



## **PPL IXP2000-Family Device Map**

**January 25, 2005**

**Copyright © IP Fabrics, Inc. 2005**

Information in this document is furnished in connection with IP Fabrics products. No license, express or implied, to any intellectual property rights is granted by this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license.

Copyright © 2005, IP Fabrics, Inc. All rights reserved.

Packet Processing Language™, PPL™ and PPL-VM™ are owned and copyrighted by IP Fabrics, Inc. Microsoft®, Windows® and Windows® XP are registered trademarks of Microsoft Corporation.

Linux® is a registered trademark of Linus Torvalds.

Red Hat® is a registered trademark of Red Hat, Inc.

MontaVista® is a registered trademark of MontaVista Software Inc.

Intel® and Pentium® are registered trademarks of Intel Corporation.

\*Other brands, trademarks and names are property of their respective owners.

## Contents

Note: Items that appear with a gray background are not available in the current release. They are included here in order to present the complete language definition.

Contents .....	2
Device_map for the Intel IXP2xxx NPUs .....	4
NPU Model .....	5
Physical Memory Specification .....	6
Execution Resource Allocation.....	7
AVAILABLE_PROCESSORS(me_num,me_num).....	7
PPL_PROCESSORS(function(percentage), function(percentage),...) .....	7
Memory Usage.....	9
PACKET_MEM(DRAM,ksize[,prepend]).....	9
SADB_MEM(memory,ksize).....	9
LOG_MEM(DRAM,ksize[,bytes]).....	9
JOURNAL_MEM(memory,ksize).....	9
CONNECTIONS_MEM(memory).....	9
ASSOCIATIONS_MEM(memory) .....	9
ARRAY_MEM(memory).....	9
Array Map.....	10
ARRAY_MAP(array,location[,memory]) .....	10
Physical Network Interfaces (LINK Parameter).....	11
LINK(logical_port_number,direction,link_type,link-type-dependent values) .....	11
External Program Binding .....	14
PROG(logical_port_number/external_name,prog_type,prog-type-dependent values) .....	14
Packet Control.....	17
PACKET_CONTROL(flow_ordering,arp) .....	17
Optional Rejection of Malformed Packets.....	18
REJECT_MALFORMED( <i>list of constants</i> ).....	18
Connection Tables and Association Tables Controls .....	19
CONNECTIONS_CONTROL(collision_tries,timeout_rate,virtual_network) .....	19
ASSOCIATIONS_CONTROL(collision_tries,timeout_rate) .....	19
Debug Controls .....	20
DEBUG(event,mode,one or more values depending on mode) .....	20
Implementation-Specific FORWARD Actions .....	21
Implementation-Specific Exceptions .....	22

## Device\_map for the Intel IXP2xxx NPUs

PPL proper is a language that is independent of a specific processor model or architecture. However, there are certain relationship that need to be expressed between PPL and a specific hardware implementation underneath of processing units, memories, physical network ports, etc. The Device\_map statement of PPL serves this purpose. Device mapping is highly implementation specific. What is presented here is the definition of device mapping for the Intel IXP2xxx network processors.

Note that all parameter names and named values are local to the DeviceMap statement and thus are not global PPL reserved names. Lower-case names used herein to represent values must be numerical constants unless otherwise noted.

### DeviceMap

NPU(model,speed,options)  
 MEMORY\_MBYTES(dram,sram0,sram1,sram2,sram3)  
 AVAILABLE\_MEMORY\_BASE(dram,sram0,sram1,sram2,sram3,scratch)  
 AVAILABLE\_PROCESSORS(me\_num,me\_num)  
 PPL\_PROCESSORS(function(percentage), function(percentage),...)  
 PACKET\_MEM(memory,ksize,prepend)  
 SADB\_MEM(memory,ksize)  
 LOG\_MEM(memory,ksize)  
 JOURNAL\_MEM(memory,ksize)  
 CONNECTIONS\_MEM(memory)  
 ASSOCIATIONS\_MEM(memory)  
 ARRAY\_MEM(memory)  
 ARRAY\_MAP(array,location,memory)  
 LINK(logical\_port\_number,direction,link\_type,link-type-dependent values)  
 PROG(logical\_port\_number/external\_name,prog\_type,prog-type-dependent values)  
 REJECT\_MALFORMED(checks)  
 PACKET\_CONTROL(flow\_ordering,arp)  
 CONNECTIONS\_CONTROL(collision\_tries,timeout\_rate,virtual\_network)  
 ASSOCIATIONS\_CONTROL(collision\_tries,timeout\_rate)  
 DEBUG(event,mode,one or more values depending on mode)

The parameters are explained below in groups. NPU is the only required parameter; all others are optional.

## ***NPU Model***

The NPU parameter specifies the NPU model and speed and optionally certain additional attributes.

NPU(model,speed[,options])

Model	defines the NPU model number. Valid values are 2350, 2800, and 2850.
Speed	defines the microengine speed in MHz. For instance, NPU(2850,1400) expressed an IXP2850 running at 1.4 GHz. For the 2350, the value should be 300, 600, or 900. PPL uses this value to calculate its time base for operations related to time.
Options	for the 2800 or 2850, this must be absent. For the 2350, it must be 1..5. These represent the five alternative ways of configuring the media and switch fabric interface: <ol style="list-style-type: none"><li>1 32-bit SPI-3 interface</li><li>2 16-bit SPI-3, internal GIGE0, internal NPE</li><li>3 16-bit SPI-3, internal GIGE0 and GIGE1</li><li>4 Internal GIGE0 and GIGE1, internal NPE</li><li>5 same as description 3</li></ol>

## ***Physical Memory Specification***

`MEMORY_MBYTES(dram,sram0,sram1,sram2,sram3)`

Specifies the size in megabytes of the NPU DRAM (RDRAM in IXP28xx, CPP-DDR in IXP23xx) and of the four SRAMs (on QDR busses). An sram value of 0 denotes no SRAM present on that channel. For instance stating `MEMORY_MBYTES(1024,8,8,0,0)` denotes 1 GB of DRAM, 8 MB each of SRAM on channels 0 and 1, and no memory on channels 2 and 3.

`AVAILABLE_MEMORY_BASE(dram,sram0,sram1,sram2,sram3,scratch)`

Specifies the lowest addresses in each addressable memory that the PPL implementation may use for its own purposes. If this parameter is missing, the values 0 are assumed. Typically this would be the case. An exception is where there is software outside of the scope of PPL that coexists in the NPU and needs its own memory. For instance, stating `AVAILABLE_MEMORY_BASE(0x01000000,0x00100000,0,0,0x00000100)` reserves the first 16 MB of DRAM, the first one MB of the SRAM on channel 0, and the first 256 words of the scratch RAM on chip.

If omitted the default values are

`MEMORY_MBYTES(1024,64,0,0,0)`  
`AVAILABLE_MEMORY_BASE(0,0,0,0,0,0)`

### **IXP23xx Considerations**

In the IXP23xx, the first channel of SRAM is optional, the second channel is fixed in size as 128 KB and is on-chip, and the other two channels do not physically exist. Therefore 0 must be specified for sram2 and sram3, and any value specified for sram1 in `MEMORY_MBYTES` is ignored.

## ***Execution Resource Allocation***

These parameters specify how the micro engines and other resources are to be allocated.

### **AVAILABLE\_PROCESSORS(me\_num,me\_num)**

Specifies specific set of processors that may be used by PPL. If not specified, all processors are available. The two values specify the numbers of the lowest and highest available microengines. Note that in the IXP23xx and IXP28xx, the microengines are numbered 0-3 and 0-15 respectively. For instance, on an IXP2800, specifying AVAILABLE\_PROCESSORS(2,15) means that microengines 2-15 are available and thus 0 and 1 are not (e.g., might be used by software outside of the content of PPL).

### **PPL\_PROCESSORS(function(percentage), function(percentage),...)**

This parameter suggests to the implementation how to allocate physical processing resources to specific internal functions. The implementation can be influenced by the specifications in the parameter, but only as an approximation. If the PPL\_PROCESSORS parameter is omitted, the implementation attempts to optimize the processing resources based on the NPU model, attributes of the specific PPL program, and certain other DeviceMap parameters, such as LINK. It is recommended that one start out by not using the PPL\_PROCESSORS parameter.

One can use the PPL\_PROCESSORS parameter to suggest processing resource allocation for specific parts of the PPL virtual machine. If a function is not specified, its resource needs are deduced from other things. If a function is specified with a percentage of zero, this suggests the omission of the function.

For instance PPL\_PROCESSORS(ER(10%)) suggests that 10% of the processing resources be devoted to Ethernet frame receiving, and, by omission, that the implementation decide on the best use of the remaining 90%.

The functions can be expressed as

Notation	Description	Typical values
AE	Action engines – PPL event processing	> 50%
BE	Broad expression evaluation engines – certain preprocessing of PPL rule expressions	6-12%
CE	Connection engines – preprocessing of received packets and connection state lookup	6-12%
EE	Encryption engines	0-15%
PE	PATTERNS policy search engines	0-15%
TE	Timeout engines – needed if timeout is used with CONNECTIONS and ASSOCIATE policies	0-1%
I4	IPv4 forwarding	0-10%
I6	IPv6 forwarding	0-10%
ER	Ethernet receive	0-10%

ET	Ethernet transmit	0-10%
CR	CSIX receive	0-5%
CT	CSIX transmit	0-5%
PR	PCI receive	0-3%
PT	PCI transmit	0-3%
SR	POS receive	0-5%
ST	POS transmit	0-5%

As an example, for a typical PPL program on the IXP2800 where the PATTERNS and IPsec policies are not used, the only forwarding is IPv4, and the only I/O is Ethernet, if the PPL\_PROCESSORS parameter is not used, the implementation settles on a configuration approximating the following

```
PPL_PROCESSORS(ER(6%),ET(6%),I4(6%),BE(6%),CE(6%),TE(1%),AE(69%))
```

## Memory Usage

These parameters specify how memory shall be allocated. There can be at most one of each parameter specified.

**PACKET\_MEM(DRAM,ksize[,prepend])**

**SADB\_MEM(memory,ksize)**

**LOG\_MEM(DRAM,ksize[,bytes])**

**JOURNAL\_MEM(memory,ksize)**

**CONNECTIONS\_MEM(memory)**

**ASSOCIATIONS\_MEM(memory)**

**ARRAY\_MEM(memory)**

For the parameters, *memory* must have the value DRAM or SRAM, and *ksize* is the number of kilobytes to be allocated. For instance `PACKET_MEM(DRAM,16392)` specifies that 16 MB of space for packet buffers should be allocated in DRAM. For `PACKET_MEM` and `LOG_MEM`, the memory *must* specify DRAM.

`SADB_MEM` specifies the information for security associations, `LOG_MEM` for the log, and `JOURNAL_MEM` for the debug journal. The remaining three specify the location of connection tables, association tables, and arrays. The size is not included because that comes from the PPL program. Note that for arrays, this specifies the location of any and all arrays not explicitly mapped via the `ARRAY_MAP` parameter discussed later.

All of these parameters are optional. If not specified, the *memory* default is DRAM. The size default is 8192 for `PACKET_MEM` and 0 for the others.

The optional *prepend* value in `PACKET_MEM` is the number of bytes to leave unused at the front of a packet buffer. For instance, `PACKET_MEM(DRAM,16392,20)` will leave at least 20 bytes of space prior to the L2 and packet headers. This can be used in conjunction with the `INSERT` parameter in the `PACKET` policy to avoid copying packets when inserting space.

The optional *bytes* value in `LOG_MEM` is the number of bytes from the front of the packet to log for each packet logged. If omitted at least 40 bytes are logged.

Note that PPL has additional requirements for memory beyond these values, but these are fixed requirements that cannot be changed.<sup>1</sup>

---

<sup>1</sup> E.g., packet queues are always maintained in SRAM.

## Array Map

The ARRAY\_MAP parameter specifies optional mapping information about an array defined in an ARRAY statement elsewhere in the PPL program.

### ARRAY\_MAP(array,location[,memory])

array is the name of the array  
 location specifies the physical address of the first element of the array. It can be expressed as an *external\_name* or as a specific value. Must be a multiple of 16.  
 memory specifies the memory for this mapping. It can have the values DRAM, SRAM, or SCRATCH. When location is specified as *external\_name*, the memory is implied and thus this value must not be present.

```
ARRAY_MAP(IP_TABLE,0x0C000000,DRAM)
ARRAY_MAP(SERVLIST,ext_$$pdk servlist)
ARRAY_MAP(IKEkey,0x00010000,SRAM)
ARRAY_MAP(SRAM_MEMORY,0,SRAM)
```

establish the physical locations of four arrays. IP\_TABLE is based at a specific address in DRAM and SERVLIST is based at some location specified by an external name. IKEkey, which perhaps is a memory-mapped device, is located at a specific SRAM address. The last case, SRAM\_MEMORY, shows a way to create an array that maps if needed to all of a memory.

Array mappings may overlap. One can map multiple arrays to the same location.

If SCRATCH is specified as the memory, the array must be 32- or 128-bits in width.

## Physical Network Interfaces (*LINK Parameter*)

The LINK parameter describes a physical interface external to the network processor. Typically a program would specify multiple LINK parameters. The general form is

### **LINK(logical\_port\_number,direction,link\_type,link-type-dependent values)**

Logical_port_number	corresponds to the values used in PPL event statements and forward actions. LINK thus provides a way of connecting the program to external interfaces. There can be at most one description per direction (IN and OUT, INOUT meaning both IN and OUT) of a specific logical port number in the DeviceMap. <sup>2</sup>
Direction	specifies the possible flow directions of data through the link. It must be specified as one of IN, OUT, and INOUT. One of the purposes of direction is to determine the need for device receivers and transmitters. For instance, if a PPL program specifies that all Ethernet-type links are IN, no Ethernet transmitter will be configured.
Link_type	specifies the type of physical link and, usually, its protocol.

### **Ethernet Links**

LINK(lpn,dir,GE\_ON\_SPI,spi\_port,max\_frame\_size,smac,pm,sfc,rfc,ctrl\_type,ctrl\_port)

LINK(lpn,dir,GE\_INT,ctrl\_port,max\_frame\_size,smac,pm,sfc,rfc)

GE\_ON\_SPI describes a gigabit Ethernet port on the NPU SPI bus. GE\_INT describes an internal Ethernet MAC (IXP23xx family capability).

Spi_port	is the port on the SPI bus.
Max_frame_size	is the largest Ethernet frame size supported. The current implementation requires the sum of this plus the prepend value to be less than 2048. An extension to this implementation will increase this to 16383 to support jumbo frames.
Smac	is the source MAC address - the MAC address used as the source MAC address in transmitted frames and used as the default "listening" MAC address when receiving. If one does not want to alter the MAC address, use 0 as this value.
Prom	must be either 0 (disable) or 1 (enable) promiscuous mode.
Sfc	must be either 0 (disable) or 1 (enable) sending flow control.
Rfc	must be either 0 (disable) or 1 (enable) receiving flow control.
Ctrl_type	specifies the controller chip name. For instance "IXF1010" specifies the Intel IXP1010.
Ctrl_port	is the port of the controller chip.

<sup>2</sup> This applies as well to the external program binding parameters discussed later, which define logical port numbers as well; there may only be one definition of a specific logical port number in DeviceMap.

**CSIX Links**

`LINK(lpn,dir,CSIX,type,class)`

This describes a link over a CSIX switch fabric. There may be multiple CSIX\_FABRIC parameters specified; each needs a unique logical port number.

**Type** specifies the CSIX CFrame type created. The only supported value is 1, which denotes unicast. Type has meaning only when sending (forwarding).

**Class** specifies the class value (0-255) in the CFrame. Class has meaning only when sending (forwarding).

If one needs, for instance, eight classes of service, one would use eight CSIX LINK parameters covering eight logical port numbers.

**PCI Links**

This describes the use of the PCI bus as a packet-transfer mechanism using IP Fabrics' PCI-PTM protocol.

`LINK(lpn,direction,PCI,drop,ltb_desire,in_notice,out_notice,out_msg)`

**Drop** specification of whether the packet is to be dropped upon transmission. 0 denotes not and 1 denotes drop.

**Ltb\_desire** The number of transmit buffers the NPU likes to possess from the other agent.

**In\_notice** Message register to be polled for transmission (0-3).

**Out\_notice** The address of the message register of the other agent.

**Out\_message** The data value written in the other agent's message register.

The values lpb\_desire and beyond apply to the mechanism, not by link. Therefore, if multiple logical ports are needed, multiple PCI LINK parameters can be used and only the first one need specify the PCI-specific parameters (ltb\_desire, ...). If they are specified, only the values in the first PCI LINK are used.

For instance the LINK parameters

```
LINK(12,INOUT,PCI,0,16,0x07000050,1)
LINK(13,IN,PCI)
```

define logical ports 12 and 13 as being mapped to the PCI-PTM mechanism. Port 12 is in-out, meaning that received packets will go to event 12 and packets forwarded to port 12 will go to PCI-PTM. Also, packets received by PCI-PTM indicating port 13 will be forwarded on to wherever port 13 is mapped (e.g., to an event 13, or possibly directly to an Ethernet output port 13).

If PCI links are specified but none specify ltb\_desire and the remaining values, the default is 8,,0,0. If drop is not specified in a link, the default is not drop.

### POS Links

`LINK(lpn,dir,POS_ON_SPI,spi_port)`

This describes a packet-over-SONET port on the NPU SPI bus, using the PPP protocol.

`Spi_port` is the port on the SPI bus.

## External Program Binding

The PROG parameter describes one or more interfaces via which the PPL program directly interacts with external programs, and vice versa. By an external program we mean

- A program in the XScale processor in the same NPU
- A microcode program in the same NPU
- A program in a general-purpose processor connected via the PCI bus
- Another PPL program in an NPU connected via the PCI bus

The table below shows the possible interactions between PPL and something else.<sup>3</sup> The two basic mechanisms are the forwarding of a packet to a program or the asynchronous invocation of a program with the transmission of data parameters to the program.

Initiator \ Receptient	Forward from PPL	Apply PPL Program policy	Forward API from microcode, XScale, and remote program	Program invocation API from microcode, XScale, and remote program
PPL Event	X	X	X	X
XScale program	X	X		
Microengine ring	X	X		
Microengine thread		X		
Remote program	X	X		

The PROG parameter's general form is

### **PROG(logical\_port\_number/external\_name,prog\_type,prog-type-dependent values)**

Logical\_port\_number corresponds to a value used in PPL forward actions.  
 External\_name is an external name of a program. Depending on the prog\_type, either or only one of logical port number and external name can be used.  
 Prog\_type specifies the type of external program.

A packet can be forwarded to anything with a logical port number (unless its direction attribute is IN). PPL events have one or more logical port numbers, and the LINK parameter described elsewhere in this specification associates logical port numbers with physical interfaces. In addition, the following PROG parameters can associate logical port numbers to entities:

PROG(lpn,RING,ring)  
 PROG(lpn,CONTROL,external\_name)

<sup>3</sup> We have omitted interactions where PPL is not the initiator or recipient. However, for instance, the PCI-PTM mechanism allows a remote program to write to a ring or invoke an XScale program.

A program (PPL or otherwise) can asynchronously invoke another program and in most cases pass data parameters to that program. This is done using an external name rather than a logical port number. The following PROG parameters can describe external names for program invocation:

```

PROG(external_name,PPL,event)
PROG(external_name,RING,ring)
PROG(external_name,CONTROL)
PROG(external_name,REMOTE)
PROG(external_name,MTHREAD,me,thread,signal)

```

### Ring Interface

The ring interface causes a forwarded packet or an asynchronous program invocation to be placed on the specified ring in the NPU. The ring value specifies a number of a scratch or SRAM hardware ring. Values 0-63, 64-127, and 128-191 are, respectively, the rings of SRAM channels 0, 1, and 2. Values 192-207 are scratch rings 0-15.

The DATA values 32- and/or 128-bit values) from the PROGRAM policy are placed on the ring.

The PROGRAM policy returns a value in Rr0. Rr0 is set to FuF if the operation was unsuccessful (e.g., insufficient space on the ring). For SRAM rings, a non-FuF RR0 value defines the number of words on the ring just prior to this operation.

If invoked via FORWARD, two words are placed on the ring – the handle of the forwarded packet and a word of data from the FORWARD.

### Control Interface

This is the means of connecting the PPL program to a program in the XScale processor. If listed with two values, the first is a logical\_port\_number to which a packet can be forwarded, and the second is an external program name. If listed with one value, it is an external program name of an XScale program that can be invoked via the PROGRAM policy.

### Microthread Interface

This interface describes a signal within the NPU that is asserted via the PROGRAM policy. No parameters are passed. The values *me*, *thread*, and *signal* specify a signal (0-15) on a specific thread (0-7) on a specific microengine (0-15). The microengine (me) should not be a number within the range defined in AVAILABLE\_PROCESSORS.

### PPL Interface

This is the means of connecting the PPL program to another part of the PPL program that applies a PROGRAM policy. If the external name from the policy matches the name here, the specified PPL event is invoked as a packetless event. The value *event* must match the first or only event number in a corresponding EVENT statement in the PPL program.

A packetless event invoked by a PPL PROGRAM policy is passed 0-8 parameters in its event registers. If a message string was specified, Re0 and Re1 describe the message string and Re2 onward contain the data values specified in the policy. Re0 is the length of the string and

Re1 is a packet handle of a newly created packet that contains the string. L2FIELD, PFIELD, and CONTENT all “point” to the beginning of the string. The packet type (PS\_TYPE) is data. The PPL event receiving this “pseudo packet” is responsible for explicitly dropping it when done.

When no message string is specified in the invoking PROGRAM policy, 0-6 data values are placed in the first six Re registers.

### **Remote Interface**

The remote interface specifies an external name of a program that is presumed to be in a processor connected via the PCI bus and PCI-PTM protocol.

## ***Packet Control***

The PACKET\_CONTROL parameter specifies several overall attributes of how packets are handled. The form is

### **PACKET\_CONTROL(flow\_ordering,arp)**

Flow\_ordering Must be 0 or 1. If 1, the implementation will order packet processing such that if packet  $x_1$  arrives before packet  $x_2$  and both are part of the same “flow,” packets  $x_1$  and  $x_2$  are processed sequentially and never concurrently.

Arp Must be 0 or 1. If 1, ARPs received from Ethernet ports are handled by XScale software (provided by IP Fabrics or the equivalent). If 0, ARPs are passed through to the PPL program.

If PACKET\_CONTROL is absent, the default is 1 and 1.

### **Flow Control**

For the purposes of the above, a “flow” is defined as any of the following

1. The two packets are part of the same connection in a connections table.
2. If the IP\_PROT value of the two packets is the same but other than TCP or UDP, and the two packets have the same source and destination IP addresses
3. If the IP\_PROT value of the two packets is the same and one of TCP or UDP, and the two packets have the same source and destination IP addresses and same L4 source and destination port numbers.

Note that even if flow ordering is turned off (value 0), the virtual machine still ensures the following:

1. If two packets are part of the same connection in a connections table, the event that processes them does so serially (i.e., they are never processed concurrently).
2. Events marked as SERIAL are always serial.

## ***Optional Rejection of Malformed Packets***

The PPL language defines implicit expressions evaluated when referring to classes of named packet fields to avoid writing programs that misbehave when a packet contains inconsistencies. In addition, DeviceMap contains an optional parameter that will cause certain received packets to be immediately rejected instead of delivered to a PPL event.

### **REJECT\_MALFORMED(*list of constants*)**

The *list of constants* specifies the types of packets to be dropped. For instance, REJECT\_MALFORMED(1,3,4) specifies that checks 1, 3, and 4 as defined below should be performed on every IP packet received.

- 1 IP version is not IP (IPv4) or IPv6.
- 2 If an IPv4 packet, the header length (IHL) is less than 5, or the total length is less than  $4 \times \text{IHL}$ , or the header checksum is invalid.
- 3 If an IP packet, IP\_PROT is equal to or greater than 136.
- 4 The IP packet size is inconsistent with the entity (e.g., buffer, layer-2 frame) holding it (e.g., IP packet would extend beyond end of frame).
- 5 Improper source address. If IPv4, source address is 255.255.255.255 or 0.0.0.0 or 224.0.0.0/4 or 240.0.0.0/4 or 127.0.0.0/8. If IPv6, source address is  $::/128$  or  $::1/128$  or  $FF00::/8$ .
- 6 Improper destination address. If IPv4, destination address is 0.0.0.0 or 127.0.0.0/8. If IPv6, destination address is  $::/128$  or  $::1/128$ .
- 7 If an IP packet, source IP address = destination IP address
- 8 Malformed TCP packet. If IPv4 and TCP protocol and fragoffset is 0, then total length must be at least 20 plus  $\text{IHL} \times 4$ . If IPv6 and TCP protocol and either no fragment header present or fragment header present with fragment offset of 0, payload length must be at least 20 plus the size of any extension headers.
- 9 Malformed UDP packet. If IPv4 and UDP protocol and fragoffset is 0, then total length must be at least 8 plus  $\text{IHL} \times 4$ . If IPv6 and UDP protocol and either no fragment header present or fragment header present with fragment offset of 0, payload length must be at least 8 plus the size of any extension headers.
- 10 Malformed ICMP packet. If IPv4 and ICMP protocol and fragoffset is 0, then total length must be at least 8 plus  $\text{IHL} \times 4$ . If IPv6 and ICMPv6 protocol and either no fragment header present or fragment header present with fragment offset of 0, payload length must be at least 8 plus the size of any extension headers.
- 11 Bad fragment. The IP packet is a non-final fragment with a size < 400 bytes or a fragment with fragment offset > 0 but < 5.

## **Connection Tables and Association Tables Controls**

These parameters gives one control over certain timing and other aspects of internal implementation.

**CONNECTIONS\_CONTROL(collision\_tries,timeout\_rate,virtual\_network)**

**ASSOCIATIONS\_CONTROL(collision\_tries,timeout\_rate)**

collision_tries	describes how extensively the system should try to resolve collisions in connection and association tables. A lower number will result in faster search speed in the case of an unsuccessful search (item not found) but a higher rate of “no space available” errors when creating an entry. Suggested range is 2..4.
timeout_rate	is a value N denoting that approximately $2^N$ entries in the set of connection or association tables should be examined per second for timeout checking. If this value is zero, timeout checking that type of table is disabled.
virtual_network	describes the source of the virtual network number when locating a connections entry upon the arrival of a packet. Currently the only valid value is 0, which denotes: if the packet is in an 802.1Q tagged Ethernet frame, use the VLAN identifier as the value; otherwise use 0.

If the CONNECTIONS\_CONTROL parameter is omitted, the default used is CONNECTIONS\_CONTROL(2,10,0). If the ASSOCIATIONS\_CONTROL parameter is omitted, the default used is ASSOCIATIONS\_CONTROL(2,10).

## ***Debug Controls***

The DEBUG parameter invokes a number of debugging modes. Omitting the parameter disables the debugging modes.

### **DEBUG(event,mode,one or more values depending on mode)**

Debugging is specified by event.<sup>4</sup> The *event* number specifies the event in the PPL program to which this applies.

Mode RULE\_AND\_HEADER\_TRACE creates a record upon the completion of processing of the specified event. The record contains (1) the event number, (2) the number of the first true rule, (3) the current time, (4) the first 80 bytes of the current packet,<sup>5</sup> and (5) the connection state. The following are the options for RULE\_AND\_HEADER\_TRACE:

```
DEBUG(event,RULE_AND_HEADER_TRACE)
DEBUG(event,RULE_AND_HEADER_TRACE,ip_address,udp_port)
DEBUG(event,RULE_AND_HEADER_TRACE,ip_address,udp_port,lpn,id)
```

The first form places the information in memory in a debug journal. The second and third forms denote that a UDP/IP packet should be created containing the record, and that packet should be forwarded.<sup>6</sup> Udp\_port specifies the destination UDP port. The second form is analogous to saying FORWARD in a PPL program and the third form analogous to saying FORWARD(*lpn,id*).

```
DEBUG(event,RULE_AND_PACKET_TRACE)
```

Mode RULE\_AND\_PACKET\_TRACE is similar, except that the full packets are contained in the record rather than the first 80 bytes. The records are placed in the debug journal.<sup>7</sup>

---

<sup>4</sup> The ability to debug by event provides a lot of power. Obviously, one can debug a specific event that might be causing trouble. Or one could debug the exception event. Or if there were a particular type of packet that presents trouble, one could have the PPL program forward that packet to an event in which debug is enabled.

<sup>5</sup> 80 bytes gives you, for the typical TCP/IP packet, the layer 3 and layer 4 headers and 40 bytes (IPv4) or 20 bytes (IPv6) of the payload.

<sup>6</sup> The ability to have a debug record forwarded as a packet provides a lot of flexibility. One can obviously have them stream out of a network connection. Or they could be forwarded to a control-plane program or even another event in the PPL program. The option to keep the records in memory may seem superfluous, but it provides a higher-performance way of debugging when one is looking, for instance, just for history immediately before some occurrence.

<sup>7</sup> We have not provided a packetized record here because it would often exceed the MTU and thus require fragmenting.

## Implementation-Specific FORWARD Actions

If the PPL FORWARD action is specified with no parameters, the action is defined in the PPL specification as a layer-3 IP forwarding operation (i.e., forwarding via the destination IP address). FORWARD can also be expressed as FORWARD(*lpn*) and FORWARD(*lpn*,*id*), in which case the semantics are determined by the DeviceMap. Note that if *id* is omitted, its value is assumed 0.

If matches <i>lpn</i>	Consequence	Use of <i>id</i>	Packet dropped?
LINK( <i>lpn</i> , <i>dir</i> ,GE_ON_SPI,...) LINK( <i>lpn</i> , <i>dir</i> ,GE_INT,...)	Packet transmitted on this Ethernet port.	Next-hop number. But if FuF, denotes that no next-hop lookup done; packet has a full and complete Ethernet frame	Yes
LINK( <i>lpn</i> , <i>dir</i> ,CSIX, <i>type</i> , <i>class</i> )	One or more unicast CFrames created from packet and transmitted.	Number of destination fabric port (also called traffic-manager number or blade ID). Must be 0-4095.	Yes
LINK( <i>lpn</i> , <i>dir</i> ,POS_ON_SPI,...)	Packet transmitted on this POS port.	Next-hop number. But if FuF, denotes that no next-hop lookup done; packet has a full and complete PPP frame	Yes
LINK( <i>lpn</i> , <i>dir</i> ,PCI,...)	Copy of packet transmitted via PCI-PTM	32 bit value transferred with copy of packet	Optional
PROG( <i>lpn</i> ,RING, <i>ring</i> )	Packet is enqueued on ring	32 bit value is placed on ring as second word	No
PROG( <i>lpn</i> ,CONTROL,...)	Packet is passed to XScale program	None (ignored)	No
None of the above, but matches EVENT( <i>lpn</i> , <i>lpn</i> ,...)	Packet enqueued on incoming packet stream to this event	None (ignored)	No
None of the above	PPL exception (unresolvable <i>lpn</i> )		See note

In the last case, the packet is dropped if not the current packet.

## Implementation-Specific Exceptions

Exception number	Meaning
33	An error occurred while trying to transmit a packet created by the DEBUG parameter. E.g., the lpn specified in the DEBUG parameter is undefined or not a network interface; the id does not specify a valid next hop.