CE2003 Digital Design

**Laboratory 4 Report**

Name: CHEN LIYANG

Matric Number: U1220601J

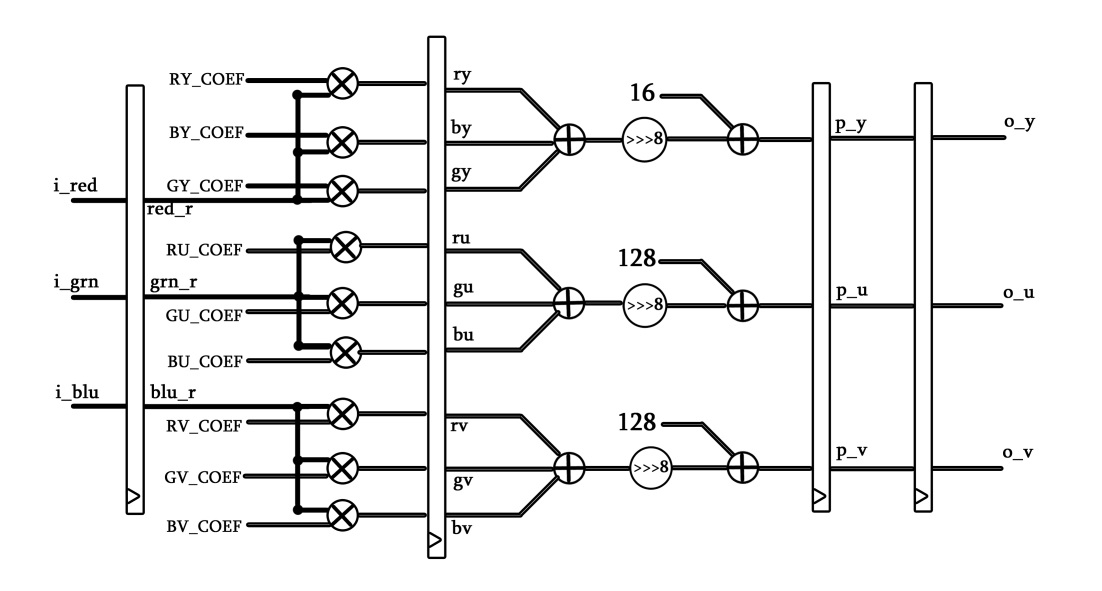
March 2013

**1. Outline description of the experiment design**

The basic process of the design is to get the video stream from the camera, convert input data in the RGB color model into the PUV color model, determine the skin pixels, make some modification on them and then send the modified output to the screen.

The *rgbyuv* module takes inputs in the RGB model and applies a matrix on it converting the RGB model into YUV model. *rgbyuv* module also determines whether a pixel is a skin pixel by judging whether U and V values are in a certain range. The *delay\_line* module delays the original R, G, B signals and control signal to make sure that R, G, B, Y, U, V and the control signal reach the *pixsel* module simultaneously. The *pixsel* module takes charge of what values or signals to output and how the skin pixels should be modified.

**2. Description of the *rgbyuv* module**



For the *rgbyuv* module, the first part is the ports, registers and parameter declarations. *rgbyuv* module takes the R, G, B signals as inputs as well as clock and reset signal, and output the Y, U, V values and a skin detection signal. Since the Y, U, V values are computed from the R, G, B values basing on a matrix, so we declare these coefficients from the matrix as the parameters.



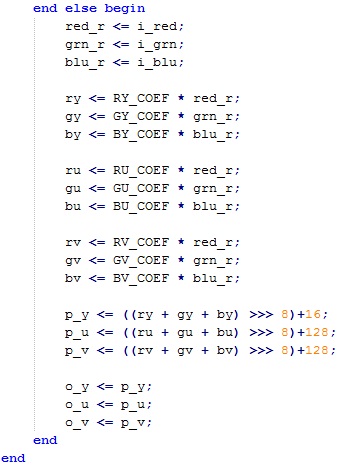


The second part is for resetting the registering when the rst signal is high.

The third part of the *rgbyuv* module is to convert R, G, B signals into Y, U, V model. Basing on the matrix given, using parameters to replace the numbers, the expressions for outputs should be:

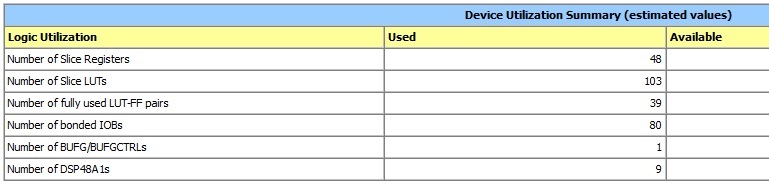


Because in this way the Y, U, V values are generated in a single expression, the clock speed should be slow enough to wait the computation of the expressions to complete. As a result, the clock speed will be too slow for the video datapath. So the *rgbyuv* module should be pipelined. By separating the multiplication operations away from the addition and shift, we pipeline the *rgbyuv* module and shorten the time required for one clock cycle.

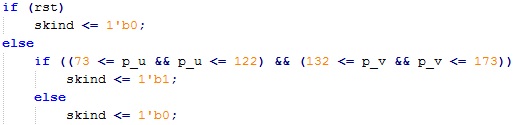


**Result:**

Now the 9 multiplications are computed first, and the addition, shift and offsets adding are done later. Pipelining raises the clock speed up to 193.293MHz. The number of pipelined stages is 5 and the number of multipliers is 9.



The last part of the *rgbyuv* module is to determine whether a pixel is a skin pixel by looking whether the Y and V values are in a particular range. If 73 <= *p\_u* <= 122 and 132 <= *p\_v* <= 173 are true, the pixel is a skin pixel and the corresponding *skind* bit is set to 1(high). Because the Y, U, V values should be outputted with their corresponding *skind* bit simultaneously, so the *p\_u* and *p\_v* is taken into determination. As a result, when *p\_y*, *p\_u* , *p\_v* is calculated, at next positive clock edge the *rgbyuv* module can output the *skind* with right value as well as *o\_y*, *o\_u*, *o\_v*.

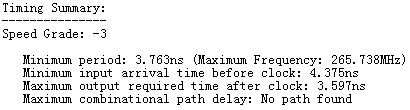
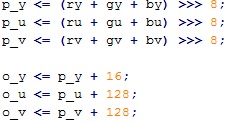


Because in the previous pipelined scenario, the slowest stage is

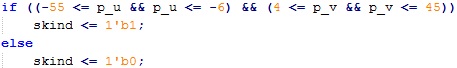


**Pipeline further:**

Thus, if we further pipeline the *rgbyuv* module by splitting the expression into two steps as below, the time for a clock cycle can reduce. As a result, the clock speed can reach 265.738MHz.



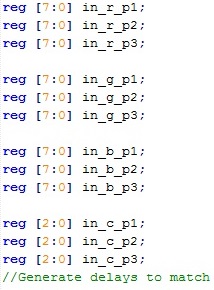
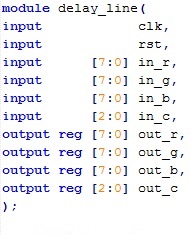
What should be careful is that now the *p\_y*, *p\_u*, *p\_v* values are no longer the value with offset added. So the range in the if statement should be adjusted. Previously the *p\_u* is in 73 ~ 122 and the *p\_v* is in 132 ~ 173, but now what should be in the ranges are (*p\_u*+128) and (*p\_v*+128). That is, the *p\_u* should be in -55 ~ -6 and *p\_v* should be in 4 ~ 45. The reason for using *p\_u* and *p\_v* values but *o\_u* and *o\_v* values is to make sure the *skind* bit and the *o\_y*, *o\_u*, *o\_v* are still produced simultaneously.



In addition, if the time required for the multiplication is still longer than the two additions and one shift operation [ e.g. time(*RY\_COEF \* red\_r*) > time(*(ry + gy + by) >>> 8*) ], we can further separate the two additions and the shift operation ( separate the (*ry + gy + by*) and the ( *>>> 8* ). In this case, the pipeline stages are more, so we should remember to increase the delay in the *delay\_line* module.

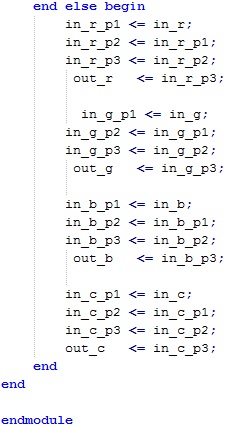
**3. Description of the *delay\_line* module**

For the *delay\_line* module, the first part is also the port and register declarations. The *delay\_line* module should have enough registers to delay original R, G, B signals.



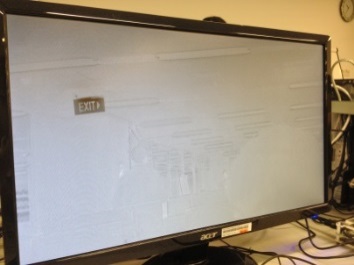
Then if the reset is high, the output should be reset.

Since the *rgbyuv* module takes 5 clock cycles to convert R, G, B into Y, U, V, the *delay\_line* module should delay the R, G, B signals so that they take 5 clock cycles to reach the *pixsel* module at the same time as the Y, U, V signals.

**`**

**4. Description of the *pixsel* module**

The *pixsel* module takes the original R, G, B signals and Y, U, V values as inputs, and then it determines which signal among them to output. The three outputs *out\_r*, *out\_g* and *out\_b* are in the R, G, B model for each single pixel. When the selection input *in\_swt* is 1, the Y value of the video stream is assigned to three output signal in R, G, B model. And when *in\_swt* is 2 or 3, the output *out\_r*, *out\_g* and *out\_b* are set to be the U value or the V value.

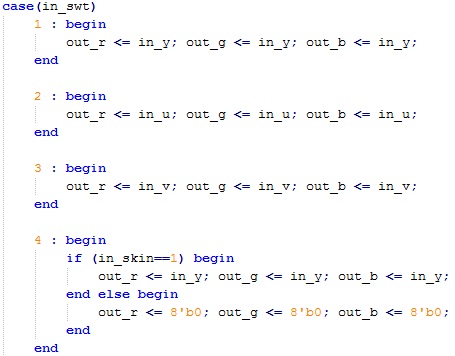


*in\_swt* = 3, V is shown

*in\_swt* = 2, U is shown

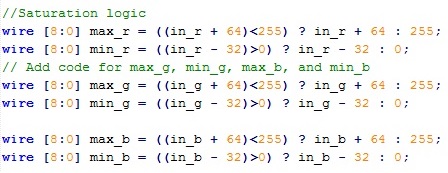
*in\_swt* = 1, Y is shown

When the *in\_swt* equals to 4, the output for a pixel is Y value if the corresponding *skind* bit is 1(high). Otherwise the pixel is assigned to be 0, 0, 0 which is black.

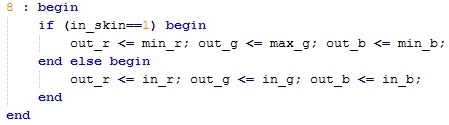
 ****

*in\_swt* = 4, only skin pixels are shown

If the *in\_swt* equals to 8, 9 or 10, the module modifies the R, G, B values of a pixel. To achieve it, 6 wires *max\_r*, *max\_g*, *max\_b*, *min\_r*, *min\_g* and *min\_b* are declared. For the wire *max\_r*, it takes the original R value added by 64. Similar to the *max\_r*, the *max\_g* and max \_b take the increased G and B value and the *min\_r*, *min\_g* and *min\_b* take the decreased R, G, B values. To make sure that the new values stay within the range of R, G, B model, if new values would exceed the color range after adding or subtracting, the max or min will just take the edge value which is 255 or 0.

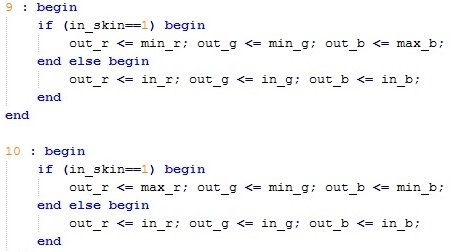


Thus, if the *in\_swt* equals to 8, skin pixels are set to have bigger G value, smaller R and B values. Then the pixel becomes greener, less redder and less bluer with other pixels remaining the same.



*in\_swt* = 8, skin pixels are greener

Similarly, if the *in\_swt* equals to 9, skin pixels are set to be bluer, less redder and less greener. If the *in\_swt* equals to 10, skin pixels are set to be redder, less greener and less bluer while non-skin pixels remain original.





*in\_swt* = 10, skin pixels are redder

*in\_swt* = 9, skin pixels are bluer

The default case where the original R, G, B values are outputted will take effect if the cases above are not met. It makes sure every clock cycle the output registers are assigned some values.



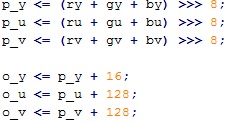
Default case: original video stream is shown

**5. Summary and suggestions**

From lab 4, firstly I got a roughly knowledge on how the FPGA board can be designed to do a specific kind of processing. Suppose there is a need for a specific work, the FPGA board can be designed to have particular components and layouts to fit the operation needed. In this lab 4, the input is the video stream which is taken at the present. Then the FPGA board is programmed to do the particular operation which is to analyze the pixels and change their values in a very short time.

Secondly, I have a deeper understanding about how the design can be pipelined to fit the requirement of high throughput. The maximum clock speed is limited by the slowest stage. While if the output is required to have a high throughput, we can pipeline the design by splitting it into several shorter stages. Thus, the length of a clock cycle can be shorter with higher clock speed and in a certain period time we can feed more data to the FPGA design.

Last but not least, before typing the code in Verilog, we should always have a full understanding about the structure of the design. During the lab I was stuck in a problem that if I separated the expressions of *p\_y*, *p\_u*, *p\_v* as:



It couldn’t detect the skin pixels correctly. Later I realized that the values used to determine the *skind* must be the final value in the right range while I was using the *p\_u* and *p\_v* without adding 128. So I think when doing the coding, we should be aware of the effect on not only the part we are editing, but also the other part of the whole project.

However, I spent much time on waiting the synthesis tool to generate the bit stream. So if we could shorten this time the lab will be more efficient. In addition, in the outcome there are many pixels are wrongly detect. I think if we adjust the detect algorithm so that it can be more accurate, the output will be better. In general, the lab helps me apply what I learn to the real task and have a deeper understanding of them. What’s more, the lab is quite interesting and I’m more willing to learn in the latter lectures.