ComS / CprE 583 – Reconfigurable Computing

MP-2: Simple Packet Processing

Assigned: Friday of Week 5
Due: Friday of Week 7
Points: 100 + bonus for inter-packet counting

[Note from Joe: The goal of this Machine Problem is for you to work with your group to gain experience in two main areas:

1. *Networking hardware interfacing – you will explore an advanced FPGA component for UDP networking over Ethernet, and will learn how to change the basic functionality and test in simulation as well as through a hw/sw framework.*
2. *Finite State Machine design – you will design and implement several FPGA-based string matching components using finite state machines in VHDL. The FPGA designs will be evaluated for correctness of output.]*

Reading: It is *strongly* recommended that you review the VHDL tutorial [MeaTap12A], chapter 7, before beginning this assignment. Although it is a bit lengthy, chapter 7 covers two and three-process Finite State Machine design in VHDL and provides a good overview of this industry-standard approach for FSMs.

1) Platform Overview. Figure 1 shows the high-level structure of the MP-2 hardware setup. Similar to MP-1, for this first part, you will first generate a hardware configuration (bitfile) for the FPGA using the existing VHDL source, and then will download that bitfile to the FPGA. You will then test that the initial design works before making modifications in subsequent sections. Keep in mind that although the setup appears similar at first glance to MP-1, the interface we are using (UDP over Ethernet) is significantly more advanced than Serial UART. This extra complexity is something we will be exploring in MP-2.

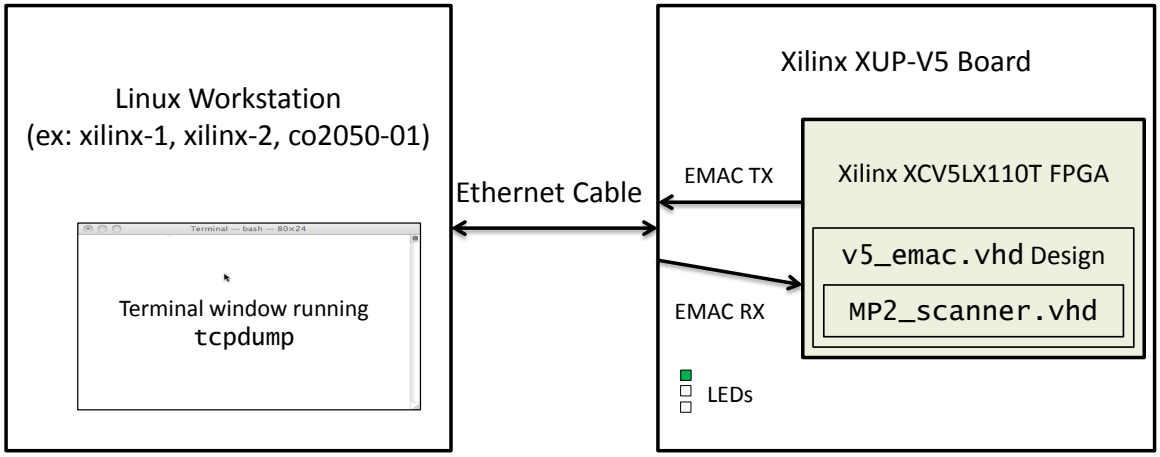

Figure 1: MP-2 Hardware Setup

After using NX client (or software of your choice) to connect to an appropriate Linux machine, download the `MP-2.zip` file from Blackboard into your ISU home directory. Don't forget to source the `setup.sh` file each time you open a new terminal – for your convenience a copy of `setup.sh` is included in the `MP-2.zip` attachment.

MP-2 has a slightly different structure, with some software applications for testing the UDP socket in the `sw/` directory, and the main Xilinx ISE project in the `hw/` directory. The Xilinx ISE project is located in `MP-2/MP1.xise`, and can be opened using the `ise` command:

```
ise MP2.xise &
```

Review [Jon12A] as well as MP-1 for instructions on how to connect to the FPGA and work remotely. Browse through the design in ISE, and using reference [UG340] as a guide, provide a 1-paragraph description of the basic structure of the design. It is recommended that you complete this step early on in the MP-2 assignment window, as your understanding of the basic structure will enable you to make faster progress on the remainder of the design tasks.

2) Basic Scanning. The `MP2_scanner` module (in `hw/vhdl/client/`) is where you will be putting most of your VHDL code. Read through this file to get a feel for the basic functionality:

- Swapping of the src and dst IP addresses
- Zeroing out of the UDP checksum
- Locating a string "CO" in the packet

The other module of interest at this point is testbench: `emac0_phy_tb.vhd` (located in `hw/simulation/`). Browse this file to make sure you understand how frames are being generated so that you can add or modify existing frames in simulation. Simulate the project (Modelsim may print out some errors regarding the `gmii_txd` module – these can be ignored).

Identify where in the simulation the string "CO" is detected. Save the dataset and wave format as `MP2_part2.wlf` and `MP2_part2.do`. Draw out the state machine for detecting "CO". Assuming that patterns may need to be detected over multiple packets, is the current FSM correct? Defend your answer.

Create a hardware diagram that details how the UDP checksum value is being set to zero.

Before continuing on, verify that MP-2 is working in hardware:

- Generate and load the bitfile in `impact`
- Send a packet containing "CO" to the hardware. This should cause LED 0 to toggle. You can use the `Test_gen` application for this purpose. See the README file in `sw/` for directions.

3) String Detection. Extend the "CO" FSM to detect "CORN!". Each time "CORN!" is detected in simulation, the LED should toggle.

Next, create separate FSMs to detect strings "ECE" and "GATAGA" – each string should toggle a separate LED. Some notes on this step:

- You will need to modify more than the `MP2_scanner` module, as the LEDs are mapped at the top level.
- Modify the testbench to send 3 "CORN!" strings, 5 "ECE" strings (at least one should be "ECECECEC", 3 "GATAGA" strings (at least one should be "GATAGATAGATAGA"). The `emac0_phy_tb.vhd` testbench will take some time to go through and understand.
- In simulation, ensure that the LEDs are toggling correctly.
- Save the dataset and wave format as `MP2_part3.wlf` and `MP2_part3.do`. Make a note of the final state of the three LEDs.
- Create a bitfile and test in hardware. When your design is fully functional, save this version of the design as `MP2_scanner_part3.vhd` and `MP2_part3.bit`.

4) String Counting. Create counters for the three strings that your MP2_scanner currently matches, and use the FSM outputs from the previous part to control the counters. An 8-bit counter should be sufficient:

- Tie the lower 2 bits of the "CORN!" counter to LED0 and LED1
- Tie the lower 3 bits of the "ECE" counter to LED2, LED3, and LED4
- Tie the lower 3 bits of the "GATAGA" counter to LED5, LED6, and LED7

Simulate using the testbench input, and verify that your count values are correct. Write down the observed counter value, and save the dataset and wave format as `MP2_part4.wlf` and `MP4_part4.do`. Create a bitfile and test in hardware. When your design is fully functional, save this version of the design as `MP2_scanner_part4.vhd` and `MP2_part4.bit`.

5) Message Return. The goal of this part is to overwrite the last 3 bytes of a packet with the 3 counter values that indicate the number of times "CORN!", "ECE", and "GATAGA" have been seen.

Take a close look at how the UDP checksum is set to zero, and how the src IP and dst IP addresses are swapped in hardware. Modify this technique to insert the 3 8-bit count values by overwriting the last 3 bytes of the UDP data payload. Simulate using the input from the modified testbench, verify that your count values were correctly written to the output packet, and save the dataset and wave format as `MP2_part5.wlf` and `MP4_part5.do`.

Create a bitfile and test in hardware. Use either the `tcpdump` utility or the UDP test applications to verify that the packet you get back from the hardware contains the correct counter values. When your design is fully functional, save this version of the design as `MP2_scanner_part5.vhd` and `MP2_part5.bit`. Also, briefly explain in your writeup what happens in your design if you send a message that is less than 3 characters long. Use `tcpdump` to verify.

6) Bonus – Inter-Packet Counting. Modify the FSMs in `MP2_scanner` so that they correctly count over multiple packets / frames. Provide a short written explanation of your solution, including the FSM diagram. Modify the testbench in order to test your solution in simulation. Save the dataset and wave format as `MP2_part6.wlf` and `MP4_part6.do`. Provide evidence

in your writeup that the modifications are working – for example, isolate which time stamps to inspect in the wave file that indicate matches over multiple packets.

Finally, create a bitfile and test in hardware. When your design is fully functional, save this version of the design as `MP2_scanner_part6.vhd` and `MP2_part6.bit`.

Summary – What to Submit. Create a zip file containing any modified and newly-created .vhd files, as well as a copy of your wavefile format (.do), dataset (.wlf), and bitfile (.bit) for parts 2, 3, 4, 5, and (bonus question) 6. Attach this zip file with your writeup as a PDF file. Make sure the writeup covers all the highlighted sections in this document. In the Blackboard submission field, list each team member with a percentage of their overall effort on MP-2 (with percentages summing to 100%).

Summary – What to Demo. At least one group member must be available to demo your MP-2 applications running on the FPGA and to review aspects of your implementation. A full demo score requires correct results when running the MP on the FPGA board, but partial credit will be given for effort. Be prepared to briefly discuss your source code.

Notes on Ethernet and UDP. The following are some links to protocol resources**:**

- Ethernet:http://wiki.wireshark.org/Ethernet#head-477fea80232d5062bbea553c84d4691d42fc9f80
- IP: http://www.networksorcery.com/enp/protocol/ip.htm
- UDP: http://www.tcpipguide.com/free/t_UDPMessageFormat.htm

If you want to change the length of the Ethernet frame make sure to 1) update the Ethernet payload length field properly, and 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 in the links above.

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.

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.

Notes on Using `tcpdump`. View the packets using the following command (note: similar to using `minicom`, run these commands in their 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. The packet was sent using the following command from `sw/: ./exe_Test_gen 192.168.1.12 'HiCO!'`

We can use the network protocol links provided 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 XUP-V5 to toggle, since it contains the string "CO".