ECE 697J Fall 2003:
Lab 2 – IPv4 Forwarding and Classification on IXP 1200

Due: 11/18/03 (Part I), 11/25/03 (Part II)

The goal of this lab is to explore the basic functionalities of the IXP1200 software development kit and microengines. There are two parts to the project. Part I uses an existing sample application (IP forwarding) to collect a number of workload statistics from the IXP simulator. Part II requires the programming of a small application on the microengines to implement a simple packet classification mechanism.

Both parts require access to a machine that has the Intel SDK installed. You can remotely log on to the course machine (address and account information given in class) or you can use a machine in Marcus 15B. If you want, you can also request an installation CD for your own machine.

Part I: IPv4 Forwarding Simulation

In this part, you will run an implementation of IP forwarding on the IXP1200 simulator. All the code is provided to you. You will just need to collect a set of workload statistics that are reported by the simulator.

It is recommended that you go through the following steps before you explore the simulator on your own:

1. Log on to the machine with the SDK installed. Please be aware that this machine is shared. Be careful not to destroy anything (yes, you do have Administrator privileges because you need them for the SDK). Also, please log out as soon as you are done with your simulations.
2. Start "Developer Workbench"
3. Click on File->Open Project to open the IP forwarding project. In order to ensure that nobody overwrites other people's files, your personal copy of the forwarding code can be found at: C:\IXP1200\MicroCode\ECE697J\[your name]\ L3fwd16 . Open the project file L3fwd16.dwp in that directory.
4. You should see the list of project files on the right. Expand "Assembler Source Files" and double-click on "rx_ether100m.uc" to access the IP forwarding code. Read through the code to get a basic understanding of the structure. Some functions (e.g., ip_verify) are implemented in "ip.uc".
5. Build the project but clicking on Build->Rebuild.
6. Before starting the simulation, you need to ensure it stops at some point. Click on Simulation->IX Bus Device Simulator->Options. In the "Stop Control" tab, checkmark the option "Stop the simulation after the next 100 packets are received by the IXP from this bus" and click "OK".
7. Click Debug->Start Debugging.
8. Click Debug->Run Control->Go. The simulation should start and run for several seconds. Then a window pops up that says "The simulation is being stopped because the IXP received 100 packets from IX bus". Click "OK". At this point the simulation is interrupted and the simulation results are available. You can now access these statistics by looking at the appropriate windows.
9. Click Simulation->Simulation Statistics. In the "Summary" tab, you can see the microengine utilization and the MIPS rating as well as memory statistics. The "Microengine" tab shows more detailed results for each thread. Expand by clicking on the "+" next to a microengine. The "All" tab has a lot of statistics with detailed event counts and percentages. You need to select the parameter that you want to have displayed. Note that "f0" corresponds to Microengine 0. There is also a number of statistics on memory performance (e.g., latency distributions for various shared resources: R-FIFOs, SRAM, etc.).
10. Click on View->Debug Windows-> History and make sure the checkmark is set. This will display a window that shows the thread status and memory queue status for the entire execution. It might be necessary that you close/resize other windows to ensure good visibility. At the top of the history window, there are two checkmarks: Threads and Queues. Mark Threads and unmark Queues. You should see a graph with a number of lines and symbols. Click "Legend…" to display a window with

explanations. Look at microengine 0 (threads 0-3) and see how context switching occurs. Look at the solid and dotted lines that indicate the progress on resource requests. Find an example where you see an SDRAM and SRAM request from one thread in parallel. Does the microengine stall as a result of that request? Uncheck Threads and check Queues. This shows the queue lengths of various memory requests etc. Identify the queues that are heavily used.

11. Click Simulation->Execution Coverage… This will pop up a window that shows the source code of IP forwarding and a number on the left of some lines. This number indicates how often a particular instruction on a microengine was executed. Identify code blocks that are executed frequently and those that are not used at all.
12. Explore other windows, run simulation for more packets etc.

Statistics that you should collect, present, and discuss are:

- Microengine utilization for all microengines and detailed statistics of one thread from uE 0 and one from uE 5 (execution % and aborted %).
- Processing power of microengines (in MIPS).
- Memory utilization and bandwidth.
- Latency distribution for SDRAM refs for microengine 0 and SRAM non-read_lock refs for microengine 0 (you might want to run the simulation for a large number of packets (10000) before you collect these statistics). Show a graph (that you generate from this data).
- Show a screenshot (Alt+PrtScn and the paste into MS Paint or Word) for the thread history that shows overlapping SRAM and SDRAM requests by the same microengine. Identify the overall delay for either request (in cycles). What factors (queuing, actual access, …) contributed how much to the overall delay?

Write a short (3-4 pages) report (that requires some text!) that presents the above statistics (plus graphs and screenshots where indicated) and discusses them briefly (i.e., is there anything surprising, is it what you expect, etc.).

This report is due 11/18/03 in class.

Part II: Packet Classification

Details to be announced 11/18/03.