EECS150 - Digital Design
Lecture 18 – Line Drawing Engine

November 1, 2011

Elad Alon
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www-inst.eecs.berkeley.edu/~cs150

Announcements

- Project checkpoint #3 due this week
- Checkpoint #4 due in two weeks
  - Intermediate check point due in one week
  - More details on the web soon

Project Notes

- Offer: GSIs will be available to look at your code with you today in office hours/lab
- Make sure you "check-off" even if your design isn't fully functional
  - Note that in this case the only thing we can really give partial credit for are your **tests**…

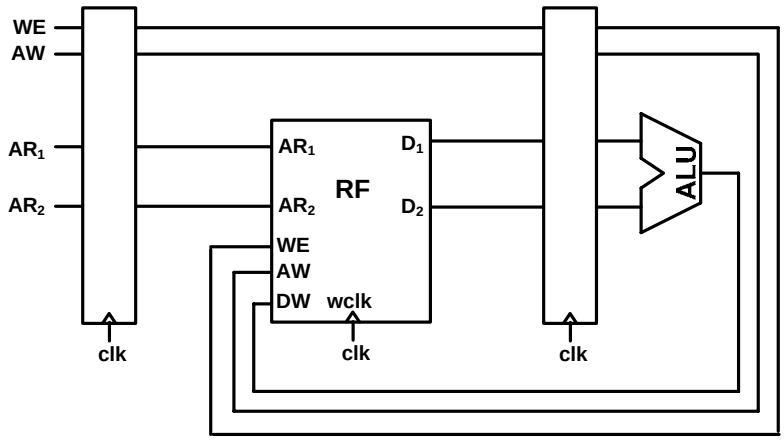
Modularity and Verification

- You are dealing with a **complex** design
- You therefore need to **decompose** your design into "simple" sub-elements
- You also need to **verify** that each of these sub-elements works as desired
  - I.e., unit testing matters!
  - And yes, writing the tests could easily take you twice (or more) as long as writing the module itself

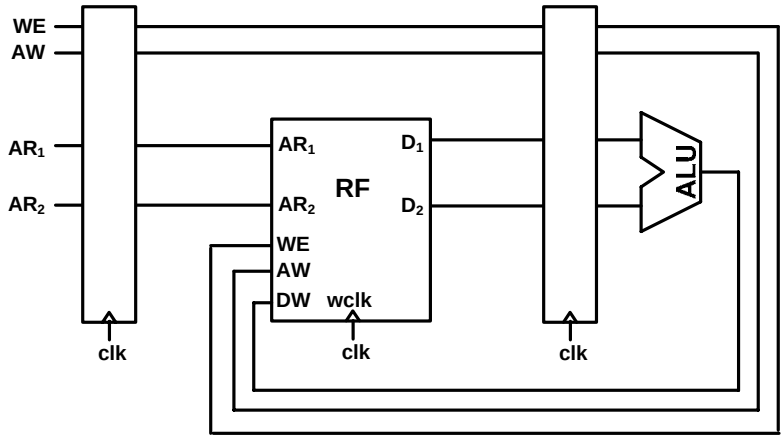
Additional Project Tips

- Use "make report" to look at the hardware you actually got
  - In particular, double check that you got the latches you wanted (not any more or any less)
- Look at the block diagrams you drew
  - Make sure you understand how everything is working together
  - And where the synchronous boundaries are
  - Midterm problem #4 was there for a reason…

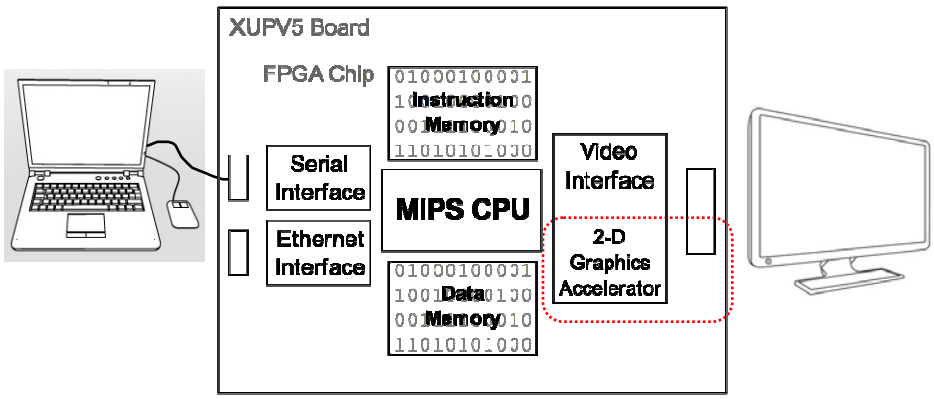
Midterm Problem 4 Review



Midterm Problem 4 Review



MIPS150 Video Subsystem



- Gives software ability to display information on screen.
- Equivalent to standard graphics cards:
- Processor can directly write the display bit map
- 2D Graphics acceleration

Graphics Software

"Clearing" the screen - fill the entire screen with same color

Remember Framebuffer base address: `0x8000_0000`

Size: `1024 x 768`

```
clear:  # a0 holds 4-bit pixel color
        # t0 holds the pixel pointer
        ori     $t0, $0, 0x8000          # top half of frame address
        sll     $t0, $t0, 16             # form framebuffer beginning address
        # t2 holds the framebuffer max address
        ori     $t2, $0, 768             # 768 rows
        sll     $t2, $t2, 12             #  * 1K pixels/row * 4 Bytes/address
        addu    $t2, $t2, $t0            # form ending address
        addiu   $t2, $t2, -4             # minus one word address
        #
        # the write loop
L0:     sw      $a0, 0($t0)              # write the pixel
        bneq    $t0, $t2, L0             # loop until done
        addiu   $t0, $t0, 4              # bump pointer
        jr      $ra
```

How long does this take? What do we need to know to answer?

How does this compare to the frame rate?

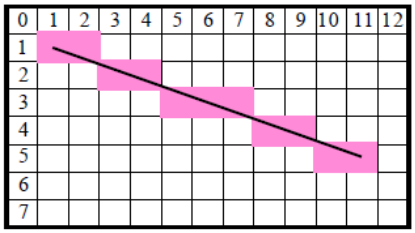
Optimized Clear Routine

Amortizing the loop overhead.

```
clear:
        .
        .
        .
        # the write loop
L0:     sw      $a0, 0($t0)              # write some pixels
        sw      $a0, 4($t0)
        sw      $a0, 8($t0)
        sw      $a0, 12($t0)
        sw      $a0, 16($t0)
        sw      $a0, 20($t0)
        sw      $a0, 24($t0)
        sw      $a0, 28($t0)
        sw      $a0, 32($t0)
        sw      $a0, 36($t0)
        sw      $a0, 40($t0)
        sw      $a0, 44($t0)
        sw      $a0, 48($t0)
        sw      $a0, 52($t0)
        sw      $a0, 56($t0)
        sw      $a0, 60($t0)
        bneq    $t0, $t2, L0             # loop until done
        addiu   $t0, $t0, 64             # bump pointer
        jr      $ra0
```

What's the performance of this one?

Line Drawing



From (x_0,y_0) to (x_1,y_1)

Line equation defines all the points:

$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

For each x value, could compute y, with: $\frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$

then round to the nearest integer y value.

Slope can be precomputed, but still requires floating point * and + in the loop: slow or expensive!

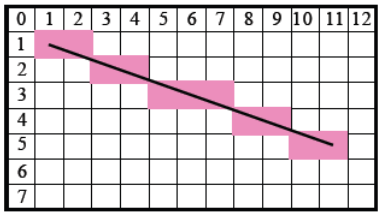
Bresenham Line Drawing Algorithm

Developed by Jack E. Bresenham in 1962 at IBM.
"I was working in the computation lab at IBM's San Jose development lab. A Calcomp plotter had been attached to an IBM 1401 via the 1407 typewriter console. …"

- Computers of the day, slow at complex arithmetic operations, such as multiply, especially on floating point numbers.
- Bresenham's algorithm works with integers and without multiply or divide.
- Simplicity makes it appropriate for inexpensive hardware implementation.
- With extension, can be used for drawing circles.

Line Drawing Algorithm

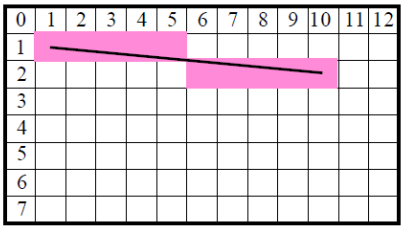
This version assumes: $x_0 < x_1$, $y_0 < y_1$, slope =< 45 degrees



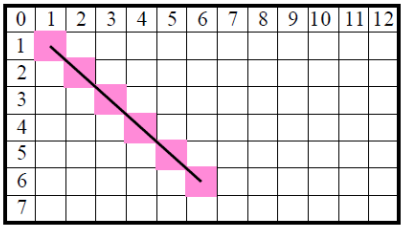

```
function line(x0, x1, y0, y1)
   int deltax := x1 - x0
   int deltay := y1 - y0
   int error := deltax / 2
   int y := y0
   for x from x0 to x1
      plot(x,y)
      error := error - deltay
      if error < 0 then
         y := y + 1
         error := error + deltax
```

Note: error starts at deltax/2 and gets decremented by deltay for each x, y gets incremented when error goes negative, therefore y gets incremented at a rate proportional to deltax/deltay.

Line Drawing, Examples

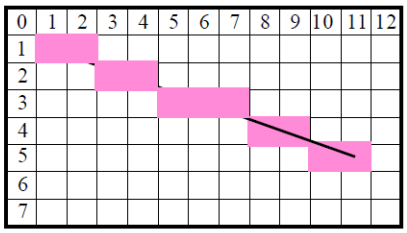


deltay = 1 (very low slope). y only gets incremented once (halfway between x0 and x1)



deltay = deltax (45 degrees, max slope). y gets incremented for every x

Line Drawing Example




```
function line(x0, x1, y0, y1)
   int deltax := x1 - x0
   int deltay := y1 - y0
   int error := deltax / 2
   int y := y0
   for x from x0 to x1
      plot(x,y)
      error := error - deltay
      if error < 0 then
         y := y + 1
         error := error + deltax
```

(1,1) -> (11,5)

deltax = 10, deltay = 4, error = 10/2 = 5, y = 1

x = 1: plot(1,1)
error = 5 - 4 = 1

x = 2: plot(2,1)
error = 1 - 4 = -3
y = 1 + 1 = 2
error = -3 + 10 = 7

x = 3: plot(3,2)
error = 7 - 4 = 3

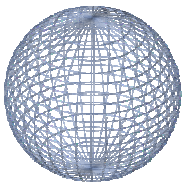
x = 4: plot(4,2)
error = 3 - 4 = -1
y = 2 + 1 = 3
error = -1 + 10 = 9

x = 5: plot(5,3)
error = 9 - 4 = 5

x = 6: plot(6,3)
error = 5 - 4 = 1

x = 7: plot(7,3)
error = 1 - 4 = -3
y = 3 + 1 = 4
error = -3 + 10 -= 7

C Version



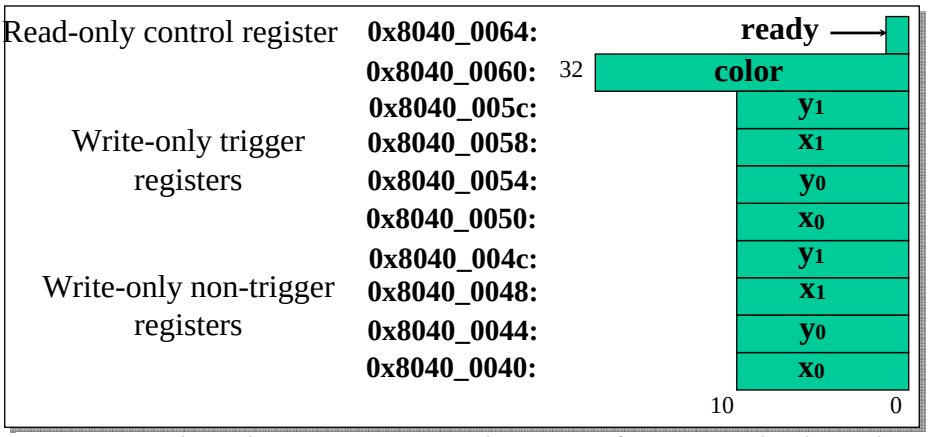
```
#define SWAP(x, y) (x ^= y ^= x ^= y)
#define ABS(x) (((x)<0) ? -(x) : (x))

void line(int x0, int y0, int x1, int y1) {
  char steep = (ABS(y1 - y0) > ABS(x1 - x0)) ? 1 : 0;
  if (steep) {
    SWAP(x0, y0);
    SWAP(x1, y1);
  }
  if (x0 > x1) {
    SWAP(x0, x1);
    SWAP(y0, y1);
  }
  int deltax = x1 - x0;
  int deltay = ABS(y1 - y0);
  int error = deltax / 2;
  int ystep;
  int y = y0
  int x;
  ystep = (y0 < y1) ? 1 : -1;
  for (x = x0; x <= x1; x++) {
    if (steep)
      plot(y,x);
    else
      plot(x,y);
    error = error - deltay;
    if (error < 0) {
      y += ystep;
      error += deltax;
    }
  }
}
```

Modified to work in any quadrant and for any slope.

What's needed to do it in hardware?

Goal is one pixel per cycle. Pipelining might be necessary.

Hardware Implementation Notes



- CPU initializes line engine by sending pair of points and color value to use. Writes to 0x8040_005* trigger engine.
- Framebuffer has one write port - Shared by CPU and line engine. Priority to CPU - Line engine stalls when CPU writes.