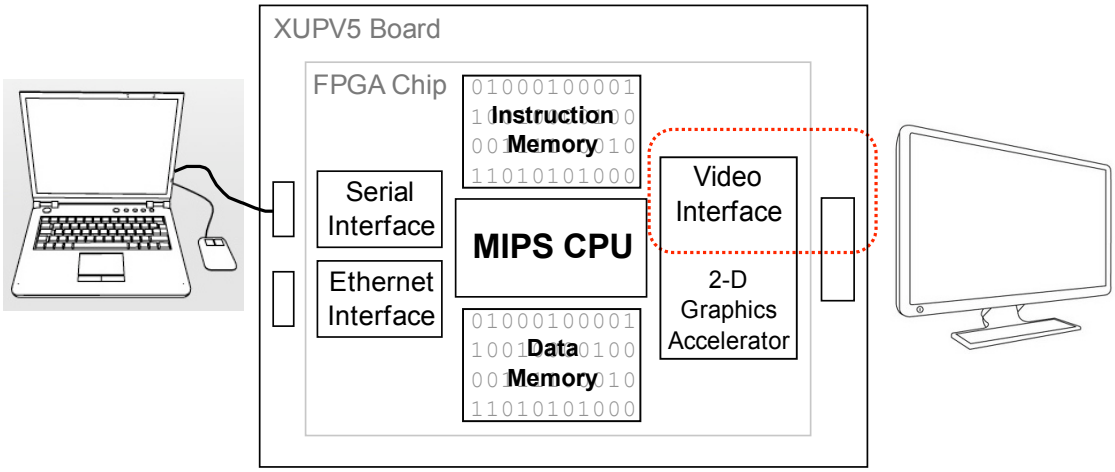
EECS150 - Digital Design

Lecture 16 - Project Description, Part 5

March 11, 2010

John Wawrzynek

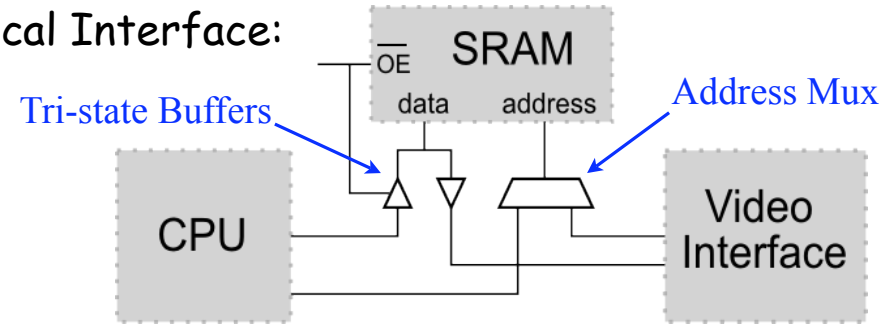
MIPS150 Video Subsystem



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

Video Interface Details

- Physical Interface:



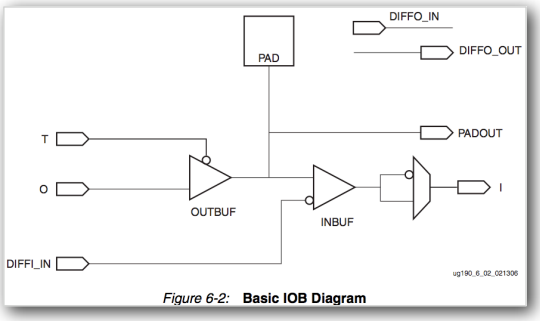
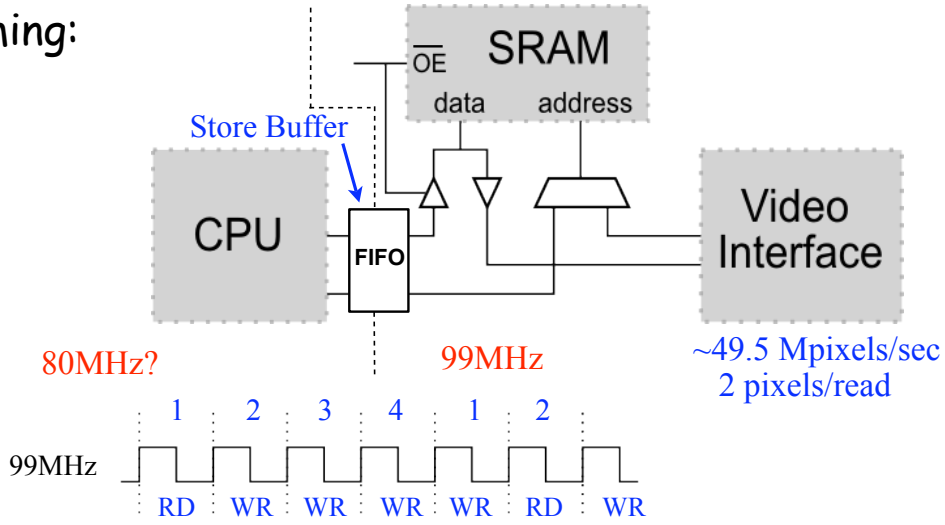


Figure 6-2: **Basic IOB Diagram**

Xilinx I/O buffer

Video Interface Details

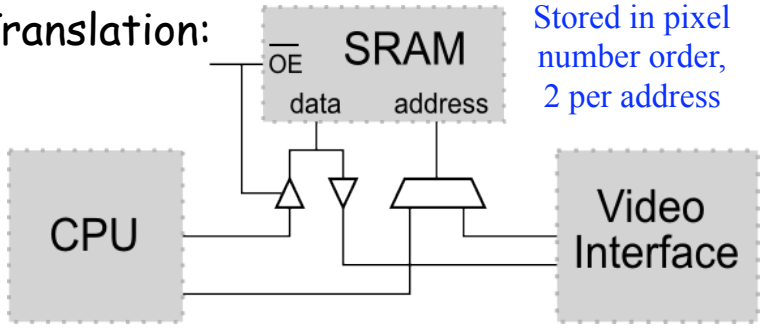
- Timing:



- All CPU frame buffer writes go through FIFO (and crosses clock domain boundary).
- Store Buffer writes to SRAM 3/4 SRAM cycles (4/4 during retrace?).
- Can Store Buffer fill up? If so, need to stall CPU. What if CPU runs at lower clock rate?

Video Interface Details

- Address Translation:



Stored in pixel number order, 2 per address

Random writes using frame buffer addresses: Y[9:0], X[9:0]

Reads in pixel number order: 0 ... 480000

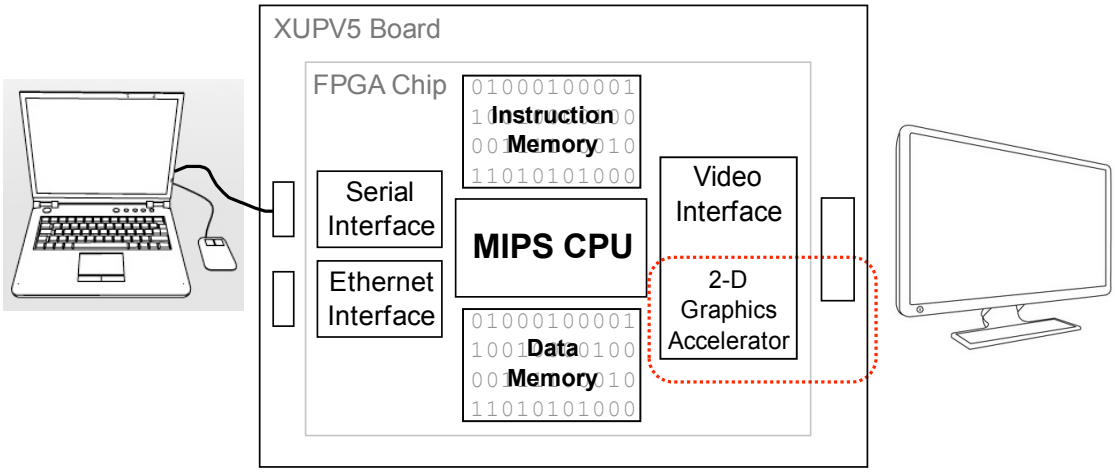
CPU writes need translation to convert from 20-bit frame buffer address to 19-bit SRAM address:

PN = X + 800*Y

How to do this on FPGA? 800 = 0x320 = 512 + 256 + 32

How do we write a single pixel? Use SRAM "Byte Write Enables"

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: **800 x 600 (for simplicity assume 1024 x 600)**

```
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, 600            # 600 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?

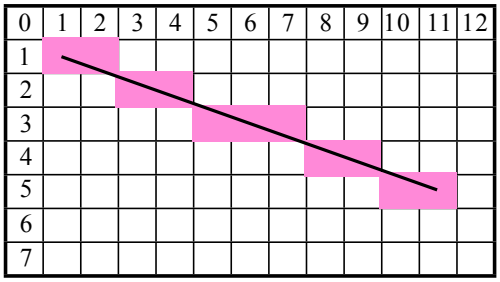
How does this compare to the frame rate?

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

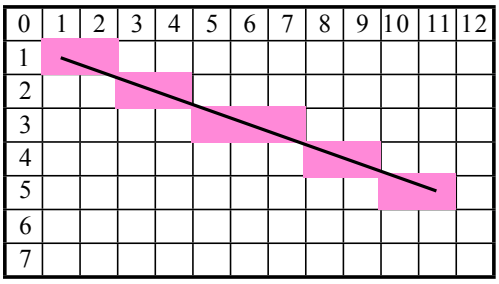
Developed by Jack E. Bresenham in 1962 at IBM.

"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

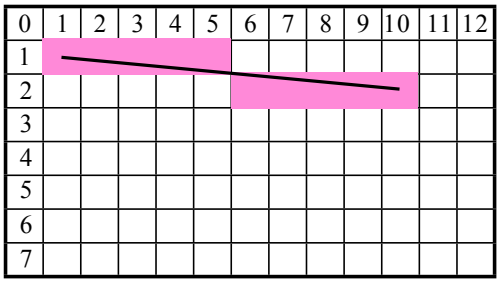
This version assumes: $x_0 < x_1$, $y_0 < y_1$, slope =< 45 degrees



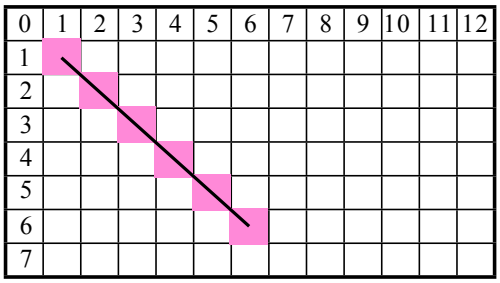

```
function line(x0, x1, y0, y1)
   int deltax := x1 - x0
   int deltay := y1 - y0
   int error := deltax / 2
   int y := y0
   for x from x0 to x1
      plot(x,y)
      error := error - deltay
      if error < 0 then
         y := y + 1
         error := error + deltax
```

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

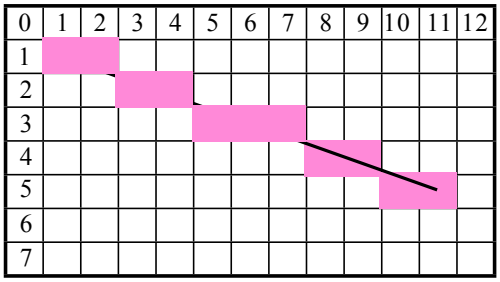


deltay = 1 (very low slope). y only gets incremented once (halfway between x0 and x1)



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

Line Drawing Example



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

```
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
```

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

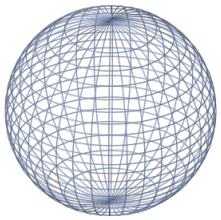
x = 4: plot(4,2)
error = 3 - 4 = -1
y = 2 + 1 = 3
error = -1 + 10 = 9

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;
    }
  }
}
```

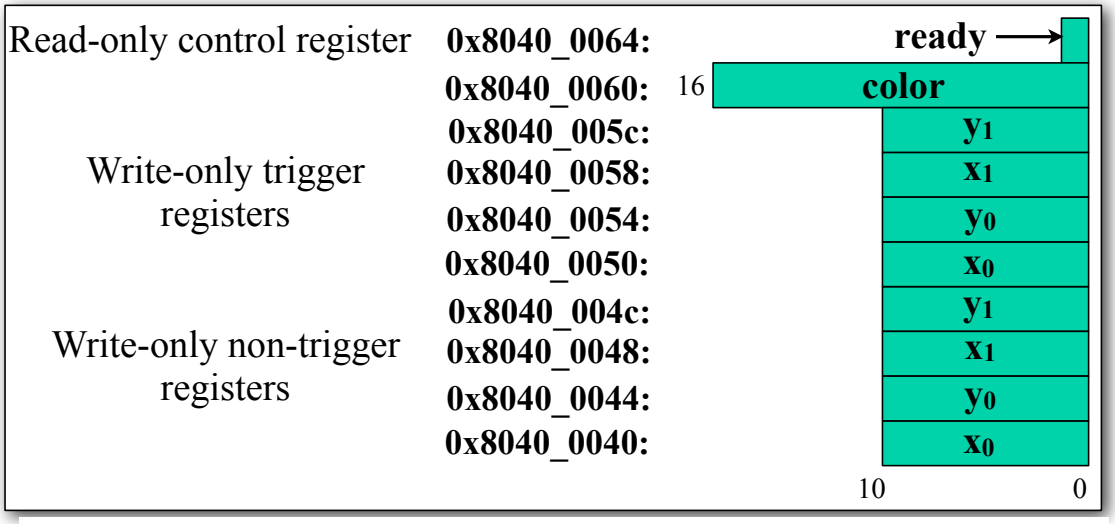
Modified to work in any quadrant and for any slope.

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

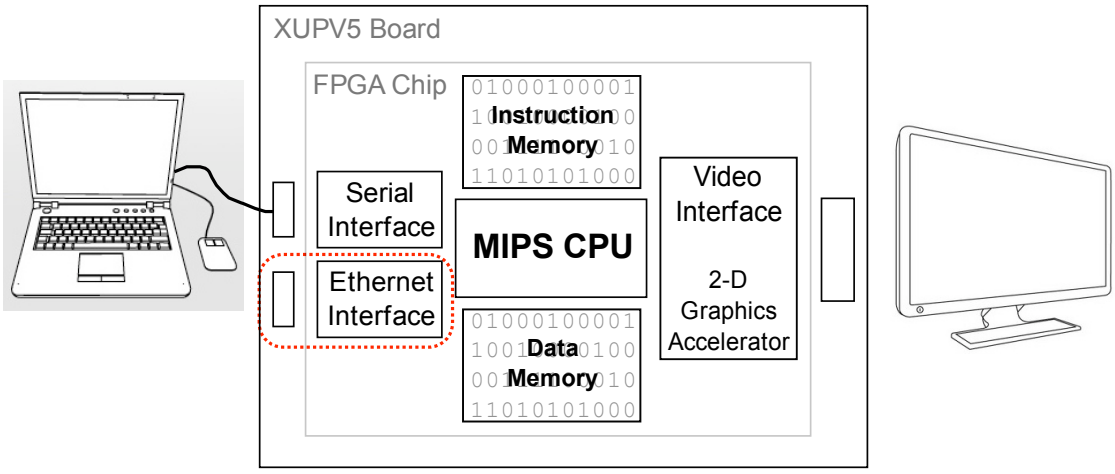


- CPU initializes line engine by sending pair of points and color value to use. Writes to 0x8040_005* trigger engine.
- Framebuffer has one write port - Shared by CPU and line engine. Priority to CPU - Line engine stalls when CPU writes.

Survey Results

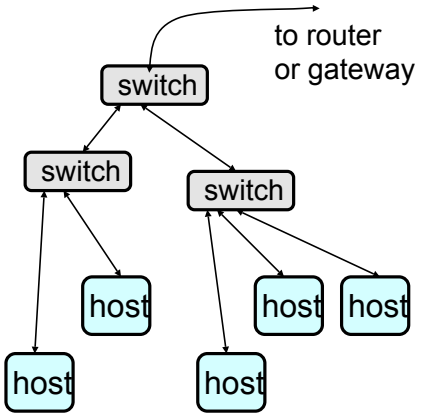
- Summary (high-level bits only):
  - GSIs are doing a great job!
  - Frustration about grades not online.
  - Lab Lectures not so useful.
  - Drawing figures for homework is a drag.
- What we will do in response:
  - GSIs: more of the same.
  - Online grades: will work on getting all grades online ASAP (and keeping up with it).
  - Lab Lectures: After quizzes, a couple of announcements, then optional Q/A session.
  - Scanner in 199 Cory, maybe in lab later.

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

Local Area Network (LAN) Basics

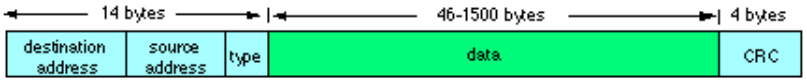


- A LAN is made up physically of a set of switches, connections (wired or wireless), and hosts. Routers and gateways provide connectivity out to other LANs and to the internet.
- Ethernet defines a set of standards for data-rate (10/100Mbps, 1/10Gbps), and signaling to allow switches and computers to communicate.
- 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

- 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.

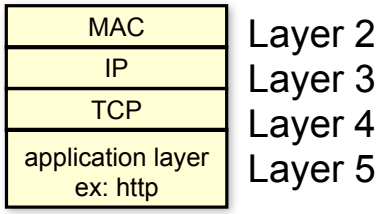
Ethernet Medium Access Control (MAC)

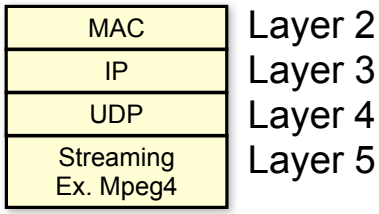


- MAC protocol encapsulates a payload by adding a 14 byte header before the data and a 4-byte cyclic redundancy check (CRC) after the data.
- A 6-byte **destination address**, specifies either a single recipient node (unicast mode), a group of recipient nodes (multicast mode), or the set of all recipient 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.





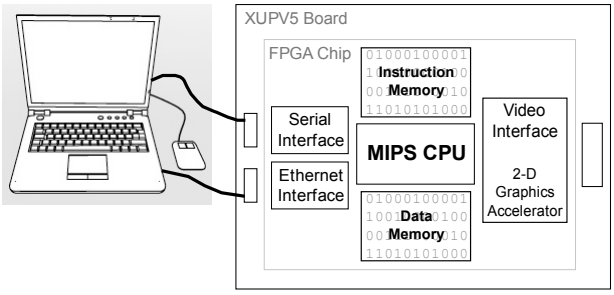
CS150 File Transfers

- Our Boot Monitor program will use TFTP protocol.

MAC	IP	UDP	TFTP		CRC

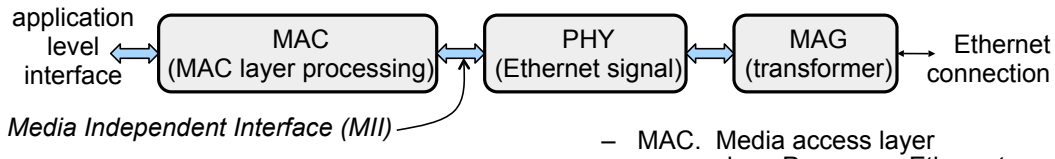
  - "Trivial file transfer protocol" - per packet acks for reliability.
  - Server (tftpd) will run on host machine (lab machine or your laptop).
  - MIPS150 acts like client (built into the boot monitor).

```
> 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/prot a web server if interested.

Standard Hardware-Network-Interface



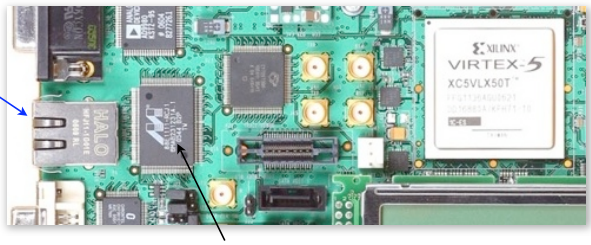
- Usually divided into three hardware blocks. (Application level processing could be either hardware or software.)
  - MAG. "Magnetics" chip is a transformer for providing electrical isolation.
  - PHY. Provides serial/parallel and parallel/serial conversion and encodes bit-stream for Ethernet signaling 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.

Ethernet Connector (RJ45) with built-in MAG



Marvell "Alaska" (88E111) PHY

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