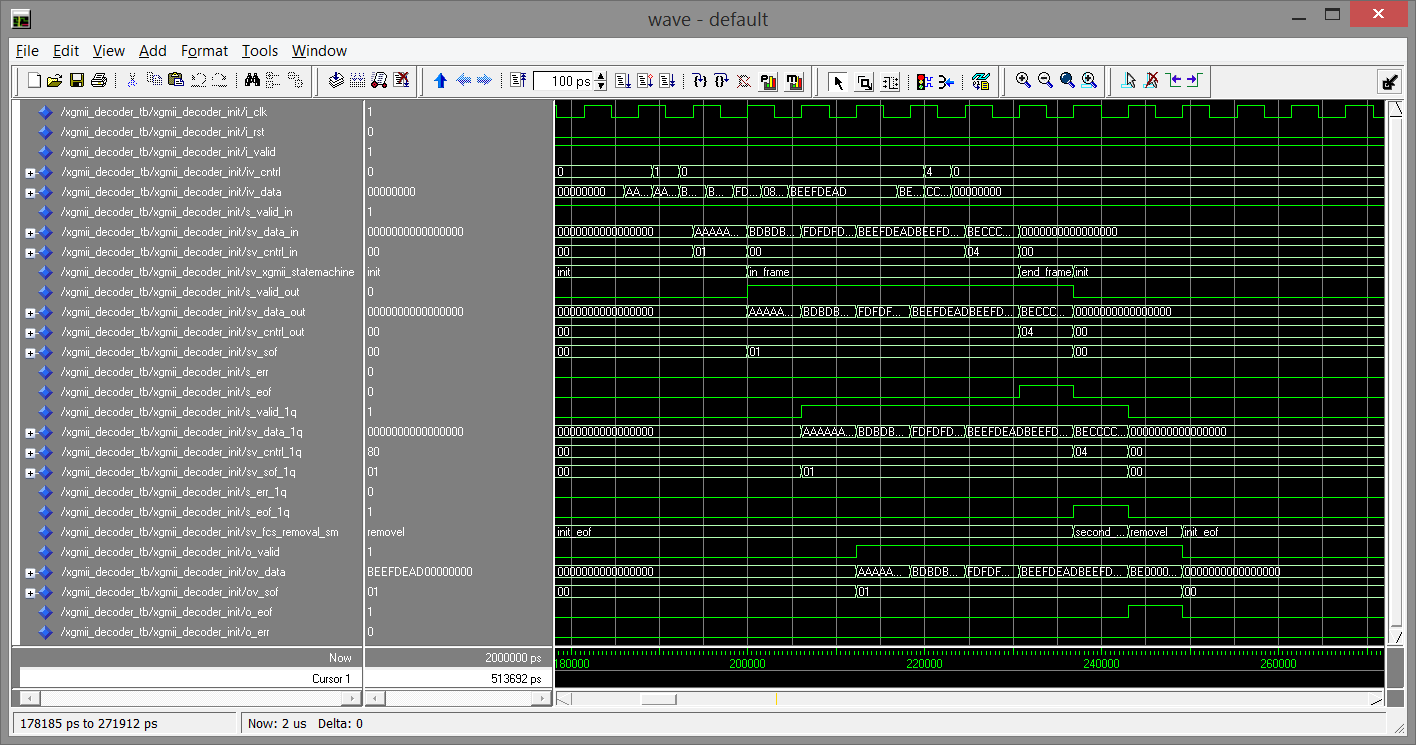
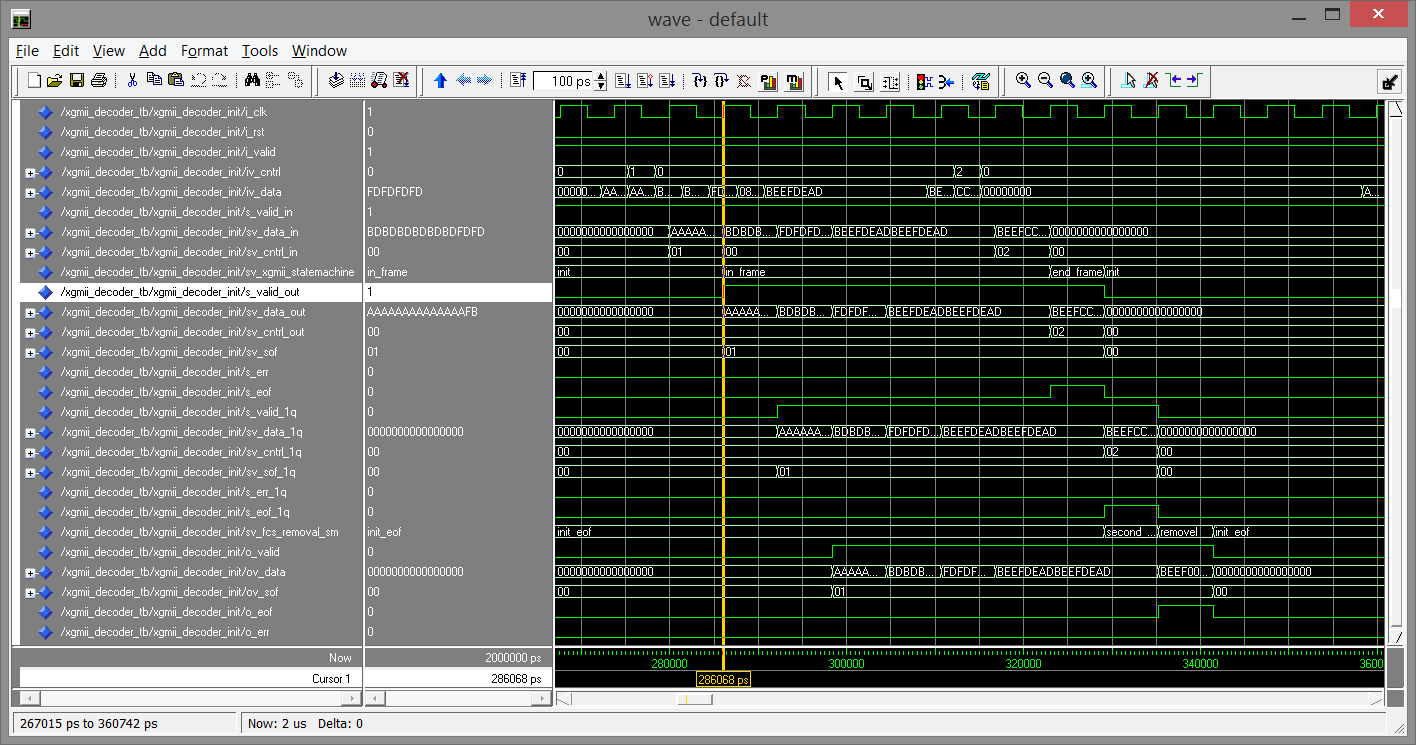
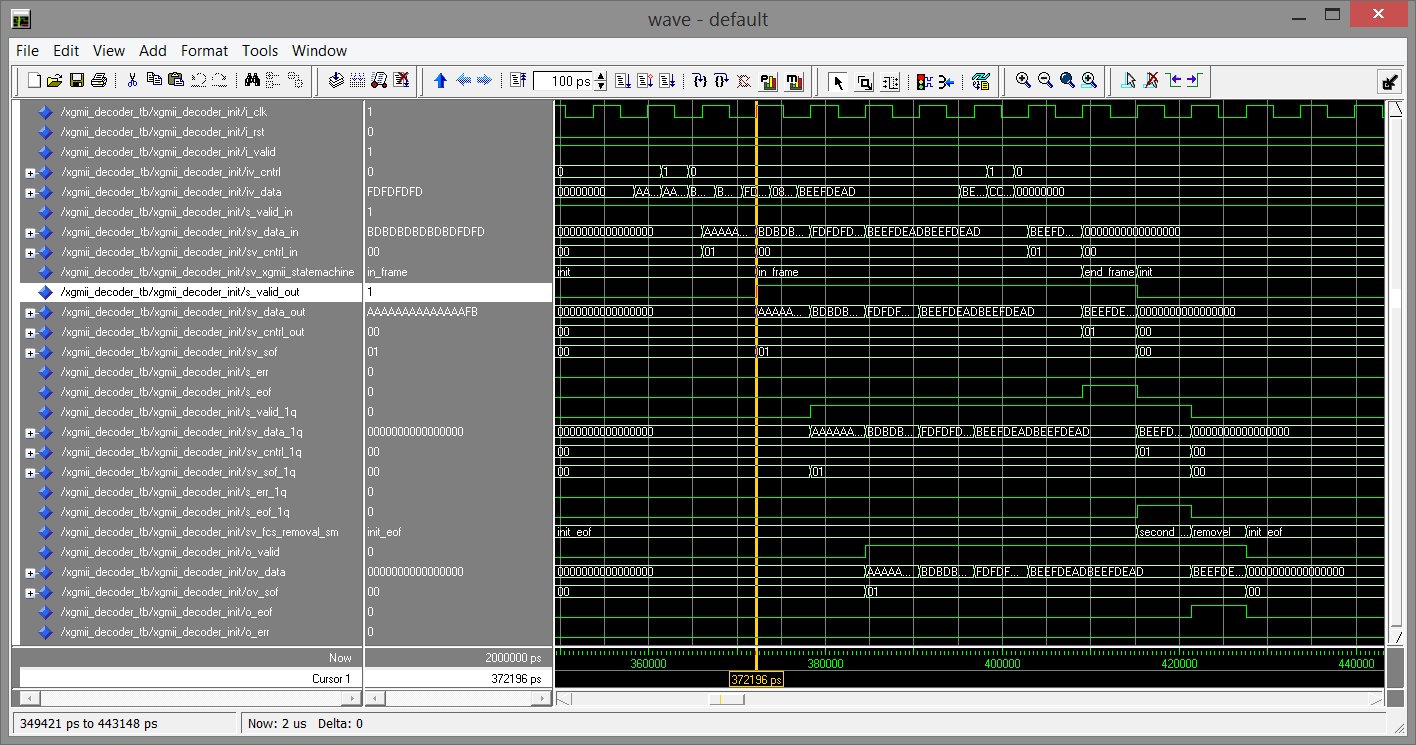
**Appendix A**



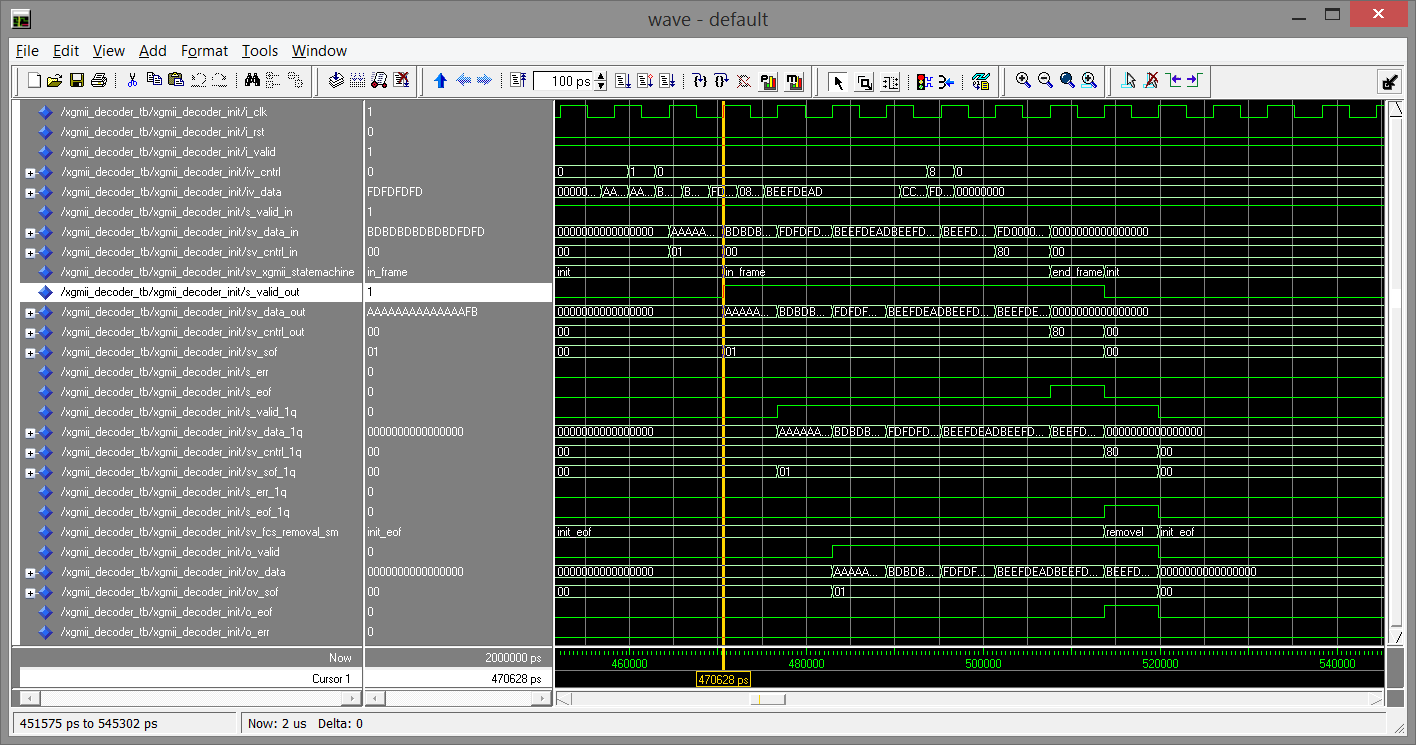
XGMII FCS Removal FSM Test Case 0, Test Frame 2



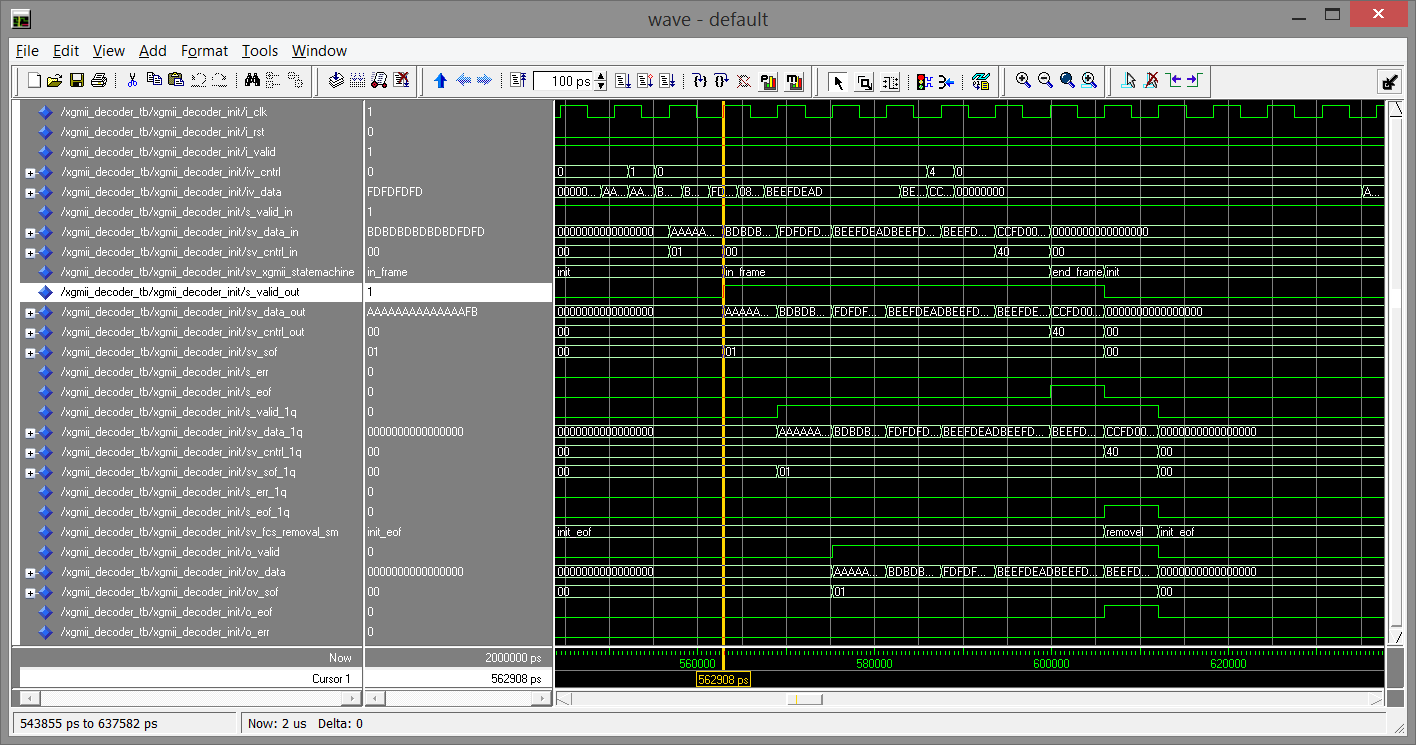
XGMII FCS Removal FSM Test Case 0, Test Frame 3



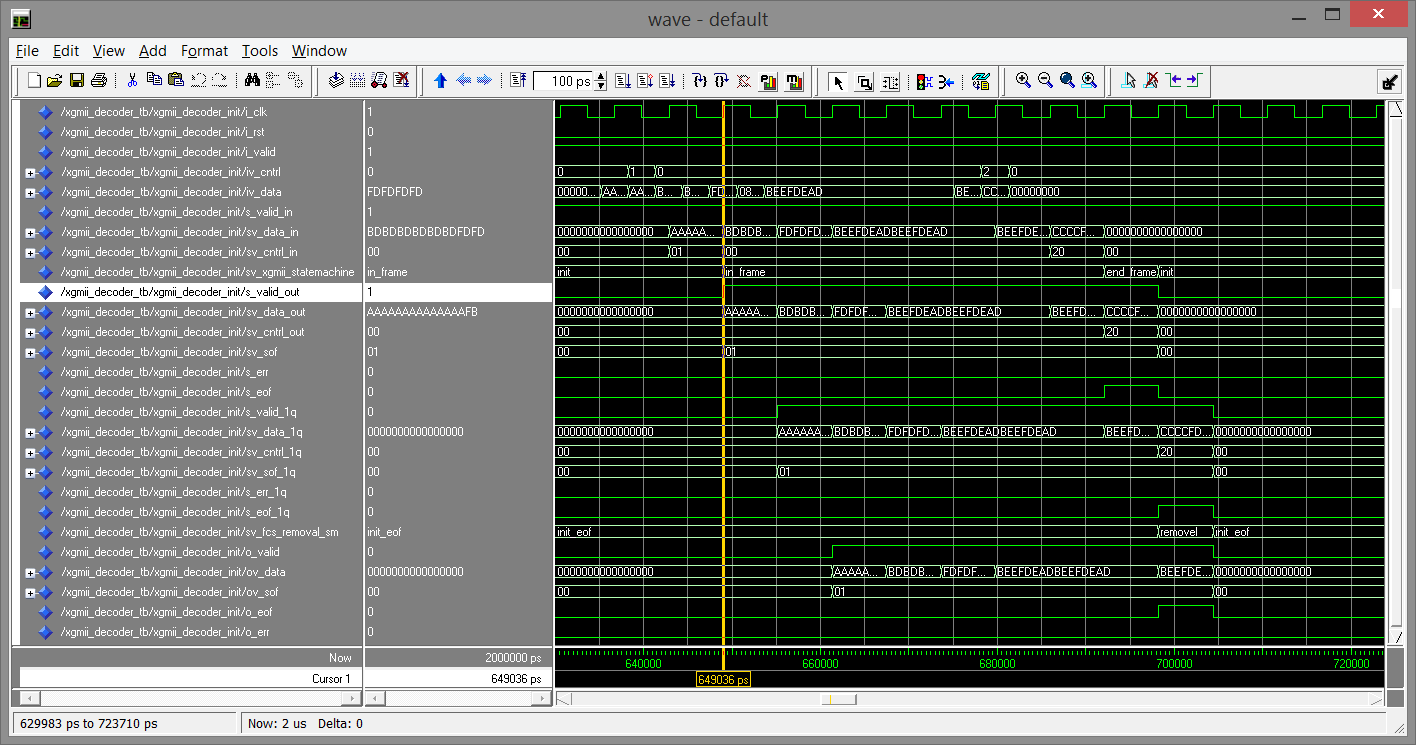
XGMII FCS Removal FSM Test Case 0, Test Frame 4



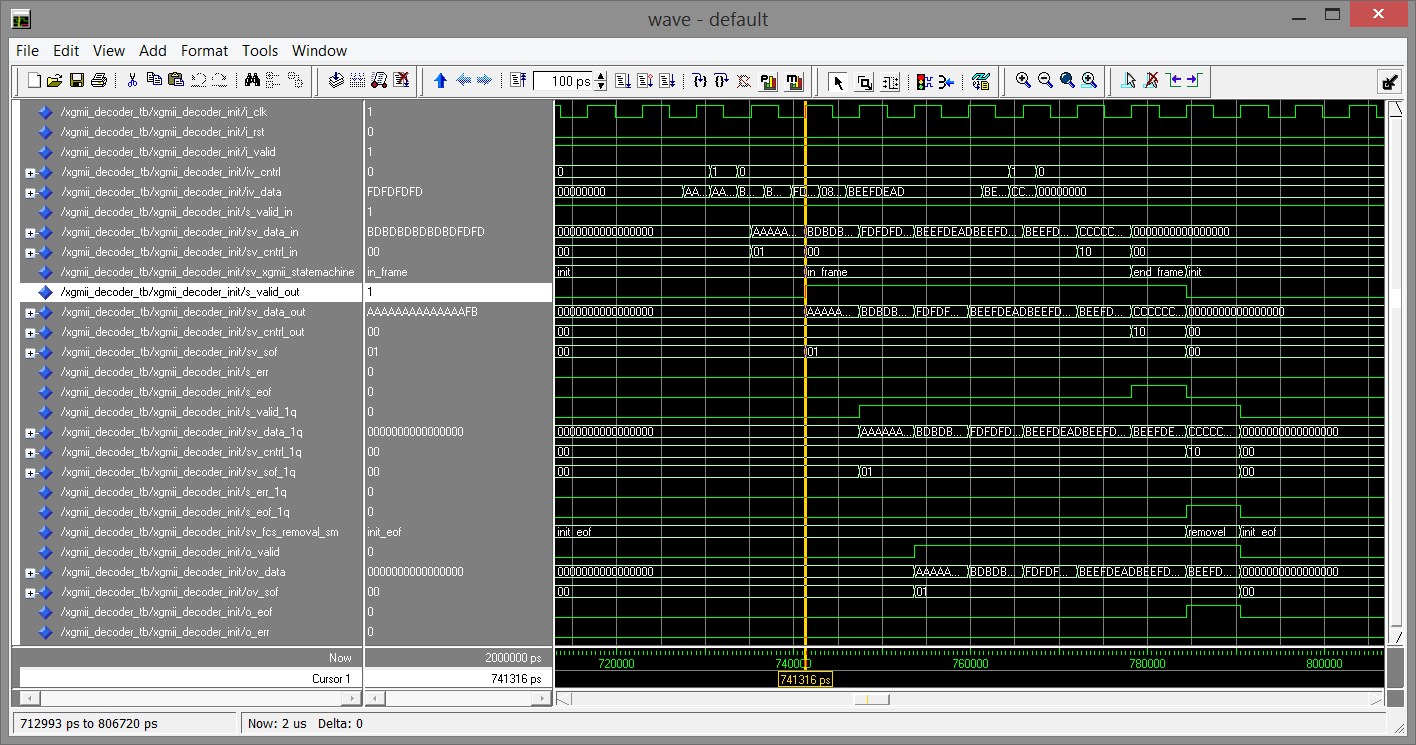
XGMII FCS Removal FSM Test Case 0, Test Frame 5



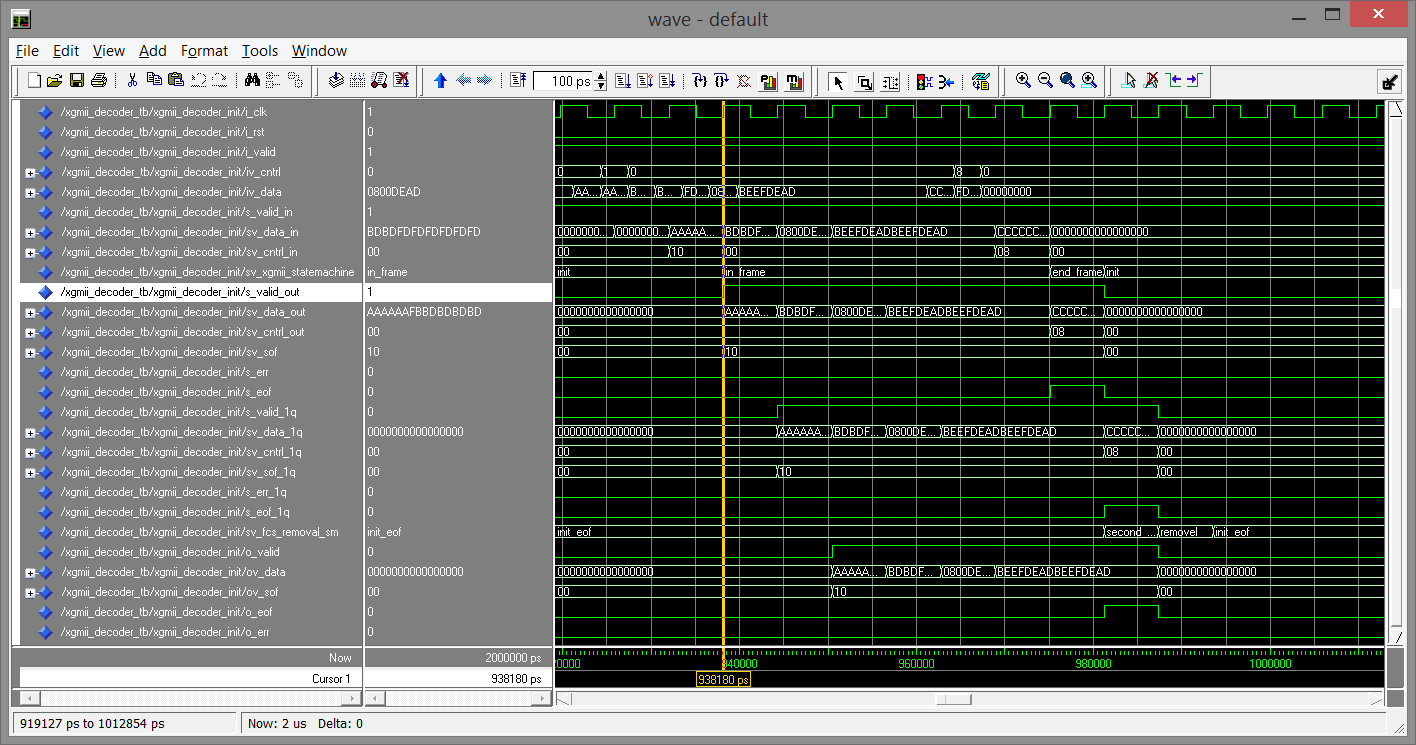
XGMII FCS Removal FSM Test Case 0, Test Frame 6



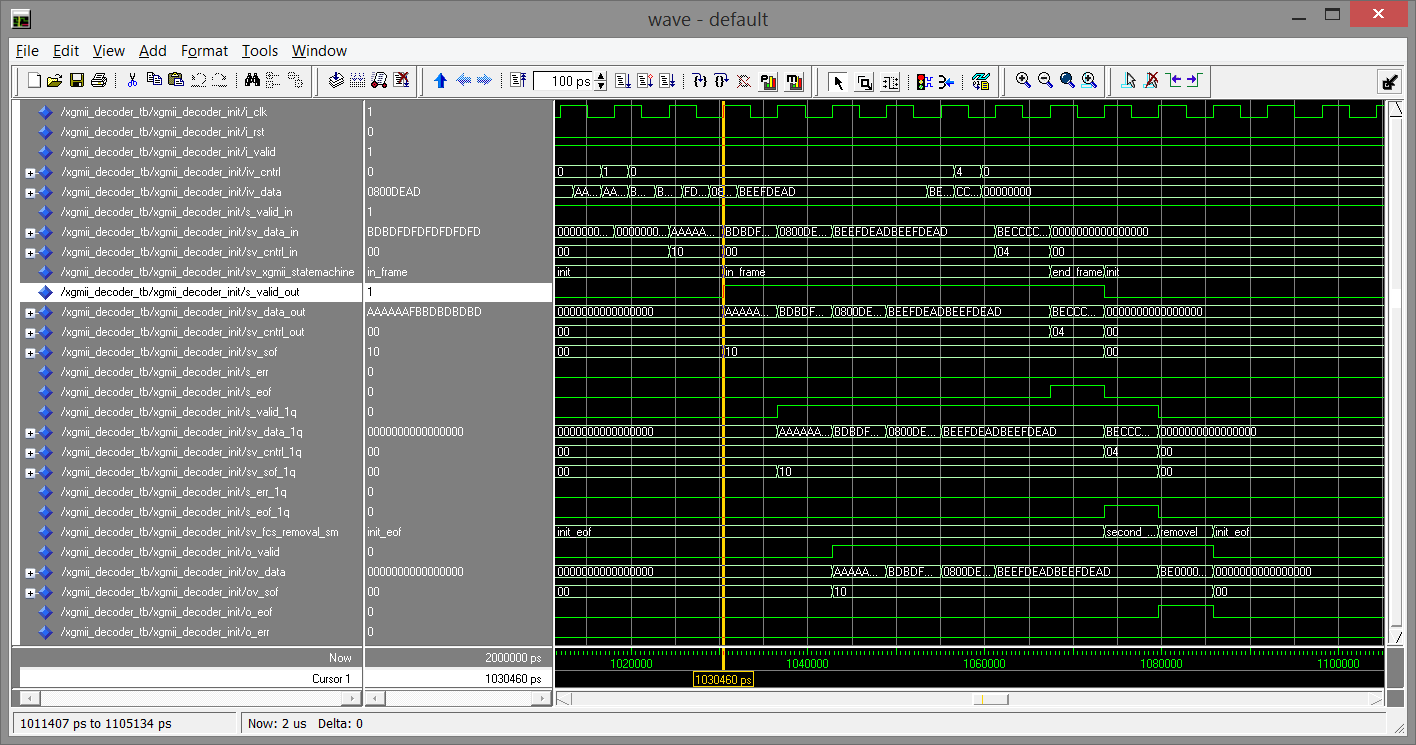
XGMII FCS Removal FSM Test Case 0, Test Frame 7



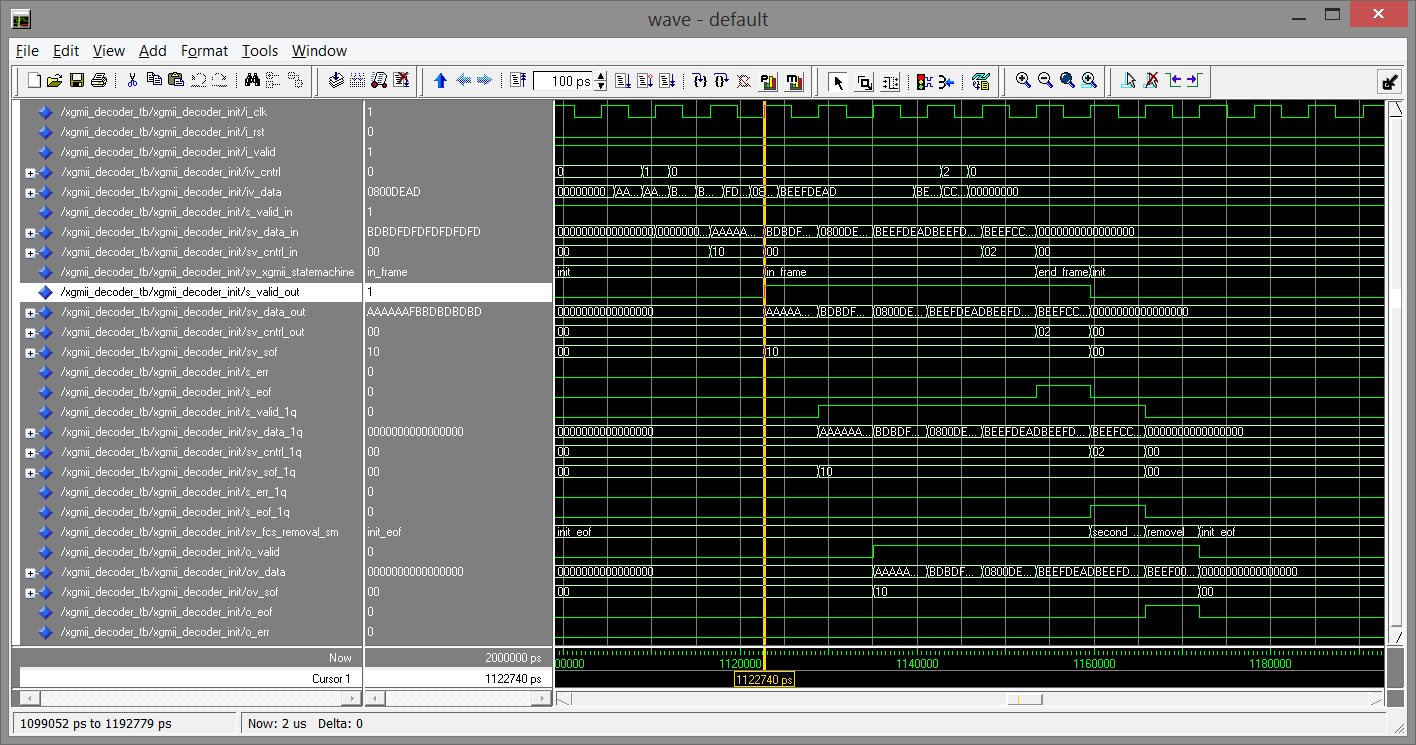
XGMII FCS Removal FSM Test Case 0, Test Frame 8



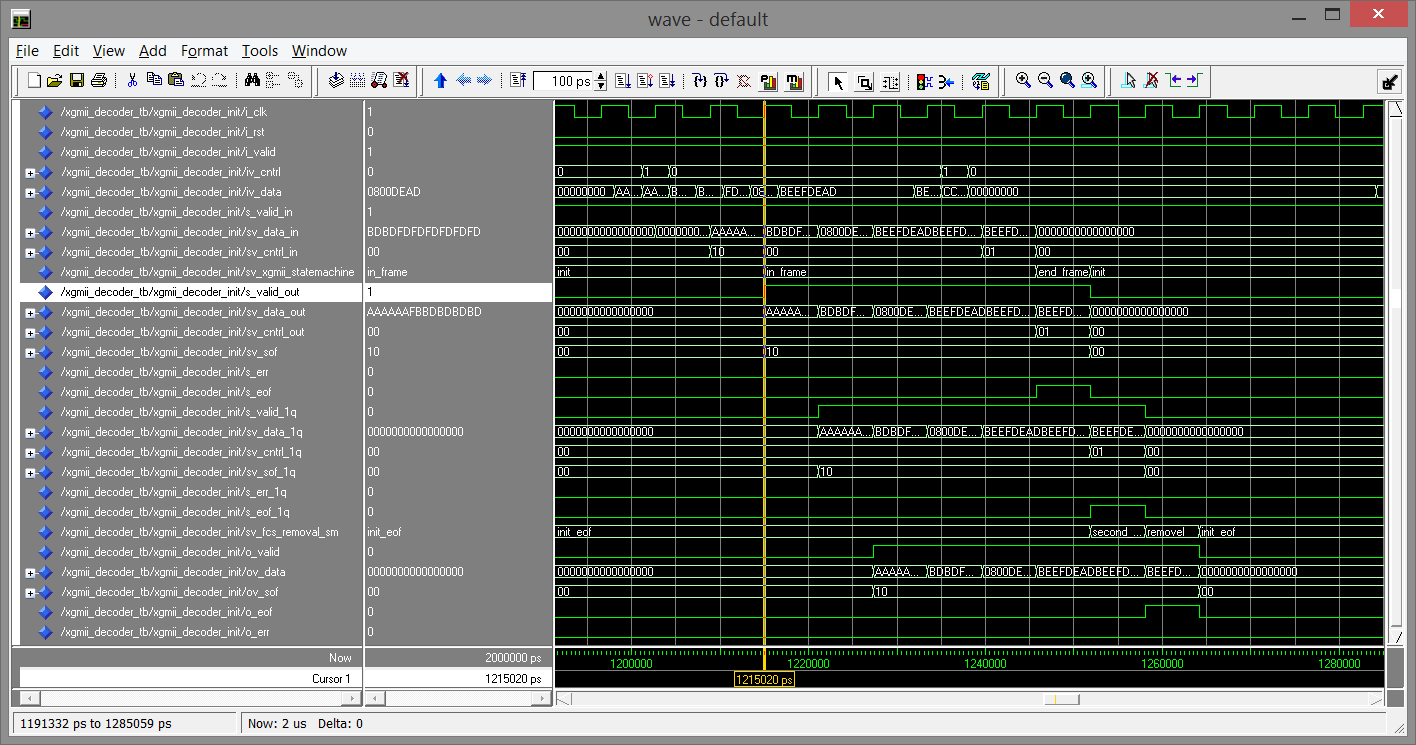
XGMII FCS Removal FSM Test Case 0, Test Frame 9



XGMII FCS Removal FSM Test Case 0, Test Frame 10



XGMII FCS Removal FSM Test Case 0, Test Frame 11



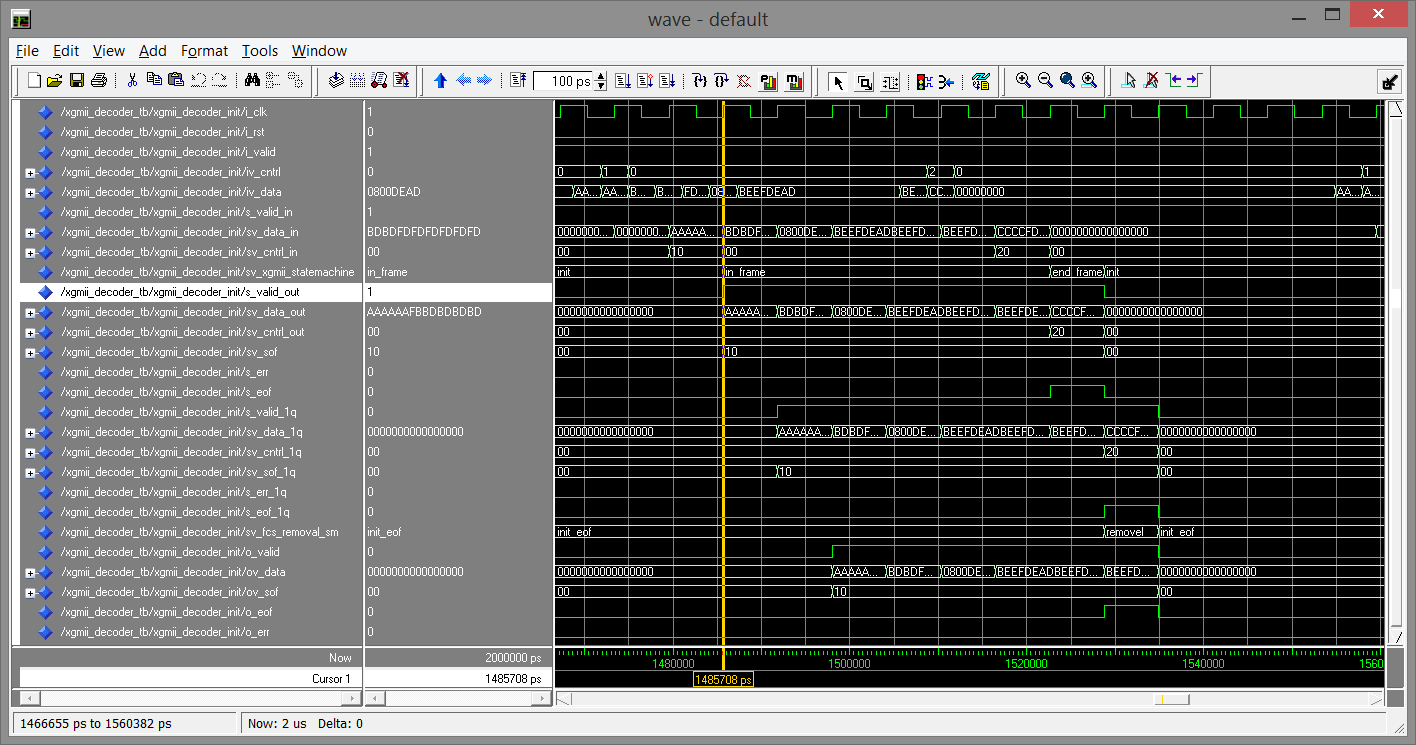
XGMII FCS Removal FSM Test Case 0, Test Frame 12



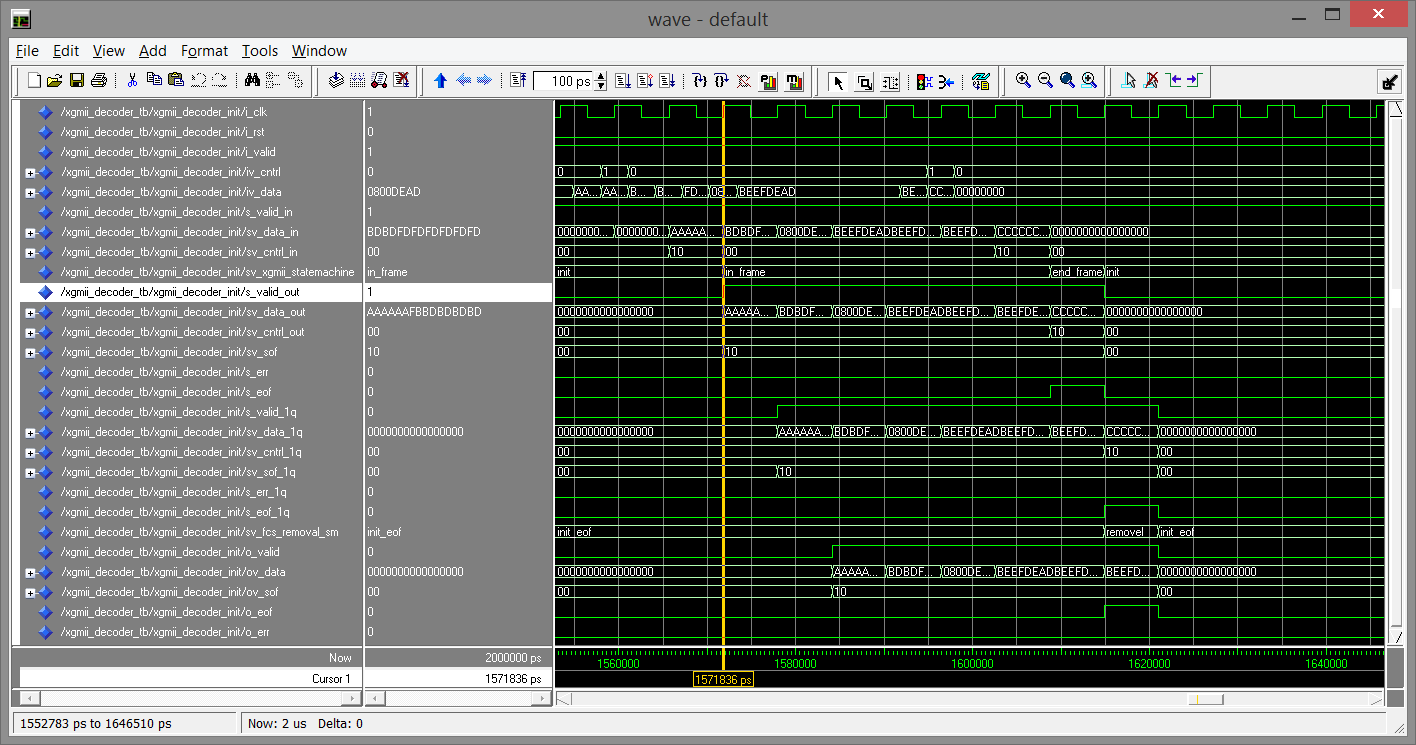
XGMII FCS Removal FSM Test Case 0, Test Frame 13



XGMII FCS Removal FSM Test Case 0, Test Frame 14

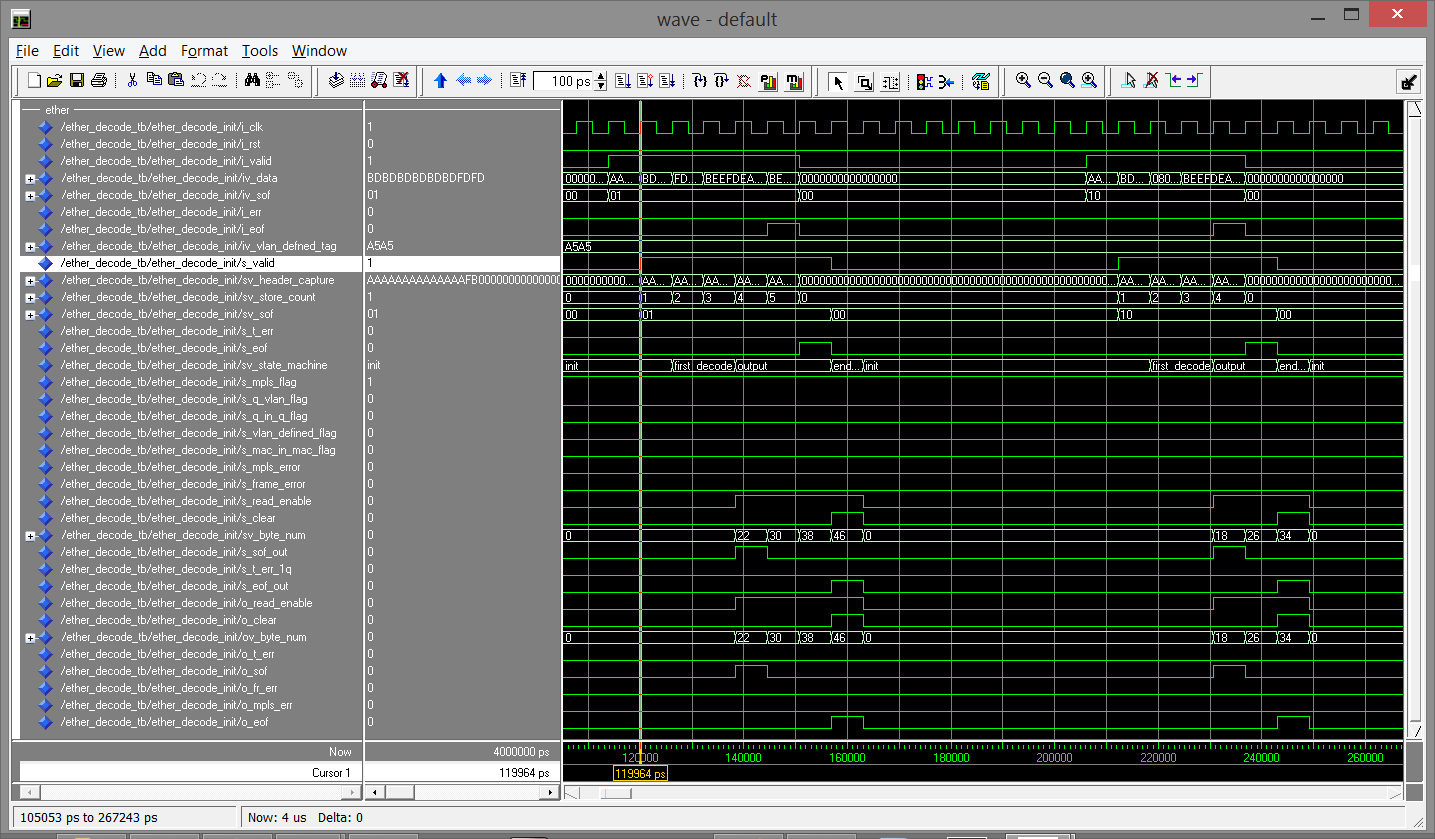


XGMII FCS Removal FSM Test Case 0, Test Frame 15

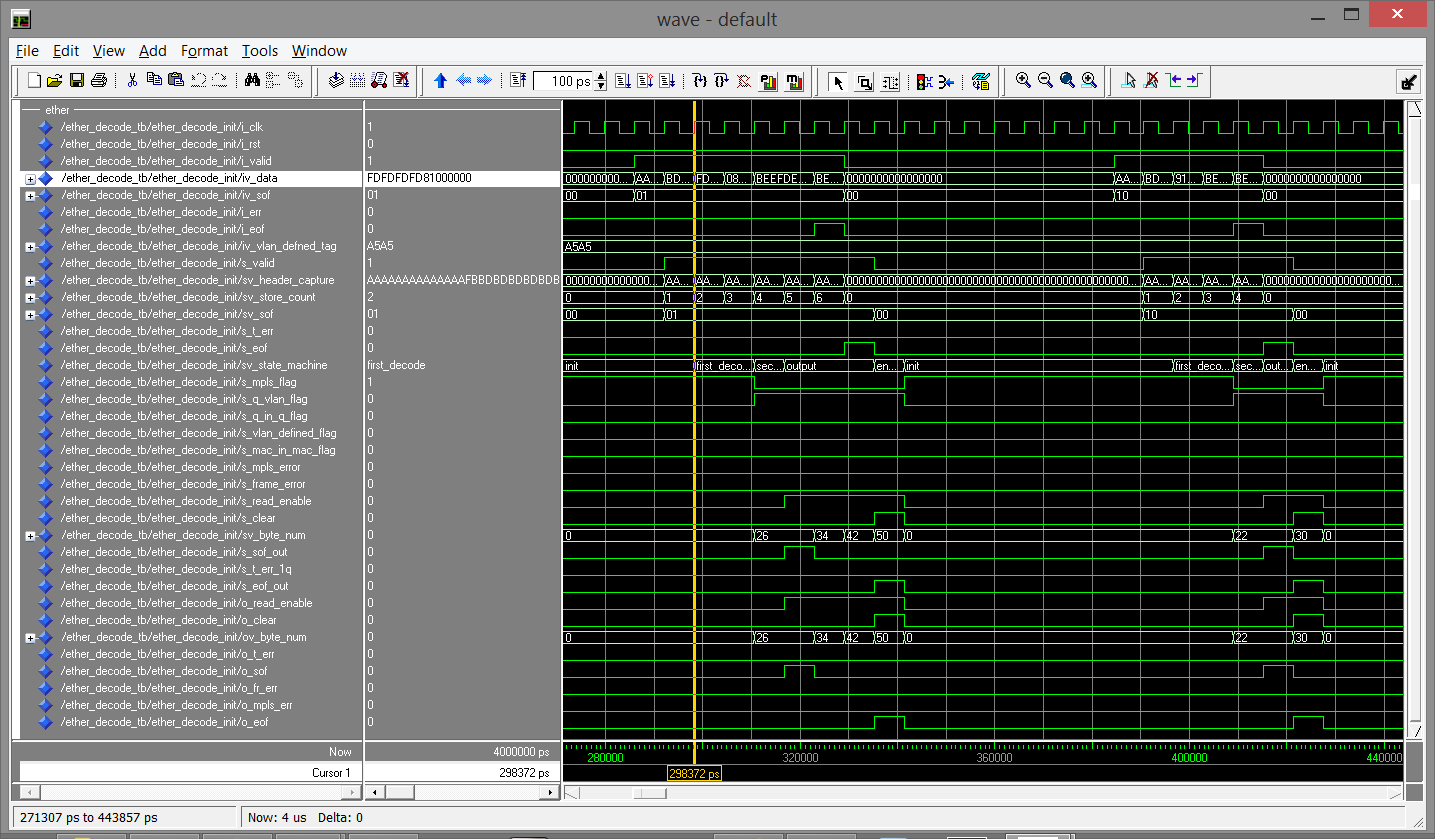


XGMII FCS Removal FSM Test Case 0, Test Frame 16

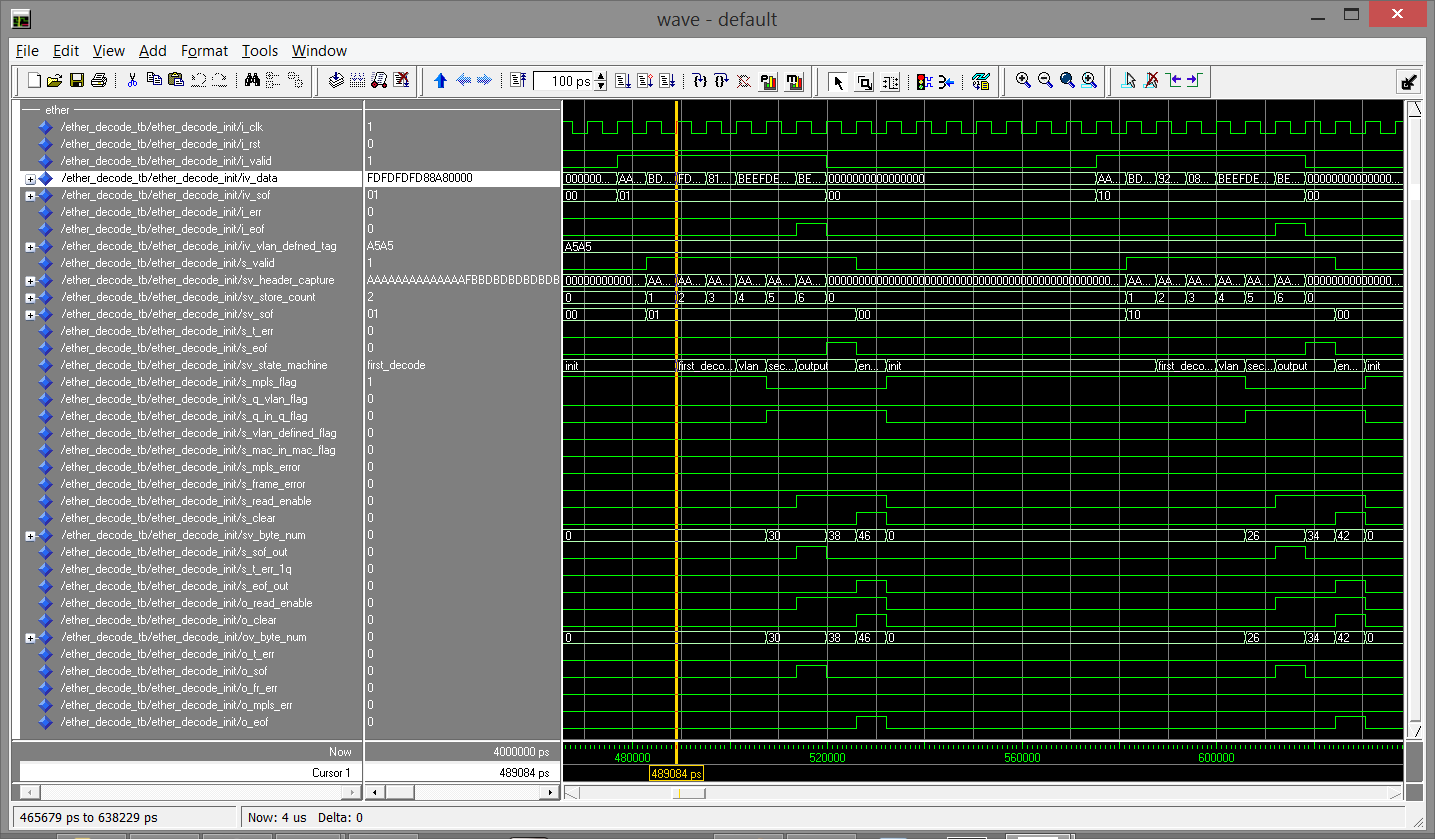
**Appendix B**



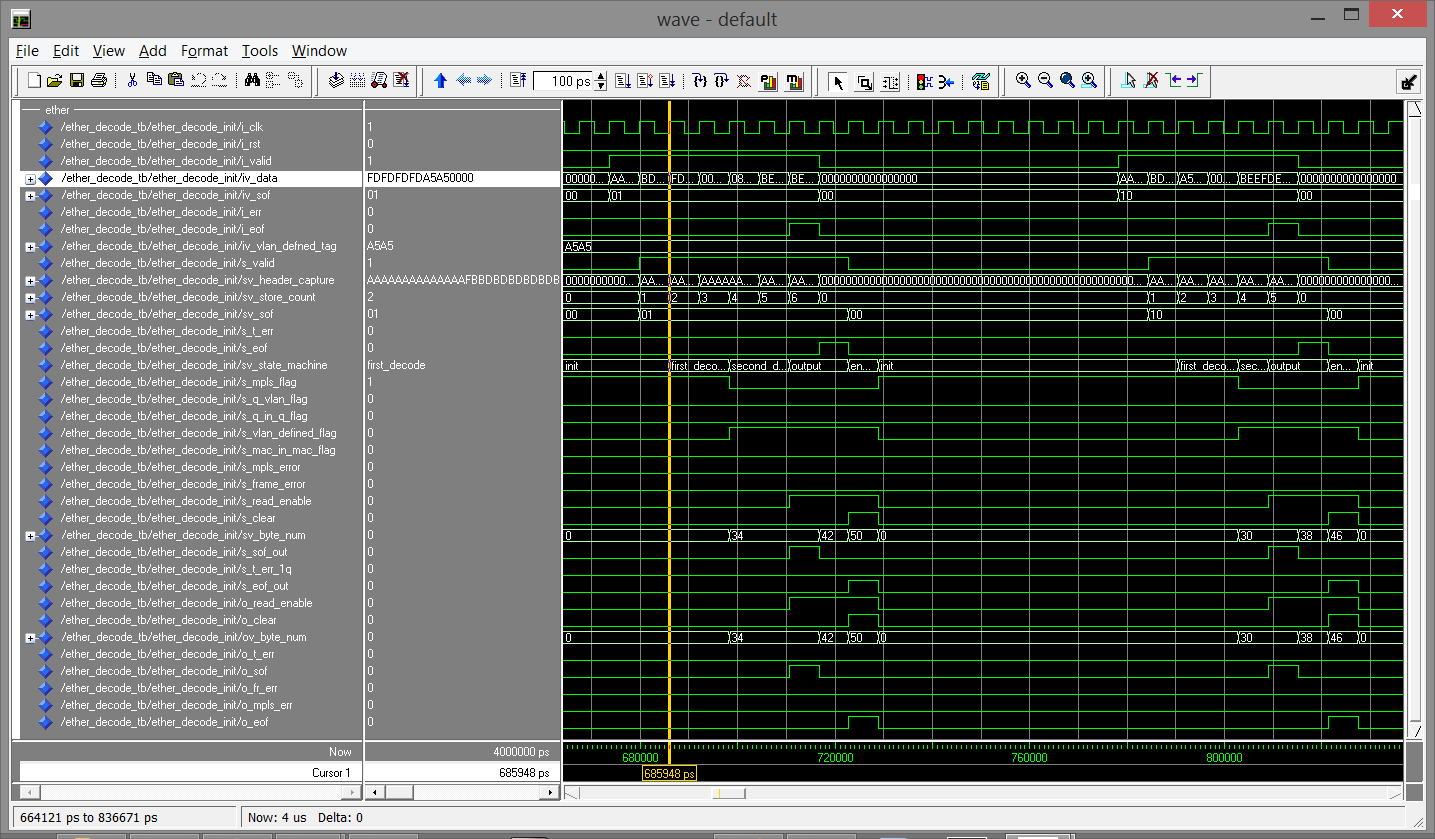
Ether-type FSM Test Case 0, Test Frame 1 and 2



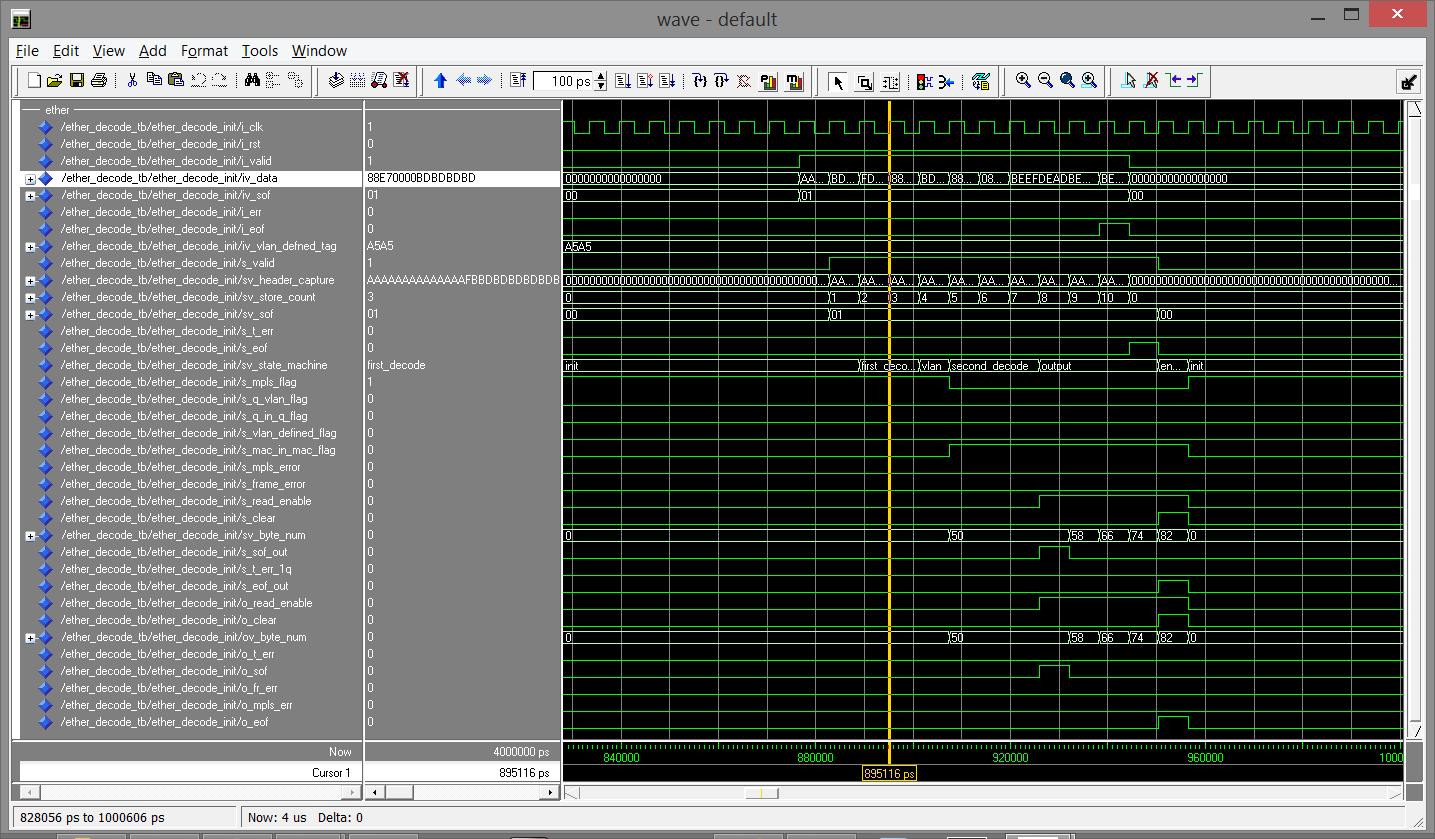
Ether-type FSM Test Case 0, Test Frame 3 and 4



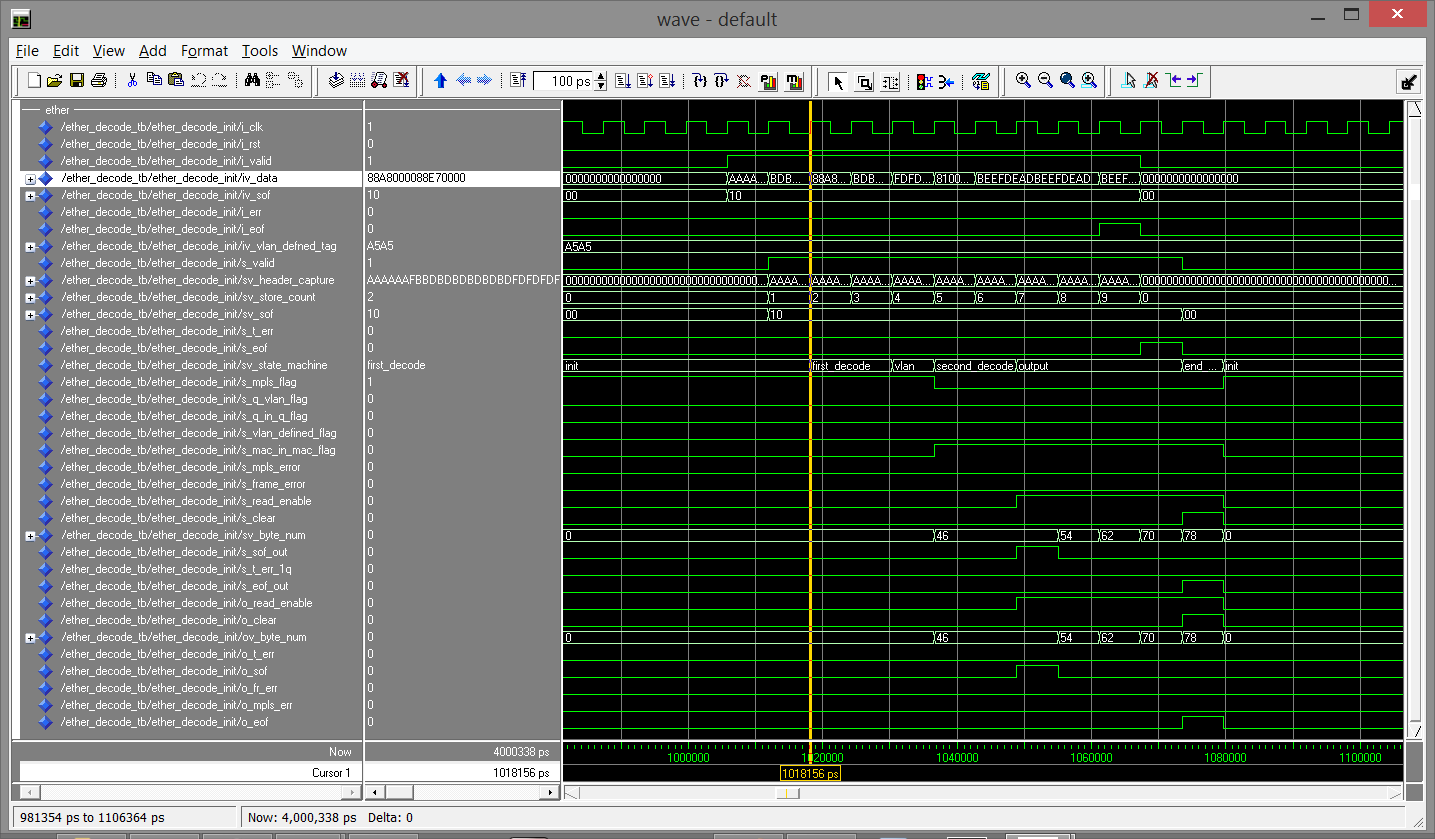
Ether-type FSM Test Case 0, Test Frame 5 and 6



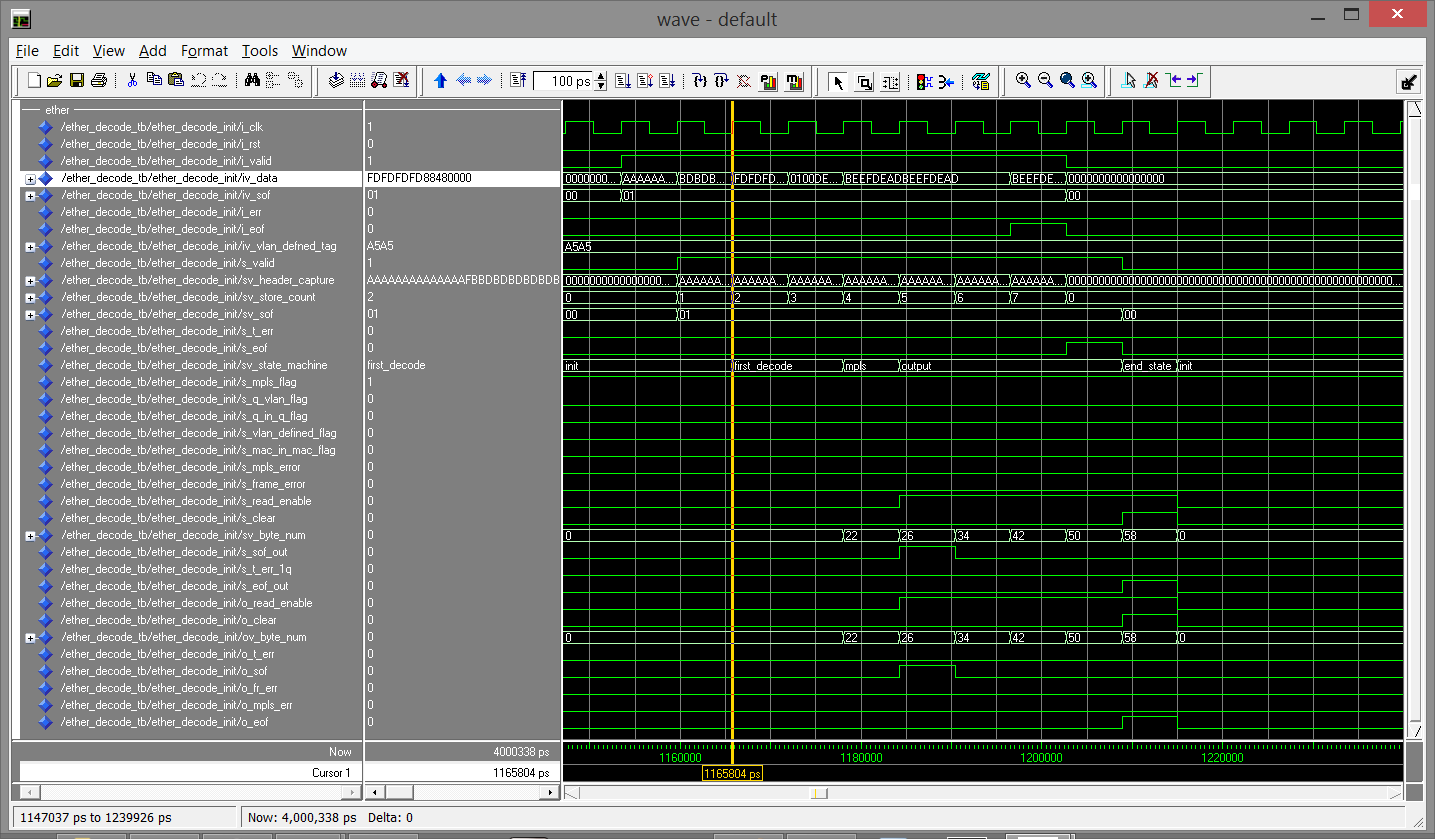
Ether-type FSM Test Case 0, Test Frame 7 and 8



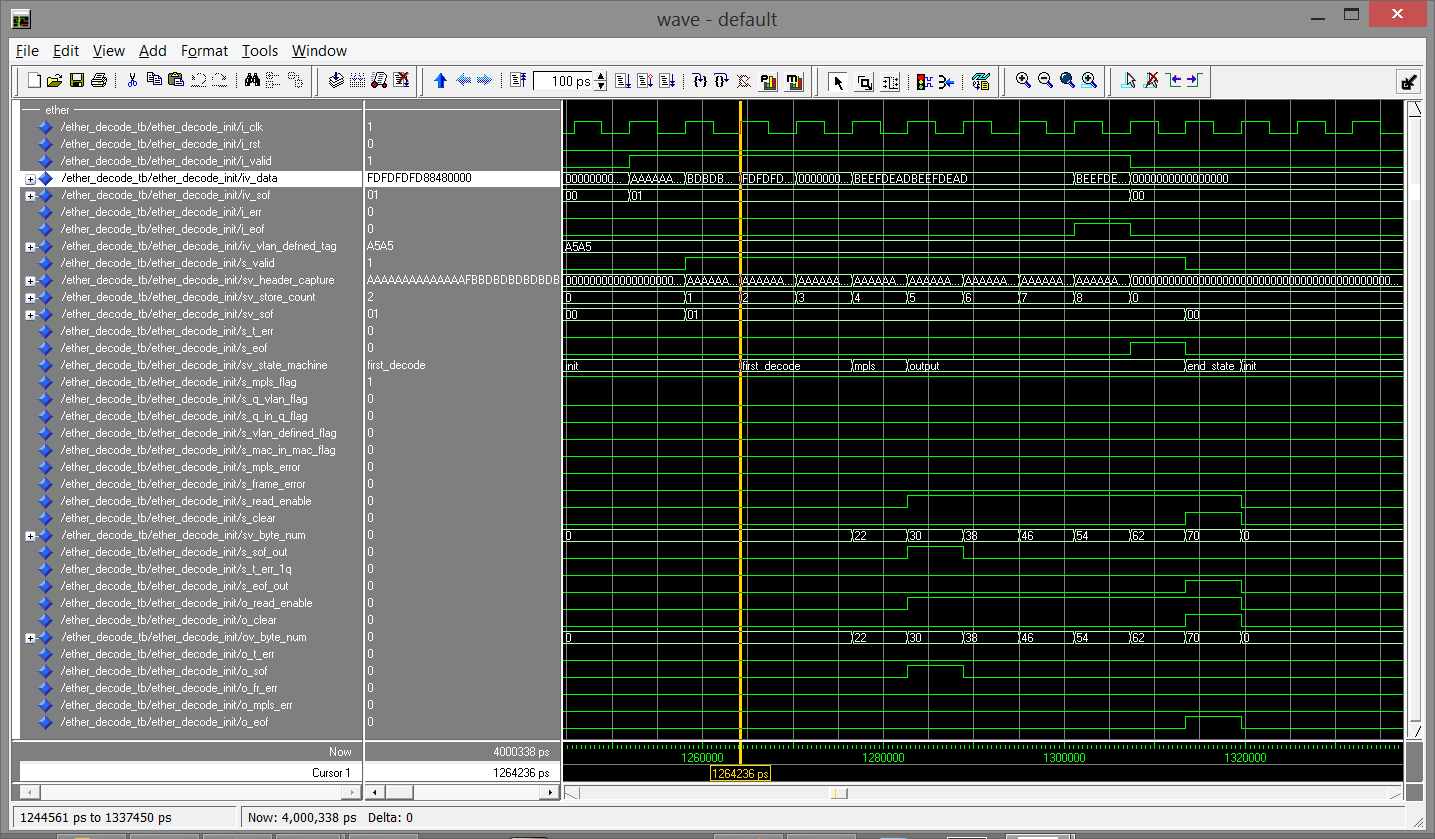
Ether-type FSM Test Case 0, Test Frame 9



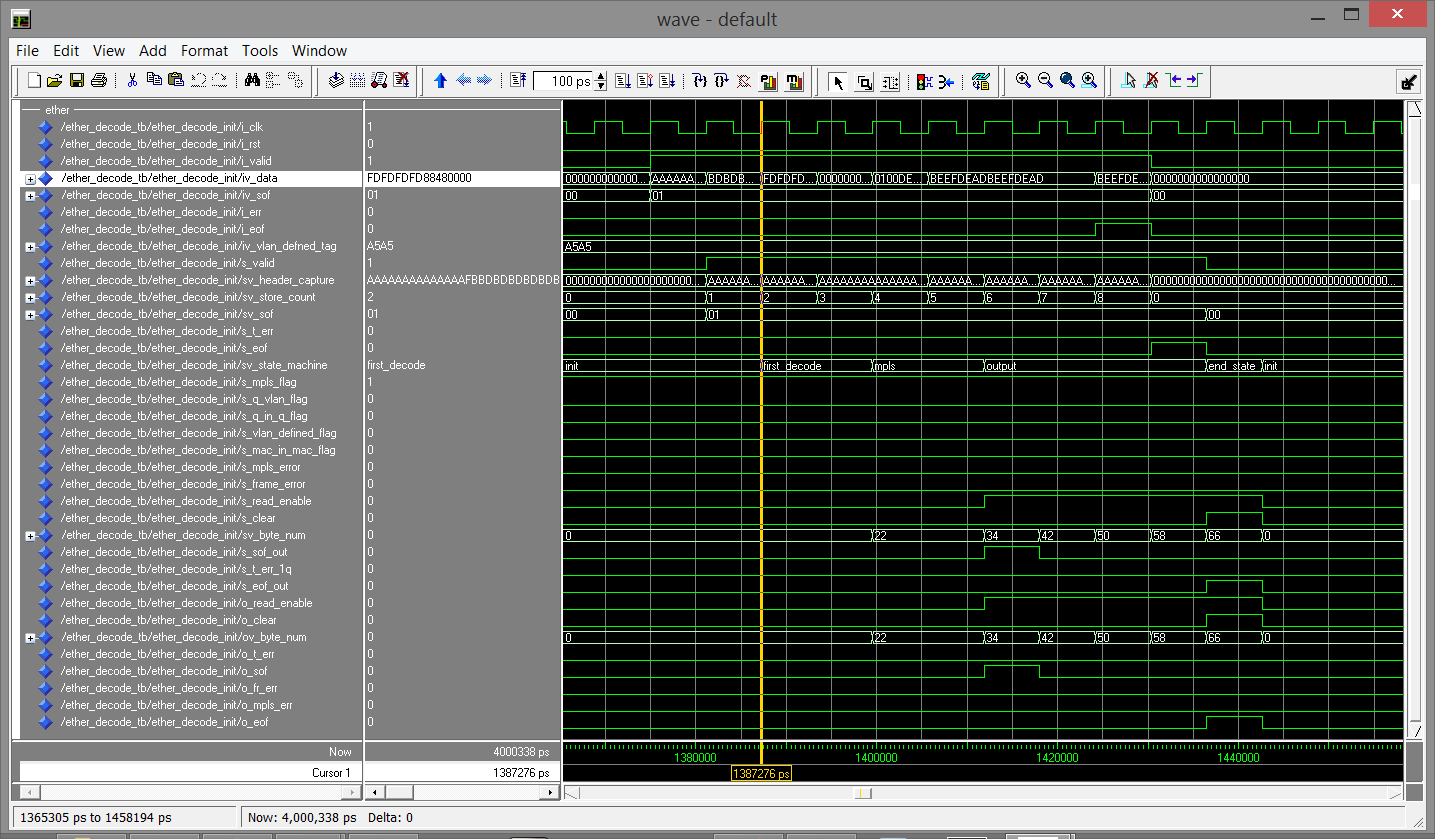
Ether-type FSM Test Case 0, Test Frame 10



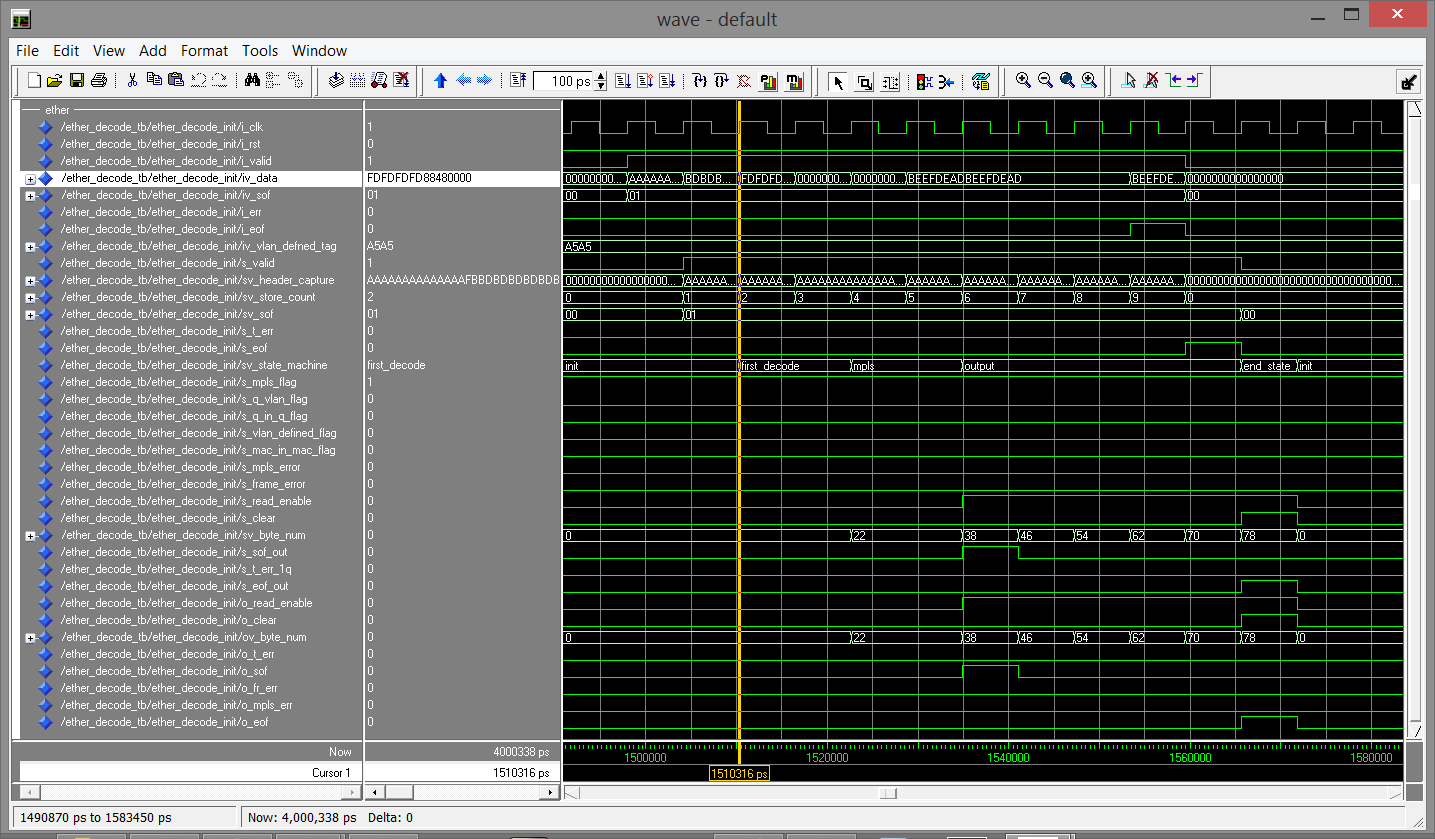
Ether-type FSM Test Case 0, Test Frame 11



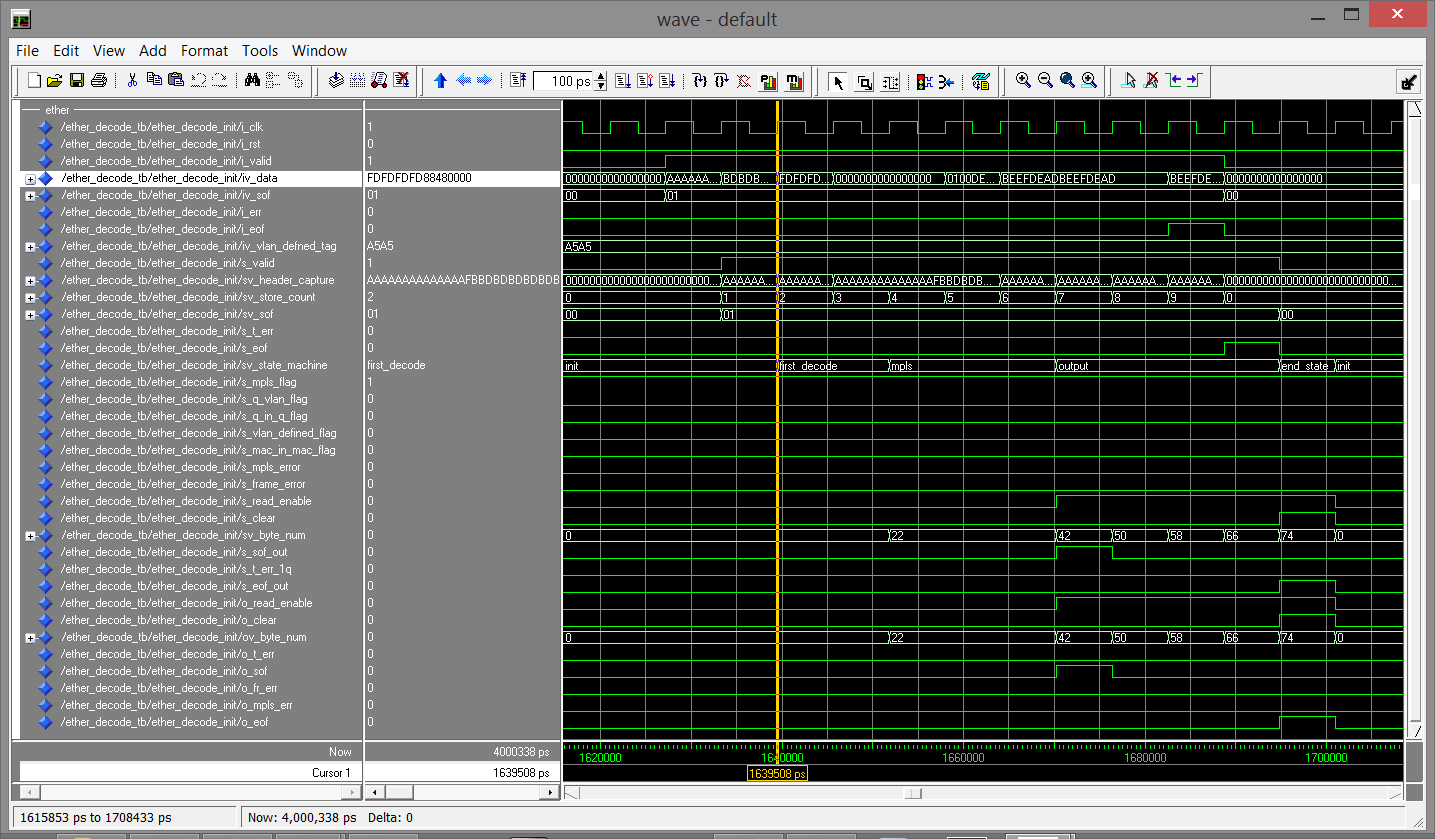
Ether-type FSM Test Case 0, Test Frame 12



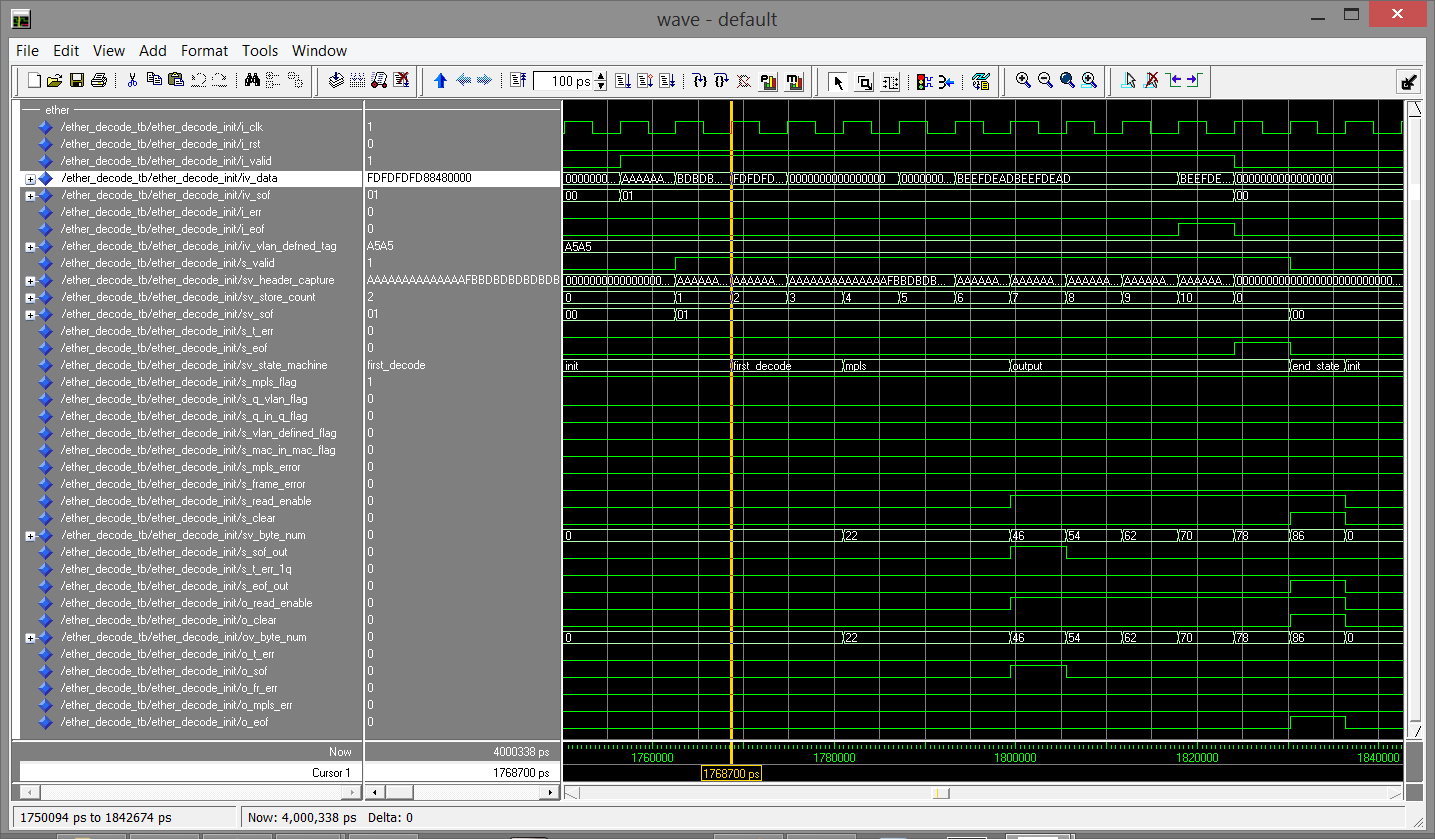
Ether-type FSM Test Case 0, Test Frame 13



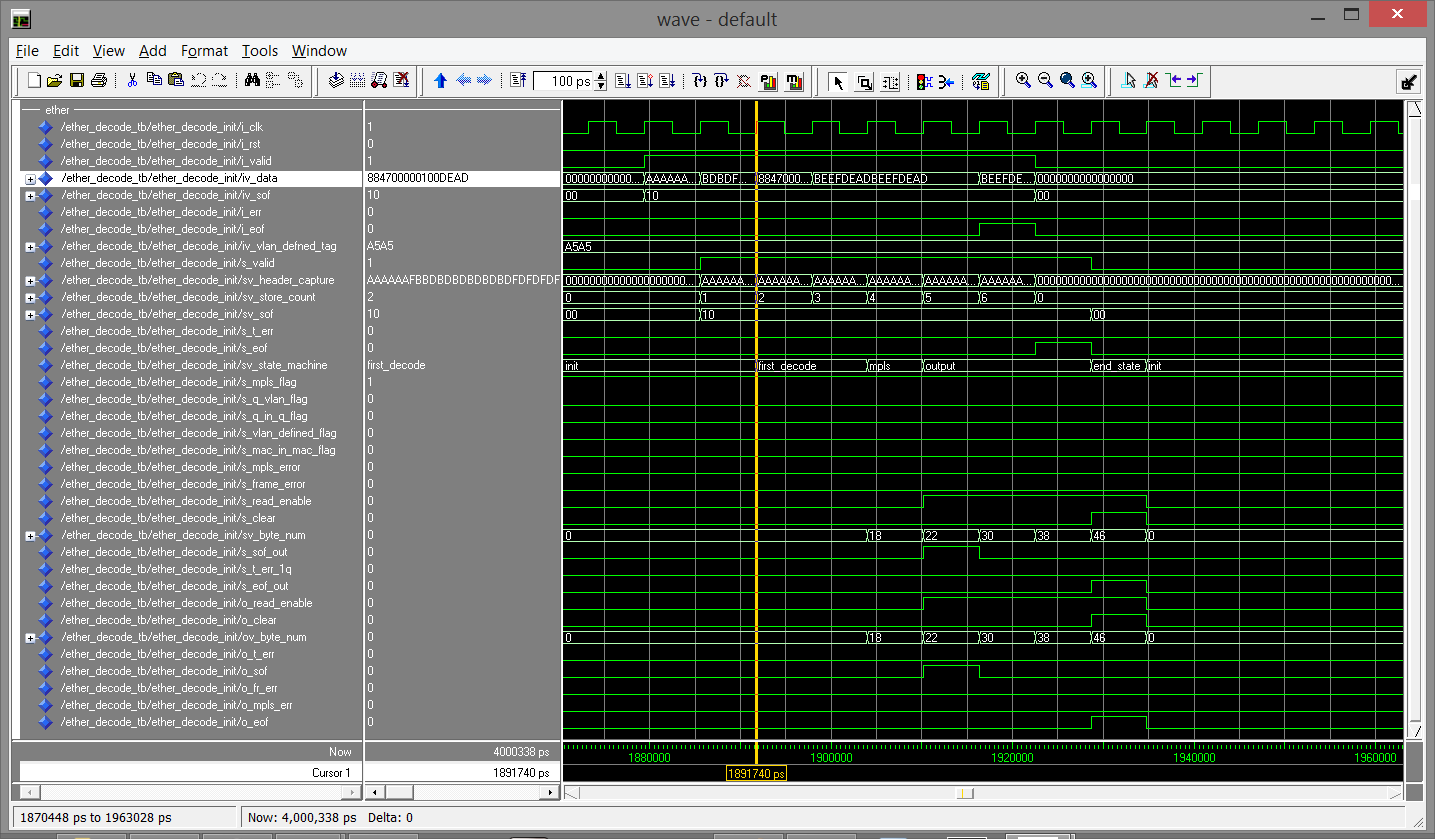
Ether-type FSM Test Case 0, Test Frame 14



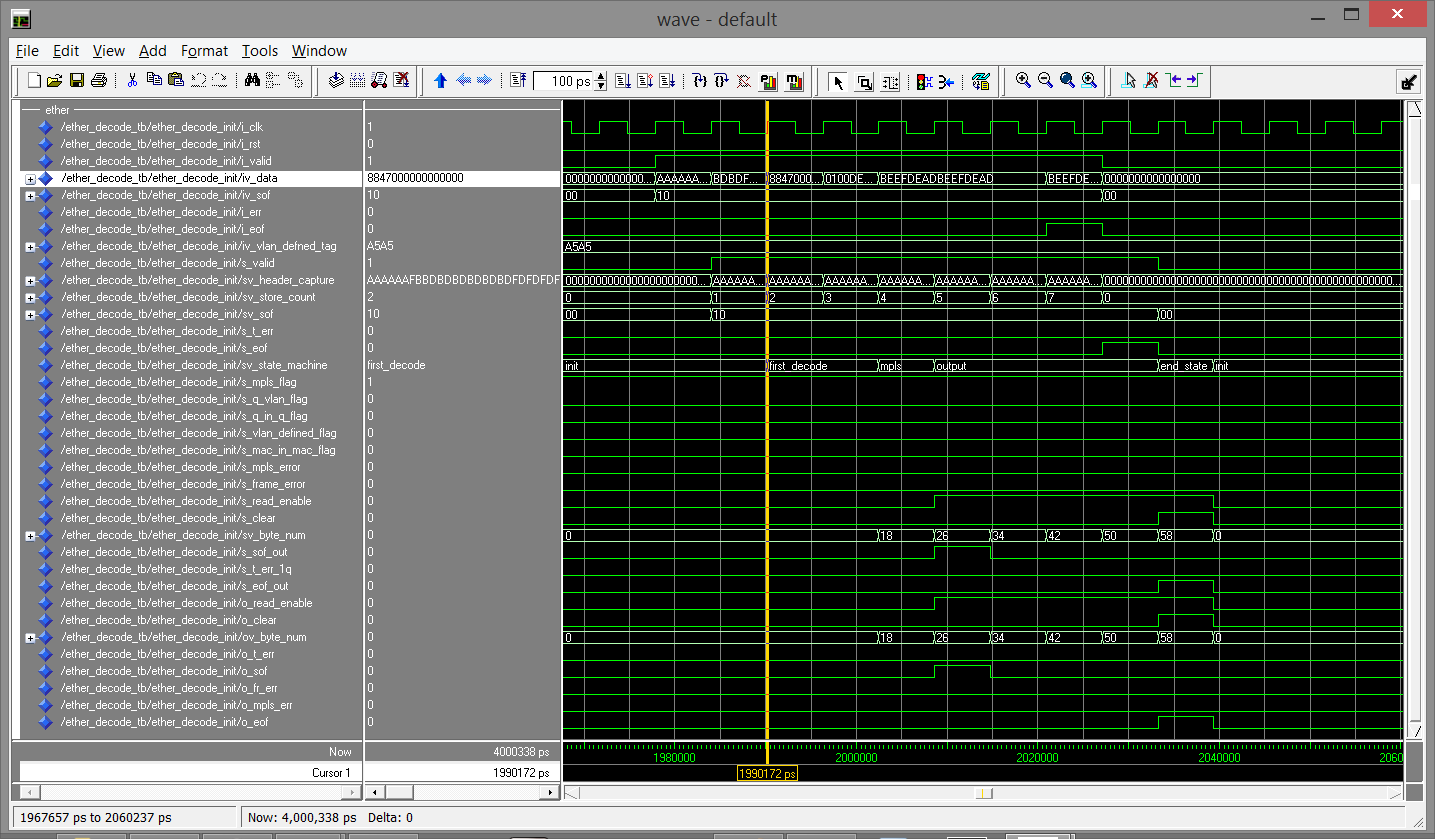
Ether-type FSM Test Case 0, Test Frame 15



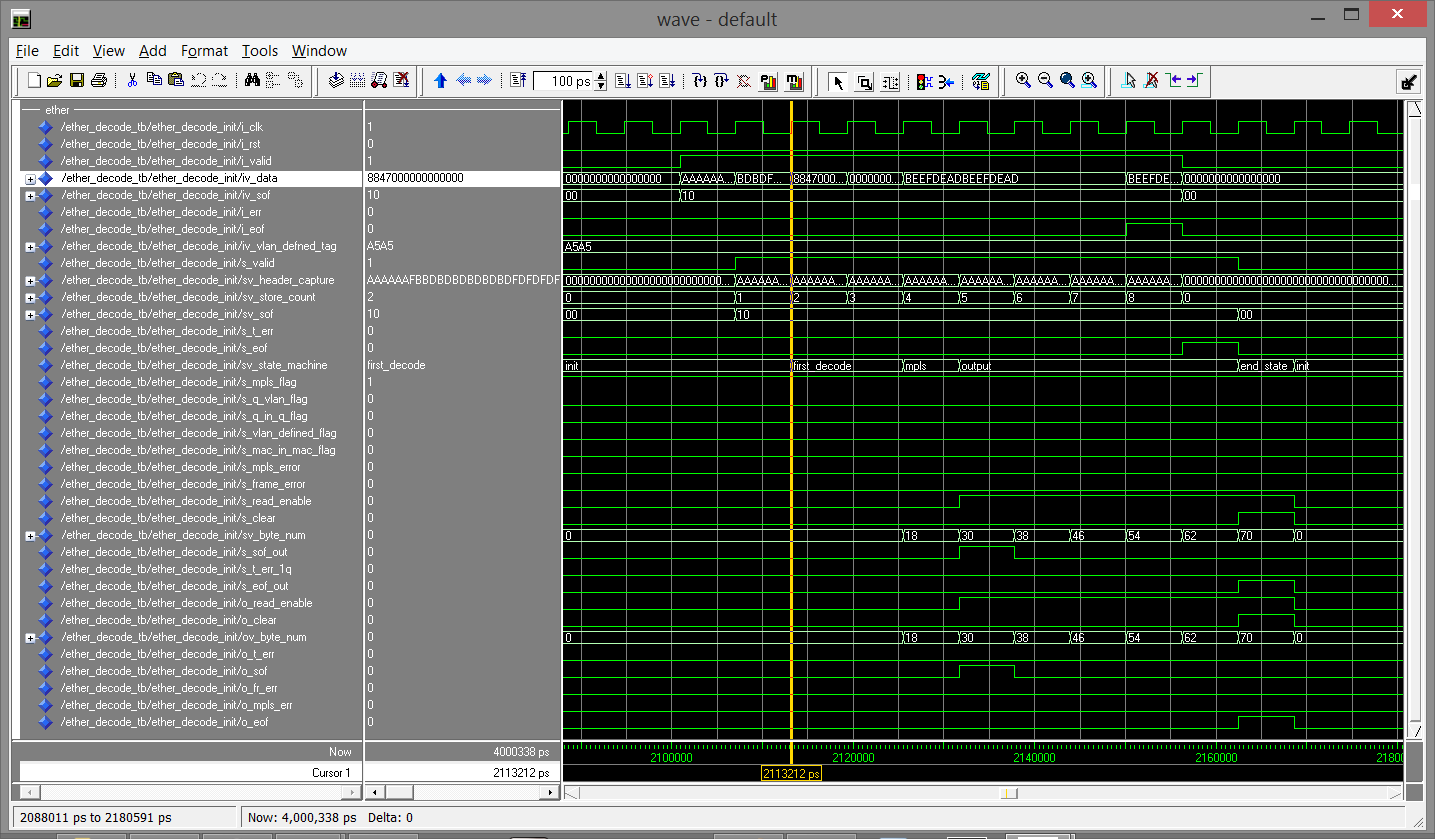
Ether-type FSM Test Case 0, Test Frame 16



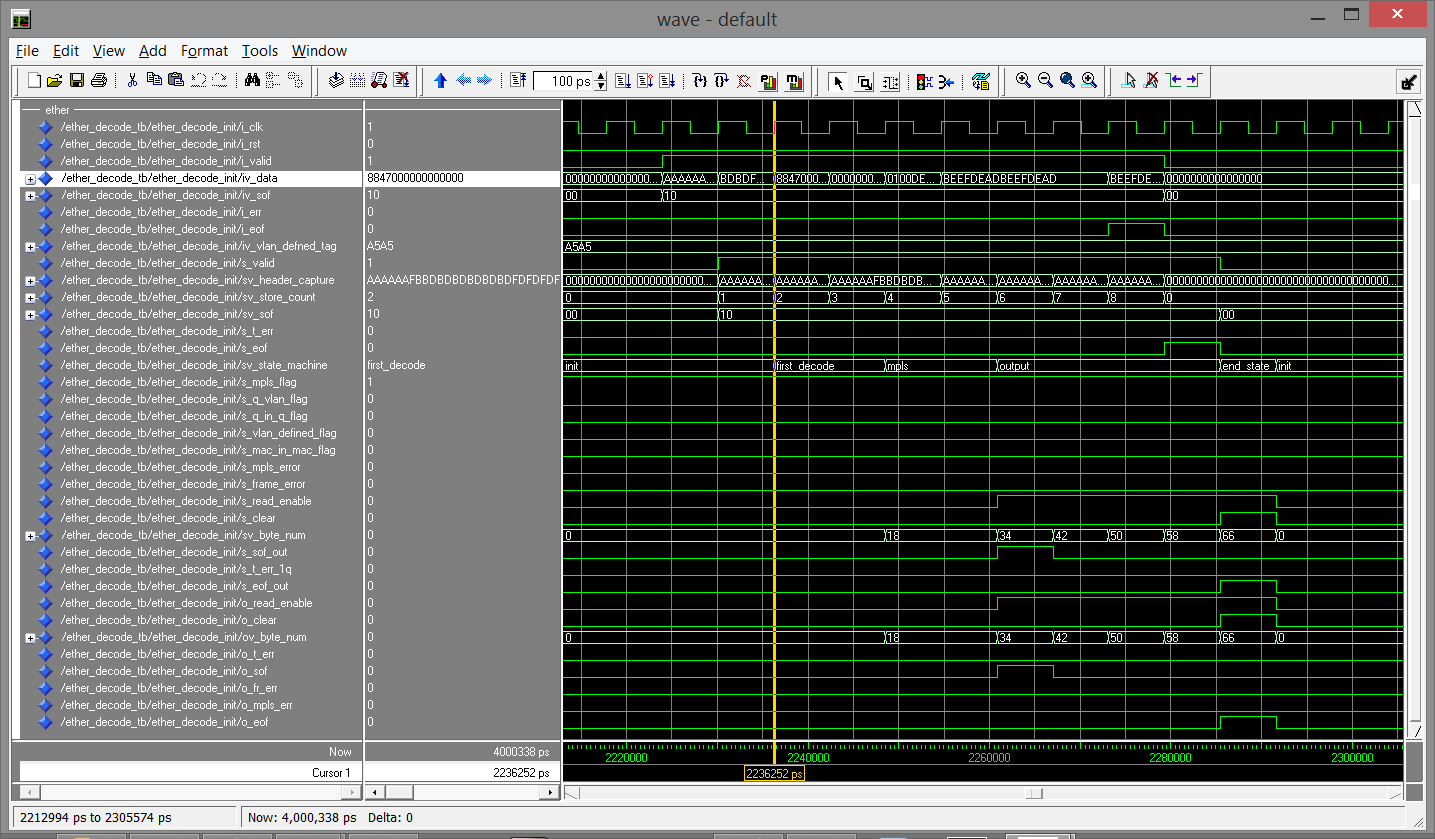
Ether-type FSM Test Case 0, Test Frame 17



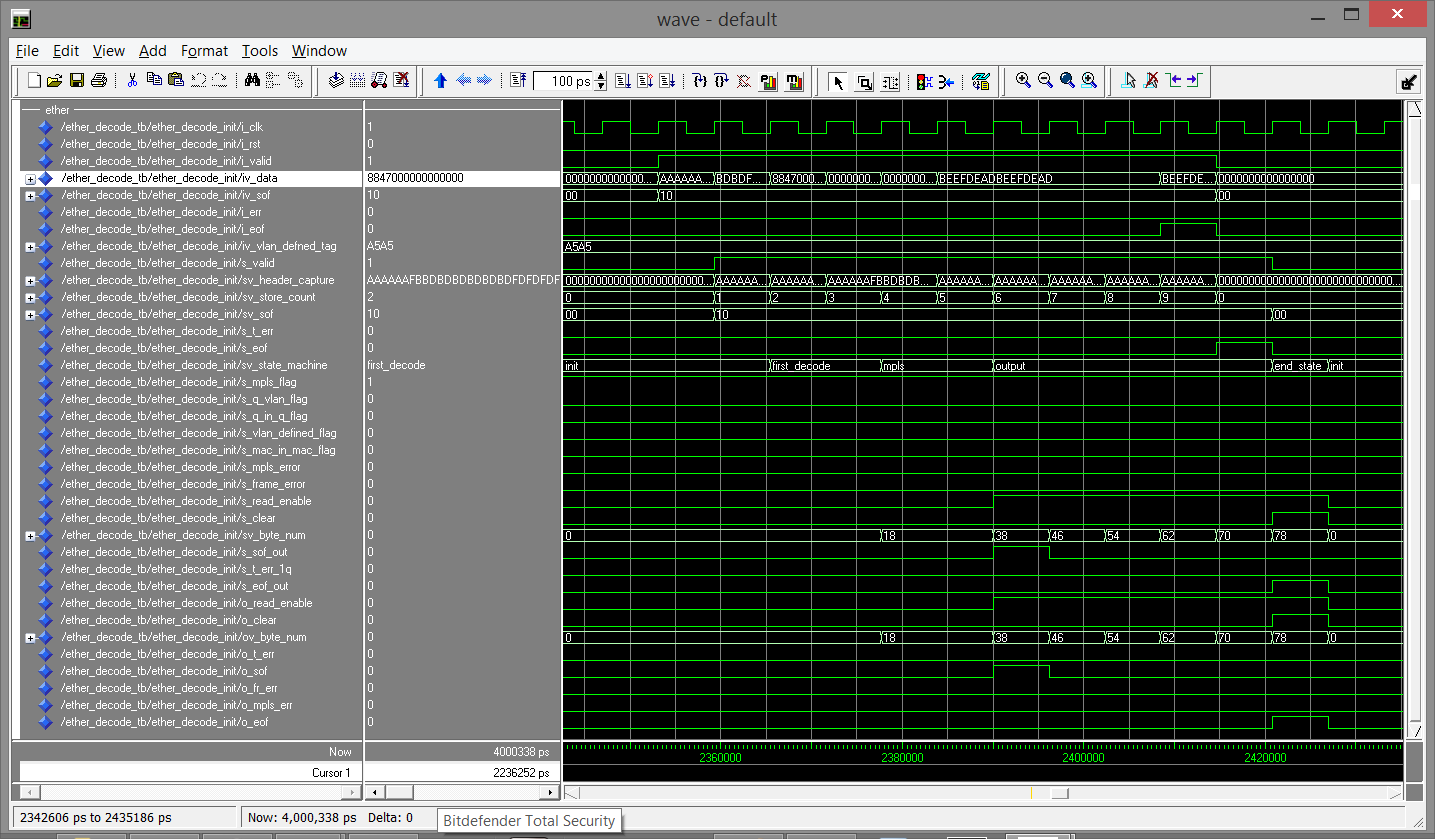
Ether-type FSM Test Case 0, Test Frame 18



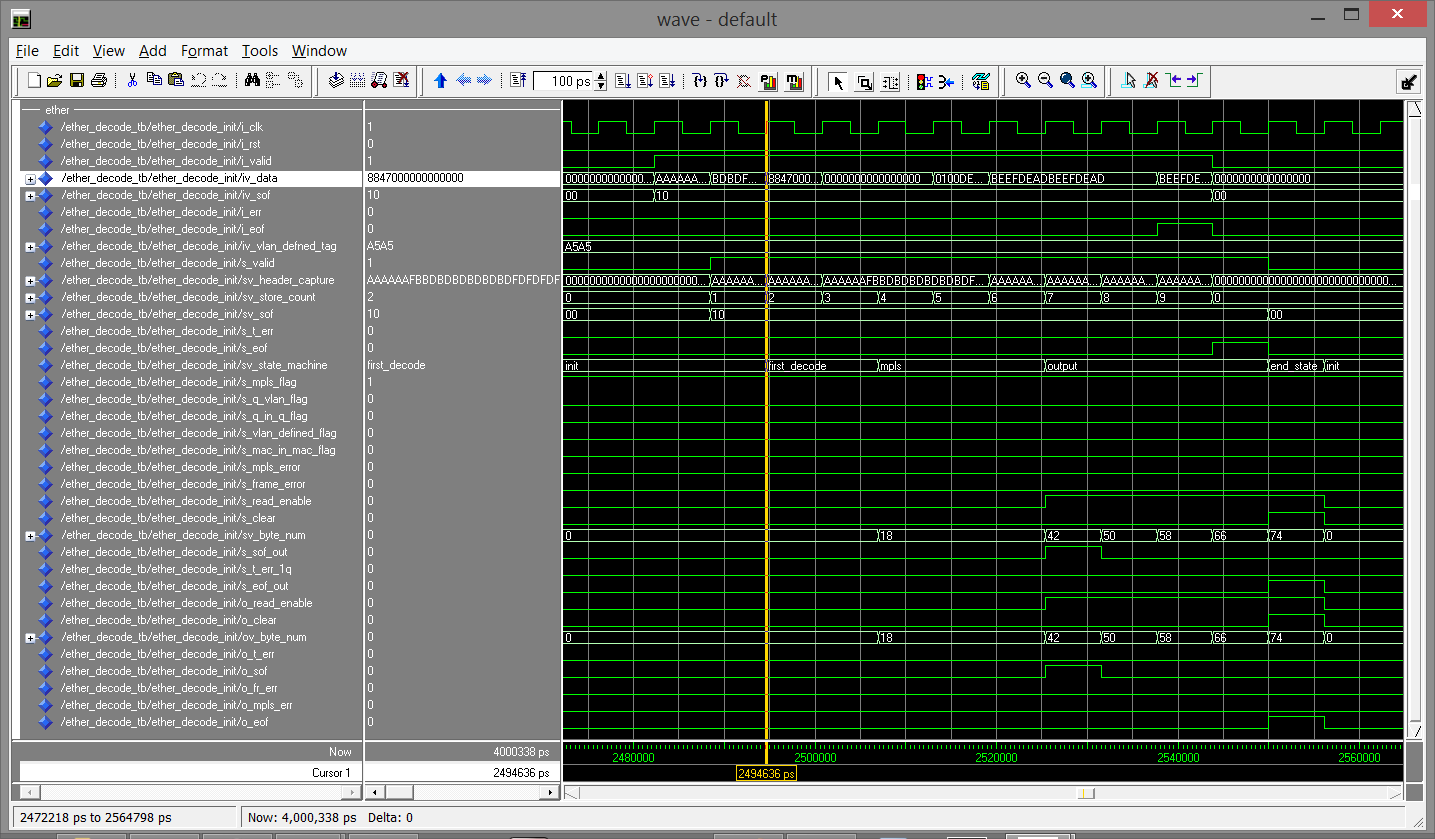
Ether-type FSM Test Case 0, Test Frame 19



Ether-type FSM Test Case 0, Test Frame 20



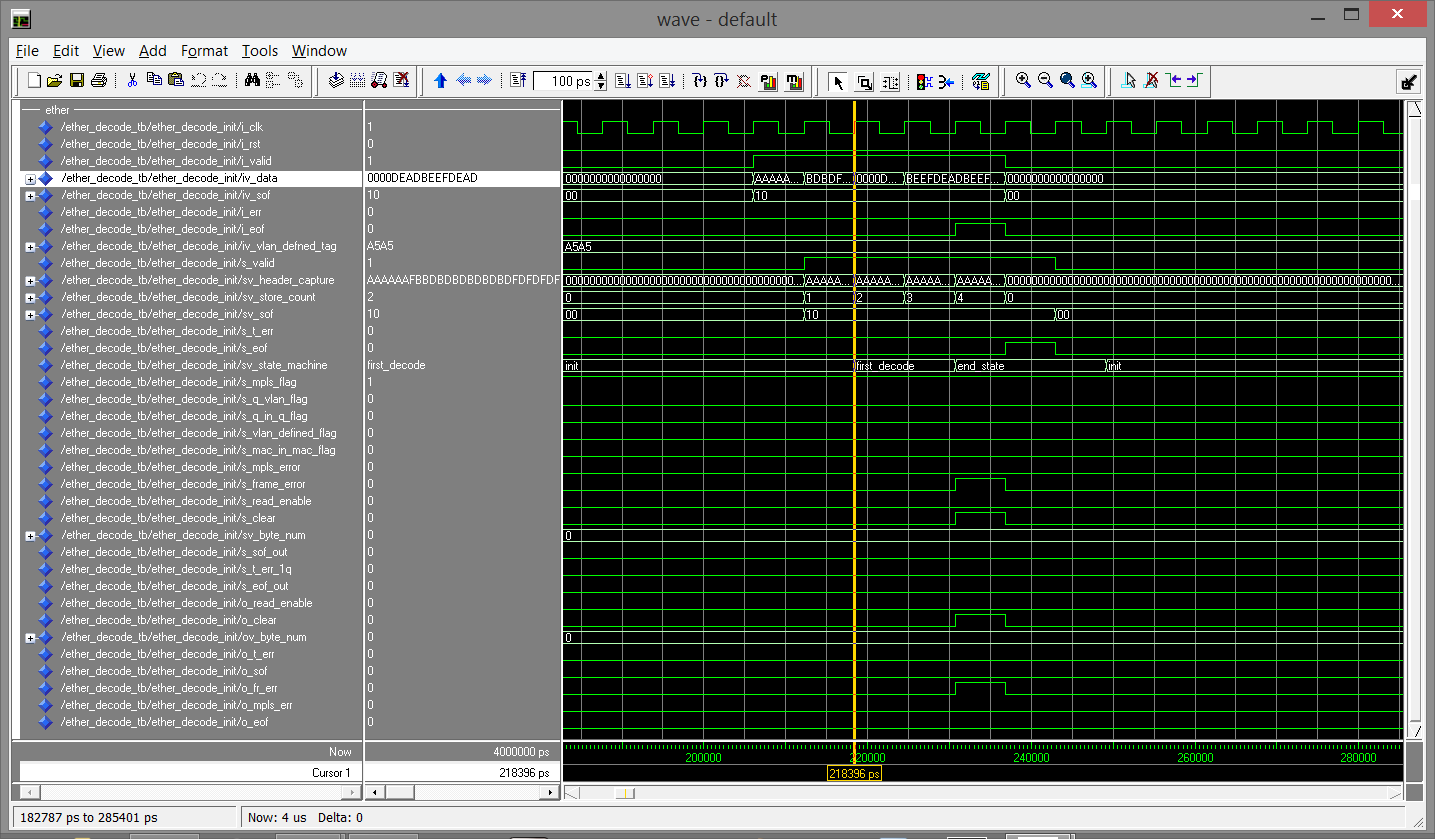
Ether-type FSM Test Case 0, Test Frame 21



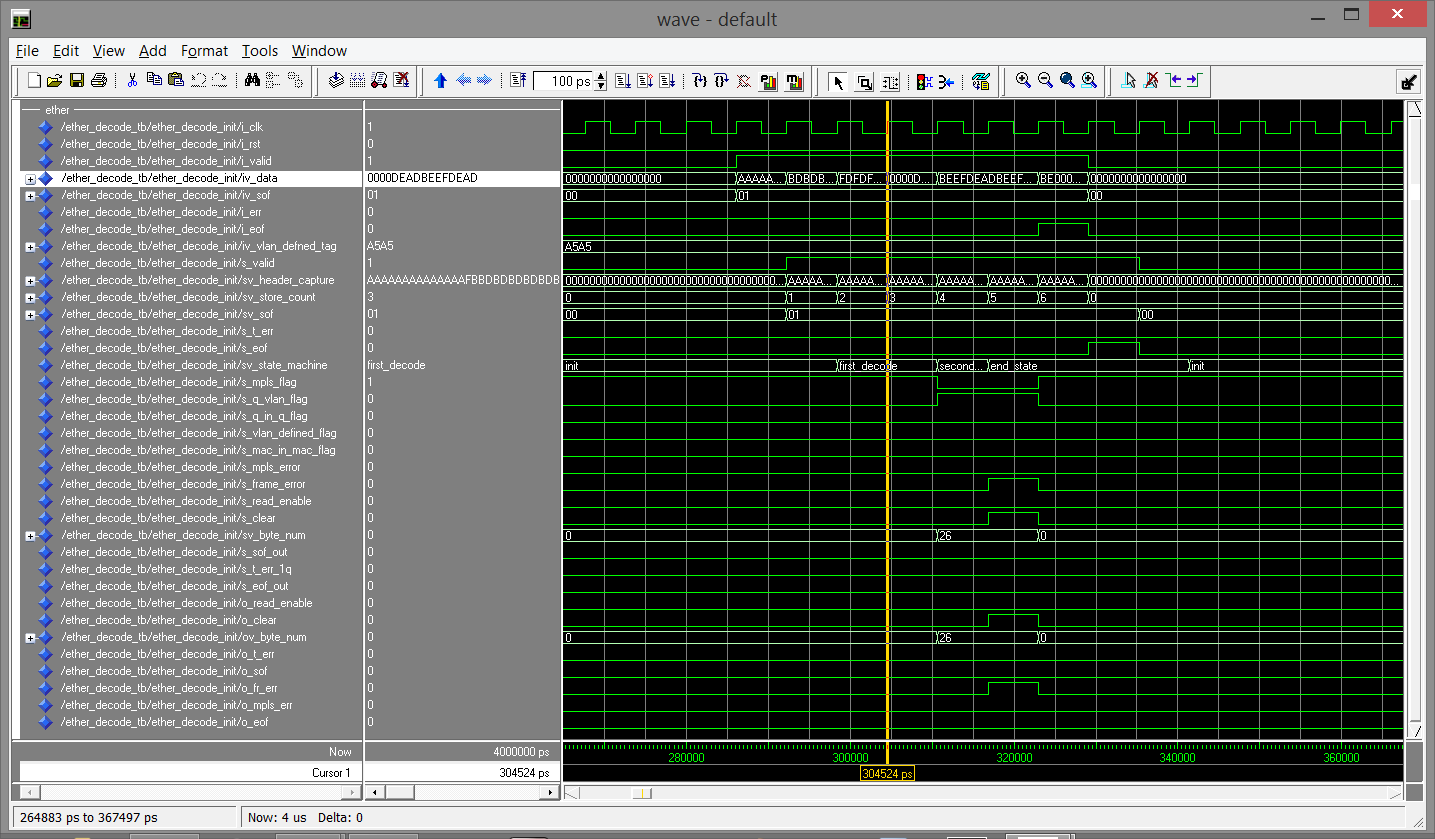
Ether-type FSM Test Case 0, Test Frame 22



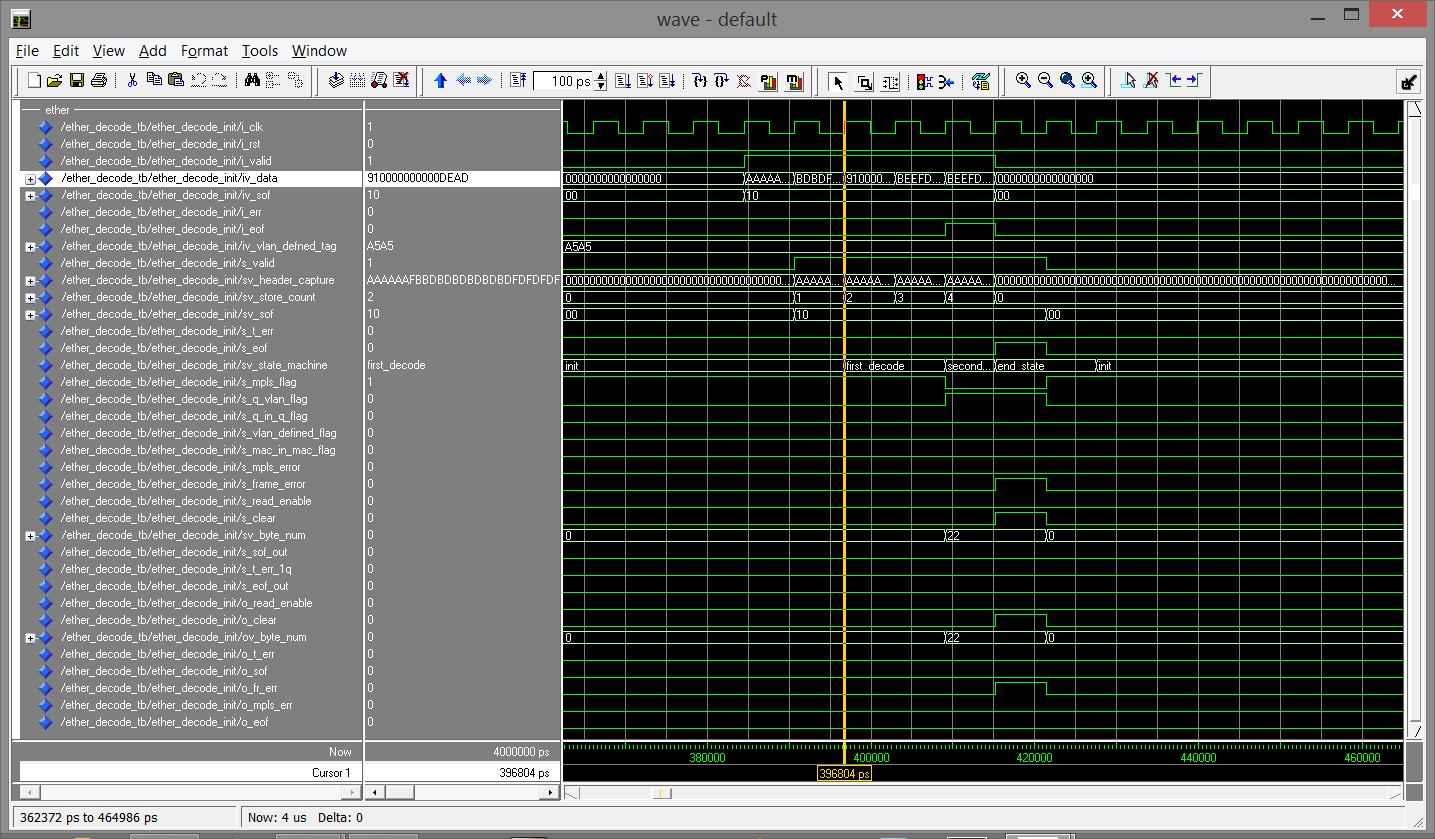
Ether-type FSM Test Case 7, Test Frame 1



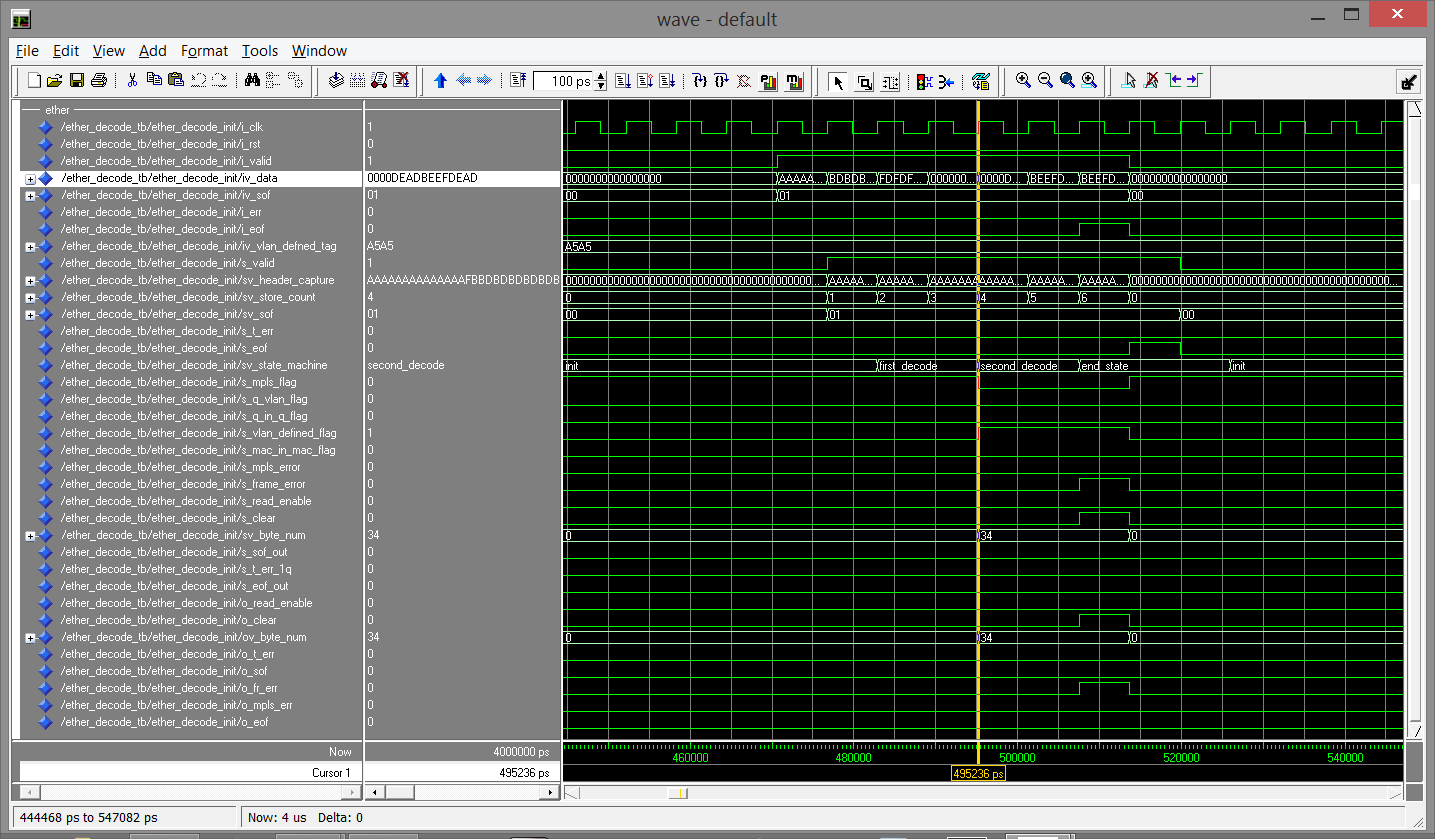
Ether-type FSM Test Case 7, Test Frame 2



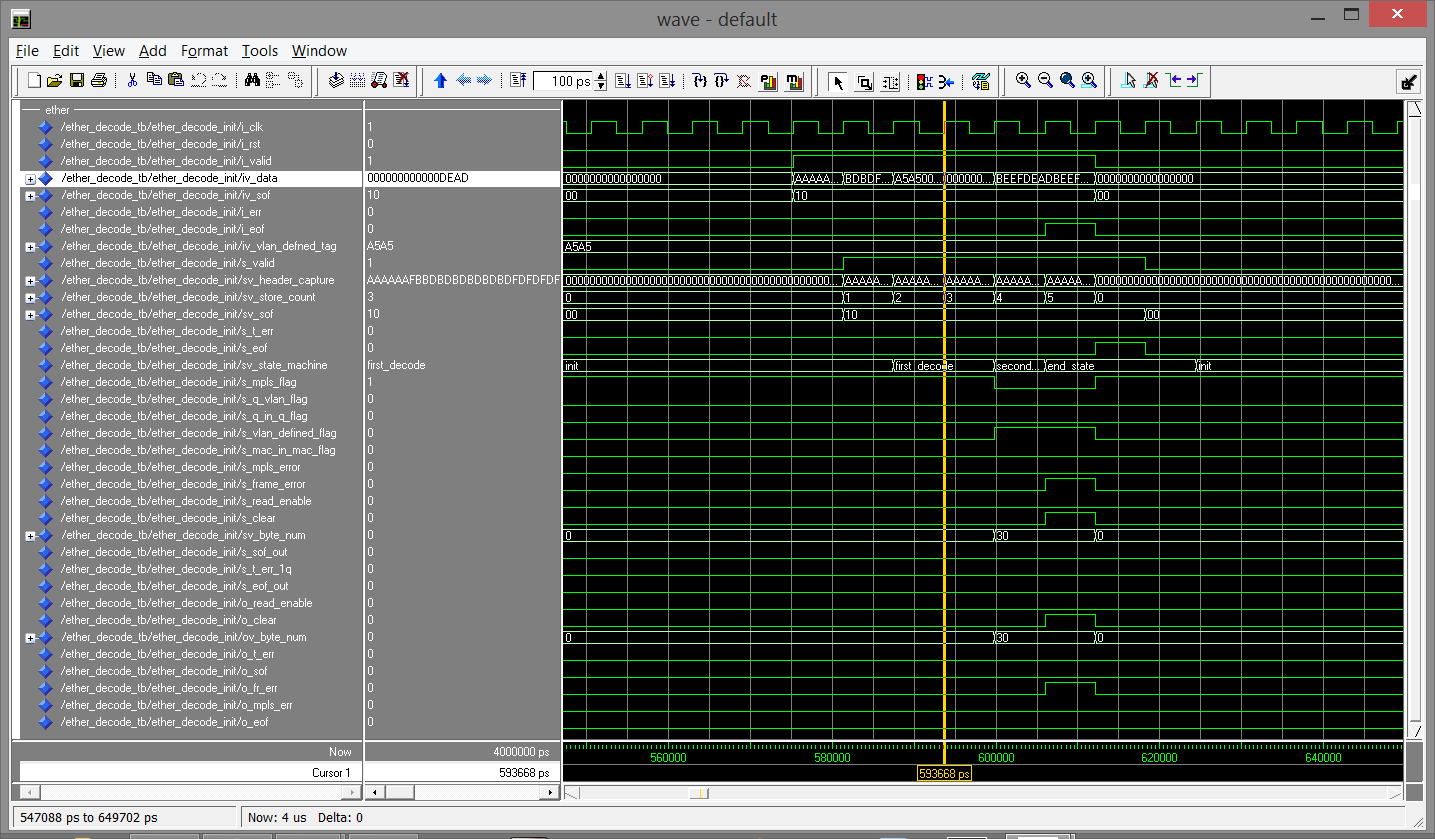
Ether-type FSM Test Case 7, Test Frame 3



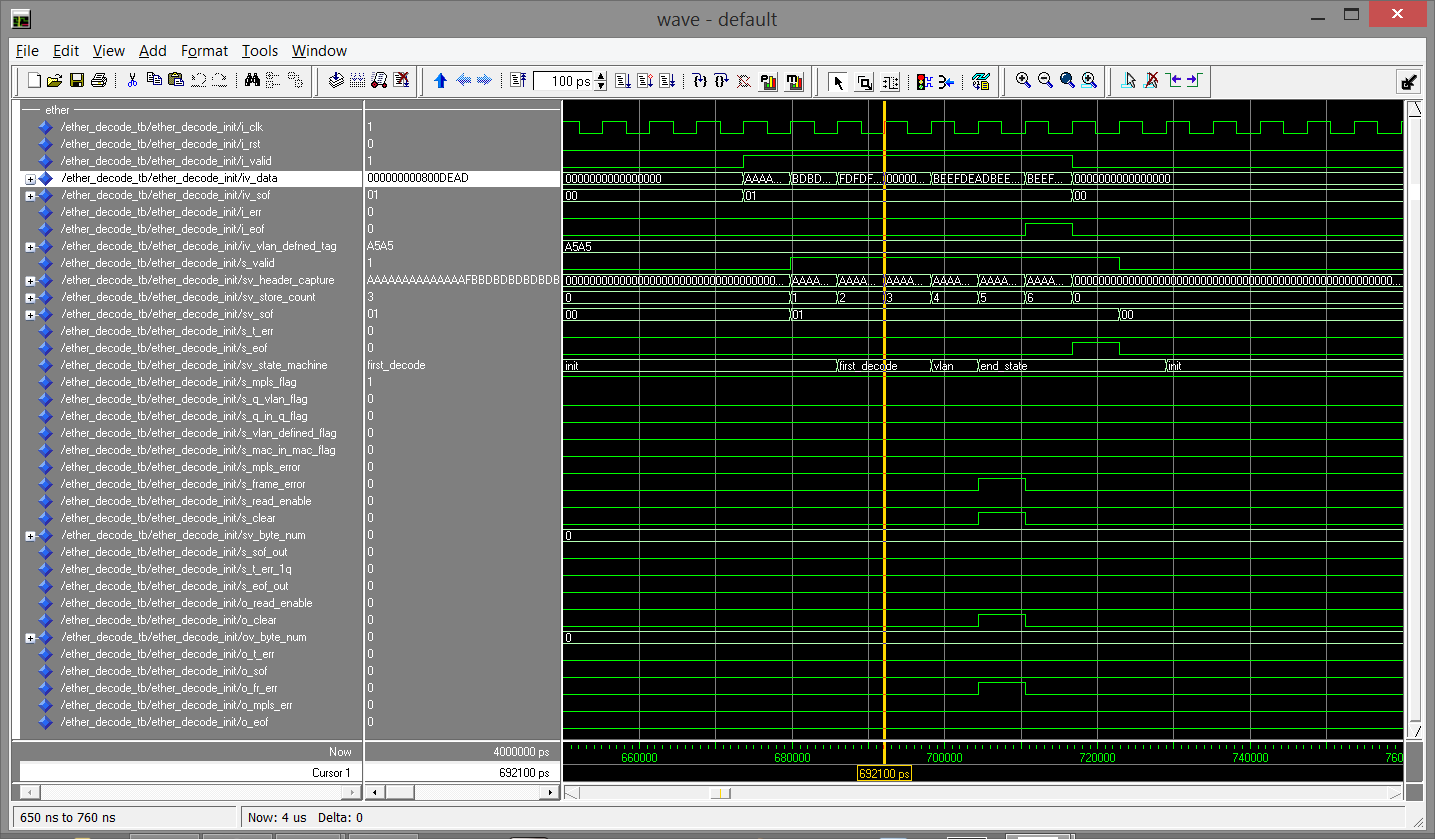
Ether-type FSM Test Case 7, Test Frame 4



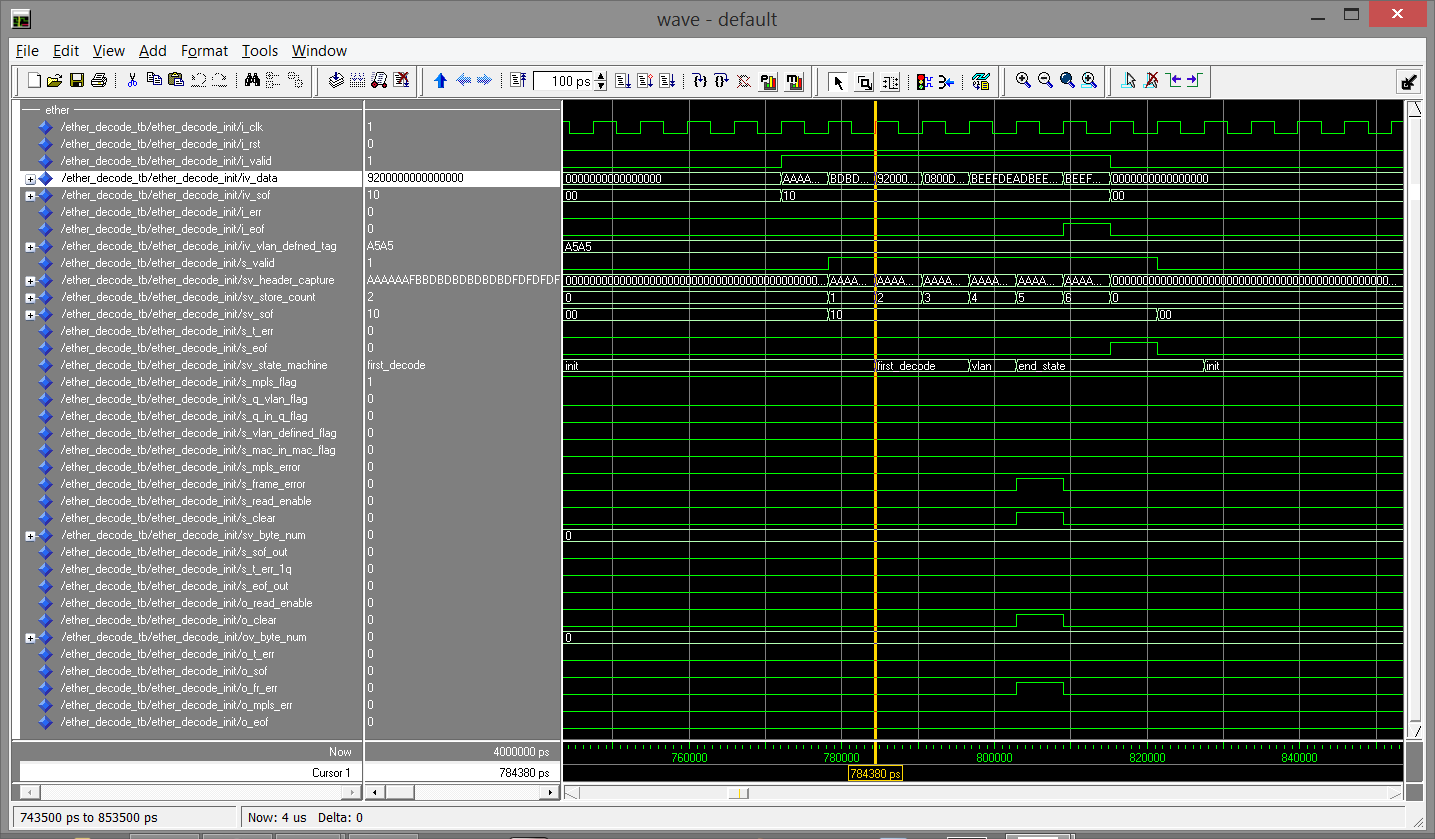
Ether-type FSM Test Case 7, Test Frame 5



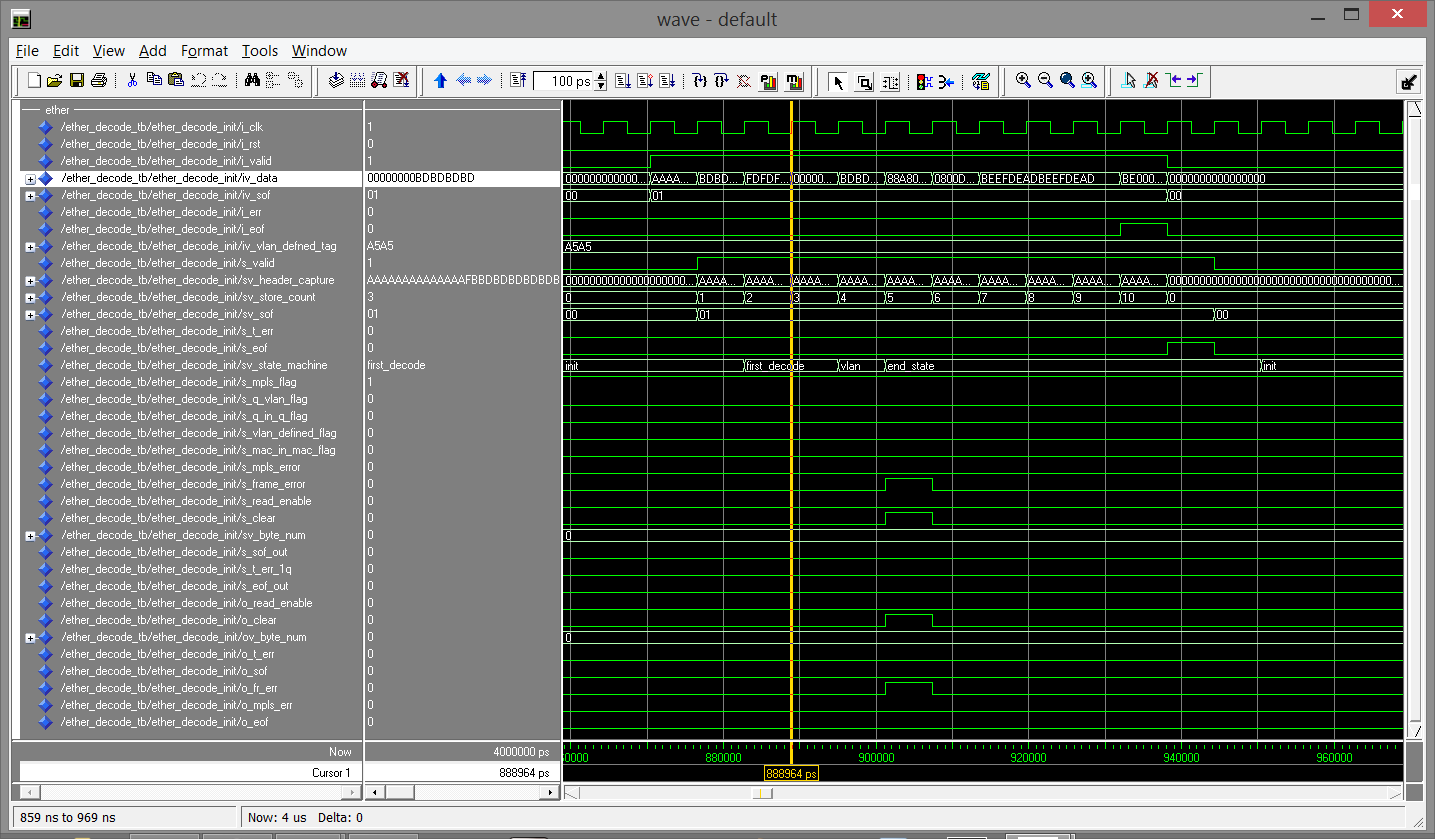
Ether-type FSM Test Case 7, Test Frame 6



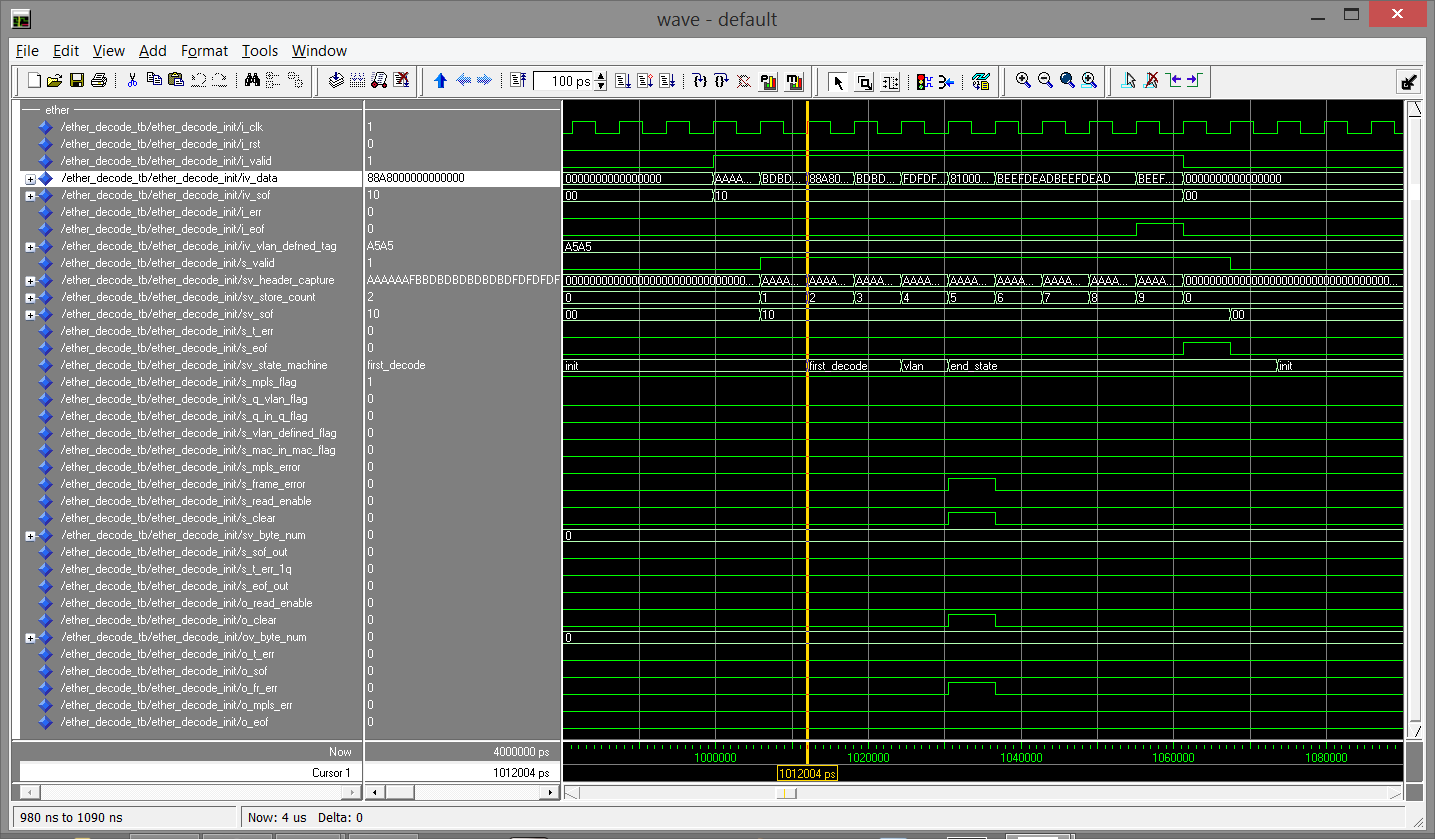
Ether-type FSM Test Case 7, Test Frame 7



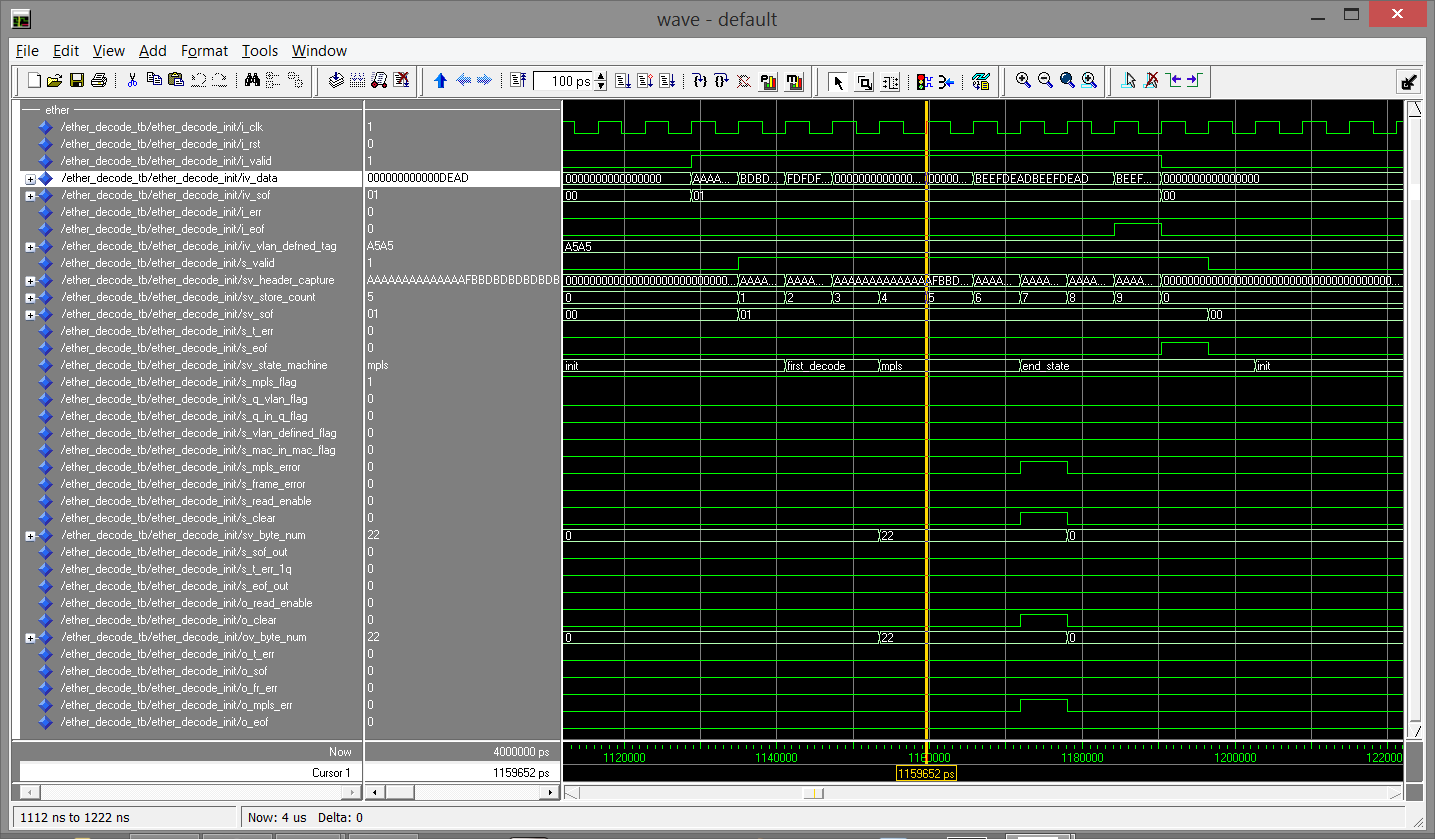
Ether-type FSM Test Case 7, Test Frame 8



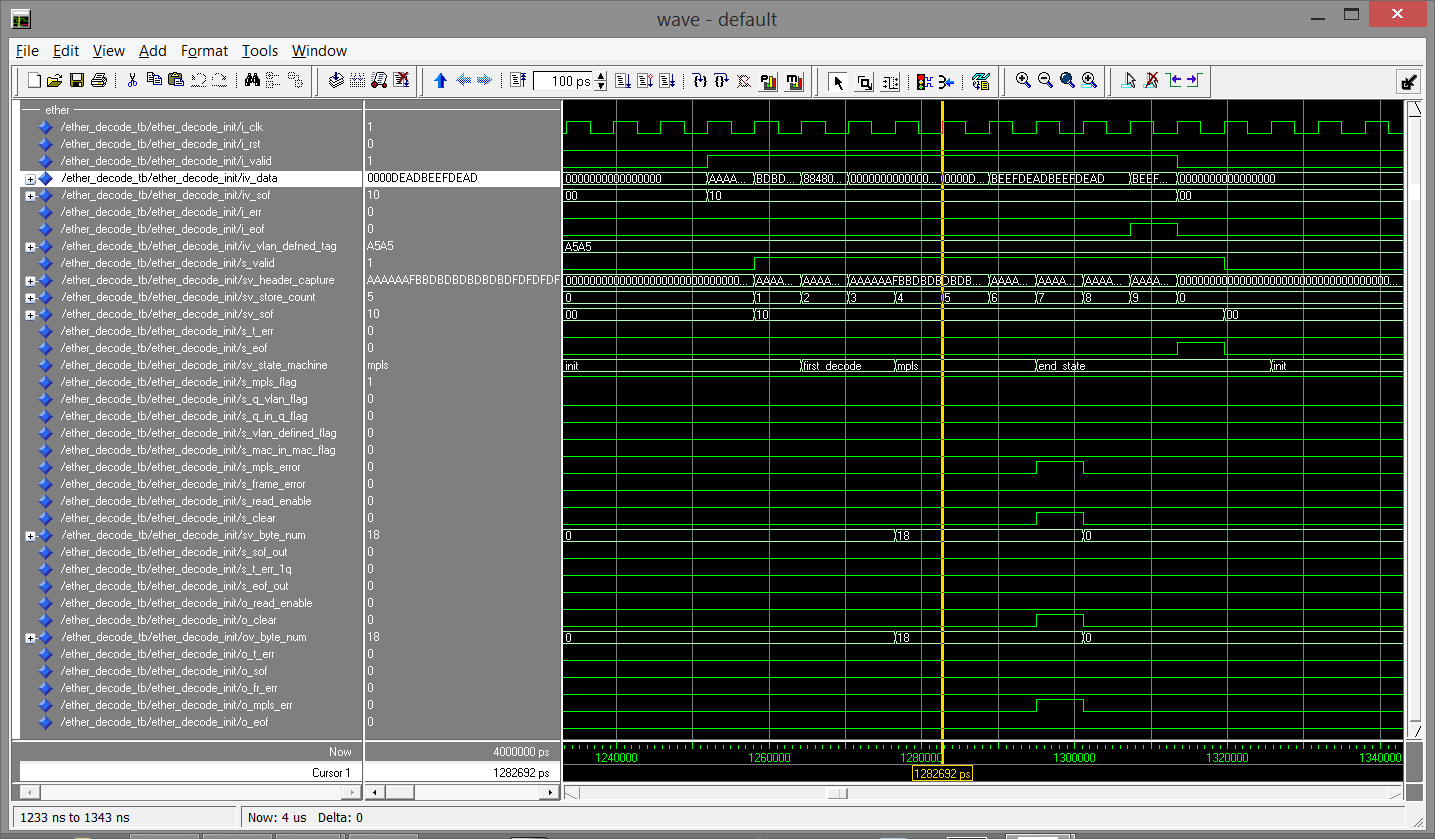
Ether-type FSM Test Case 7, Test Frame 9



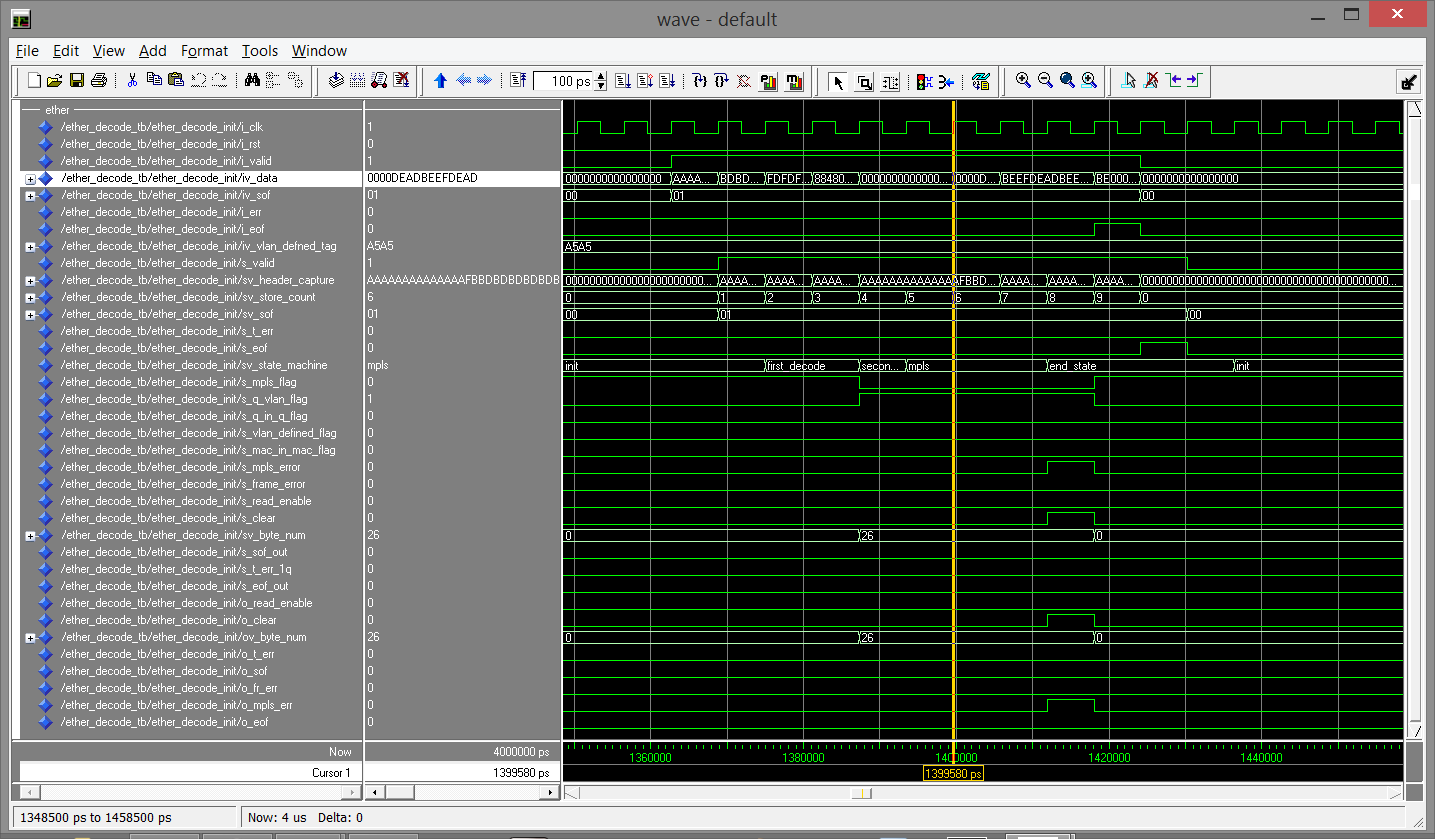
Ether-type FSM Test Case 7, Test Frame 10



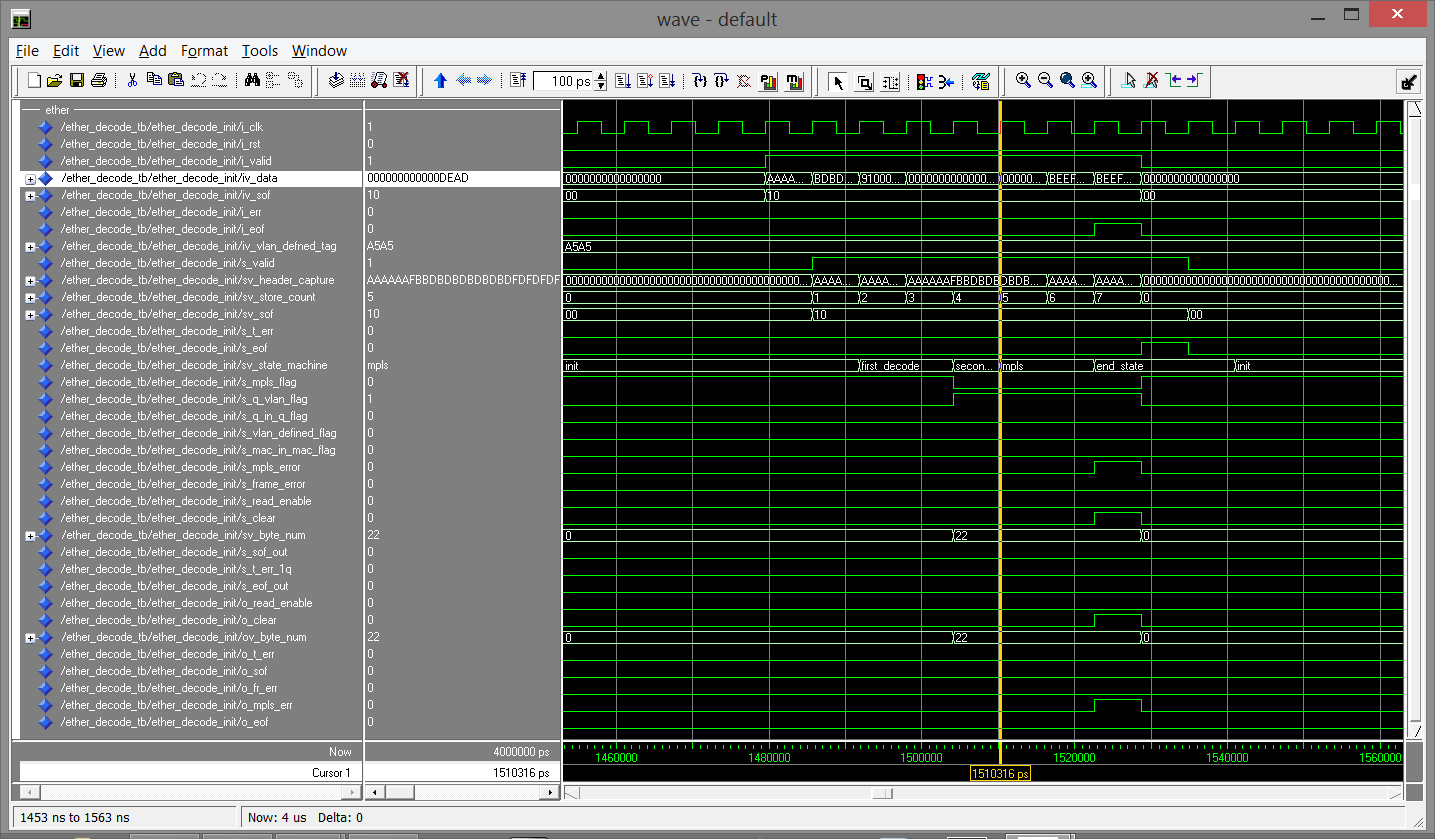
Ether-type FSM Test Case 7, Test Frame 11



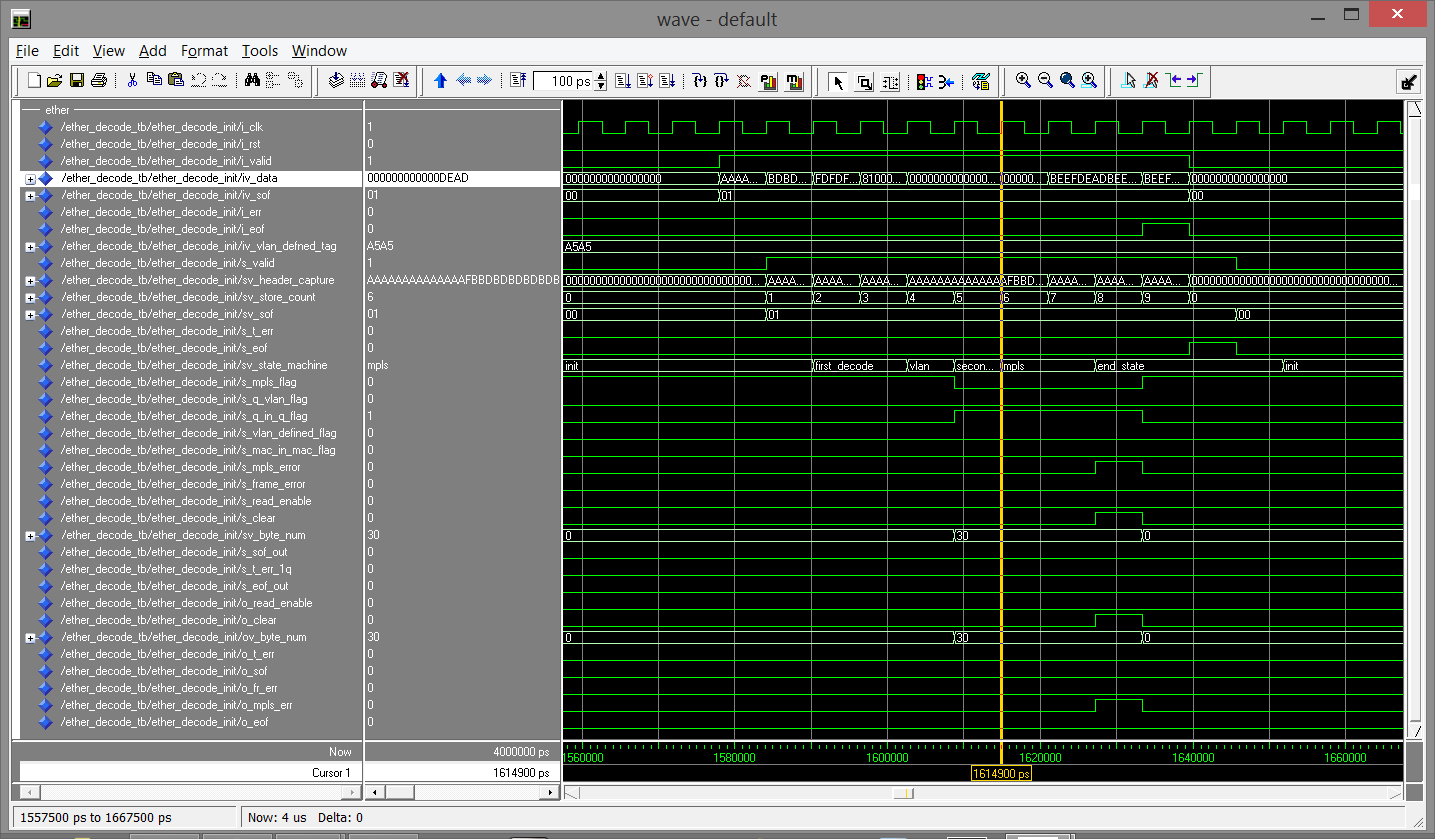
Ether-type FSM Test Case 7, Test Frame 12



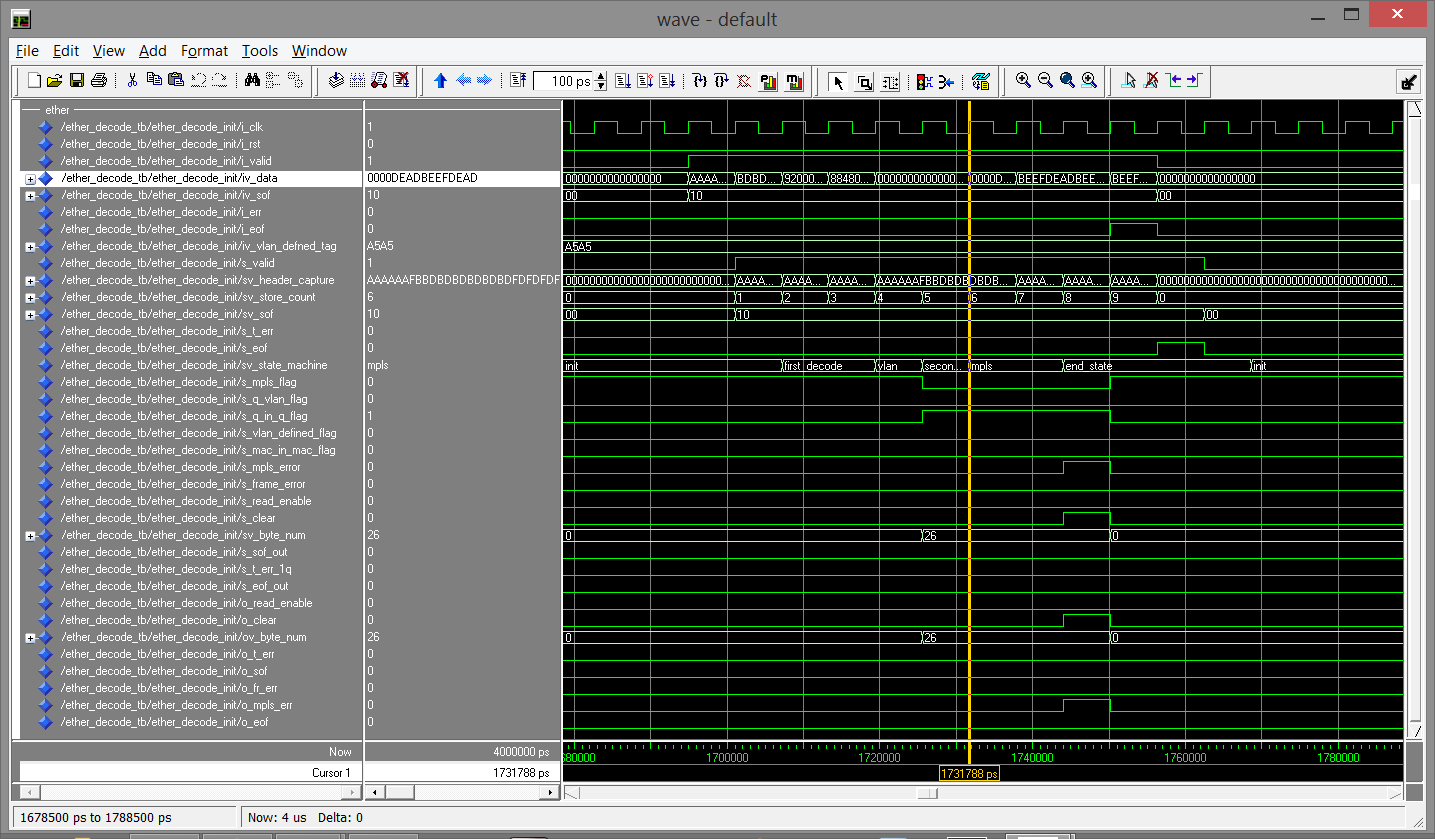
Ether-type FSM Test Case 7, Test Frame 13



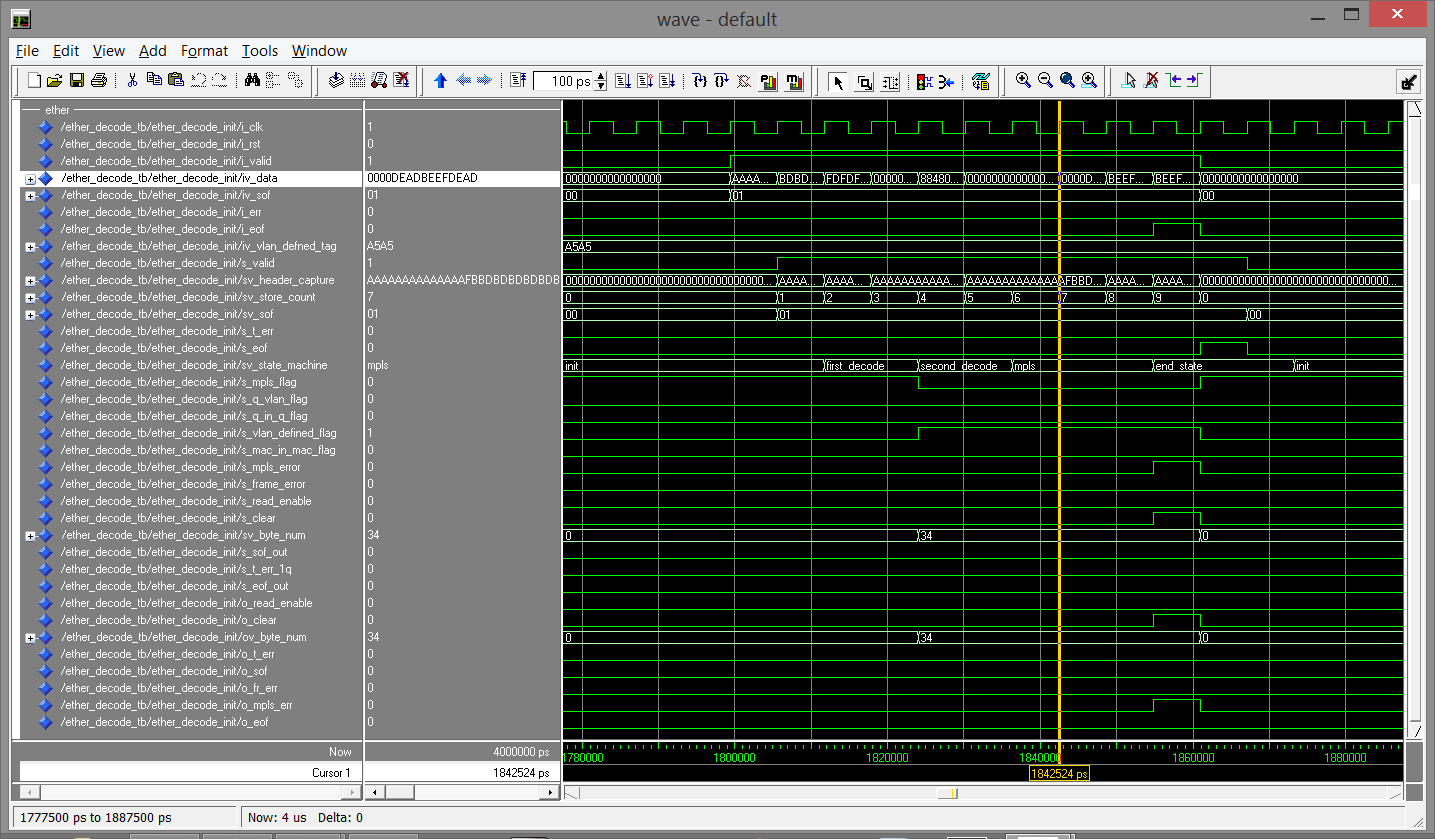
Ether-type FSM Test Case 7, Test Frame 14



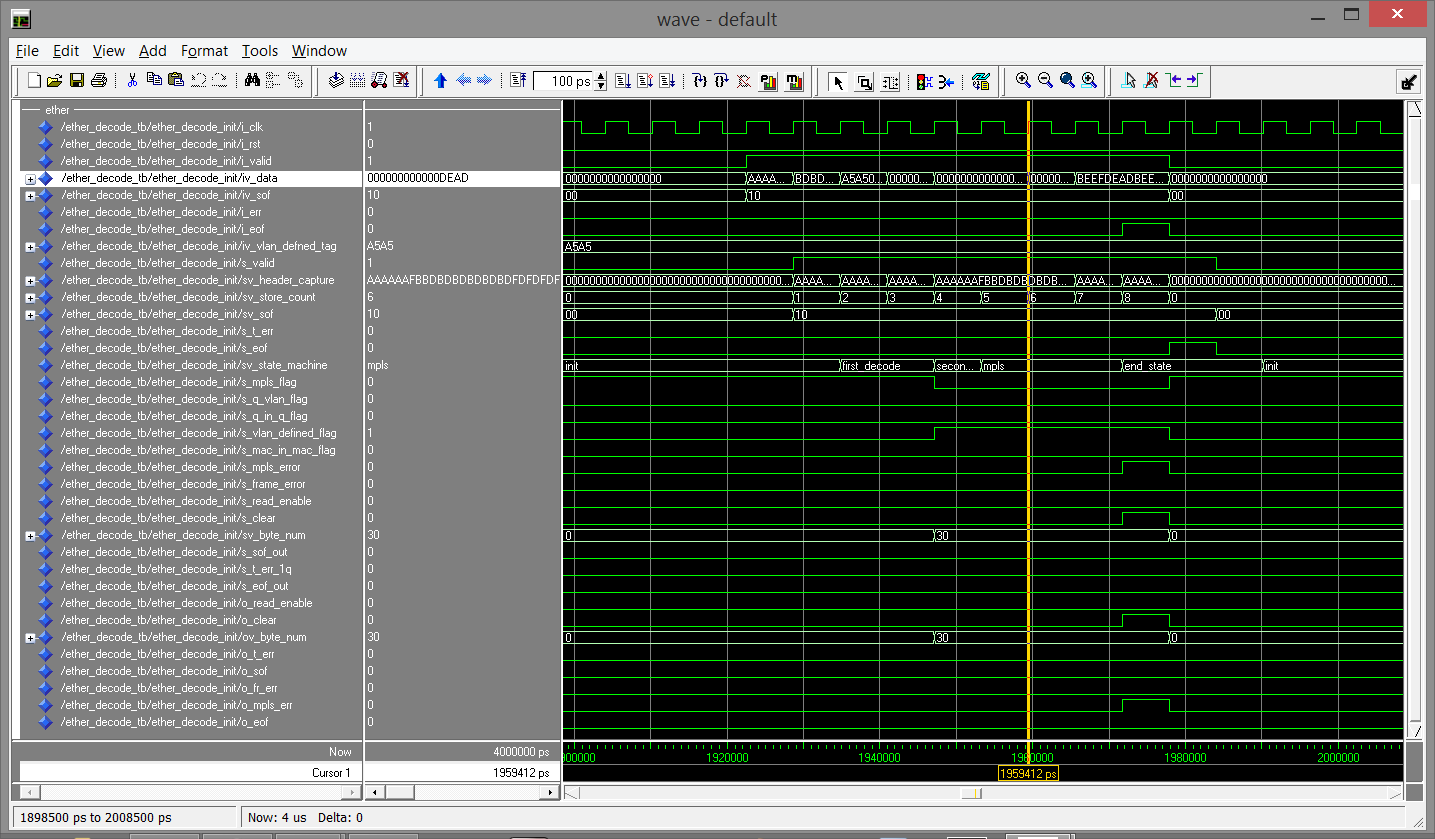
Ether-type FSM Test Case 7, Test Frame 15



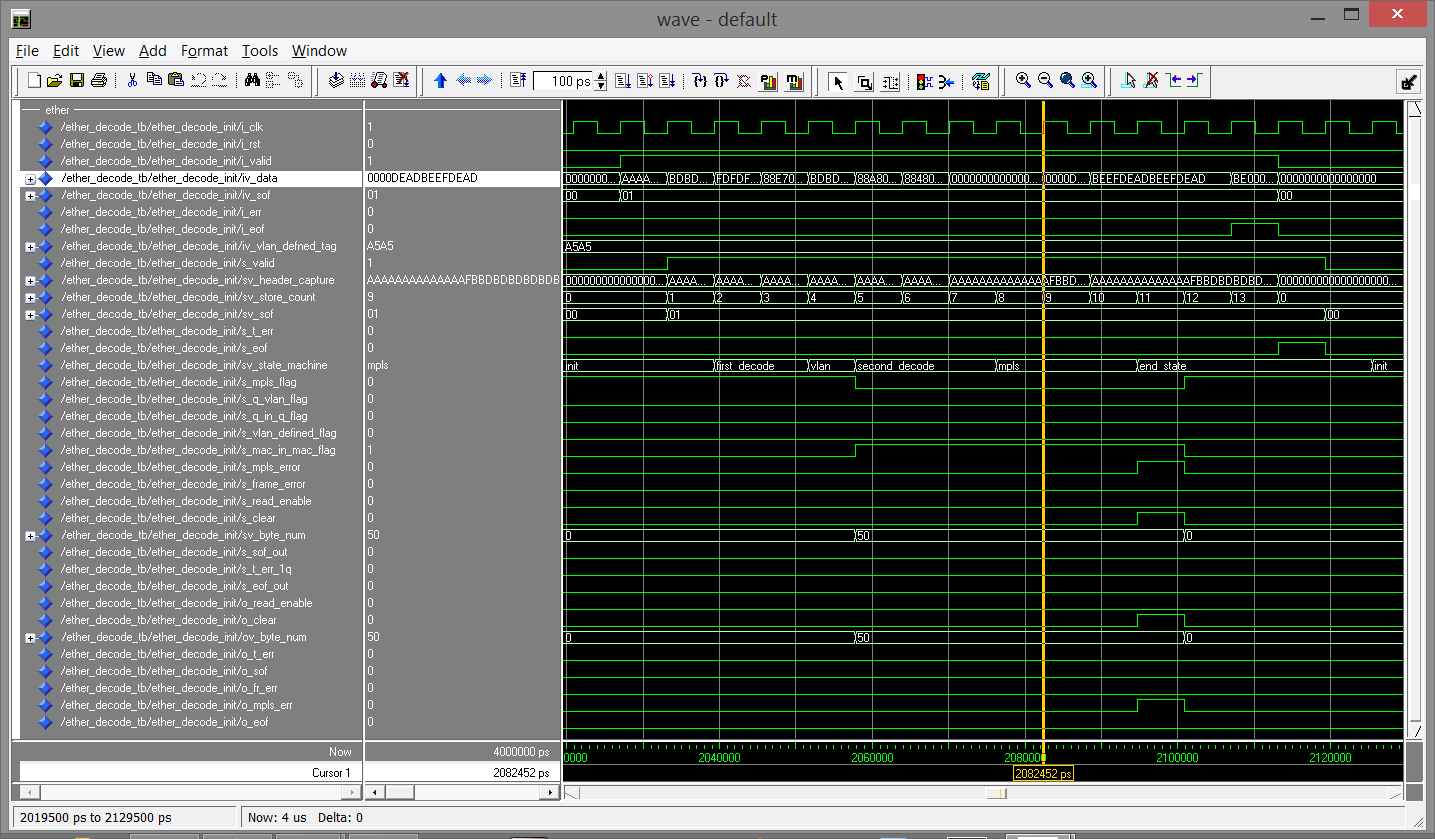
Ether-type FSM Test Case 7, Test Frame 16



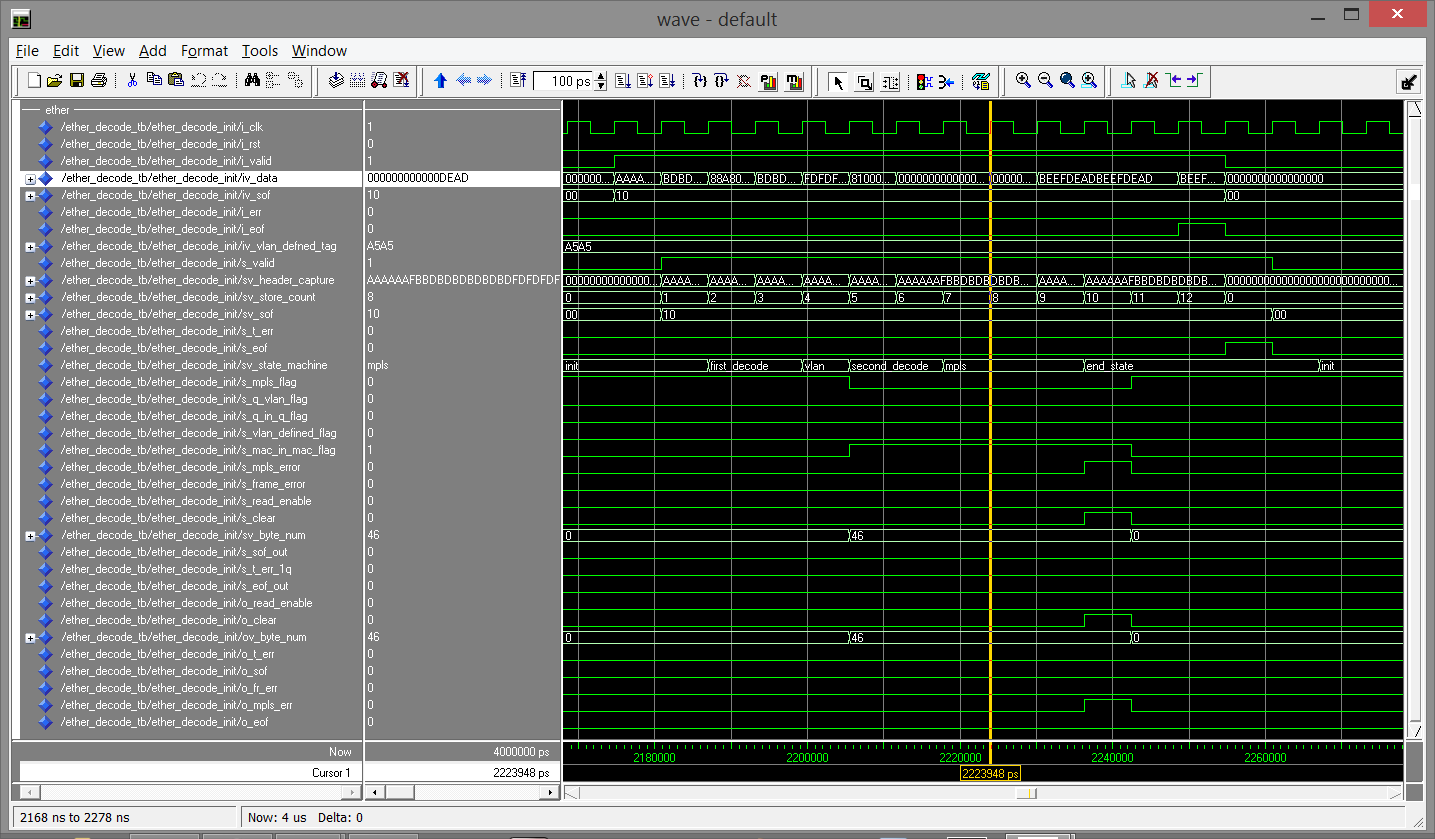
Ether-type FSM Test Case 7, Test Frame 17



Ether-type FSM Test Case 7, Test Frame 18



Ether-type FSM Test Case 7, Test Frame 19

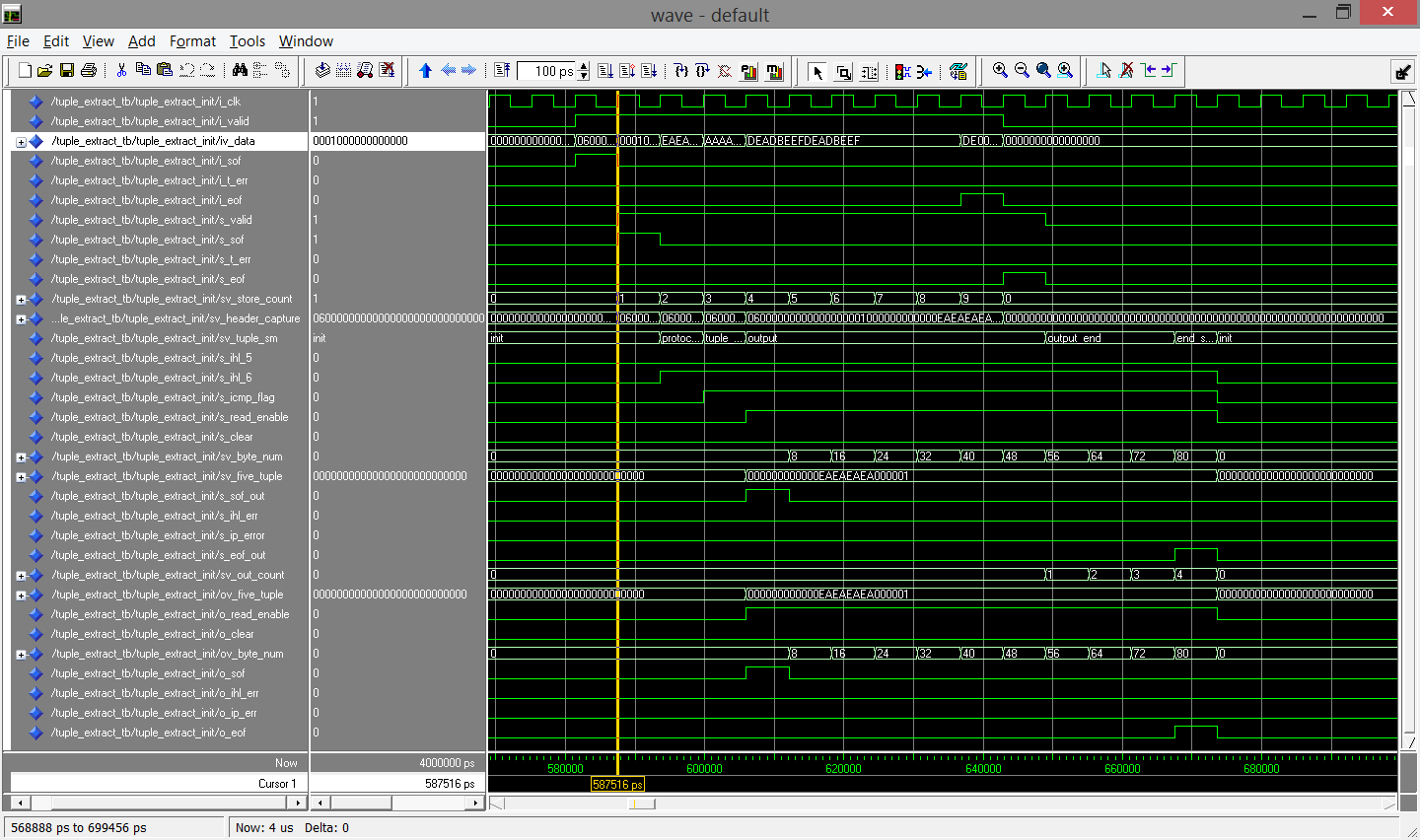


Ether-type FSM Test Case 7, Test Frame 20

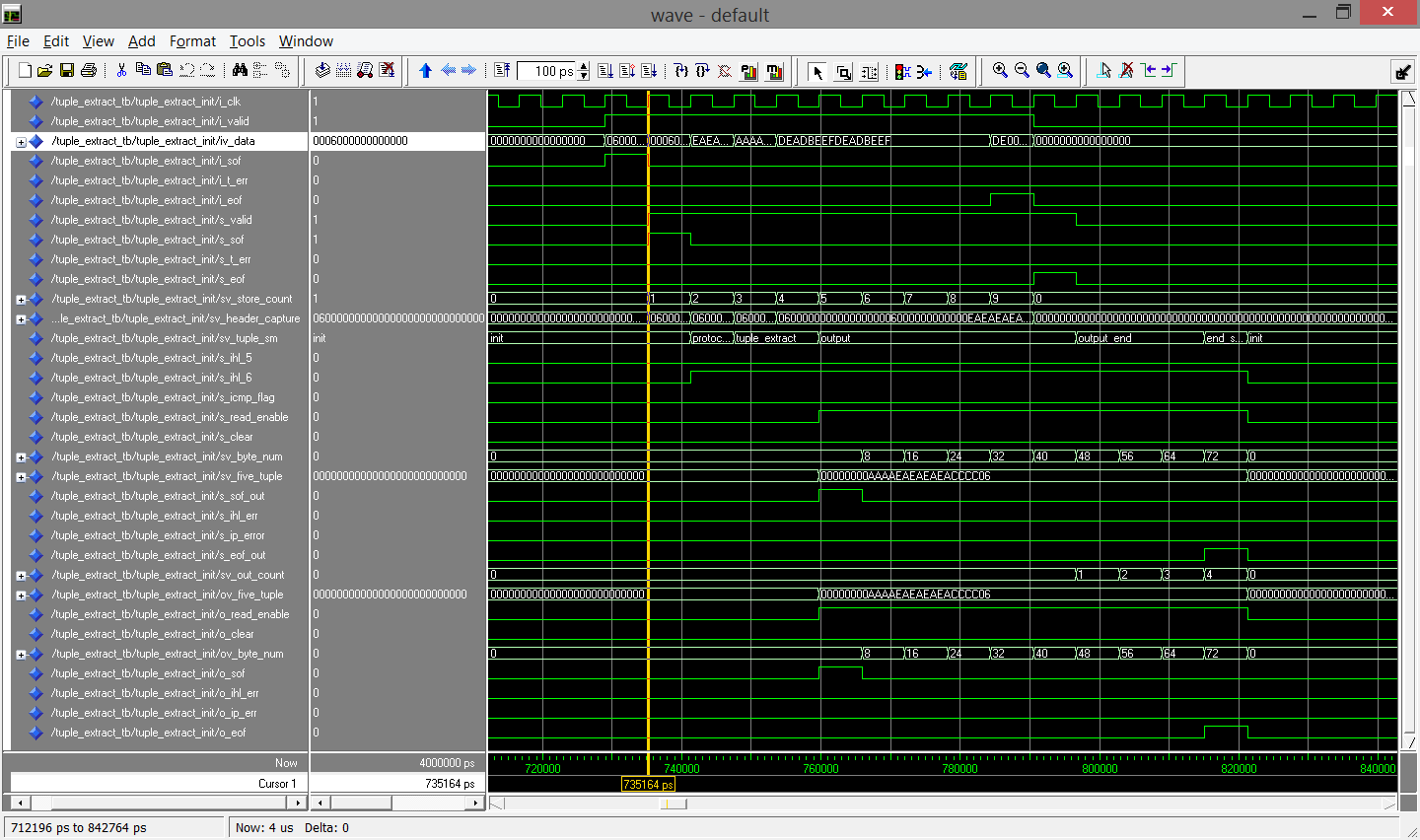
**Appendix C**



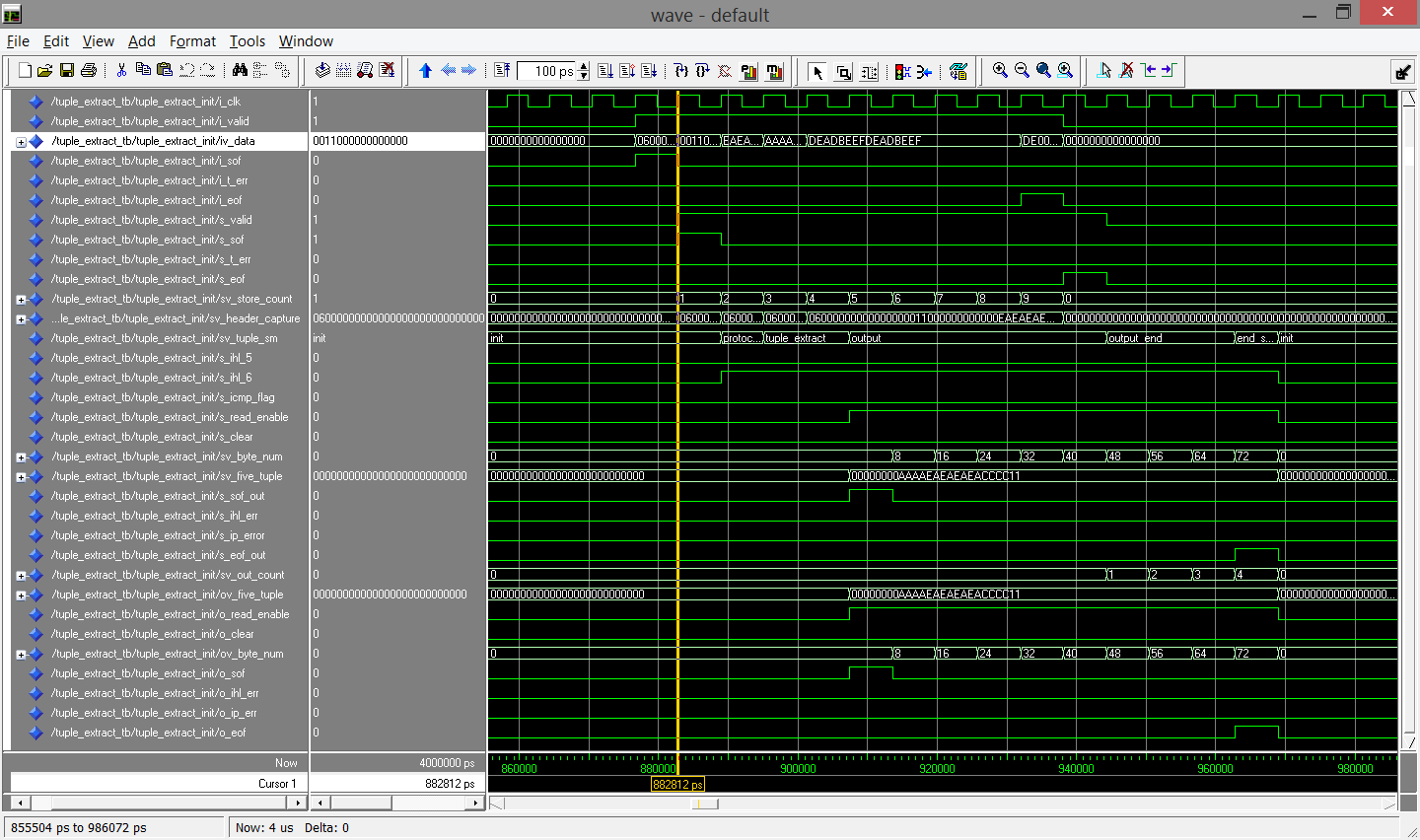
5 Tuple FSM Test Case 0, Test Frame 3



5 Tuple FSM Test Case 0, Test Frame 4



5 Tuple FSM Test Case 0, Test Frames 5



5 Tuple FSM Test Case 0, Test Frame 6

**Appendix D**

**File Pump Verilog Code**

`timescale 1ps/1ps

module file\_pump #(

parameter DATA\_WIDTH = 8 )(

input wire i\_clk,

input wire i\_rst,

input wire i\_enable,

output reg o\_data\_valid,

output reg[DATA\_WIDTH-1:0] ov\_data

);

integer data\_file, read\_file ;

reg[DATA\_WIDTH-1:0] data;

`define NULL 0

`define EMPTY 32'hffff\_ffff

//-----------------------------------------------------------------------

//-- file extraction

//-----------------------------------------------------------------------

initial begin

data\_file = $fopen("input\_config.txt", "r");

if (data\_file == `NULL) begin

$display("data\_file handle was NULL");

$finish;

end

end

always @(posedge i\_clk or negedge i\_rst) begin

if (i\_rst) begin

read\_file <= 0;

o\_data\_valid <= 1'd0;

data <= {DATA\_WIDTH{1'b0}};

ov\_data <= {DATA\_WIDTH{1'b0}};

end else begin

if (i\_enable) begin

read\_file = $fscanf(data\_file, "%x", data);

if (read\_file != `EMPTY) begin

o\_data\_valid <= 1'd1;

ov\_data <= data;

end else begin

o\_data\_valid <= 1'd0;

ov\_data <= {DATA\_WIDTH{1'd0}};

end

end else begin

o\_data\_valid <= 1'd0;

ov\_data <= {DATA\_WIDTH{1'd0}};

end

end

end

endmodule

**Barrel Shifter VHDL Code**

-------------------------------------------------------------------------------

-- barrel\_shifter.vhd

--

-- delays signals for a set number of clock cycles

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use IEEE.numeric\_bit.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

-------------------------------------------------------------------------------

ENTITY barrel\_shifter IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

iv\_byte\_num : IN STD\_LOGIC\_VECTOR (12-1 DOWNTO 0); --input start of frame

i\_read\_enable : IN STD\_LOGIC; --input error signal, used to reset the state machine and barrel shifter

i\_clear\_reg : IN STD\_LOGIC; --input end of frame

o\_valid : OUT STD\_LOGIC; --output read enable to enable reading from the barrel shifter

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0) --output byte number to indicate to the barrel shifter where to read from next

);

END barrel\_shifter;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF barrel\_shifter IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- input\_store\_reg -- MTU is 1526 bytes for a simple IP packet including preammble and SFD

-- -- with PBB VLAN and max MPLS MTU is 1578 bytes

SIGNAL sv\_store\_count : STD\_LOGIC\_VECTOR(6-1 DOWNTO 0);

SIGNAL sv\_frame\_capture : STD\_LOGIC\_VECTOR(64\*198-1 DOWNTO 0); -- 1578 \* 8

-- output\_data\_reg

SIGNAL s\_valid : STD\_LOGIC;

SIGNAL sv\_data\_out : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- capture the input data into the barrel

-------------------------------------------------------------------------------

input\_store\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

sv\_store\_count <= (OTHERS => '0');

sv\_frame\_capture <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

IF (i\_clear\_reg = '1') THEN

sv\_store\_count <= (OTHERS => '0');

sv\_frame\_capture <= (OTHERS => '0');

ELSE

IF (i\_valid = '1') THEN

sv\_store\_count <= sv\_store\_count + 1;

sv\_frame\_capture((((198-CONV\_INTEGER(sv\_store\_count))\*64)-1) DOWNTO (((198-CONV\_INTEGER(sv\_store\_count))-1)\*64)) <= iv\_data;

ELSE

sv\_store\_count <= sv\_store\_count;

sv\_frame\_capture <= sv\_frame\_capture;

END IF;

END IF;

END IF;

END PROCESS input\_store\_reg;

-------------------------------------------------------------------------------

-- output from barrel starting at byte sv\_byte\_num when read enable is high

-------------------------------------------------------------------------------

output\_data\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_valid <= '0';

sv\_data\_out <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_valid <= '0';

sv\_data\_out <= (OTHERS => '0');

IF (i\_read\_enable = '1') THEN

s\_valid <= '1';

sv\_data\_out <= sv\_frame\_capture((((1584 - CONV\_INTEGER(iv\_byte\_num))\*8)-1) DOWNTO ((1576 - CONV\_INTEGER(iv\_byte\_num))\*8));

END IF;

END IF;

END PROCESS output\_data\_reg;

o\_valid <= s\_valid;

ov\_data <= sv\_data\_out;

END ARCHITECTURE rtl;

**Double Data Rate Converter VHDL Code**

-- ddr\_to\_sdr.vhd

--

-- converts a double data rate to a single data rate on the same clock

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

-------------------------------------------------------------------------------

ENTITY ddr\_to\_sdr IS

GENERIC (

DATA\_WIDTH\_IN : INTEGER

);

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (DATA\_WIDTH\_IN-1 DOWNTO 0); --input data

o\_valid : OUT STD\_LOGIC; --output data valid

ov\_data : OUT STD\_LOGIC\_VECTOR ((DATA\_WIDTH\_IN\*2)-1 DOWNTO 0) --output data

);

END ddr\_to\_sdr;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF ddr\_to\_sdr IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- ddr\_to\_sdr\_reg -- signals

SIGNAL s\_single\_rate\_valid : STD\_LOGIC;

SIGNAL sv\_r\_double\_rate\_date : STD\_LOGIC\_VECTOR(DATA\_WIDTH\_IN-1 DOWNTO 0);

SIGNAL sv\_f\_double\_rate\_date : STD\_LOGIC\_VECTOR(DATA\_WIDTH\_IN-1 DOWNTO 0);

SIGNAL sv\_single\_rate\_date : STD\_LOGIC\_VECTOR((DATA\_WIDTH\_IN\*2)-1 DOWNTO 0);

SIGNAL s\_test : STD\_LOGIC;

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- register data on both edges of the clock and then re-register on then rising edge

-------------------------------------------------------------------------------

s\_test <= i\_valid xor i\_clk;

ddr\_to\_sdr\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

s\_single\_rate\_valid <= '0';

o\_valid <= '0';

sv\_r\_double\_rate\_date <= (OTHERS => '0');

sv\_f\_double\_rate\_date <= (OTHERS => '0');

sv\_single\_rate\_date <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN --rising

s\_single\_rate\_valid <= i\_valid ;

o\_valid <= s\_single\_rate\_valid;

sv\_single\_rate\_date <= sv\_r\_double\_rate\_date & sv\_f\_double\_rate\_date;

IF ( i\_valid = '1') THEN

sv\_r\_double\_rate\_date <= iv\_data;

END IF;

ELSIF (i\_clk'event and i\_clk = '0') THEN --falling

IF ( i\_valid = '1') THEN

sv\_f\_double\_rate\_date <= iv\_data;

END IF;

END IF;

END PROCESS ddr\_to\_sdr\_reg;

ov\_data <= sv\_single\_rate\_date;

END ARCHITECTURE rtl;

--------------------------------------------------------------

**XGMII FCS Removal FSM VHDL Code**

-------------------------------------------------------------------------------

--

-- xgmii\_decoder.vhd

--

-- converts a double data rate to a single data rate on the same clock

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use work.component\_package.all;

-------------------------------------------------------------------------------

ENTITY xgmii\_decoder IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_cntrl : IN STD\_LOGIC\_VECTOR (4-1 DOWNTO 0); --input xgmii control

iv\_data : IN STD\_LOGIC\_VECTOR (32-1 DOWNTO 0); --input data

o\_valid : OUT STD\_LOGIC; --output data valid

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --output data

ov\_sof : OUT STD\_LOGIC\_VECTOR (2-1 DOWNTO 0); --output start of frame

o\_err : OUT STD\_LOGIC; --output frame error

o\_eof : OUT STD\_LOGIC --output end of frame

);

END xgmii\_decoder;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF xgmii\_decoder IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

TYPE XGMII\_SM\_TYPE IS (INIT, IN\_FRAME, END\_FRAME);

TYPE FCS\_REMOVAL\_SM\_TYPE IS (INIT\_EOF, SECOND\_EOF, REMOVEL);

-- ddr\_to\_sdr\_reg -- signals

SIGNAL s\_valid\_in : STD\_LOGIC;

SIGNAL sv\_data\_in : STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); -- double input data

SIGNAL sv\_cntrl\_in : STD\_LOGIC\_VECTOR (8-1 DOWNTO 0); -- double input xgmii control

SIGNAL sv\_xgmii\_statemachine : XGMII\_SM\_TYPE;

SIGNAL s\_valid\_out : STD\_LOGIC;

SIGNAL sv\_data\_out : STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); -- output data

SIGNAL sv\_cntrl\_out : STD\_LOGIC\_VECTOR (8-1 DOWNTO 0); -- xgmii control used to remove FCS

SIGNAL sv\_sof : STD\_LOGIC\_VECTOR (2-1 DOWNTO 0);

SIGNAL s\_err : STD\_LOGIC;

SIGNAL s\_eof : STD\_LOGIC;

-- delay\_sigals\_reg -- signals

SIGNAL s\_valid\_1q : STD\_LOGIC;

SIGNAL sv\_data\_1q : STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); -- output data

SIGNAL sv\_cntrl\_1q : STD\_LOGIC\_VECTOR (8-1 DOWNTO 0); -- xgmii control used to remove FCS

SIGNAL sv\_sof\_1q : STD\_LOGIC\_VECTOR (2-1 DOWNTO 0);

SIGNAL s\_err\_1q : STD\_LOGIC;

SIGNAL s\_eof\_1q : STD\_LOGIC;

-- fcs\_removal\_regg -- signals

SIGNAL sv\_fcs\_removal\_sm : FCS\_REMOVAL\_SM\_TYPE;

SIGNAL s\_valid\_2q : STD\_LOGIC;

SIGNAL sv\_data\_2q : STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); -- output data

SIGNAL sv\_sof\_2q : STD\_LOGIC\_VECTOR (2-1 DOWNTO 0);

SIGNAL s\_err\_2q : STD\_LOGIC;

SIGNAL s\_eof\_2q : STD\_LOGIC;

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

ddr\_to\_sdr\_data\_inst : ddr\_to\_sdr

GENERIC MAP(

DATA\_WIDTH\_IN => 32

)

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => i\_valid,

iv\_data => iv\_data,

o\_valid => s\_valid\_in,

ov\_data => sv\_data\_in

);

ddr\_to\_sdr\_control\_inst : ddr\_to\_sdr

GENERIC MAP(

DATA\_WIDTH\_IN => 4

)

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => i\_valid,

iv\_data => iv\_cntrl,

o\_valid => OPEN,

ov\_data => sv\_cntrl\_in

);

-------------------------------------------------------------------------------

--

-------------------------------------------------------------------------------

xgmii\_decode\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

sv\_sof <= "00";

s\_eof <= '0';

s\_err <= '0';

s\_valid\_out <= '0';

sv\_cntrl\_out <= (OTHERS => '0');

sv\_data\_out <= (OTHERS => '0');

sv\_xgmii\_statemachine <= INIT;

ELSIF (i\_clk'event and i\_clk = '1') THEN

CASE sv\_xgmii\_statemachine IS

WHEN INIT =>

s\_err <= '0';

s\_valid\_out <= '0';

sv\_data\_out <= (OTHERS => '0');

IF ( s\_valid\_in = '1') THEN

IF (sv\_cntrl\_in(0) = '1') THEN

IF (sv\_data\_in(7 DOWNTO 0) = x"FE")THEN -- error in position 0

s\_err <= '1';

ELSIF (sv\_data\_in(7 DOWNTO 0) = x"FB")THEN -- start of frame in position 0

sv\_sof <= "01";

s\_valid\_out <= s\_valid\_in;

sv\_data\_out <= sv\_data\_in;

sv\_xgmii\_statemachine <= IN\_FRAME;

END IF;

END IF;

IF (sv\_cntrl\_in(1) = '1') THEN

IF (sv\_data\_in(15 DOWNTO 8) = x"FE")THEN -- error in position 1

s\_err <= '1';

END IF;

END IF;

IF (sv\_cntrl\_in(2) = '1') THEN

IF (sv\_data\_in(23 DOWNTO 16) = x"FE")THEN -- error in position 2

s\_err <= '1';

END IF;

END IF;

IF (sv\_cntrl\_in(3) = '1') THEN

IF (sv\_data\_in(31 DOWNTO 24) = x"FE")THEN -- error in position 3

s\_err <= '1';

END IF;

END IF;

IF (sv\_cntrl\_in(4) = '1') THEN

IF (sv\_data\_in(39 DOWNTO 32) = x"FE")THEN -- error in position 4

s\_err <= '1';

ELSIF (sv\_data\_in(39 DOWNTO 32) = x"FB")THEN -- start of frame in position 4

sv\_sof <= "10";

s\_valid\_out <= s\_valid\_in;

sv\_data\_out <= sv\_data\_in;

sv\_xgmii\_statemachine <= IN\_FRAME;

END IF;

END IF;

IF (sv\_cntrl\_in(5) = '1') THEN -- start of frame in position 5

IF (sv\_data\_in(47 DOWNTO 40) = x"FE")THEN -- error in position 0

s\_err <= '1';

END IF;

END IF;

IF (sv\_cntrl\_in(6) = '1') THEN -- start of frame in position 6

IF (sv\_data\_in(55 DOWNTO 48) = x"FE")THEN -- error in position 0

s\_err <= '1';

END IF;

END IF;

IF (sv\_cntrl\_in(7) = '1') THEN -- start of frame in position 7

IF (sv\_data\_in(63 DOWNTO 56) = x"FE")THEN -- error in position 0

s\_err <= '1';

END IF;

END IF;

END IF;

WHEN IN\_FRAME =>

s\_valid\_out <= s\_valid\_in;

sv\_cntrl\_out <= sv\_cntrl\_in;

IF (s\_err = '1') THEN

sv\_xgmii\_statemachine <= END\_FRAME;

ELSE

-------------------------------------------------------------------------------

-- control bit 0

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(0) = '1') THEN

IF (sv\_data\_in(7 DOWNTO 0) = x"FE")THEN -- error in position 0

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(7 DOWNTO 0) = x"FD")THEN -- end of frame in position 0

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= sv\_data\_in(63 DOWNTO 8) & x"00";

END IF;

ELSE

sv\_data\_out <= sv\_data\_in;

END IF;

-------------------------------------------------------------------------------

-- control bit 1

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(1) = '1') THEN

IF (sv\_data\_in(15 DOWNTO 8) = x"FE")THEN -- error in position 1

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(15 DOWNTO 8) = x"FD")THEN -- end of frame in position 1

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= sv\_data\_in(63 DOWNTO 16) & x"00" & sv\_data\_in(7 DOWNTO 0);

END IF;

ELSE

sv\_data\_out <= sv\_data\_in;

END IF;

-------------------------------------------------------------------------------

-- control bit 2

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(2) = '1') THEN

IF (sv\_data\_in(23 DOWNTO 16) = x"FE")THEN -- error in position 2

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(23 DOWNTO 16) = x"FD")THEN -- end of frame in position 2

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= sv\_data\_in(63 DOWNTO 24) & x"00" & sv\_data\_in(15 DOWNTO 0);

END IF;

ELSE

sv\_data\_out <= sv\_data\_in;

END IF;

-------------------------------------------------------------------------------

-- control bit 3

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(3) = '1') THEN

IF (sv\_data\_in(31 DOWNTO 24) = x"FE")THEN -- error in position 3

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(31 DOWNTO 24) = x"FD")THEN -- end of frame in position 3

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= sv\_data\_in(63 DOWNTO 32) & x"00" & sv\_data\_in(23 DOWNTO 0);

END IF;

END IF;

-------------------------------------------------------------------------------

-- control bit 4

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(4) = '1') THEN

IF (sv\_data\_in(39 DOWNTO 32) = x"FE")THEN -- error in position 4

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(39 DOWNTO 32) = x"FD")THEN -- end of frame in position 4

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= sv\_data\_in(63 DOWNTO 40) & x"00" & sv\_data\_in(31 DOWNTO 0);

END IF;

ELSE

sv\_data\_out <= sv\_data\_in;

END IF;

-------------------------------------------------------------------------------

-- control bit 5

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(5) = '1') THEN

IF (sv\_data\_in(47 DOWNTO 40) = x"FE")THEN -- error in position 5

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(47 DOWNTO 40) = x"FD")THEN -- end of frame in position 5

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= sv\_data\_in(63 DOWNTO 48) & x"00" & sv\_data\_in(39 DOWNTO 0);

END IF;

ELSE

sv\_data\_out <= sv\_data\_in;

END IF;

-------------------------------------------------------------------------------

-- control bit 6

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(6) = '1') THEN

IF (sv\_data\_in(55 DOWNTO 48) = x"FE")THEN -- error in position 6

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(55 DOWNTO 48) = x"FD")THEN -- end of frame in position 6

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= sv\_data\_in(63 DOWNTO 56) & x"00" & sv\_data\_in(47 DOWNTO 0);

END IF;

ELSE

sv\_data\_out <= sv\_data\_in;

END IF;

-------------------------------------------------------------------------------

-- control bit 7

-------------------------------------------------------------------------------

IF (sv\_cntrl\_in(7) = '1') THEN

IF (sv\_data\_in(63 DOWNTO 56) = x"FE")THEN -- error in position 7

sv\_sof <= "00";

s\_err <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

ELSIF (sv\_data\_in(63 DOWNTO 56) = x"FD")THEN -- end of frame in position 7

s\_eof <= '1';

sv\_xgmii\_statemachine <= END\_FRAME;

sv\_data\_out <= x"00" & sv\_data\_in(55 DOWNTO 0);

END IF;

ELSE

sv\_data\_out <= sv\_data\_in;

END IF;

END IF;

WHEN END\_FRAME =>

sv\_sof <= "00";

s\_eof <= '0';

s\_err <= '0';

s\_valid\_out <= '0';

sv\_cntrl\_out <= (OTHERS => '0');

sv\_data\_out <= (OTHERS => '0');

sv\_xgmii\_statemachine <= INIT;

WHEN OTHERS =>

sv\_sof <= "00";

s\_eof <= '0';

s\_err <= '0';

s\_valid\_out <= '0';

sv\_cntrl\_out <= (OTHERS => '0');

sv\_data\_out <= (OTHERS => '0');

sv\_xgmii\_statemachine <= INIT;

END CASE;

END IF;

END PROCESS xgmii\_decode\_reg;

-------------------------------------------------------------------------------

--

-------------------------------------------------------------------------------

delay\_sigals\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

s\_valid\_1q <= '0';

sv\_data\_1q <= (OTHERS => '0');

sv\_cntrl\_1q <= (OTHERS => '0');

sv\_sof\_1q <= (OTHERS => '0');

s\_err\_1q <= '0';

s\_eof\_1q <= '0';

ELSIF (i\_clk'event and i\_clk = '1') THEN

IF (s\_valid\_out <= '1') THEN

s\_valid\_1q <= s\_valid\_out;

sv\_data\_1q <= sv\_data\_out;

sv\_cntrl\_1q <= sv\_cntrl\_out;

sv\_sof\_1q <= sv\_sof;

s\_err\_1q <= s\_err;

s\_eof\_1q <= s\_eof;

END IF;

END IF;

END PROCESS delay\_sigals\_reg;

-------------------------------------------------------------------------------

--

-------------------------------------------------------------------------------

fcs\_removal\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

s\_valid\_2q <= '0';

sv\_data\_2q <= (OTHERS => '0');

sv\_sof\_2q <= (OTHERS => '0');

s\_err\_2q <= '0';

s\_eof\_2q <= '0';

sv\_fcs\_removal\_sm <= INIT\_EOF;

ELSIF (i\_clk'event and i\_clk = '1') THEN

CASE sv\_fcs\_removal\_sm IS

WHEN INIT\_EOF =>

IF (s\_err = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q;

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '0';

sv\_fcs\_removal\_sm <= INIT\_EOF;

ELSE

IF (s\_eof = '1') THEN

IF (sv\_cntrl\_out(7) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q(63 DOWNTO 32) & x"00000000";

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '1';

sv\_fcs\_removal\_sm <= REMOVEL;

ELSIF (sv\_cntrl\_out(6) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q(63 DOWNTO 24) & x"000000";

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '1';

sv\_fcs\_removal\_sm <= REMOVEL;

ELSIF (sv\_cntrl\_out(5) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q(63 DOWNTO 16) & x"0000";

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '1';

sv\_fcs\_removal\_sm <= REMOVEL;

ELSIF (sv\_cntrl\_out(4) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q(63 DOWNTO 8) & x"00";

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '1';

sv\_fcs\_removal\_sm <= REMOVEL;

ELSIF (sv\_cntrl\_out(3) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q;

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '1';

sv\_fcs\_removal\_sm <= SECOND\_EOF;

ELSE

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q;

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

sv\_fcs\_removal\_sm <= SECOND\_EOF;

END IF;

ELSE

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q;

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

sv\_fcs\_removal\_sm <= INIT\_EOF;

END IF;

END IF;

WHEN SECOND\_EOF =>

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q;

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

IF (s\_eof\_1q = '1') THEN

IF (sv\_cntrl\_1q(3) = '1') THEN

s\_valid\_2q <= '0';

sv\_data\_2q <= (OTHERS => '0');

sv\_sof\_2q <= (OTHERS => '0');

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '0';

sv\_fcs\_removal\_sm <= REMOVEL;

ELSIF (sv\_cntrl\_1q(2) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q(63 DOWNTO 56) & x"00000000000000";

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

sv\_fcs\_removal\_sm <= REMOVEL;

ELSIF (sv\_cntrl\_1q(1) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q(63 DOWNTO 48) & x"000000000000";

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

sv\_fcs\_removal\_sm <= REMOVEL;

ELSIF (sv\_cntrl\_1q(0) = '1') THEN

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q(63 DOWNTO 40) & x"0000000000";

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

sv\_fcs\_removal\_sm <= REMOVEL;

ELSE

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q;

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

sv\_fcs\_removal\_sm <= INIT\_EOF;

END IF;

ELSE

s\_valid\_2q <= s\_valid\_1q;

sv\_data\_2q <= sv\_data\_1q;

sv\_sof\_2q <= sv\_sof\_1q;

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= s\_eof\_1q;

sv\_fcs\_removal\_sm <= INIT\_EOF;

END IF;

WHEN REMOVEL =>

s\_valid\_2q <= '0';

sv\_data\_2q <= (OTHERS => '0');

sv\_sof\_2q <= (OTHERS => '0');

s\_err\_2q <= s\_err\_1q;

s\_eof\_2q <= '0';

sv\_fcs\_removal\_sm <= INIT\_EOF;

WHEN OTHERS =>

s\_valid\_2q <= '0';

sv\_data\_2q <= (OTHERS => '0');

sv\_sof\_2q <= (OTHERS => '0');

s\_err\_2q <= '0';

s\_eof\_2q <= '0';

sv\_fcs\_removal\_sm <= INIT\_EOF;

END CASE;

END IF;

END PROCESS fcs\_removal\_reg;

o\_valid <= s\_valid\_2q;

ov\_data <= sv\_data\_2q;

ov\_sof <= sv\_sof\_2q;

o\_err <= s\_err\_2q;

o\_eof <= s\_eof\_2q AND NOT(s\_err\_2q);

END ARCHITECTURE rtl;

--------------------------------------------------

**Ether-type FSM VHDL Code**

-------------------------------------------------------------------------------

-- ether\_decode.vhd

--

-- delays signals for a set number of clock cycles

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use IEEE.numeric\_bit.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

-------------------------------------------------------------------------------

ENTITY ether\_decode IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

iv\_sof : IN STD\_LOGIC\_VECTOR (2-1 DOWNTO 0); --input start of frame

i\_err : IN STD\_LOGIC; --input error signal, used to reset the state machine and barrel shifter

i\_eof : IN STD\_LOGIC; --input end of frame

iv\_vlan\_defned\_tag : IN STD\_LOGIC\_VECTOR (16-1 DOWNTO 0); --input defined vlan tag which can be set in a cpu register

o\_read\_enable : OUT STD\_LOGIC; --output read enable to enable reading from the barrel shifter

o\_clear : OUT STD\_LOGIC; --output clear used to clear the barrel shifter

ov\_byte\_num : OUT STD\_LOGIC\_VECTOR (12-1 DOWNTO 0); --output byte number to indicate to the barrel shifter where to read from next

o\_t\_err : OUT STD\_LOGIC;

o\_sof : OUT STD\_LOGIC; --output start of frame to indicate to 5 tuple fsm to start

o\_fr\_err : OUT STD\_LOGIC; --output error signal, used to indicate a ether-type field error

o\_mpls\_err : OUT STD\_LOGIC; --output error signal, used to indicate a mpls label error

o\_eof : OUT STD\_LOGIC --output end of frame to indicate to 5 tuple fsm to end

);

END ether\_decode;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF ether\_decode IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

TYPE DECODE\_STATE\_TYPE IS (INIT, FIRST\_DECODE, VLAN, SECOND\_DECODE, MPLS, OUTPUT, END\_STATE);

-- input\_store\_reg -- max header is 592 with max vlan and max mpls so header shift reg is 10x64

SIGNAL s\_valid : STD\_LOGIC;

SIGNAL sv\_sof : STD\_LOGIC\_VECTOR(2-1 DOWNTO 0);

SIGNAL s\_t\_err : STD\_LOGIC;

SIGNAL s\_eof : STD\_LOGIC;

SIGNAL sv\_store\_count : STD\_LOGIC\_VECTOR(6-1 DOWNTO 0);

SIGNAL sv\_header\_capture : STD\_LOGIC\_VECTOR(640-1 DOWNTO 0);

-- ethertype\_decode\_reg --

SIGNAL s\_t\_err\_1q : STD\_LOGIC;

SIGNAL s\_mpls\_flag : STD\_LOGIC;

SIGNAL s\_q\_vlan\_flag : STD\_LOGIC;

SIGNAL s\_vlan\_defined\_flag : STD\_LOGIC;

SIGNAL s\_q\_in\_q\_flag : STD\_LOGIC;

SIGNAL s\_mac\_in\_mac\_flag : STD\_LOGIC;

SIGNAL sv\_byte\_num : STD\_LOGIC\_VECTOR(12-1 DOWNTO 0);

SIGNAL s\_read\_enable : STD\_LOGIC;

SIGNAL s\_sof\_out : STD\_LOGIC;

SIGNAL s\_frame\_error : STD\_LOGIC;

SIGNAL s\_mpls\_error : STD\_LOGIC;

SIGNAL s\_clear : STD\_LOGIC;

SIGNAL s\_eof\_out : STD\_LOGIC;

SIGNAL sv\_state\_machine : DECODE\_STATE\_TYPE;

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

s\_mpls\_flag <= NOT(s\_q\_vlan\_flag OR s\_vlan\_defined\_flag OR s\_q\_in\_q\_flag OR s\_mac\_in\_mac\_flag);

-------------------------------------------------------------------------------

-- capture the input data into the MAC header capture register

-------------------------------------------------------------------------------

input\_store\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_valid <= '0';

sv\_sof <= (OTHERS => '0');

s\_t\_err <= '0';

s\_eof <= '0';

sv\_store\_count <= (OTHERS => '0');

sv\_header\_capture <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_valid <= i\_valid;

sv\_sof <= iv\_sof;

s\_t\_err <= i\_err;

s\_eof <= i\_eof;

IF (i\_valid = '1') THEN

sv\_store\_count <= sv\_store\_count + 1;

IF (i\_eof = '1') OR

(i\_err = '1') THEN

sv\_store\_count <= (OTHERS => '0');

sv\_header\_capture <= (OTHERS => '0');

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 10) THEN

sv\_header\_capture <= sv\_header\_capture;

ELSE

sv\_header\_capture((((10-CONV\_INTEGER(sv\_store\_count))\*64)-1) DOWNTO (((10-CONV\_INTEGER(sv\_store\_count))-1)\*64)) <= iv\_data;

END IF;

END IF;

ELSE

sv\_store\_count <= sv\_store\_count;

sv\_header\_capture <= sv\_header\_capture;

END IF;

END IF;

END PROCESS input\_store\_reg;

-------------------------------------------------------------------------------

-- ethertype decode state machine

-------------------------------------------------------------------------------

ethertype\_decode\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_t\_err\_1q <= '0';

s\_q\_vlan\_flag <= '0';

s\_vlan\_defined\_flag <= '0';

s\_q\_in\_q\_flag <= '0';

s\_mac\_in\_mac\_flag <= '0';

s\_sof\_out <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_frame\_error <= '0';

s\_mpls\_error <= '0';

s\_clear <= '0';

s\_eof\_out <= '0';

sv\_state\_machine <= INIT;

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_t\_err\_1q <= s\_t\_err;

CASE sv\_state\_machine IS

WHEN INIT =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

---------------------------------------------------------------------------------------------------------

-- pass to FIRST\_DECODE if the start or frame is detected and increment the count

---------------------------------------------------------------------------------------------------------

IF (s\_valid = '1') THEN

IF ((sv\_sof = "01") OR

(sv\_sof = "10")) THEN

sv\_state\_machine <= FIRST\_DECODE;

END IF;

END IF;

END IF;

WHEN FIRST\_DECODE =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1' ) THEN

---------------------------------------------------------------------------------------------------------

-- decode first ether-type if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (CONV\_INTEGER(sv\_store\_count) >= 3) THEN

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set starting byte and read enable for barrel shifter

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(479 DOWNTO 464) = x"0800") THEN

s\_sof\_out <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(22,12);

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

---------------------------------------------------------------------------------------------------------

-- if ether-type is Q VLAN set starting byte and

-- enable Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(479 DOWNTO 464) = x"8100") OR

(sv\_header\_capture(479 DOWNTO 464) = x"9100") THEN

s\_q\_vlan\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(26,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is stacked VLAN then the state machine moves on to the VLAN state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(479 DOWNTO 464) = x"88A8") OR

(sv\_header\_capture(479 DOWNTO 464) = x"9200") THEN

sv\_state\_machine <= VLAN;

---------------------------------------------------------------------------------------------------------

-- if ether-type is user defined VLAN set starting byte and

-- enable user defined VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(479 DOWNTO 464) = iv\_vlan\_defned\_tag) OR

(sv\_header\_capture(511 DOWNTO 496) = x"9300") THEN

s\_vlan\_defined\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(34,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(479 DOWNTO 464) = x"8847") OR

(sv\_header\_capture(479 DOWNTO 464) = x"8848") THEN

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(22,12);

sv\_state\_machine <= MPLS;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set starting byte and read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(511 DOWNTO 496) = x"0800") THEN

s\_sof\_out <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(18,12);

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

---------------------------------------------------------------------------------------------------------

-- if ether-type is Q VLAN set starting byte and

-- enable Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(511 DOWNTO 496) = x"8100") OR

(sv\_header\_capture(511 DOWNTO 496) = x"9100") THEN

s\_q\_vlan\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(22,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is stacked VLAN then the state machine moves on to the VLAN state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(511 DOWNTO 496) = x"88A8") OR

(sv\_header\_capture(511 DOWNTO 496) = x"9200") THEN

sv\_state\_machine <= VLAN;

---------------------------------------------------------------------------------------------------------

-- if ether-type is user defined VLAN set starting byte and

-- enable user defined VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(511 DOWNTO 496) = iv\_vlan\_defned\_tag) OR

(sv\_header\_capture(511 DOWNTO 496) = x"9300") THEN

s\_vlan\_defined\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(30,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(511 DOWNTO 496) = x"8847") OR

(sv\_header\_capture(511 DOWNTO 496) = x"8848") THEN

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(18,12);

sv\_state\_machine <= MPLS;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN VLAN =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

---------------------------------------------------------------------------------------------------------

-- decode stack VLAN ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 8100 or 9100 set starting byte and

-- enable Q in Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(447 DOWNTO 432) = x"8100") OR

(sv\_header\_capture(447 DOWNTO 432) = x"9100") THEN

s\_q\_in\_q\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(30,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 88E7 set starting byte and

-- enable MAC in MAC VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(447 DOWNTO 432) = x"88E7") THEN

s\_mac\_in\_mac\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(50,12);

sv\_state\_machine <= SECOND\_DECODE;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode stack VLAN ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 8100 or 9100 set starting byte and

-- enable Q in Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(479 DOWNTO 464) = x"8100") OR

(sv\_header\_capture(479 DOWNTO 464) = x"9100") THEN

s\_q\_in\_q\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(26,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 88E7 set starting byte and

-- enable MAC in MAC VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(479 DOWNTO 464) = x"88E7") THEN

s\_mac\_in\_mac\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(46,12);

sv\_state\_machine <= SECOND\_DECODE;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

WHEN SECOND\_DECODE =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- decode if the Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(447 DOWNTO 432) = x"8847") OR

(sv\_header\_capture(447 DOWNTO 432) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(447 DOWNTO 432) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the Q in Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(415 DOWNTO 400) = x"8847") OR

(sv\_header\_capture(415 DOWNTO 400) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(415 DOWNTO 400) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the user defined VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(383 DOWNTO 368) = x"8847") OR

(sv\_header\_capture(383 DOWNTO 368) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(383 DOWNTO 368) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the MAC in MAC VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(255 DOWNTO 240) = x"8847") OR

(sv\_header\_capture(255 DOWNTO 240) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(255 DOWNTO 240) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- decode if the Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 3 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(479 DOWNTO 464) = x"8847") OR

(sv\_header\_capture(479 DOWNTO 464) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(479 DOWNTO 464) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the Q in Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(447 DOWNTO 432) = x"8847") OR

(sv\_header\_capture(447 DOWNTO 432) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(447 DOWNTO 432) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the user defined VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(415 DOWNTO 400) = x"8847") OR

(sv\_header\_capture(415 DOWNTO 400) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(415 DOWNTO 400) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the MAC in MAC VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_header\_capture(287 DOWNTO 272) = x"8847") OR

(sv\_header\_capture(287 DOWNTO 272) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_header\_capture(287 DOWNTO 272) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN MPLS =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1' ) THEN

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- if MPLS is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mpls\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_header\_capture(440) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if Q VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_header\_capture(408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(376) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_header\_capture(248) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(344) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(280) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_header\_capture(248) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(216) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_header\_capture(248) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(216) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 8 ) THEN

IF (sv\_header\_capture(184) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if MACinMAC VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_header\_capture(216) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 8 ) THEN

IF (sv\_header\_capture(184) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(152) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 9 ) THEN

IF (sv\_header\_capture(120) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(88) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 10 ) THEN

IF (sv\_header\_capture(56) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- if MPLS is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mpls\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 3 ) THEN

IF (sv\_header\_capture(472) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_header\_capture(440) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if Q VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_header\_capture(440) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(408) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_header\_capture(408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(376) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_header\_capture(248) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_header\_capture(376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(344) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_header\_capture(312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(280) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_header\_capture(248) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(216) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if MACinMAC VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_header\_capture(248) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(216) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 8 ) THEN

IF (sv\_header\_capture(184) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(152) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 9 ) THEN

IF (sv\_header\_capture(120) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_header\_capture(88) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN OUTPUT =>

IF (s\_t\_err = '1') THEN

s\_sof\_out <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

s\_sof\_out <= '0';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

IF (s\_valid = '1') THEN

IF (s\_eof = '1') THEN

s\_clear <= '1';

s\_eof\_out <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

WHEN END\_STATE =>

s\_sof\_out <= '0';

s\_q\_vlan\_flag <= '0';

s\_q\_in\_q\_flag <= '0';

s\_vlan\_defined\_flag <= '0';

s\_mac\_in\_mac\_flag <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_frame\_error <= '0';

s\_mpls\_error <= '0';

s\_clear <= '0';

s\_eof\_out <= '0';

------------------------------------------------------------------------------------------------------------

-- waits until the sof goes low so as to not get repeated frame errors from incorrect ether-types

------------------------------------------------------------------------------------------------------------

IF (sv\_sof = "00") THEN

sv\_state\_machine <= INIT;

END IF;

WHEN OTHERS =>

s\_sof\_out <= '0';

s\_q\_vlan\_flag <= '0';

s\_q\_in\_q\_flag <= '0';

s\_vlan\_defined\_flag <= '0';

s\_mac\_in\_mac\_flag <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_frame\_error <= '0';

s\_mpls\_error <= '0';

s\_clear <= '0';

s\_eof\_out <= '0';

sv\_state\_machine <= INIT;

END CASE;

END IF;

END PROCESS ethertype\_decode\_reg;

o\_read\_enable <= s\_read\_enable;

o\_clear <= s\_clear;

ov\_byte\_num <= sv\_byte\_num;

o\_t\_err <= s\_t\_err\_1q;

o\_sof <= s\_sof\_out;

o\_fr\_err <= s\_frame\_error;

o\_mpls\_err <= s\_mpls\_error;

o\_eof <= s\_eof\_out;

END ARCHITECTURE rtl;

--------------------------------------------------------------

**5 Tuple FSM VHDL Code**

-- tuple\_extract.vhd

--

-- extracts the 5 tuple and outputs it with the ip packet

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use IEEE.numeric\_bit.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

-------------------------------------------------------------------------------

ENTITY tuple\_extract IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

i\_sof : IN STD\_LOGIC;

i\_t\_err : IN STD\_LOGIC;

i\_eof : IN STD\_LOGIC;

ov\_five\_tuple : OUT STD\_LOGIC\_VECTOR (104-1 DOWNTO 0);

o\_read\_enable : OUT STD\_LOGIC;

o\_clear : OUT STD\_LOGIC;

ov\_byte\_num : OUT STD\_LOGIC\_VECTOR (12-1 DOWNTO 0);

o\_t\_err : OUT STD\_LOGIC;

o\_sof : OUT STD\_LOGIC; --output start of frame

o\_ihl\_err : OUT STD\_LOGIC; --output internet header length error

o\_ip\_err : OUT STD\_LOGIC; --output ip protocol error

o\_eof : OUT STD\_LOGIC --output end of frame

);

END tuple\_extract;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF tuple\_extract IS

-------------------------------------------------------------------------------

----------------------------------------------------------------------------

TYPE TUPLE\_SM\_TYPE IS (INIT, PROTOCOL\_DECODE, TUPLE\_EXTRACT, OUTPUT\_END, OUTPUT, END\_STATE);

-- input\_store\_reg -- signals

SIGNAL s\_valid : STD\_LOGIC;

SIGNAL s\_sof : STD\_LOGIC;

SIGNAL s\_t\_err : STD\_LOGIC;

SIGNAL s\_eof : STD\_LOGIC;

SIGNAL sv\_store\_count : STD\_LOGIC\_VECTOR(8-1 DOWNTO 0);

SIGNAL sv\_header\_capture : STD\_LOGIC\_VECTOR(256-1 DOWNTO 0);

-- tuple\_extract\_reg -- signals

SIGNAL s\_t\_err\_1q : STD\_LOGIC;

SIGNAL s\_ihl\_5 : STD\_LOGIC;

SIGNAL s\_ihl\_6 : STD\_LOGIC;

SIGNAL s\_icmp\_flag : STD\_LOGIC;

SIGNAL s\_read\_enable : STD\_LOGIC;

SIGNAL s\_clear : STD\_LOGIC;

SIGNAL sv\_byte\_num : STD\_LOGIC\_VECTOR (12-1 DOWNTO 0);

SIGNAL sv\_five\_tuple : STD\_LOGIC\_VECTOR (104-1 DOWNTO 0);

SIGNAL s\_sof\_out : STD\_LOGIC;

SIGNAL s\_ihl\_err : STD\_LOGIC;

SIGNAL s\_ip\_error : STD\_LOGIC;

SIGNAL s\_eof\_out : STD\_LOGIC;

SIGNAL sv\_out\_count : STD\_LOGIC\_VECTOR(3-1 DOWNTO 0);

SIGNAL sv\_tuple\_sm : TUPLE\_SM\_TYPE;

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

-- capture the input data into the IP/TCP header capture register

-------------------------------------------------------------------------------

input\_store\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_valid <= '0';

s\_sof <= '0';

s\_t\_err <= '0';

s\_eof <= '0';

sv\_store\_count <= (OTHERS => '0');

sv\_header\_capture <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_valid <= i\_valid;

s\_sof <= i\_sof;

s\_t\_err <= i\_t\_err;

s\_eof <= i\_eof;

IF (i\_eof = '1') OR

(i\_t\_err = '1') OR

(s\_clear = '1') THEN

sv\_store\_count <= (OTHERS => '0');

sv\_header\_capture <= (OTHERS => '0');

ELSE

IF (i\_valid = '1') THEN

sv\_store\_count <= sv\_store\_count + 1;

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

sv\_header\_capture <= sv\_header\_capture;

ELSE

sv\_header\_capture((((4-CONV\_INTEGER(sv\_store\_count))\*64)-1) DOWNTO (((4-CONV\_INTEGER(sv\_store\_count))-1)\*64)) <= iv\_data;

END IF;

ELSE

sv\_store\_count <= sv\_store\_count;

sv\_header\_capture <= sv\_header\_capture;

END IF;

END IF;

END IF;

END PROCESS input\_store\_reg;

-------------------------------------------------------------------------------

--

-------------------------------------------------------------------------------

tuple\_extract\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

s\_t\_err\_1q <= '0';

s\_ihl\_5 <= '0';

s\_ihl\_6 <= '0';

s\_icmp\_flag <= '0';

s\_read\_enable <= '0';

s\_clear <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_sof\_out <= '0';

s\_ihl\_err <= '0';

s\_ip\_error <= '0';

s\_eof\_out <= '0';

sv\_out\_count <= (OTHERS => '0');

sv\_five\_tuple <= (OTHERS => '0');

sv\_tuple\_sm <= INIT;

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_t\_err\_1q <= s\_t\_err;

CASE sv\_tuple\_sm IS

WHEN INIT =>

IF (s\_t\_err = '1') THEN

sv\_tuple\_sm <= INIT;

ELSE

IF (s\_valid = '1') THEN

IF (s\_sof = '1') THEN

IF (CONV\_INTEGER(sv\_header\_capture(251 DOWNTO 248)) = 5 ) THEN

s\_ihl\_5 <= '1';

sv\_tuple\_sm <= PROTOCOL\_DECODE;

ELSIF (CONV\_INTEGER(sv\_header\_capture(251 DOWNTO 248)) = 6 ) THEN

s\_ihl\_6 <= '1';

sv\_tuple\_sm <= PROTOCOL\_DECODE;

ELSE

s\_ihl\_err <= '1';

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

WHEN PROTOCOL\_DECODE =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 2 ) THEN

IF (CONV\_INTEGER(sv\_header\_capture(183 DOWNTO 176)) = 1 ) THEN

s\_icmp\_flag <= '1';

sv\_tuple\_sm <= TUPLE\_EXTRACT;

ELSIF (CONV\_INTEGER(sv\_header\_capture(183 DOWNTO 176)) = 6 ) OR

(CONV\_INTEGER(sv\_header\_capture(183 DOWNTO 176)) = 17 ) THEN

sv\_tuple\_sm <= TUPLE\_EXTRACT;

ELSE

s\_ip\_error <= '1';

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

WHEN TUPLE\_EXTRACT =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

IF (s\_icmp\_flag = '1') THEN

sv\_five\_tuple <= sv\_header\_capture(159 DOWNTO 128) & -- ip source

x"0000" & -- source port

sv\_header\_capture(127 DOWNTO 96) & -- ip destination

x"0000" & -- destination port

sv\_header\_capture(183 DOWNTO 176); -- protocol

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_tuple\_sm <= OUTPUT;

ELSE

IF (s\_ihl\_5 = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 3 ) THEN

sv\_five\_tuple <= sv\_header\_capture(159 DOWNTO 128) & -- ip source

sv\_header\_capture(95 DOWNTO 80) & -- source port

sv\_header\_capture(127 DOWNTO 96) & -- ip destination

sv\_header\_capture(79 DOWNTO 64) & -- destination port

sv\_header\_capture(183 DOWNTO 176); -- protocol

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_tuple\_sm <= OUTPUT;

END IF;

END IF;

IF (s\_ihl\_6 = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

sv\_five\_tuple <= sv\_header\_capture(159 DOWNTO 128) & -- ip source

sv\_header\_capture(63 DOWNTO 48) & -- source port

sv\_header\_capture(127 DOWNTO 96) & -- ip destination

sv\_header\_capture(47 DOWNTO 32) & -- destination port

sv\_header\_capture(183 DOWNTO 176); -- protocol

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_tuple\_sm <= OUTPUT;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN OUTPUT =>

IF (s\_t\_err = '1') THEN

s\_read\_enable <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

s\_sof\_out <= '0';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

IF (s\_valid = '1') THEN

IF (s\_eof = '1') THEN

sv\_out\_count <= sv\_out\_count + 1;

sv\_tuple\_sm <= OUTPUT\_END;

END IF;

END IF;

END IF;

WHEN OUTPUT\_END =>

IF (s\_t\_err = '1') THEN

s\_read\_enable <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

sv\_out\_count <= sv\_out\_count + 1;

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

IF (s\_ihl\_5 = '1') THEN

IF (CONV\_INTEGER(sv\_out\_count) = 2 ) THEN

s\_eof\_out <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

IF (s\_ihl\_6 = '1') THEN

IF (CONV\_INTEGER(sv\_out\_count) = 3 ) THEN

s\_eof\_out <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

END IF;

WHEN END\_STATE =>

s\_ihl\_5 <= '0';

s\_ihl\_6 <= '0';

s\_icmp\_flag <= '0';

s\_read\_enable <= '0';

s\_clear <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_sof\_out <= '0';

s\_ihl\_err <= '0';

s\_ip\_error <= '0';

s\_eof\_out <= '0';

sv\_out\_count <= (OTHERS => '0');

sv\_five\_tuple <= (OTHERS => '0');

sv\_tuple\_sm <= INIT;

WHEN OTHERS =>

s\_ihl\_5 <= '0';

s\_ihl\_6 <= '0';

s\_icmp\_flag <= '0';

s\_read\_enable <= '0';

s\_clear <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_sof\_out <= '0';

s\_ihl\_err <= '0';

s\_ip\_error <= '0';

s\_eof\_out <= '0';

sv\_out\_count <= (OTHERS => '0');

sv\_five\_tuple <= (OTHERS => '0');

sv\_tuple\_sm <= INIT;

END CASE;

END IF;

END PROCESS tuple\_extract\_reg;

ov\_five\_tuple <= sv\_five\_tuple;

o\_read\_enable <= s\_read\_enable;

o\_clear <= s\_clear;

ov\_byte\_num <= sv\_byte\_num;

o\_t\_err <= s\_t\_err\_1q;

o\_sof <= s\_sof\_out;

o\_ihl\_err <= s\_ihl\_err;

o\_ip\_err <= s\_ip\_error;

o\_eof <= s\_eof\_out;

END ARCHITECTURE rtl;

--------------------------------------------------

**5 Tuple Top Level VHDL Code**

-- tuple\_extract\_top\_level.vhd

--

-- extracts the 5 tuple and outputs it with the ip packet

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use IEEE.numeric\_bit.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

use work.component\_package.all;

-------------------------------------------------------------------------------

ENTITY tuple\_extract\_top\_level IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_cntrl : IN STD\_LOGIC\_VECTOR (4-1 DOWNTO 0); --input xgmii control

iv\_data : IN STD\_LOGIC\_VECTOR (32-1 DOWNTO 0); --input data

iv\_vlan\_defned\_tag : IN STD\_LOGIC\_VECTOR (16-1 DOWNTO 0); --input defined vlan tag which can be set in a cpu register

ov\_five\_tuple : OUT STD\_LOGIC\_VECTOR (104-1 DOWNTO 0); -- output 5 tuple

o\_valid : OUT STD\_LOGIC; --output data valid

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --output data

o\_sof : OUT STD\_LOGIC; --output start of frame

o\_eof : OUT STD\_LOGIC; --output end of frame

ov\_status\_reg : OUT STD\_LOGIC\_VECTOR (5-1 DOWNTO 0) --output status register, used to signal errors to the next module

);

END tuple\_extract\_top\_level;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF tuple\_extract\_top\_level IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- xgmii to ether-type fsm -- signals

SIGNAL s\_fr\_valid : STD\_LOGIC;

SIGNAL sv\_fr\_data : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

SIGNAL sv\_sof : STD\_LOGIC\_VECTOR(2-1 DOWNTO 0);

SIGNAL s\_eof : STD\_LOGIC;

SIGNAL s\_t\_err : STD\_LOGIC;

SIGNAL s\_fr\_rd : STD\_LOGIC;

SIGNAL s\_fr\_clr : STD\_LOGIC;

SIGNAL sv\_fr\_byte\_num : STD\_LOGIC\_VECTOR(12-1 DOWNTO 0);

SIGNAL s\_tuple\_t\_err : STD\_LOGIC;

SIGNAL s\_tuple\_sof : STD\_LOGIC;

SIGNAL s\_fr\_err : STD\_LOGIC;

SIGNAL s\_mpls\_err : STD\_LOGIC;

SIGNAL s\_tuple\_eof : STD\_LOGIC;

SIGNAL s\_tuple\_valid : STD\_LOGIC;

SIGNAL sv\_tuple\_data : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

-- signal\_delay\_reg -- signals

SIGNAL s\_tuple\_sof\_1q : STD\_LOGIC;

SIGNAL s\_tuple\_t\_err\_1q : STD\_LOGIC;

SIGNAL s\_tuple\_eof\_1q : STD\_LOGIC;

SIGNAL sv\_five\_tuple : STD\_LOGIC\_VECTOR(104-1 DOWNTO 0);

SIGNAL s\_pkt\_rd : STD\_LOGIC;

SIGNAL s\_pkt\_clr : STD\_LOGIC;

SIGNAL sv\_pkt\_byte\_num : STD\_LOGIC\_VECTOR(12-1 DOWNTO 0);

SIGNAL s\_pkt\_t\_err : STD\_LOGIC;

SIGNAL s\_pkt\_sof : STD\_LOGIC;

SIGNAL s\_ihl\_err : STD\_LOGIC;

SIGNAL s\_ip\_err : STD\_LOGIC;

SIGNAL s\_pkt\_eof : STD\_LOGIC;

SIGNAL s\_pkt\_valid : STD\_LOGIC;

SIGNAL sv\_pkt\_data : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

-- signal\_delay\_2\_reg -- signals

SIGNAL sv\_status\_reg : STD\_LOGIC\_VECTOR(5-1 DOWNTO 0);

SIGNAL sv\_five\_tuple\_out : STD\_LOGIC\_VECTOR(104-1 DOWNTO 0);

SIGNAL s\_pkt\_sof\_out : STD\_LOGIC;

SIGNAL s\_pkt\_eof\_out : STD\_LOGIC;

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

xgmii\_decoder\_inst : xgmii\_decoder

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => i\_valid,

iv\_cntrl => iv\_cntrl,

iv\_data => iv\_data,

o\_valid => s\_fr\_valid,

ov\_data => sv\_fr\_data,

ov\_sof => sv\_sof,

o\_err => s\_t\_err,

o\_eof => s\_eof

);

ether\_decode\_inst : ether\_decode

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => s\_fr\_valid,

iv\_data => sv\_fr\_data,

iv\_sof => sv\_sof,

i\_err => s\_t\_err,

i\_eof => s\_eof,

iv\_vlan\_defned\_tag => iv\_vlan\_defned\_tag,

o\_read\_enable => s\_fr\_rd,

o\_clear => s\_fr\_clr,

ov\_byte\_num => sv\_fr\_byte\_num,

o\_t\_err => s\_tuple\_t\_err,

o\_sof => s\_tuple\_sof,

o\_fr\_err => s\_fr\_err,

o\_mpls\_err => s\_mpls\_err,

o\_eof => s\_tuple\_eof

);

barrel\_shifter\_inst : barrel\_shifter

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => s\_fr\_valid,

iv\_data => sv\_fr\_data,

iv\_byte\_num => sv\_fr\_byte\_num,

i\_read\_enable => s\_fr\_rd,

i\_clear\_reg => s\_fr\_clr,

o\_valid => s\_tuple\_valid,

ov\_data => sv\_tuple\_data

);

-------------------------------------------------------------------------------

-- register sof,eof and tuple frame error so match up with barrel output

-------------------------------------------------------------------------------

signal\_delay\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_tuple\_sof\_1q <= '0';

s\_tuple\_t\_err\_1q <= '0';

s\_tuple\_eof\_1q <= '0';

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_tuple\_sof\_1q <= s\_tuple\_sof;

s\_tuple\_t\_err\_1q <= s\_tuple\_t\_err;

s\_tuple\_eof\_1q <= s\_tuple\_eof;

END IF;

END PROCESS signal\_delay\_reg;

tuple\_extract\_inst : tuple\_extract

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => s\_tuple\_valid,

iv\_data => sv\_tuple\_data,

i\_sof => s\_tuple\_sof\_1q,

i\_t\_err => s\_tuple\_t\_err\_1q,

i\_eof => s\_tuple\_eof\_1q,

ov\_five\_tuple => sv\_five\_tuple,

o\_read\_enable => s\_pkt\_rd,

o\_clear => s\_pkt\_clr,

ov\_byte\_num => sv\_pkt\_byte\_num,

o\_t\_err => s\_pkt\_t\_err,

o\_sof => s\_pkt\_sof,

o\_ihl\_err => s\_ihl\_err,

o\_ip\_err => s\_ip\_err,

o\_eof => s\_pkt\_eof

);

barrel\_shifter\_inst\_2 : barrel\_shifter

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => s\_tuple\_valid,

iv\_data => sv\_tuple\_data,

iv\_byte\_num => sv\_pkt\_byte\_num,

i\_read\_enable => s\_pkt\_rd,

i\_clear\_reg => s\_pkt\_clr,

o\_valid => s\_pkt\_valid,

ov\_data => sv\_pkt\_data

);

-------------------------------------------------------------------------------

-- register sof,eof and tuple frame error so match up with barrel output

-------------------------------------------------------------------------------

signal\_delay\_2\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

sv\_status\_reg <= (OTHERS => '0');

sv\_five\_tuple\_out <= (OTHERS => '0');

s\_pkt\_sof\_out <= '0';

s\_pkt\_eof\_out <= '0';

ELSIF (i\_clk'event and i\_clk = '1') THEN

sv\_status\_reg <= s\_pkt\_t\_err & s\_fr\_err & s\_mpls\_err & s\_ihl\_err & s\_ip\_err;

sv\_five\_tuple\_out <= sv\_five\_tuple;

s\_pkt\_sof\_out <= s\_pkt\_sof;

s\_pkt\_eof\_out <= s\_pkt\_eof;

END IF;

END PROCESS signal\_delay\_2\_reg;

ov\_five\_tuple <= sv\_five\_tuple\_out;

o\_valid <= s\_pkt\_valid;

ov\_data <= sv\_pkt\_data;

o\_sof <= s\_pkt\_sof\_out;

o\_eof <= s\_pkt\_eof\_out;

ov\_status\_reg <= sv\_status\_reg;

END ARCHITECTURE rtl;

-------------------------------------------

**Component Package VHDL Code**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

package component\_package is

COMPONENT ddr\_to\_sdr

GENERIC (

DATA\_WIDTH\_IN : INTEGER

);

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (DATA\_WIDTH\_IN-1 DOWNTO 0); --input data

o\_valid : OUT STD\_LOGIC; --output data valid

ov\_data : OUT STD\_LOGIC\_VECTOR ((DATA\_WIDTH\_IN\*2)-1 DOWNTO 0) --output data

);

END COMPONENT;

COMPONENT xgmii\_decoder

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_cntrl : IN STD\_LOGIC\_VECTOR (4-1 DOWNTO 0); --input xgmii control

iv\_data : IN STD\_LOGIC\_VECTOR (32-1 DOWNTO 0); --input data

o\_valid : OUT STD\_LOGIC; --output data valid

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --output data

ov\_sof : OUT STD\_LOGIC\_VECTOR (2-1 DOWNTO 0); --output start of frame

o\_err : OUT STD\_LOGIC; --output frame error

o\_eof : OUT STD\_LOGIC --output end of frame

);

END COMPONENT;

COMPONENT ether\_decode

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

iv\_sof : IN STD\_LOGIC\_VECTOR (2-1 DOWNTO 0); --input start of frame

i\_err : IN STD\_LOGIC; --input error signal, used to reset the state machine and barrel shifter

i\_eof : IN STD\_LOGIC; --input end of frame

iv\_vlan\_defned\_tag : IN STD\_LOGIC\_VECTOR (16-1 DOWNTO 0); --input defined vlan tag which can be set in a cpu register

o\_read\_enable : OUT STD\_LOGIC; --output read enable to enable reading from the barrel shifter

o\_clear : OUT STD\_LOGIC; --output clear used to clear the barrel shifter

ov\_byte\_num : OUT STD\_LOGIC\_VECTOR (12-1 DOWNTO 0); --output byte number to indicate to the barrel shifter where to read from next

o\_t\_err : OUT STD\_LOGIC;

o\_sof : OUT STD\_LOGIC; --output start of frame to indicate to 5 tuple fsm to start

o\_fr\_err : OUT STD\_LOGIC; --output error signal, used to indicate a ether-type field error

o\_mpls\_err : OUT STD\_LOGIC; --output error signal, used to indicate a mpls label error

o\_eof : OUT STD\_LOGIC --output end of frame to indicate to 5 tuple fsm to end

);

END COMPONENT;

COMPONENT barrel\_shifter

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

iv\_byte\_num : IN STD\_LOGIC\_VECTOR (12-1 DOWNTO 0); --input start of frame

i\_read\_enable : IN STD\_LOGIC; --input error signal, used to reset the state machine and barrel shifter

i\_clear\_reg : IN STD\_LOGIC; --input end of frame

o\_valid : OUT STD\_LOGIC; --output read enable to enable reading from the barrel shifter

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0) --output byte number to indicate to the barrel shifter where to read from next

);

END COMPONENT;

COMPONENT tuple\_extract

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

i\_sof : IN STD\_LOGIC;

i\_t\_err : IN STD\_LOGIC;

i\_eof : IN STD\_LOGIC;

ov\_five\_tuple : OUT STD\_LOGIC\_VECTOR (104-1 DOWNTO 0);

o\_read\_enable : OUT STD\_LOGIC;

o\_clear : OUT STD\_LOGIC;

ov\_byte\_num : OUT STD\_LOGIC\_VECTOR (12-1 DOWNTO 0);

o\_t\_err : OUT STD\_LOGIC;

o\_sof : OUT STD\_LOGIC; --output start of frame

o\_ihl\_err : OUT STD\_LOGIC; --output internet header length error

o\_ip\_err : OUT STD\_LOGIC; --output ip protocol error

o\_eof : OUT STD\_LOGIC --output end of frame

);

END COMPONENT;

end;

package body component\_package is

end package body;

**Appendix E**

**Corrected Double Data Rate Converter VHDL Code**

-- ddr\_to\_sdr.vhd

--

-- converts a double data rate to a single data rate on the same clock

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

-------------------------------------------------------------------------------

ENTITY ddr\_to\_sdr IS

GENERIC (

DATA\_WIDTH\_IN : INTEGER

);

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (DATA\_WIDTH\_IN-1 DOWNTO 0); --input data

o\_valid : OUT STD\_LOGIC; --output data valid

ov\_data : OUT STD\_LOGIC\_VECTOR ((DATA\_WIDTH\_IN\*2)-1 DOWNTO 0) --output data

);

END ddr\_to\_sdr;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF ddr\_to\_sdr IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- ddr\_to\_sdr\_reg -- signals

SIGNAL s\_clk : STD\_LOGIC;

SIGNAL s\_rst : STD\_LOGIC;

SIGNAL s\_single\_rate\_valid : STD\_LOGIC;

SIGNAL sv\_r\_double\_rate\_date : STD\_LOGIC\_VECTOR(DATA\_WIDTH\_IN-1 DOWNTO 0);

SIGNAL sv\_f\_double\_rate\_date : STD\_LOGIC\_VECTOR(DATA\_WIDTH\_IN-1 DOWNTO 0);

SIGNAL s\_valid : STD\_LOGIC;

SIGNAL sv\_single\_rate\_date : STD\_LOGIC\_VECTOR((DATA\_WIDTH\_IN\*2)-1 DOWNTO 0);

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- register data on both edges of the clock and then re-register on then rising edge

-------------------------------------------------------------------------------

s\_clk <= NOT(i\_clk);

re\_reg\_reset\_reg: PROCESS(i\_clk)

BEGIN

IF (i\_clk'event and i\_clk = '1') THEN --rising

s\_rst <= i\_rst;

END IF;

END PROCESS re\_reg\_reset\_reg;

ddr\_to\_sdr\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

s\_single\_rate\_valid <= '0';

sv\_r\_double\_rate\_date <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN --rising

s\_single\_rate\_valid <= i\_valid ;

sv\_r\_double\_rate\_date <= iv\_data;

END IF;

END PROCESS ddr\_to\_sdr\_reg;

ddr\_to\_sdr\_f\_reg: PROCESS(s\_clk,s\_rst)

BEGIN

IF s\_rst = '1' THEN

sv\_f\_double\_rate\_date <= (OTHERS => '0');

ELSIF (s\_clk'event and s\_clk = '1') THEN --falling

sv\_f\_double\_rate\_date <= iv\_data;

END IF;

END PROCESS ddr\_to\_sdr\_f\_reg;

output\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

s\_valid <= '0';

sv\_single\_rate\_date <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN --rising

s\_valid <= s\_single\_rate\_valid;

sv\_single\_rate\_date <= sv\_r\_double\_rate\_date & sv\_f\_double\_rate\_date;

END IF;

END PROCESS output\_reg;

o\_valid <= s\_valid;

ov\_data <= sv\_single\_rate\_date;

END ARCHITECTURE rtl;

--------------------------------------------------------------

**Appendix E**

**Optimied 5 Tuple Top Level VHDL Code**

-- tuple\_extract\_top\_level.vhd

--

-- extracts the 5 tuple and outputs it with the ip packet

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use IEEE.numeric\_bit.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

use work.component\_package.all;

-------------------------------------------------------------------------------

ENTITY tuple\_extract\_top\_level IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_cntrl : IN STD\_LOGIC\_VECTOR (4-1 DOWNTO 0); --input xgmii control

iv\_data : IN STD\_LOGIC\_VECTOR (32-1 DOWNTO 0); --input data

iv\_vlan\_defned\_tag : IN STD\_LOGIC\_VECTOR (16-1 DOWNTO 0); --input defined vlan tag which can be set in a cpu register

ov\_five\_tuple : OUT STD\_LOGIC\_VECTOR (104-1 DOWNTO 0); -- output 5 tuple

o\_valid : OUT STD\_LOGIC; --output data valid

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --output data

o\_sof : OUT STD\_LOGIC; --output start of frame

o\_eof : OUT STD\_LOGIC; --output end of frame

ov\_status\_reg : OUT STD\_LOGIC\_VECTOR (5-1 DOWNTO 0) --output status register, used to signal errors to the next module

);

END tuple\_extract\_top\_level;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF tuple\_extract\_top\_level IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

-- xgmii to ether-type fsm -- signals

SIGNAL s\_fr\_valid : STD\_LOGIC;

SIGNAL sv\_fr\_data : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

SIGNAL sv\_sof : STD\_LOGIC\_VECTOR(2-1 DOWNTO 0);

SIGNAL s\_eof : STD\_LOGIC;

SIGNAL s\_t\_err : STD\_LOGIC;

SIGNAL s\_tuple\_t\_err : STD\_LOGIC;

SIGNAL s\_tuple\_sof : STD\_LOGIC;

SIGNAL s\_fr\_err : STD\_LOGIC;

SIGNAL s\_mpls\_err : STD\_LOGIC;

SIGNAL s\_tuple\_eof : STD\_LOGIC;

SIGNAL s\_tuple\_valid : STD\_LOGIC;

SIGNAL sv\_tuple\_data : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

-- signal\_delay\_reg -- signals

SIGNAL s\_tuple\_sof\_1q : STD\_LOGIC;

SIGNAL s\_tuple\_t\_err\_1q : STD\_LOGIC;

SIGNAL s\_tuple\_eof\_1q : STD\_LOGIC;

SIGNAL sv\_five\_tuple : STD\_LOGIC\_VECTOR(104-1 DOWNTO 0);

SIGNAL s\_pkt\_t\_err : STD\_LOGIC;

SIGNAL s\_pkt\_sof : STD\_LOGIC;

SIGNAL s\_ihl\_err : STD\_LOGIC;

SIGNAL s\_ip\_err : STD\_LOGIC;

SIGNAL s\_pkt\_eof : STD\_LOGIC;

SIGNAL s\_pkt\_valid : STD\_LOGIC;

SIGNAL sv\_pkt\_data : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

-- signal\_delay\_2\_reg -- signals

SIGNAL sv\_status\_reg : STD\_LOGIC\_VECTOR(5-1 DOWNTO 0);

SIGNAL sv\_five\_tuple\_out : STD\_LOGIC\_VECTOR(104-1 DOWNTO 0);

SIGNAL s\_pkt\_sof\_out : STD\_LOGIC;

SIGNAL s\_pkt\_eof\_out : STD\_LOGIC;

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

xgmii\_decoder\_inst : xgmii\_decoder

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => i\_valid,

iv\_cntrl => iv\_cntrl,

iv\_data => iv\_data,

o\_valid => s\_fr\_valid,

ov\_data => sv\_fr\_data,

ov\_sof => sv\_sof,

o\_err => s\_t\_err,

o\_eof => s\_eof

);

ether\_decode\_inst : ether\_decode

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => s\_fr\_valid,

iv\_data => sv\_fr\_data,

iv\_sof => sv\_sof,

i\_err => s\_t\_err,

i\_eof => s\_eof,

iv\_vlan\_defned\_tag => iv\_vlan\_defned\_tag,

o\_valid => s\_tuple\_valid,

ov\_data => sv\_tuple\_data,

o\_t\_err => s\_tuple\_t\_err,

o\_sof => s\_tuple\_sof,

o\_fr\_err => s\_fr\_err,

o\_mpls\_err => s\_mpls\_err,

o\_eof => s\_tuple\_eof

);

-------------------------------------------------------------------------------

-- register sof,eof and tuple frame error so match up with barrel output

tuple\_extract\_inst : tuple\_extract

PORT MAP(

i\_clk => i\_clk,

i\_rst => i\_rst,

i\_valid => s\_tuple\_valid,

iv\_data => sv\_tuple\_data,

i\_sof => s\_tuple\_sof,

i\_t\_err => s\_tuple\_t\_err,

i\_eof => s\_tuple\_eof,

ov\_five\_tuple => sv\_five\_tuple,

o\_valid => s\_pkt\_valid,

ov\_data => sv\_pkt\_data,

o\_t\_err => s\_pkt\_t\_err,

o\_sof => s\_pkt\_sof,

o\_ihl\_err => s\_ihl\_err,

o\_ip\_err => s\_ip\_err,

o\_eof => s\_pkt\_eof

);

-------------------------------------------------------------------------------

-- register sof,eof and tuple frame error so match up with barrel output

-------------------------------------------------------------------------------

signal\_delay\_2\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

sv\_status\_reg <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

sv\_status\_reg <= s\_pkt\_t\_err & s\_fr\_err & s\_mpls\_err & s\_ihl\_err & s\_ip\_err;

END IF;

END PROCESS signal\_delay\_2\_reg;

ov\_five\_tuple <= sv\_five\_tuple;

o\_valid <= s\_pkt\_valid;

ov\_data <= sv\_pkt\_data;

o\_sof <= s\_pkt\_sof;

o\_eof <= s\_pkt\_eof;

ov\_status\_reg <= sv\_status\_reg;

END ARCHITECTURE rtl;

-------------------------------------------

**Optimied 5 Tuple FSM VHDL Code**

-- tuple\_extract.vhd

--

-- extracts the 5 tuple and outputs it with the ip packet

--

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use IEEE.numeric\_bit.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

-------------------------------------------------------------------------------

ENTITY tuple\_extract IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

i\_sof : IN STD\_LOGIC;

i\_t\_err : IN STD\_LOGIC;

i\_eof : IN STD\_LOGIC;

ov\_five\_tuple : OUT STD\_LOGIC\_VECTOR (104-1 DOWNTO 0);

o\_valid : OUT STD\_LOGIC;

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0);

o\_t\_err : OUT STD\_LOGIC;

o\_sof : OUT STD\_LOGIC; --output start of frame

o\_ihl\_err : OUT STD\_LOGIC; --output internet header length error

o\_ip\_err : OUT STD\_LOGIC; --output ip protocol error

o\_eof : OUT STD\_LOGIC --output end of frame

);

END tuple\_extract;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF tuple\_extract IS

-------------------------------------------------------------------------------

----------------------------------------------------------------------------

TYPE TUPLE\_SM\_TYPE IS (INIT, PROTOCOL\_DECODE, TUPLE\_EXTRACT, OUTPUT\_END, OUTPUT, END\_STATE);

-- input\_store\_reg -- MTU is 1526 bytes for a simple IP packet including preammble and SFD

-- -- with PBB VLAN and max MPLS MTU is 1578 bytes

SIGNAL sv\_store\_count : STD\_LOGIC\_VECTOR(6-1 DOWNTO 0);

SIGNAL sv\_frame\_capture : STD\_LOGIC\_VECTOR(64\*198-1 DOWNTO 0); -- 1578 \* 8

SIGNAL s\_valid : STD\_LOGIC;

SIGNAL s\_sof : STD\_LOGIC;

SIGNAL s\_t\_err : STD\_LOGIC;

SIGNAL s\_eof : STD\_LOGIC;

-- tuple\_extract\_reg -- signals

SIGNAL s\_t\_err\_1q : STD\_LOGIC;

SIGNAL s\_ihl\_5 : STD\_LOGIC;

SIGNAL s\_ihl\_6 : STD\_LOGIC;

SIGNAL s\_icmp\_flag : STD\_LOGIC;

SIGNAL s\_read\_enable : STD\_LOGIC;

SIGNAL s\_clear : STD\_LOGIC;

SIGNAL sv\_byte\_num : STD\_LOGIC\_VECTOR (12-1 DOWNTO 0);

SIGNAL sv\_five\_tuple : STD\_LOGIC\_VECTOR (104-1 DOWNTO 0);

SIGNAL s\_sof\_out : STD\_LOGIC;

SIGNAL s\_ihl\_err : STD\_LOGIC;

SIGNAL s\_ip\_error : STD\_LOGIC;

SIGNAL s\_eof\_out : STD\_LOGIC;

SIGNAL sv\_out\_count : STD\_LOGIC\_VECTOR(3-1 DOWNTO 0);

SIGNAL sv\_tuple\_sm : TUPLE\_SM\_TYPE;

-- output\_data\_reg

SIGNAL sv\_five\_tuple\_1q : STD\_LOGIC\_VECTOR (104-1 DOWNTO 0);

SIGNAL s\_sof\_out\_1q : STD\_LOGIC;

SIGNAL s\_eof\_out\_1q : STD\_LOGIC;

SIGNAL s\_valid\_out : STD\_LOGIC;

SIGNAL sv\_data\_out : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

-- capture the input data into the barrel

-------------------------------------------------------------------------------

input\_store\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_valid <= '0';

s\_sof <= '0';

s\_t\_err <= '0';

s\_eof <= '0';

sv\_store\_count <= (OTHERS => '0');

sv\_frame\_capture <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_valid <= i\_valid;

s\_sof <= i\_sof;

s\_t\_err <= i\_t\_err;

s\_eof <= i\_eof;

IF (s\_clear = '1') THEN

sv\_store\_count <= (OTHERS => '0');

sv\_frame\_capture <= (OTHERS => '0');

ELSE

IF (i\_valid = '1') THEN

sv\_store\_count <= sv\_store\_count + 1;

sv\_frame\_capture((((198-CONV\_INTEGER(sv\_store\_count))\*64)-1) DOWNTO (((198-CONV\_INTEGER(sv\_store\_count))-1)\*64)) <= iv\_data;

ELSE

sv\_store\_count <= sv\_store\_count;

sv\_frame\_capture <= sv\_frame\_capture;

END IF;

END IF;

END IF;

END PROCESS input\_store\_reg;

-------------------------------------------------------------------------------

--

-------------------------------------------------------------------------------

tuple\_extract\_reg: PROCESS(i\_clk,i\_rst)

BEGIN

IF i\_rst = '1' THEN

s\_t\_err\_1q <= '0';

s\_ihl\_5 <= '0';

s\_ihl\_6 <= '0';

s\_icmp\_flag <= '0';

s\_read\_enable <= '0';

s\_clear <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_sof\_out <= '0';

s\_ihl\_err <= '0';

s\_ip\_error <= '0';

s\_eof\_out <= '0';

sv\_out\_count <= (OTHERS => '0');

sv\_five\_tuple <= (OTHERS => '0');

sv\_tuple\_sm <= INIT;

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_t\_err\_1q <= s\_t\_err;

CASE sv\_tuple\_sm IS

WHEN INIT =>

IF (s\_t\_err = '1') THEN

sv\_tuple\_sm <= INIT;

ELSE

IF (s\_valid = '1') THEN

IF (s\_sof = '1') THEN

IF (CONV\_INTEGER(sv\_frame\_capture(12667 DOWNTO 12664)) = 5 ) THEN

s\_ihl\_5 <= '1';

sv\_tuple\_sm <= PROTOCOL\_DECODE;

ELSIF (CONV\_INTEGER(sv\_frame\_capture(12667 DOWNTO 12664)) = 6 ) THEN

s\_ihl\_6 <= '1';

sv\_tuple\_sm <= PROTOCOL\_DECODE;

ELSE

s\_ihl\_err <= '1';

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

WHEN PROTOCOL\_DECODE =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 2 ) THEN

IF (CONV\_INTEGER(sv\_frame\_capture(12599 DOWNTO 12592)) = 1 ) THEN

s\_icmp\_flag <= '1';

sv\_tuple\_sm <= TUPLE\_EXTRACT;

ELSIF (CONV\_INTEGER(sv\_frame\_capture(12599 DOWNTO 12592)) = 6 ) OR

(CONV\_INTEGER(sv\_frame\_capture(12599 DOWNTO 12592)) = 17 ) THEN

sv\_tuple\_sm <= TUPLE\_EXTRACT;

ELSE

s\_ip\_error <= '1';

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

WHEN TUPLE\_EXTRACT =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

IF (s\_icmp\_flag = '1') THEN

sv\_five\_tuple <= sv\_frame\_capture(12575 DOWNTO 12544) & -- ip source

x"0000" & -- source port

sv\_frame\_capture(12543 DOWNTO 12512) & -- ip destination

x"0000" & -- destination port

sv\_frame\_capture(12599 DOWNTO 12592); -- protocol

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_tuple\_sm <= OUTPUT;

ELSE

IF (s\_ihl\_5 = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 3 ) THEN

sv\_five\_tuple <= sv\_frame\_capture(12575 DOWNTO 12544) & -- ip source

sv\_frame\_capture(12511 DOWNTO 12496) & -- source port

sv\_frame\_capture(12543 DOWNTO 12512) & -- ip destination

sv\_frame\_capture(12495 DOWNTO 12480) & -- destination port

sv\_frame\_capture(12599 DOWNTO 12592); -- protocol

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_tuple\_sm <= OUTPUT;

END IF;

END IF;

IF (s\_ihl\_6 = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

sv\_five\_tuple <= sv\_frame\_capture(12575 DOWNTO 12544) & -- ip source

sv\_frame\_capture(12479 DOWNTO 12464) & -- source port

sv\_frame\_capture(12543 DOWNTO 12512) & -- ip destination

sv\_frame\_capture(12463 DOWNTO 12448) & -- destination port

sv\_frame\_capture(12599 DOWNTO 12592); -- protocol

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_tuple\_sm <= OUTPUT;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN OUTPUT =>

IF (s\_t\_err = '1') THEN

s\_read\_enable <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

s\_sof\_out <= '0';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

IF (s\_valid = '1') THEN

IF (s\_eof = '1') THEN

sv\_out\_count <= sv\_out\_count + 1;

sv\_tuple\_sm <= OUTPUT\_END;

END IF;

END IF;

END IF;

WHEN OUTPUT\_END =>

IF (s\_t\_err = '1') THEN

s\_read\_enable <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_clear <= '1';

sv\_tuple\_sm <= END\_STATE;

ELSE

sv\_out\_count <= sv\_out\_count + 1;

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

IF (s\_ihl\_5 = '1') THEN

IF (CONV\_INTEGER(sv\_out\_count) = 2 ) THEN

s\_clear <= '1';

s\_eof\_out <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

IF (s\_ihl\_6 = '1') THEN

IF (CONV\_INTEGER(sv\_out\_count) = 3 ) THEN

s\_clear <= '1';

s\_eof\_out <= '1';

sv\_tuple\_sm <= END\_STATE;

END IF;

END IF;

END IF;

WHEN END\_STATE =>

s\_ihl\_5 <= '0';

s\_ihl\_6 <= '0';

s\_icmp\_flag <= '0';

s\_read\_enable <= '0';

s\_clear <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_sof\_out <= '0';

s\_ihl\_err <= '0';

s\_ip\_error <= '0';

s\_eof\_out <= '0';

sv\_out\_count <= (OTHERS => '0');

sv\_five\_tuple <= (OTHERS => '0');

sv\_tuple\_sm <= INIT;

WHEN OTHERS =>

s\_ihl\_5 <= '0';

s\_ihl\_6 <= '0';

s\_icmp\_flag <= '0';

s\_read\_enable <= '0';

s\_clear <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_sof\_out <= '0';

s\_ihl\_err <= '0';

s\_ip\_error <= '0';

s\_eof\_out <= '0';

sv\_out\_count <= (OTHERS => '0');

sv\_five\_tuple <= (OTHERS => '0');

sv\_tuple\_sm <= INIT;

END CASE;

END IF;

END PROCESS tuple\_extract\_reg;

-------------------------------------------------------------------------------

-- output from barrel starting at byte sv\_byte\_num when read enable is high

-------------------------------------------------------------------------------

output\_data\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

sv\_five\_tuple\_1q <= (OTHERS => '0');

s\_sof\_out\_1q <= '0';

s\_eof\_out\_1q <= '0';

s\_valid\_out <= '0';

sv\_data\_out <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

sv\_five\_tuple\_1q <= sv\_five\_tuple;

s\_sof\_out\_1q <= s\_sof\_out;

s\_eof\_out\_1q <= s\_eof\_out;

s\_valid\_out <= '0';

sv\_data\_out <= (OTHERS => '0');

IF (s\_read\_enable = '1') THEN

s\_valid\_out <= '1';

sv\_data\_out <= sv\_frame\_capture((((1584 - CONV\_INTEGER(sv\_byte\_num))\*8)-1) DOWNTO ((1576 - CONV\_INTEGER(sv\_byte\_num))\*8));

END IF;

END IF;

END PROCESS output\_data\_reg;

ov\_five\_tuple <= sv\_five\_tuple\_1q;

o\_valid <= s\_valid\_out;

ov\_data <= sv\_data\_out;

o\_t\_err <= s\_t\_err\_1q;

o\_sof <= s\_sof\_out\_1q;

o\_ihl\_err <= s\_ihl\_err;

o\_ip\_err <= s\_ip\_error;

o\_eof <= s\_eof\_out\_1q;

END ARCHITECTURE rtl;

-------------------------

**Optimied Ether-tpye FSM VHDL Code**

-------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use IEEE.numeric\_bit.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

-------------------------------------------------------------------------------

ENTITY ether\_decode IS

PORT(

i\_clk : IN STD\_LOGIC;

i\_rst : IN STD\_LOGIC;

i\_valid : IN STD\_LOGIC; --input data valid

iv\_data : IN STD\_LOGIC\_VECTOR (64-1 DOWNTO 0); --input data

iv\_sof : IN STD\_LOGIC\_VECTOR (2-1 DOWNTO 0); --input start of frame

i\_err : IN STD\_LOGIC; --input error signal, used to reset the state machine and barrel shifter

i\_eof : IN STD\_LOGIC; --input end of frame

iv\_vlan\_defned\_tag : IN STD\_LOGIC\_VECTOR (16-1 DOWNTO 0); --input defined vlan tag which can be set in a cpu register

o\_valid : OUT STD\_LOGIC;

ov\_data : OUT STD\_LOGIC\_VECTOR (64-1 DOWNTO 0);

o\_t\_err : OUT STD\_LOGIC;

o\_sof : OUT STD\_LOGIC; --output start of frame to indicate to 5 tuple fsm to start

o\_fr\_err : OUT STD\_LOGIC; --output error signal, used to indicate a ether-type field error

o\_mpls\_err : OUT STD\_LOGIC; --output error signal, used to indicate a mpls label error

o\_eof : OUT STD\_LOGIC --output end of frame to indicate to 5 tuple fsm to end

);

END ether\_decode;

-------------------------------------------------------------------------------

ARCHITECTURE rtl OF ether\_decode IS

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

TYPE DECODE\_STATE\_TYPE IS (INIT, FIRST\_DECODE, VLAN, SECOND\_DECODE, MPLS, OUTPUT, END\_STATE);

-------------------------------------------------------------------------------

-- input\_store\_reg -- MTU is 1526 bytes for a simple IP packet including preammble and SFD

-- -- with PBB VLAN and max MPLS MTU is 1578 bytes

SIGNAL sv\_store\_count : STD\_LOGIC\_VECTOR(6-1 DOWNTO 0);

SIGNAL sv\_frame\_capture : STD\_LOGIC\_VECTOR(64\*198-1 DOWNTO 0); -- 1578 \* 8

SIGNAL s\_valid : STD\_LOGIC;

SIGNAL sv\_sof : STD\_LOGIC\_VECTOR(2-1 DOWNTO 0);

SIGNAL s\_t\_err : STD\_LOGIC;

SIGNAL s\_eof : STD\_LOGIC;

-- ethertype\_decode\_reg --

SIGNAL s\_t\_err\_1q : STD\_LOGIC;

SIGNAL s\_mpls\_flag : STD\_LOGIC;

SIGNAL s\_q\_vlan\_flag : STD\_LOGIC;

SIGNAL s\_vlan\_defined\_flag : STD\_LOGIC;

SIGNAL s\_q\_in\_q\_flag : STD\_LOGIC;

SIGNAL s\_mac\_in\_mac\_flag : STD\_LOGIC;

SIGNAL sv\_byte\_num : STD\_LOGIC\_VECTOR(12-1 DOWNTO 0);

SIGNAL s\_read\_enable : STD\_LOGIC;

SIGNAL s\_sof\_out : STD\_LOGIC;

SIGNAL s\_frame\_error : STD\_LOGIC;

SIGNAL s\_mpls\_error : STD\_LOGIC;

SIGNAL s\_clear : STD\_LOGIC;

SIGNAL s\_eof\_out : STD\_LOGIC;

SIGNAL sv\_state\_machine : DECODE\_STATE\_TYPE;

-- output\_data\_reg

SIGNAL s\_sof\_out\_1q : STD\_LOGIC;

SIGNAL s\_eof\_out\_1q : STD\_LOGIC;

SIGNAL s\_t\_err\_2q : STD\_LOGIC;

SIGNAL s\_valid\_out : STD\_LOGIC;

SIGNAL sv\_data\_out : STD\_LOGIC\_VECTOR(64-1 DOWNTO 0);

-------------------------------------------------------------------------------

BEGIN -- rtl

-------------------------------------------------------------------------------

s\_mpls\_flag <= NOT(s\_q\_vlan\_flag OR s\_vlan\_defined\_flag OR s\_q\_in\_q\_flag OR s\_mac\_in\_mac\_flag);

-------------------------------------------------------------------------------

-- capture the input data into the barrel

-------------------------------------------------------------------------------

input\_store\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_valid <= '0';

sv\_sof <= (OTHERS => '0');

s\_t\_err <= '0';

s\_eof <= '0';

sv\_store\_count <= (OTHERS => '0');

sv\_frame\_capture <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_valid <= i\_valid;

sv\_sof <= iv\_sof;

s\_t\_err <= i\_err;

s\_eof <= i\_eof;

IF (s\_clear = '1') THEN

sv\_store\_count <= (OTHERS => '0');

sv\_frame\_capture <= (OTHERS => '0');

ELSE

IF (i\_valid = '1') THEN

sv\_store\_count <= sv\_store\_count + 1;

sv\_frame\_capture((((198-CONV\_INTEGER(sv\_store\_count))\*64)-1) DOWNTO (((198-CONV\_INTEGER(sv\_store\_count))-1)\*64)) <= iv\_data;

ELSE

sv\_store\_count <= sv\_store\_count;

sv\_frame\_capture <= sv\_frame\_capture;

END IF;

END IF;

END IF;

END PROCESS input\_store\_reg;

-------------------------------------------------------------------------------

-- ethertype decode state machine

-------------------------------------------------------------------------------

ethertype\_decode\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_t\_err\_1q <= '0';

s\_q\_vlan\_flag <= '0';

s\_vlan\_defined\_flag <= '0';

s\_q\_in\_q\_flag <= '0';

s\_mac\_in\_mac\_flag <= '0';

s\_sof\_out <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_frame\_error <= '0';

s\_mpls\_error <= '0';

s\_clear <= '0';

s\_eof\_out <= '0';

sv\_state\_machine <= INIT;

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_t\_err\_1q <= s\_t\_err;

CASE sv\_state\_machine IS

WHEN INIT =>

s\_clear <= '0';

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

---------------------------------------------------------------------------------------------------------

-- pass to FIRST\_DECODE if the start or frame is detected and increment the count

---------------------------------------------------------------------------------------------------------

IF (s\_valid = '1') THEN

IF ((sv\_sof = "01") OR

(sv\_sof = "10")) THEN

sv\_state\_machine <= FIRST\_DECODE;

END IF;

END IF;

END IF;

WHEN FIRST\_DECODE =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1' ) THEN

---------------------------------------------------------------------------------------------------------

-- decode first ether-type if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (CONV\_INTEGER(sv\_store\_count) >= 3) THEN

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set starting byte and read enable for barrel shifter

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12511 DOWNTO 12496) = x"0800") THEN

s\_sof\_out <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(22,12);

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

---------------------------------------------------------------------------------------------------------

-- if ether-type is Q VLAN set starting byte and

-- enable Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12511 DOWNTO 12496) = x"8100") OR

(sv\_frame\_capture(12511 DOWNTO 12496) = x"9100") THEN

s\_q\_vlan\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(26,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is stacked VLAN then the state machine moves on to the VLAN state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12511 DOWNTO 12496) = x"88A8") OR

(sv\_frame\_capture(12511 DOWNTO 12496) = x"9200") THEN

sv\_state\_machine <= VLAN;

---------------------------------------------------------------------------------------------------------

-- if ether-type is user defined VLAN set starting byte and

-- enable user defined VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12511 DOWNTO 12496) = iv\_vlan\_defned\_tag) OR

(sv\_frame\_capture(12511 DOWNTO 12496) = x"9300") THEN

s\_vlan\_defined\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(34,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12511 DOWNTO 12496) = x"8847") OR

(sv\_frame\_capture(12511 DOWNTO 12496) = x"8848") THEN

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(22,12);

sv\_state\_machine <= MPLS;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set starting byte and read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12543 DOWNTO 12528) = x"0800") THEN

s\_sof\_out <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(18,12);

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

---------------------------------------------------------------------------------------------------------

-- if ether-type is Q VLAN set starting byte and

-- enable Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12543 DOWNTO 12528) = x"8100") OR

(sv\_frame\_capture(12543 DOWNTO 12528) = x"9100") THEN

s\_q\_vlan\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(22,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is stacked VLAN then the state machine moves on to the VLAN state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12543 DOWNTO 12528) = x"88A8") OR

(sv\_frame\_capture(12543 DOWNTO 12528) = x"9200") THEN

sv\_state\_machine <= VLAN;

---------------------------------------------------------------------------------------------------------

-- if ether-type is user defined VLAN set starting byte and

-- enable user defined VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12543 DOWNTO 12528) = iv\_vlan\_defned\_tag) OR

(sv\_frame\_capture(12543 DOWNTO 12528) = x"9300") THEN

s\_vlan\_defined\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(30,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12543 DOWNTO 12528) = x"8847") OR

(sv\_frame\_capture(12543 DOWNTO 12528) = x"8848") THEN

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(18,12);

sv\_state\_machine <= MPLS;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN VLAN =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

---------------------------------------------------------------------------------------------------------

-- decode stack VLAN ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 8100 or 9100 set starting byte and

-- enable Q in Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12479 DOWNTO 12464) = x"8100") OR

(sv\_frame\_capture(12479 DOWNTO 12464) = x"9100") THEN

s\_q\_in\_q\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(30,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 88E7 set starting byte and

-- enable MAC in MAC VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12479 DOWNTO 12464) = x"88E7") THEN

s\_mac\_in\_mac\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(50,12);

sv\_state\_machine <= SECOND\_DECODE;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode stack VLAN ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 8100 or 9100 set starting byte and

-- enable Q in Q VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12511 DOWNTO 12496) = x"8100") OR

(sv\_frame\_capture(12511 DOWNTO 12496) = x"9100") THEN

s\_q\_in\_q\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(26,12);

sv\_state\_machine <= SECOND\_DECODE;

---------------------------------------------------------------------------------------------------------

-- if the second VLAN tag is 88E7 set starting byte and

-- enable MAC in MAC VLAN flag to indicate to second decode

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12511 DOWNTO 12496) = x"88E7") THEN

s\_mac\_in\_mac\_flag <= '1';

sv\_byte\_num <= CONV\_STD\_LOGIC\_VECTOR(46,12);

sv\_state\_machine <= SECOND\_DECODE;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

WHEN SECOND\_DECODE =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1') THEN

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- decode if the Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12479 DOWNTO 12464) = x"8847") OR

(sv\_frame\_capture(12479 DOWNTO 12464) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12479 DOWNTO 12464) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the Q in Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12447 DOWNTO 12432) = x"8847") OR

(sv\_frame\_capture(12447 DOWNTO 12432) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12447 DOWNTO 12432) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the user defined VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12415 DOWNTO 12400) = x"8847") OR

(sv\_frame\_capture(12415 DOWNTO 12400) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12415 DOWNTO 12400) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the MAC in MAC VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12287 DOWNTO 12272) = x"8847") OR

(sv\_frame\_capture(12287 DOWNTO 12272) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12287 DOWNTO 12272) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode ether-type field depending on start of frame position

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- decode if the Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 3 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12511 DOWNTO 12496) = x"8847") OR

(sv\_frame\_capture(12511 DOWNTO 12496) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12511 DOWNTO 12496) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the Q in Q VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12479 DOWNTO 12464) = x"8847") OR

(sv\_frame\_capture(12479 DOWNTO 12464) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12479 DOWNTO 12464) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the user defined VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12447 DOWNTO 12432) = x"8847") OR

(sv\_frame\_capture(12447 DOWNTO 12432) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12447 DOWNTO 12432) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- decode if the MAC in MAC VLAN flag is set and if enough data has been collected

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1') AND

(CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

---------------------------------------------------------------------------------------------------------

-- if ether-type is MPLS then the state machine moves on to the MPLS state

---------------------------------------------------------------------------------------------------------

IF (sv\_frame\_capture(12319 DOWNTO 12304) = x"8847") OR

(sv\_frame\_capture(12319 DOWNTO 12304) = x"8848") THEN

sv\_state\_machine <= MPLS;

---------------------------------------------------------------------------------------------------------

-- if ether-type is IP set read enable for barrel shifter

-- state machine moves on to the OUTPUT state

---------------------------------------------------------------------------------------------------------

ELSIF (sv\_frame\_capture(12319 DOWNTO 12304) = x"0800") THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_state\_machine <= OUTPUT;

ELSE

s\_frame\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN MPLS =>

IF (s\_t\_err = '1') THEN

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

IF (s\_valid = '1' ) THEN

IF (sv\_sof = "01") THEN

---------------------------------------------------------------------------------------------------------

-- if MPLS is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mpls\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_frame\_capture(12472) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12440) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if Q VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_frame\_capture(12440) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12408) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_frame\_capture(12280) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12376) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12312) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_frame\_capture(12280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12248) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_frame\_capture(12280) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12248) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 8 ) THEN

IF (sv\_frame\_capture(12216) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if MACinMAC VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_frame\_capture(12248) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 8 ) THEN

IF (sv\_frame\_capture(12216) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12184) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 9 ) THEN

IF (sv\_frame\_capture(12152) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12120) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 10 ) THEN

IF (sv\_frame\_capture(12088) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

ELSIF (sv\_sof = "10") THEN

---------------------------------------------------------------------------------------------------------

-- if MPLS is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mpls\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 3 ) THEN

IF (sv\_frame\_capture(12504) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_frame\_capture(12472) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12440) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if Q VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_vlan\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_frame\_capture(12472) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12440) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_q\_in\_q\_flag = '1') THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 4 ) THEN

IF (sv\_frame\_capture(12440) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12408) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12376) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12312) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_frame\_capture(12280) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if QinQinQ VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_vlan\_defined\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 5 ) THEN

IF (sv\_frame\_capture(12408) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12376) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 6 ) THEN

IF (sv\_frame\_capture(12344) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12312) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_frame\_capture(12280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12248) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

---------------------------------------------------------------------------------------------------------

-- if MACinMAC VLAN is decoded in first decode check for S bit and add corresponding amount of bytes

---------------------------------------------------------------------------------------------------------

IF (s\_mac\_in\_mac\_flag = '1' ) THEN

IF (CONV\_INTEGER(sv\_store\_count) >= 7 ) THEN

IF (sv\_frame\_capture(12280) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(4,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12248) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 8 ) THEN

IF (sv\_frame\_capture(12216) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(12,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12184) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(16,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (CONV\_INTEGER(sv\_store\_count) >= 9 ) THEN

IF (sv\_frame\_capture(12152) = '1') THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(20,12);

sv\_state\_machine <= OUTPUT;

ELSE

IF (sv\_frame\_capture(12120) = '1' ) THEN

s\_sof\_out <= '1';

s\_read\_enable <= '1';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(24,12);

sv\_state\_machine <= OUTPUT;

ELSE

s\_mpls\_error <= '1';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

END IF;

WHEN OUTPUT =>

IF (s\_t\_err = '1') THEN

s\_sof\_out <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_clear <= '1';

sv\_state\_machine <= END\_STATE;

ELSE

s\_sof\_out <= '0';

sv\_byte\_num <= sv\_byte\_num + CONV\_STD\_LOGIC\_VECTOR(8,12);

IF (s\_valid = '1') THEN

IF (s\_eof = '1') THEN

s\_clear <= '1';

s\_eof\_out <= '1';

sv\_state\_machine <= END\_STATE;

END IF;

END IF;

END IF;

WHEN END\_STATE =>

s\_sof\_out <= '0';

s\_q\_vlan\_flag <= '0';

s\_q\_in\_q\_flag <= '0';

s\_vlan\_defined\_flag <= '0';

s\_mac\_in\_mac\_flag <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_frame\_error <= '0';

s\_mpls\_error <= '0';

s\_eof\_out <= '0';

------------------------------------------------------------------------------------------------------------

-- waits until the sof goes low so as to not get repeated frame errors from incorrect ether-types

------------------------------------------------------------------------------------------------------------

IF (sv\_sof = "00") THEN

sv\_state\_machine <= INIT;

END IF;

WHEN OTHERS =>

s\_sof\_out <= '0';

s\_q\_vlan\_flag <= '0';

s\_q\_in\_q\_flag <= '0';

s\_vlan\_defined\_flag <= '0';

s\_mac\_in\_mac\_flag <= '0';

sv\_byte\_num <= (OTHERS => '0');

s\_read\_enable <= '0';

s\_frame\_error <= '0';

s\_mpls\_error <= '0';

s\_clear <= '0';

s\_eof\_out <= '0';

sv\_state\_machine <= INIT;

END CASE;

END IF;

END PROCESS ethertype\_decode\_reg;

-------------------------------------------------------------------------------

-- output from barrel starting at byte sv\_byte\_num when read enable is high

-------------------------------------------------------------------------------

output\_data\_reg: PROCESS(i\_clk, i\_rst)

BEGIN

IF (i\_rst = '1') THEN

s\_sof\_out\_1q <= '0';

s\_eof\_out\_1q <= '0';

s\_t\_err\_2q <= '0';

s\_valid\_out <= '0';

sv\_data\_out <= (OTHERS => '0');

ELSIF (i\_clk'event and i\_clk = '1') THEN

s\_sof\_out\_1q <= s\_sof\_out;

s\_eof\_out\_1q <= s\_eof\_out;

s\_t\_err\_2q <= s\_t\_err\_1q;

s\_valid\_out <= '0';

sv\_data\_out <= (OTHERS => '0');

IF (s\_read\_enable = '1') THEN

s\_valid\_out <= '1';

sv\_data\_out <= sv\_frame\_capture((((1584 - CONV\_INTEGER(sv\_byte\_num))\*8)-1) DOWNTO ((1576 - CONV\_INTEGER(sv\_byte\_num))\*8));

END IF;

END IF;

END PROCESS output\_data\_reg;

o\_valid <= s\_valid\_out;

ov\_data <= sv\_data\_out;

o\_t\_err <= s\_t\_err\_2q;

o\_sof <= s\_sof\_out\_1q;

o\_fr\_err <= s\_frame\_error;

o\_mpls\_err <= s\_mpls\_error;

o\_eof <= s\_eof\_out\_1q;

END ARCHITECTURE rtl;

--------------------------------------------------------------