Bluespec-4
The IP Lookup Problem

Arvind
Laboratory for Computer Science
M.I.T.

January 14, 2003

http://www.csg.lcs.mit.edu/IAPBlue

The IP lookup problem

- An IP lookup table contains IP prefixes and associated data
- The problem: given an IP address, return the data associated with the longest prefix match ("LPM")

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.14.<em>.</em></td>
<td>A</td>
</tr>
<tr>
<td>7.14.7.3</td>
<td>B</td>
</tr>
<tr>
<td>10.18.200.*</td>
<td>C</td>
</tr>
<tr>
<td>10.18.200.5</td>
<td>D</td>
</tr>
<tr>
<td>5.<em>.</em>.*</td>
<td>E</td>
</tr>
<tr>
<td>*</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IP address</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.13.7.3</td>
<td>F</td>
</tr>
<tr>
<td>10.7.12.15</td>
<td>F</td>
</tr>
<tr>
<td>10.18.201.5</td>
<td>F</td>
</tr>
<tr>
<td>7.14.7.2</td>
<td>A</td>
</tr>
<tr>
<td>5.13.7.2</td>
<td>E</td>
</tr>
<tr>
<td>8.0.0.0</td>
<td>F</td>
</tr>
<tr>
<td>10.18.200.7</td>
<td>C</td>
</tr>
</tbody>
</table>
Sparse tree representation

Table representation issues

- LPM is used for CIDR (Classless Inter-Domain Routing)
- Number of memory accesses for an LPM?
  - Too many ➞ difficult to do table lookup at line rate (say at 10Gbps)
- Table size
  - Too big ➞ bigger SRAM ➞ more latency, cost, power
- Control-plane issues:
  - incremental table update
  - size, speed of table maintenance software
- In this lecture (to fit the code on slides!):
  - Level 1: 16 bits, Level 2: 8 bits, Level 3: 8 bits ➞ from 1 to 3 memory accesses for an LPM
Outline

- Example: IP Lookup
- Three solutions
  - Statically scheduled memory pipeline
  - Straight pipeline with uncoordinated memory references
  - Circular pipeline for 100% memory utilization
- Modeling RAMs
  - Synchronous vs. Asynchronous view
- Bluespec coding for the circular pipeline
  - completion buffer

Static scheduling solution

- Assume the SRAM containing the table has latency of $n$ cycles, lay out a pipeline so that the memory accesses are precisely scheduled

- Issues:
  - Since an LPM may take 1-3 mem accesses, unused slots may be left idle
  - May have to replan the pipeline for a different latency memory
  - Very difficult to plan if memory is also to be used for some unrelated task.
**LPM: Straight Pipeline Solution**

- **ROM**: IP Address Table
- **Port replicator**
- **Start lookup 1**: IP addr 18.100.32.127
- **Finish lookup 1**: Lookup Result (Egress port, ...)
- **Finish lookup 2**: (Start lookup 3)
- **Finish lookup 3**: 

---

**LMP: Circular pipeline solution**

- **getToken**
- **getTokenAck**
- **Completion buffer**
- **luResp**
- **luResp Ack**
- **LuReq**
- **tf**
- **Enter**
- **IP Address Table**
- **RAM**
- **Node**
- **Complete**
- **Move**
- **Opsi**
- **Recirc**
- **Buf**

**Completion buffer**
- gives out tokens to control the entry into the circular pipeline
- ensures that departures take place in order even if lookups complete out-of-order
Outline

- Example: IP Lookup
- Three solutions
  - Statically scheduled memory pipeline
  - Straight pipeline with uncoordinated memory references
  - Circular pipeline for 100% memory utilization
- Modeling RAMs
  - Synchronous vs. Asynchronous view
- Bluespec coding for circular pipeline

RAMs

- Basic memory components are "synchronous":
  - Present a read-address \( A_J \) on clock \( J \)
  - Data \( D_J \) arrives on clock \( J+N \)
  - If you don't "catch" \( D_J \) on clock \( J+N \), it may be lost, i.e., data \( D_{J+1} \) may arrive on clock \( J+1+N \)
- This kind of synchronicity can pervade the design and cause complications
Asynchronous RAMs

It's easier to work with an "asynchronous" block:

```
interface AsyncRAM lat addr data =
  read :: addr -> Action
  result :: data
  ack :: Action
```

- A sync memory can be converted into an async memory with a Bluespec function:

```
syncToAsync :: SyncRAM lat addr data ->
             AsyncRAM lat+1 addr data
```

RAMs: Synchronous vs Asynchronous

- The async memory has interface:
Outline

• Example: IP Lookup √
• Three solutions √
  – Statically scheduled memory pipeline
  – Straight pipeline with uncoordinated memory references
  – Circular pipeline for 100% memory utilization
• Modeling RAMs √
  – Synchronous vs. Asynchronous view
• Bluespec coding for circular pipeline ⇐

Completion buffer

interface CBuffer n a =
gToken :: CBToken n
gTokenAck :: Action
done :: CBToken n ->
  a -> Action
get :: a
ack :: Action

data CBToken n = CBToken (Bit (TLog n))

mkCBuffer =
module
  buf :: Array (Bit ln) (Maybe a) <- mkArray 0 hi
  i :: Reg (Bit ln) <- mkReg 0 -- input index
  o :: Reg (Bit ln) <- mkReg 0 -- output index
  n :: Counter ln1 <- mkCounter 0 -- number of filled slots
**Completion buffer**

\[
\text{mkCBuffer} :: (\log \ n \ \ln, \ \text{Add} \ \ln \ \ln 1) \Rightarrow \text{Module} (\text{CBuffer} \ n \ a)
\]

\[
\text{mkCBuffer} =
\]

\[
\text{module} \ \ldots \ \text{state elements} \ \text{buf}, \ i, \ o, \ n \ \ldots
\]

\[
\text{interface}
\]

\[
\begin{align*}
\text{getToken} & = \text{CBToken} \ i \\
& \text{when} \ n.\text{value} \leq \text{hi}
\end{align*}
\]

\[
\begin{align*}
\text{getTokenAck} & = \text{action} \ \text{incr} \ i \\
& \ \text{n.up} \\
& \ \text{buf}.\text{upd} \ i \ \text{Nothing} \\
& \ \text{when} \ n.\text{value} \leq \text{hi}
\end{align*}
\]

\[
\begin{align*}
\text{done} (\text{CBToken} \ t) \ a & = \text{buf}.\text{upd} \ t \ (\text{Just} \ a) \\
\text{get} & = x \ \text{when} \ n.\text{value} > 0, \ \text{Just} \ x & < - \ \text{buf}.\text{sub} \ o
\end{align*}
\]

\[
\begin{align*}
\text{ack} & = \text{action} \ \text{incr} \ o \\
& \ \text{n.down} \\
& \ \text{when} \ n.\text{value} > 0, \ \text{Just} \ x & < - \ \text{buf}.\text{sub} \ o
\end{align*}
\]

**Bluespec code: Circular pipeline**

\[
\text{mkLPM} :: \text{AsyncRAM} \ \text{lat LuAddr LuData} \rightarrow \text{Module} \ \text{LPM}
\]

\[
\text{mkLPM} \ \text{ram} =
\]

\[
\text{module}
\]

\[
\begin{align*}
\text{cb} & :: \text{CBuffer} \ \text{NStg LuResult} \ \leftarrow \text{mkCBuffer} \\
\text{tf} & :: \text{FIFO} (\text{CBToken}, \ \text{IPaddr}) \ \leftarrow \text{mkFIFO} \\
\text{ops} & :: \text{FIFO} (\text{CBToken}, \ \text{IPaddr}) \ \leftarrow \text{mkSizedFIFO} (\text{lat}+1) \\
\text{buf} & :: \text{FIFO} ((\text{CBToken}, \ \text{IPaddr}), \ \text{LuAddr})
\end{align*}
\]

\[
\text{rules} \ \ldots
\]

\[
\text{interface}
\]

\[
\begin{align*}
\text{luReq} \ \text{ipa} & = \text{action} \ \text{tf}.\text{enq} (\text{cb}.\text{getToken}, \ \text{ipa}) \\
\text{cb}.\text{getTokenAck} \\
\text{luResp} & = \text{cb}.\text{get} \\
\text{luRespAck} & = \text{cb}.\text{ack}
\end{align*}
\]
Circular pipeline rules

"Enter":
when \((\text{tok}, \text{ipa}) \leftarrow \text{tf.first}\)
\[\Rightarrow\] action \(\text{tf.deq}\)
\(\text{ram.read}(\text{zeroExt ipa}[31:16])\)
\(\text{ops.enq}(\text{tok}, \text{ipa} \ll 16)\)

"Complete":
when \((\text{Leaf res}) \leftarrow \text{ram.result},\)
\((\text{tok}, \text{ipa}) \leftarrow \text{ops.first}\)
\[\Rightarrow\] action \(\text{ram.ack}\)
\(\text{ops.deq}\)
\(\text{cb.done tok res}\)

"Recirculate":
when \((\text{Node addr}) \leftarrow \text{ram.result},\)
\((\text{tok}, \text{ipa}) \leftarrow \text{ops.first}\)
\[\Rightarrow\] action \(\text{ram.ack}\)
\(\text{ops.deq}\)
\(\text{buf.enq}((\text{tok}, \text{ipa} \ll 8),\)
\(\text{addr+(zeroExt ipa}[31:24])\)
Circular pipeline rules \textit{Continued-2}

\begin{itemize}
\item \textbf{Move}:
  \begin{verbatim}
  when (tokipa, addr) <= buf.first
  ==> action buf.deq
      ram.read addr
      ops.enq tokipa
  \end{verbatim}
  \text{notice the conflict!}

\item \textbf{Enter}:
  \begin{verbatim}
  when (tok, ipa) <= tf.first
  ==> action tf.deq
      ram.read (zeroExt ipa[31:16])
      ops.enq (tok, ipa << 16)
  \end{verbatim}
\end{itemize}

Some observations

\begin{itemize}
\item Rules conflicts should be carefully studied to make sure that there are no races
  \begin{itemize}
  \item Atomic firing of rules is very helpful in reasoning about such matters
  \end{itemize}
\item Buffer sizes have to be set appropriately otherwise RAM may not be fully utilized
\item Rule priorities have to be specified correctly to avoid deadlocks
\item Timing Closure may require insertion of more pipeline stages (i.e. FIFO buffers)!
\item Circular pipeline solution extends to IPv6 in a straightforward manner
\end{itemize}