可编程解析器设计

1. 整体设计

模块整体设计如图所示，每一级解析一层协议。每一层解析单元包含5级，分别是协议头定位（locate header）、关键字提取（Extract fields）、选择类型/长度字段（select type field）、查找协议类型（lookup type field）、关键字重组（recombine）。



图1 可编程解析器整体设计

* 1. 解析报文流程



图2 解析Ethernet协议

如图2所示，我们使用可编程解析器解析Ethernet协议，并提取源、目的MAC地址，其处理流程为：

1. 输入初始化的PHV，包含pkt header（即图中的ParsData）、Metadata（即途中的ProcData）和Type info；
2. Locate header根据Pre len定位Ethernet头部，即需要将Ethernet移置最左端。由于pre len等于0，所以无需移动parsData；
3. 根据Extract info提取MAC地址，即16b提取单元需要提取起始位置为0B的2B数据，16b提取单元需要提取起始位置为6B的2B数据，32b提取单元需要提取起始位置为2B的4B数据，32b提取单元需要提取起始位置为0B的4B数据；根据type field offset提取type字段（0x0800），根据length field offset提取长度字段（由于有效位为0，无需提取）；
4. 计算协议长度，采用定长，即14B
5. 根据Type查找协议类型，即根据0x0800获得下一层协议的处理信息（IPv4 info）；从提取字段中选择有效字段（有效长度为12B），即源、目的MAC地址
6. 将源、目的MAC地址根据construction info覆盖Metadata中，即填写在起始位置为0的元数据位置。
   1. 封装报文头流程



图3 封装Ethernet协议

如图3所示，我们使用可编程解析器封装Ethernet协议，其处理流程为：

1. 输入初始化的PHV，包含Metadata（即途中的ParsData），pkt header（即图中的ProcData）和Type info；
2. Locate header根据Pre len定位Ethernet头部，即需要将Ethernet移置最左端。由于pre len等于0，所以无需移动parsData；
3. 根据Extract info提取待封装Ethernet头，即16b提取单元需要提取起始位置为0B的2B数据，32b提取单元需要提取起始位置为2B的4B数据，64b提取单元需要提取起始位置为6B的8B数据；根据type field offset提取type字段（有效位为0，无需提取），根据length field offset提取长度字段（由于有效位为0，无需提取）；
4. 计算协议长度，采用定长，即14B；
5. 根据Type查找协议类型，即根据0x0获得下一层协议的处理信息（Ethernet info）；从提取字段中选择有效字段（有效长度为14B），即新Ethernet头；
6. 将新Ethernet头根据construction info覆盖pkt header中，即插入在起始位置为0的元数据位置。
   1. 解封装报文头流程



图4 解封装VLAN协议

如图4所示，我们使用可编程解析器解封装VLAN协议，其处理流程为：

1. 输入初始化的PHV，包含Metadata（即途中的ParsData），pkt header（即图中的ProcData）和Type info；
2. Locate header根据Pre len定位VLAN头部，即需要将VLAN移置最左端。由于pre len等于14，所以需要左移parsData14B；（值得注意到是，输出的parsData并未移动）
3. 根据Extract info无需提取数据；根据type field offset提取type字段（即0x0800），根据length field offset提取长度字段（由于有效位为0，无需提取）；
4. 计算协议长度，采用定长，即4B；
5. 根据Type查找协议类型，即根据0x0800获得下一层协议的处理信息（IPv4 info）；
6. 根据construction info删除VLAN头，即删除在起始位置为4的4B数据。
   1. 替换报文字段流程



图5 替换Ethernet协议字段

如图5所示，我们使用可编程解析器替换Ethernet协议字段，即替换源、目的MAC地址，其处理流程为：

1. 输入初始化的PHV，包含pkt header（即图中的ParsData）、Metadata（即途中的ProcData）和Type info；
2. Locate header根据Pre len定位Ethernet头部，即需要将Ethernet移置最左端。由于pre len等于0，所以无需移动parsData；
3. 根据Extract info提取MAC地址，即16b提取单元需要提取起始位置为0B的2B数据，16b提取单元需要提取起始位置为6B的2B数据，32b提取单元需要提取起始位置为2B的4B数据，32b提取单元需要提取起始位置为0B的4B数据；根据type field offset提取type字段（0x0800），根据length field offset提取长度字段（由于有效位为0，无需提取）；
4. 计算协议长度，采用定长，即14B
5. 根据Type查找协议类型，即根据0x0800获得下一层协议的处理信息（IPv4 info）；从提取字段中选择有效字段（有效长度为12B），即源、目的MAC地址
6. 将源、目的MAC地址根据construction info覆盖Metadata中，即填写在起始位置为0的12B数据。
7. 模块接口信号定义

接口信号如下表所示，包含两组信号，即规则读写，数据处理。

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | 方向 | 含义 |
| clk | 1 | input | 时钟 |
| reset | 1 | input | 复位，低有效 |
| Wren\_rule | 1 | input | 写规则，高有效 |
| Rden\_rule | 1 | input | 读规则 |
| addr\_rule | 3 | input | 规则ID，共8条规则  （每一级支持8种协议类型） |
| data\_rule | 177 | input | 规则内容  [176] 有效位  [175:160] 协议字段数值  [159:144] 协议字段掩码  [143:136] 下一层协议类型标识（未使用）  [135:128] 协议字段位置  [135] 有效位，表示要提取  [134:128] 在当前协议中的位置  [127:96] 长度字段信息  [127] 变长标识，1为变长  [126:120] 在协议中的位置  [119:108] 长度字段掩码  [107:96] 变长左移值/固定长度值  [95:32] 待提取关键字长度  [31:0] 重构元数据信息  [31:24] 将关键字填写入元数据的位置  [23:16] 待填写关键字/封装头长度  [15:8] 填写的动作类型，0为删除定长数据，1为封装定长数据，2为替换关键字；  [7:0] 当前头剩余长度 |
| rdata\_rule\_valid | 1 | output | 读规则有效位，高有效 |
| rdata\_rule | 177 | output | 读取的规则内容 |
| phv\_in\_valid | 1 | input | 输入数据有效位 |
| phv\_in | 2208 | input | 输入数据  [2207:2048] type info  [2207:2200] 已解析协议长度  [2199:2192] 已填写元数据长度  [2192:2048] 与data\_rule定义相同  [2047:1024] 待解析数据  [1023:0] 元数据 |
| phv\_out\_valid | 1 | output | 输出数据有效位 |
| phv\_out | 2208 | output | 输出数据，与输入数据定义相同 |

1. 模块内部实现

划分为10个阶段：

1-2阶段：功能为根据preLen定位ParsData当前协议的位置，即截取512b的协议数据

|  |
| --- |
| always @(posedge clk) begin  // stage 1: shift\_stage\_1, Aligned by 64b;  (\*parallel\_case, full\_case \*)  case(lenPars\_8b[0][6:4])  3'd0: parsDataTemp\_1024b <= parsData[0];  3'd1: parsDataTemp\_1024b <= {parsData[0][895:0],128'b0};  3'd2: parsDataTemp\_1024b <= {parsData[0][767:0],256'b0};  3'd7: parsDataTemp\_1024b <= {parsData[0][639:0],384'b0};  3'd3: parsDataTemp\_1024b <= {parsData[0][511:0],512'b0};  3'd4: parsDataTemp\_1024b <= {parsData[0][383:0],640'b0};  3'd5: parsDataTemp\_1024b <= {parsData[0][255:0],768'b0};  3'd6: parsDataTemp\_1024b <= {parsData[0][127:0],896'b0};  endcase  // stage 2: shift\_stage\_2, Aligned by 16b;  (\*parallel\_case, full\_case \*)  case(lenPars\_8b[1][3:1])  3'd0: parsDataTemp\_512b <= parsDataTemp\_1024b[1023:512];  3'd1: parsDataTemp\_512b <= parsDataTemp\_1024b[1007:496];  3'd2: parsDataTemp\_512b <= parsDataTemp\_1024b[991:480];  3'd3: parsDataTemp\_512b <= parsDataTemp\_1024b[975:464];  3'd4: parsDataTemp\_512b <= parsDataTemp\_1024b[959:448];  3'd5: parsDataTemp\_512b <= parsDataTemp\_1024b[943:432];  3'd6: parsDataTemp\_512b <= parsDataTemp\_1024b[927:416];  3'd7: parsDataTemp\_512b <= parsDataTemp\_1024b[911:400];  endcase  end |

3-4阶段：各个提取单元（1B，2B，4B，8B）从512b数据中提取关键字，需要两个阶段实现；根据constrOffset定位ProcData待修改字段的位置；生成更新ProcData的掩码（根据修改字段的长度，偏移）。代码如下：

|  |
| --- |
| 各个提取单元（1B，2B，4B，8B）从512b数据中提取关键字 |
| generate  genvar gen\_i;  for(gen\_i=0; gen\_i<2; gen\_i=gen\_i+1) begin: gen\_extractor  always @(posedge clk) begin  // stage 3: extract 64b from 512b: Aligned by 64b;  (\* parallel\_case, full\_case \*)  case(exOffset\_8b[2][(5+7\*gen\_i):(3+7\*gen\_i)])  3'd7: exField1\_8b[gen\_i] <= parsDataTemp\_512b[63:0];  3'd6: exField1\_8b[gen\_i] <= parsDataTemp\_512b[127:64];  3'd5: exField1\_8b[gen\_i] <= parsDataTemp\_512b[191:128];  3'd4: exField1\_8b[gen\_i] <= parsDataTemp\_512b[255:192];  3'd3: exField1\_8b[gen\_i] <= parsDataTemp\_512b[319:256];  3'd2: exField1\_8b[gen\_i] <= parsDataTemp\_512b[383:320];  3'd1: exField1\_8b[gen\_i] <= parsDataTemp\_512b[447:384];  3'd0: exField1\_8b[gen\_i] <= parsDataTemp\_512b[511:448];  endcase  (\* parallel\_case, full\_case \*)  case(exOffset\_16b[2][(5+7\*gen\_i):(3+7\*gen\_i)])  3'd7: exField1\_16b[gen\_i] <= {parsDataTemp\_512b[63:0],8'b0};  3'd6: exField1\_16b[gen\_i] <= parsDataTemp\_512b[127:64-8];  3'd5: exField1\_16b[gen\_i] <= parsDataTemp\_512b[191:128-8];  3'd4: exField1\_16b[gen\_i] <= parsDataTemp\_512b[255:192-8];  3'd3: exField1\_16b[gen\_i] <= parsDataTemp\_512b[319:256-8];  3'd2: exField1\_16b[gen\_i] <= parsDataTemp\_512b[383:320-8];  3'd1: exField1\_16b[gen\_i] <= parsDataTemp\_512b[447:384-8];  3'd0: exField1\_16b[gen\_i] <= parsDataTemp\_512b[511:448-8];  endcase  (\* parallel\_case, full\_case \*)  case(exOffset\_32b[2][(5+7\*gen\_i):(3+7\*gen\_i)])  3'd7: exField1\_32b[gen\_i] <= {parsDataTemp\_512b[63:0],24'b0};  3'd6: exField1\_32b[gen\_i] <= parsDataTemp\_512b[127:64-24];  3'd5: exField1\_32b[gen\_i] <= parsDataTemp\_512b[191:128-24];  3'd4: exField1\_32b[gen\_i] <= parsDataTemp\_512b[255:192-24];  3'd3: exField1\_32b[gen\_i] <= parsDataTemp\_512b[319:256-24];  3'd2: exField1\_32b[gen\_i] <= parsDataTemp\_512b[383:320-24];  3'd1: exField1\_32b[gen\_i] <= parsDataTemp\_512b[447:384-24];  3'd0: exField1\_32b[gen\_i] <= parsDataTemp\_512b[511:448-24];  endcase  (\* parallel\_case, full\_case \*)  case(exOffset\_64b[2][(5+7\*gen\_i):(3+7\*gen\_i)])  3'd7: exField1\_64b[gen\_i] <= {parsDataTemp\_512b[63:0],56'b0};  3'd6: exField1\_64b[gen\_i] <= parsDataTemp\_512b[127:64-56];  3'd5: exField1\_64b[gen\_i] <= parsDataTemp\_512b[191:128-56];  3'd4: exField1\_64b[gen\_i] <= parsDataTemp\_512b[255:192-56];  3'd3: exField1\_64b[gen\_i] <= parsDataTemp\_512b[319:256-56];  3'd2: exField1\_64b[gen\_i] <= parsDataTemp\_512b[383:320-56];  3'd1: exField1\_64b[gen\_i] <= parsDataTemp\_512b[447:384-56];  3'd0: exField1\_64b[gen\_i] <= parsDataTemp\_512b[511:448-56];  endcase  (\* parallel\_case, full\_case \*)  case(exOffsetTypeLength\_16b[2][(5+7\*gen\_i):(3+7\*gen\_i)])  3'd7: exField1\_typeLength\_16b[gen\_i] <= {parsDataTemp\_512b[63:0],8'b0};  3'd6: exField1\_typeLength\_16b[gen\_i] <= parsDataTemp\_512b[127:64-8];  3'd5: exField1\_typeLength\_16b[gen\_i] <= parsDataTemp\_512b[191:128-8];  3'd4: exField1\_typeLength\_16b[gen\_i] <= parsDataTemp\_512b[255:192-8];  3'd3: exField1\_typeLength\_16b[gen\_i] <= parsDataTemp\_512b[319:256-8];  3'd2: exField1\_typeLength\_16b[gen\_i] <= parsDataTemp\_512b[383:320-8];  3'd1: exField1\_typeLength\_16b[gen\_i] <= parsDataTemp\_512b[447:384-8];  3'd0: exField1\_typeLength\_16b[gen\_i] <= parsDataTemp\_512b[511:448-8];  endcase  // stage 4: extract 8b/16b/32b from 64b;  (\* parallel\_case, full\_case \*)  case(exOffset\_8b[3][(2+7\*gen\_i):(7\*gen\_i)])  3'd7: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][7:0];  3'd6: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][15:8];  3'd5: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][23:16];  3'd4: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][31:24];  3'd3: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][39:32];  3'd2: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][47:40];  3'd1: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][55:48];  3'd0: exField2\_8b[gen\_i] <= exField1\_8b[gen\_i][63:56];  endcase  (\* parallel\_case, full\_case \*)  case(exOffset\_16b[3][(2+7\*gen\_i):(7\*gen\_i)])  3'd7: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][7+8:0];  3'd6: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][15+8:8];  3'd5: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][23+8:16];  3'd4: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][31+8:24];  3'd3: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][39+8:32];  3'd2: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][47+8:40];  3'd1: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][55+8:48];  3'd0: exField2\_16b[gen\_i] <= exField1\_16b[gen\_i][63+8:56];  endcase  (\* parallel\_case, full\_case \*)  case(exOffset\_32b[3][(2+7\*gen\_i):(7\*gen\_i)])  3'd7: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][7+24:0];  3'd6: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][15+24:8];  3'd5: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][23+24:16];  3'd4: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][31+24:24];  3'd3: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][39+24:32];  3'd2: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][47+24:40];  3'd1: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][55+24:48];  3'd0: exField2\_32b[gen\_i] <= exField1\_32b[gen\_i][63+24:56];  endcase  (\* parallel\_case, full\_case \*)  case(exOffset\_64b[3][(2+7\*gen\_i):(7\*gen\_i)])  3'd7: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][63:0];  3'd6: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][71:8];  3'd5: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][79:16];  3'd4: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][87:24];  3'd3: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][95:32];  3'd2: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][103:40];  3'd1: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][111:48];  3'd0: exField2\_64b[gen\_i] <= exField1\_64b[gen\_i][119:56];  endcase  if(bmExTypeLength\_2b[gen\_i]) begin  (\* parallel\_case, full\_case \*)  case(exOffsetTypeLength\_16b[3][(2+7\*gen\_i):(7\*gen\_i)])  3'd7: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][7+8:0];  3'd6: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][15+8:8];  3'd5: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][23+8:16];  3'd4: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][31+8:24];  3'd3: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][39+8:32];  3'd2: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][47+8:40];  3'd1: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][55+8:48];  3'd0: exField2\_typeLength\_16b[gen\_i] <= exField1\_typeLength\_16b[gen\_i][63+8:56];  endcase  end  else begin  exField2\_typeLength\_16b[gen\_i] <= 16'b0;  end  end  end  endgenerate |
| 根据constrOffset定位ProcData待修改字段的位置。 |
| (\*parallel\_case, full\_case \*)  case(constrOffset\_8b[2][5:3])  3'd0: procData[3] <= procData[2];  3'd1: procData[3] <= {procData[2][64\*7+511:0], procData[2][1023:64\*7+512]};  3'd2: procData[3] <= {procData[2][64\*6+511:0], procData[2][1023:64\*6+512]};  3'd3: procData[3] <= {procData[2][64\*5+511:0], procData[2][1023:64\*5+512]};  3'd4: procData[3] <= {procData[2][64\*4+511:0], procData[2][1023:64\*4+512]};  3'd5: procData[3] <= {procData[2][64\*3+511:0], procData[2][1023:64\*3+512]};  3'd6: procData[3] <= {procData[2][64\*2+511:0], procData[2][1023:64\*2+512]};  3'd7: procData[3] <= {procData[2][64\*1+511:0], procData[2][1023:64+512]};  endcase  // stage 4: shift\_stage\_2, Aligned by 16b;  (\*parallel\_case, full\_case \*)  case(constrOffset\_8b[3][2:0])  3'd0: procData[4] <= procData[3];  3'd1: procData[4] <= {procData[3][8\*7+959:0], procData[3][1023:8\*7+960]};  3'd2: procData[4] <= {procData[3][8\*6+959:0], procData[3][1023:8\*6+960]};  3'd3: procData[4] <= {procData[3][8\*5+959:0], procData[3][1023:8\*5+960]};  3'd4: procData[4] <= {procData[3][8\*4+959:0], procData[3][1023:8\*4+960]};  3'd5: procData[4] <= {procData[3][8\*3+959:0], procData[3][1023:8\*3+960]};  3'd6: procData[4] <= {procData[3][8\*2+959:0], procData[3][1023:8\*2+960]};  3'd7: procData[4] <= {procData[3][8\*1+959:0], procData[3][1023:8\*1+960]};  endcase |
|  |
| 生成更新ProcData的掩码（根据修改字段的长度，偏移） |
| // stage 3: calculate mask, aligned by 32b  (\*parallel\_case, full\_case \*)  case(constrLength\_8b[2][4:2])  3'd0: exMask0\_240b <= {240{1'b1}};  3'd1: exMask0\_240b <= {{32{1'b0}}, {208{1'b1}}};  3'd2: exMask0\_240b <= {{64{1'b0}}, {176{1'b1}}};  3'd3: exMask0\_240b <= {{96{1'b0}}, {144{1'b1}}};  3'd4: exMask0\_240b <= {{128{1'b0}}, {112{1'b1}}};  3'd5: exMask0\_240b <= {{160{1'b0}}, {80{1'b1}}};  3'd6: exMask0\_240b <= {{192{1'b0}}, {48{1'b1}}};  3'd7: exMask0\_240b <= {{224{1'b0}}, {16{1'b1}}};  endcase  // stage 4: calculate mask, aligned by 8b  (\*parallel\_case, full\_case \*)  case(constrLength\_8b[3][1:0])  3'd0: exMask1\_240b <= exMask0\_240b;  3'd1: exMask1\_240b <= {{8{1'b0}}, exMask0\_240b[239:8]};  3'd2: exMask1\_240b <= {{16{1'b0}}, exMask0\_240b[239:16]};  3'd3: exMask1\_240b <= {{24{1'b0}}, exMask0\_240b[239:24]};  endcase |

5-6阶段：合并提取的字段（最多1B、2B、4B、8B各两个）；根据掩码将待修改字段置0；将procData划分为两段，方便删除、插入数据；根据type字段查找规则表；计算协议长度；代码如下：

|  |
| --- |
| 合并提取的字段（最多1B、2B、4B、8B各两个） |
| // // stage 5: calculate valid length in 48b;  bmLen\_7b[0] <= (bmEx\_8b[4][3:0] == 4'b0000)? 1'b1 : 1'b0; // length is 0;  bmLen\_7b[1] <= (bmEx\_8b[4][3:0] == 4'b0001)? 1'b1 : 1'b0; // length is 8;  bmLen\_7b[2] <= (bmEx\_8b[4][3:0] == 4'b0011 || bmEx\_8b[4][3:0] == 4'b0100)? 1'b1 : 1'b0;  bmLen\_7b[3] <= (bmEx\_8b[4][3:0] == 4'b0101)? 1'b1 : 1'b0;  bmLen\_7b[4] <= (bmEx\_8b[4][3:0] == 4'b1100 || bmEx\_8b[4][3:0] == 4'b0111)? 1'b1 : 1'b0;  bmLen\_7b[5] <= (bmEx\_8b[4][3:0] == 4'b1101)? 1'b1 : 1'b0;  bmLen\_7b[6] <= (bmEx\_8b[4][3:0] == 4'b1111)? 1'b1 : 1'b0;  // stage 5: merge 8b and 16b extracted fields, and merge 32b and 64b  (\* parallel\_case, full\_case \*)  case(bmEx\_8b[4][1:0])  2'b00: exField\_48b <= {exField2\_16b[0],exField2\_16b[1],16'b0};  2'b01: exField\_48b <= {exField2\_8b[0],exField2\_16b[0],exField2\_16b[1],8'b0};  2'b11: exField\_48b <= {exField2\_8b[0],exField2\_8b[1],exField2\_16b[0],exField2\_16b[1]};  2'b10: exField\_48b <= 48'b0;// exception  endcase  (\* parallel\_case, full\_case \*)  case(bmEx\_8b[4][5:4])  2'b00: exField\_192b <= {exField2\_64b[0],exField2\_64b[1],64'b0};  2'b01: exField\_192b <= {exField2\_32b[0],exField2\_64b[0],exField2\_64b[1],32'b0};  2'b11: exField\_192b <= {exField2\_32b[0],exField2\_32b[1],exField2\_64b[0],exField2\_64b[1]};  2'b10: exField\_192b <= 192'b0;// exception  endcase  // stage 6: merge 48b with 192b  (\* parallel\_case \*)  case(1'b1)  bmLen\_7b[0]: exField\_240b <= {exField\_192b, 48'b0};  bmLen\_7b[1]: exField\_240b <= {exField\_48b[47:40], exField\_192b, 40'b0};  bmLen\_7b[2]: exField\_240b <= {exField\_48b[47:32], exField\_192b, 32'b0};  bmLen\_7b[3]: exField\_240b <= {exField\_48b[47:24], exField\_192b, 24'b0};  bmLen\_7b[4]: exField\_240b <= {exField\_48b[47:16], exField\_192b, 16'b0};  bmLen\_7b[5]: exField\_240b <= {exField\_48b[47:8], exField\_192b, 8'b0};  bmLen\_7b[6]: exField\_240b <= {exField\_48b, exField\_192b};  endcase |
|  |
| 根据掩码将待修改字段置0； |
| //\* stage 5: mask procData;  if(constrAct\_8b[4] == 8'd2)  procData[5] <= {(procData[4][1023:784]&exMask1\_240b),procData[4][784:0]};  else  procData[5] <= procData[4];  notExMask0\_240b <= ~exMask1\_240b; |
|  |
| 将procData划分为两段，方便删除、插入数据 |
| //\* stage 6: procData is devided into two part  procData1\_1024b[6] <= 1024'b0;  procData2\_1024b[6] <= 1024'b0;  (\*parallel\_case, full\_case \*)  case(lenProc\_8b[5][6:4])  3'd0: {procData1\_1024b[6][1023:128],procData2\_1024b[6][127:0]} <= procData[5];  3'd1: {procData1\_1024b[6][1023:256],procData2\_1024b[6][255:0]} <= procData[5];  3'd2: {procData1\_1024b[6][1023:384],procData2\_1024b[6][383:0]} <= procData[5];  3'd3: {procData1\_1024b[6][1023:512],procData2\_1024b[6][511:0]} <= procData[5];  3'd4: {procData1\_1024b[6][1023:640],procData2\_1024b[6][639:0]} <= procData[5];  3'd5: {procData1\_1024b[6][1023:768],procData2\_1024b[6][767:0]} <= procData[5];  3'd6: {procData1\_1024b[6][1023:896],procData2\_1024b[6][895:0]} <= procData[5];  3'd7: procData2\_1024b[6] <= procData[5];  endcase  notExMask1\_240b <= notExMask0\_240b; |
|  |
| 根据type字段查找规则表，命中的规则形成bmHitRule |
| /\*\* lookup type \*/  //\* stage 5: lookup rule;  for(i=0; i<num\_rule; i=i+1) begin  if((validRule[i] == 1'b1) && ((exField2\_typeLength\_16b[0]&ruleMask[i]) == ruleKey[i]))  bmHitRule[i] <= 1'b1;  else  bmHitRule[i] <= 1'b0;  end  // stage 6: get action;  (\* parallel\_case \*)  casez(bmHitRule)  8'b1???????: ruleActionHit\_144b[6] <= ruleAction\_144b[7];  8'b01??????: ruleActionHit\_144b[6] <= ruleAction\_144b[6];  8'b001?????: ruleActionHit\_144b[6] <= ruleAction\_144b[5];  8'b0001????: ruleActionHit\_144b[6] <= ruleAction\_144b[4];  8'b00001???: ruleActionHit\_144b[6] <= ruleAction\_144b[3];  8'b000001??: ruleActionHit\_144b[6] <= ruleAction\_144b[2];  8'b0000001?: ruleActionHit\_144b[6] <= ruleAction\_144b[1];  8'b00000001: ruleActionHit\_144b[6] <= ruleAction\_144b[0];  8'b00000000: ruleActionHit\_144b[6] <= 144'b0;  endcase |
|  |
| 计算协议长度 |
| // stage 5: calc length;  lenCalc\_8b <= exField2\_typeLength\_16b[1][7:0]&lenMask\_16b[4][7:0];  // stage 6: get previous length;  if(bmExTypeLength\_2b[5][1] == 1'b0) begin  //\* fixed length  lenPars\_8b[6] <= fixLen\_or\_shiftVal\_8b[5] + lenPars\_8b[5];  end  else begin  (\* parallel\_case, full\_case \*)  case(fixLen\_or\_shiftVal\_8b[5][1:0])  2'd0: lenPars\_8b[6] <= lenCalc\_8b + lenPars\_8b[5];  2'd1: lenPars\_8b[6] <= {lenCalc\_8b[6:0],1'b0} + lenPars\_8b[5];  2'd2: lenPars\_8b[6] <= {lenCalc\_8b[5:0],2'b0} + lenPars\_8b[5];  2'd3: lenPars\_8b[6] <= {lenCalc\_8b[4:0],3'b0} + lenPars\_8b[5];  endcase  end |

7阶段：若需要替换字段，则根据constrInfo更新协议相关字段，即求&即可，因为之前待修改字段已经通过掩码置0。代码如下：

|  |
| --- |
| //\* stage 7: replace header;  if(constrAct\_8b[6] == 8'd2)  procData1\_1024b[7] <= {(procData1\_1024b[6][1023:784]|(exField\_240b&notExMask1\_240b)),procData1\_1024b[6][783:0]};  else  procData1\_1024b[7] <= procData1\_1024b[6];  procData2\_1024b[7] <= procData2\_1024b[6];  exFieldTemp\_240b <= exField\_240b; |

8-9阶段：若需要删除、插入数据，则根据constrInfo更新数据，移位操作同样采用两级case实现。代码如下：

|  |
| --- |
| //\* stage 8: left shift part one, aligned by 64b  (\* parallel\_case, full\_case \*)  case(constrShiftVal\_8b[7][5:3])  3'd0: procData1\_1024b[8] <= procData1\_1024b[7];  3'd1: procData1\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd1)? procData1\_1024b[7]:{procData1\_1024b[7][959:0], 64'b0};  3'd2: procData1\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd1)? procData1\_1024b[7]:{procData1\_1024b[7][895:0], 128'b0};  3'd3: procData1\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd1)? procData1\_1024b[7]:{procData1\_1024b[7][831:0], 192'b0};  3'd4: procData1\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd1)? procData1\_1024b[7]:{procData1\_1024b[7][767:0], 256'b0};  3'd5: procData1\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd1)? procData1\_1024b[7]:{procData1\_1024b[7][703:0], 320'b0};  3'd6: procData1\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd1)? procData1\_1024b[7]:{procData1\_1024b[7][639:0], 384'b0};  3'd7: procData1\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd1)? procData1\_1024b[7]:{procData1\_1024b[7][575:0], 448'b0};  endcase  //\* left shift part two, aligned by 64b  (\* parallel\_case, full\_case \*)  case(constrShiftVal\_8b[7][5:3])  3'd0: procData2\_1024b[8] <= procData2\_1024b[7];  3'd1: procData2\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd0)? procData2\_1024b[7]: (constrAct\_8b[7][1:0] == 2'd1)?  {procData2\_1024b[7][959:0], exFieldTemp\_240b[239:176]}:  {procData2\_1024b[7][959:0], procData1\_1024b[7][1023:960]};  3'd2: procData2\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd0)? procData2\_1024b[7]: (constrAct\_8b[7][1:0] == 2'd1)?  {procData2\_1024b[7][895:0], exFieldTemp\_240b[239:112]}:  {procData2\_1024b[7][895:0], procData1\_1024b[7][1023:896]};  3'd3: procData2\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd0)? procData2\_1024b[7]: (constrAct\_8b[7][1:0] == 2'd1)?  {procData2\_1024b[7][831:0], exFieldTemp\_240b[239:48]}:  {procData2\_1024b[7][831:0], procData1\_1024b[7][1023:832]};  3'd4: procData2\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd0)? procData2\_1024b[7]:  {procData2\_1024b[7][767:0], procData1\_1024b[7][1023:768]};  3'd5: procData2\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd0)? procData2\_1024b[7]:  {procData2\_1024b[7][703:0], procData1\_1024b[7][1023:704]};  3'd6: procData2\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd0)? procData2\_1024b[7]:  {procData2\_1024b[7][639:0], procData1\_1024b[7][1023:640]};  3'd7: procData2\_1024b[8] <= (constrAct\_8b[7][1:0] == 2'd0)? procData2\_1024b[7]:  {procData2\_1024b[7][575:0], procData1\_1024b[7][1023:576]};  endcase  (\* parallel\_case, full\_case \*)  case(constrShiftVal\_8b[7][4:3])  2'd0: exFieldTemp\_64b <= exFieldTemp\_240b[239:196];  2'd1: exFieldTemp\_64b <= exFieldTemp\_240b[175:112];  2'd2: exFieldTemp\_64b <= exFieldTemp\_240b[111:48];  2'd3: exFieldTemp\_64b <= {exFieldTemp\_240b[47:0], 16'b0};  endcase    //\* stage9: shift, aligned by 8b  (\* parallel\_case, full\_case \*)  case(constrShiftVal\_8b[8][2:0])  3'd0: procData1\_1024b[9] <= procData1\_1024b[8];  3'd1: procData1\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd1)? procData1\_1024b[8]:{procData1\_1024b[8][1015:0],8'b0};  3'd2: procData1\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd1)? procData1\_1024b[8]:{procData1\_1024b[8][1007:0],16'b0};  3'd3: procData1\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd1)? procData1\_1024b[8]:{procData1\_1024b[8][999:0], 24'b0};  3'd4: procData1\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd1)? procData1\_1024b[8]:{procData1\_1024b[8][991:0], 32'b0};  3'd5: procData1\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd1)? procData1\_1024b[8]:{procData1\_1024b[8][983:0], 40'b0};  3'd6: procData1\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd1)? procData1\_1024b[8]:{procData1\_1024b[8][975:0], 48'b0};  3'd7: procData1\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd1)? procData1\_1024b[8]:{procData1\_1024b[8][967:0], 56'b0};  endcase  (\* parallel\_case, full\_case \*)  case(constrShiftVal\_8b[7][2:0])  3'd0: procData2\_1024b[9] <= procData2\_1024b[8];  3'd1: procData2\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd0)? procData2\_1024b[8]: (constrAct\_8b[8][1:0] == 2'd1)?  {procData2\_1024b[8][1015:0], exFieldTemp\_64b[63:56]}:  {procData2\_1024b[8][1015:0], procData1\_1024b[8][1023:1016]};  3'd2: procData2\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd0)? procData2\_1024b[8]: (constrAct\_8b[8][1:0] == 2'd1)?  {procData2\_1024b[8][1007:0], exFieldTemp\_64b[55:48]}:  {procData2\_1024b[8][1007:0], procData1\_1024b[8][1023:1008]};  3'd3: procData2\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd0)? procData2\_1024b[8]: (constrAct\_8b[8][1:0] == 2'd1)?  {procData2\_1024b[8][999:0], exFieldTemp\_64b[47:40]}:  {procData2\_1024b[8][999:0], procData1\_1024b[8][1023:1000]};  3'd4: procData2\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd0)? procData2\_1024b[8]: (constrAct\_8b[8][1:0] == 2'd1)?  {procData2\_1024b[8][991:0], exFieldTemp\_64b[47:32]}:  {procData2\_1024b[8][991:0], procData1\_1024b[8][1023:992]};  3'd5: procData2\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd0)? procData2\_1024b[8]: (constrAct\_8b[8][1:0] == 2'd1)?  {procData2\_1024b[8][983:0], exFieldTemp\_64b[47:24]}:  {procData2\_1024b[8][983:0], procData1\_1024b[8][1023:984]};  3'd6: procData2\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd0)? procData2\_1024b[8]: (constrAct\_8b[8][1:0] == 2'd1)?  {procData2\_1024b[8][975:0], exFieldTemp\_64b[47:16]}:  {procData2\_1024b[8][975:0], procData1\_1024b[8][1023:976]};  3'd7: procData2\_1024b[9] <= (constrAct\_8b[8][1:0] == 2'd0)? procData2\_1024b[8]: (constrAct\_8b[8][1:0] == 2'd1)?  {procData2\_1024b[8][967:0], exFieldTemp\_64b[47:8]}:  {procData2\_1024b[8][967:0], procData1\_1024b[8][1023:968]};  endcase |

10阶段：合并所有关键字。代码如下

|  |
| --- |
| //\* stage 10: recomebine \*/  always @(posedge clk or negedge reset) begin  if(!reset) begin  phv\_out\_valid <= 1'b0;  phv\_out <= 0;  end  else begin  phv\_out\_valid <= valid\_temp[9];  phv\_out <= {lenPars\_8b[9], lenProc\_8b[9], ruleActionHit\_144b[9], parsData[9],  procData1\_1024b[9]|procData2\_1024b[9]};  end  end |

1. 综合结果

基于Xilinx Virtex-7 (xc7v690tffg1761-3)实现可编程解析器，测得其资源开销为：

|  |  |  |  |
| --- | --- | --- | --- |
| 总资源 | Slice LUTs（433200） | Slice Register（866400） | Block Memory（850） |
| 使用资源 | 29648 | 16386 | 0 |

可综合频率为：freq = 1000/(2+0.450)= 408.163265306 MHz