import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Replace 'your\_file.csv' with the actual file path

file\_path = 'ss1\_ALL.csv'

# Load the CSV file with 23 header lines

data\_1 = pd.read\_csv(file\_path, header=10)

data = data\_1.iloc[:-3]

data.drop(data.index[1])

#%% get Data info

sampling\_interval = data["TIME"][1]-data["TIME"][0]

MSPS = np.ceil((1/sampling\_interval)/1e6)

d = data.iloc[:, -8:].values# [:, ::-1] MSB Corrected not required

num\_samples = d.shape[0]

duration\_data\_us = num\_samples\*sampling\_interval\*1e6

#%% prepare clock

# u = yuv[0];

# y1 = yuv[1];

# v = yuv[2];

# y2 = yuv[3];

llc = data["CH2"].values

th = 1.5 # Threshold value

# Apply the threshold

clk = (llc > th).astype(int)

# Find the rising edges in the thresholded\_array

rising\_edges\_index = np.where(np.diff(clk) > 0)[0]

# Extract values from another\_array at the rising edge indexes

resampled\_CH8 = d[rising\_edges\_index,:]

t = data["TIME"][rising\_edges\_index]

CH8\_dec = np.packbits(resampled\_CH8, axis=1)

#%% extract pixel data from [ITU-R BT.656 Tx CONFIGURATION] 8 bit data

start\_y\_pos = 1

y = CH8\_dec[start\_y\_pos::2] #as 4:2:2 is rcvd

start\_cb\_pos = 0

cb = CH8\_dec[start\_cb\_pos::4] #as 4:2:2 is rcvd

start\_cr\_pos = 2

cr = CH8\_dec[start\_cr\_pos::4] #as 4:2:2 is rcvd

#%%

YCbCr = []

RGB = []

for i in np.arange(len(y)-1):

# print(i,i//2,i//2)

# https://en.wikipedia.org/wiki/YCbCr

Y = y[i]

Cb = cb[i//2]

Cr = cr[i//2]

YCbCr.append([Y, Cb, Cr])

#ITU-R BT.656 conversion factors https://techdocs.altium.com/display/FPGA/BT656+-+Color+Conversion

R = int(np.floor(Y + 1.402\*Cr))

G = int(np.floor(Y - 0.344\*Cb - 0.714\*Cr))

B = int(np.floor(Y + 1.772\*Cb))

RGB.append([R,G,B])

i += 1

YCbCr = np.array(YCbCr).reshape(y.shape[0]-1,3)

RGB = np.array(RGB).reshape(y.shape[0]-1,3)

fig, ax = plt.subplots(1, 1)

ax.plot(y[0::2], 'k')

ax.plot(cb, 'b')

ax.plot(cr, 'r')

plt.show()

#%%

image\_height = 1

image\_width = RGB.shape[0]

image = np.zeros((image\_height, image\_width, 3), dtype=np.uint8)

pixel\_line = RGB

# Create a figure and axis for the plot

fig, ax = plt.subplots(1, 1, figsize=(len(pixel\_line), 1))

# Display the line of pixels as an image

ax.imshow([pixel\_line], aspect='auto')

# Remove axis labels and ticks

ax.axis('off')

# Show the plot

plt.show()

#%%

# Constants

width = 720 # Width of the frame (pixels)

height = 576 # Height of the frame (pixels)

num\_bars = 8 # Number of grayscale bars

bar\_width = width // num\_bars # Width of each bar

# Create an empty frame (all black)

frame = np.zeros((height, width), dtype=np.uint8)

# Generate grayscale bars

for i in range(num\_bars):

# Calculate the brightness value for the bar (gradually increasing)

brightness = int((i / num\_bars) \* 255)

# Set the pixels in the current bar to the calculated brightness

frame[:, i \* bar\_width : (i + 1) \* bar\_width] = brightness

# Display the frame using Matplotlib

plt.figure()

plt.imshow(frame, cmap='gray', vmin=0, vmax=255)

plt.axis('off')

plt.show()

# Create R, G, and B matrices by replicating the grayscale frame

R\_matrix = frame.copy()

G\_matrix = frame.copy()

B\_matrix = frame.copy()

# Display the grayscale R, G, and B matrices

print("R Matrix:")

print(R\_matrix)

print("\nG Matrix:")

print(G\_matrix)

print("\nB Matrix:")

print(B\_matrix)