Lab 7 : Simple Image Processing Unit

Advisor: Lih-Yih Chiou
Speaker: John(張峻豪)
Date: 2018/04/25

Outline

- Introduction
- Architecture
- Image Format
- Grayscale
- Floyd–Steinberg dithering
- Components
- Boundary Cases
- Final Result
- Lab 7: Homework

Introduction

- Learn how to implement digital Image Processing Unit through Verilog code.
- Design a system to perform simple Image Processing Unit.
- We provide top module and sub modules skeleton, please follow our I/O ports specifications.

Architecture

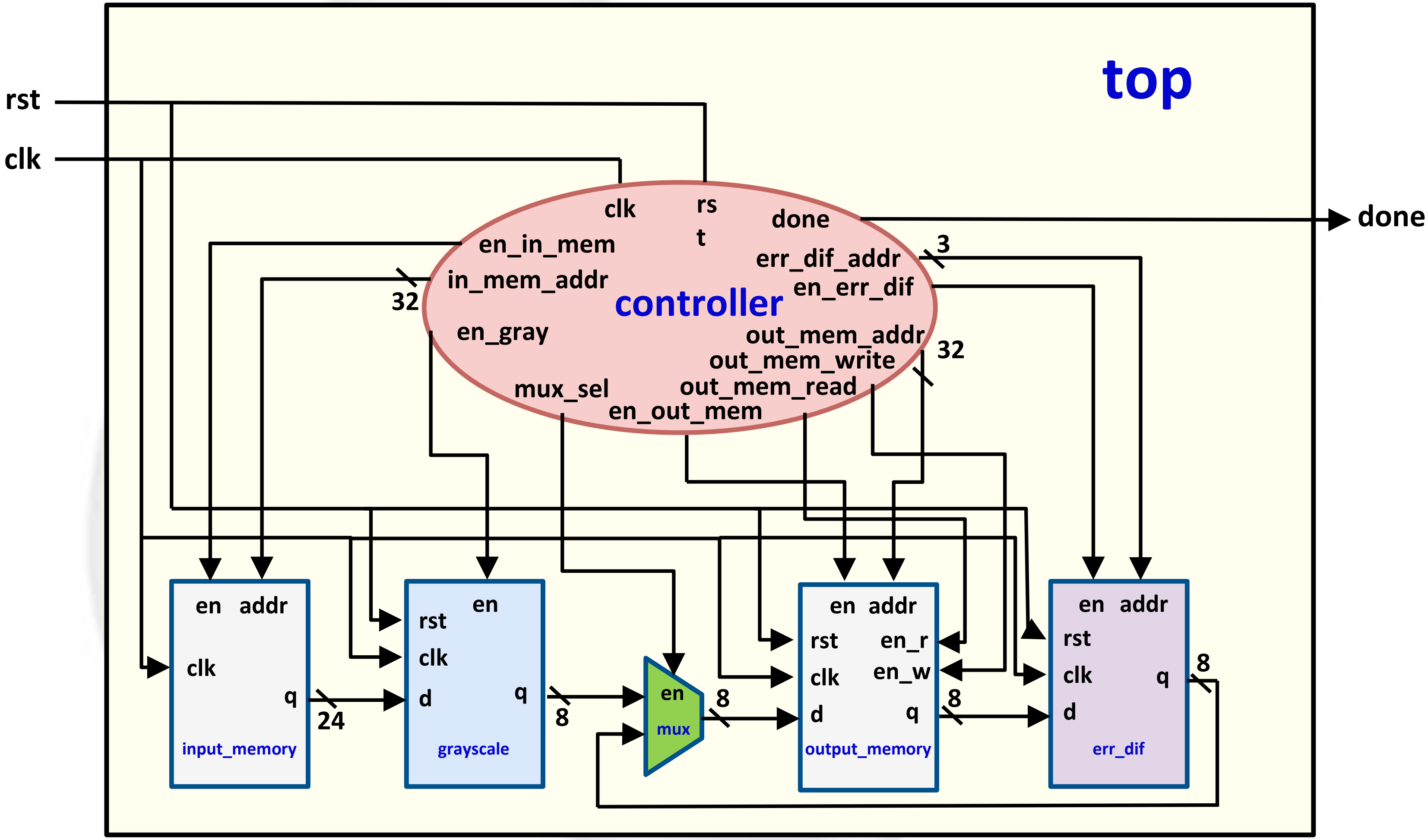


Image Format

- **RGB (Red, Green, Blue)**
  - ➔ Each pixel can be represented in the computer memory or interface as binary values for the red, green, and blue color components.
- **Current typical display adapters use 24 bits of information for each pixel.**
  - ➔ Each color has 8 bits (0-255)
  - ➔ Represent as (255, 0, 0)
  - ➔ In hexadecimal #FF0000
- **Total color**
  - ➔ **256 * 256 * 256 = 16,777,216**

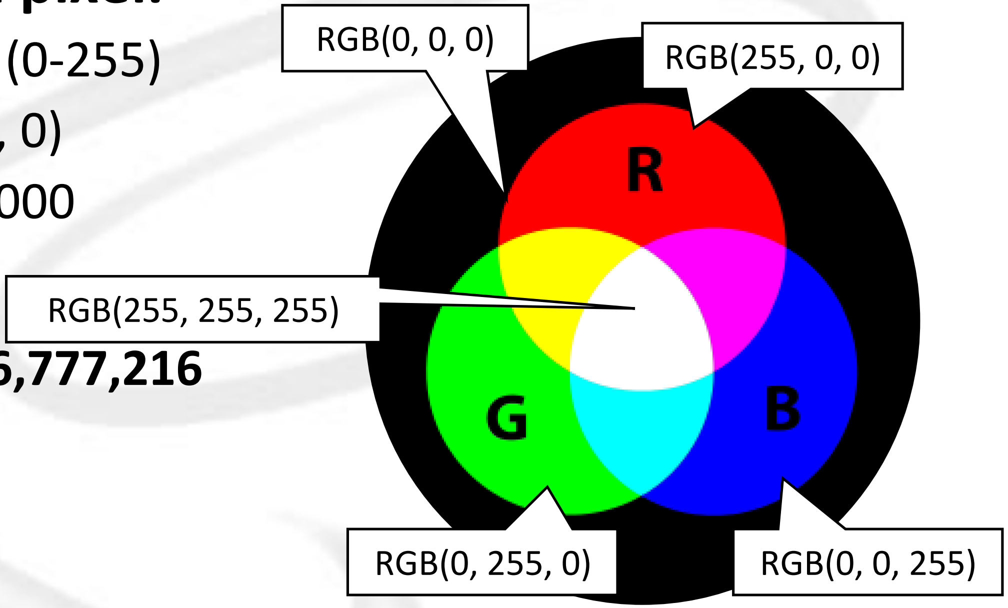
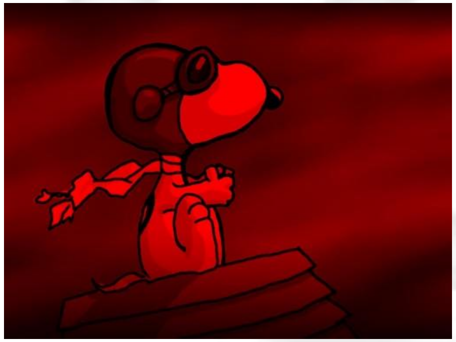


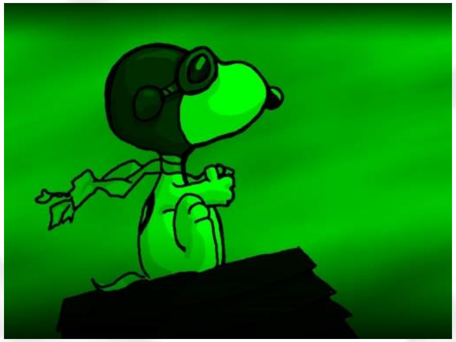
Image Format

- **Image decomposed into red, green and blue component**

Red component



Green component



Blue component

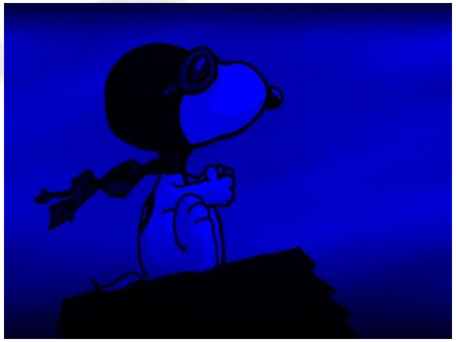
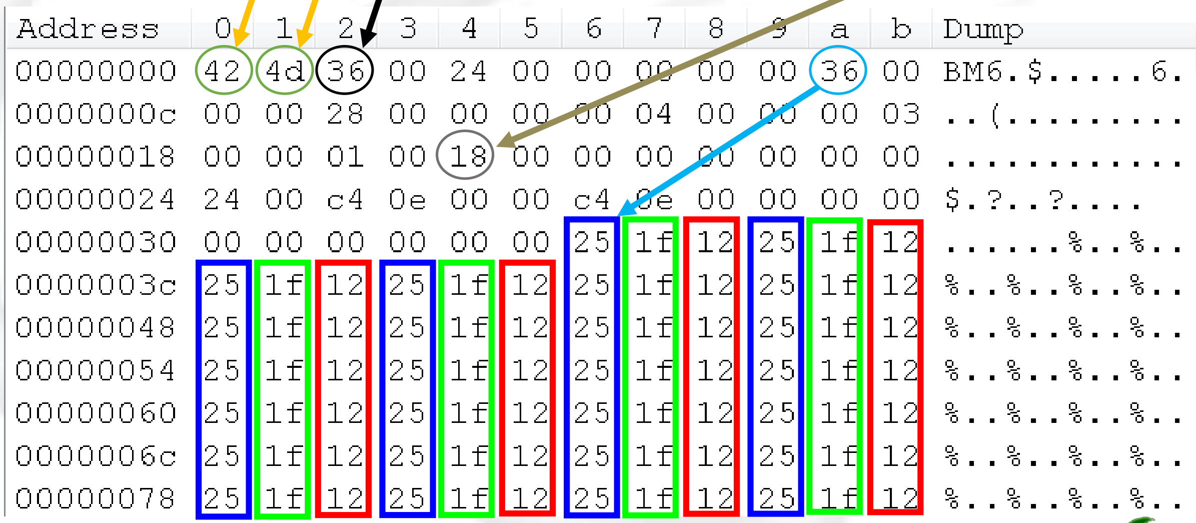


Image Format

- **Bitmap image file (.bmp)**

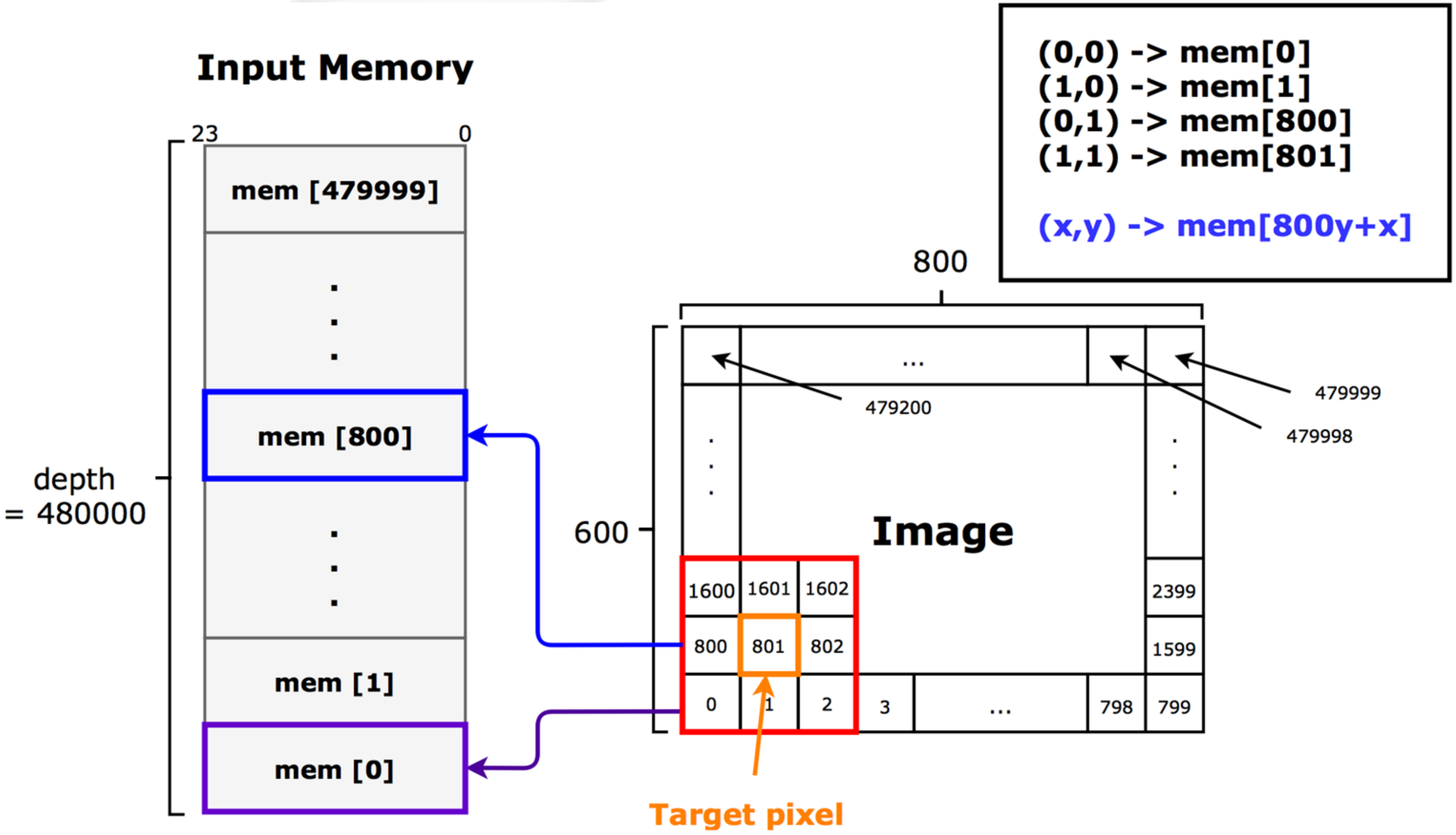
B M Size of BMP file (byte) The number of bits per pixel



```
Address   0  1  2  3  4  5  6  7  8  9  a  b  Dump
00000000  42 4d 36 00 24 00 00 00 00 00 36 00 BM6.$.....6.
0000000c  00 00 28 00 00 00 00 04 00 00 00 03 ..(.........
00000018  00 00 01 00 18 00 00 00 00 00 00 00 ............
00000024  24 00 c4 0e 00 00 c4 0e 00 00 00 00 $.?..?....
00000030  00 00 00 00 00 00 25 1f 12 25 1f 12 ......%..%..
0000003c  25 1f 12 25 1f 12 25 1f 12 25 1f 12 %..%..%..%..
00000048  25 1f 12 25 1f 12 25 1f 12 25 1f 12 %..%..%..%..
00000054  25 1f 12 25 1f 12 25 1f 12 25 1f 12 %..%..%..%..
00000060  25 1f 12 25 1f 12 25 1f 12 25 1f 12 %..%..%..%..
0000006c  25 1f 12 25 1f 12 25 1f 12 25 1f 12 %..%..%..%..
00000078  25 1f 12 25 1f 12 25 1f 12 25 1f 12 %..%..%..%..
```

Image Format

- Image(Here we take 800 * 600 picture for example)



(0,0) -> mem[0]
(1,0) -> mem[1]
(0,1) -> mem[800]
(1,1) -> mem[801]

(x,y) -> mem[800y+x]

Input Memory

23 0

mem [479999]

mem [800]

mem [1]

mem [0]

depth = 480000

800

600

Image

479200

479999

479998

1600 1601 1602 2399

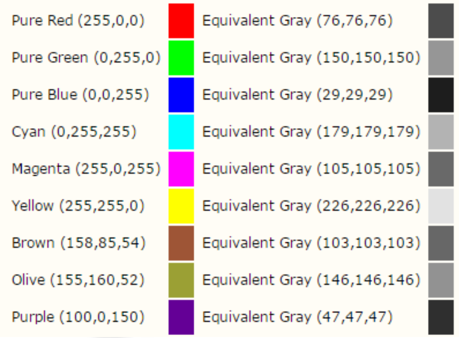
800 801 802 1599

0 1 2 3 … 798 799

Target pixel

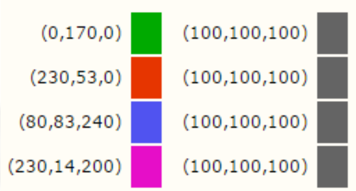
Grayscale

- How to turn RGB image into grayscale?
  - ➔ Suppose the RGB value of a pixel is (r, g, b)
  - ➔ The grayscale y=0.299r+0.587g+0.114b (0-255)
  - ➔ The pixel is now (y, y, y)
  - ➔ **For this Lab , y = 0.3125r + 0.5625g + 0.125b (0-255)



Pure Red (255,0,0)	■	Equivalent Gray (76,76,76)	■
Pure Green (0,255,0)	■	Equivalent Gray (150,150,150)	■
Pure Blue (0,0,255)	■	Equivalent Gray (29,29,29)	■
Cyan (0,255,255)	■	Equivalent Gray (179,179,179)	■
Magenta (255,0,255)	■	Equivalent Gray (105,105,105)	■
Yellow (255,255,0)	■	Equivalent Gray (226,226,226)	■
Brown (158,85,54)	■	Equivalent Gray (103,103,103)	■
Olive (155,160,52)	■	Equivalent Gray (146,146,146)	■
Purple (100,0,150)	■	Equivalent Gray (47,47,47)	■

- Can you convert a grayscale value back to an RGB color code?
  - ➔ NO!



(0,170,0)	■	(100,100,100)	■
(230,53,0)	■	(100,100,100)	■
(80,83,240)	■	(100,100,100)	■
(230,14,200)	■	(100,100,100)	■

Floyd–Steinberg error diffusion

- For every pixel: (scan from the left to the right, top to bottom)
  - ➔ New data = $\begin{cases} 255, \text{old data} \geq 128 \\ 0, \text{old data} < 128 \end{cases}$
  - ➔ Error diffusion = Old data – New data

$$\begin{bmatrix} & & * & \frac{7}{16} & \ldots \\ \ldots & \frac{3}{16} & \frac{5}{16} & \frac{1}{16} & \ldots \end{bmatrix}$$

  - ➔ Right pixel = Right pixel data + $\frac{7}{16}$ Error diffusion
  - ➔ Lower left pixel = Lower left pixel data + $\frac{3}{16}$ Error diffusion
  - ➔ Lower pixel = Lower pixel data + $\frac{5}{16}$ Error diffusion
  - ➔ Lower right pixel = Lower right pixel data + $\frac{1}{16}$ Error diffusion

Study and Comparison of Various Image Edge Detection Techniques
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.927&rep=rep1&type=pdf

Floyd–Steinberg error diffusion

- For example:(after grayscale)

$$\begin{bmatrix} & & * & \frac{7}{16} & \ldots \\ \ldots & \frac{3}{16} & \frac{5}{16} & \frac{1}{16} & \ldots \end{bmatrix}$$

Output_mem

…	…	…
…	159	162
1	200	120

Output_mem

…	…	…
…	255	120
0	170	114

Output_mem

…	…	…	
…	255	120	…
0	170	114	…

Center pixel:
159 > 128 ➔ 255

Error = 159 – 255 = -96

Right pixel
162 + (– 96)*(7/16) = 120

Lower left pixel
1 + (– 96)*(3/16) = -17
(less than 0, replace by 0)

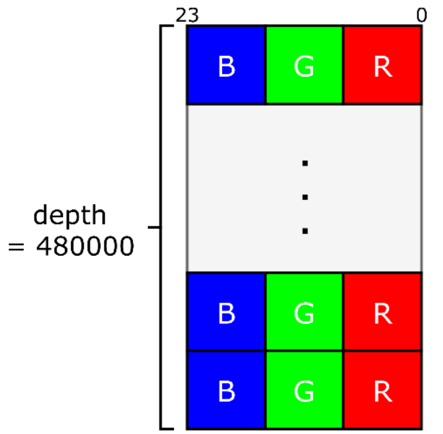
Lower pixel
200 + (– 96)*(5/16) = 170

Lower right pixel
120 + (– 96)*(1/16) = 114

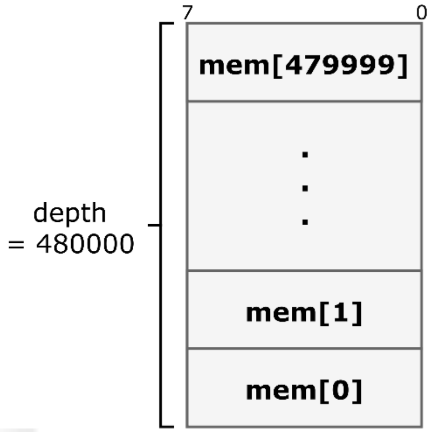
Components(Input / Output Memory)

- Input Memory
  - → Store pixels of the original image
  - → Memory depth : 800x600=480000
  - → Size per entry : 24-bit (B,G,R)
- Output Memory
  - → Store pixels of the processed image
  - → Memory depth : 800x600=480000
  - → Size per entry : 8-bit

Input Memory

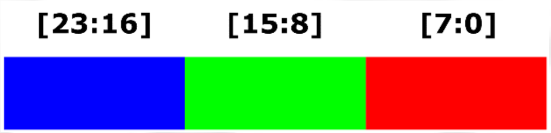


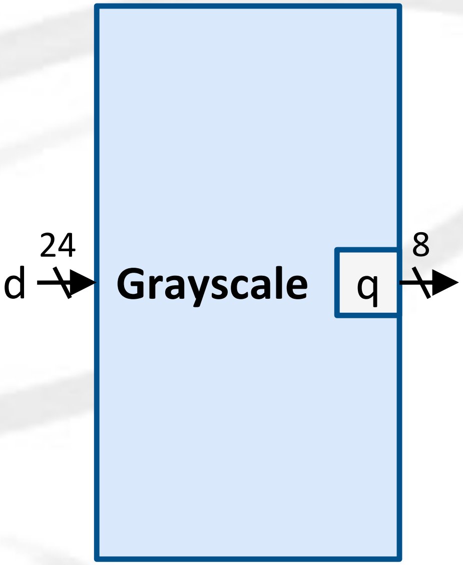
Output Memory



Components(Grayscale)

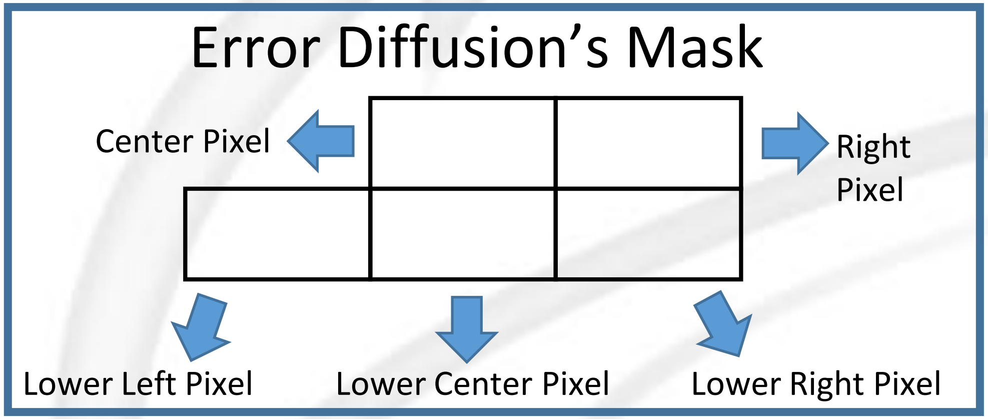
- Grayscale
  - → The grayscale operation y = 0.3125r + 0.5625g + 0.125b (0-255)
  - → **24-bit input** for pixel RGB value
  - → **8-bit output** for pixel grayscale value



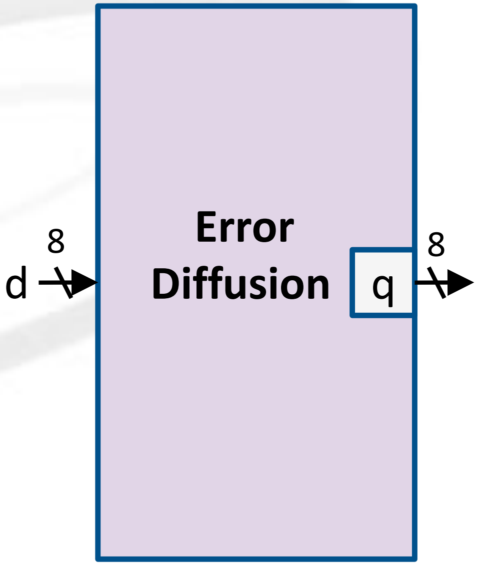


Components(Error Diffusion)

- **Error Diffusion**
  - → **There is only one 8-bits input/output in this project.**

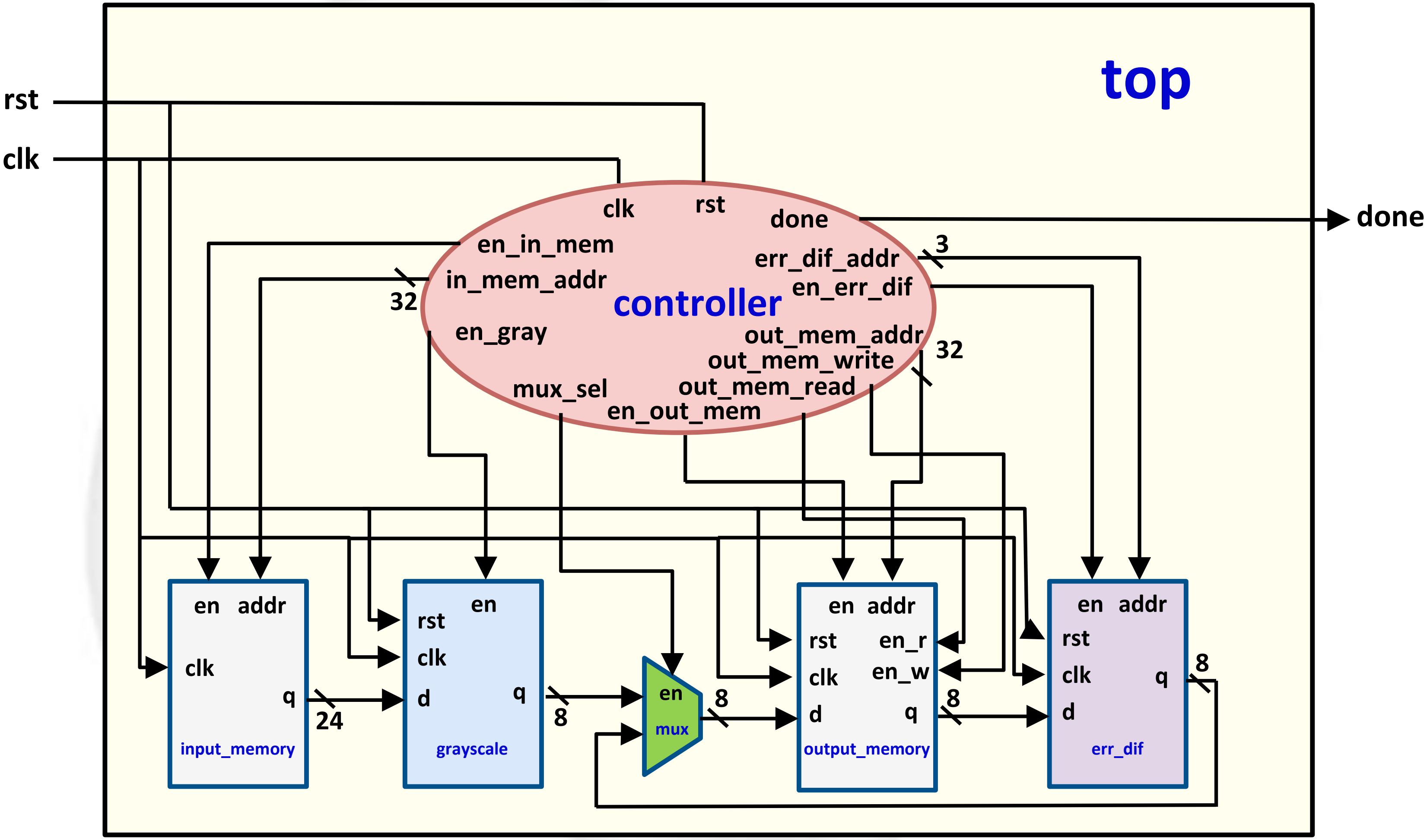


$$\begin{bmatrix} & & * & \frac{7}{16} & \cdots \\ \cdots & \frac{3}{16} & \frac{5}{16} & \frac{1}{16} & \cdots \end{bmatrix}$$



Components

- Controller , Mux .

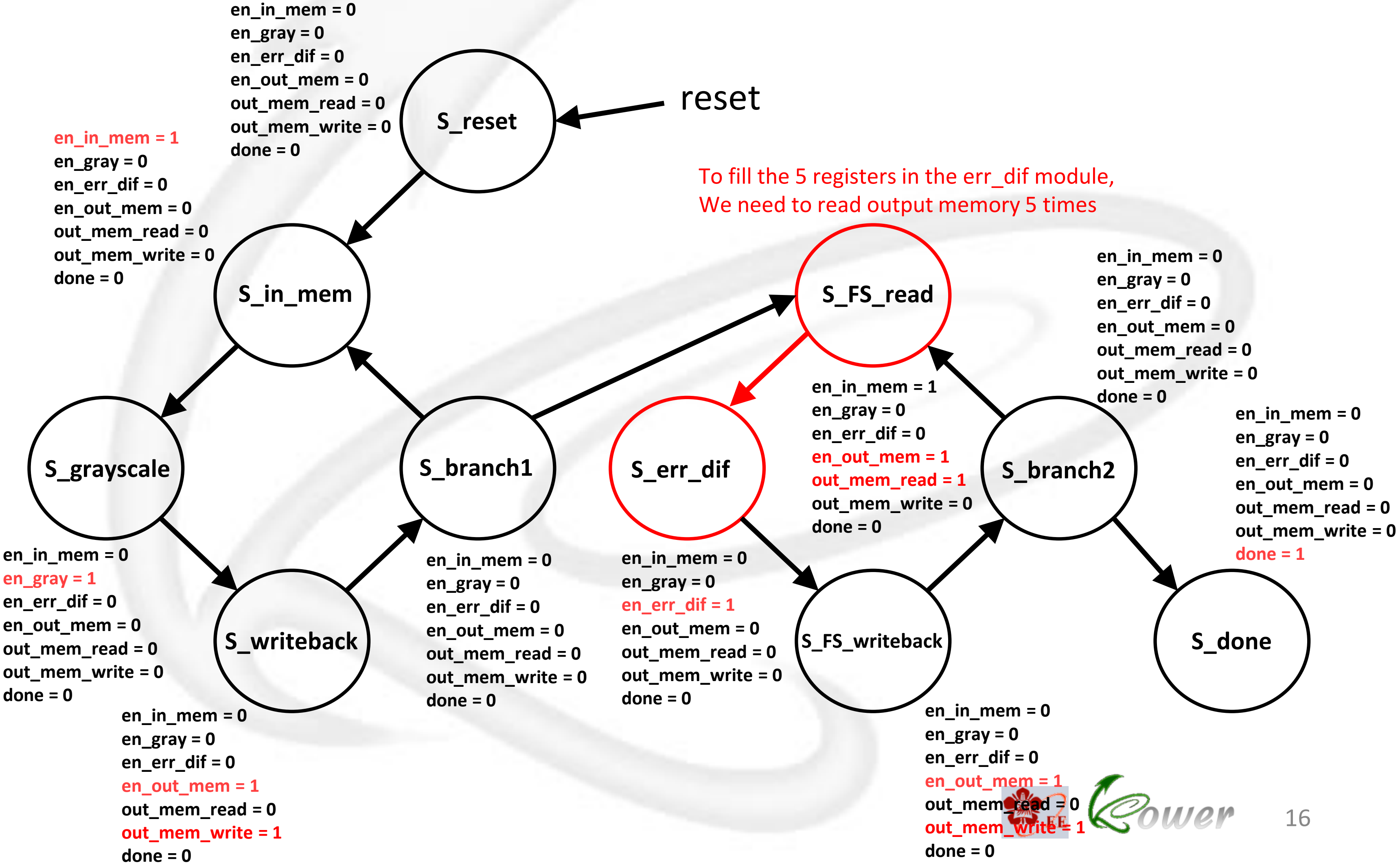


Components

- **Controller**

** You can design your own FSM in this lab if you want **

→ Control sub modules by a finite state machine (FSM)



en_in_mem = 0
en_gray = 0
en_err_dif = 0
en_out_mem = 0
out_mem_read = 0
out_mem_write = 0
done = 0

en_in_mem = 1
en_gray = 0
en_err_dif = 0
en_out_mem = 0
out_mem_read = 0
out_mem_write = 0
done = 0

To fill the 5 registers in the err_dif module,
We need to read output memory 5 times

en_in_mem = 0
en_gray = 0
en_err_dif = 0
en_out_mem = 0
out_mem_read = 0
out_mem_write = 0
done = 0

en_in_mem = 1
en_gray = 0
en_err_dif = 0
en_out_mem = 1
out_mem_read = 1
out_mem_write = 0
done = 0

en_in_mem = 0
en_gray = 0
en_err_dif = 0
en_out_mem = 0
out_mem_read = 0
out_mem_write = 0
done = 1

en_in_mem = 0
en_gray = 1
en_err_dif = 0
en_out_mem = 0
out_mem_read = 0
out_mem_write = 0
done = 0

en_in_mem = 0
en_gray = 0
en_err_dif = 0
en_out_mem = 0
out_mem_read = 0
out_mem_write = 0
done = 0

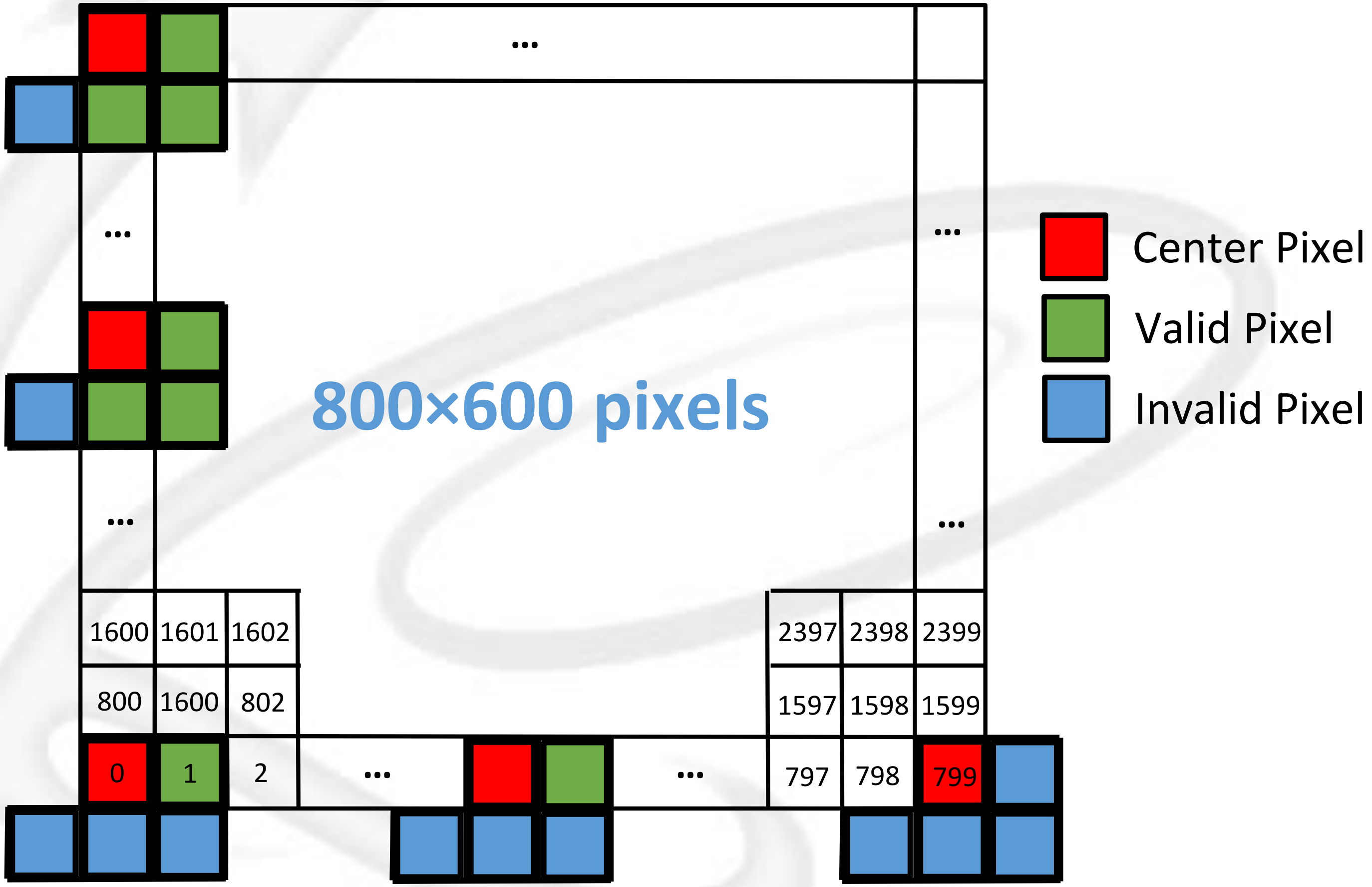
en_in_mem = 0
en_gray = 0
en_err_dif = 1
en_out_mem = 0
out_mem_read = 0
out_mem_write = 0
done = 0

en_in_mem = 0
en_gray = 0
en_err_dif = 0
en_out_mem = 1
out_mem_read = 0
out_mem_write = 1
done = 0

en_in_mem = 0
en_gray = 0
en_err_dif = 0
en_out_mem = 1
out_mem_read = 0
out_mem_write = 1
done = 0

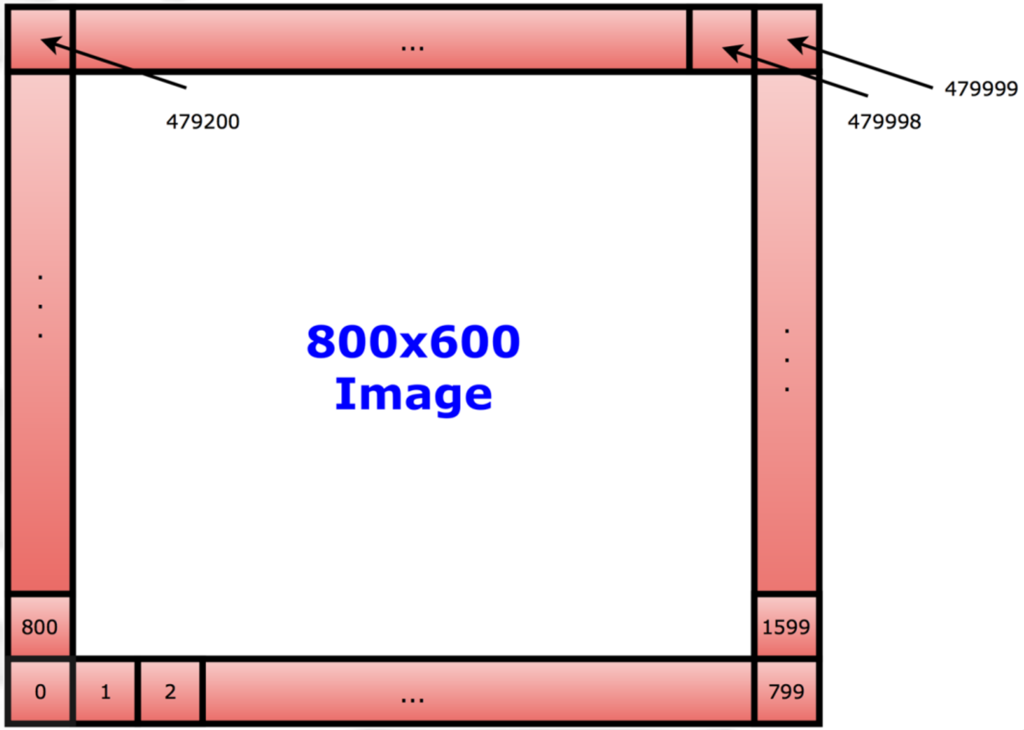
Boundary Cases

- **Boundary cases you need to consider**

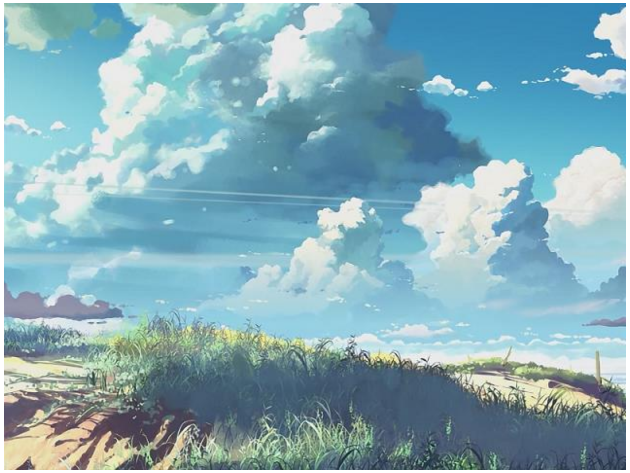
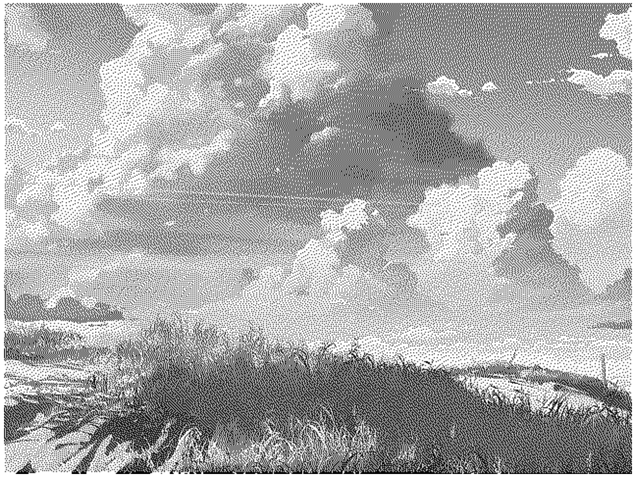
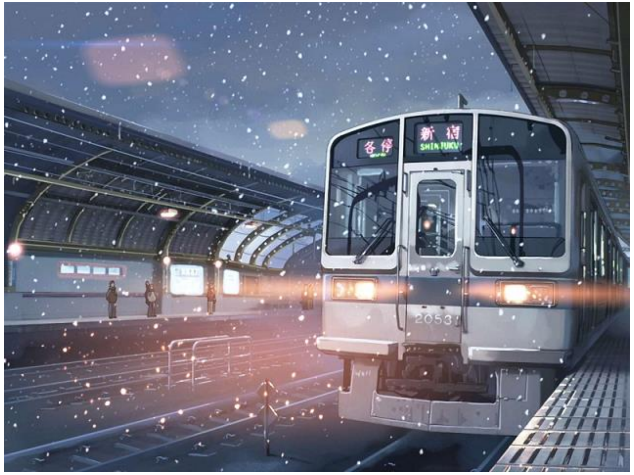
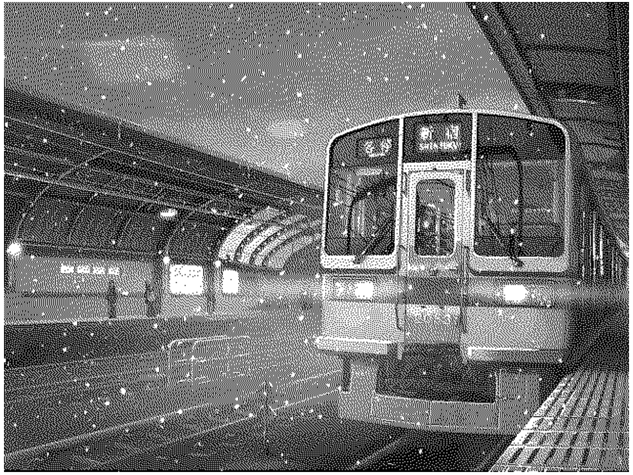


Boundary Cases

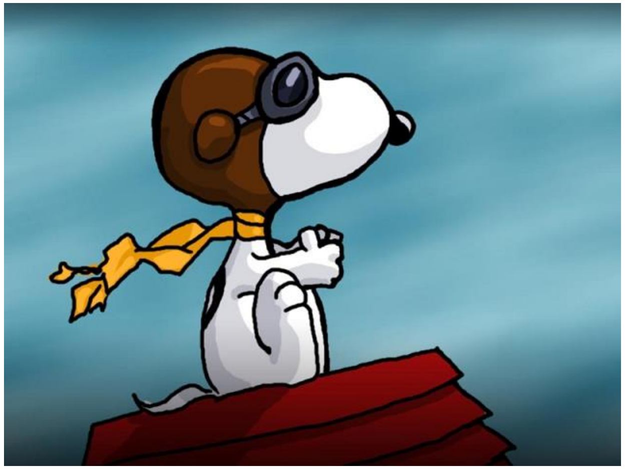
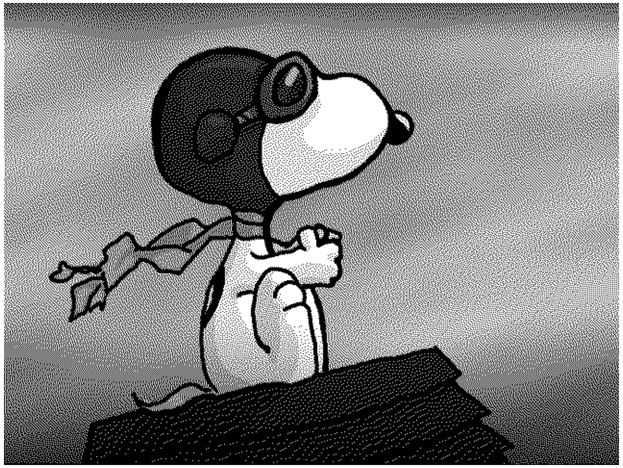
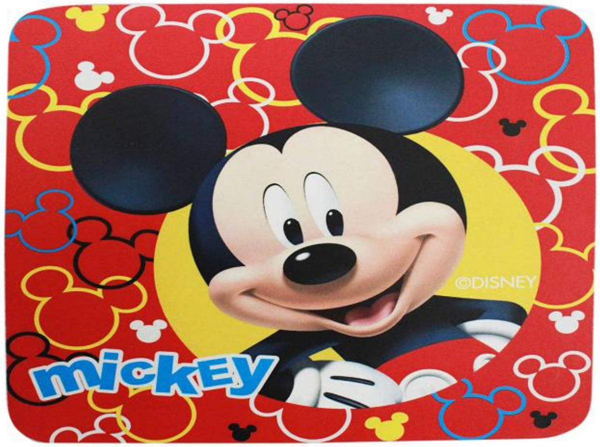
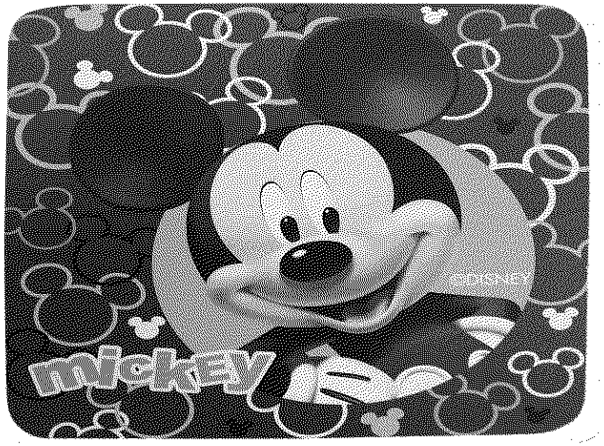
- How to deal with boundary cases?
  - → Focus on boundary address .
  - → How to define the address on boundary cases .



Final Resluts

Original Image	results
	
	

Final Resluts

Original Image	results
	
	

Lab 7 : Homework

- Two people for each group !
- Deadline : 05/13 (Sunday) 23:50
- Demo time : 05/14 (Monday) ~ 05/18 (Friday)
- Lab 7 Homework :
  - → Design a SIPU system based on Lab 7 's structure.
  - → Your design should be synthesized.
  - → You can use behavior modeling in this problem.
- % cp -r /home/user2/vlsi18/vlsi1890/Lab7 .
- ** Remember to fill out the Demo timetable on moodle

  So we can make sure that each group 's Demo time will not conflict. **

Thank you for your participation and attendance ! !