# CPRE 583
# MP2: UDP packet processing
# (Due Fri: 10/14, Midnight)

**I. Download and file locations**

1. Download MP2.tar.gz (zip) from: http://class.ee.iastate.edu/cpre583

2. Uncompress into your U: (home directory)

3. Quickly look over section VII. of this document to see what should be turned in
    a) Also pay special attention to I.10

4. Do one read through this entire document before starting the MP. This includes going through the FAQs.

5. Read the README file located in the top directory of MP2

6. ISE project is located here: MP2/ise_proj/MP2.ise
    a) source ./Xilinx_12_4_src.txt
    b) To open project: ise MP2.xise &

7. You should be able to edit your VHDL, simulate, and build your bitfile from ISE

8. How to save a dataset and waveform format in Modelsim
    a) Dataset
        i. File -> Datasets -> (highlight vsim.wlf) -> Save As -> name_file.wlf -> Done
    b) Waveform format
        i. File -> Save Format -> name_file.do
9. How to load a data set and waveform format (Note load dataset first)
    a) Dataset
        i. File Datasets -> Open -> Browse -> (select wanted .wlf file)
    b) Waveform format
        i. File -> Load -> (select wanted .do file)
10. Note, anytime you change your wave format (e.g. add signals, or change signal radix) it is a good idea to save the format (or you will have to redo your changes the next time you run simulation). Currently the simulator opens wave_mti.do by default. So if you want your default waveform to change, then save your changes as wave_mti.do.

11. When using ISE make sure the upper left hand window drop box is set correctly
    a) Implementation: when making a bitfile, editing code
    b) Behavioral Simulation: when doing simulation.

12. Sending UDP packets to the real hardware: The following directory has a README file, read it. And several simple .c programs for sending UDP packets. Let me know if you run into any issues or have any questions.
    a) MP2/UDP_SW

13. You will be using a Linux program called tcpdump to examine the packets you get back in hardware to make sure they have the correct count values.

14. For those of you that are interested in some of the low level details of the MP2 infrastructure you can down load the .pdf called "v5_emac_gsg340" from the course website. I used this document to help create MP2.

**II. (5 pts) Getting started**

1. MP2_scanner (MP2/vhdl/client/): This is the module where you will be writing 99% of your VHDL. Please read through this file to get a feel for what is going on.
    a. Basic functions
        i. Swaps the src and dst IP address
        ii. Zero's out the UDP checksum
        iii. Finds the string "CO"
2. emac0_phy_tb.vhd (MP2/simulation/): This is the primary part of the test bench you will be using. Make sure you understand how frames are being generated so that you can add your own, and/or modify the existing frames. The frames are what contain the data that is being sent to your hardware in simulation.

3. Before modifying anything simulate the project (Modelsim will print some errors: **gmii_txd incorrect**, and then stop with an error that says simulation "**Failure: Simulation stopped**", ignore this).
    a. Identify where the simulation finds the string "CO", and then toggles a flag once "CO" is detected.
        i. Save the dataset and wave format:
            1. call dataset: MP2_org.wlf
            2. call wave format: MP2_org.do
    b. Become familiar with the find "CO" state-machine
    c. (2 pts) Draw out the state machine for "CO"
        i. Patterns may need to be counted over many packets/frames. Why could the given FSM have a problem CORRECTLY counting the number of "CO"s over multiple packets (Note: for MP2 this issue should not be fixed until you complete everything else, it is bonus)
    d. (3 pts) Write a hardware diagram for how the UDP check sum is being set to zero (.pdf or Power Point slide). Look at the VHDL and simulation to figure this out

4. Make sure MP2 is working in hardware
    a. Generate and load the bitfile
    b. Send a packet containing "CO" to the hardware. This should cause LED 0 to toggle (See FAQ 1 for viewing LEDs remotely)
        i. See the README file in UDP_SW
        ii. ./exe_Test_gen 192.168.1.12 'CO'

**III. (15 pts) Detect: CORN!, ECE, GATAGA**

1. Extend the "CO" FSM to detect "CORN!". Each time "CORN!" is detected in simulation the LED should toggle.
2. Now create an "ECE" and "GATAGA" FSM: give them each their own LED (See how to add LEDs by looking at the LED for "CORN!". You **WILL** need to modify more files than just MP2_scanner.
    a. Modify the test bench to send 3 "CORN!" strings, 5 "ECE" strings (at least one should be "ECECECEC", 3 "GATAGA" strings (at least one should be "GATAGATAGATAGA").
        i. **Note:** emac0_phy_tb.vhd is the part of the testbench that sends packets to your design. You should be able to see how emac0_phy_tb.vhd is being setup to send "CO". The testbench will look quite intimidating at first, but I've commented the parts that are import to you to help you understand it.
        ii. **See FAQ for more info about the testbench**
    b. In simulation make sure the LEDs are toggling correctly (make sure they are toggling correct at the top level entity)
    c. (10 pts) Save the dataset and wave format:
        i. Name dataset: MP2_toggle.wlf
        ii. Name wave format: MP2_toggle.do
        iii. Make a note of the final state of each of the three LEDs
    d. (5 pts) Create a bitfile and test in hardware
        i. save the bitfile as: MP2_toggle.bit
    e. Save this version of the VHDL as MP2_scanner_toggle.vhd

**IV. (30 pts) Count: CORN!, ECE, GATAGA using LEDs**

1. Create counters for "CORN!", "ECE", and "GATAGA", and use the FSM outputs from the previous part to control the counters (make 8-bit counters).
    a. Tie the lower 2-bits of the "CORN!" counter to LED0, LED1
    b. Tie the lower 3-bits of the "ECE" counter to LED2, LED3, LED4
    c. Tie the lower 3-bits of the "GATAGA" counter to LED5, LED6,LED7

2. (20 pts) Simulate using the testbench input from III.2a.
    a. Verify that your count values are correct
    b. Write the value down (this will be turned in)
    c. Save the dataset, and wave format:
        i. Name dataset: MP2_led_cnt.wlf
        ii. Name wave format : MP2_led_cnt.do

3. (10 pts) Create a bitfile and test it in hardware.
    a. Name it MP2_led_cnt.bit

4. Save this version of the VHDL as MP2_scanner_led_cnt.vhd

**V. (50 pts) Return count at the end of a packet: CORN!, ECE, GATAGA**

1. The goal of this part is to overwrite the last 3 bytes of a packet with the 3 counters that indicate the number of times "CORN!", "ECE", and "GATAGA" have been seen.

2. Take a close look at how the UDP checksum is set to zero, and how the src IP and dst IP address are swapped. Use this technique to insert the 3 8-bit count values by overwriting the last 3 bytes of the UDP data payload.

3. Simulate using the input from III.2a
    a. Verify that your count values were correctly written to the output packet
    b. Write the values down (this will be turned in)
    c. (35 pts) Save the dataset, and wave format:
        i. Name dataset: MP2_pkt_cnt.wlf
        ii. Name wave format: MP2_pkt_cnt.do

4. (10 pts) Create a bitfile and test it hardware
    a. Name it MP2_pkt_cnt.bit
    b. You will use tcpdump to verify the packet you get back from the hardware contains the correct counter values. This will be needed if you've corrupted the packet
        i. 5-pts for seeing the correct count values in the correct position in the packet using tcpdump
        ii. 5-pts for the correct values printed by the C-program, in the correct place. Note the count values are returned in binary so you will likely want to modify the C program to print the received data using the %x formatting flag for printf.

5. Save this version of the VHDL as MP2_scanner_pkt_cnt.vhd

6. (5 pts) What happens if you send a message that is less than 3 characters long (use tcpdump)? Why is it important to send a message that is 3 characters or longer.

**VI. Bonus, count over many packets/frames: CORN!, ECE, GATAGA (+5)**

1. (+2) Fix the FSMs so that they will CORRECTLY count over many packets/frames (see II.3.c.i).
    a. Give a short written explanation of your solution
        i. Give the before and after drawing of one of your FSMs

2. (+2) Simulate your solution
    a. Modify your testbench in order to test the fix
    b. Save the dataset, and wave format
        i. Name dataset: MP2_fix_cnt.wlf
        ii. Name format: MP2_fix_cnt.do
    c. Write down 2 time points that show the fix working

3. (+1) Create a bitfile and test it in hardware
    a. Name it MP2_fix_cnt.bit

4. Save this version of the VHDL as MP2_scanner_fix_cnt.vhd

**VII. What to turn in**

1. MP2_username.tar.gz (or .MP2_username.zip). Just gzip up you whole MP2 directory structure, and send me the full directory path to your .tar.gz file.
2. Double check to make sure you have completed everything that has points associated with it. Please place all of these files in a "doc" directory within your MP2 directory structure.

**FAQs:**

1. **Viewing LEDs remotely**

There are webcams attached to xilinx-1, xilinx-2, and xilinx-3.ece.iastate.edu. xilinxcam1, xilinxcam3, and http://xilinxcam5.ece.iastate.edu monitor the FPGA board connected to each of the xilinx machines respectively (i.e. xilinxcam5 monitors xilinx-3).

Login name: xilinxuser
Password: drjones

Select: Active X or Java depending on your web browser's setup.

2. **Links to protocol resources:**
    a. Ethernet: http://wiki.wireshark.org/Ethernet#head-477fea80232d5062bbea553c84d4691d42fc9f80

    b. IP: http://www.networksorcery.com/enp/protocol/ip.htm

    c. UDP: http://www.tcpipguide.com/free/t_UDPMessageFormat.htm

Note you can just set the checksum field to 0, since it is zero'ed out by the MP2_scanner.

3. **Changing the length of the Ethernet frames in the testbench (/simulation/emac0_phy_tb.vhd)**

If you want to change the length of the Ethernet frame make sure to:

1. Update the Ethernet payload length filed properly
2. Update the UDP packet length properly

Note 1: I've labeled the position of both of these fields in each frame of the testbench. You should also be able to verify these positions using the Ethernet and UDP spec that I provide links to (see FAQ 2).

Note 2: If you do not update these fields the testbench will drop the frame, and it will never get to you module in simulation.

Note 3: The maximum frame length you can have is 255 (take a look at the frame_data data structure to see why this is. You can change this is you want, but for this MP there should be no need to)

4. **Adding Ethernet frames**

The testbench is only aware of the Ethernet protocol. It knows nothing about IP or UDP. So as far as the testbench is concerned, everything after the Ethernet header is just the payload. I have added comments to Frame 0 to indicate where various IP and UDP header fields start, and where the UDP payload data should start. If you would like to add more frames I would suggest copying Frame 0, and overwriting the existing Frames that currently exist in the testbench. Though I believe you should be able to also append frames to the end. If you are curious, then give appending frames to the end a try, and let me know if it works)

5. **Options to adding new signals to Modelsim**

Recommended: Save the wave format file as "wave_mti.do" (overwrite). This is what Modelsim loads by default for MP2.

OR

Modify simulate_mti.do to point to whatever .do name you are using for your waveform format.

OR

After launching modelsim and after it simulates you can load your wave format file, then restart simulation (restart -f), followed by the command run 40 us (or however long you want to run.

6. **How to use TCP dump:**

View the packets using the following command (note: run command in its own window):

sudo /usr/sbin/tcpdump -i eth0 -v -s 0 -XX

Example output for two packets. First from PC to HW, Second packet is echo'ed back to PC. I have labeled some of the important things in red (**Note to see the red use the Word version of the write up**). The packet was sent using the following command from UDP_SW: ./exe_Test_gen 192.168.1.12 'HiCO!'

Use the network protocol links given earlier to identify the different parts of the packet. Note: 000d is the UDP packet length, 4869 434f 21 is the UDP payload you sent, HiCO!

Src IP (192.168.1.5:port) Dest IP(192.168.1.12:port)
15:52:36.178183 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto: UDP (17), length: 33) 192.168.1.5.44578 > 192.168.1.12.44578: UDP, length 5
0x0000: aabb ccdd ee00 001b 2123 3354 0800 4500 .....!#3T..E.
0x0010: 0021 0000 4000 4011 b76a c0a8 0105 c0a8 .!..@.@..j.....
0x0020: 010c ae22 ae22 000d 8380 4869 434f 21 ..."."....HiCO! Hex | ASCII

15:52:36.178220 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto: UDP (17), length: 33) 192.168.1.12.44578 > 192.168.1.5.44578: UDP, length 5
0x0000: 001b 2123 3354 aabb ccdd ee00 0800 4500 ..!#3T.....E.
0x0010: 0021 0000 4000 4011 b76a c0a8 010c c0a8 .!..@.@..j.....
0x0020: 0105 ae22 ae22 000d 0000 4869 434f 2100 ..."."....HiCO!.
0x0030: 0000 0000 0000 0000 0000 0000 ............

For this case the message sent was HiCO! (Which should cause LED 0 on the ML507 to toggle, since it has the string "CO").