REFERENCE LIST

[1.] R. Seifert and J. Edwards, The All-New Switch Book, The Complete Guide to LAN Switching Technology, 2nd ed., New York: John Wiley & Sons, 2011.

[2.] Ethernet Task Force (2011, Mar), Home page of the IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force, [Online]. Available: http://grouper.ieee.org/groups/802/3/ba/public/index.html.

[3.] C. Cole, J. D'Ambrosia, C. DiMinico, H. Frazier, A. Healey, J. Jaeger, J. Jewell, M. Nowell, and S. Trowbridge, (2007, Nov.). An overview: The next generation of ethernet, IEEE 802.3-HSSG Meeting [Online]. Available: http://www.ieee802.org/3/hssg/public/nov07/index.htm

[4.] P. Reviriego, B. Huiszoon, V. L´opez, R. B. Coenen, J. A. Hern´andez, and J. A. Maestro, "Improving Energy Efficiency in IEEE 802.3ba High-Rate Ethernet Optical Links", IEEE Syst. J., vol. 17, no 2, pp. 419-427, March-April 2011.

[5.] IEEE Standards for Ethernet, IEEE Std 802.3™, 2012.

[6.] Merilee Ford et al., "Chapter 7 Ethernet Technologies," in Internetworking technology Overview, Cisco Systems, 1999.

[7.] Mark Gustlin et al. (2007 September).100GE and 40GEPCS Proposal [Online]. Available: http://grouper.ieee.org/groups/802/3/hssg/public/sept07/gustlin_01_0907.pdf

[8.] John Ambrosia et al., 40 Gigabit Ethernet and 100 Gigabit Ethernet Technology Overview, ethernet alliance, June 2010.

[9.] Mark Gustlin, 40 and 100 Gigabit Ethernet PCS and PMA Overview, Cisco Systems Inc., October 2010.

[10.] Jorg Sommer et al., “Ethernet – A Survey on its fields of Application,” in IEEE Communications Surveys and Communications, 2010, Vol. 12, No. 2.

[11.] Subramaniam Thayaparan and Anuradha Nanayakkara, “FIFO Design for IEEE 802.3 Standard 10GBase-X PCS and XGXS Sublayers,” in 2013 4th International Conference on Intelligent Systems, Modelling and Simulation, Bangkok, 2013, pp. 589-591

[12.] Faisal Dada and Norbert Folkens. IPG Considerations [Online]. Available: http://grouper.ieee.org/groups/802/3/ba/public/may08/folkens_01_0508.pdf#page=3

[13.] Sowmya S Luckloor, “Introduction to 10 Gigabit 64B/66B”, October, 2001.

[14.] Kushan B Vadawala. “Universal Scrambler by using Verilog HDL”, International Journal of Engineering Sciences and Research Technology, March, 2014.

[15.] W.P. Ranjaula et al., “Implementation Techniques for IEEE 802.3ba 40Gbps Ethernet Physical Coding Sublayer (PCS)”, in 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2015.

[16.] Tom Warland. (2011, December 14). Understanding Skew in 100GBASE-R4 applications [Online]. Available: http://www.eetimes.com/documnet.asp?doc_id=1279294

[17.] Robert Winter et al., Ethernet Jumbo Frames, ethernet alliance, November 12 2009.

[18.] Wikipedia, the free encyclopedia (2015, February 11). Jumbo frames [online]. Available: https://en.wikipedia.org/wiki/Jumbo_frame

[19.] Sarath Pillai, (2014, July 15). What is a Jumbo Frame in Ethernet [Online]? Available: http://www.slashroot.in/networking

[20.] Extended Frame Sizes for Next Generation Ethernet [Online]. Available: https://www.psc.edu/~mathis/MTU/AlteonExtendedFrames_W0601.pdf

[21.] Doug Raid. (2007, October 26). Need To Know: Jumbo Frames in Small Networks [Online]. Available: http://www.smallnetbuilder.com/lanwan/lanwan-features/30201-need-to-know-jumbo-frames-in-small-networks?limitstart=0

[22.] ModelSim User's Manual, Software version 6.6d, Mentor Graphics Corporation, Oregon, 2010.

[23.] IEEE Standard Verilog Hardware Description Language, IEEE Std 1364-2001, 2001.

Appendix A: Verilog test bench for 10GBASE-X PCS Sublayer FIFO design

```
// -----------------------------------------------------------------
// This is the implementation of TestBench for Single Asynchronous
FIFO
// Note: FIFO FULL :-Wr_Clk>Rd_Clk; Wr_Clk T = 3199
//       FIFO EMPTY :- Rd Clk>Wr_Clk; Wr_Clk T = 3200
// -----------------------------------------------------------------
// timescale unit/precision
`timescale 1ps / 1ps // each unit is 1 ps,& simulation has 1 ps
precision

moduleTestLaneFIFO;

parameterFIFO_width = 10; // 10 bits
parameterADDR_bits = 3; // 3 bits

// stimuli signal generation
regtstWr_Clock, tstRd_Clock, tstFIFOReset;
wiretstFIFO_Full, tstFIFO_Empty;
wiretstInsert_Req, tstDelete_Req;
wire [2:0] tstPtrDiff;

reg [FIFO_width-1:0] tstWrData; // FIFO Wirte In data
reg [31:0] TestData;
reg [11:0] ClkCounter;

wire [FIFO_width-1:0] tstDataOut;
wire [FIFO_width-1:0] tstNxtRdData;
wire [(ADDR_bits -1):0] tstFIFO_WrPtr;
wire [(ADDR_bits -1):0] tstFIFO_RdPtr;
wire [(ADDR_bits -1):0] tstFIFO_NxtRdPtr;
wire [(ADDR_bits -1):0] tstCrntWrPtrGray;
wire [(ADDR_bits -1):0] tstCrntRdPtrGray;
wire [(ADDR_bits -1):0] tstNxtRdPtrGray;
wire [(ADDR_bits -1):0] tstNxtNxtRdPtrGray;
wire [(ADDR_bits -1):0] tstSyncdWrPtr;
wiretstReadEn;
wire [2:0] tstCounter;
// UUT instantation

LaneFIFOUUT(
  .Wr_Clock(tstWr_Clock),
  .Rd_Clock(tstRd_Clock),
  .FIFOReset(tstFIFOReset),
  .CrntWrData(tstWrData),
  .InsertReq(tstInsert_Req),
  .DeleteReq(tstDelete_Req),
  .CrntWrPtr(tstFIFO_WrPtr),
  .CrntRdPtr(tstFIFO_RdPtr),
  .NxtRdPtr(tstFIFO_NxtRdPtr),
  .PtrDiff(tstPtrDiff),
  .FIFO_Full(tstFIFO_Full),
  .FIFO_Empty(tstFIFO_Empty),
  // Testing purpose only
  ///  TestClkCounter,
  .CrntRdData(tstDataOut),
  .NxtRdData(tstNxtRdData),
  .CrntWrPtrGray(tstCrntWrPtrGray),
  .CrntRdPtrGray(tstCrntRdPtrGray),
  .NxtRdPtrGray(tstNxtRdPtrGray),
  .NxtNxtRdPtrGray(tstNxtNxtRdPtrGray),
  .SyncWrPtr(tstSyncdWrPtr),
  .ReadEn(tstReadEn),
  .Counter(tstCounter)
 );

// Define characters
`defineStartChar 10'b1101101000 // K27.7
`defineTerminateChar 10'b1011101000 // K29.7
`defineCharA 10'b0011110011 // K28.3
`defineCharK 10'b0011111010 // K28.5
`defineCharR 10'b0011110100 // K28.0

`defineWrClkCycles 12'd2500 // 10k / 4 = 2500

  // -----------------------------------------------------------------
  // Initialization
  // -----------------------------------------------------------------
initial
begin
tstFIFOReset = 1'b0; // reset FIFO
    #16010 tstFIFOReset = 1'b1;
end

  // -----------------------------------------------------------------
  // Clock signal generation
  // -----------------------------------------------------------------
  /* Notes:
   FIFO_FULL & Delete request condition being testing
tstWr_Clock: 312.5 MHz + 100 ppm => 312531.25 kHz => 3199.68 ps
tstRd_Clock: 312.5 Mhz - 100 ppm => 312468.75 kHz => 3200.3 ps

tstWr_Clock: 312.5 MHz + 200 ppm => 312562.5 kHz => 3199.3 ps
tstRd_Clock: 312.5 Mhz - 200 ppm => 312437.5 kHz => 3200.6 ps
  */

  // Write pointer to be derived on received/recovered clock
initial // Clock generator
begin
tstWr_Clock = 1'b1;
forever #1599 tstWr_Clock = !tstWr_Clock; // invert every 3199.68 /
2 = 1599.84ps
end

  // Read pointer to be derived on the local clock
initial // Clock generator
begin
tstRd_Clock = 1'b1;
forever #1600 tstRd_Clock = !tstRd_Clock; // invert every 3200.3 / 2
= 1600.15ps
end

  // -----------------------------------------------------------------
  // Data packet generation
  // -----------------------------------------------------------------
```

```
always @ (posedgetstWr_Clock or negedgetstFIFOReset)
if (!tstFIFOReset)
begin
ClkCounter = 0;
end
else
begin
if(ClkCounter>= `WrClkCycles)
ClkCounter = 1;
else
ClkCounter = ClkCounter + 1;
end

  //always @(ClkCounter>= 0) // start character
always @ (posedgetstWr_Clock or negedgetstFIFOReset)
if (!tstFIFOReset)
begin
tstWrData = `CharA; // idle
TestData = 32'b0;
end
else
begin
TestData = $random;
if(ClkCounter == 1)
tstWrData = `StartChar;
else if(ClkCounter == (`WrClkCycles - 3)) // 2497 = T
tstWrData = `TerminateChar;
else if(ClkCounter == (`WrClkCycles - 2)) // 2498 = A
tstWrData = `CharA;
else if(ClkCounter == (`WrClkCycles - 1)) // 2499 = R is chosen as
the second I following T.
tstWrData = `CharR;
else if(ClkCounter == `WrClkCycles) // 2500 = K
tstWrData = `CharK;
else
        // TxData = TestData[(Lane0FIFO_width - 1):0];
tstWrData = TestData[(FIFO_width - 1):0];
end
// ---------------------------------------------------------------


// ---------------------------------------------------------------
initial
#81_000_000 $stop;
// ----------------------------------------------------------------
endmodule
```

Appendix B: Verilog testbench for 40GBASE-R PCS sub layer Model

```
// ------------------------------------------------------------
// File Name: TestFourtyG_PCS.v
// Description: This is the implementation of TestBench for 40G
transmitter
// Input: tstPCSReset - Active Low Reset
// Output:
// Notes:
// TODO:
// ------------------------------------------------------------

// timescale unit/precision
`timescale 1ps / 1ps // each unit is 1 ps, & the simulation has 1 ps
precision

module TestFourtyG_PCS;

reg tstTxClock, tstPCSReset, tstPCSEn, tstRxClock;
reg tstSerialLaneClk, tstSerialLaneWrClk, tstSerialLaneRdClk,
NxtSerialLaneWrClk, NxtSerialLaneRdClk;
reg [14:0] SerialClkCntr;
reg [14:0] SerialClkCntrNxt;
reg tstLaneWrClk, tstLaneRdClk, nxtLaneWrClk, nxtLaneRdClk;
reg [63:0] tstTxData;
reg [7:0] tstTxCtrl;

wire [63:0] tstRxData;
wire [7:0] tstRxCtrl;
wire tstRxRdy, tstTxRdy;


`define WrClkCycles 13'd5001
parameter TxClkPeriodbyTwo = 10'd799;
parameter RXClkPeriodbyTwo = 10'd800;
parameter SerialLaneWrClkPeriodbyTwo = 6'd48;
parameter SerialLaneRdClkPeriodbyTwo = 6'd48;

parameter BlockType_S = 8'h78; // PCS /S/ = 0x78
parameter BlockType_T0 = 8'h87; // PCS /T0/ = 0x87
parameter BlockType_T1 = 8'h99;
parameter BlockType_T2 = 8'hAA;
parameter BlockType_T3 = 8'hB4;
parameter BlockType_T4 = 8'hCC;
parameter BlockType_T5 = 8'hD2;
parameter BlockType_T6 = 8'hE1;
parameter BlockType_T7 = 8'hFF;

reg [63:0] TestData;
reg [12:0] ClkCounter;
reg [7:0] CkEdgeCounterTx;
reg [7:0] CkEdgeCounterTxNxt;
reg [7:0] CkEdgeCounterRx;
reg [7:0] CkEdgeCounterRxNxt;

integer i;
```

```
// -----------------------------------------------------------
// UUT Instantation
// -----------------------------------------------------------
FourtyG_PCS UUT_PCS(
 .TxClock(tstTxClock), //   input TxClock,
 .SerialLaneWrClk(tstSerialLaneWrClk),
 .LaneWrClk(tstLaneWrClk),
 .LaneRdClk(tstLaneRdClk),
 .TxData(tstTxData), //   input [63:0] TxData,
 .TxCtrl(tstTxCtrl), //   input [7:0] TxCtrl,
 .PCSReset(tstPCSReset), //   input PCSReset,
 .PCSEn(tstPCSEn), //   input PCSEn,
 .RxClock(tstRxClock), //   input RxClock,
 .SerialLaneRdClk(tstSerialLaneRdClk),
 .RxData(tstRxData), //   output [63:0] RxData,
 .RxCtrl(tstRxCtrl), //   output [7:0] RxCtrl
 .TxRdy(tstTxRdy), //  output TxRdy,
 .RxRdy(tstRxRdy) // output RxRdy

 );

// -----------------------------------------------------------
// Clock signal generation
// Serial Lane clock: 10.3125 G => 96.9697 ps => T/2 = 48 ps
// tstSerialLaneClk = 24 ps is used to obtain serial lane Rd Wr
//                    clocks of T/2 = 48 ps with =/- 100 ppm
// -----------------------------------------------------------
 initial // Clock Generator
  begin
    tstSerialLaneClk = 1'b0;
    forever #24 tstSerialLaneClk = !tstSerialLaneClk; // invert
every (1/10.3125) / 4 = 24.2424
  end

// -----------------------------------------------------------
// Clock signal generation
// Serial Lane clock: 10.3125 G => 96.9697 ps
// 100 ppm => 100 for 10^6 => 1 for 10 000 clks
// stop the 10001 clk -> slower clk
// having the 10001 as a normal clock -> faster clk
// tstSerialLaneWrClk(47) > tstSerialLaneRdClk(49) => FIFO_FULL
// tstSerialLaneWrClk(49) < tstSerialLaneRdClk(47) => FIFO_EMPTY
// Generated period of both tstSerialLaneWrClk & tstSerialLaneWrClk
are 96 ps
// Expected period of both tstSerialLaneWrClk & tstSerialLaneWrClk
are 96.9697 ps
// -----------------------------------------------------------
initial
  begin
    tstSerialLaneWrClk = 1'b0;
    tstSerialLaneRdClk = 1'b0;
    SerialClkCntr = 15'd0;
  end

always @(*) // Combinational logic
  begin
    if (SerialClkCntr >= 15'd20001) // counts 0 : 20001
      SerialClkCntrNxt <= 15'd0;
    else
      SerialClkCntrNxt <= #1 SerialClkCntr + 15'd1;
  end
always @ (posedge tstSerialLaneClk)
  begin
    SerialClkCntr <= #1 SerialClkCntrNxt;
  end

always @ (posedge tstSerialLaneClk)
begin
  if (SerialClkCntr >= 15'd20000)
    begin
      // Delete req Assert: FIFO Full possible
      tstSerialLaneRdClk <= #1 1'b0; // slow clk
      tstSerialLaneWrClk <= #1 ~tstSerialLaneWrClk; // fast clk
//      // Insert req Assert: FIFO Empty possible
//      tstSerialLaneWrClk <= #1 1'b0; // slow clk
//      tstSerialLaneRdClk <= #1 ~tstSerialLaneRdClk; // fast clk
    end
  else
    begin
      tstSerialLaneWrClk <= #1 !tstSerialLaneWrClk;
      tstSerialLaneRdClk <= #1 !tstSerialLaneRdClk;
  end
end

// ---------------------------------------------------------------
// Clock Signal Generation: Lane Parallel input/output clock
// tstSerialLaneWrClk /66 = tstLaneWrClk; tstLaneWrClk * 4 =
tstTxClock
// clock: 156.25 M: period 6336 ps(expected 6400 ps)
//          When crossing boundaaries having a diff of 96 ps  6330ps :
6432ps
// ---------------------------------------------------------------
initial
  begin
    tstLaneWrClk = 1'b0;
    CkEdgeCounterTx = 7'b0000000;
    tstLaneRdClk = 1'b0;
    CkEdgeCounterRx = 7'b0000000;
  end
// Tx parallel clock generation
always @ (posedge tstSerialLaneWrClk)
  begin
    tstLaneWrClk <= #5 nxtLaneWrClk;
  end

always @ (posedge tstSerialLaneWrClk)
  begin
    CkEdgeCounterTx <= #1 CkEdgeCounterTxNxt;
  end
always @(*) // Combinational logic
  begin
    if (CkEdgeCounterTx >= 7'd32)
      CkEdgeCounterTxNxt = 7'b0000000;
    else
      CkEdgeCounterTxNxt = #1  CkEdgeCounterTx + 7'b0000001;
  end
```

```
always @(*)
  begin
    if (CkEdgeCounterTx >= 7'd32)
      nxtLaneWrClk = !tstLaneWrClk;
    else
      nxtLaneWrClk = tstLaneWrClk;
  end

// Rx parallel clock generation
always @ (posedge tstSerialLaneRdClk)
  begin
    tstLaneRdClk <= #5 nxtLaneRdClk;
  end

always @ (posedge tstSerialLaneRdClk)
  begin
    CkEdgeCounterRx<= #1 CkEdgeCounterRxNxt;
  end
always @(*) // Combinational logic
  begin
    if (CkEdgeCounterRx >= 7'd32)
      CkEdgeCounterRxNxt = 7'b0000000;
    else
      CkEdgeCounterRxNxt = #1  CkEdgeCounterRx + 7'b0000001;
  end

always @(*)
  begin
    if (CkEdgeCounterRx >= 7'd32)
      nxtLaneRdClk = !tstLaneRdClk;
    else
      nxtLaneRdClk = tstLaneRdClk;
  end


// --------------------------------------------------------------
// Clock signal generation
// RX_CLK & TX_CLK 625 MHz: period = 1591ps (1600 expected)
// Sometimes this goes for 1584: 1679 diff of 95 ps
// --------------------------------------------------------------
initial
  begin
    tstTxClock = 1'b0;
    tstRxClock = 1'b0;
  end

// Clock Generator Tx
always @(posedge tstLaneWrClk)
  begin
   repeat (8)
    begin
      tstTxClock = # ((SerialLaneWrClkPeriodbyTwo * 2 * 66) / 8)
~tstTxClock;
    end
  end

// Clock Generator Rx
```

```
always @(posedge tstLaneRdClk)
  begin
    repeat (8)
      begin
      tstRxClock = # ((SerialLaneRdClkPeriodbyTwo * 2 * 66) / 8)
~tstRxClock;
    end
  end


// --------------------------------------------------------------
// Initialization
// --------------------------------------------------------------
// Reset Signal : Reset pulse width = (2 * 1584) + 5
initial
  begin
    tstPCSReset = 1'b0; // Active low reset pulse
    @ (posedge tstLaneWrClk);
    @ (posedge tstLaneWrClk);
    #6
    tstPCSReset = 1'b1;
  end
 // Enable signal: Enable is activated at(3 * 1584) + 25
initial
  begin
    tstPCSEn = 1'b0; // not eanabled
    @ (posedge tstLaneWrClk);
    @ (posedge tstLaneWrClk);
    @ (posedge tstLaneWrClk);
    #26
    tstPCSEn = 1'b1;
   end
// --------------------------------------------------------------
// Test Data Generation
// --------------------------------------------------------------
always @ (posedge tstTxClock)
  begin
    if((tstPCSReset == 1'b0) || (!tstPCSEn) || (!tstRxRdy))
      begin
        ClkCounter = 13'd0;
      end
    else
      begin
        if(ClkCounter >= `WrClkCycles)
          ClkCounter = 13'd1;
        else
          ClkCounter = ClkCounter + 13'd1;
      end
  end

// --------------------------------------------------------------
// Test Data Packet considerarions
// Minimum IPG = 96 bits = 12 bytes
// Maximum clock rate diff +/- 100 ppm => 1 clk diff for 5000 clks
// Data Pkt :-
//            Clk1: S0 D1 D2 D3 D4 D5 D6 D7 - Data 7
//            Clk5000: D0 D1 D2 T  I0 I1 I2 I3 - Data 3 : 5000 clks
//            Clk5001: I0 I1 I2 I3 I4 I5 I6 I7
//            Size: 7 + 3 + (8 * 4998) = 39994 Bytes
// -------------------------------------------------------------
always @ (posedge tstTxClock or negedge tstPCSReset)
begin
 if ((tstPCSReset == 1'b0)|| (!tstPCSEn)) // Active Low Reset
assumed
   begin
    TestData = {64{1'b0}};
    tstTxData =
64'b00000111_00000111_00000111_00000111_00000111_00000111_00000111_0
0000111; // idle
    tstTxCtrl = 8'b11111111;
   end
 else
   begin
    TestData = $random;
    if (!tstRxRdy)
      begin
        tstTxData =
64'b00000111_00000111_00000111_00000111_00000111_00000111_00000111_0
0000111; // idle
        tstTxCtrl = 8'b11111111;
      end
    else if(ClkCounter == 1)
      begin
        tstTxData ={TestData[55:0],8'hFB}; // Start: SDDD DDDD
        tstTxCtrl = 8'b00000001;
      end

    else if(ClkCounter == (`WrClkCycles - 1)) // T3: DDDT IIII
      begin
        tstTxData = {8'h07, 8'h07, 8'h07, 8'h07, 8'hFD,
TestData[23:0]};
        tstTxCtrl = 8'b11111000;
      end
    else if(ClkCounter == `WrClkCycles) // I
      begin
        tstTxData =
64'b00000111_00000111_00000111_00000111_00000111_00000111_00000111_0
0000111; // idle 8 octects
        tstTxCtrl = 8'b11111111;
      end
  else
      begin
        tstTxData = TestData;
        tstTxCtrl = 8'h00; // Data
      end
  end
end

// -------------------------------------------------------------
initial
 #600_000_000 $stop;
// -------------------------------------------------------------
endmodule
```