2012 Project Technical Description

History:
2/3/2012 : Authored P.D. Franzon
2/11/2012 : Added interface specification and number of ports
3/31/2012 : Some minor clarifications in edit mode

ECE 520-001 Packet Forwarding Engine

This project will be performed individually.

The internet is built as a series of store and forward IP packet processing switches. At each switch packets are incoming on a set of ports and one function of the switch is to determine which port to forward the incoming packets too. In principle this could be done with a Content Addressable Memory (CAM), but such a large CAM is too large and too slow. Thus it can be done by a compression technique using SRAMs and DRAMs. One such compression technique is described in the attached paper. You are to implement the compression technique. You will be given the following memories:

- Two 30 ns “DRAMs”. To simplify the interface, these will be implemented as SRAMs. You are to put one copy of the final table in each DRAM. They don’t count towards your area. (sram60.v)
- 0.3 ns SRAMs. You can have as many of these as you need but they can ONLY be used to implement the compression bit tables, not for anything else. They DO count towards your area. (sram5.v).

Your design objective is to maximize performance per unit area (not including the DRAMs in the area). One way to achieve this is to minimize the area required while ensuring you can sustain a rate of complete one output port lookup every 15 ns (by operating the “DRAMs” in parallel.) However, you can operate slower if you wish. Performance is the rate at which you complete lookups. You can replicate the SRAM as many times as you like but they can only be used to store replicas of the compression bit tables.

Inputs and Outputs (added 2/11/12)

- You are to support 16 incoming packet ports and 16 outgoing packet ports. This means that you have an input port on which the 32 bit packet is delivered and an output port on which the 4 bit packet address is output. The details of the interface of these ports to the test fixture is up to you. We are going to provide you with a text file containing a list of packets. You output a matching list of port IDs. A TA will generate the following files:
  - Sample data for each of the DRAMs (each DRAM stores identical data)
  - Sample data for the compressed SRAM structure
  - A sample incoming data stream of “traffic” organized as <incoming port><IP address>
  - NOTE: A second file will be provided close to project turn in. Your hardware is expected to work with this second file.
What I expect is that you load the DRAMs and SRAMs using appropriate initial statements. You then apply the incoming data stream through the test fixture to your design. You then collect the outgoing data stream as a series of <IP address><outgoing port>, again through the test fixture.

**ECE 464 Project: String Search Engine**

The purpose of this project is to find matching 5-character words in a two serial incoming string sequence. You are to report a match if a word on one port appeared on the other port, as one of the previous three words to appear there. i.e. You are reading in words on two ports, and if any of the three words on one port, match any of the three words on the other port, you are to report a match. Your I/O will be as follows:

**Inputs**
Clock
Dataln[7:0]; // Input ASCII port #1
Dataln2[7:0]; // Input ASCII port #2
Go; // High when Dataln is changing

**Outputs**
Match; // = 1 on cycle match is found (within two cycles of the matching word being read)

The operation will be as follows:

An ongoing (never ending) stream of ASCII characters will be presented at Dataln while Go is high. (i.e. If Go is low, Dataln is not changing). Each word will be 5 characters long with no separators (i.e. no spaces or punctuation) and will take 5 clock cycles to be “entered”. The matches are case insensitive, e.g. “Asic1” matches “asic1”. When a match is found, match should go high within two clock cycles of the last character of the matching word appearing on the input. Match should stay high for 5 cycles. For example consider the following sequence:

<table>
<thead>
<tr>
<th>Clock</th>
<th>Go</th>
<th>Dataln1</th>
<th>Dataln2</th>
<th>Match (1 to 2 cycles later)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-5</td>
<td>1</td>
<td>first</td>
<td>third</td>
<td>0</td>
</tr>
<tr>
<td>6-10</td>
<td></td>
<td>third</td>
<td>dirty</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>// Dataln1 found a match in previous Dataln2</td>
</tr>
<tr>
<td>11-15</td>
<td></td>
<td>found</td>
<td>found</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>// match in same cycle</td>
</tr>
<tr>
<td>16-20</td>
<td></td>
<td>aSics</td>
<td>bases</td>
<td>0</td>
</tr>
<tr>
<td>21-25</td>
<td></td>
<td>rough</td>
<td>first</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>// not a match – “first” is fourth previous word</td>
</tr>
<tr>
<td>26-31</td>
<td></td>
<td>moron</td>
<td>asics</td>
<td>1</td>
</tr>
</tbody>
</table>

The approach you use is up to you. However, to me, this screams for a couple of FIFOs as part of the core of the datapath. Do NOT use any srams.

Your objective is to meet the spec and maximize performance/area. In your report, describe your approach to maximizing performance/area.
Memory Model
See sram5.v listed with this project. Note, do NOT synthesize the memories. Also note that data has to be stable at the memory input 0.3 ns before the clock edge, and is not available until 0.3 ns after the clock edge. Your input and output delays to the memory ports must be modified appropriately in the Synthesis script. Note the area of this memory is 0.0666 sq.mm. (Note: Synopsys reports area in square micrometers.)