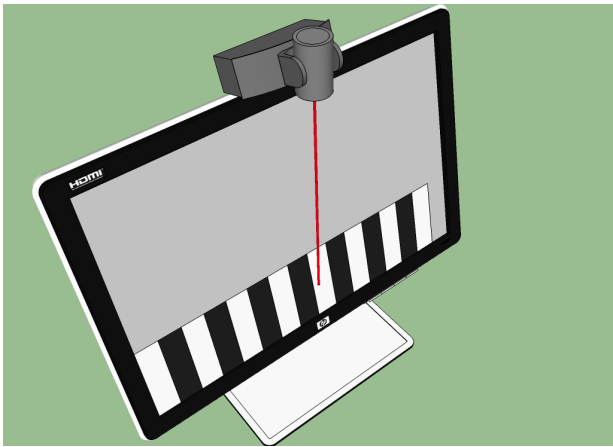
DETAILED IMPLEMENTATION

Image capture and preprocessing module

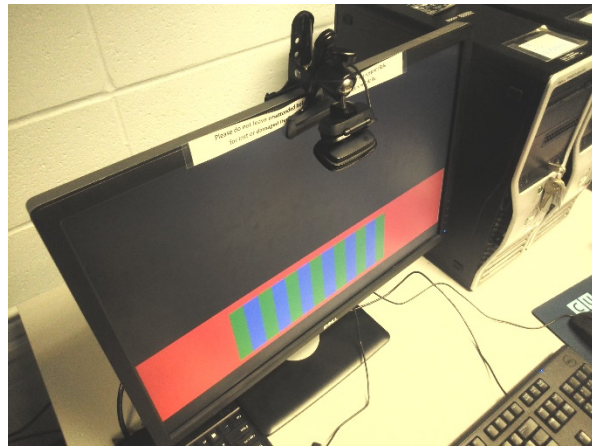
Hardware implementation

As design sketch in Figure 33(a) showed, in order to implement a virtual touch display screen, we have to use a camera to capture the image of finger and display monitor in a special angle. To approach this design, in real hardware implementation for capturing image, there are three main parts which are a web-camera, a camera clip and a normal display monitor.

In our implementation, we fix the web-camera with the camera clip and fix this camera clip at the top of the display monitor. Since the camera clip we used can be set to almost any angle, we will set it to an appropriate angle to make sure the web-camera can capture the entire keyboard area in the display monitor. The whole hardware implementation is showed in Figure 33(b).



(a) Desin sketch.



(b) Real implementation.

FIGURE 3: THE WHOLE HARDWARE IMPLEMENT FOR IMAGE CAPTURE.

We will introduce each component of this implementation in detail below.

WEB-CAMERA

Since there are some USB ports in our development board, we decide to use a USB Web-camera as the camera to capture the image of finger and display monitor. In order to make this real-time reorganization system has enough sensitivity to the position of the finger, we hope this Web-Camera can have higher resolution and fps. Therefore, finally, we decide to use Logitech C615 Web-Camera and the specifications of the Web-Camera are list below:



FIGURE 4: LOGITECH C615 WEB-CAMERA

- Full HD 1080p video capture (up to 1920 x 1080 pixels) with recommended system
- Autofocus
- Hi-Speed USB 2.0 certified (recommended)

- Universal clip fits laptops, LCD or CRT monitors

CAMERA CLIP

For the camera clip, it should satisfy the follow requirements:

1. Can be fixed at any kinds of display monitor.
2. Can hold the Web-Camera stable.
3. Can be set to any angle.

After saw a lot of kinds of camera clip, we found the camera clip in Figure 55 can satisfy those requirements.

1. The clip part of this camera clip can fix at almost any kind of display monitor.
2. By using screw to combine with Web-Camera, it can make sure the stability of the Web-Camera.
3. The ball head part of this clip make it possible to be set to the any angle.

Therefore, in our design, we decide to use this camera clip.



FIGURE 5: NEEWER® MULTI-FUNCTION SPRING CLAMP CLIP HOLDER MOUNT WITH BALL HEAD W/ STANDARD 1/4" SCREW FOR SLR, DIGITAL SLR, VIDEO CAMERAS

Software implementation

The key part of our design is to use a Web-Camera to capture the image of the finger and the display monitor and recognize the position of the finger on the display monitor. We decide to split this part in two main subparts. One is to get the real time captured image from the Web-Camera and send it to the memory in FPGA and the other one is to do some image processing program on this captured image to find the position of the finger. We decide to implement the first part in LINUX environment named “image capture and preprocessing” and implement the second part in FPGA environment named “finger detection”. In this section, we will talk about the software implementation of the part “image capture and preprocessing”. The high level block diagram of this part is showed below:

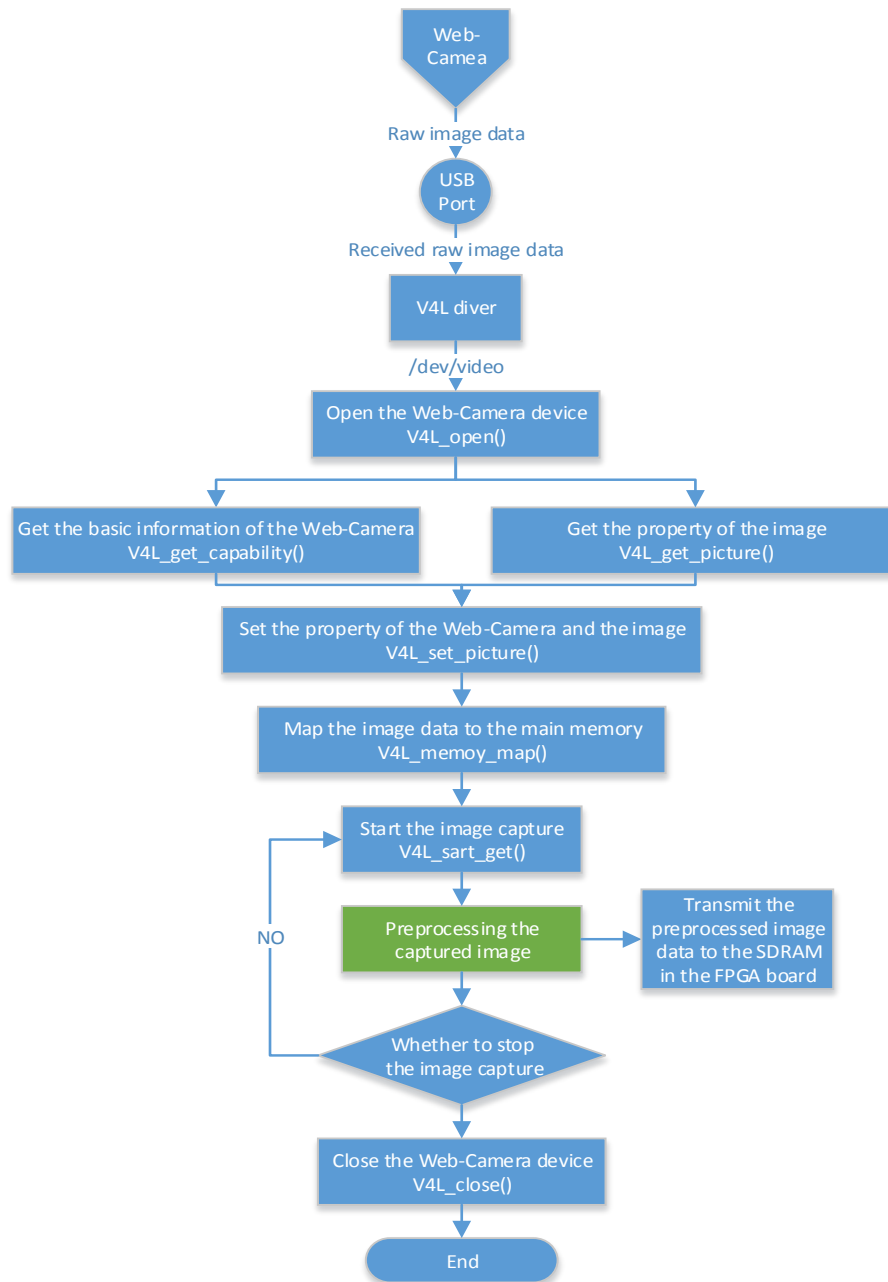


FIGURE 6: HIGH LEVEL BLOCK DIAGRAM FOR SOFTWARE IMPLEMENTATION OF "IMAGE CAPTURE AND PREPROCESSING" PART

INTRODUCTION TO V4L

Video4Linux (V4L) is the driver for video devices from LINUX kernel, it can provide a series of APIs for the program which intend to use video devices as input. If we want to use this driver, we have to make sure our Web-Camera can support UVC (USB Video Class). Fortunately, the Web-Camera we used can support UVC.

Under the LINUX environment and the function of V4L, all the peripherals are considered as special files named device file. The APIs the V4L driver provided can allow us to read and set these files to get the image data we want. This device file will be stored at the folder “/dev/video”.

SET THE FRAME IMAGE PROPERTY AND MAP IT TO THE MAIN MEMORY

- Use V4L_open() to open the device file, set the path of the device file to “/dev/video”.
- Use V4L_get_capability() and ioctl() to get the information of the Web-Camera, and store those information to the struct video_capability.
- Use V4L_get_picture() and ioctl() to get the property of the image, and store those property to the struct video_picture.
- Use the V4L_set_picture() and ioctl() to set the property of the image in order to get the appropriate resolution and so on.
- Use V4L_get_mbuf() and ioctl() to get the information of frame image from the buffer of Web-Camera, and store information to the struct video_mbuf.
- Use V4L_set_mmap() to set the property of the frame image we want to get from the device file.
- Use V4L_mmap() and ioctl() to map the device file to the main memory.

CAPTURE THE FRAME IMAGE DATA

Use V4L_start_get() and ioctl() to begin mapping the frame image to the main memory. After getting the frame image, convert this image to the 3-D array to present the RGB parameter of each pixel. The block diagram is shown below.

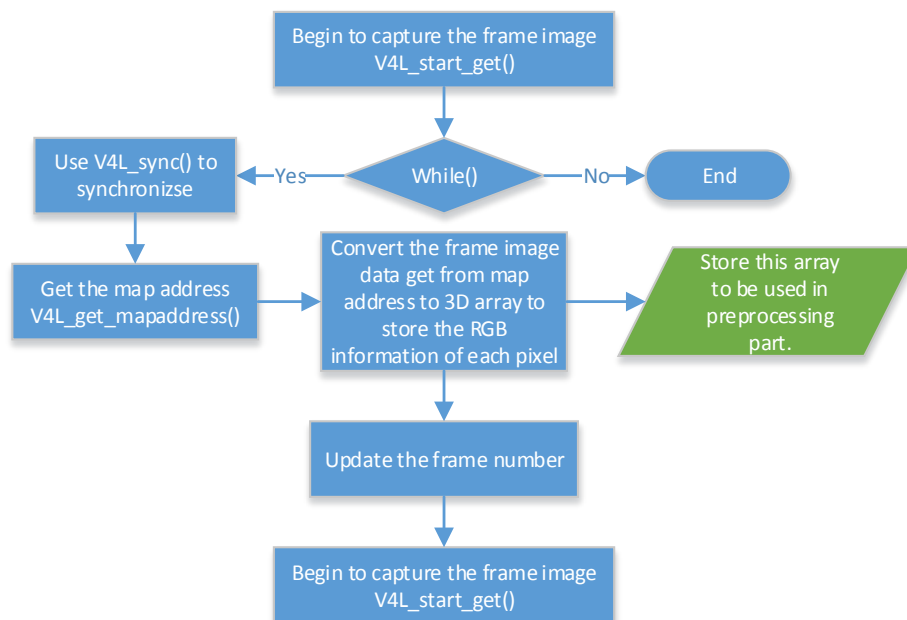


FIGURE 7: BLOCK DIAGRAM FOR CAPTURING THE FRAME IMAGE

PREPROCESSING THE CAPTURED IMAGE

The input of this part is the 3D array of the captured frame image which includes the RGB parameter for each pixel. In order to make it convenience for “finger detection” in hardware and decrease the memory cost, we want to use software part to do some preprocessing of the frame image. The block diagram for this algorithm is showed below:

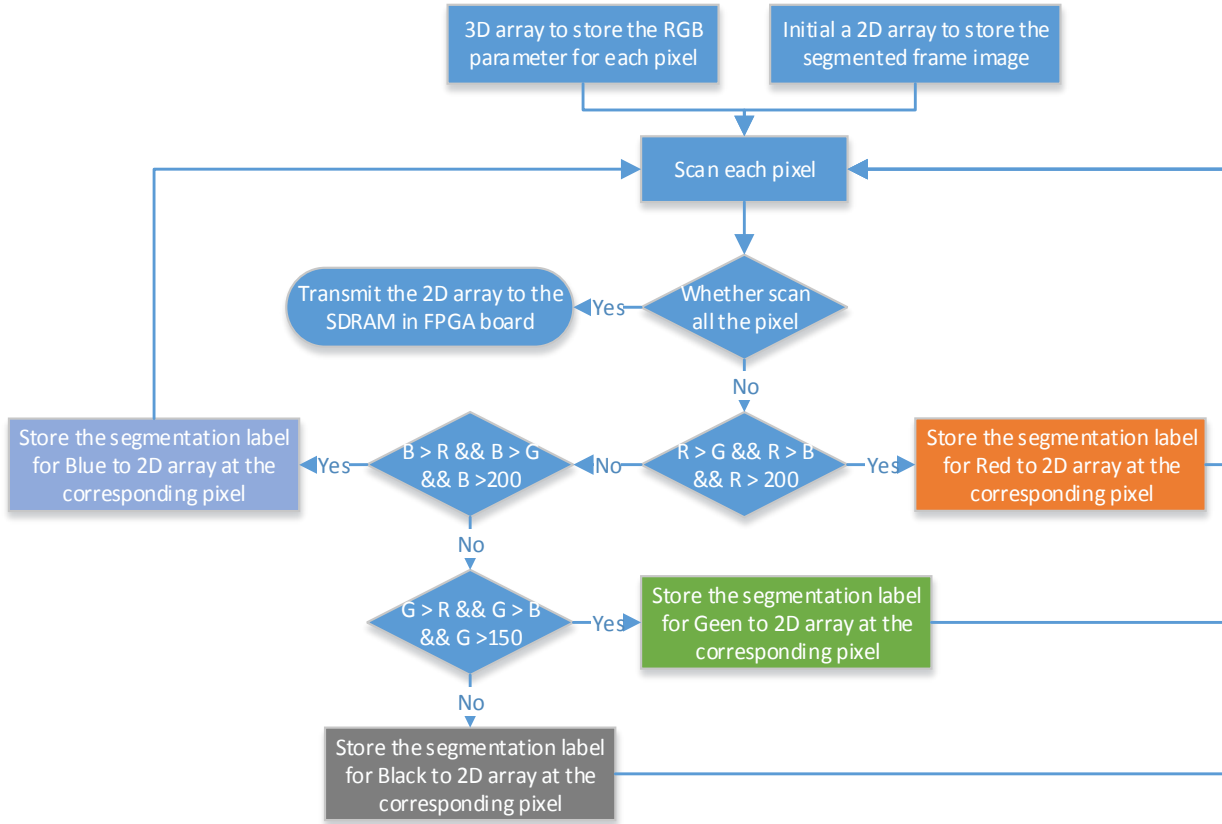


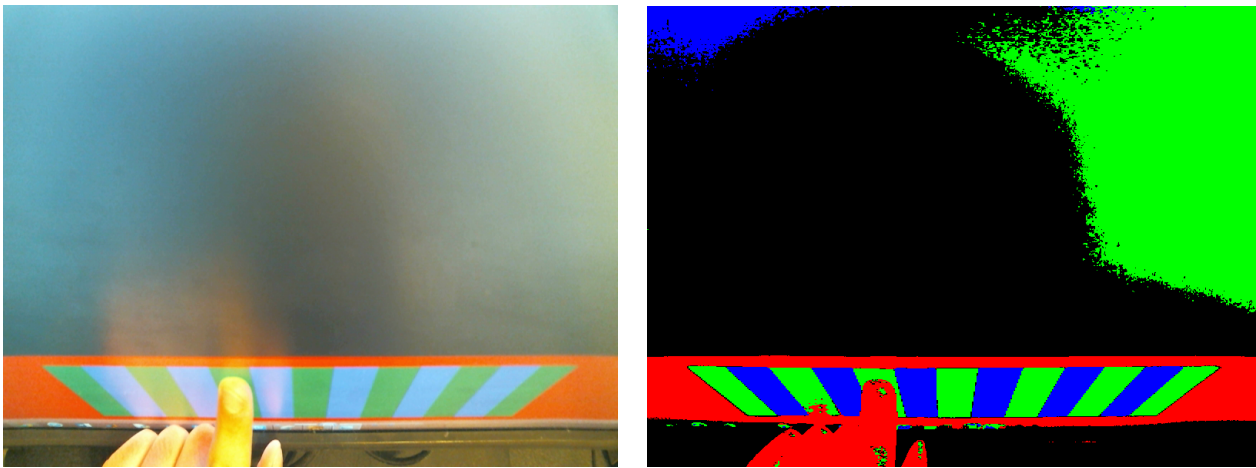
FIGURE 8: BLOCK DIAGRAM FOR THE ALGORITHM OF "PREPROCESSING THE CAPTURED IMAGE"

Firstly, we will use the RGB parameter for each pixel to segment the frame image to four kinds of colors, the red, the blue, the green and the black. The rule of this segmentation is presented in the table below.

TABLE 1: THE RULE OF THE SEGMRNTATION

Color	Condition	Segmentation label
Red	$R > G \ \&\& \ R > B \ \&\& \ R > 200$	1
Blue	$B > R \ \&\& \ B > G \ \&\& \ B > 200$	2
Green	$G > R \ \&\& \ G > B \ \&\& \ G > 150$	3
Black	Otherwise	0

The Figure 98 show the result of the segmentation of a frame image. However, in order to compress the image data which will be stored in SDRAM in FPGA, we will create a 2D array to store the segmentation label for each pixel and transmit this to the “finger detection” part in hardware implementation.



(a)The frame image captured by Web-Camera. (b)The frame image after segmentation.

FIGURE 9: SEGMENTATION OF THE FRAME IMAGE

Finger detection module

The finger detection algorithm takes an 800*600 array as input and outputs two signals: one 1-bit signal telling if there is a key being pressed, one 4-bit signal telling which key is pressed since we could have 8, 12 or 16 keys displayed on screen.

The image data we get from the SDRAM only has four colors: black, red, green, and blue. The background is in black. Red is used to separate the background and keyboard area. Green and blue

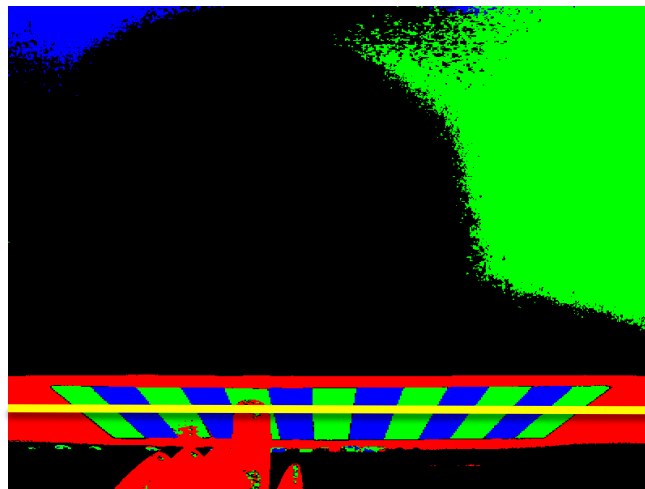


FIGURE 10: IMAGE DATA

are used as two alternating colors of keys. The actual image is shown in figure 10.

The basic idea can be described below.

1. Scan the vertical pixels, if we find several consecutive red pixels (There may be noise pixels and a single red pixel cannot tell anything), we will know that the area below these red pixels are keyboard area. Then we assign the horizontal line, say 30 pixels below the last detected red

- pixel, as the target horizontal line. This target horizontal line is used to label different keys and detect if there is a key being pressed.
2. Scan the target line from left to right. If there are several pixels that are green, we will assign the position of the first detected green pixel to the label0_start signal. Continue scan the target line, if there are several blue pixels, we will assign the position of the first detected blue pixel to the label1_start signal. The label0_end will be assigned as label1_start - 1. After scanning the whole line, we compare the number of labels with the number of keyboards. If they are different, we must do it again. The label information is used as a reference when trying to detect if there is a finger. (Here we need to design a finite state machine to get rid of noise, since there may be some noise pixels in the middle of a key and will cause disaster. This is also for the purpose of robustness.)
 3. Finally, we compare the difference in each label of our real time image with the original color (Green or blue). If the different pixels exceed the threshold value, we could say that the corresponding key is pressed.

The detailed design is below.

The whole algorithm should be divided into 3 modules: keyboard detection, label, diff. Figure 11 shows a high level block diagram of the connection between these three parts.

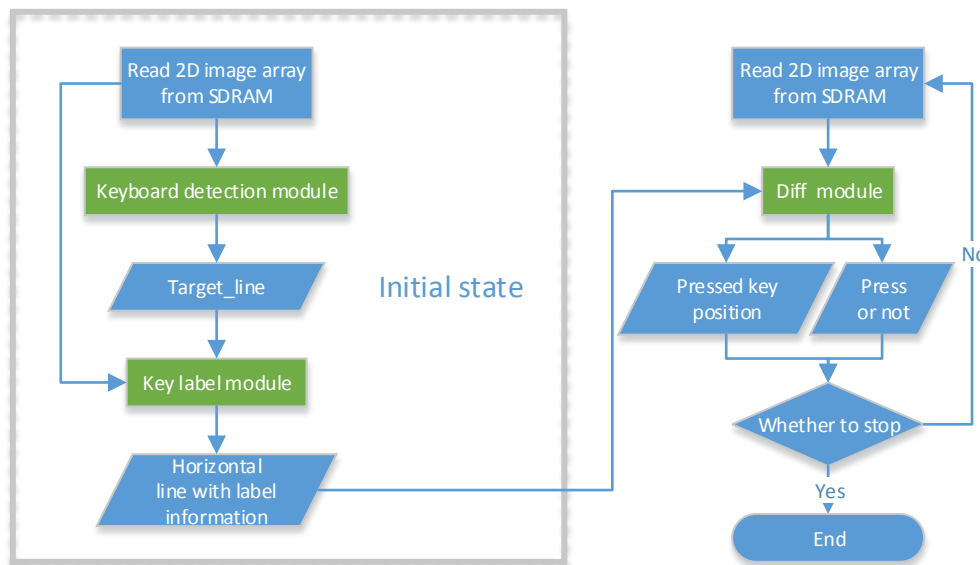


FIGURE 11: FINGER DETECTION BLOCK DIAGRAM

Keyboard detection module

The keyboard detection module takes the image array as input and outputs a 10-bit signal telling which horizontal line we are using to label and detect finger positions. The image array has $800 * 600$ pixels and each pixel is implemented using 2 bits. 00 stands for black, 01 stands for red, 10 stands for green, and 11 stands for blue.

At the rising edge of clock, compare the color of each vertical pixel with red. If it is not red, the vertical counter increases by 1. If it is red, assign the target signal with the current vertical counter plus 30. We also need a FSM to control the flow of operation.

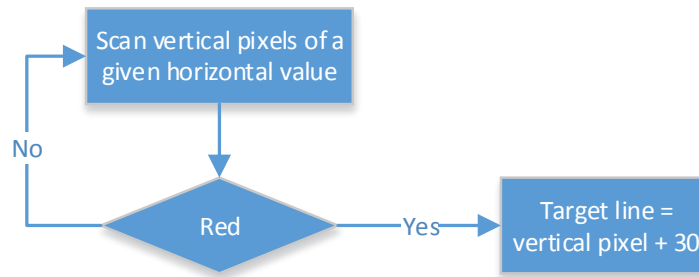


FIGURE 12: KEYBOARD DETECTION MODULE

Key Label module

The label module takes the image data and the 10-bit target horizontal line value as input. It outputs up to 32 10-bit signals telling the beginning and ending position of each label.

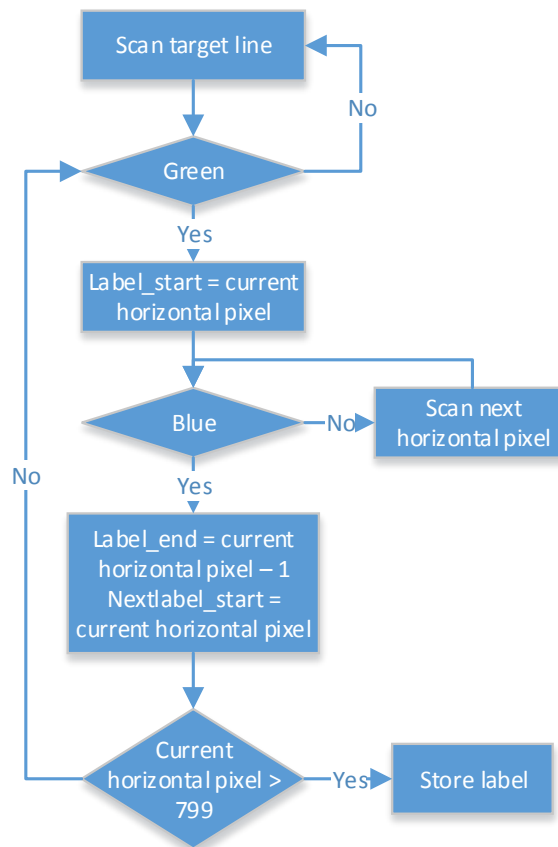


FIGURE 13: LABEL MODULE

Diff module

The diff module takes the label information as inputs and outputs a 1-bit signal and a 4-bit signal telling the sound and display block whether there is a key being pressed and which key is pressed.

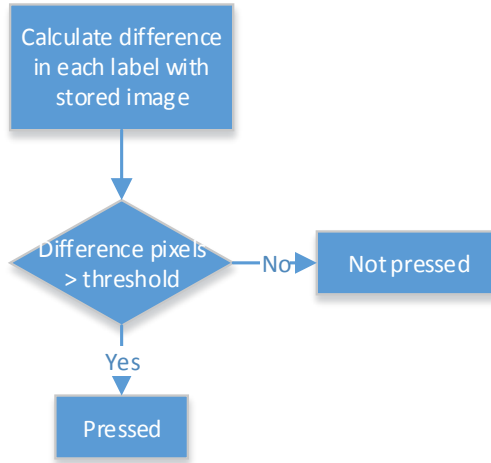


FIGURE 14: DIFF MODULE

Audio module

For our project, we are going to store the audio file in the SDRAM. The ARM processor will fetch audio data from the SDRAM, computes and sends commands to the audio controller to play the sound.

The hardware part of the audio is simple. The hardware structure of the audio is built in the SRAM, and it will connect with CPU, who connects data in SDRAM. Memory is big enough to store sound samples (see in memory requirements) and CPU can easily call those samples.

The board could provide 24-bit audio via the CODEC encoder/decoder. This chip supports microphone-in, line-in, and line-out ports with a sample rate adjustable from 8kHz to 96kHz. The audio chip is configured in master mode. The audio interface is configured as I2S and 16-bit mode. We will also use an 8kHz sample rate. (Try to make the memory requirement small. If there is a need, we will adjust the sample rate later.)

The slide switch setting for sample rate for audio player is shown below.

TABLE 2: SLIDE SWITCH SETTING FOR SAMPLE RATE FOR AUDIO PLAYER

SW3 (0-down, 1-up)	SW2 (0-down, 1-up)	SW1 (0-down, 1-up)	Sample rate
0	0	0	96K
0	0	1	48K
0	1	0	44.1K
0	1	1	32K
1	0	0	8K
Unlisted combination			96K

The software implementation of audio part is done by CPU. When a key is pressed, the finger is detected by the camera and after the image processing, it will call the corresponding sound samples and the piano sound will be played via speaker.

VGA Display module

There are five kinds of elements to be shown

- A keyboard on the bottom of the screen
- A background on the rest of the screen
- A shadow effect when a key is being pressed
- Falling blocks
- An effect when a falling block hits the corresponding key

Detailed arrangement is shown below.

TABLE 3: ELEMENTS DEINITION FOR SPRITE

Sprite #	content	Sprite #	content
0	keyboard	1	background
2	shadow	3	blocks
4	Hitting effect		

Sprite with bigger number of index will overlap on the sprite with smaller number of index. Hence, we manually divide these five sprites into two levels. Sprites 0 and 1 consist the basic first level, they are fixed images on the screen and are not involved in any kinds of motion. Sprites 2, 3, and 4 consist the second level, they are not fixed, involving changing position and motion, and they are not always displayed on the screen, and there are interactions between them and the information we get from the image processed.

The level of the sprites is shown as follows,

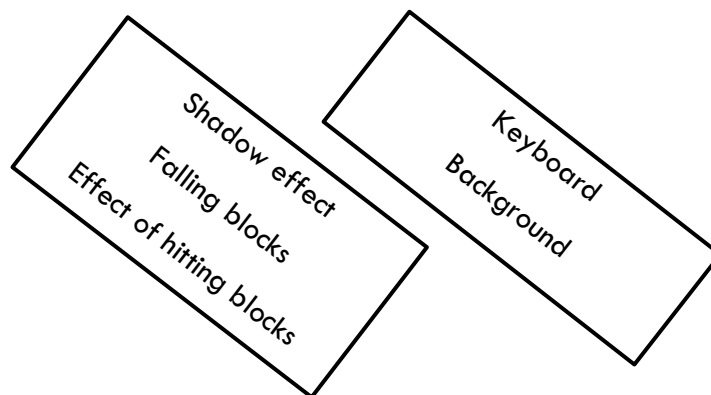


FIGURE 15: THE LEVEL OF THE SPRITES

All those elements can be stored as fixed figures in the SDRAM on the board. The background on the top of the screen is 320*640 pixels. And it is drawn as one sprite.

The lower area with blue and green blocks is used as keyboard area. It is 160*640 pixels. It is also drawn as one sprite.

When the user pressed the key, when the image processing ends, it send the command to the sound, and simultaneously, the pressed key will be shadowed. And will be light gray. If we use 16 keys keyboard as an example, the shadow will be a 160*40 pixels sprite.

The above are the basic function we want to implement. The following are the higher level task we want to implement. And it is a harder part.

There are two kinds of sprites in this part. One is a small ball, the other is a larger ball.

The upper area is used to display notes that the user supposed to play. The note to be pressed is in a small ball. If the note pressed at the right time, it will turned to a larger ball. If the user miss the note during the appropriate time, the note will not change. Also, the key which is pressed would be shadowed.

CRITICAL PATH ANALYSIS

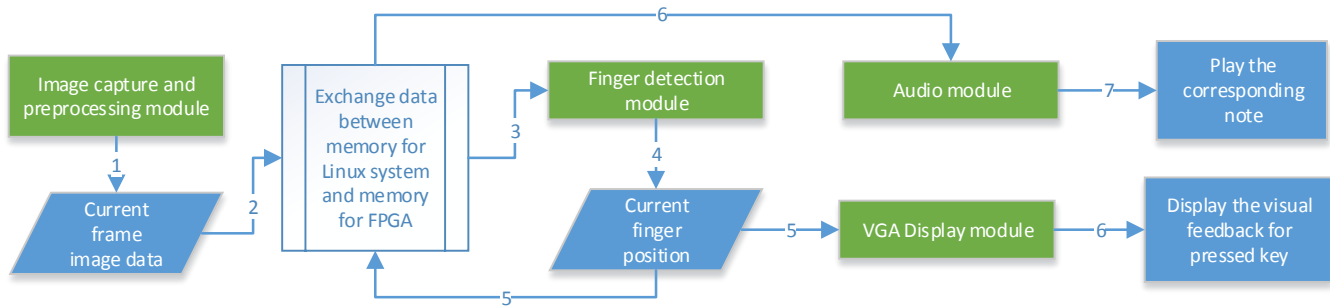


FIGURE 16: THE BLOCK DIAGRAM FOR CRITICAL PATH

The response time for the whole system should be the time interval between the finger pressing a key and the audio feedback or visual feedback of the pressed key. As Figure 16 showed, the critical path should be the longer one between two paths. The one is from the beginning of capturing image to the visual feedback is displayed in monitor, and the other one is from the beginning of capturing image to the note of that pressed key is played.

MEMORY REQUIREMENTS

Camera

We will store the image data in SDRAM. The preprocessed image data has $800 * 600$ pixels. Each pixel is represented using 2 bits, 00 for black, 01 for red, 10 for green, and 11 for blue. The memory required to store this image is $800 * 600 * 2 = 960000\text{bits} = 120\text{K bytes}$.

User interface

We will use SDRAM to store the sprites. The sprites should contain five basic parts: a keyboard on the bottom of the screen, a background on the rest of the screen, a shadow effect when a key is being pressed, falling blocks, and an effect when a falling block hits the corresponding key.

Considering the screen is $640*480$ pixels, we intend to use $240*480$ pixels to display the keyboard, and the rest is the background.

We will make three kinds of modes which have 8 keys, 12 keys and 16 keys. The memory the

- Keyboard: $160*640 * 3 = 307200$ bytes;
- Background: $320*640 * 3 = 614400$ bytes;
- Keyboard pressed shadow: $160*640 * 3 = 307200$ bytes;
- Falling blocks: $50*640 * 3 = 96000$ bytes.
- Effect of hitting blocks: $100*640 * 3 = 192000$ bytes.

The total memory usage is nearly 2MB.

Sound samples

TABLE 4: MEMORY COST FOR AUDIO NOTE

Note	C3	D3	E3	F3	G3	A3	B3	C4
Memory cost	613KB	613KB	613KB	613KB	613KB	613KB	613KB	613KB
Note	D4	E4	F4	G4	A4	B4	C5	C6
Memory cost	613KB	613KB	613KB	613KB	613KB	613KB	613KB	613KB

Total memory cost: $613\text{KB} * 16 = 9808\text{KB}$, nearly 10MB.

MILESTONE

Milestone 1 (April 1st)

- Buy the camera and support it on the top of the screen.
- Build a Linux system on an SD card.
- Realize image capturing and preprocessing in Ubuntu operating system, develop algorithms for finger detection.
- Try to draw a simple keyboard using sprites.

Milestone 2 (April 15th)

- Complete image capturing and processing on the board, modify the algorithm and realizing finger detection.
- Finish all sprites for user interface, including keyboard, background, shadow feedback and blocks. Test shadow feedback using USB keyboard.
- Add sound samples and test them using USB keyboard.

Milestone 3 (April 29th)

- Connect all parts together and do joint debugging.(camera & sprite & sound)
- Make falling blocks and hitting effects.
- Try to realize changing modes and changing tones.

REFERENCE

1. Cornell project link:

http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/s2013/cl972_rh523/cl972_rh523/index.html

2. Sockit user manual.