Trie路由表设计文档

最简单的二进制Trie路由表，一共32层，每层一个流水线。

符号定义

符号	含义
W_N	Next-hop项地址宽度（**N**ext-hop Address **W**idth），即我们的路由表能够存储2^{W_N}个`<nexthop, port>`对。
W_{T_k}	第k层Trie节点地址宽度（**T**rie Node Address **W**idth），即我们的Trie树第k层最多有$2^{W_{T_k}}$个节点。
N	我们的路由表所能存下的最大路由条目数。

其中我们路由表能存的条目数有下限：

$$N \geq 2^{W_{T_{32}}}$$

即我们的路由表至少能存Trie树最后一层节点数个路由条目。

如果我们设计的路由表能够保存$N = 2^{13} = 8192$条路由表，那么我们最后一层BRAM的地址位宽要设计为13，听起来也不大。

存储

Next-hop项内存空间

这个内存空间用来存储`<nexthop_ip, port>`对，将其设计为共享内存，便与软件访问，也就是说设计成双口BRAM，一端用于硬件读写，一端留给软件读写。

这个空间的长为2^{W_N}，宽为$32 + 3 = 35$。

其中每行是这样一个`struct`：

```
typedef struct packed {
    logic[31:0] ip;
    logic[2:0]  port;
} nexthop_t;
```

一共有2^{W_N}行，因此我们可以用一个字宽为W_N的地址索引到一个Next-hop项。

在实际情况中，Next-hop的种类不会很多，因此这个表开到256就够用了，即令$W_N = 8$，因此这个空间总共大小为1.12KiB.

Trie节点内存空间

这个内存空间用来存储Trie树的节点，将其设计为共享内存，便与软件访问，也就是说设计成双口BRAM，一端用于硬件读，一端留给软件读写。

由于我们访问Trie树中的每一层都是流水线中的一级，它们都是并行的，因此我们Trie树每一层要单独开一个内存空间，一共开32个BRAM。

第k层空间的长为$2^{W_{T_k}}$，宽为$W_N + 2 \times W_{T_{k+1}} = 2 \times W_{T_{k+1}} + 8$。

其中每行是这样一个`struct`：

```
typedef struct packed {
    logic[W_N-1:0] nexthopAddr;
    logic[W_T_{k+1}-1:0] lcAddr;
    logic[W_T_{k+1}-1:0] rcAddr;
} trie_node_t;
```

一共有$2^{W_{T_k}}$行，因此我们可以用一个字宽为W_{T_k}的地址索引到一个Trie Node。

假设内存空间足够，金字塔式的空间分配是理想的，即第0层有1个节点，第1层有两个节点，……，第32层有2^{32}个节点。但显然我们没这么多内存，因此先限定住最后一层只能有2^{13}个节点，因此第13层~第32层就都只用设计成2^{13}个节点就足够了，之前12层逐层减半。

这种金字塔设计听起来节约内存，但是算一算就知道没节约多少内存（大概节约了$2^{13} * (12-1) * 34/8 \approx 400KiB$？），因此为了设计简便我们就不做金字塔设计了，直接所有层的容量均为2^{13}，因此所有的$W_{T_K} = 13$。

最终，我们有32个长为2^{13}，宽为$2 \times 13 + 8 = 34$的BRAM作为Trie节点内存空间，总共大小为1114KiB。

算法

约定

- 两种地址空间的首地址（地址为0）的内存我们都不用，这样当`nexthopAddr`为0我们就知道这个Trie Node不对应路由表项，当`lcAddr`或`rcAddr`为0我们就知道没有左or右孩子。
- 第一层地址为1的Trie Node是整个Trie树的根节点。

查找

```
input wire [31:0] i_ip,
output nexthop_t o_nexthop,
output reg o_valid,
output reg o_finish,
reg[7:0] nexthop_addr
```

沿着Trie树一级一级往下查，如果到某个匹配查到了就将o_valid置为1，并将nexthop_addr设为Next-hop表中的相应表项地址。

到了最后一层进行Next-Hop表访存，将nexthop_addr中的下一跳信息取出，存入o_nexthop中，并将o_finish置为1。

如果到了某一层某个Trie节点在应该转向的位置没有左or右孩子了怎么办？

没关系我们可以接着让往下流，因为每一层的0号节点都是连接到下一层的0号节点的，他们不会对我们的最终查询到的nexthop_addr信息产生影响，流到最后一层还是正确的结果。

此时有两种可能：

- o_valid = 1，说明我们找到了一个下一条信息，并且由于是一级一级往下找的，自然有最长前缀匹配特性。
- o_valid = 0，说明我们的路由表中没有对应的下一跳

插入和删除（更新）

插入和删除不进行硬件实现，而是提供给软件操作两个内存空间的接口，由软件来负责路由表项的插入和删除。

复杂度计算

空间复杂度

由以上设计可知，我们的路由表空间消耗在$1200KiB$以内，满足实验板的硬件条件。

时间复杂度

查询、更新的时间开销主要都是在查询上，都是32级流水线，流起来就快了。