CS2261 Media Device Architecture

* Introducing in class activities and questions that may allow for extra credit later on
* Quiz on Wednesday

Notes:

* Reviewed the code for setPixel in Mode 4 with DMA
  + Void setPixel4 (int row, int col unsigned char colorIndex) {
    - Unsigned short pixelData = videoBuffer[OFFSET(row, col, SCREENWIDTH)/2];
    - if (col & 1) {
      * pixelData &= 0x00FF;
      * pixeldata |= (colorIndex<<8);
    - } else {
      * pixelData &= 0xFF00;
      * pixeldata |= (colorIndex);
    - }
  + videoBuffer[OFFSET(row, col, SCREENWIDTH)/2] = pixelData;
  + }
* Found the controlling short by using OFFSET. OFFSET tells us what index we need to use to get to a pixel, but in Mode4 we get a value that assumes video memory is single buffer. Divide by two, because recall that in Mode4 its two pixels to one short with 2 buffers.
  + Controlling short = index into video memory
  + Test last bit of a column to see if its odd or even
  + Even columns end in 0
  + Odd columns will end in 1, so to check if in an odd column, use (col & 1)
* Drawrect (in mode4 but no DMA) also poses a problem. In out original Mode 3 function, it took in an unsigned short color. But shorts are 16 bits and in Mode 4 we have 8 bits per pixel
  + 38400 shorts (16 bits/pixel) or 76800 **chars** (8 bits/pixel)
  + Have it take a char instead
* fillScreen has a similar problem, change it to take an unsigned char
  + another problem is that DMA does 16 or 32 bit transfers and inputting color is an 8bit index this time
  + but in fillScreen, all the pixels are taking in the same info
  + 16 bits worth of red on Mode3 is 8 bits worth of red twice in Mode4
  + We can have 2 values filled in at a time
    - **volatile unsigned short color = (index<<8) | index;**
    - creates a 16bit value with the 8bit value in twice
    - **make note of the use of |**

void fillScreen4(volatile unsigned char index) {

**volatile unsigned short color = (index<<8) | index;**

DMA[3].src = &color;

DMA[3].dst = videoBuffer;

DMA[3].cnt = (240\*160/2) | DMA\_SOURCE\_FIXED | DMA\_ON;

}

* flipPage()
  + switch to the alternate screen
  + Recall, one buffer will be drawn on while the other is displayed and then switch when that one side is finished drawing
  + Must also keep track of which one is being drawn on so you don’t display it and cause a glitch, or accidentally try to draw on the one being displayed
  + This is done using pointers
  + Just switch the address referenced by the videoBuffer variable (x6000000 <-> x600A000)

void flipPage() {

if (REG\_DISPCTL & BUFFER1FLAG) //which buffer are we displaying? This checks buffer1

{

REG\_DISPCTL &= ~BUFFER1FLAG;/\* Clear flag (display BUFFER0) \*/

videoBuffer = BUFFER1;/\* Start drawing on (writing to) BUFFER1 \*/

} else { not buffer1 so buffer2

REG\_DISPCTL |= BUFFER1FLAG;/\* Set flag (display BUFFER1) \*/

videoBuffer = BUFFER0; /\* Draw on BUFFER0 \*/

}

}

* Switching between modes
  + Can switch between modes, maybe go to Mode3 for more color with still images and mode4 for animation and faster paced stuff
* Setting palette
  + Use USENTI
* drawRect with DMA:
  + DMA moves chunks of 16-bits or 32-bits
    - Each pixel is 8 bits, and depending one where and the size of the rectangle we want to draw, we can run into some problems.
    - Say we want to rectangle to start drawing at pixel 1. Pixel one is sharing space with short 0 in video memory. How do we access just pixel 1??
  + Solution: setPixel() of the hanging pixel and DMA the even sized rest
  + Use setPixel for pixel 1, and DMA for pixels that are even length that we don’t need to just set one pixel sharing space with two

void drawRect3(int row, int col, int height, int width, unsigned short color) {

int r;

u16 color = index<<8 | index;

DMA[3].src = &color;

for (r = 0; r < height; r++) {

DMA[3].dst = &videoBuffer[OFFSET(row+r, col, 240)/2];

DMA[3].cnt = width/2 | DMA\_SOURCE\_FIXED | DMA\_ON;

}

}