l2:

```
Pins (eth, 112)   // size: 8*14
Pins (vlan, 32)   // size: 8*4
Pins (qinq, 64)   // size: 8*8

Abegin

//cella_pb(407bit*32)
//hdr_id(7)+mask(24*8b)+value(24*8b)+sub_id(7)+nxt_id(7)+bypass(2:mainbypass(1
)+subbypass(1))

0x1,{(eth, 96, 16) == 0x8100, (vlan, 16, 16) == 0x0800}, 0x1, 0x3, 1//eth+vlan+ipv4
0x1, {(eth, 96, 16) == 0x88a8, (qinq, 16, 16) == 0x8100, (qinq, 48, 16) ==
0x0800},0x2,0x3,1//eth+qinq+ipv4
0x1, {(eth, 96, 16) == 0x9200, (qinq, 16, 16) == 0x8100, (qinq, 48, 16) ==
0x0800},0x2,0x3,1//eth+qinq+ipv4
0x1, {(eth, 96, 16) == 0x9300, (qinq, 16, 16) == 0x8100, (qinq, 48, 16) ==
0x0800},0x2,0x3,1//eth+qinq+ipv4
0x1, {(eth, 96, 16) == 0x0800}, 0x3, 0x3, 1//eth+ipv4
0x1, {(eth, 96, 16) == 0x88cc}, 0x4, 0x6, 2//eth+lldp
//

Aend
```

注释：

1）Pins (eth, 112)，其中 eth是ethernet的实例，而ethernet各个fields的总字节数是14，总bit数即为122（8*14）；类似地，vlan是ieee802-1qTag的实例，总字节数是4；qinq是ieee802-1OuterTag的实例，总字节数是8。

2）上述6行，每一行分5个部分（参见文档中3.1的 <cella_pb_item>）：第1部分0x1为数字1,可暂时理解为l2是layer集合lset中的第1个;第2部分(如,,{(eth, 96, 16) == 0x8100, (vlan, 16, 16) == 0x0800}）代表一个组合条件，比较复杂，下面单独说；第3部分的0x1，0x2，0x3，0x4分别对应4条 <cella_pc_cur_item>中的1条，见跟随Abegin ... Aend 之后的 ACbegin ... ACend，其中每一条的第一项依次为0x1，0x2，0x3，0x4；第4部分对应next_header，如0x3代表ipv4（位于Pset的第3个位置），0x6代表lldp（位于Pset的第6个位置）；第4部分对应bypass，可能的取值有0,1,2.

3) 关于每一行的第2部分,对应于一个分支的条件,即用于查表的条件。比如,第1行的{(eth, 96, 16) == 0x8100, (vlan, 16, 16) == 0x0800}，对应test01.ppp中的条件(eth.etherType == 0x8100) && (vlan.etherType == 0x0800)。观察一下，etherType在protocol ethernet中的偏移位置和位数，可知其偏移量为96（0~95是dmac和smac占据），位数为16。类似地，

protocol ieee802-1qTag中etherType的偏移量为16，位数为16。第5行的{(eth, 96, 16) == 0x0800}对应，以及第5行的{(eth, 96, 16) == 0x88cc}对应。第2，3，4行的情况复杂一点，下面分开说。

4） 可将test01.ppp中的条件 ((eth.etherType == 0x88a8 || eth.etherType == 0x9200 || eth.etherType == 0x9300) && (qinq.ethertype_o == 0x8100) && (qinq.etherType_i == 0x0800))分解成3个合取项，其中一个为 eth.etherType == 0x88a8 &&eth.etherType == 0x9300) && (qinq.ethertype_o == 0x8100) && (qinq.etherType_i == 0x0800) ，类似于3）的解释，它对应于{(eth, 96, 16) == 0x88a8, (qinq, 16, 16) == 0x8100, (qinq, 48, 16) == 0x0800}，见上述第2行。那么，第3和第4行的{(eth, 96, 16) == 0x9200, (qinq, 16, 16) == 0x8100, (qinq, 48, 16) == 0x0800} 和 {(eth, 96, 16) == 0x9300, (qinq, 16, 16) == 0x8100, (qinq, 48, 16) == 0x0800}也就可以类似理解了。这里涉及到一个将条件分解为仅含合取项的问题。下面还会遇到另一种复杂的情况，需要将并行的条件分支进行组合。之前也提到过，这里提到的条件组合与分解，初步估计是翻译过程中最难的部分了。

****

ACbegin

//cella_pc_cur(328bit*32)
//vliw(320:alu(8*24b)+mov(8*8b)+set(8*8b))+lyr_offset(8)
0x1,{(mov (IRF,192,16), (vlan,0,16)),(set (IRF,16,8), 1), (set (IRF,24,8), 0)},
0x12//sub_id:01,mov {IRF_outer_vlan_high, IRF_outer_vlan_low},
{vlan.pcp,vlan.cfi,vlan.vid};set IRF_tag_type_2b, 1;set IRF_pkt_type_3b, 0;
0x2,{(mov (IRF,192,16), (vlan,0,16)),(mov (IRF,208,16), (vlan,0,16)),(set (IRF,16,8), 2), (set (IRF,24,8), 0)}, 0x16//sub_id:02,mov {IRF_outer_vlan_high, IRF_outer_vlan_low}, {qinq.pcp_o,qinq.cfi_o,qinq.vid_o};mov {IRF_inner_vlan_high, IRF_inner_vlan_low},
{qinq.pcp_i,qinq.cfi_i,qinq.vid_i};set IRF_tag_type_2b, 2;set IRF_pkt_type_3b, 0;
0x3,{(set (IRF,16,8), 0), (set (IRF,24,8), 0)}, 0xc//sub_id:03,set IRF_tag_type_2b, 0;set IRF_pkt_type_3b, 0;
0x4,{(set (IRF,32,8), 66)}, 0xc//sub_id:04,set IRF_l2_protocol_flag_type_8b, 66;
//

ACend

****

注释： ACbegin ... ACend 中的每一项对应 <cella_pc_cur_item> （参见文档中3.1）。测例中L2的这部分共有4项。每一项分3个部分。

1）第1部分对应一个索引号，分别为0x1，0x2，0x3，0x4。第3部分为 <lyr_offset>，原测例中用了这个名字，不清楚为什么是这个名字，但其实际上对应于当前分支对应的length取值。比如，上述 Abegin... Aend 的第1个分支中，有length = eth.length + vlan.length; ethernet节数14，加上 ieee802-1qTag字节数是4，等于18，即0x12，对应0x1这一项的第3部分。同样，第2、3、4分支中，有length = eth.length + qinq.length，14+8=22，即0x16，对应0x2这一项的第3部分。第5、6分支的length = eth.length=14，即 0xc，对应0x3和0x4这两项的第3部分。第2部分为所有动作（指令）的集合，这些指令在硬件中会并行执行，下面单独说明。

2）比如，0x1这一项的第2部分，{(mov (IRF,192,16), (vlan,0,16)),(set (IRF,16,8), 1), (set (IRF,24,8), 0)}, 对应于下列3条指令的集合：mov IRF_outer_vlan_high++IRF_outer_vlan_low, vlan.pcp++vlan.cfi++vlan.vid; set IRF_tag_type_2b, 1;set IRF_pkt_type_3b, 0。其中， IRF_outer_vlan_high 和 IRF_outer_vlan_low在ARegisters中的定义分别是：IRF_outer_vlan_high = IRF[199:192]; IRF_outer_vlan_low = IRF[207:200]，注意它们是连续的两个字节，对应IRF的192~207位，共16位，所以我们在asm中用(IRF,192,16)表示。而vlan.pcp++vlan.cfi++vlan.vid表示vlan（ieee802-1qTag的实例）中从开始连续的域pcp、cfi和vid的连接，长度也是16位，对应的偏移量为0，因此我们在asm中用(vlan,0,16)表示。另两条(set (IRF,16,8), 1)和(set (IRF,24,8), 0)分别对应两条set指令：set IRF_tag_type_2b, 1;和 set IRF_pkt_type_3b, 0;。从ARegisters中IRF_tag_type_2b和IRF_pkt_type_3b的定义容易看出二者分别对应(IRF,16,8)和(IRF,24,8)。对于0x2，0x3和0x4这三项的第2部分（指令集合）的理解是类似的。

****

ANbegin

//cella_pc_nxt(583bit*32)
//nxt_id(7)+pa_offset(3*24*8b:cellA(irf/2+fra/22)+cellB0(irf/2+fra/22)+cellB1(irf/2+fra/22))

0x3, {( ) + (0 9)}, {( ) +(0xc 0xd 0xe 0xf 0x10 0x11 0x12 0x13}, {( ) + (6 7)}//ipv4
0x6, {( ) + ( )}, {( ) + ( )}, {( ) + ( )}//lldp
//

ANend

****

注释： ANbegin ... ANend 中的每一项对应 <cella_pc_nxt_item> （参见文档中3.1）。测例中L2的这部分共有2项。每一项分2个部分。

1）第1部分对应 next_header 指向的协议号，第1行的 0x3 代表ipv4（位于Pset的第3个

位置），第2行的0x6代表lldp（位于Pset的第6个位置）。L2的 Abegin ... Aend 中第1~5项（分支）的next_header 指向0x3（ipv4），第6项（分支）的next_header 指向0x6（lldp）。
2）第2部分是一个稍微麻烦的地方，也略为难处理。由于例子不是很全，我只按目前的理解来设计，可能有偏差，需要时再修订。这一部分含两个子部分，第1个子个子部分与通用寄存器（这里指IRF）哪些位需要传递给后续层的后续层 CellA，CellB0 和 CellB1中使用，这里缺例子，不清楚如何设计，所以暂时将其置为空集（空表）。第2个子个子部分对应于在后续层 CellA，CellB0 和 CellB1的分支条件中要用到该协议（ipv4/lldp）的所有字段所在的偏移位置（以字节为单位），下面单独通过例子来解释。
3） 对于第1行，对应于 ipv4。{( ) + (0 9)}对应于CellA，{( ) +(0xc 0xd 0xe 0xf 0x10 0x11 0x12 0x13} 对应于CellB0，{( ) + (6 7)}对应于CellB1。它们的第一个部分对应于IRF位的偏移，均暂时置为空表（）。我们先来看CellB0和CellB1这两部分。参见test_01.ppp,ipv4会在L3层处理，见实例v4。在L3的cellB0中，v4的srcAddr和dstAddr字段被用在分支条件中，而在ipv4协议包头中，这两个字段的偏移位置处于第13，14，...,20字节，若从0开始编号，则对应的字节偏移位置分别为12，13，14，15，16，17，18，19，换作hexadecimal数字即为0xc，0xd 0xe 0xf 0x10 0x11 0x12 0x13。同样，在L3的cellB1中，v4的flagOffset和flags字段被用在分支条件中，而在ipv4协议包头中，这两个字段的偏移位置处于第7和8字节，由于从0开始编号，因此用表/集合(6 7)表示。理解了这两个之后，我们再来看CellA对应的部分。虽然在L3的cellA描述中没有直接的分支条件，但在ipv4协议中包含了分支，我们的设计默认在协议中的语句都将划归相应的cellA（这一点也是我们需要特别注意的地方），这就不难理解(0 9)这个表/集合了。在ipv4协议中包含的分支条件中用到字段ihl和theProtocol，可对应到协议包头中的第1和第10个字节，以0作为初始编号的话，就对应了(0 9)。
4）第2行对应的协议lldp（0x6），在当前测例中未给出，所以将所有的表/集合暂时置为空。

相对于cellA,与cellB0和cellB1对应的部分就简单多了。下面以L2的cellB0为例。cellB1如果有内容的话，与cellB0的结构一致。

**********//

B0begin

//cellb0_pb(398bit*32)
//hdr_id(7)+mask(24*8b)+value(24*8b)+sub_id(7)

0x1, {(eth, 0, 48) == 0xFFFFFFFFFFFF}, 0x1//eth.dmac == 0xFFFFFFFFFFFF
0x1, {(eth, 40, 1) == 1}, 0x2//eth.dmac[40] == 1

B0end

**********//

B0Cbegin

//cellb0_pc_cur(320bit*32)
//vliw(320:alu(8*24b)+mov(8*8b)+set(8*8b))
0x2, {(set (IRF,0,8), 3)}//sub_id:01,set IRF_l2_type = 3;
0x2, {(set (IRF,0,8), 2)}//sub_id:02,set IRF_l2_type = 2;
//

B0Cend

******

注释：

1）注意，cellA包含3个 begin ... end，而cellB0仅包含2个 begin ... end，而且其结构也更加简单，具体下面分开来说。cellB1也一样，略。

2）B0begin ... B0end中的每一项有3个部分，同 Abegin ... Aend 的前3部分对应，少了nxt_id和bypass两个部分。

3）B0Cbegin ... B0Cend中的每一项只有2个部分，同 Abegin ... Aend 的前2部分对应，少了lyr_offset。

还有最后一个重要方面，然后就差不多了。这就是前面提到过的，需要将并行的条件分支进行组合。L2 的当前描述中没有这种情况，在 L3 中有这样的情形。

##################
l3:

Pins (v4, 224)   // size: 8*28

Abegin

//cella_pb(407bit*32)
//hdr_id(7)+mask(24*8b)+value(24*8b)+sub_id(7)+nxt_id(7)+bypass(2:mainbypass(1)+subbypass(1))

```
0x3, {(v4, 4, 4) == 5, (v4, 72, 8) == 2}, 0x1, 0x9, 2//ihl == 5,theProtocol == 2
0x3, {(v4, 4, 4) == 5, (v4, 72, 8) == 4}, 0x2, 0x3, 0//ihl == 5,theProtocol == 4
0x3, {(v4, 4, 4) == 5, (v4, 72, 8) == 6}, 0x3, 0xc, 1//ihl == 5,theProtocol == 6
0x3, {(v4, 4, 4) == 5, (v4, 72, 8) == 0x11}, 0x4, 0xd, 1//ihl == 5,theProtocol ==
0x11
0x3, {(v4, 4, 4) == 6, (v4, 72, 8) == 2}, 0x5, 0x9, 2//ihl == 6,theProtocol == 2
0x3, {(v4, 4, 4) == 6, (v4, 72, 8) == 4}, 0x6, 0x3, 0//ihl == 6,theProtocol == 4
0x3, {(v4, 4, 4) == 6, (v4, 72, 8) == 6}, 0x7, 0xc, 1//ihl == 6,theProtocol == 6
0x3, {(v4, 4, 4) == 6, (v4, 72, 8) == 0x11}, 0x8, 0xd, 1//ihl == 6,theProtocol ==
0x11
0x3, {(v4, 4, 4) == 7, (v4, 72, 8) == 2}, 0x9, 0x9, 2//ihl == 7,theProtocol == 2
0x3, {(v4, 4, 4) == 7, (v4, 72, 8) == 4}, 0xa, 0x3, 0//ihl == 7,theProtocol == 4
0x3, {(v4, 4, 4) == 7, (v4, 72, 8) == 6}, 0xb, 0xc, 1//ihl == 7,theProtocol == 6
0x3, {(v4, 4, 4) == 7, (v4, 72, 8) == 0x11}, 0xc, 0xd, 1//ihl == 7,theProtocol ==
0x11

//

Aend

*********
```

注释:

1) L3 CellA中包含了 protocol ipv4 中描述的分支。放到protocol中，猜测是为了共享吧。顺便， 在 cellA 中所定义的语句应该对于各个分支是共享的内容。

2）这个例子中有12行，每行对应一个条件分支。0x3对应 l3。在ipv4的声明里，有两部分并行的条件分支。一部分对应以ihl取值的条件语句，另一部分对应以theProtocol取值的条件语句。除去else的情形（暂不考虑，后续如何考虑需要与硬件的设计进行沟通），ihl这部分对应有3个互斥的条件（ihl分别取值5，6和7），theProtocol这部分对应有4个互斥的条件（theProtocol分别取值2，4，6和0x11）。这样，就可以组合成3*4=12个分支条件，每个对应上面的一行。