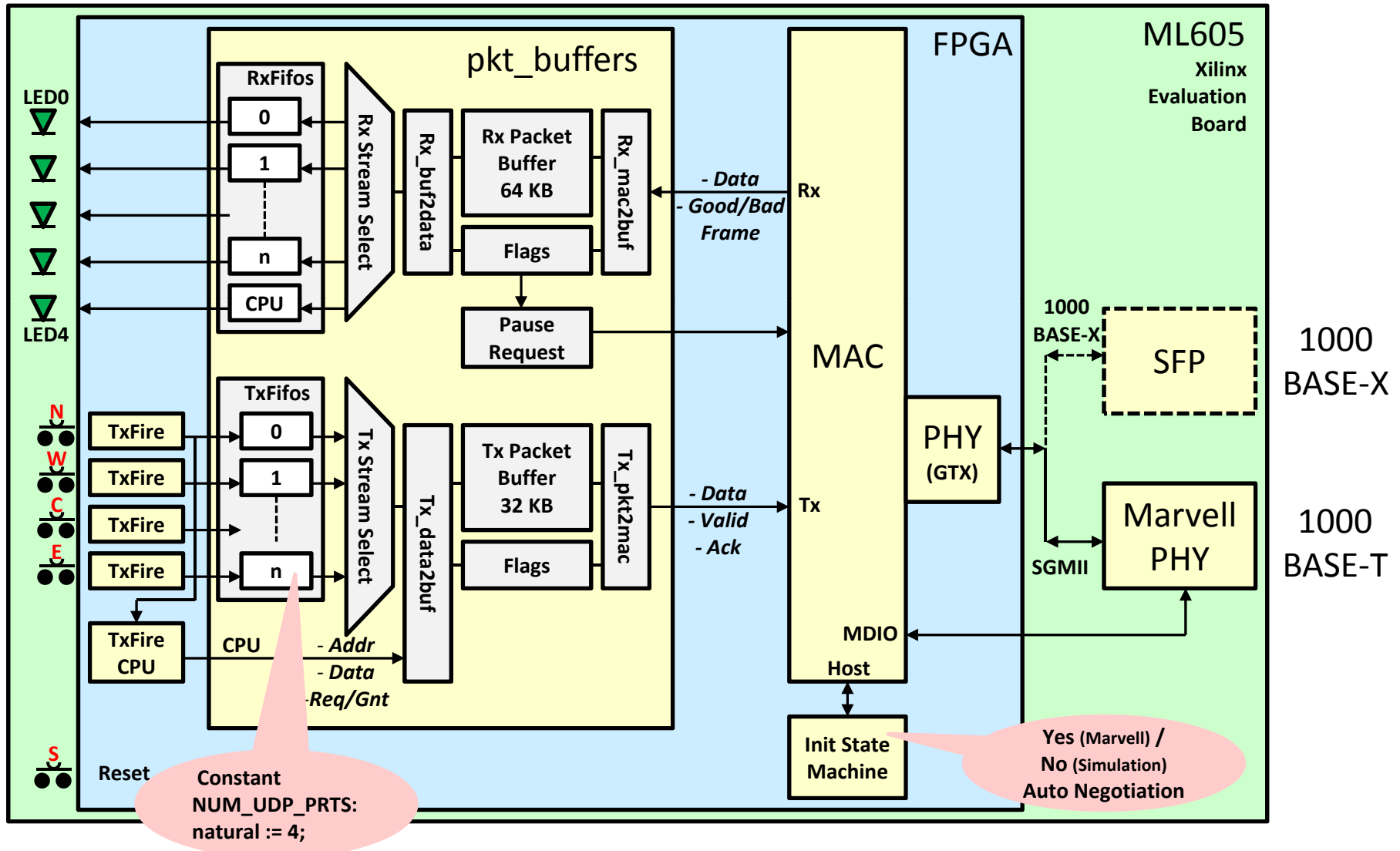


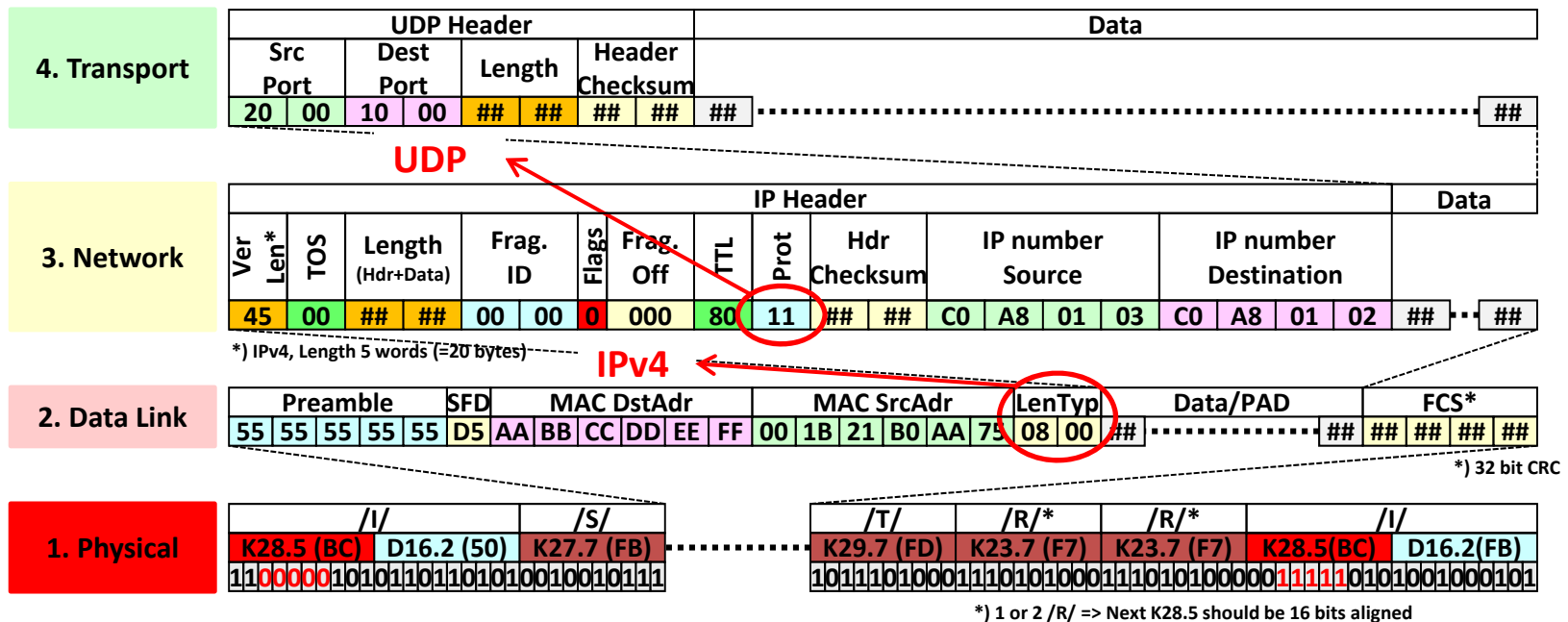
GB ethernet UDP interface in FPGA

Overview

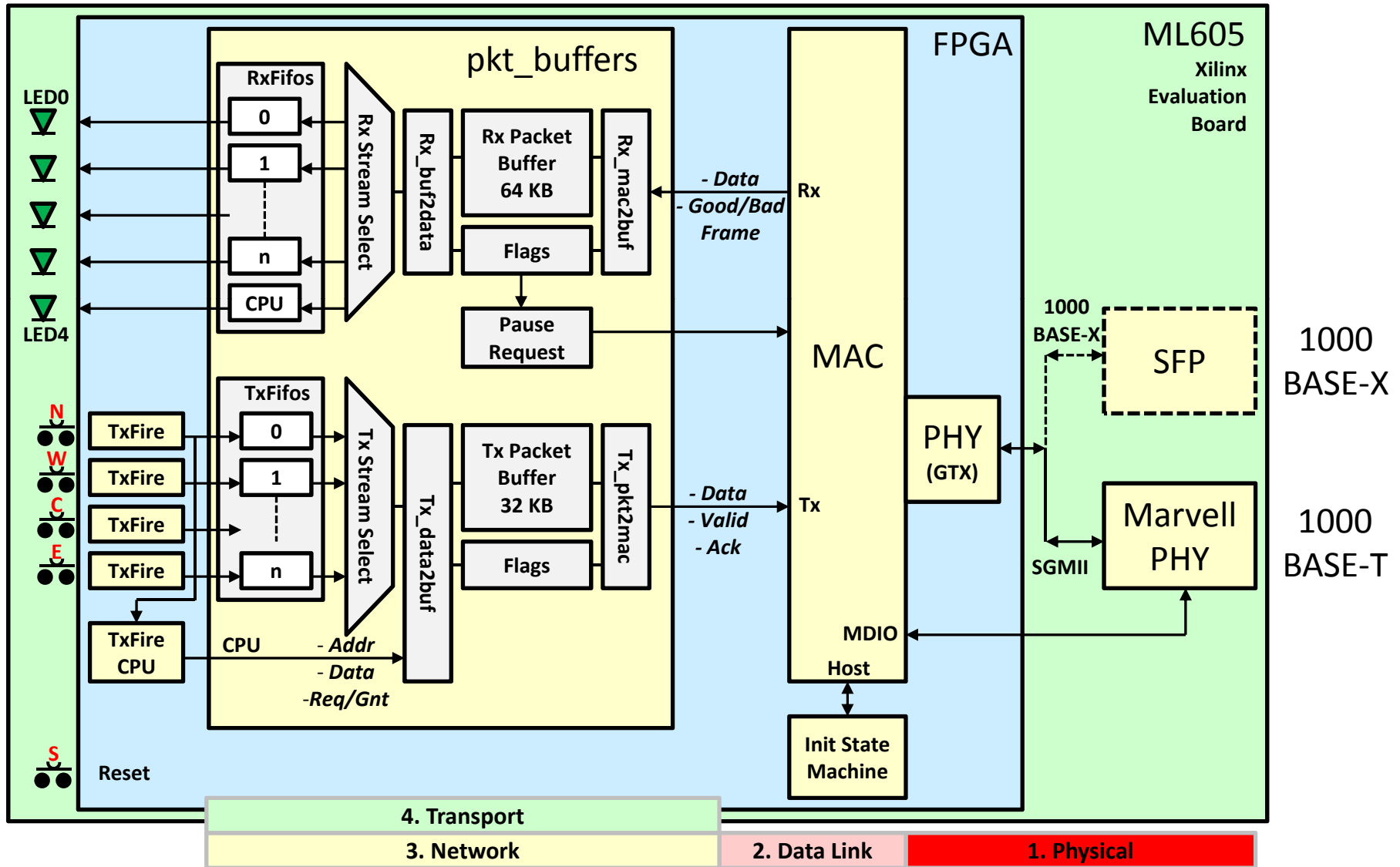


OSI Model

	Data Unit	Layer	Standard	
Host Layers	Data	7. Application		
		6. Presentation		
		5. Session		
	Segment	4. Transport	RFC-768	UDP
Media Layers	Packet	3. Network	RFC-791	IPv4
	Frame	2. Data Link	IEEE 802.3	MAC
	Bit	1. Physical		PHY



Overview



tx_data2buf (input from FIFO)

1. Initialize fixed header values



If no CPU Access Request AND there is input data Then



2. Read input FIFO -> Start Write from PAYLOAD (offset + 0x15)
Calculate UDP checksum (Pseudo Header)
3. Packet End := Last written offset
4. Write UDP header (offset + 0x11)
Source/Dest port pointed by "stream"
Length(bytes) = 8 (UDP Header) + number of data words(16) * 2
Calculated UDP Checksum
5. Write IP header (offset + 0x07)
Length(bytes) = 20 (IP Header) + number of data words(16) * 2
Calculate IP Checksum
Fragment-ID, Fragment-Offset, More-Fragments
6. Write MAC header (offset + 0x00)
Type = EtherType x0800
7. Next Packet Pointer := Packet End + 1

		Offset (@ 16 bits)		
		Dec	Hex	
PKT_STR	MAC_HDR	0	0x0	Destination Address (47:40)
		1	0x1	Destination Address (39:32)
		2	0x2	Destination Address (31:24)
		3	0x3	Destination Address (23:16)
		4	0x4	Destination Address (15:8)
		5	0x5	Destination Address (7:0)
IP_HDR		6	0x6	Source Address (47:40)
		7	0x7	Source Address (39:32)
		8	0x8	Source Address (31:24)
		9	0x9	Source Address (23:16)
		10	0xA	Source Address (15:8)
		11	0xB	Source Address (7:0)
UDP_HDR		12	0xC	Length / Type (15:8)
		13	0xD	Length / Type (7:0)
		14	0xE	Version
		15	0xF	Header Length
		16	0x10	Type Of Service (TOS)
		17	0x11	Length (7:0)
PAYLOAD		18	0x12	Length (15:8)
		19	0x13	ID(7:0)
		20	0x14	ID(15:8)
		21	0x15	Flags
		22	0x16	FragmentOffset (11:8)
		23	0x17	FragmentOffset (7:0)
		24	0x18	Time To Live (TTL)
		25	0x19	Protocol
		26	0x1A	Header CheckSum (7:0)
		27	0x1B	Header CheckSum (15:8)
		28	0x1C	IP Source (7:0)
		29	0x1D	IP Source (15:8)
		30	0x1E	IP Source (23:16)
		31	0x1F	IP Source (31:24)
		32	0x20	IP Destination (7:0)
		33	0x21	IP Destination (15:8)
		34	0x22	IP Destination (23:16)
		35	0x23	IP Destination (31:24)
		36	0x24	Port Source (7:0)
		37	0x25	Port Source (15:8)
		38	0x26	Port Destination (7:0)
		39	0x27	Port Destination (15:8)
		40	0x28	UDP Length (7:0)
		41	0x29	UDP Length (15:8)
		42	0x2A	Checksum (7:0)
		43	0x2B	Checksum (15:8)
		44	0x2C	CC
		45	0x2D	33
		46	0x2E	0
		47	0x2F	0
		48	0x30	11
		49	0x31	11
		50	0x32	22
		51	0x33	22
		52	0x34	EE
		53	0x35	11



tx_data2buf (input from CPU)

1. Initialize fixed header values



If there is a CPU Access Request Then



2. Start Write (offset + CPU_Addr)



Until CPU releases Access Request

Note: The last word of the packet must be accompanied with an End Of Packet (EOP ; Packet End := CPU_Addr) (note, since the CPU has random access to the packet buffer this doesn't necessarily mean that the last word of the packet is actually written last to the packet buffer). The CPU is responsible for generating the proper packet data format (i.e. MAC, IP etc. headers, checksums and data).

3. Next Packet Pointer := Packet End + 1

		Offset (@ 16 bits)		
		Dec	Hex	
PKT_STR	MAC_HDR	0	0x0	Destination Address (47:40)
		1	0x1	Destination Address (39:32)
		2	0x2	Destination Address (31:24)
		3	0x3	Destination Address (23:16)
		4	0x4	Destination Address (15:8)
		5	0x5	Destination Address (7:0)
		6	0x6	Source Address (47:40)
IP_HDR		7	0x7	Source Address (39:32)
		8	0x8	Source Address (31:24)
		9	0x9	Source Address (23:16)
		10	0xA	Source Address (15:8)
		11	0xB	Source Address (7:0)
		12	0xC	Length / Type (15:8)
		13	0xD	Length / Type (7:0)
UDP_HDR		14	0xE	Version
		15	0xF	Header Length
		16	0x10	Type Of Service (TOS)
		17	0x11	Length (7:0)
		18	0x12	Length (15:8)
		19	0x13	ID(7:0)
		20	0x14	ID(15:8)
PAYLOAD		21	0x15	Flags
		22	0x16	FragmentOffset (11:8)
		23	0x17	FragmentOffset (7:0)
		24	0x18	Time To Live (TTL)
		25	0x19	Protocol
		26	0x1A	Header CheckSum (7:0)
		27	0x1B	Header CheckSum (15:8)
		28	0x1C	IP Source (7:0)
		29	0x1D	IP Source (15:8)
		30	0x1E	IP Source (23:16)
		31	0x1F	IP Source (31:24)
		32	0x20	IP Destination (7:0)
		33	0x21	IP Destination (15:8)
		34	0x22	IP Destination (23:16)
		35	0x23	IP Destination (31:24)
		36	0x24	Port Source (7:0)
		37	0x25	Port Source (15:8)
		38	0x26	Port Destination (7:0)
		39	0x27	Port Destination (15:8)
		40	0x28	UDP Length (7:0)
		41	0x29	UDP Length (15:8)
		42	0x2A	Checksum (7:0)
		43	0x2B	Checksum (15:8)
		44	0x2C	CC
		45	0x2D	33
		46	0x2E	0
		47	0x2F	0
		48	0x30	11
		49	0x31	11
		50	0x32	22
		51	0x33	22
		52	0x34	EE
		53	0x35	11
		54	0x36	
		55	0x37	



Tx buffer memory flags

Example:
32KB

Tx Buffer of 32 KB can hold 3 complete Jumbo Frames
(9000 bytes + MAC_Hdr{6 + 6 + 2} = 9014 bytes)

Jumbo	Number of Bytes	% of ADRSIZE 14 (32678 Bytes)	% of ADRSIZE 15 (65536 Bytes)
1	9014	27,51%	13,75%
2	18028	55,02%	27,51%
3	27042	82,53%	41,26%
4	36056	110,03%	55,02%
5	45070		68,77%
6	54084		82,53%
7	63098		96,28%
8	72112		110,03%
9	81126		

Signal Full, as soon as there is no room left for a complete (Jumbo) packet



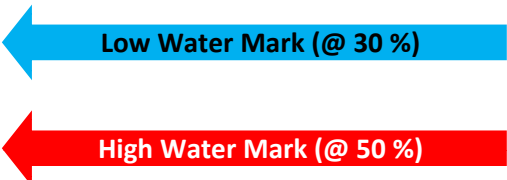
rx_mac2buf

1. When Rx buffer is *not* Full then there is at least space for one Jumbo Frame
(If Rx buffer is Full but MAC starts writing data then signal this with “Overflow”)
2. Assemble 8-bit MAC output bytes into 16-bit words and store them
3. As soon as the MAC signals
 “**Good Frame**” then increment the Packet Pointer
 “**Bad Frame**” then leave the Packet Pointer thus overwriting the bad frame with the next frame

Rx buffer memory flags

Rx Buffer of 64 KB can hold 7 complete Jumbo Frames (9000 bytes + MAC_Hdr{6 + 6 + 2} = 9014 bytes)

Jumbo	Number of Bytes	% of ADRSIZE 14 (32678 Bytes)	% of ADRSIZE 15 (65536 Bytes)
1	9014	27,51%	13,75%
2	18028	55,02%	27,51%
3	27042	82,53%	41,26%
4	36056	110,03%	55,02%
5	45070		68,77%
6	54084		82,53%
7	63098		96,28%
8	72112		110,03%
9	81126		



When “High Water Mark” is reached (i.e. buffer fills up after reception of 4 Jumbo Frames; Note that the Packet Pointer is only updated after the complete reception of the Packet), then a Pause Request (0xFFFF) is issued.

When “Low Water Mark” is reached (i.e. buffer empties and there is still 1 Jumbo Frame available), then a Pause Request (0x0000) is issued which means “Cancel the Pause Request”

Pause Request

- Defined in See IEEE802.3-2005 Annex 31B
- Operates on the MAC-Control level:
 - Uses globally assigned 48-bit multicast address 01-80-C2-00-00-01
- One bit time = 0,8 ns @ 1Gb-ethernet
- One Pause Quanta = 512 bit times (= $512 * 0,8$ ns = 409,6 ns)
- 0xFFFF Pause Quanta's = 26,8 ms.

Pause Request test

- Hold Rx buffer readout while sending Jumbo Frames from PC to ML605 Evaluation kit
- Implemented a “EmptyRxBuffer” push button

`./ipsar udp_tx_jumbo rcv.bin`
(Command sends 4 Jumbo frames)

ML605 responds with Pause Request 65535

Push “EmptyRxBuffer” button (SW_North)
ML605 sends Pause Request 0

Send 4 Jumbo frames

ML605 responds with Pause Request 65535

Receive 14 jumbo frames from M605 (pushed TxFire button (SW_Centre))

Wireshark packet capture showing a pause request test. The capture shows 25 packets. Packets 1-4 are UDP frames from 192.168.1.2 to 192.168.1.3. Packets 5-6 are Spanning-tree protocol frames. Packets 7-10 are UDP frames from 192.168.1.2 to 192.168.1.3. Packets 11-14 are Spanning-tree protocol frames. Packets 15-25 are UDP frames from 192.168.1.3 to 192.168.1.2. Red arrows point from the text labels to specific packets in the capture.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
2	0.000066	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
3	0.000121	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
4	0.000178	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
5	0.000464	aa:bb:cc:dd:ee:ff	Spanning-tree-(for-br:MAC	CTRMAC	PAUSE: pause_time: 65535 quanta
6	16.690149	aa:bb:cc:dd:ee:ff	Spanning-tree-(for-br:MAC	CTRMAC	PAUSE: pause_time: 0 quanta
7	18.869955	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
8	18.870018	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
9	18.870073	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
10	18.870127	192.168.1.2	192.168.1.3	UDP	Source port: fronet Destination port: 8227
11	18.870362	aa:bb:cc:dd:ee:ff	Spanning-tree-(for-br:MAC	CTRMAC	PAUSE: pause_time: 65535 quanta
12	22.500954	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
13	22.500971	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
14	22.500976	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
15	22.500979	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
16	22.500985	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
17	22.500989	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
18	22.501170	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
19	22.501175	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
20	22.501434	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
21	22.501440	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
22	22.501444	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
23	22.501447	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
24	22.501666	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet
25	22.501670	192.168.1.3	192.168.1.2	UDP	Source port: 8227 Destination port: fronet

Frame 1: 9014 bytes on wire (72112 bits), 9014 bytes captured (72112 bits)

Ethernet II, Src: IntelCor_b0:aa:75 (00:1b:21:b0:aa:75), Dst: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)

Destination: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)

Source: IntelCor_b0:aa:75 (00:1b:21:b0:aa:75)

Type: IP (0x0800)

Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 192.168.1.3 (192.168.1.3)

User Datagram Protocol, Src Port: fronet (4136), Dst Port: 8227

Data (8972 bytes)

```

0000 aa bb cc dd ee ff 00 1b 21 b0 aa 75 08 00
0010 23 28 00 00 40 00 11 94 6f c0 a8 01 02
0020 01 03 10 22 20 23 14 72 25 05 0f 00 02
0030 00 04 00 05 00 06 00 07 00 08 00 09 00 0a
0040 00 0c 00 0d 00 0e 00 0f 00 10 00 11 00 12
0050 00 14 00 15 00 16 00 17 00 18 00 19 00 1a
0060 00 1c 00 1d 00 1e 00 1f 00 20 00 21 00 22
0070 00 24 00 25 00 26 00 27 00 28 00 29 00 2a
0080 00 2c 00 2d 00 2e 00 2f 00 30 00 31 00 32
0090 00 34 00 35 00 36 00 37 00 38 00 39 00 3a
00a0 00 3c 00 3d 00 3e 00 3f 00 40 00 41 00 42
00b0 00 44 00 45 00 46 00 47 00 48 00 49 00 4a
    
```

Terminal

```

user@unstrut ~/IPSAR_V3 $ ./ipsar udp_tx_jumbo rcv.bin
^CTraceback (most recent call last):
  File "ipsar.py", line 375, in <module>
    iss.execute(filename, args)
  File "ipsar.py", line 352, in execute
    func(*cmdArgs)
  File "ipsar.py", line 136, in cmd_receive
    data = self._sock.recv(self._recv_buf)
KeyboardInterrupt
user@unstrut ~/IPSAR_V3 $ ./ipsar udp_tx_jumbo rcv.bin
user@unstrut ~/IPSAR_V3 $
    
```

rx_data2buf

1. If Rx buffer is *not* Empty then
2. Assumption is made that an IPv4 / UDP header is received
3. Check:
 - EtherType = IPv4 (=0x0800)
 - IP header checksum
 - IP Version = 4 and IP Header Length = 5
 - Protocol = UDP (0x11)
4. Check if the UDP destination port corresponds to one of the predefined (originating from either constants or registers) output streams
5. Fragment-ID, Fragment-Offset and More-Fragments flag are updated.
6. Only the data is routed to the corresponding output Stream FIFO
7. After routing, UDP checksum error is asserted when applicable.

All packets that fail the above checks are routed *integral* (i.e. MAC, IP, UDP header and Data) to the CPU Stream FIFO

		Offset (@ 16 bits)		
		Dec	Hex	
PKT_STR	MAC_HDR	0	0x0	MAC (Layer 2)
IP_HDR		1	0x1	IP Header
UDP_HDR		2	0x2	UDP Header (RFC 768)
PAYLOAD		3	0x3	Data
PKT_END		4	0x4	
		5	0x5	
		6	0x6	
		7	0x7	
		8	0x8	
		9	0x9	
		10	0xA	
		11	0xB	
		12	0xC	
		13	0xD	
		14	0xE	
		15	0xF	
		16	0x10	
		17	0x11	
		18	0x12	
		19	0x13	
		20	0x14	
		21	0x15	
		22	0x16	
		23	0x17	
		24	0x18	
		25	0x19	

Checksum calculation

- IP and UDP Checksums
 - Checksum is a sum, not a CRC!
 - Add all and finally add the overflow (i.e. bits 31:16) to the lower 16 bits (15:0)
 - Take the ones complement of the lower 16 bits => this is the checksum
- IP Checksum
 - Mandatory
 - Calculated over the IP header (Not including IP header checksum field which is added as 0x0000)
- UDP Checksum
 - Optional
 - Calculated using a UDP *Pseudo-Header* that contains:
 - IP Source address
 - IP Destination address
 - x"00" & Protocol
 - UDP Length
 - Further calculation includes UDP Header:
 - UDP Source Port
 - UDP Destination Port
 - UDP Length (Note that UDP Length is present twice (in the Pseudo header and in the UDP header))
 - UDP Checksum is the sum over the Pseudo Header, UDP header and all data words.

Resources

Entity “pkt_buffers”

implementing 4 Tx/Rx Streams:

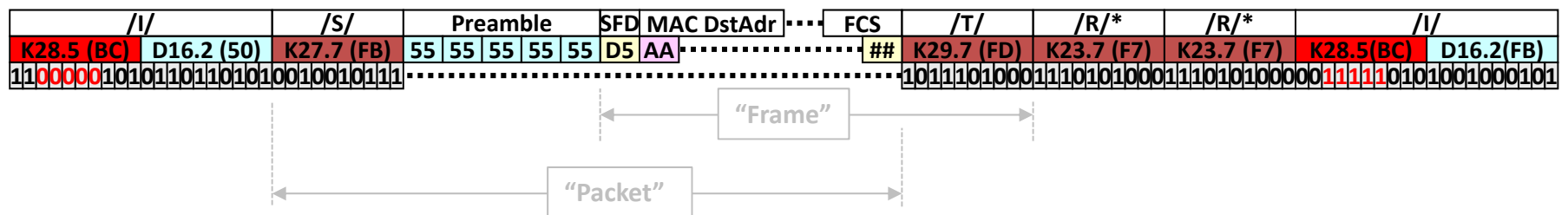
- Dffs or Latches 644
- Block RAMs
 - 32 KB + 64 KB, Packet buffers = 24 * RAMB36E1
 - 5 Rx FIFOs, 4 Tx FIFOs = 9 * RAMB18E1

Virtex-6 “xc6vlx240t” => 5% RAM, < 1% Slices

Frame and Packet Definitions

IEEE802.3-2008

Clause 30 (see also 46.2.5)



Frame and Packet Definitions

	Data Unit	Layer	Standard	
Host Layers	Data	7. Application		
		6. Presentation		
		5. Session		
	Segment	4. Transport	RFC-768	UDP
Media Layers	Packet	3. Network	RFC-791	IPv4
	Frame	2. Data Link	IEEE 802.3	MAC
	Bit	1. Physical		PHY

