Lab 8

Tasks to be done in this Lab:

1. In this Lab we will implement and verify the questions given in the Mid Semester Lab Exam.

Topics to explore: 1) Use of Floating-Point IP, 2) Floating point numbers, 3) Test bench to verify the functionality

Part -1

Implement the following function and demonstrate the functionality using testbench for two sequential and different combinations of X and T (Note that X < T and both are positive integers).

$$Q(X,T,N) = \frac{X}{T} + \sqrt{\left[\frac{X}{T} - \left(\frac{X}{T}\right)^2\right]}$$

1. The code for the Q-Function is given below. Please note that there will be a multi-driven net on **FT_ready** as it will be used for (X/T) and (X/T)^2 calculation. This will make the code non synthesizable. Please observe the code and find how are we handling this.

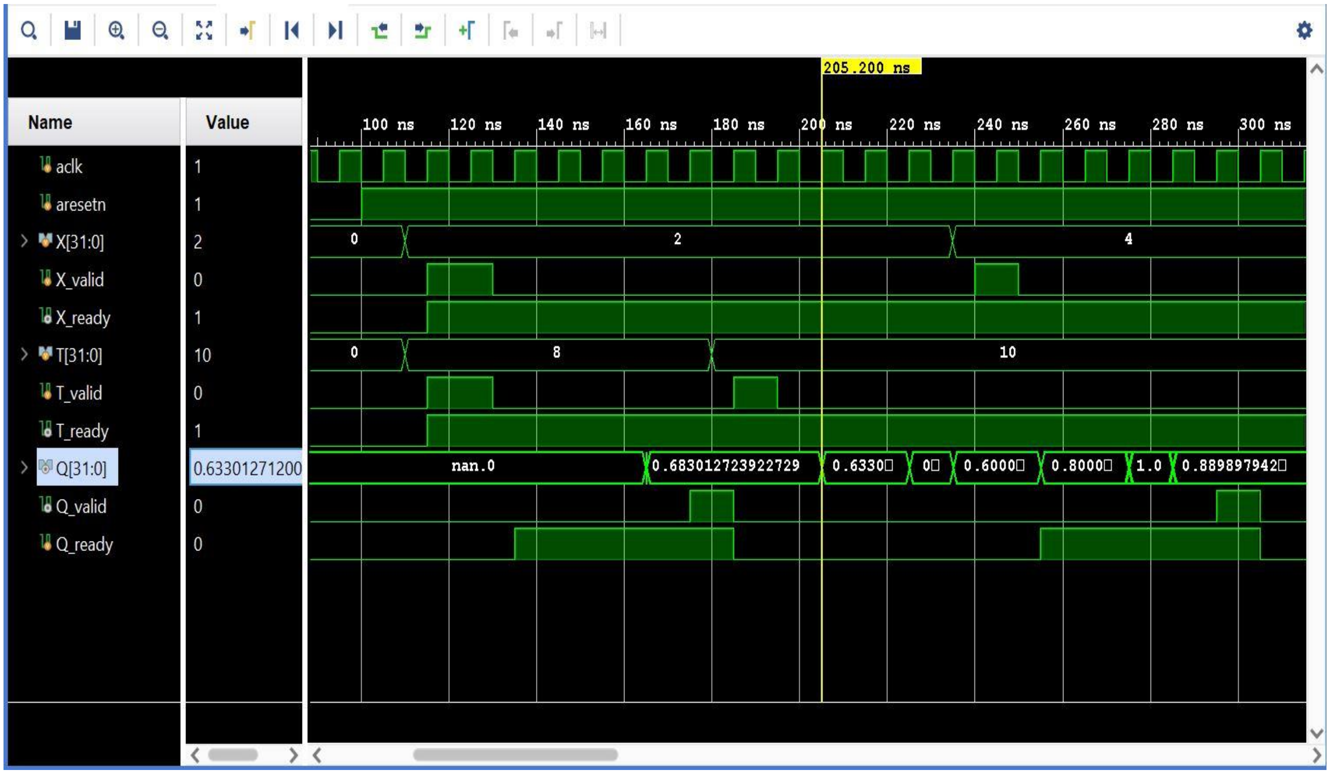
```
`timescale 1ns / 1ps

module QFunc(
    input aclk,
    input aresetn,
    input [31:0] X,
    input X_valid,
    output X_ready,
    input [31:0] T,
    input T_valid,
    output T_ready,
    output [31:0] Q,
    output Q_valid,
    input Q_ready
    );

// Step: 1 Convert the fixed point X and T to floating point by using the fixed to float option of floating point IP
wire [31:0] FX;
wire FX_valid,FX_ready;

    fixed_to_float X_float (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(X_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(X_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(X),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(FX_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(FX_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(FX)    // output wire [31 : 0] m_axis_result_tdata
);
```

```
    wire [31:0] FT;
    wire FT_valid,FT_ready;

    fixed_to_float T_float (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(T_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(T_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(T),               // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(FT_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(FT_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(FT)     // output wire [31 : 0] m_axis_result_tdata
);

// Step 2: Use the divide operation to calculate X/T
wire [31:0] X_div_T;
wire X_div_T_ready,X_div_T_valid;

divide x_div_T (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(FX_valid),           // input wire s_axis_a_tvalid
  .s_axis_a_tready(FX_ready),           // output wire s_axis_a_tready
  .s_axis_a_tdata(FX),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(FT_valid),           // input wire s_axis_b_tvalid
  .s_axis_b_tready(FT_ready),           // output wire s_axis_b_tready
  .s_axis_b_tdata(FT),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(X_div_T_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(X_div_T_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(X_div_T)     // output wire [31 : 0] m_axis_result_tdata
);

// Step 3 : Use the multiply operation to calculate (X/T)^2
wire [31:0] X_div_T_squared;
wire X_div_T_squared_ready,X_div_T_squared_valid;

multiply X_T_Squared (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(X_div_T_valid),          // input wire s_axis_a_tvalid
  .s_axis_a_tready(X_div_T_ready),          // output wire s_axis_a_tready
  .s_axis_a_tdata(X_div_T),             // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(X_div_T_valid),          // input wire s_axis_b_tvalid
  //.s_axis_b_tready(X_div_T_ready),          // output wire s_axis_b_tready
  .s_axis_b_tdata(X_div_T),             // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(X_div_T_squared_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(X_div_T_squared_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(X_div_T_squared)     // output wire [31 : 0] m_axis_result_tdata
);
```

```
// Step 4: Calculation of X/T - (X/T)^2. This time we are not using the operation port and rather we have selected the subtract option while customizing the IP
wire [31:0] difference;
wire difference_valid,difference_ready;

sub subtraction (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(X_div_T_valid),            // input wire s_axis_a_tvalid
  //.s_axis_a_tready(X_div_T_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(X_div_T),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(X_div_T_squared_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(X_div_T_squared_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(X_div_T_squared),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(difference_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(difference_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(difference)    // output wire [31 : 0] m_axis_result_tdata
);

// Step 5: Calculation of sqrt[X/T - (X/T)^2]
wire [31:0] root;
wire root_valid,root_ready;

sqrt squareroot (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(difference_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(difference_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(difference),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(root_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(root_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(root)    // output wire [31 : 0] m_axis_result_tdata
);


// Step 6: Calculation of X/T + sqrt[X/T - (X/T)^2]

add final_result (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(X_div_T_valid),            // input wire s_axis_a_tvalid
  //.s_axis_a_tready(X_div_T_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(X_div_T),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(root_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(root_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(root),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(Q_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(Q_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(Q)    // output wire [31 : 0] m_axis_result_tdata
);
endmodule
```

2. Write the test-bench as given below and verify the functionality.

```
module q_tb(

   );

   reg aclk,aresetn,X_valid,T_valid,Q_ready;
   reg [31:0] X,T;
   wire [31:0] Q;
   wire X_ready,T_ready,Q_valid;

   QFunc tb1(.aclk(aclk),.aresetn(aresetn),.X(X),.T(T),.Q(Q),
             .X_valid(X_valid),.T_valid(T_valid),.Q_valid(Q_valid),
             .X_ready(X_ready),.T_ready(T_ready),.Q_ready(Q_ready));

    initial
       begin
           aclk=0;
           aresetn=0;
           X=0;
           X_valid=0;
           T=0;
           T_valid=0;
           Q_ready=0;
       end

    always #5 aclk=~aclk;



initial
     begin
     #100 aresetn=1;
     #10 T=8;
     #5 T_valid=1;
         while(T_ready==0)  // Wait for the ready signal from the IP
         #5 T_valid=1;
      #10 T_valid=0;       // Set the valid signal '0' once the handshake is complete

     #50 T=10;
     #5 T_valid=1;
         while(T_ready==0)  // Wait for the ready signal from the IP
         #5 T_valid=1;
      #10 T_valid=0;       // Set the valid signal '0' once the handshake is complete
 end

 initial
     begin
     #110 X=2;
     #5 X_valid=1;
         while(X_ready==0)  // Wait for the ready signal from the IP
         #5 X_valid=1;
      #10 X_valid=0;        // Set the valid signal '0' once the handshake is complete
      #5 Q_ready=1'b1;

      wait(Q_valid==1'b1)  // Wait for Q_valid(will be set by the IP) to go high
         #5 Q_ready=1'b1; // Keep Q_ready high for one clock cycle for handshake to take place
      #5 Q_ready=1'b0;

     #50 X=4;
     #5 X_valid=1;
         while(X_ready==0)  // Wait for the ready signal from the IP
         #5 X_valid=1;
      #10 X_valid=0;        // Set the valid signal '0' once the handshake is complete
      #5 Q_ready=1'b1;

      wait(Q_valid==1'b1)  // Wait for Q_valid(will be set by the IP) to go high
         #5 Q_ready=1'b1; // Keep Q_ready high for one clock cycle for handshake to take place
      #5 Q_ready=1'b0;
 end
endmodule
```


205.200 ns
Name
Value
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
260 ns
280 ns
300 ns
aclk
1
aresetn
1
X[31:0]
2
0
2
4
X_valid
0
X_ready
1
T[31:0]
10
0
8
10
T_valid
0
T_ready
1
Q[31:0]
0.63301271200
nan.0
0.683012723922729
0.6330
0
0.6000
0.8000
1.0
0.889897942
Q_valid
0
Q_ready
0

Part-2

Extend the above to implement the following function. Demonstrate the functionality using testbench for two different combinations of X, N and T (Note that X < T< N and all are positiveintegers).

$$Q(X,T,N) = \frac{X}{T} + \sqrt{\frac{\log N}{T}\min\left\{\frac{1}{4}, \left[\frac{X}{T} - \sqrt{\frac{4\log N}{T}}\right]\right\}}$$

1. The Code of the Q Function is given below:

```
`timescale 1ns / 1ps

module QFunc(
    input aclk,
    input aresetn,
    input [31:0] N,
    input N_valid,
    output N_ready,
    input [31:0] X,
    input X_valid,
    output X_ready,
    input [31:0] T,
    input T_valid,
    output T_ready,
    input Q_ready,
    output [31:0] Q,
    output Q_valid
    );

//Step 1: Convert X,N and T in Floating Point Representation
    wire [31:0] FN;
    wire FN_valid,FN_ready;

    fixed_to_float N_float (
  .aclk(aclk),                               // input wire aclk
  .aresetn(aresetn),                         // input wire aresetn
  .s_axis_a_tvalid(N_valid),             // input wire s_axis_a_tvalid
  .s_axis_a_tready(N_ready),             // output wire s_axis_a_tready
  .s_axis_a_tdata(N),                // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(FN_valid),   // output wire m_axis_result_tvalid
  .m_axis_result_tready(FN_ready),   // input wire m_axis_result_tready
  .m_axis_result_tdata(FN)     // output wire [31 : 0] m_axis_result_tdata
);
```

```
    wire [31:0] FX;
    wire FX_valid,FX_ready;

    fixed_to_float X_float (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(X_valid),           // input wire s_axis_a_tvalid
  .s_axis_a_tready(X_ready),           // output wire s_axis_a_tready
  .s_axis_a_tdata(X),             // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(FX_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(FX_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(FX)    // output wire [31 : 0] m_axis_result_tdata
);

    wire [31:0] FT;
    wire FT_valid,FT_ready;

    fixed_to_float T_float (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(T_valid),           // input wire s_axis_a_tvalid
  .s_axis_a_tready(T_ready),           // output wire s_axis_a_tready
  .s_axis_a_tdata(T),             // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(FT_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(FT_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(FT)    // output wire [31 : 0] m_axis_result_tdata
);
```

```
//Step 2: Calculate X/T using the divide operation
wire [31:0] X_div_T;
wire X_div_T_ready,X_div_T_valid;

divide x_div_T (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(FX_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(FX_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(FX),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(FT_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(FT_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(FT),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(X_div_T_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(X_div_T_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(X_div_T)    // output wire [31 : 0] m_axis_result_tdata
);

//Step 3: Calculate ln(N) using the logarithm operation
wire [31:0] ln_N;
wire ln_N_ready,ln_N_valid;

ln ln__N(
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(FN_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(FN_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(FN),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(ln_N_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(ln_N_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(ln_N)    // output wire [31 : 0] m_axis_result_tdata
);
```

```
//Step 4: Calculate ln(N)/T using the divide operation
wire [31:0] ln_N_div_T;
wire ln_N_div_T_ready,ln_N_div_T_valid;

divide lnN_div_T (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(ln_N_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(ln_N_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(ln_N),               // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(FT_valid),            // input wire s_axis_b_tvalid
  //.s_axis_b_tready(FT_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(FT),               // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(ln_N_div_T_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(ln_N_div_T_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(ln_N_div_T)    // output wire [31 : 0] m_axis_result_tdata
);
```

```
/*Step 5: Calculate 4*ln(N)/T
 One thing to note here we have provided the single precison converted value of 4. In the valid signal we have used the
 ln_N_div_T_valid instead of 1'b1 so that the multiplication takes place whenever there is a change in inputs.
*/
wire [31:0] four_Ln_div_T;
wire four_Ln_div_T_ready,four_Ln_div_T_valid;

multiply fourLnbyT (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(ln_N_div_T_valid),            // input wire s_axis_a_tvalid
  //.s_axis_a_tready(F_Four_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(32'b01000000100000000000000000000000),             // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(ln_N_div_T_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(ln_N_div_T_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(ln_N_div_T),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(four_Ln_div_T_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(four_Ln_div_T_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(four_Ln_div_T)    // output wire [31 : 0] m_axis_result_tdata
);
```

```verilog
// Step 6: Calculate sqrt(4*ln(N)/T)
wire [31:0] root_ln_fxn;
wire root_ln_fxn_ready,root_ln_fxn_valid;

sqrt ln_fxn_root (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(four_Ln_div_T_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(four_Ln_div_T_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(four_Ln_div_T),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(root_ln_fxn_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(root_ln_fxn_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(root_ln_fxn)    // output wire [31 : 0] m_axis_result_tdata
);

// Step 7: Calculate X/T - sqrt(4*ln(N)/T)
wire [31:0] difference;
wire difference_valid,difference_ready;

subtract subtraction (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(X_div_T_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(X_div_T_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(X_div_T),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(root_ln_fxn_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(root_ln_fxn_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(root_ln_fxn),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(difference_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(difference_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(difference)    // output wire [31 : 0] m_axis_result_tdata
);
```

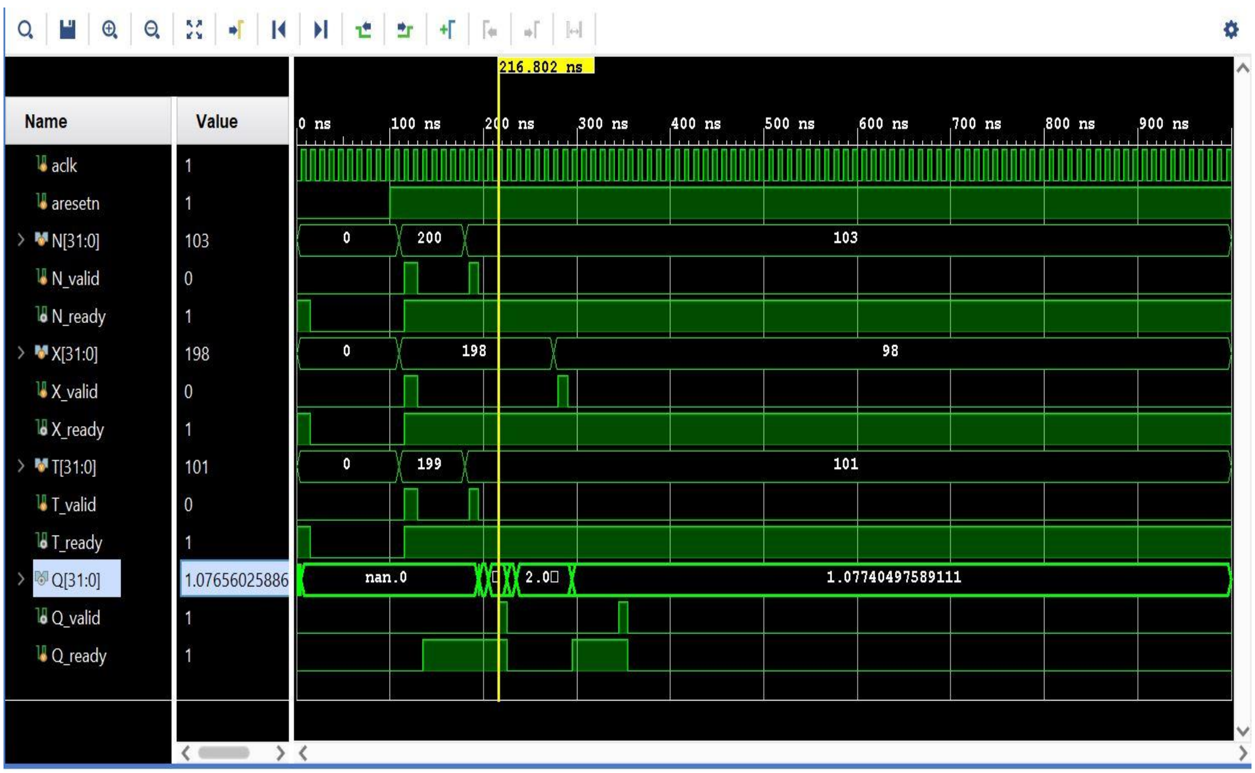
```verilog
/*Step 8: To find the minimum, we have used the compare operation with less than option. This operation results in 1 if A<B.
As A=0.25 we have used the single precision converted value and in valid we have used difference_valid.
*/
wire [7:0] less_than_res;
wire less_than_res_ready,less_than_res_valid;

compare_less_than min_fxn (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(difference_valid),            // input wire s_axis_a_tvalid
  //.s_axis_a_tready(one_div_four_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(32'b00111110100000000000000000000000),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(difference_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(difference_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(difference),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(less_than_res_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(less_than_res_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(less_than_res)    // output wire [7 : 0] m_axis_result_tdata
);
```

```
wire [31:0] min;
wire min_ready,min_valid;
/* We have defined a wire min which will be connected to next IP,and it will take the value 0.25 if compare IP results in 1
else it will take the value of difference.
*/

assign min=less_than_res?32'b00111110100000000000000000000000:difference;

//Step 9: Multiply the minimum value with ln(T)/T
wire [31:0] min_multiply;
wire min_multiply_ready,min_multiply_valid;

multiply min_multiplier (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(less_than_res_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(less_than_res_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(min),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(ln_N_div_T_valid),            // input wire s_axis_b_tvalid
  //.s_axis_b_tready(ln_N_div_T_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(ln_N_div_T),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(min_multiply_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(min_multiply_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(min_multiply)    // output wire [31 : 0] m_axis_result_tdata
);


//Step 10: Take the square root of the operation performed in Step 9
wire [31:0] outer_root;
wire outer_root_ready,outer_root_valid;

sqrt root_outer (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(min_multiply_valid),            // input wire s_axis_a_tvalid
  .s_axis_a_tready(min_multiply_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(min_multiply),              // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(outer_root_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(outer_root_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(outer_root)    // output wire [31 : 0] m_axis_result_tdata
);

// Step 10: FInally add X/T to get the final answer.

add final_result (
  .aclk(aclk),                                  // input wire aclk
  .aresetn(aresetn),                            // input wire aresetn
  .s_axis_a_tvalid(X_div_T_valid),            // input wire s_axis_a_tvalid
  //.s_axis_a_tready(X_div_T_ready),            // output wire s_axis_a_tready
  .s_axis_a_tdata(X_div_T),              // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(outer_root_valid),            // input wire s_axis_b_tvalid
  .s_axis_b_tready(outer_root_ready),            // output wire s_axis_b_tready
  .s_axis_b_tdata(outer_root),              // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(Q_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(Q_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(Q)    // output wire [31 : 0] m_axis_result_tdata
);
endmodule
```

2. Write the testbench as given below and verify the functionality.

```
module q_tb(

  );

  reg aclk,aresetn,N_valid,X_valid,T_valid,Q_ready;
  reg [31:0] N,X,T;
  wire [31:0] Q;
  wire N_ready,X_ready,T_ready,Q_valid;

  QFunc tb1(.aclk(aclk),.aresetn(aresetn),.N(N),.X(X),.T(T),.Q(Q),
             .N_valid(N_valid),.X_valid(X_valid),.T_valid(T_valid),.Q_valid(Q_valid),
             .N_ready(N_ready),.X_ready(X_ready),.T_ready(T_ready),.Q_ready(Q_ready));

   initial
      begin
          aclk=0;
          aresetn=0;
          N=0;
          N_valid=0;
          X=0;
          X_valid=0;
          T=0;
          T_valid=0;
          Q_ready=0;
      end

   always #5 aclk=~aclk;

  initial
      begin
      #100 aresetn=1;
      #10 N=200;
      #5 N_valid=1;
          while(N_ready==0)     // Wait for the ready signal from the IP
          #5 N_valid=1;
       #10 N_valid=0;          // Set the valid signal '0' once the handshake is complete

      #50 N=103;
      #5 N_valid=1;
          while(N_ready==0)     // Wait for the ready signal from the IP
          #5 N_valid=1;
       #10 N_valid=0;          // Set the valid signal '0' once the handshake is complete
  end

 initial
     begin
     #110 T=199;
     #5 T_valid=1;
         while(T_ready==0)  // Wait for the ready signal from the IP
         #5 T_valid=1;
      #10 T_valid=0;       // Set the valid signal '0' once the handshake is complete

     #50 T=101;
     #5 T_valid=1;
         while(T_ready==0)  // Wait for the ready signal from the IP
         #5 T_valid=1;
      #10 T_valid=0;       // Set the valid signal '0' once the handshake is complete
  end
```

```
initial
    begin
    #110 X=198;
    #5 X_valid=1;
        while(X_ready==0)  // Wait for the ready signal from the IP
        #5 X_valid=1;
     #10 X_valid=0;        // Set the valid signal '0' once the handshake is complete
     #5 Q_ready=1'b1;

     wait(Q_valid==1'b1)  // Wait for Q_valid(will be set by the IP) to go high
        #5 Q_ready=1'b1; // Keep Q_ready high for one clock cycle for handshake to take place
     #5 Q_ready=1'b0;

    #50 X=98;
    #5 X_valid=1;
        while(X_ready==0)  // Wait for the ready signal from the IP
        #5 X_valid=1;
     #10 X_valid=0;        // Set the valid signal '0' once the handshake is complete
     #5 Q_ready=1'b1;

     wait(Q_valid==1'b1)  // Wait for Q_valid(will be set by the IP) to go high
        #5 Q_ready=1'b1; // Keep Q_ready high for one clock cycle for handshake to take place
     #5 Q_ready=1'b0;
  end
endmodule
```



Note: For conversion to single precision floating point number you can use the following website.
https://www.h-schmidt.net/FloatConverter/IEEE754.html