MP-2 Write-Up

Blake Vermeer
Kris Hall
Rohit Zambre

October 9, 2014

Date Due: October 10, 2014
Instructors: Joseph Zambreno

1 Platform Overview

In this section we were asked to review the project and provide a short description of the project structure. Here is our description:

The core part of this project is split into three main files: *v5_emac_v1_4_example_design.vhd*, *v5_emac_v1_4_locallink.vhd*, and *v5_emac_v1_4_block.vhd*. The *v5_emac_v1_4_block.vhd* is the lowest level element of the three. This wrapper file is responsible for interfacing with the physical Ethernet hardware on the FPGA. The *v5_emac_v1_4_locallink.vhd* is the next level up file. It is responsible for interfacing with the with the block level wrapper file and providing FIFOs for the input and output of the Ethernet module. The highest level of the design is *v5_emac_v1_4_example_design.vhd*. This module interfaces with the locallink wrapper file and provides a basic loopback interface.

2 Basic Scanning

The state machine diagram for the detection of "CO" is as follows:

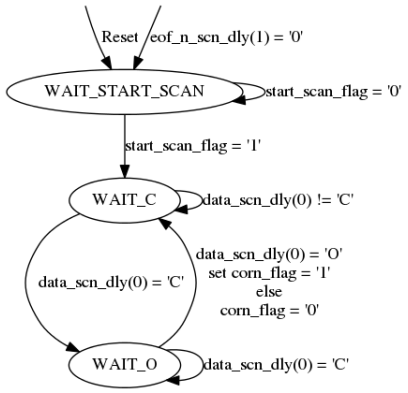

Figure 1: State machine diagram for the detection of "CO"

This diagram is correct so long as the string "CO" is not split up over multiple packets. If the string is split over multiple packets, the *eof_n_scn_dly(1)* flag will reset the state machine.

Here is the screenshot of the simulation where we can see CO being detected:

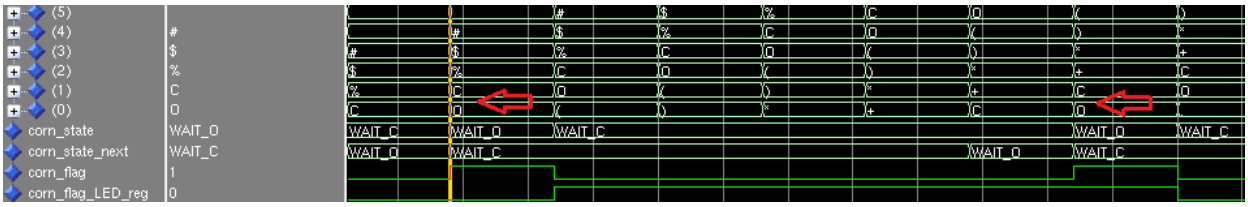

Figure 2: Detection of "CO" in simulation

The hardware diagram for how the UDP checksum is being set to zero is shown below:

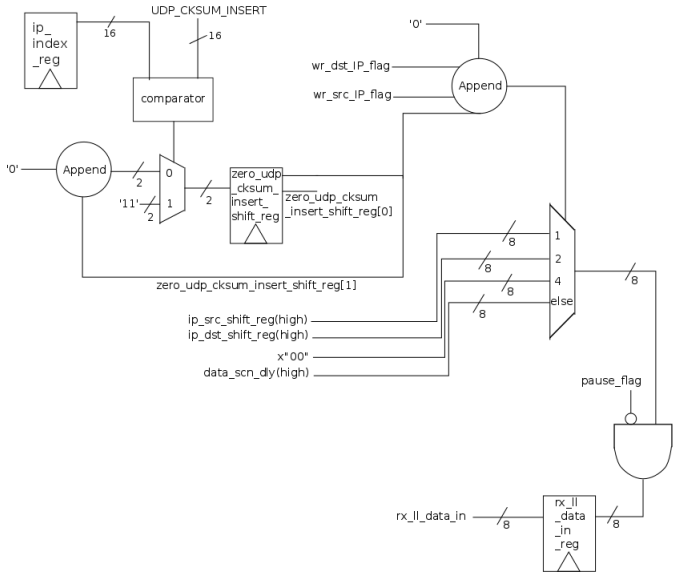

Figure 3: UDP checksum zeroing diagram

3 String Detection

Adding on to the previous section, the task for this section is to extend the finite state machine that can detect the string "CO" to be able to detect the string "CORN!" along with the creation of two extra finite state machines that detect the strings "ECE", and "GATAGA".

The extended finite state machine for detecting the string "CO", now extended to detect the string "CORN!", has the following state diagram:

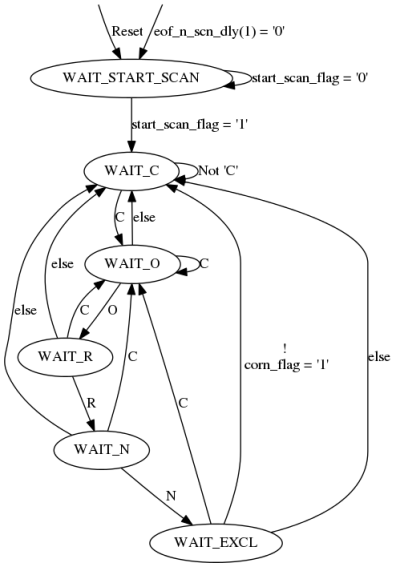

Figure 4: Extended "CO" FSM to detect "CORN!"

A new finite state machine for the detection of the string "ECE" has the following state diagram:

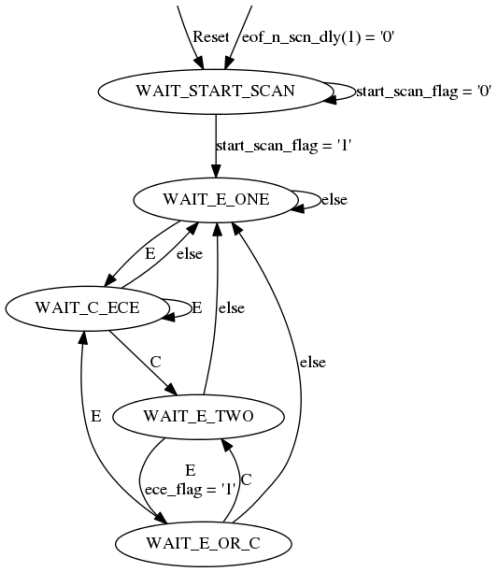

Figure 5: FSM to detect "ECE"

Another new finite state machine for the detection of the string "GATAGA" has the following state diagram:

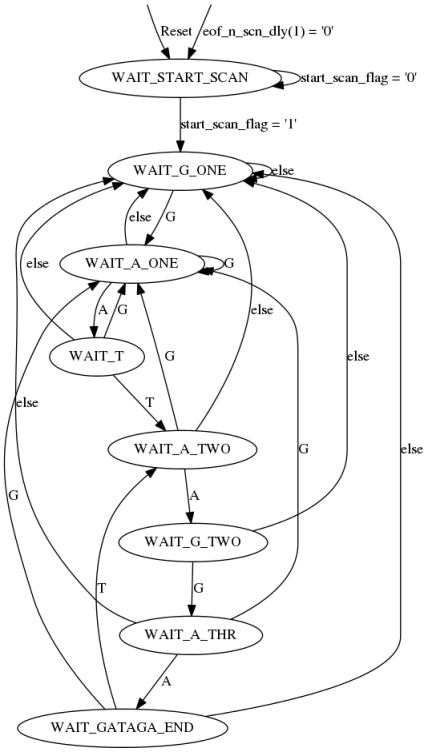

Figure 6: FSM to detect "GATAGA"

4 String Counting

In this section, the task given is that of creating counters to count how many times the strings CORN!, ECE, or GATAGA are detected in a given UDP packet. These counters are to be 8-bit counters, and the bits of the separate counters are to be tied to the following LEDs:

corn_counter[0] = LED0
corn_counter[1] = LED1
ece_counter[0] = LED2
ece_counter[1] = LED3
ece_counter[2] = LED4
gataga_counter[0] = LED5
gataga_counter[1] = LED6
gataga_counter[2] = LED7

In simulation, the counters work as expected, as can be seen in this screenshot of Modelsim.

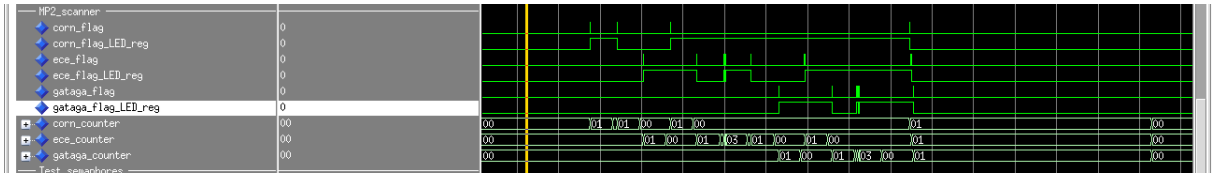

Figure 7: Relevant signals for the simulation of the counters

5 Message Return

In this section, we added the ability for our module to overwrite the last three bytes of the return packet with the counts for the number of times "CORN!", "ECE", and "GATAGA" were detected within the packet respectively. We decided to return these counts as ASCII numbers since it is unlikely that there would be more than 9 of any one of the words in a packet and it makes debugging substantially easier.

This portion of the project was accomplished by adding a simple process that tests when the UDP index is three bytes from the end of the UDP packet and sends a signal to preload a 3 bit shift register with all ones. This process also makes sure that the length of the UDP packet is greater than 10 (3 or more bytes of payload) so that the checksum cannot be overwritten with the counter values. The shift register then controls the output of a mux to the *rx_ll_in_insert* signal from the MP2_scanner.vhd module.

Here is a screen-shot from a portion of the testbench that shows the counter values being output at the end of the UDP packet.

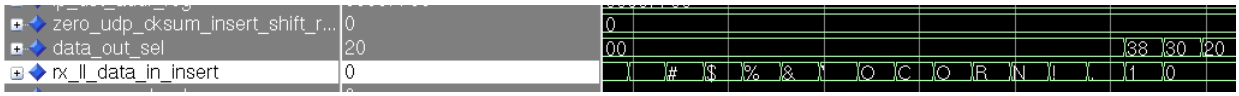

Figure 8: Part 5 simulation screen-shot

Here is a screen-shot of the output from the module implemented on hardware with various inputs. Notice that the module does not output the counter values when the input is less than 3 characters long.

```
┌(bvermeer@xilinx-3)-(✓)-(07:57 PM Thu Oct 09)
└(~/cpre583/CprE583/MP-2/sw)-(7 files, 64Kb)-> ./exe_Test_gen 192.168.1.12 'CORN!CORN!'
sent 10 bytes
got 10 bytes with content CORN!CO200

┌(bvermeer@xilinx-3)-(✓)-(08:00 PM Thu Oct 09)
└(~/cpre583/CprE583/MP-2/sw)-(7 files, 64Kb)-> ./exe_Test_gen 192.168.1.12 'ECE'
sent 3 bytes
got 3 bytes with content 010

┌(bvermeer@xilinx-3)-(✓)-(08:00 PM Thu Oct 09)
└(~/cpre583/CprE583/MP-2/sw)-(7 files, 64Kb)-> ./exe_Test_gen 192.168.1.12 'EC'
sent 2 bytes
got 2 bytes with content EC
```

Figure 9: Part 5 output

6 Bonus - Inner-Packet Counting

For the bonus portion, we could accomplish the goal of counting the number of occurrences of the various target strings over multiple packets without any modifications to our state machine. Our state machines for our target strings were signaling a corresponding counter to advance on string detection. The counters where then being reset when a device reset signal or a new packet signal were received. To allow our module to count over multiple packets, we simply had to change the reset conditions for the counters and only have the counters reset on device reset and not on the new packet signal.

For proof in the simulation of the string counters counting over multiple packets, compare time-stamp 22843000 ps and time-stamp 23491000 ps in the simulation (specifically, look at the *rx_ll_data_in_insert* signal to see the UDP output).

Here is a screen-shot of the output from the device over multiple packets. Notice how the counter values keep advancing over multiple packets.

```
┌(bvermeer@xilinx-3)─(✓)─(08:48 PM Thu Oct 09)
└─(~/cpre583/CprE583/MP-2/sw)─(7 files, 64Kb)─> ./exe_Test_gen 192.168.1.12 'ECE
 CORN!sfd'
sent 12 bytes
got 12 bytes with content ECE CORN!110

┌(bvermeer@xilinx-3)─(✓)─(08:48 PM Thu Oct 09)
└─(~/cpre583/CprE583/MP-2/sw)─(7 files, 64Kb)─> ./exe_Test_gen 192.168.1.12 'ECE
sdfs'
sent 7 bytes
got 7 bytes with content ECEs120

┌(bvermeer@xilinx-3)─(✓)─(08:49 PM Thu Oct 09)
└─(~/cpre583/CprE583/MP-2/sw)─(7 files, 64Kb)─> ./exe_Test_gen 192.168.1.12 'GAT
AGA'
sent 6 bytes
got 6 bytes with content GAT121

┌(bvermeer@xilinx-3)─(✓)─(08:49 PM Thu Oct 09)
└─(~/cpre583/CprE583/MP-2/sw)─(7 files, 64Kb)─> ./exe_Test_gen 192.168.1.12 'GAT
AGATAGA'
sent 10 bytes
got 10 bytes with content GATAGAT123
```

Figure 10: Part 6 output

7 Conclusion

In this lab we learned more about state machines and how to better design a VHDL project solution and how to better collaborate as a team. We also learned how to examine an existing VHDL project and determine where to modify it to obtain the desired functionality.