EPC-Ambric Lab1

Name:…….

Email:…….

Reg ID:……...

Group No.:…….

Exercise 1- Single Object Color Space Conversion

Operation to perform

In this lab a single object (i.e. single processor) will be used to convert from one color space format to another color space format. The incoming stream of pixels will be in the RGB color space with one byte used for each color. The three bytes are packed into a word as shown below:

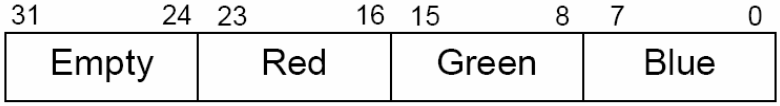

Figure 1.

Each pixel must first be converted from the RGB color space to the YUV color space. Use the equations provided below to perform that transformation:

Y = ((66*R + 129*G + 25*B + 128) >> 8) + 16 ……Eq.1
U = ((-38*R - 74*G + 112*B + 128) >> 8) + 128 ……Eq.2
V = ((112*R - 94*G - 18*B + 128) >> 8) + 128 ……Eq.3

Note that the input values for R, G and B are all 8 bit unsigned values and the resulting values for Y, U and V are also 8 bit unsigned values. To perform the calculation, the intermediate values may require up to 16 bits of precision.

At this point the YUV format is in the 4:4:4 color space (one value for each color for each pixel).

Convert from this format to the 4:2:2 color space. The 4:2:2 format will have one value of Y for every pixel and one value for each of U and V for every two pixels taken horizontally.

```
Y Y Y  Y Y Y
U   U    U
V   V    V
```

The downsampling that occurs for U and V only happens in the horizontal direction. There is no further change in the vertical direction. The filter is the same for both U and V. The filter function to use for this lab is:

$$U422[x] = (U444[2*x-1] + (U444[2*x] << 1) + U444[2*x+1] + 2) >> 2 \quad \text{....Eq.4}$$

The left edge of the picture should be handled by replicating the left pixel (i.e. U444[-1] = U444[0]). Because of the positioning of the U and V samples at the left edge of a pair of samples, there isn't an special edge condition for the right edge provided the width is an even number.

The output from this object will be the YUV image in 4:2:2 format with the packing for two pixels performed as shown:

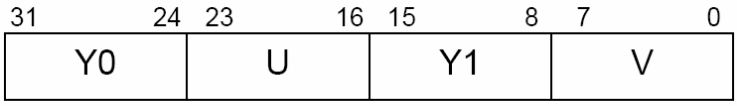

Figure 2.

Create the project

Your design will be built within a project. That project needs to be created.

- Start the aDesigner application by double clicking the icon on the desktop.
- Select a directory for your workspace
- When Eclipse opens select the "Go to the workbench" icon on the far right
- Right click in the "Package Explorer" window and perform File->New Project
- Expand the "Ambric" item and select "aDesigner Project"
- Select "Next" then give your project the name "ColorConvertLabs"
- Select "Finish" and your project will be created

Create a package

All files for your design (Java, aStruct and Assembly) need to be placed within packages. For this design we'll use a single package that will be called "colorConvert".

- Expand your project in the "Project Explorer" window
- Right click on "src" and perform New->Package
- Give the package the name "colorConvert" and select "Finish"

Create the Java code

This lab is implemented using a single object written in Java.

- Right click on the "colorConvert" package and perform New->Other…
- Expand "Ambric" and select "aDesigner aCompiler Class" then "Next"
- Give it the name "Convert" and select "Finish"

It will automatically create a shell for your Java code. Since there are several topics that have only been partially introduced that you are going to need to complete this Java code, a more detailed shell for you to use as the basis for your Java file is provided below:

```
package colorConvert;

import ajava.io.InputStream;
import ajava.io.OutputStream;

public class Convert {
	private int[] yArray = new int[2];
	private int[] uArray = new int[3];
	private int[] vArray = new int[3];
	private int width = 8;

public void run(InputStream<Integer> in, OutputStream<Integer> out) {
	// Process a line of video
	for (int pixel = 0; pixel < width; pixel += 2) {
		// Convert from RGB to YUV for 2 pixels
		for (int i=0; i < 2; i++) {
			// Get an RGB pixel value
			int rgb = in.readInt();

			// Extract its 3 component values (See Fig.2) and
			// Convert to YUV values according to Eq.1 to Eq.3

			//Store the intermediate values
			yArray[i] = y;
			uArray[i+1] = u;
			vArray[i+1] = v;
		}
```

```
            // Handle the special case of the left edge
            if (pixel == 0) {
                  uArray[0] = uArray[1];
                  vArray[0] = vArray[1];
            }

            // Convert from 4:4:4 to 4:2:2 (See Eq.4)

            // Format the data for output at channel (See Fig.2)
            out.writeInt(result);

            // Store away the latest U and V value read
            uArray[0] = uArray[2];
            vArray[0] = vArray[2];
      }
   }
}
```

The functionality for your code goes within the run() method. The syntax for your code is standard Java syntax. For the functionality that you'll be coding this syntax is also identical to the 'C' syntax.

Write the implementation of this object and save the file.

Create the aStruct file

The aStruct description for the Java object needs to be created.

- Right click on the "colorConvert" package and perform New->Other…
- Expand "Ambric" and select "aDesigner AStruct File" then "Next"
- Give it the name "Convert" and select "Finish"

It will automatically create a shell for your aStruct file. Since there are several topics that have only been partially introduced that you are going to need to complete this file, the complete code for this file is provided below:

```
package colorConvert;

interface Convert {
      inbound in;
      outbound out;
}

binding JConvert implements Convert {
      implementation "Convert.java";
}
```

Enter this content and save the file.

Create the design file

There must be a top level design file that has the “design” declaration. The design file can also include any other aStruct code. For this example create a design file that includes the “design” declaration and also has the aStruct code for the “Root” interface.

- Right click on the “colorConvert” package and perform New->Other…
- Expand “Ambric” and select “aDesigner Design File” then “Next”
- Give it the name “Lab1” and select “Finish”

It will automatically create a shell for your Design file. Since there are several topics that have only been partially introduced that you are going to need to complete this file, the complete code for this file is provided below:

```
package colorConvert;

import astruct.pcie.Vio;
import colorConvert.Convert;

interface Root {}
binding CRoot implements Root {
	Vio Vio = {
		numSources = 1,
		numSinks = 1
	};
	Convert Convert;

	channel c0 = {Vio.out[0], Convert.in};
	channel c1 = {Convert.out, Vio.in[0]};
}

design Lab1 {
	Root Root;
}
```

Enter this content and save the file.

Create the test files

The design will be tested using test vector files that provide the stimulus and check the response.

- Right click on the top level project and perform New->Folder
- Give it the name “test” and select “Finish”

- Right click on the “test” directory and perform New->File
- Give it the name “test.in” and select “Finish”
- Right click on the “test” directory and perform New->File
- Give it the name “test.out” and select “Finish”

The content for the input stimulus file (test.in) and the output result file (test.out) are as follows:

```
# test.in
# Line 0 (8 pixels)
00082B361
0007162E9
000392723
000B46278
00027490B
000518276
00078E990
000729734
# Line 1 (8 pixels)
000897972
000920B21
000234799
00023175E
000729742
000287723
000979723
000231123

# test.out
# Line 0 (8 pixels, 4 words)
095797573
0368C7C8C
04075727E
0B26C7F63
# Line 1 (8 pixels, 4 words)
07B7A3E94
04C992E84
080715A74
08662257F
```

Launching the simulation

The design can now be simulated to verify the functionality.

- Right click on your design file and perform Run As->Run…
- Select “Ambric launch design” and press the “New” button in the upper left
- Click on “Search” next to the “Design” selection and select your design “Lab1.design”
- Select the “Functional Simulation” radio button and choose “Simulate Java classes in ISS”
- Select the “Test Data” tab at the top
- Click “Add…” and click “Browse” on the “Stimulus” line
- Navigate to the “test.in” file and select “Open” and then “OK”
- Do the same thing for “test.out” but with the “Compare” radio button selected
- Finally click “Run” and that will start the simulation

If your simulation succeeds the console window will display something similar to:

```
Welcome to aSim 1.0.10
aSim total iteration count: 18
aSim completed with no errors
```

If the results don’t match the values in the response file the console window will display something similar to:

```
Welcome to aSim 1.0.10
Error: TestSink data value mismatch file=0x80715a84 channel=0x80715a74
inst=Root.Vio.sink0 position=7
Error: TestSink data value mismatch file=0x8662258f channel=0x8662257f
inst=Root.Vio.sink0 position=8
aSim total iteration count: 18
Error: aSim encountered 2 simulation errors.
```

Once aSim reports that it has “completed with no errors” this lab exercise is complete. Please show your results to the lab supervisor.