Network VPI Library

Version 0 Revision 0

May 30, 2019 (June 24, 2014)

Ando Ki, Ph.D.
andoki@gmail.com / adki@future-ds.com

Copyright notice

License notice

Abstract

This document addresses VPI (Verilog Programming Interface) library for network applications, which include Ethernet, IP, UDP, TCP and PTPv2.

Table of contents

1 Introduction

Network VPI (Verilog Programming Interface) Library is a collection of VPI tasks that handles network packets, which include Ethernet, IP, UDP, TCP and PTPv2.

2 Quick start

```
module top;
... ...
reg [ 7:0] pkt_eth[0:4095];
   reg [47:0] mac_src;
   reg [47:0] mac_dst;
   reg [15:0] type_len;
   reg [15:0] bnum_payload;
   reg [ 7:0] payload[0:4095];
   reg [15:0] bnum_pkt;
   integer add_crc;
   integer add_preamble;
   integer idx;
   initial begin
        for (idx=0; idx<4096; idx=idx+1) pkt_eth[idx] = 8'hFF;
        mac_src=48'h02_12_34_56_78_9A;
        mac_dst=48'h02_11_22_33_44_55;
        for (idx=0; idx<4096; idx=idx+1) payload[idx] = idx;
        type_len=0; // 0 menas use 'bnum_payload'
        bnum_payload = 10;
        type_len = bnum_payload;
        add_crc = 0;
        add_preamble = 0;
        bnum_pkt=0
        // this task fills Ethernet packet in the ‘pkt_eth'
        // that contains 'bnum_pkt' bytes.
        $pkt_ethernet(pkt_eth
                ,bnum_pkt
                ,mac_src
                ,mac_dst
                ,type_len
                ,bnum_payload
                ,payload
                ,add_crc
                ,add_preamble
                );
   end
... ...
endmodule
```

Following shows an example ‘Makefile’ to compile and simulate the above code, where ModelSim is used for HDL simulator.

```
all: vlib vlog vsim

vlib:
    if [ -f compile.log ]; then /bin/rm -f compile.log; fi
    if [ -d work ]; then /bin/rm -rf work; fi
    vlib work 2>&1 | tee -a compile.log

vlog:
    vlog -work work top.v 2>&1 | tee -a compile.log

vsim:
    vsim -pli …../network_vpi.dll -novopt -c -do "run -all; quit" -lib work work.top
```

3 Building library

3.1 Directory

directory			remarks
doc			document
vpi			VPI sources
	src		
vpi_llib			by product from ‘vpi’
	modelsim/10.3		
		linux_x86_64/libnetwork_vpi.so	Linux 64-bit (GCC)
		mingw_x86_64/network_vpi.dll	MinGW 64-bit (GCC)
		MS64/network_vpi.dll	Visual Studio 64-bit
		MS32/network_vpi.dll	Visual Studio 32-bit
vpi_test			Testing

3.2 Building library

Figure 1 shows a flow to prepare VPI library, where ‘VPI routines header file’ and ‘System’s VPI library’ are simulator dependent.

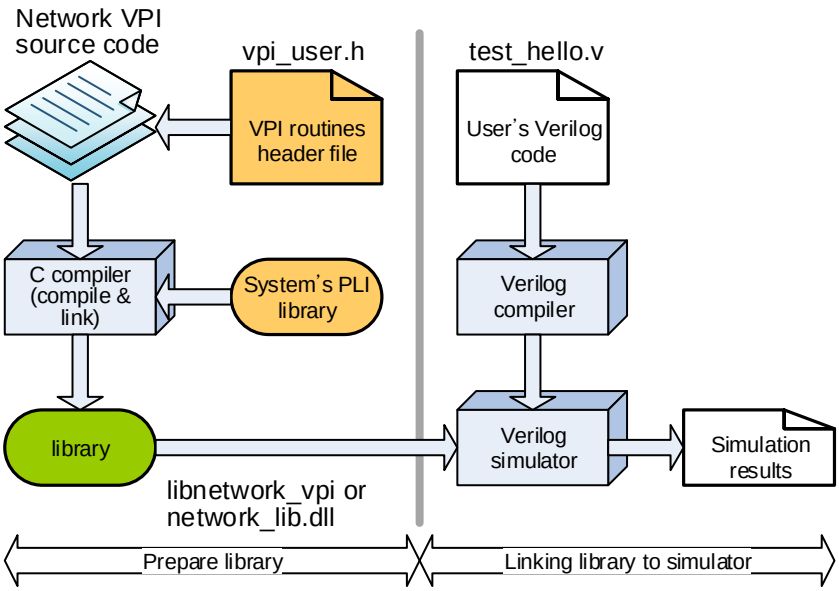


Figure 1: General flow to prepare VPI library

3.2.1 Linux or MinGW case

Simply go to 'vpi' directory and run 'make clean; make; make install'. Then, following dynamically linkable shared library will be ready in 'vpi_lib' directory.

```
$ cd vpi
$ make clean
$ make
$ make install
```

- For Linux 64-bit
  - vpi_lib/modelsim/10.3/linux_x86_64/libnetwork_vpi.so
- For MinGW 64-bit
  - vpi_lib/modelsim/10.3/linux_x86_64/network_vpi.dll

Where '10.3' indicates version of ModelSim.

3.2.2 Visual Studio case

Simply go to 'vpi' directory and do as described below. Then, following dynamically linkable shared library will be ready in 'vpi_lib' directory.

```
Open CMD window
WIN_RUN.bat
nmake –f NMAKEFILE clean
nmake –f NMAKEFILE
nmake –f
```

- For Visual Studio 64-bit

- ✧ vpi_lib/modelsim/10.3/MS64/network_vpi.dll
- For Visual Studio 32-bit
  - ✧ vpi_lib/modelsim/10.3/MS32/network_vpi.dll

Where ’10.3’ indicates version of ModelSim.

4 VPI tasks

4.1 Ethernet packet handling tasks

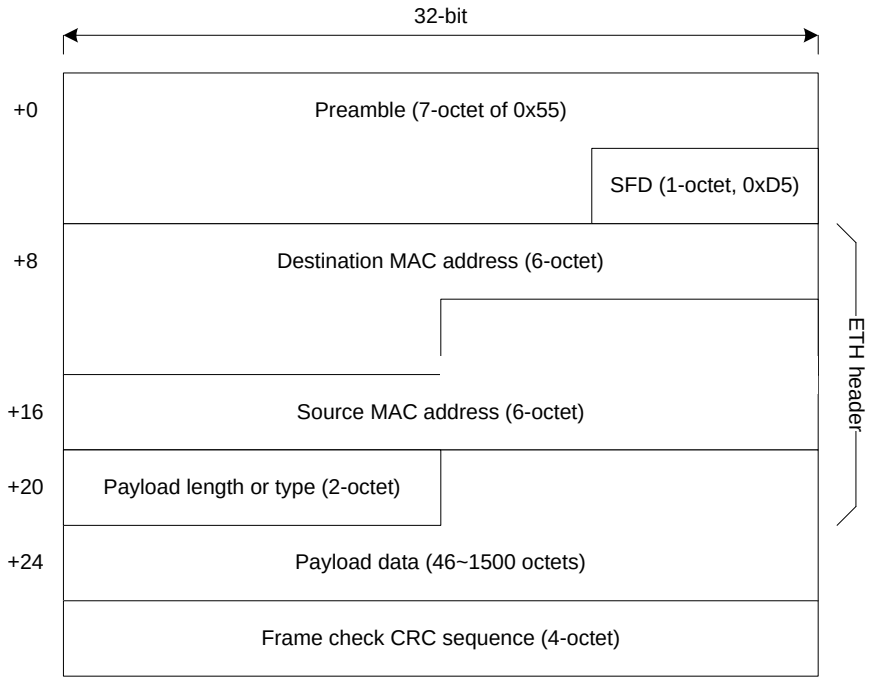
4.1.1 $pkt_ethernet()

```
$pkt_ethernet( pkt[7:0][0:4095]   // output
        , bnum_pkt[15:0] // output
        , mac_src [47:0] // input
        , mac_dst [47:0] // input
        , type_len[15:0] // input
        , bnum_payload[15:0] // input
        , payload[7:0][0:4095] // input
        , add_crc // input
        , add_preamble // input
        );
```

‘$pkt_ethernet()’ builds Ethernet packet in the ‘pkt’ argument, which is 8-bit 4096-entry register.

- ✧ pkt[7:0][0:4095]: register argument where Ethernet packet is filled; it is 8-bit width and can be up 4096 entries.
- ✧ bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the ‘pkt[...][...]’.
- ✧ mac_src[47:0]: 48-bit Ethernet physical address for source, where [47:40] is MSByte
- ✧ mac_dst[47:0]: 48-bit Ethernet physical address for destination
- ✧ type_len[15:0]: 16-bit type-length value of Ethernet packet, use ‘bnum_payload[..]’ when it is zero.
- ✧ bnum_payload[15:0]: num of bytes of in the ‘payload[...][...]’ argument.
- ✧ payload[7:0][0:4095]: register argument where Ethernet payload is given
- ✧ add_crc: add CRC at the end of the packet, if it is 1.
- ✧ add_preamble : add preamble[1] at the beginning of ‘pkt[...][...]’, if it is 1.

[1] 0x55/0x55/0x55/0x55/0x55/0x55/0x55/0xD5



4.1.2 $pkt_ethernet_parser()

```
$pkt_ethernet_parser( pkt[ 7:0][0:4095] // input
                    , bnum_pkt[15:0] // input
                    , crc // input
                    , preamble // input
                    );
```

'$pkt_ethernet_parser()' prints contents of 'pkt[...][...]' by interpreting it as Ethernet packet.

- ✧ pkt[7:0][0:4095]: register argument where Ethernet packet resides; it is 8-bit width and can be up 4096 entries.
- ✧ bnum_pkt[15:0]: the number of bytes in the 'pkt[...][...]'.
- ✧ crc: the packet contains CRC, if it is 1.
- ✧ preamble: the packet contains preamble, if it is 1.

4.2 IP packet handling tasks

4.2.1 $pkt_ip()

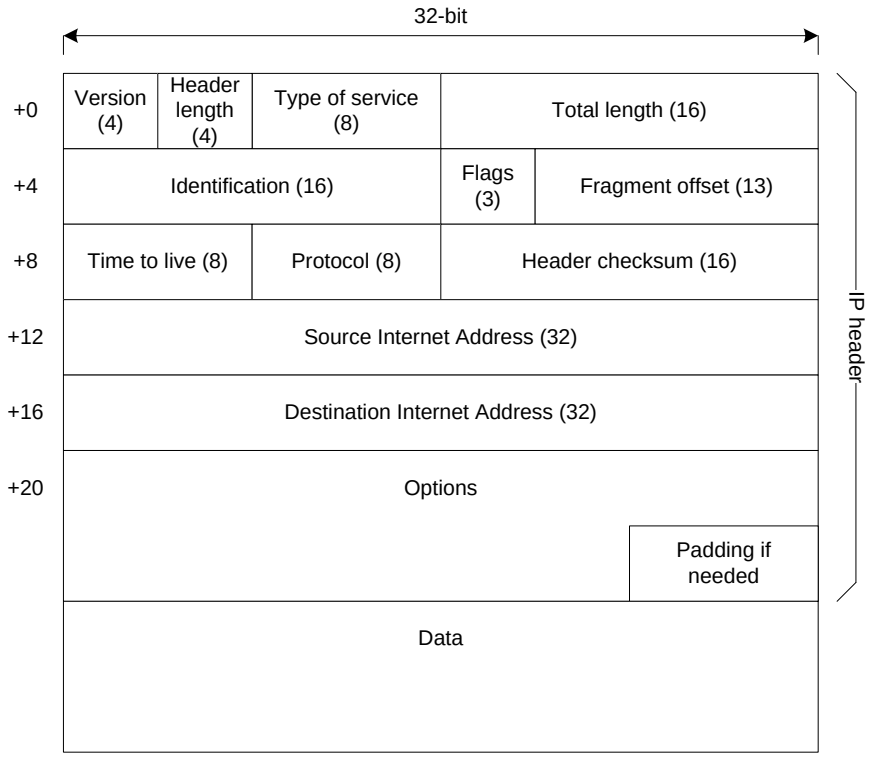
```
$pkt_ip( pkt[7:0][0:4095] // output
      , bnum_pkt[15:0] // output
      , ip_src[31:0] // input
      , ip_dst[31:0] // input
      , protocol[7:0] // input
```

```
    , ttl[7:0] // input
    , bnum_payload[15:0] // input
    , payload[7:0][0:4095] // input
    , tcp_check // input
    );
```

‘$pkt_ip()’ builds IPv4 packet in the ‘pkt’ argument, which is 8-bit 4096-entry register.

- pkt[7:0][0:4095]: register argument where IP packet is filled; it is 8-bit width and can be up 4096 entries.
- bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the ‘pkt[...][...]’.
- ip_src[31:0]: 31-bit IPv4 address for source, where [31:24] is MSByte
- ip_dst[31:0]: 31-bit IPv4 address for destination
- protocol[7:0]: 8-bit protocol value of the packet
- ttl[7:0]: 8-bit Time-To-Live value of the packet
- bnum_payload[15:0]: num of bytes of in the ‘payload[...][...]’ argument.
- payload[7:0][0:4095]: register argument where IP payload is given
- tcp_check: update TCP header check sum when it is 1 and ‘protocol’ is 0x06.

It fills other fields including ‘Version’, ‘Header Length’, ‘Header Checksum’, and so on.



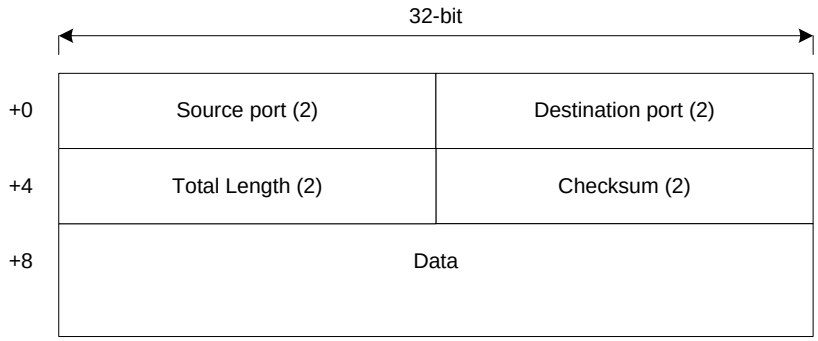
4.3 UDP packet handling tasks

4.3.1 $pkt_udp()

```
$pkt_udp( pkt[7:0][0:4095] // output
        , bnum_pkt[15:0] // output
        , port_src[15:0] // input
        , port_dst[15:0] // input
        , bnum_payload[15:0] // input
        , payload[7:0][0:4095] // input
        );
```

'$pkt_udp()' builds UDP packet in the 'pkt' argument, which is 8-bit 4096-entry register.

- ✧ pkt[7:0][0:4095]: register argument where UDP packet is filled; it is 8-bit width and can be up 4096 entries.
- ✧ bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the 'pkt[...][...]'.
- ✧ port_src[31:0]: 31-bit port for source, where [31:24] is MSByte
- ✧ port_dst[31:0]: 31-bit port for destination
- ✧ bnum_payload[15:0]: num of bytes of in the 'payload[...][...]' argument.
- ✧ payload[7:0][0:4095]: register argument where UDP payload is given



4.3.2 $pkt_udp_ip_ethernet()

```
$pkt_udp_ip_ethernet( pkt     [7:0][0:4095] // output
        , bnum_pkt[15:0] // output, num of bytes of the whole packet
        , port_src[15:0]
        , port_dst[15:0]
        , ip_src  [31:0]
        , ip_dst  [31:0]
        , ttl     [ 7:0]
        , mac_src [47:0]
        , mac_dst [47:0]
        , bnum_payload[15:0] // num of bytes of payload
        , payload [7:0][0:4095]
        , add_crc      //
        , add_preamble //
```

```
    );
```

‘$pkt_udp_ip_ethernet()’ builds UDP/IP/Ethernet packet in the ‘pkt’ argument, which is 8-bit 4096-entry register.

- pkt[7:0][0:4095]: register argument where UDP/IP/Ethernet packet is filled; it is 8-bit width and can be up 4096 entries.
- bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the ‘pkt[...][...]’.
- port_src[31:0]: 31-bit port for source, where [31:24] is MSByte
- port_dst[31:0]: 31-bit port for destination
- ip_src[31:0]: 31-bit IPv4 address for source, where [31:24] is MSByte
- ip_dst[31:0]: 31-bit IPv4 address for destination
- ttl[7:0]: 8-bit Time-To-Live value of the packet
- bnum_payload[15:0]: num of bytes of in the ‘payload[...][...]’ argument.
- payload[7:0][0:4095]: register argument where IP payload is given
- mac_src[47:0]: 48-bit Ethernet physical address for source, where [47:40] is MSByte
- mac_dst[47:0]: 48-bit Ethernet physical address for destination
- bnum_payload[15:0]: num of bytes of in the ‘payload[...][...]’ argument.
- payload[7:0][0:4095]: register argument where UDP payload is given

‘protocol[7:0]’ field of IP will be 0x11 (UDP), and ‘type_len[15:0]’ field of Ethernet will be 0x0800

4.4 TCP packet handling tasks

4.4.1 $pkt_tcp()

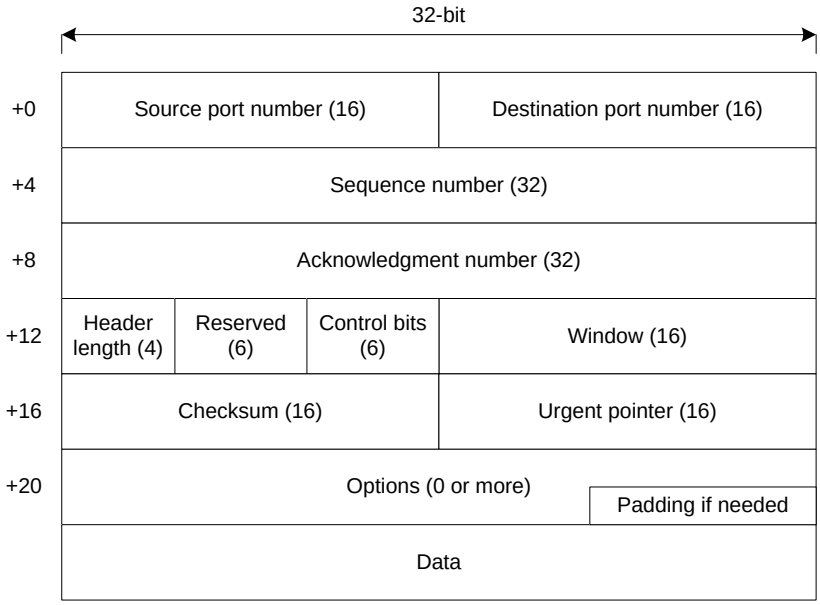
```
$pkt_tcp( pkt[7:0][0:4095] // output
        , bnum_pkt[15:0] // output
        , port_src[15:0] // input
        , port_dst[15:0] // input
        , seq_num[31:0] // input
        , ack_num[31:0] // input
        , bnum_payload[15:0] // input
        , payload[7:0][0:4095] // input
);
```

‘$pkt_tcp()’ builds TCP packet in the ‘pkt’ argument, which is 8-bit 4096-entry register.

- pkt[7:0][0:4095]: register argument where TCP packet is filled; it is 8-bit width and can be up 4096 entries.
- bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the ‘pkt[...][...]’.
- port_src[31:0]: 31-bit port for source, where [31:24] is MSByte
- port_dst[31:0]: 31-bit port for destination

- ✧ seq_num[31 :0] :
- ✧ ack_num[31 :0] :
- ✧ bnum_payload[15:0]: num of bytes of in the ‘payload[...][...]’ argument.
- ✧ payload[7:0][0:4095]: register argument where UDP payload is given

It fills ‘Checksum’ with 0, since pseudo header is not available; ‘Checksum’ can be updated when it is merged in IP packet.



4.5 PTPv2 packet handling tasks

4.5.1 $msg_ptpv2_set_context()

```
$msg_ptpv2_set_context( ptp_version // input
                      , ptp_domain // input
                      , one_step_clock // input
                      , unicast_port // input
                      , profile_spec1 // input
                      , profile_spec2 // input
                      );
```

‘$msg_ptpv2_set_context()’ fills PTPv2 context data structure, which is required to build PTPv2 message.

- ✧ ptp_version: it should be 2 for PTPv2
- ✧ ptp_domain:
- ✧ one_step_clock: 1 means one-step[2] clock, 0 means two-step clock
- ✧ unicast_port

[2] ‘Sync’ message carries time-stamp; It does not require ‘Follow_Up’ message aftward.

- ✧ profile_spec1
- ✧ profile_spec2

4.5.2 $msg_ptpv2_get_context()

```
$msg_ptpv2_get_context( ptp_version // output
                      , ptp_domain // output
                      , one_step_clock // output
                      , unicast_port // output
                      , profile_spec1 // output
                      , profile_spec2  // output
                      );
```

'$msg_ptpv2_get_context()' geturns PTPv2 context information, which is required to build PTPv2 message.

- ✧ ptp_version: it should be 2 for PTPv2
- ✧ ptp_domain:
- ✧ one_step_clock: 1 means one-step[3] clock, 0 means two-step clock
- ✧ unicast_port
- ✧ profile_spec1
- ✧ profile_spec2

4.5.3 $msg_ptpv2()

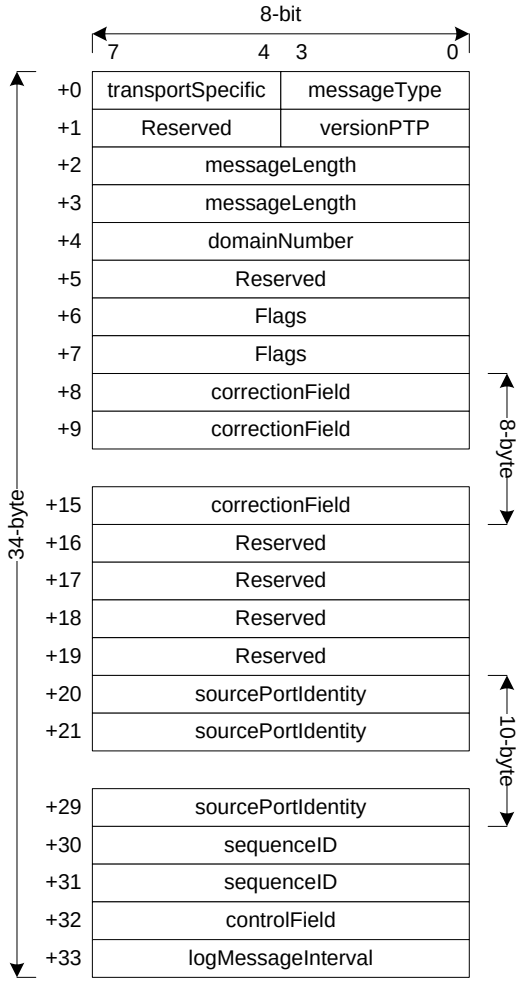
```
$$msg_ptpv2( pkt[7:0][0:4095] // output
           , bnum_pkt[15:0] // output
           , messageType[3:0] // input
           , secondsField[47:0] // input
           , nanosecondsField[31:0] // input
           , sequenceID[15:0] // input
           , sourcePortIdentity[79:0] // input
           , correctionField[63:0] // input
           , flagField[15:0] // input
           );
```

'$msg_ptpv2()' builds PTPv2 message in the 'pkt' argument, which is 8-bit 4096-entry register.

- ✧ pkt[7:0][0:4095]: register argument where PTPv2 message is filled; it is 8-bit width and can be up 4096 entries.
- ✧ bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the 'pkt[...][...]'.
- ✧ messageType[3:0]: 4-bit message type
- ✧ secondsField[47:0]: second value
- ✧ nanosecondsField[31:0]: nano-second value

[3] 'Sync' message carries time-stamp; It does not require 'Follow_Up' message aftward.

- ✧ sequenceID[15:0]: sequence identification
- ✧ sourcePortIdentiry[79:0]: 10-byte identity
  - 3-byte OUI
  - 5-byte UUID
  - 2-byte PTP node port number
- ✧ correctionField[63:0]: 8-byte
- ✧ flagField[15:0]: 2-byte
  - flagField[9]: twoStepFlag when 1
  - flagField[10]: unicastFlag when 1



4.5.4 $msg_ptpv2_etherent()

```
$msg_ptpv2_ethernet ( pkt[7:0][0:4095] // output
                    , bnum_pkt[15:0] // output
                    , mac_src[47 :0] // input
                    , messageType[3:0] // input
                    , flagField[15 :0] // input
                    , correctionField[63 :0] // input
                    , sourceClockID[63 :0] // input
```

```
                , sourcePortID[15:0] // input
                , sequenceID[15:0] // input
                , secondsField[47:0] // input
                , nanosecondsField[31:0] // input
                , sequenceID[15:0] // input
                , sourcePortIdentity[79:0] // input
                , reqClockID[63 :0] // input
                , reqPortID[15:0] // input
                , add_crc // input
                , add_preamble // input
                );
```

'$msg_ptpv2_ethernet()' builds PTPv2 message over Ethernet packet in the 'pkt' argument, which is 8-bit 4096-entry register.

- ✧ pkt[7:0][0:4095]: register argument where PTPv2 message over raw Ethernet packet is filled; it is 8-bit width and can be up 4096 entries.
- ✧ bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the 'pkt[...][...]'.
- ✧ mac_src[47:0]: 48-bit Ethernet physical address for source, where [47:40] is MSByte
- ✧ messageType[3:0]: 4-bit message type
- ✧ flagField[15:0]: 2-byte
  - flagField[9]: twoStepFlag when 1
  - flagField[10]: unicastFlag when 1
- ✧ correctionField[63:0]: 8-byte
- ✧ sourceClockID[63:0]: 8-byte
  - 3-byte OUI
  - 5-byte UUID
- ✧ sourcePortIdentiry[15:0]: 2-byte identity
  - 2-byte PTP node port number
- ✧ sequenceID[15:0]: sequence identification
- ✧ secondsField[47:0]: second value
- ✧ nanosecondsField[31:0]: nano-second value
- ✧ reqClockID[63:0]: 8-byte
- ✧ reqPortIdentiry[15:0]: 2-byte identity
- ✧ add_crc: add CRC at the end of the packet, if it is 1.
- ✧ add_preamble : add preamble (i.e., 0x55/0x55/0x55/0x55/0x55/0x55/0x55/0xD5) at the beginning of 'pkt[...][...]', if it is 1.

4.5.5 $msg_ptpv2_udp_ip_ethernet()

```
$msg_ptpv2_udp_ip_ethernet ( pkt[7:0][0:4095] // output
                , bnum_pkt[15:0] // output
                , mac_src[47 :0] // input
                , ip_adr[31:0] // input
                , messageType[3:0] // input
                , flagField[15 :0] // input
```

```
                , correctionField[63 :0] // input
                , sourceClockID[63 :0] // input
                , sourcePortID[15:0] // input
                , sequenceID[15:0] // input
                , secondsField[47:0] // input
                , nanosecondsField[31:0] // input
                , sequenceID[15:0] // input
                , sourcePortIdentity[79:0] // input
                , reqClockID[63 :0] // input
                , reqPortID[15:0] // input
                , add_crc // input
                , add_preamble // input
                );
```

'$msg_ptpv2_ethernet()' builds PTPv2 message over UDP/IP/Ethernet packet in the 'pkt' argument, which is 8-bit 4096-entry register.

- ✧ pkt[7:0][0:4095]: register argument where PTPv2 message over raw Ethernet packet is filled; it is 8-bit width and can be up 4096 entries.
- ✧ bnum_pkt[15:0]: num of bytes of the whole packet that has been built by this task, i.e., the number of bytes in the 'pkt[...][...]'.
- ✧ mac_src[47:0]: 48-bit Ethernet physical address for source, where [47:40] is MSByte
- ✧ ip_addr[31:0]: 32-bit IP address
- ✧ messageType[3:0]: 4-bit message type
- ✧ flagField[15:0]: 2-byte
  - flagField[9]: twoStepFlag when 1
  - flagField[10]: unicastFlag when 1
- ✧ correctionField[63:0]: 8-byte
- ✧ sourceClockID[63:0]: 8-byte
  - 3-byte OUI
  - 5-byte UUID
- ✧ sourcePortIdentiry[15:0]: 2-byte identity
  - 2-byte PTP node port number
- ✧ sequenceID[15:0]: sequence identification
- ✧ secondsField[47:0]: second value
- ✧ nanosecondsField[31:0]: nano-second value
- ✧ reqClockID[63:0]: 8-byte
- ✧ reqPortIdentiry[15:0]: 2-byte identity
- ✧ add_crc: add CRC at the end of the packet, if it is 1.
- ✧ add_preamble : add preamble (i.e., 0x55/0x55/0x55/0x55/0x55/0x55/0x55/0xD5) at the beginning of 'pkt[...][...]', if it is 1.

5 C API

5.1 CRC and checksum related API

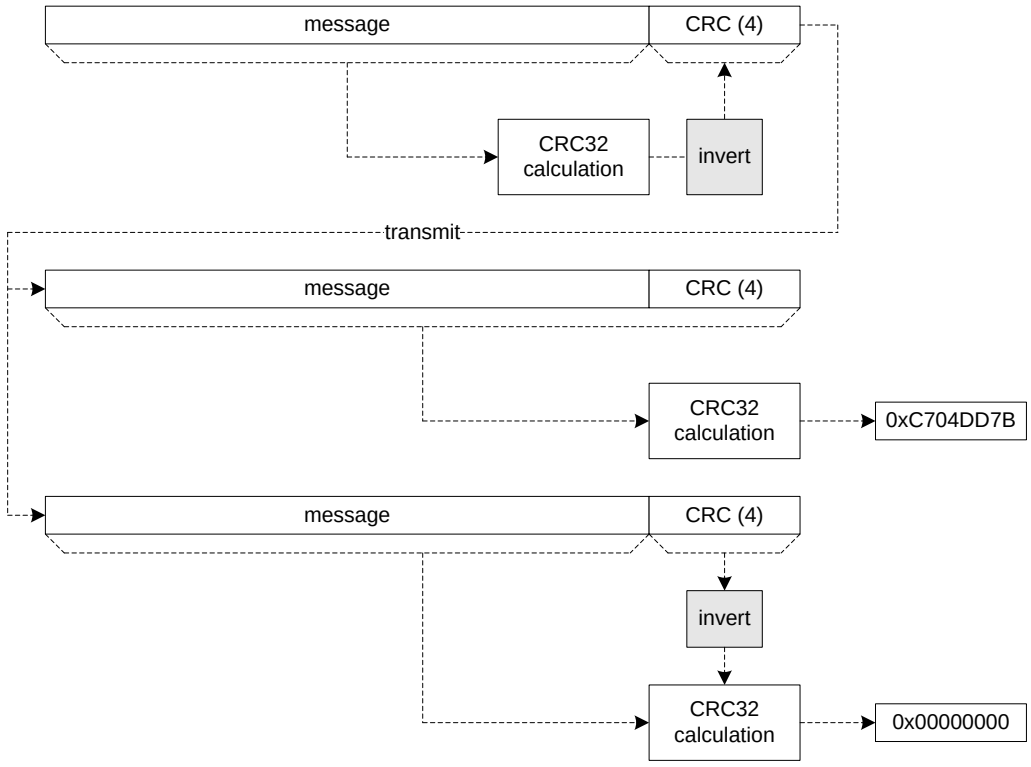
5.1.1 compute_eth_crc ()

```
#include "eth_ip_udp_tcp_pkt.h"
uint32_t compute_eth_crc(uint8_t *message // data buffer
                         , int len) ; // data length in bytes
```

'compute_eth_crc()' calculates Ethernet CRC and returns the result after inversion.

It uses following polynomial.
$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
(0x1_04C1_1DB7; 0xEDB8_8320/reverse[4])

Residue of G(x) is as folow.
$S(x)=x^{31}+x^{30}+x^{26}+x^{25}+x^{24}+x^{18}+x^{15}+x^{14}+x^{12}+x^{11}+x^{10}+x^8+x^6+x^5+x^4+x^3+x+1$
= 0xC704_DD7B (0xDEBB_20E3/reverse)



5.1.2 compute_checksum ()

```
#include "eth_ip_udp_tcp_pkt.h"
uint32_t compute_checksum(uint8_t *pkt, int bnum);
```

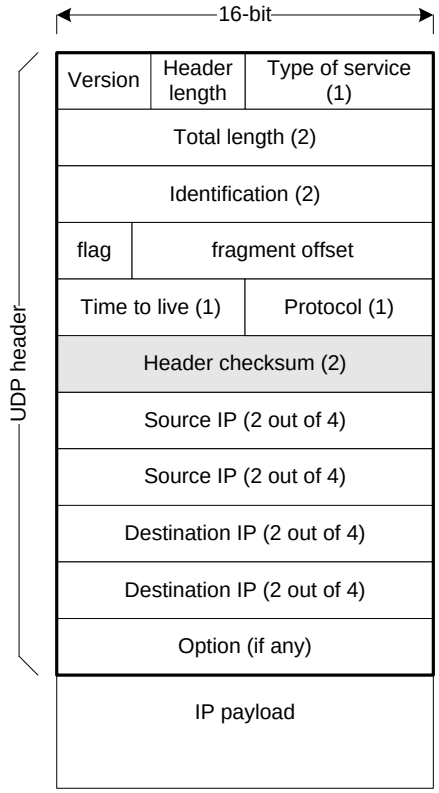
[4] When the first bit is corresponds X^{31}, 0x04C_1DB& will be used.

‘compute_checksum()’ calculates checksum and returns the result without inversion

5.1.3 compute_ip_checksum ()

```
#include "eth_ip_udp_tcp_pkt.h"
uint32_t compute_ip_checksum(ip_hdr_t *iphdr) ;
```

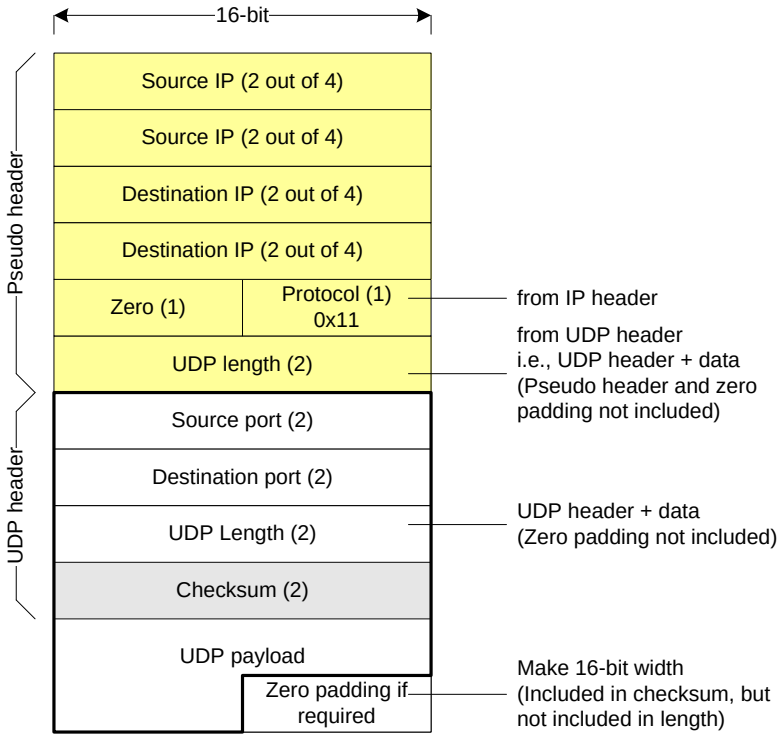
‘compute_ip_checksum()’ calculates checksum of IP header and returns the result after inversion.



5.1.4 compute_udp_checksum ()

```
#include "eth_ip_udp_tcp_pkt.h"
uint32_t compute_udp_checksum(pseudo_ip_hdr *ip_hdr
                                    , upd_hdr_t *udp_hdr) ;
```

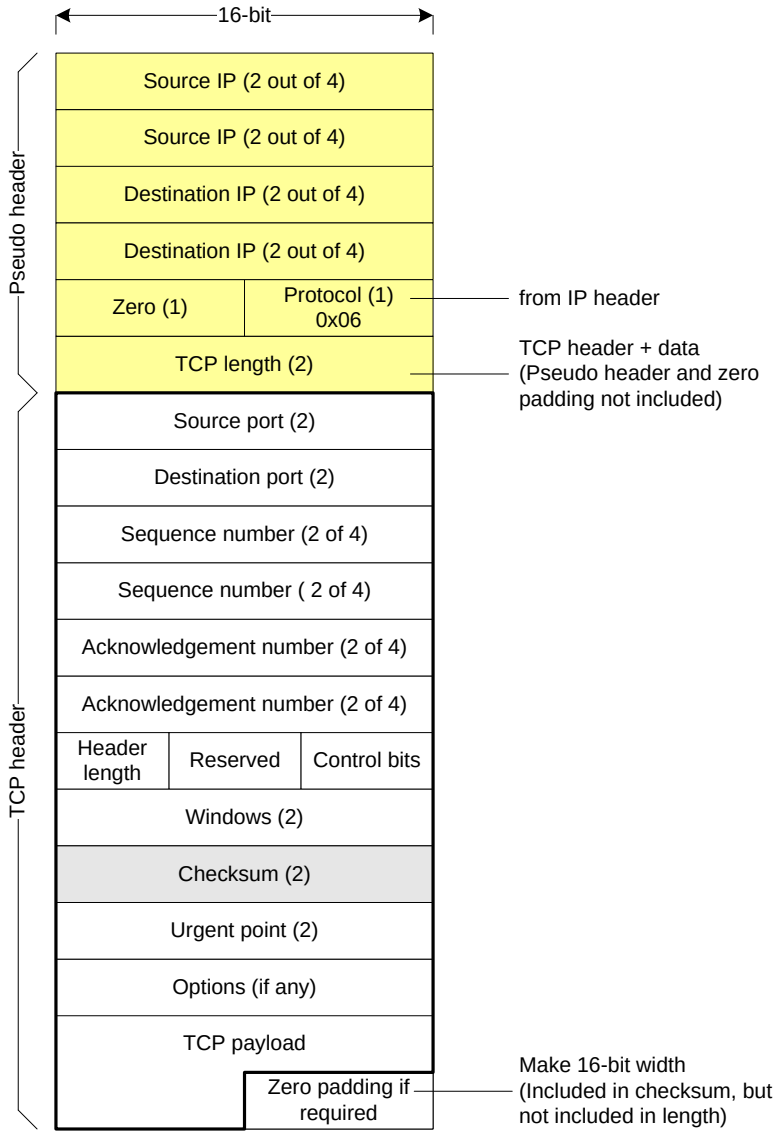
‘compute_udp_checksum()’ calculates checksum of UDP along with its pseudo IP header and returns the result after inversion.



5.1.5 compute_tcp_checksum ()

```
#include "eth_ip_udp_tcp_pkt.h"
uint32_t compute_tcp_checksum(pseudo_ip_hdr *ip_hdr
                                        , upd_hdr_t *tcp_hdr) ;
```

‘compute_tcp_checksum()’ calculates checksum of TCP along with its pseudo IP header and returns the result after inversion.

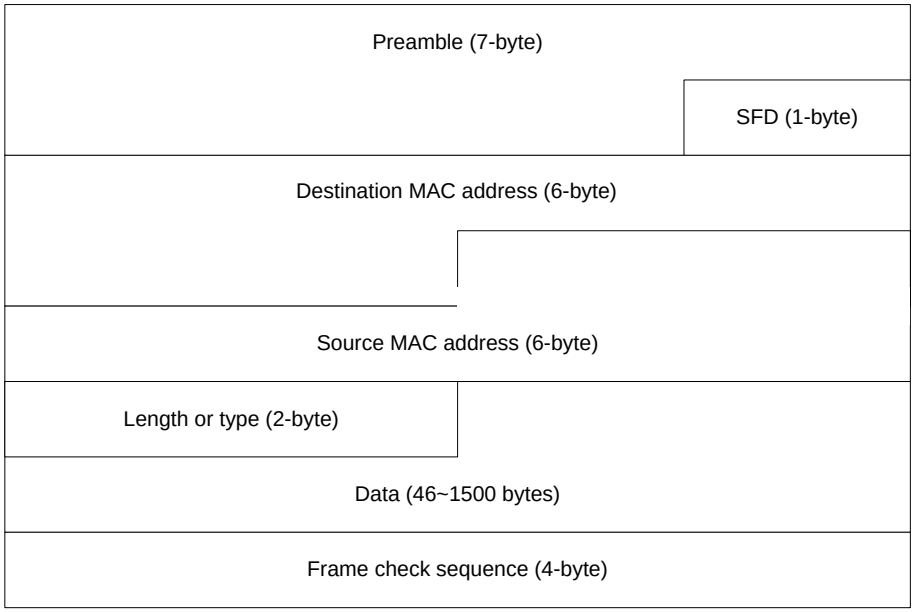


5.2 Packet header related API

5.2.1 populate_eth_hdr ()

```
#include "eth_ip_udp_pkt.h"
int populate_eth_hdr( eth_hdr_t *hdr // pointer to the buffer
                    , uint8_t    mac_src[6] // network order
                    , uint8_t    mac_dst[6] // network order
                    , uint16_t   type_leng);// host order, type an length value
```

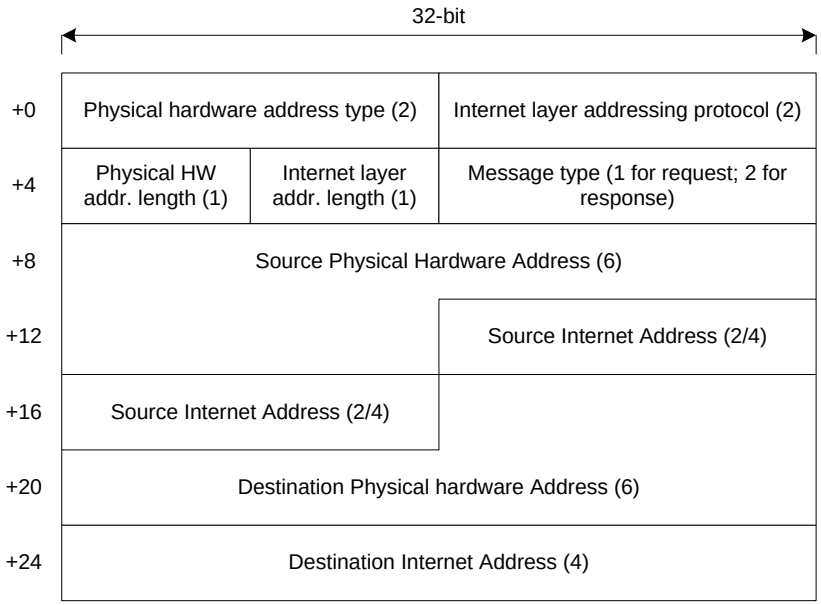
‘populate_eth_hdr()’ fills buffer pointed by ’hdr’ with Ethernet header (MAC destination to type-length) and returns the number of byes of the header.



5.2.2 populate_arp_hdr ()

```
#include "eth_ip_udp_pkt.h"
int populate_arp_hdr( arp_hdr_t *hdr
                , uint16_t   type       // message type 0: ARP req, 2: ARP reply
                , uint8_t    mac_src[6] // network order
                , uint8_t    mac_dst[6] // network order
                , uint32_t   ip_src     // host order
                , uint32_t   ip_dst);   // host order
```

‘populate_arp_hdr()’ fills buffer pointed by ’hdr’ with ARP header and returns the number of byes of the header.

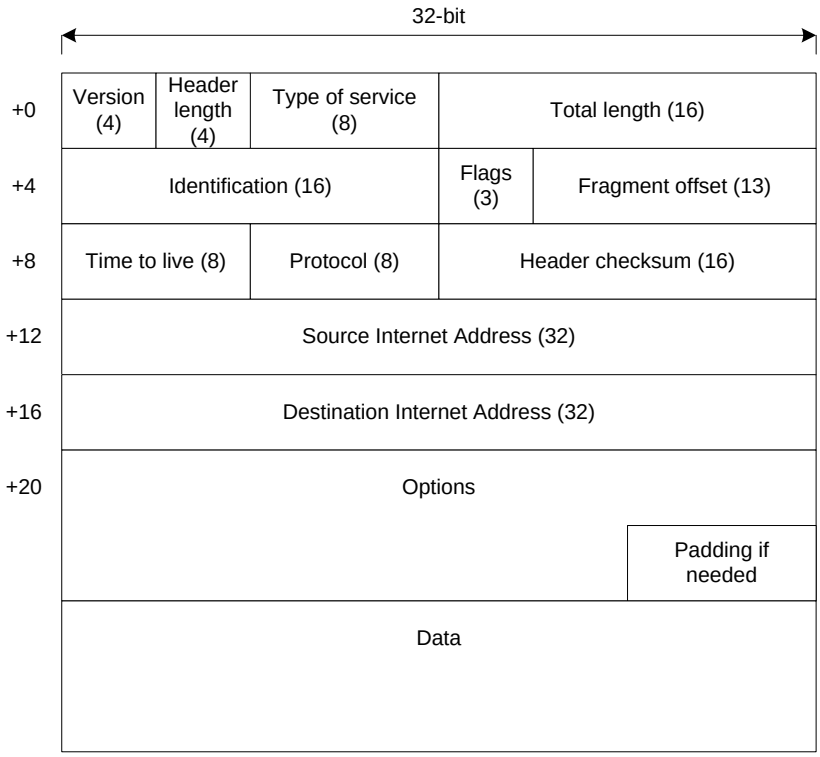


5.2.3 populate_ip_hdr()

```
#include "eth_ip_udp_pkt.h"
int populate_ip_hdr( ip_hdr_t *hdr
                   , uint32_t  ip_src    // host order
                   , uint32_t  ip_dst    // host order
                   , uint8_t   protocol  // specifies payload type
                   , uint16_t  payload_size); // pure payload size in bytes
```

‘populate_ip_hdr()’ fills buffer pointed by ’hdr’ with IP header and returns the number[5] of byes of the header.

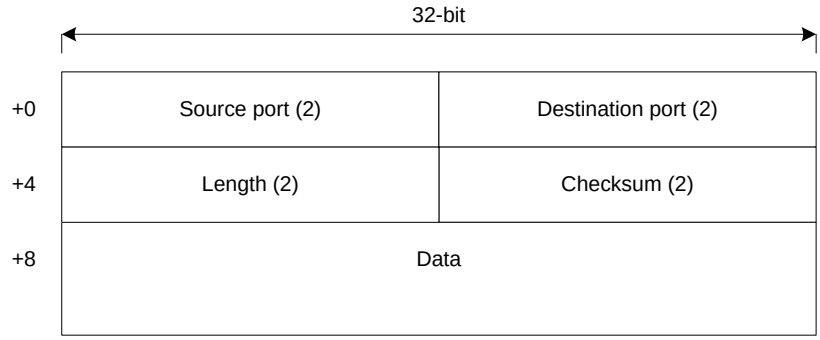
[5] It will be 20.



5.2.4 populate_udp_hdr()

```
#include "eth_ip_udp_pkt.h"
int populate_udp_hdr( udp_hdr_t *hdr
                    , uint16_t   port_src    // host order
                    , uint16_t   port_dst    // host order
                    , uint16_t   payload_size); // pure payload size in bytes
```

‘populate_udp_hdr()’ fills buffer pointed by ’hdr’ with UDP header and returns the number[6] of byes of the header.

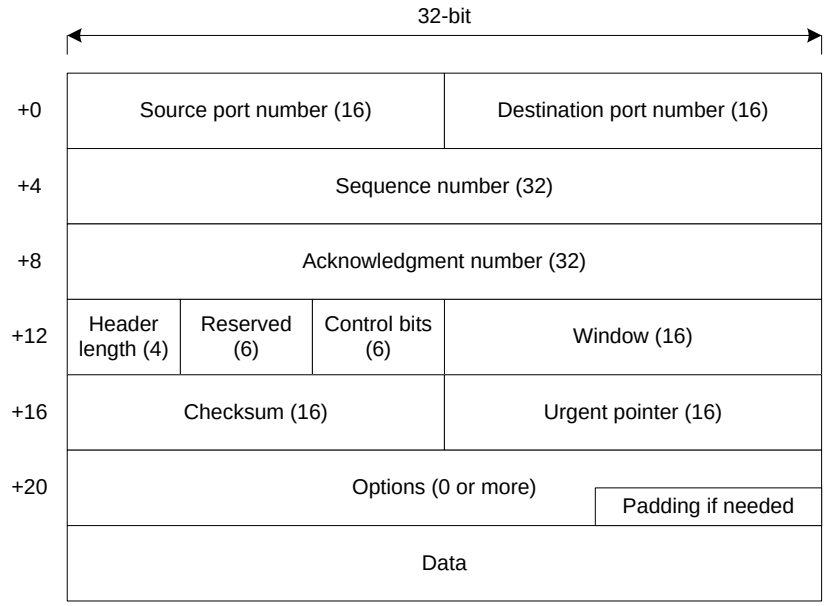


5.2.5 populate_tcp_hdr()

[6] It will be 8.

```
#include "eth_ip_udp_pkt.h"
int populate_tcp_hdr( tcp_hdr_t *hdr
                    , uint16_t   port_src // host order
                    , uint16_t   port_dst // host order
                    , uint32_t   num_seq  // host order
                    , uint32_t   num_ack  // host order
                    , uint16_t   payload_size); // pure payload size (not including header)
```

‘populate_tcp_hdr()’ fills buffer pointed by ’hdr’ with TCP header and returns the number[7] of byes of the header.



5.2.6 populate_ptpv2_msg_hdr()

```
#include "ptpv2_message.h"
int populate_ptpv2_msg_hdr( ptpv2_ctx_t     *ctx
                          , ptpv2_msg_hdr_t *msg_hdr
                          , uint8_t          type
                          , uint16_t         flag
                          , uint64_t         correction // 8-byte
                          , uint32_t         oui // 3-byte
                          , uint64_t         uuid // 5-byte
                          , uint16_t         ptp_port // 2-byte
                          , uint16_t         seq_id
                          , uint8_t          control);
```

‘populate_ptpv2_msg_hdr()’ fills buffer pointed by ’hdr’ with PTPv2 header and returns the number of byes of the header.

[7] It will be 20.

5.3 Packet related API

5.3.1 gen_eth_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_eth_packet( uint8_t  *packet
                  , uint8_t   mac_src[6] // network order
                  , uint8_t   mac_dst[6] // network order
                  , uint16_t  type_leng  // host order
                  , uint16_t  payload_len// payload length
                  , uint8_t  *payload  // payload if not 0
                  , int       add_crc // append 4-byte CRC when 1
                  , int       add_preamble);// add 8-byte preamble
```

'gen_eth_packet()' fills buffer pointed by 'packet' with Ethernet header and payload. 'type_leng' and 'payload_len' can be the same, but it will differ when 'type_leng' specifies packet type. 'payload_len' should specify the number of bytes of the payload even though 'payload' is NULL, i.e., 0.

When 'add_crc' is 1 and 'payload_len' is smaller than 46, it appends padding of 0x00 and then adds 4-byte CRC at the end of 'payload' array. For this case, the number returned is the number including paddings.

It returns the number of bytes this routine touched; it returns the length of header when 'payload' is NULL.

5.3.2 gen_arp_packet()

```
#include "eth_ip_udp_pkt.h"
#define gen_arp_packet populate_arp_hdr
```

As ARP packet does not have any payload, ARP header is ARP packet itself.

5.3.3 gen_ip_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_ip_packet( uint8_t  *packet
                 , uint32_t  ip_src     // host order
                 , uint32_t  ip_dst     // host order
                 , uint8_t   protocol   //
                 , uint16_t  payload_len// IP payload length
                 , uint8_t  *payload);  // payload if not 0
```

‘gen_ip_packet()’ fills buffer pointed by ’packet’ with IP header and payload. ‘payload_len’ should specify the number of bytes of the payload even though ‘payload’ is NULL, i.e., 0. It returns the number of bytes this routine touched; it returns the length of header when ‘payload’ is NULL.

5.3.4 gen_udp_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_udp_packet( uint8_t  *packet
                  , uint16_t  port_src   // host order
                  , uint16_t  port_dst   // host order
                  , uint16_t  payload_len// UDP payload length
                  , uint8_t  *payload); // payload if not 0
```

‘gen_udp_packet()’ fills buffer pointed by ’packet’ with UDP header and payload. ‘payload_len’ should specify the number of bytes of the payload even though ‘payload’ is NULL, i.e., 0. It returns the number of bytes this routine touched; it returns the length of header when ‘payload’ is NULL.

5.3.5 gen_tcp_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_tcp_packet( uint8_t  *packet
                  , uint16_t  port_src // host order order
                  , uint16_t  port_dst // host order order
                  , uint32_t  num_seq  // host order order
                  , uint32_t  num_ack  // host order order
                  , uint16_t  payload_len // IP payload length
                  , uint8_t  *payload); // pure payload
```

‘gen_tcp_packet()’ fills buffer pointed by ’packet’ with TCP header and payload. ‘payload_len’ should specify the number of bytes of the payload even though ‘payload’ is NULL, i.e., 0. It returns the number of bytes this routine touched; it returns the length of header when ‘payload’ is NULL. Note that it does not update checksum, instead it fills 0 for it.

5.3.6 gen_ptpv2_msg()

```
#include "ptpv2_message.h"
int gen_ptpv2_msg( ptpv2_ctx_t *ctx
                 , uint8_t      type // PTPv2 message type
                 , uint8_t     *msg // message buffer
                 , uint16_t     seq_id);
```

'gen_ptpv2_msg()' fills buffer pointed by 'msg' with PTPv2 header, body, and suffix.

5.4 Whole packet related API

5.4.1 gen_eth_arp_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_eth_arp_packet( uint8_t  *packet
                      , uint8_t   mac_src[6] // network order
                      , uint8_t   mac_dst[6] // network order
                      , uint16_t  type       // ARP type
                      , uint32_t  ip_src     // host order
                      , uint32_t  ip_dst     // host order
                      , int add_crc
                      , int add_preamble);
```

'gen_eth_arp_packet()' fills buffer pointed by 'packet' with Ethernet frame, which contains Ethernet header and a whole ARP packet.

5.4.2 gen_eth_ip_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_eth_ip_packet( uint8_t  *packet
                     , uint8_t   mac_src[6] // network order
                     , uint8_t   mac_dst[6] // network order
                     , uint32_t  ip_src     // host order
                     , uint32_t  ip_dst     // host order
                     , uint8_t   protocol   //
                     , uint16_t  payload_len// IP payload length
                     , uint8_t  *payload);  // payload if not 0
```

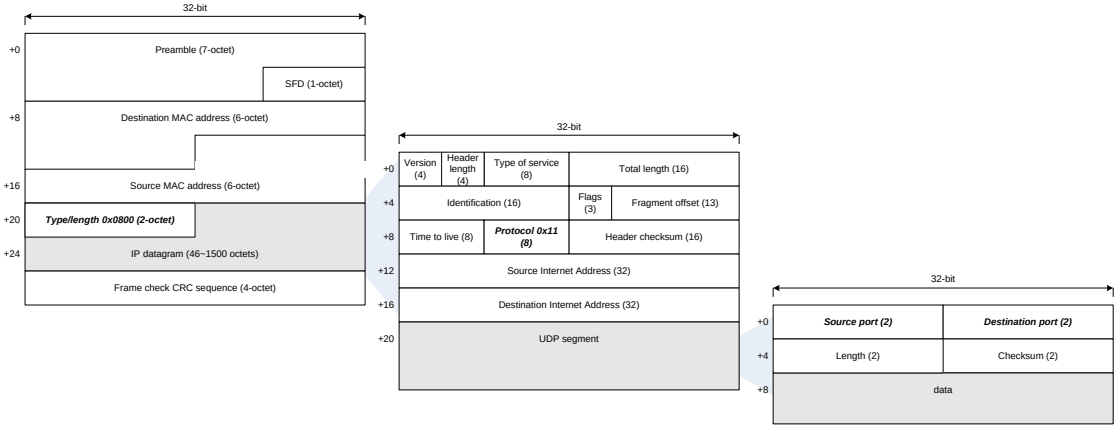
'gen_eth_ip_packet()' fills buffer pointed by 'packet' with Ethernet frame, which contains Ethernet header and a whole IP packet. 'payload_len' should specify the number of bytes of the IP pure payload even though 'payload' is NULL, i.e., 0. It returns the number of bytes this routine touched; it returns the length of header when 'payload' is NULL.

5.4.3 gen_eth_ip_udp_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_eth_ip_udp_packet( uint8_t  *packet
                         , uint8_t   mac_src[6] // network order
                         , uint8_t   mac_dst[6] // network order
                         , uint32_t  ip_src     // host order
```

```
                , uint32_t  ip_dst     // host order
                , uint16_t  port_src   // 0x0001; host order
                , uint16_t  port_dst   // 0x0002; host order
                , uint16_t  udp_payload_len // UDP payload length
                , uint8_t  *payload // udp payload (pure)
                , int add_crc
                , int add_preamble);
```

‘gen_eth_ip_udp_packet()’ fills buffer pointed by ’packet’ with Ethernet frame, which contains Ethernet header and a whole UDP over IP packet. ‘payload_len’ should specify the number of bytes of the IP pure payload even though ‘payload’ is NULL, i.e., 0. It returns the number of bytes this routine touched; it returns the length of header when ‘payload’ is NULL.

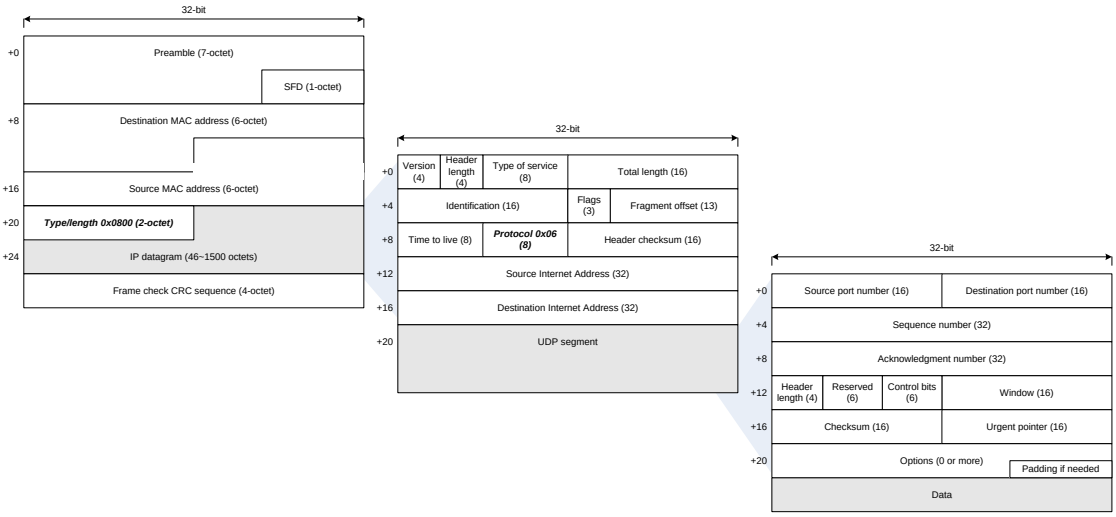


5.4.4 gen_eth_ip_tcp_packet()

```
#include "eth_ip_udp_pkt.h"
int gen_eth_ip_tcp_packet( uint8_t  *packet
                , uint8_t   mac_src[6] // network order
                , uint8_t   mac_dst[6] // network order
                , uint32_t  ip_src     // host order
                , uint32_t  ip_dst     // host order
                , uint16_t  port_src   // 0x0001; host order
                , uint16_t  port_dst   // 0x0001; host order
                , uint32_t  num_seq    // host order
                , uint32_t  num_ack    // host order
                , uint16_t  tcp_payload_len // TCP payload length
                , uint8_t  *payload // tcp payload (pure)
                , int add_crc
                , int add_preamble);
```

‘gen_eth_ip_tcp_packet()’ fills buffer pointed by ’packet’ with Ethernet frame, which contains Ethernet header and a whole TCP over IP packet. ‘payload_len’ should specify the number of bytes of the IP pure payload even though ‘payload’ is NULL, i.e., 0. It returns the number of bytes this routine touched; it

returns the length of header when ‘payload’ is NULL. It will update TCP header checksum using pseudo header information.



5.4.5 gen_ptpv2_ethernet()

```
#include "ptpv2_message.h"
int gen_ptpv2_msg_ethernet ( ptpv2_ctx_t *ctx
                        , uint8_t     *msg
                        , ptpv2_msg_hdr_t *hdr
                        , Timestamp_t     *time
                        , PortIdentity_t  *port
                        , uint8_t      mac_src[6]
                        , int          add_crc
                        , int          add_preamble);
```

‘gen_ptpv2_ethernet()’ fills buffer pointed by ’packet’ with Ethernet frame, which contains Ethernet header and a whole PTPv2 message. It uses 0x88F7 for Ethernet type-length value.

5.4.6 gen_ptpv2_upd_ip_ethernet()

```
#include "ptpv2_message.h"
int gen_ptpv2_msg_udp_ip_ethernet( ptpv2_ctx_t *ctx
                              , uint8_t     *msg
                              , ptpv2_msg_hdr_t *hdr
                              , Timestamp_t     *time
                              , PortIdentity_t  *port
                              , uint32_t     ip_src
                              , uint8_t      mac_src[6]
                              , int          add_crc
                              , int          add_preamble);
```

‘gen_ptpv2_udp_ip_ethrnet()’ fills buffer pointed by ’packet’ with Ethernet frame, which contains Ethernet header and a whole PTPv2 message over UDP over IP packet. It uses following UDP port and IP address for the destination.

- ✧ For event message
  - UDP port: 319
  - IP destination: 0xE0000181
  - MAC destination: 0x01_00_5E_00_01_81
- ✧ For general message
  - UDP port: 320
  - IP destination: 0xE000006B
  - MAC dstination: 0x01_00_5E_00_00_6B

5.5 Parsing API

5.5.1 parser_eth_packet ()

```
#include "eth_ip_udp_tcp_pkt.h"
int parser_eth_packet(uint8_t *pkt, int leng)
```

‘parser_eth_packet()’ interprets data in ‘pkt’ buffer as Ethernet frame.

5.5.2 parser_ip_packet ()

```
#include "eth_ip_udp_tcp_pkt.h"
int parser_ip_packet(uint8_t *pkt, int leng)
```

‘parser_ip_packet()’ interprets data in ‘pkt’ buffer as IP frame.

5.5.3 parser_udp_packet ()

```
#include "eth_ip_udp_tcp_pkt.h"
int parser_udp_packet(uint8_t *pkt, int leng)
```

‘parser_udp_packet()’ interprets data in ‘pkt’ buffer as UDP frame.

5.5.4 parser_tcp_packet ()

```
#include "eth_ip_udp_tcp_pkt.h"
int parser_tcp_packet(uint8_t *pkt, int leng)
```

‘parser_udp_packet()’ interprets data in ‘pkt’ buffer as TCPP frame.

6 References

[1] IEEE Standard Verilog Hardware Description Language, IEEE Std 1364-2001, IEEE Computer Society.
[2] The Verilog PLI Handbook, 2nd ED., S. Sutherland, KAP, 2002.
[3] ModelSim SE User's Manual, Mentor Graphics, 2003.
[4] VPI User Guide and Reference, Cadence Design Systems, 2002.

7 Index

The 2-Clause BSD License

Revision history

- ☐ 2019.05.30: Updated by Ando Ki.
- ☐ 2016.04.13: Started by Ando Ki (andoki@gmail.com or adki@future-ds.com).