

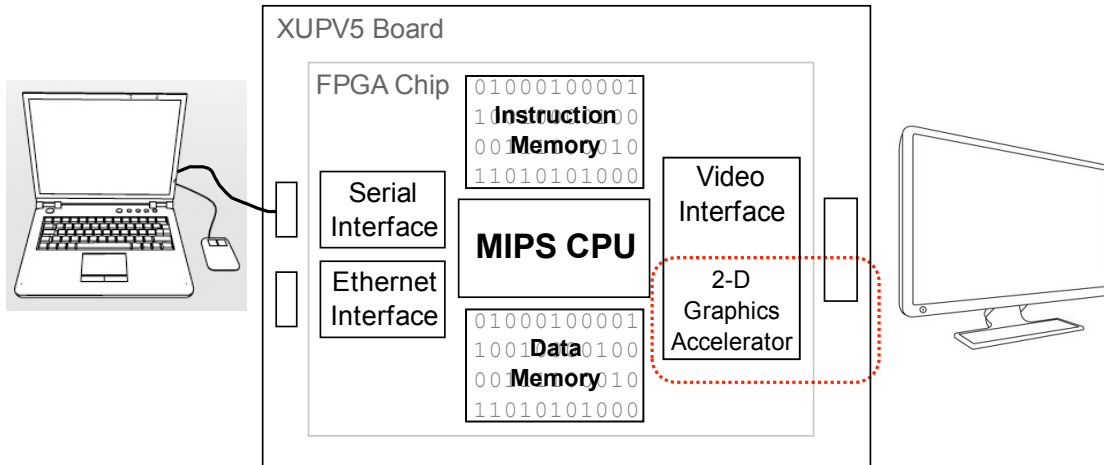
**EECS150 - Digital Design**  
**Lecture 15 - Project Description,**  
**Part 5**

March 8, 2011  
John Wawrzynek

**Announcements**

- Exam in lab tomorrow evening 6pm.

# 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?

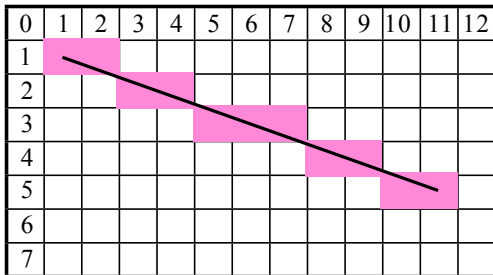
How does this compare to the frame rate?

# Optimized Clear Routine

```
clear:
    .           Amortizing the loop overhead.
    .
    .
    # 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     $ra
```

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.

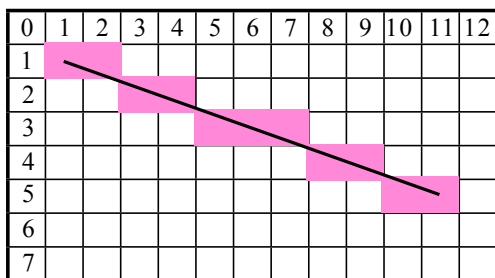
"I was working in the computation lab at IBM's San Jose development lab. A Calcomp plotter had been attached to an IBM 1401 via the 1407 typewriter console. ...



- 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

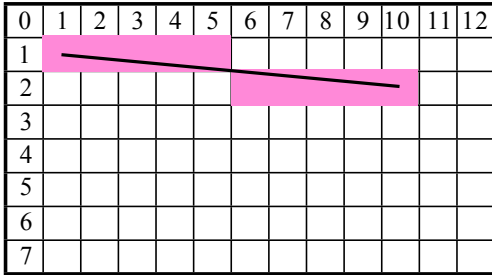
This version assumes:  $x_0 < x_1$ ,  $y_0 < y_1$ , slope  $\leq 45$  degrees



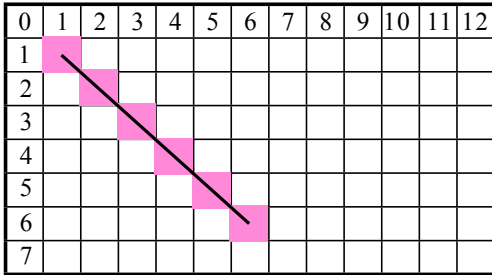
```
function line(x0, x1, y0, y1)
  int deltax := x1 - x0
  int deltay := y1 - y0
  int error := deltax / 2
  int y := y0
  for x from x0 to x1
    plot(x,y)
    error := error - deltax
    if error < 0 then
      y := y + 1
    error := error + deltax
```

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

# Line Drawing, Examples

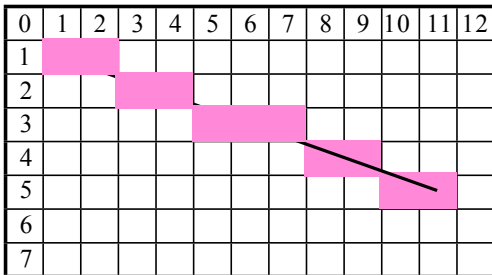


deltay = 1 (very low slope).  
y only gets incremented  
once (halfway between x0  
and x1)



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

# Line Drawing Example



(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

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

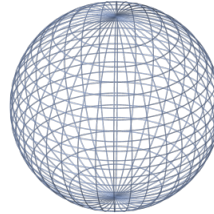
x = 6: plot(6,3)  
error = 5 - 4 = 1

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

y = 3 + 1 = 4  
error = -3 + 10 = 7

```
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
```

# 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;
        }
    }
}

```

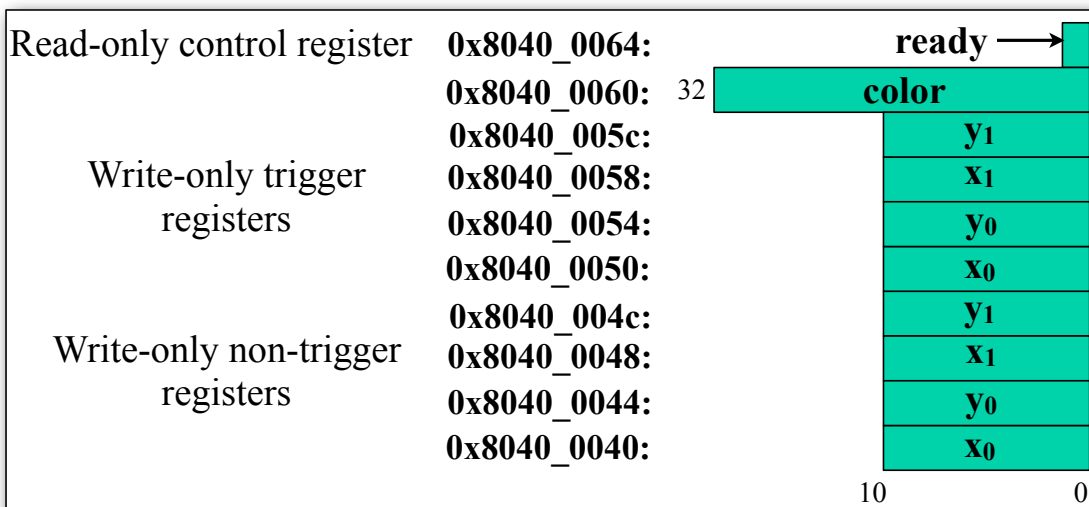
Modified to work in any quadrant and for any slope.

Estimate software performance (MIPS version)

What's needed to do it in hardware?

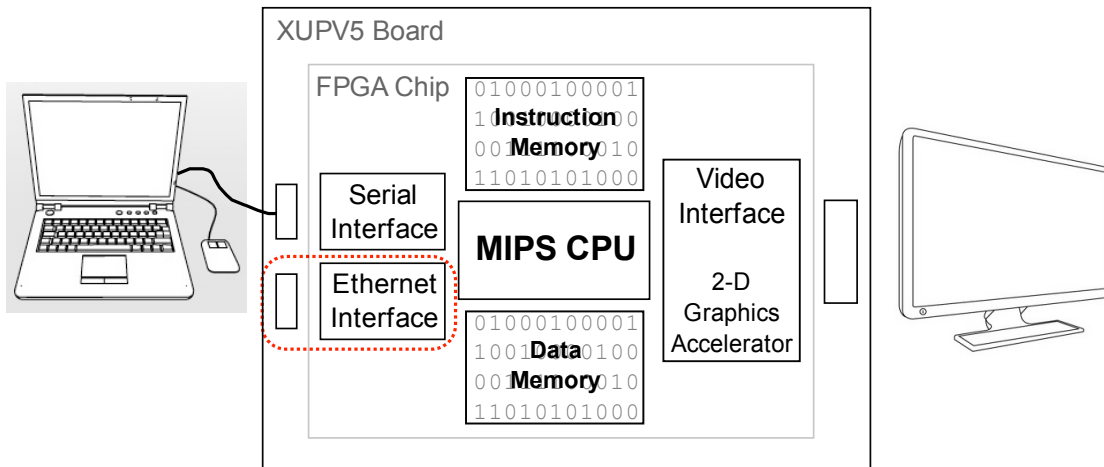
Goal is one pixel per cycle.  
Pipelining might be necessary.

## Hardware Implementation Notes



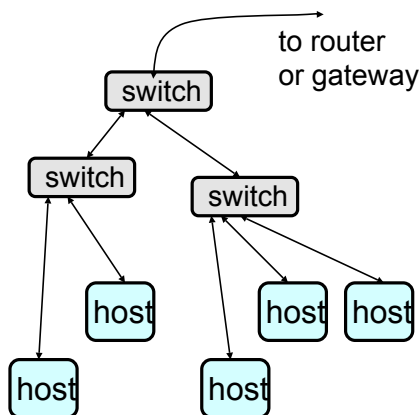
- CPU initializes line engine by sending pair of points and color value to use. Writes to 0x8040\_005\* trigger engine.
- Framebuffer has one write port - Shared by CPU and line engine. Priority to CPU - Line engine stalls when CPU writes.

# MIPS150 Project Block Diagram



- Ethernet interface allows connection to a LAN.
- Important for high-speed data-transfer.
- How much faster is it than Serial interface?

## Local Area Network (LAN) Basics



- Most Ethernet implementations these days are “switched” (point to point connections between switches and hosts, no contention or collisions).
- Information travels in variable sized blocks, called Ethernet Frames, each frame includes preamble, header (control) information, data, and error checking. We usually call these *packets*.

Preamble (8 bytes)	MAC header	Payload	CRC
-----------------------	---------------	---------	-----

- A LAN is made up physically of a set of switches, connections (wired or wireless), and hosts. Routers and gateways provide connectivity out to other LANs and to the internet.
- Ethernet defines a set of standards for data-rate (10/100Mbps, 1/10Gbps), and signaling to allow switches and computers to communicate.
- Preamble is a fixed pattern used by receivers to synchronize their clocks to the data.
- Link level protocol on Ethernet is called the Medium Access Control (MAC) protocol. It defines the format of the packets.

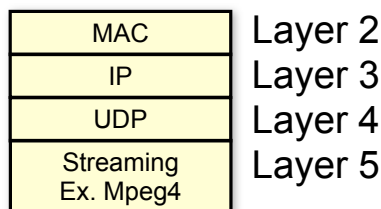
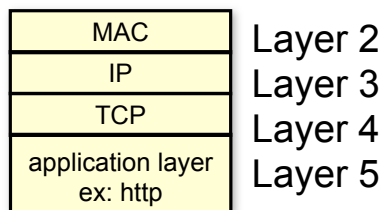
# Ethernet Medium Access Control (MAC)



- MAC protocol encapsulates a payload by adding a 14 byte header before the data and a 4-byte cyclic redundancy check (CRC) after the data.
- A 6-byte **destination address**, specifies either a single recipient node (unicast mode), a group of recipient nodes (multicast mode), or the set of all recipient nodes (broadcast mode).
- A 6-byte **source address**, is set to the sender's globally unique node address. Its common function is to allow address learning which may be used to configure the filter tables in switches.
- A 2-byte **type** field, identifies the type of protocol being carried (e.g. 0x0800 for IP protocol).
- The **CRC** provides error detection in the case where line errors result in corruption of the MAC frame. In most applications a frame with an invalid CRC is discarded by the MAC receiver.

## Protocol Stacks

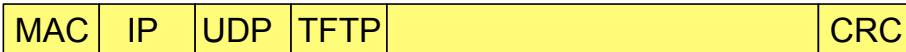
- Usual case is that MAC protocol encapsulates IP (internet protocol) which in turn encapsulates TCP (transport control protocol) with in turn encapsulates the application layer. Each layer adds its own headers (with MAC as the first).
- Other protocols exist for other network services (ex: printers).
- When the reliability features (retransmission) of TCP are not needed, UDP/IP is used. Gaming and other applications where reliability is provided at the application layer.





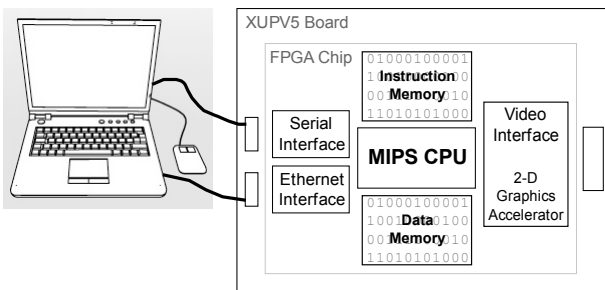
# CS150 File Transfers

- Our Boot Monitor program will use TFTP protocol.



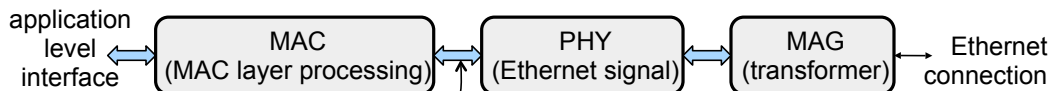
- "Trivial file transfer protocol" - per packet acks for reliability.
- Server (tftpd) will run on host machine (lab machine or your laptop).
- MIPS150 acts like client (built into the boot monitor).

```
> get 192.148.1.0 /tmp/myfile.exe 0x40000
> put 1024 0x80000 192.148.1.0 /tmp/dump.dat
```



- You must interface the FPGA MAC to the CPU
  - FIFO buffering, simple packet filtering
- You are welcome to write/prog a web server if interested.

## Standard Hardware-Network-Interface



Media Independent Interface (MII)

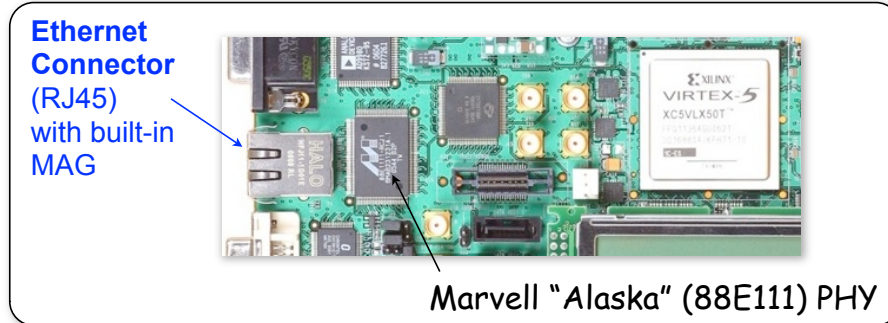
- Usually divided into three hardware blocks. (Application level processing could be either hardware or software.)
  - MAG. "Magnetics" chip is a transformer for providing electrical isolation.
  - PHY. Provides serial/parallel and parallel/serial conversion and encodes bit-stream for Ethernet signaling convention. Drives/receives analog signals to/from MAG. Recovers clock signal from data input.

- MAC. Media access layer processing. Processes Ethernet frames: preambles, headers, computes CRC to detect errors on receiving and to complete packet for transmission. Buffers (stores) data for/from application level.
- Application level interface
  - Could be a standard bus (ex: PCI)
  - or designed specifically for application level hardware.
- MII is an industry standard for connection PHY to MAC.

Virtex 6 has basic MAC block on chip. XUP board has PHY and MAG.

# XUP Board

- Virtex-5 FPGA has basic tri-mode MAC (10/100/1000 Mbps) block on chip.
- XUP board has PHY and MAG.



- You will need to interface the MAC to your processor.
  - Handshaking circuits (polling interface)
  - Packet buffering
  - MAC header creation and filtering