EECS 150 Spring 2004

Lab Lecture 10
Checkpoint3
4/2/2004

Greg Gibeling

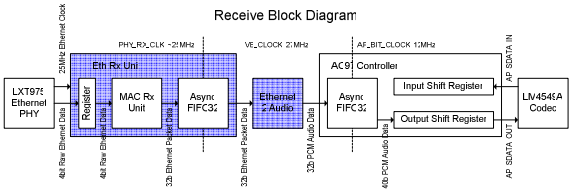
Today

- Checkpoint3
- Network Organization
- Ethernet Packets
- Ethernet Tx
- Announcements
- Ethernet Rx
- Bit/Nibble Ordering
- Synthesis Attributes

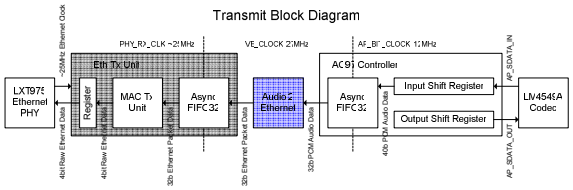
Checkpoint3 (1)

- You will Rx/Tx Audio over Ethernet
  - We're looking for receive
  - Transmit is for the project
- The Checkpoint
  - Take in ethernet nibbles
  - Convert them to 32b words
  - Filter packets
  - Send packets to AC97 Controller

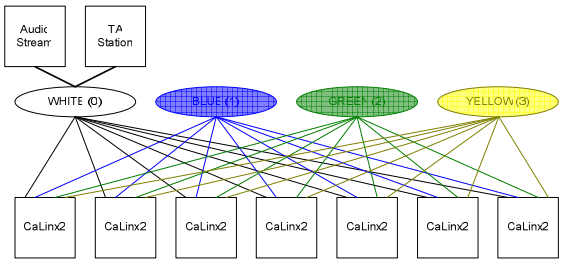
Checkpoint3 (2)



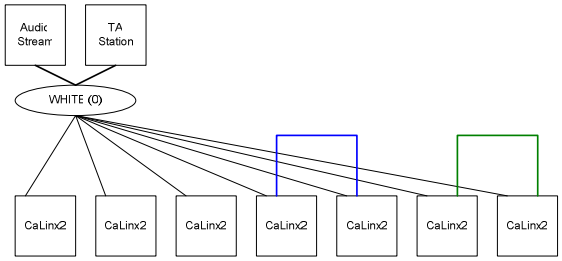
Checkpoint3 (3)



Network Organization (1)



Network Organization (2)



Network Organization (3)

- White network is reserved for broadcast
  - DO NOT TRANSMIT ON WHITE NETWORK!
- Building test networks
  - All cables are crossover
  - Connect two boards directly to each other
  - Using Blue/Green/Yellow networks

Network Organization (4)

DO NOT MODIFY THE PRODUCTION NETWORK

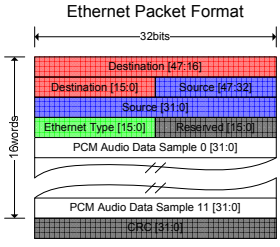
Ethernet Packets (1)

- We're using raw ethernet
  - No TCP/IP, its too complex
  - So the project wont work on the internet
- Raw Ethernet:
  - 48bit Destination MAC Address
  - 48bit Source MAC Address
  - 16bit Ethernet Type
  - Payload
  - 32bit CRC

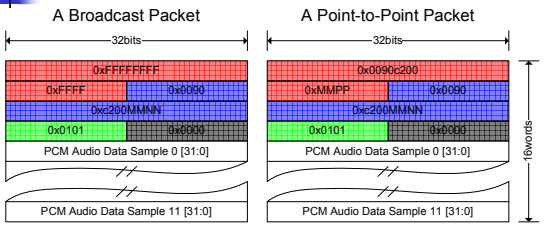
Ethernet Packets (2)

- A Little Theory of Ethernet
  - Bit Serial 100Mbps Link
    - 4/5bit Encoding takes 20% overhead
    - Bit5 is used for PHY_RX_DV and PHY_RX_ER
  - Preamble used for clock extraction
  - Inter Frame Gap ensures packets aren't back-to-back
  - CRC used to avoid errors from transmission

Ethernet Packets (3)



Ethernet Packets (4)



MM = An 8bit ID assigned to your group
This will help prevent different groups' projects from interfering with each other because the two designs will ignore packets with the wrong M code.

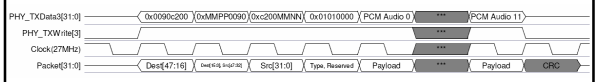
NN/PP = An 8bit ID you may select
This will allow a group to have multiple copies of their design on multiple boards, by using a different N/P code, the boards can distinguish each other.

Ethernet Tx (1)

- Given:
  - The entire transmit section (as a blackbox)
  - Why?
    - CRC is hard to do right
    - A broken solution might clog/crash the network
- A Simple FIFO Interface
  - Every 16 words you write will be sent out
  - Make sure to include the header!

Ethernet Tx (2)

- TX Operation
  - Write 16 words to FIFO
  - Transmitter sends out packet
  - Transmitter automatically generates CRC
  - Transmitter will wait for 16 words
    - We're using fixed length packets



Ethernet Tx (3)

- Broadcasting Audio
  - Write 4 words of header to the FIFO
  - Copy 12 words of PCM Audio from the AC97 Controller
  - Repeat
  - This is a very simple FSM

Ethernet Tx (4)

- Simulating Transmit…
  - You can't really simulate our blackbox
  - Use a file with a test pattern
- Simulating test patterns from a file…
  - Read hex nibbles into a memory array
  - Send them out using a counter
  - Very easy to work with
  - Remember Lab5 Part1?

Announcements (1)

- Midterms returned starting Monday
  - Re-grade Deadline is Thurs, April 15th
- Project Grading Breakdown
  - 10% per Checkpoint (4 Checkpoints)
  - 10% for the final report
  - 50% for the final project demo

Announcements (2)

- Design Review is VITAL this week
  - It is 25% of your checkpoint3
  - This checkpoint is easy to build
  - But HARD to understand!
- Group Name Signups are Up
  - Remember your group name & number
  - No spaces or punctuation

Announcements (3)

- Asking TAs Questions
  - We will clarify and help with hard problems
  - Don't ask us about things we announced
  - Re-read the lab and lab lecture slides
  - We reserve the right to say "Read the assignment"

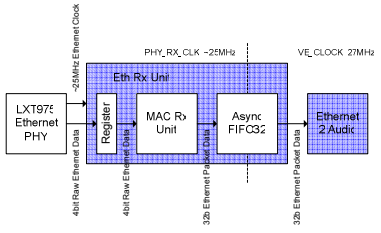
Announcements (4)

- Group Problems
  - Talk to a TA NOW
  - Don't wait until the last minute
  - Don't carry a dead weight partner
  - Companies don't like hiring people who will cover for a bad coworker

Ethernet Rx (1)

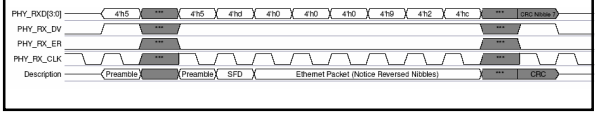
- Given:
  - Not much…
- The EthRx Module
  - Inputs
    - PHY_RX_D [3:0]
    - PHY_RX_DV
    - PHY_RX_ER
    - PHY_RXRead
  - Outputs
    - PHY_RXData [31:0]

Ethernet Rx (2)



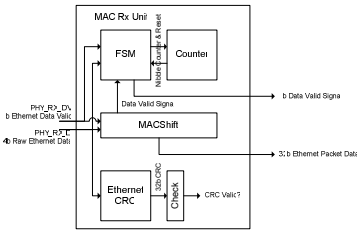
Ethernet Rx (3)

- What the receiver sees:
  - Preamble: 0x-15x 4'h5
  - Start Frame Delimiter: 1x 4'hd
  - Packet: 128x (OR MORE!!!) 4'h?
  - CRC: 8x 4'h?

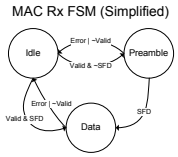


Ethernet Rx (4)

MAC Rx Detailed Block Diagram



MAC Rx FSM (Simplified)



Bit/Nibble Ordering (1)

- Bits are reversed within Nibbles
- Nibbles are reversed within Bytes
- Everything shifts the wrong way
- And you have to invert the bits for CRC
- It gets ugly…

Bit/Nibble Ordering (2)

- Bits are reversed within nibbles
  - Ethernet was originally bit-serial
  - Do we care? Not really…
  - But if you want to check CRC, you have to feed the bits in backwards

```
EthernetCRC CRCCheck(     .Clock(    Clock),
                          .Reset(    Reset | CRCReset),
                          .Data(     {D[0], D[1]. D[2]. D[3]}),
                          .Enable(   1'b1),
                          .Disable(  1'b0),
                          .CRCError(/*Check this signal*/));
```

Bit/Nibble Ordering (3)

- Nibbles are reversed within bytes
  - Again, because ethernet is bit serial
  - Do we care? YES!
  - You MUST be VERY aware of this!
  - We've given you a module (Reverse) to help with this

Synthesis Attributes (1)

- Clock and Data Skew are a real problem
  - Checkpoint1 – that negedge silliness
  - Checkpoint2 – SDataIn off by a cycle
- Combating this with Ethernet:
  - We'll avoid fiddling with the clock
  - We'll register the I/O in a special way

Synthesis Attributes (2)

- Clock Buffers
  - Clean up the clock
  - Ensure there's no on-FPGA clock skew
  - But they introduce a little delay
  - And there are only 4 of them
- The Ethernet Clocks
  - There are EIGHT of them!
  - And we want to avoid skew
  - So we must not buffer them

Synthesis Attributes (3)

On EVERY module that uses these clocks…

```
module SomeModule(PHY_RX_CLK, …);
        input PHY_RX_CLK; /* synthesis syn_noclockbuf=1 */
        /* some verilog */
endmodule

module SomeOtherModule(PHY_TX_CLK, …);
        input PHY_TX_CLK; /* synthesis syn_noclockbuf=1 */
        /* some verilog */
endmodule
```

Synthesis Attributes (4)

- Avoiding Data Skew
  - Register the data at the FPGA edge
  - Just a normal register
    - PHY_RXD
    - PHY_RX_DV
    - PHY_RX_ER
  - But we need to tell the tools where to put the register!

Synthesis Attributes (5)

On the module with the register…

```
module SomeModule(PHY_RX_CLK, PHY_RXD, PHY_RX_DV, PHY_RX_ER,…);
        input           PHY_RX_CLK; /* synthesis syn_noclockbuf=1 */
        input   [3:0]   PHY_RXD;
        input           PHY_RX_DV, PHY_RX_ER;
        reg     [3:0]   PHY_RXD_Reg; /* synthesis syn_useioff = 1 */
        reg             PHY_RX_DV_Reg; /* synthesis syn_useioff = 1 */
        reg             PHY_RX_ER_Reg; /* synthesis syn_useioff = 1 */
        always @ (posedge PHY_RX_CLK) begin
                PHY_RXD_Reg <= PHY_RXD;
                PHY_RX_DV_Reg <= PHY_RX_DV;
                PHY_RX_ER_Reg <= PHY_RX_ER;
        end
        /* The rest of the module */
endmodule
```